Reference manual
20-sim 4.5

Windows XP / Vista / 7 / 8 / 8.1

20-sim 4.5 Reference Manual

© 2015, Controllab Products B.V.
Author: Ir. C. Kleijn, Ir. M. A. Groothuis

Disclaimer

This manual describes the modeling and simulation package 20-sim.

Controllab Products B.V. makes every effort to insure this information is accurate and reliable. Controllab Products B.V. will not accept any responsibility for damage that may arise from using this manual or information, either correct or incorrect, contained in this manual.

Information in this document is subject to change without notice. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Controllab Products B.V.

Windows is a registered trademark of the Microsoft Corporation, USA.
MATLAB is a registered trademark of The MathWorks, Inc., USA.

Reference

Kleijn, C., Groothuis, M.A., Differ H.G.

20-sim 4.5 Reference Manual
Enschede, Controllab Products B.V., 2015
ISBN 978-90-79499-18-2

Information

Controllab Products B.V.
Address: Hengelosestraat 500
7521 AN Enschede
the Netherlands
Phone: +31-85-7731872
Internet: www.20sim.com
www.controllab.nl
E-mail: info@20sim.com

Table of Contents

1 Welcome to 20-sim 1

2 What is new in 20-sim? 2

3 Requirements 5

4 Installation 6

4.1 Versions of 20-sim 6
4.2 Installing 20-sim 7
4.3 Uninstalling 9
4.4 Deactivation 9
4.5 Unattended Installation 10
4.6 Unattended Uninstall 10
4.7 Matlab 10

5 Editor 12

5.1 Introduction 12
5.2 Using Models 20
5.3 Compiling 45
5.4 Icon Editor 55
5.5 Global Relations Editor 57
5.6 Interface Editor 62
5.7 Domains, Quantities and Units 66

6 Simulator 73

6.1 Introduction 73
6.2 Running a Simulation 76
6.3 Run Properties 97

7 Language Reference 107

7.1 Introduction 107
7.2 Keywords 116
7.3 Types 124

7.4 Functions 127
7.5 Operators 195
7.6 Statements 214
7.7 Matrices and Vectors 228
7.8 Advanced Topics 232

8 Toolboxes 239
8.1 3D Mechanics Toolbox 239
8.2 Animation Toolbox 273
8.3 Control Toolbox 308
8.4 Frequency Domain Toolbox 352
8.5 Mechatronics Toolbox 385
8.6 Real-Time Toolbox 454
8.7 Time Domain Toolbox 465
8.8 Scripting Toolbox 505

9 Library 515
9.1 Bond Graph 515
9.2 Iconic Diagrams 564
9.3 Signal 901

10 Modeling Tutorial 1112
10.1 Friction 1112
10.2 Bond Graphs 1122
10.3 Iconic Diagrams 1153

Index 1168

1 Welcome to 20-sim

This manual describes 20-sim 4.5 in full detail. It is not a getting started manual but intended as a detailed reference to the software. If you are a first time user you are advised to read the Getting Started manual first:

If you are an experienced 20-sim user, you can read the change notes first and then search the topic of your interest.

2 What is new in 20-sim?

General

20-sim 4.5 has been improved significantly to handle very large models:

- The code for loading and compiling of models has been rewritten to handle large models more efficiently. Loading and compiling of large models will now be up to 300% faster.
- The code for plotting graphs has been updated to handle large simulations more efficiently. This will be particularly clear when zooming into plots with many data points.
- The symbolic solver handles algebraic loops more efficiently.

Editor

1. Drag and drop of image files (bitmaps, svg images, ..) is now supported in the Editor and Icon Editor.
2. Native support for SVG background images has been added.
3. Multiple align methods for objects have been added.
4. An additional causal form has been added: Likes causality. This causal form can be used for parasitic elements like the parasitic volume.

Simulator

1. The minimum and maximum value of every variable is now shown in the Variables Chooser if you run a simulation in *Debug Mode* .
2. FFT plots can now be selected in the Simulator tree. Results are updated after every simulation run.
3. FFT plots can show the transfer function, by choosing the the option *Pair-wise transfer function*.
4. Plots and 3D animations can be shown in Full Screen Mode.

Library

1. An extra implementation has been added to the gearbox and transmission models: ideal transmission with efficiency.
2. The model unbalance has been added.
3. A friction model has been added.
4. The accumulator model has been expanded with a real gas and temperature model.
5. A loop flushing valve has been added.
6. A shuttle valve has been added.
7. A pressure compensator has been added.

8. A parasitic volume has been added.

Examples

New example models are added:

- Collision detection: CubesWallCrush, CylinderStack
- Control: Smith Predictor, AntiWindup, Euler differentiation problems, FiniteStateMachine
- Signal Processing: Phase locked loop demo
- Hydraulics: AccumulatorTest, Closed Circuit Drive Train with Loop Flushing Valve, Hydraulic Operated Disk Brake
- 1D Mechanics: Crank Rod Mechanism
- 3D Mechanics: KnucleBoomCrane
- Bond graph: TrolleyWithLoad

Language

1. Support of bitwise operators.

Frequency Domain Toolbox

1. Support of linear systems with time delay. The time delay is now stored in a linear system model and shown with a exponential added to the transfer function.
2. You can choose to unwrap or wrap the phase in a Bode plot.

3D Mechanics Editor

1. You can override each constraint joint by individual spring dampers by setting the Constraint Properties.
2. A compliant joint (6D spring) has been added.

Scripting

1. New functions are added for code generation: `xxsimGenerateCCode`, `xxsimGenerateMatlabCode`, `xxsimGetCCodeTargets`, `xxsimGetMatlabTargets`.
2. New functions are added to fetch and modify tool and model settings: `xxsimQuerySettings`, `xxsimSetSettings`
3. New functions are added fetch the list of implementations of submodels and to change the implementation: `xxsimQueryImplementations`, `xxsimSetImplementations`
4. A new function is added for SISO linearization: `xxsimLinearizeModel`

Bugfixes and Improvements

General

1. Some memory leakages have been fixed.

Library

1. The cylinder models now have a separate barrel mass and a piston mass: CylinderSingle, CylinderSingleSpringReturn, CylinderDouble.

Editor

1. The values of global variables are now correctly shown as tooltips.
2. The Zooming option Scale to Fit now takes into account a background image.
3. Imploding a set of models now maintains the arithmetic type (boolean/real/integer).
4. Expressions are better evaluated with tool tips.
5. Printing a model now layouts better.
6. Datafiles from Power Spectral Density function are also stored in .emz file.

Simulator

1. FFT plots now have sufficient height.
2. Autosave now completely stores simulation data.

3D Animation

1. Selections of objects are only shown with the plot properties open. Not when you hover your mouse over a 3D animation.
2. Camera movements with the mouse are more intuitive.

3D mechanics Editor

3. A mouse click will not move the plot with 2 pixels anymore.
4. For constraint joints that contain spring/dampers, proper code is generated now.
5. Fix for code generation of model "Generic Filter"

3 Requirements

20-sim only works on computers that will meet the following requirements:

- Operating System: Windows XP, Vista, 7, 8, 8.1
- Processor requirements: 20-sim requires a CPU with SSE2 support. Supported: Intel Pentium 4 and above, AMD Athlon x64 and above.
- Available Disk Space: 350 MB.

4 Installation

4.1 Versions of 20-sim

20-sim is available in two versions: Viewer and Professional.

- **Viewer/Demonstration version**: This is a freeware version that allows you to load and run models and evaluate the package. Saving of models is not possible in this version.
- **Professional**: This is the full version of 20-sim with all toolboxes.

The table below shows in detail the options that are available in the three versions:

	Viewer	Professional
Library Models	v*	v
3D Mechanics Toolbox	v*	v
Animation Toolbox	v*	v
Control Toolbox	v*	v
Frequency Domain Toolbox	v*	v
Mechatronics Toolbox	v*	v
Real Time Toolbox	v*	v
Time Domain Toolbox	v*	v
Scripting Toolbox	x	v

v = included
v* = included but no saving possible
x = not included

20-sim is installed, using an Installation Manager that will lock 20-sim to your computer. There are three types of licenses available:

- **Free :** The demonstration version comes with a license that is not locked to your computer. No actions have to be taken after installation of the program.
- **Single License:** A single license locks 20-sim to a specific computer. After installation you have to register to get a valid license.
- **Floating License:** A floating license allows multiple users to work with 20-sim at the same time. After installation you have to register to get a valid license.

4.2 Installing 20-sim

20-sim can be downloaded from the website www.20sim.com. This is an installation file that will install 20-sim on your computer. The first 4 steps are equal for all users. Depending on the type of license (single, floating) you have to follow different steps to activate 20-sim.

1. **Download** and **Install** 20-sim on your computer.

2. **Start** 20-sim (from the **Windows Start Menu** choose **20-sim 4.5**).

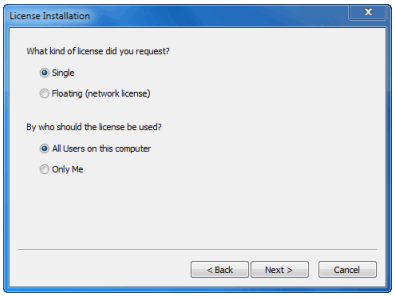
If a valid license of 20-sim 4.5 was activated before, the program will start automatically. If you have not installed 20-sim before, the *License Activation* dialog will open:

20-sim License Activation Dialog.

3. If you have a valid license key or license file, press the **Activation** button to enter your license key or browse for the license file.

If you do not yet have a valid license, press the **Trial License** button request an trial license or press the **Buy** button to purchase a license. If you want to continue in *Viewer* mode (no save functionality), just close the dialog without activating 20-sim.

4. Select **which kind of license** you have and **who** should use the license.

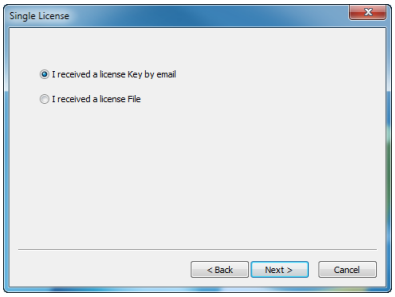


License installation dialog.

Single License

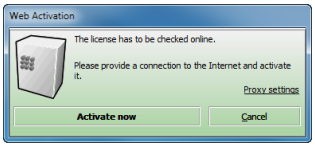
If you are using a single license, you have to enter a license key or license file.

5. On the next dialog, select ***I received a license key by e-mail*** and enter the **key** in the next dialog. When you received a **license file**, you have to enter the **location** of the license file.



Single License dialog.

You will be asked for confirmation (click **Activate Now**) and activation will be carried out. After a successful activation process the License Information dialog will show the new license.



Web Activation dialog.

Floating License

Installing a floating license (Administrator)

If you are using a license that is shared by more users (floating license, also known as concurrent license or server license), you have enter the received license key and a location on the server (a normal Windows shared folder) first. This location on the server should be accessible to all users and have read/write permission. The floating license will be stored at the selected location.

6. On the next dialog, select **First Installation** and then enter the **license key** and the **location on the server** (Windows share).

On the location that you have given, a license file *20sim.lic* will be installed. Remember the location of this file because every new user of 20-sim will need to enter it. You will be asked for confirmation (click **Activate Now**) and activation will be carried out. After a successful activation process the License Information dialog will show the new license.

Using a floating license (Other users, Administrator)

If you are using a floating license that was already installed you have to enter the location of the license file.

7. On the next dialog, select **Administrator already installed server license** and then enter the **license location** (the location of the file *20sim.lic*).

After a successful entry of the location of the license location, the License Information dialog will show the new license.

4.3 Uninstalling

You can uninstall 20-sim by clicking the Uninstall command from the 20-sim start menu. Uninstallation of 20-sim *will not deactivate* your license. If you want to move 20-sim to another computer, you have to deactivate your license first before uninstalling.

4.4 Deactivation

If you want to move 20-sim to another computer, you have to deactivate your license before uninstalling the program. On the new computer you can then install the program and activate the license. To deactivate your license:

1. From the Windows **Start menu** open **20-sim**.
2. From the **Help** menu choose **License Activation**.
3. Press **the Activation button**.
4. Choose **Deactivate Current License**.

You will be asked for confirmation and deactivation will start. After a successful deactivation, your version of 20-sim has turned into the demonstration version. You can now uninstall the software and reinstall it.

4.5 Unattended Installation

An unattended installation is an installation that is performed without user interaction during its progress or with no user present at all.
To perform an unattended installation the default 'program files' installation directory run the following command on the 20-sim installer:

```
20sim.exe /S
```

It is possible to set an alternative installation directory by specifying the /D argument. It must be the last parameter used in the command line and must not contain any quotes, even if the path contains spaces. Only absolute paths are supported.

```
20sim.exe /S /D=D:\My Installation Files\20-sim 4.5
```

4.6 Unattended Uninstall

An unattended uninstall is an uninstall that is performed without user interaction during its progress or with no user present at all.
To perform an unattended uninstall from the default 'program files' installation directory run the following command on the 20-sim uninstaller:

```
C:\Program Files (x86)\20-sim 4.5\Uninstall.exe /S
```

4.7 Matlab

In 20-sim you can exchange data with Matlab / Simulink in various ways:

- Export models as m-files
- Export models as dll-files
- Export a variable to Matlab
- Export a parameter to Matlab
- Pass a variable value to Matlab every simulation step
- Load a variable value from Matlab every simulation step
- Pass a command line string to Matlab every simulation step
- Export linear systems to Matlab
- Import Linear Systems from Matlab

Troubleshooting

If this fails check if the following three points have been fulfilled.

1. Have the correct version of 20-sim and Matlab

To make a connection with Matlab, make sure you have 20-sim 4.1.3.8 or higher installed and a 32-bit version of Matlab.

The 20-sim Matlab connection makes use of a 32-bit COM component, therefore it is not possible to connect from 20-sim to a 64-bit version of Matlab, which do not provide a 32-bit COM component.

2. Matlab registration in PATH

The standard registration of Matlab in the PATH is the following (assuming Matlab is installed in Program Files and the used version of Matlab is R2011b):

C:\Program Files\MATLAB\R2011b\bin

The following path should be added:

C:\Program Files\MATLAB\R2011b\bin\win32

In this additional path some important DLL's are present that are necessary to make the connection. for example: libeng.dll. Please check if this DLL is present at this path. If not try to find this DLL in your Matlab installation and add the found path to the PATH environment.

Changing the PATH environment variable can be done in the following manner:

1. From "My Computer", right mouse button: Properties:
2. On Windows 7: choose Advanced Properties and go further as Admin, On Windows XP: you go directly to the System Properties dialog
3. Choose the tab: Advanced
4. Choose: Environment Variables.

3. COM-registration

To make the connection to Matlab, Matlab needs to be registered as a COM-component. This can be done in the following manner from the command line (run: cmd):

matlab /regserver

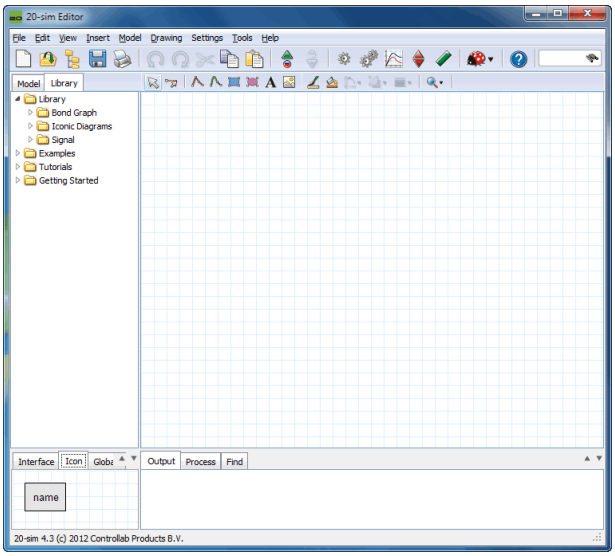
Note: The COM-registration only, is not enough for 20-sim to find the DLL's of Matlab. The second step (setting the PATH variable) must be performed.

5 Editor

5.1 Introduction

5.1.1 Editor

20-sim consists of two main windows and many tools. The first window is the *Editor* and the second is the *Simulator*. The *Editor* is used to enter and edit models. The *Editor* opens automatically when you start 20-sim:



The 20-sim Editor.

The *Editor* consists of four parts:

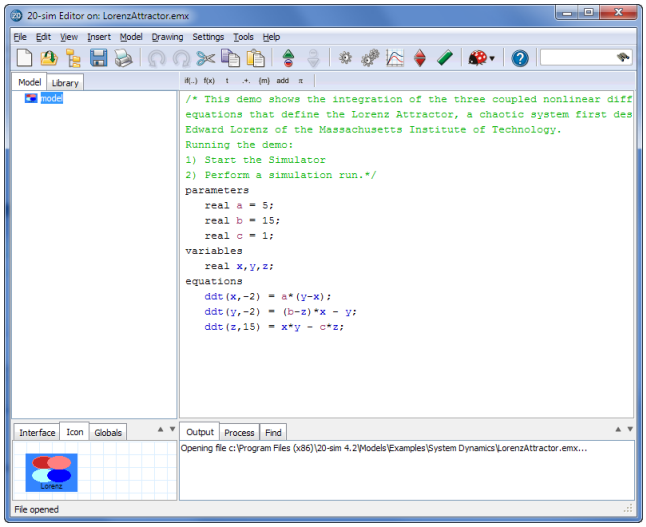
- **Model tab / Library tab**: The Model tab shows the model hierarchy, i.e. the composition of all the elements of the model. The Library tab shows the 20-sim library.
- **Graphical Editor / Equation Editor**: At the lowest level of the hierarchy this editor will show the model equations. In the higher levels this editor will show the graphical parts of your model.
- **Output tab / Process tab / Find tab**: The *Output tab* shows the files that are opened and stored. The *Process tab* shows the compiler messages. The Find tab shows the search results.
- **Interface tab / Icon tab / Globals tab**: The Interface tab shows the interface (inputs, outputs, ports) of a selected model. Double clicking it will open the Interface Editor. The Icon tab shows the icon of a selected model. Double clicking it will open the Icon Editor. The Globals tab will show the global parameters and variables of your model. Double clicking it will open the Global Relations Editor.

Using the Editor

The best way to find your way around the Editor is to read the Getting Started manual. It contains a number of topics that will explain the basics of entering equation models, graphical models and run a simulation.

5.1.2 Equation Editor

Equations models are the models at the lowest level in the model hierarchy. A model without an interface (inputs, output, ports), will have no hierarchy and is thus automatically an equation model. If you have opened an equation model, the right part of the *Editor* shows the *Equation Editor*. In the *Equation Editor* you can enter and edit equation models.



Equation model with the Equation Editor (right part).

Use

Put your mouse in the *Equation Editor* and start typing. If you click on the buttons of the taskbar, sample code is inserted. Read the language reference section to find out more.

Color Syntax Highlighting

Functions, Variables etc. are given special colors to distiguish them from the other text.

Auto Indent

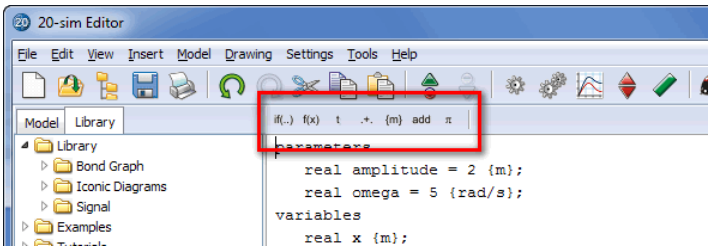
If a line is indented (using the tab), the next line will start at the same indentation. You can remove indentation by clicking the Backspace button.

Multi-Line Tabbing

You can give multiple lines of code a new indentation by clicking the tab button. Select all the lines and click Shift-tab to remove.

5.1.3 Equation Editor Taskbar

When you select an equation model, the corresponding equations are show in the *Equation Editor.* A special button bar, called the *taskbar,* is part of the *Equation Editor.* The *taskbar* helps you to enter functions, statements, templates etc.



The taskbar of the Equation Editor.

Statements: click on this button to insert if-then-else expressions and more.

Functions: click on this button to insert functions.

Specials: click on this button to insert special functions.

Operators: click on this button to insert operators.

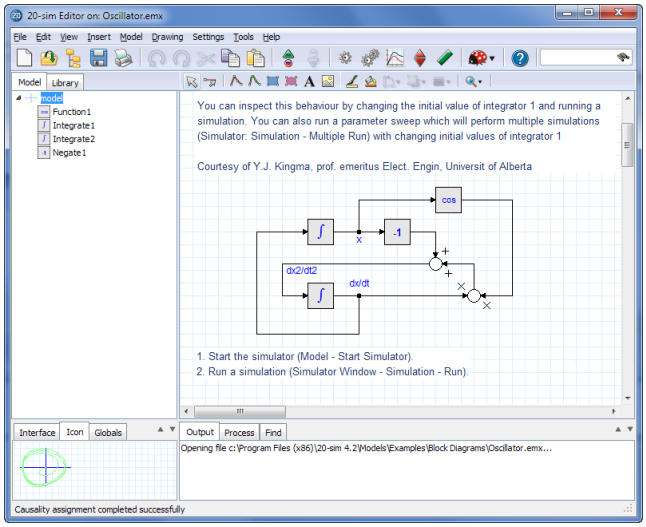
Units: click on this button to insert quantities and units.

Declarations: click on this button to insert declarations of parameters, variables and more.

Constants: click on this button to insert predefined constants.

5.1.4 Graphical Editor

If you have opened a graphical model, the right part of the *Editor* shows the *Graphical Editor*. In the *Graphical Editor* you can enter and edit block diagram models, iconic diagram models and bond graphs.



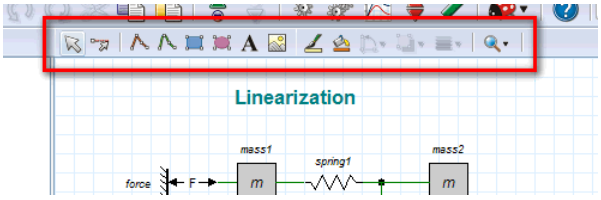
Graphical model with the Graphical Editor (right part).

Use

- Select models from the *Model Library* and drag them to the *Graphical Editor.*
- You can use the buttons of the taskbar, to connect the models and change their position.
- Drag and drop image files (bitmaps, svg images,..) to enhance the appearance of your model.
- Use the buttons of the taskbar, to create your own drawings.

5.1.5 Graphical Editor Taskbar

When you view or enter a graphical model in 20-sim it is shown in the *Graphical Editor.* A special button bar, called the *taskbar,* is part of the *Graphical Editor.* The *taskbar* helps you to select models, connect them and manipulate them. You can also use the taskbar to enter drawing objects. The taskbar is also part of the *Icon Editor.*



The taskbar of the Graphical Editor.

Selection Mode: Click this button to select models and objects

Insert Terminals: (*Icon Editor* only) Click this button to insert terminals.

Connection Mode: (*Graphical Editor* only) Click this button create connections between models.

Line: Click this button to draw lines.

Spline: click this button to draw splines.

Rectangle: click this button to draw rectangles.

Ellipse: click this button to draw ellipses.

Text: click this button insert text.

Bitmap: click this button insert bitmaps and images in svg format.

Line Color: click this button to set the line color of a selected object.

Fill Color: click this button to set the fill color of a selected object.

Rotate and Mirror: click this button to rotate or mirror a selected object.

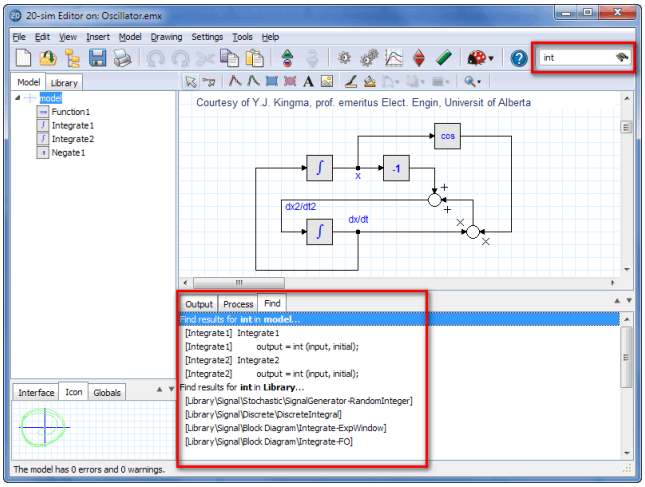
Arrange and Group: click this button to group objects, put them to the front etc.

Line Styles: click this button to select the line style of selected objects.

Zoom: click this button to select the zoom factor.

5.1.6 Find

If you make the *Editor* wide enough, you will see at the top right a *Find box*. You can enter terms here and search them throughout the model. The results are displayed in the *Find tab* at the bottom of the *Editor*.

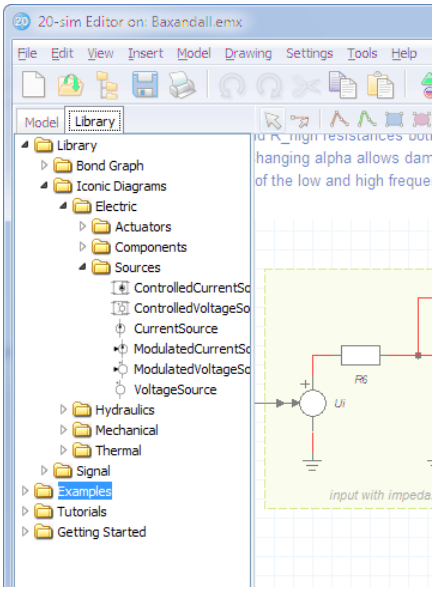


Using the Find Box to quickly search the model.

- **Scope**: The search scope depends on which level of the model you are in. Only the current submodel and all submodels below are searched. Select the top element in the Model Browser (Model tab at the left of the Editor) to search the whole model.
- **Library**: The 20-sim Library is always searched. The library results are shown below the model search.
- **Jump**: The Find tab shows the found items with an orange hyperlink. Click on the hyperlink to quickly jump to the corresponding model.
- **Find again (F3)**: Click the F3 button to quickly jump through the found items in the Find tab.
- **Menu**: You can also start a search from the menu: from the *Edit* menu select *Find*.

5.1.7 Library

In 20-sim, creating models only takes you just a few mouse clicks. By *dragging* an element from the library and *dropping* it in the graphical editor, your model is actually built the same way as you would draw an engineering scheme. 20-sim supports various model representations, such as block diagrams and iconic diagrams. These representations may be combined in one model.



You can find the library at the left of the Editor.

The library contains a various sections:

- Bond Graph: bond graph elements
- Iconic Diagrams: Physical components
- Signal: Block diagram elements
- Tutorial: example models that show you how to perform various tasks in 20-sim
- Getting Started: al the models that you need in the lessons of the Getting Started manual.

Custom libraries

You can create your own model libraries in 20-sim:

1. From the **Tools** menu click **Options** - **Folders** - **Library Folders**.
2. Add your **folder.**
3. Give it a useful **name** by clicking **Edit Label**.
4. Click OK to close the dialog.

Then you can add your own library models to the library:

1. Select the **submodel** that you want to store in your library.
2. From the **File menu** select **Save Submodel**.
3. Store the submodel in your library **folder**.

The next time you start up 20-sim, the library will show the new submodel.

5.1.8 Options

The general properties of 20-sim models are shown in the *Options* dialog.

- From the **Settings menu** choose the **Options** command.
- From the **Tools menu** choose the **Options** command.

Editor

- **Fonts**: Enter the default fonts used in graphical models (Editor) and equation models (Equation Editor).
- **Syntax Highlighting Threshold**: Select the number of characters that should be submitted for color sytax highlighting. If this number is too large, the editor may become very slow.

Plots

You can choose the default settings for a simulation plot in this tab.

- **Default Line Thickness**: Enter the default plot line thickness.

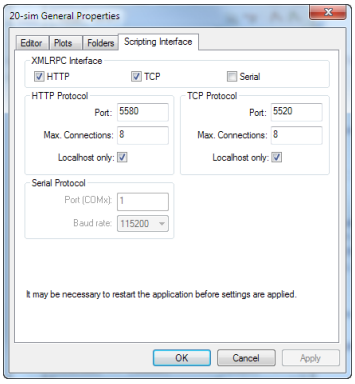
Folders

You can choose the location of libraries and files in the *Folders tab*.

- *Library* Folders: Enter the library paths and corresponding library names here. The libraries are shown in the Library tab.
- *C-code Folders*: C-code can be generated for various targets. For each target a file targets.ini defines how the C-code should be generated. You can enter the locations of ini-files here.
- *Matlab-Code folders*: 20-sim models can be exported to Matlab. Similar to C-code generation, a file targets.ini defines how the code should be generated. You can enter the locations of ini-files here.
- *Model Template Folders:* You can enter the location of model templates.

Scripting Interface

20-sim uses the XMLRPC protocol to communicate with external software and run scripts. By default, 20-sim will only accept scripting connections from your local computer (**Localhost only** option is enabled by default). The defaults setting are shown below.



The Scripting Interface.

Communication can be done by:

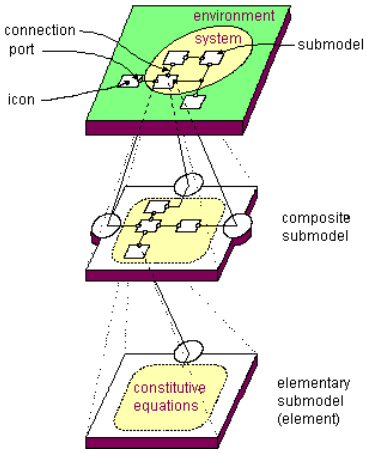
- HTTP: XMLRPC protocol.
- TCP: Optimized for performance binary communication protocol.
- Serial: Serial communication using the RS232 port of your computer.

In most cases the default settings should be fine. However, for each of these protocols, settings may be changed. Ask your system administrator for details.

5.2 Using Models

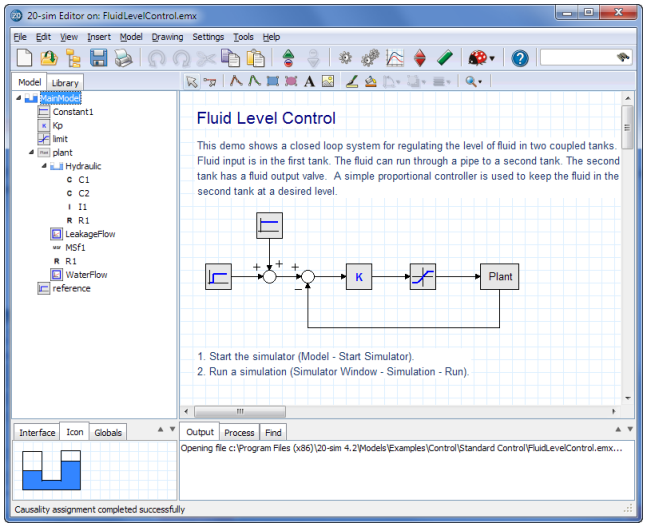
5.2.1 Hierarchical Models

Any *main model* (or system) in 20-sim may be described as a composition of lower level *submodels*. These submodels themselves may again be described as a composition of lower level submodels etc. The lowest level consists of elementary submodels, which do not consist of submodels themselves.



20-sim supports hierarchic model. The lowest model in the hierarchy is always an equation model.

You can inspect this in the *Editor*. If you click the *Model tab* at the left of the *Editor*, the *Model Browser* shows complete model hierarchy: a tree like structure showing all the submodels that are used in the model.



The Model Browser at the left of the Editor shows the complete model hierarchy.

To travel through the hierarchy, click any model in the tree. You can also select a model in the *Graphical Editor* and click *Go Up* or *Go Down* from the *Model* menu.

5.2.2 Open Models

Models in 20-sim are stored with the extension *.emx*. You can open an existing model in several ways:

1. From the **File menu**, select **Open**.
2. Click the Library tab and from the Library **drag and drop** your model to the *Graphical Editor*.
3. Open a **Windows Explorer**. **Drag and drop** your model to the *Graphical Editor.*

You can open Packed Files with the *Open* dialog by selecting a file with the extension *.emz*.

5.2.3 Save Models

Models in 20-sim are stored with the extension *.emx*. You can save an existing model in several ways.

1. From the **File menu**, select **Save**: This will save the complete model. If no filename is known a Save dialog is opened.
2. From the **File menu**, select **Save As**: This is essentially the same as the *Save* command but now the *Save* dialog is opened even if a filename is known.
3. From the **File menu**, select **Save a copy As** : This is similar to *Save As* but now a copy of the file is saved instead. This allows the user to keep on working on the model while saving intermediate experiments.
4. From the **File menu**, select **Save Submodel**: This will save the submodel that is selected in your *Graphical Editor*.
5. From the **File menu**, select **Save Encrypted**: This will allow you to store a model using encryption. If a submodel was selected, a menu is opened asking you to store the submodel or the complete model.

Once a model is encrypted, you can not decrypt it anymore! So keep cautious that you always store a non-encrypted original. Encrypted models are useful if you want others to use your 20-sim model, without seeing the underlying equations. If encrypted model is load in 20-sim, the *Go Down* command will does not work on that model.

5.2.4 Packed Files

To get a simulation running in 20-sim, the data stored in various files may be needed:

- .emx: model files
- .txt: data files (fileinput)
- .bmp: bitmap files (for use in 3D Animation)
- .dll: external dll-files (user defined external functions)
- etc.

You can pack all these files into one zip-file, using the **Pack** command from the **File** menu. With this command 20-sim will check all the files that are used and store them into a single zip-file. This option is useful for archiving and sending models by e-mail.

Pack

1. When the **Pack** command is clicked, a *Save As dialog* appears asking you to enter a name. Always save with the default extension *.emxz* because 20-sim uses this extension to recognize packed files.

After the *Save As* dialog, a *20-sim Pack dialog a*ppears, showing the collected files. In this dialog you can select the files that should be packed.

2. **Select** the files that should be packed and click the **OK** button.

Unpack

You can open Packed Files using the *Unpack* command from the *File* menu.

1. When the ***Unpack*** command is clicked, an *Open dialog* appears asking you open a file. Open a file with the extension *.emxz* because 20-sim uses this extension to recognize packed files.

After the *Open* dialog, a *20-sim Unpack dialog* appears. In this dialog you can see the file that you have selected to unpack and you can choose the method of unpacking.

2. If you want to unpack all files and put them in their original location, select the option Unpack with full path.
3. If you want to unpack all files and to a specific directory, select the option Unpack to directory.
4. Select Keep **Relative Paths** if you want to keep the original folder structure.

5.2.5 Insert Models

You can insert submodels from a library using the built in Library tab:

1. Click the Library tab to open the library.
2. Select the submodel that you want to insert and **drag and drop** it in the **Graphical Editor**.

You can also use a file browser:

1. From the File menu, select Open Browser. The File Explorer appears.
2. Select the submodel of interest and **drag and drop** it in the **Graphical Editor**.

5.2.6 Connecting Models

In 20-sim, submodels can be connected using the mouse. When a connection is created it will be displayed using straight lines. When a connection has been made you can change it into a smooth line using the right mouse menu. You can change the color of a connection using the colorbar at the bottom of the Editor. 20-sim supports two types of mouse use. "Tapping Mode" and "Pressing Mode". In the Tapping Mode you click the mouse button (do not keep it pressed but quickly "tap" the button) while making a connection. In the Pressing Mode you keep the mouse button pressed while making a connection.

Tapping Mode

To connect two submodels using straight lines, you have to:

1. In the toolbar, click to change to **connection mode**.
2. Put the **mouse pointer** on top of the **first submodel** and **click the left mouse button** (do not keep it pressed but quickly "tap" the button).
3. Drag the **mouse pointer** towards the second submodel (you will see a connection drawn from the first submodel towards the mouse pointer).
4. Put the mouse pointer on top of the **second submodel** and click the **left mouse button** again (do not keep it pressed but quickly "tap" the button).
5. While dragging from the first submodel to the second, you can click the left mouse button (do not keep it pressed but quickly "tap" the button) to create **intermediate points**.

Pressing Mode

To connect two submodels using straight lines, you have to:

1. In the toolbar, click to change to **connection mode**.
2. Put the **mouse pointer** on top of the **first submodel** and press **left mouse button** (keep it pressed).

3. Drag the **mouse pointer** towards the second submodel (you will see a connection drawn from the first submodel towards the mouse pointer).
4. Put the mouse pointer on top of the **second submodel** and release the left mouse button.
5. While dragging from the first submodel to the second, you can click the right mouse button to create **intermediate points**.

Smooth Line

To connect two submodels using a smooth line, you have to:

1. Make a **straight line** connection with **intermediate points**.
2. In the toolbar, click to change to **selection mode**.
3. **Select** the connection that was created.
4. From the **right mouse** menu select "**smooth line**".

Tips

- 20-sim will automatically detect which connection has to be made: a signals, a bond or an iconic diagram connection.
- Depending on the physical domain, every connection will have a specific color. You can change these colors in the Units Editor.
- You can toggle between connection mode and selection mode by pressing the space bar.

5.2.7 Implode / Explode

Using the **Implode** command of the **Editor** menu you can quickly create a new submodel out of a set of select submodels.

1. In the toolbar, click to change to selection mode.
2. Select the the submodels.
3. From the **Edit** menu, click the **Implode** command.

Using the **Explode** command of the **Editor** menu you can quickly open the contents of a selected submodel in the current model layer.

1. In the toolbar, click to change to selection mode.
2. Select a submodel.
3. From the **Edit** menu, click the **Explode** command.

5.2.8 Simplify Models

Use the *Simplify Model* command to simplify graphical models according to the following rules:

Block Diagrams

1. Combining splitters.
2. Combining of multiplications and/or divisions.

Bond Graphs

1. Eliminating junctions.
2. Melt equal junctions.
3. Eliminating double differences.

Iconic Diagrams

1. Eliminating nodes.

To simplify a (sub)model, you have to:

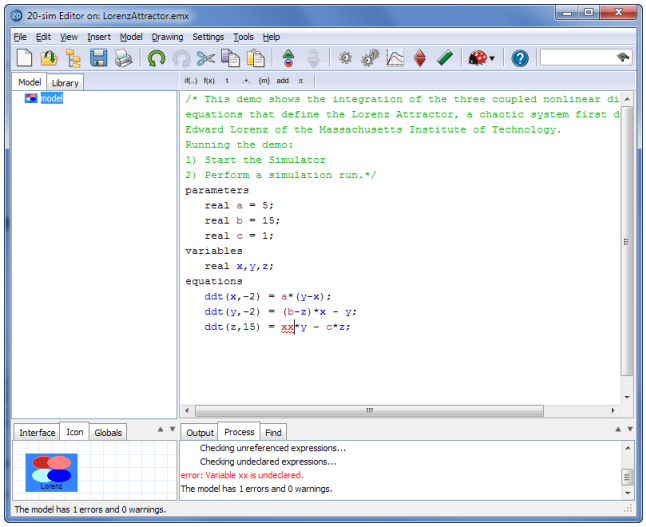
1. Select the complete model or submodel of which you want to simplify.
2. From the **Model** menu select the **Simplify Model** command.

5.2.9 Check Models

To check a complete model, you have to:

1. From the **Model** menu select the **Check Complete Model** command. Now the complete model will be checked.

If any warnings or errors are found, they are displayed in *Process tab* of the Editor.



In the Process tab the compiler messages are displayed.

2. Put you **mouse** on top of an **error message** and click.

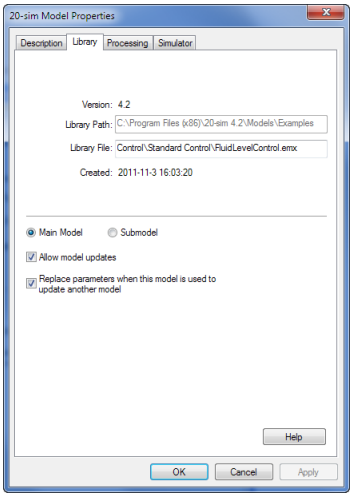
20-sim will jump to the part where the error was caused. You can try to solve the error and check again.

5.2.10 Model Properties

The properties of 20-sim models are shown in the *Model Properties* dialog. You can open the *Model Properties* in various ways.

- Select the proper model in the Model Browser and then click **Properties** from the **Right Mouse** menu.
- Select the model in the Graphical Editor and then click **Properties** from the **Right Mouse** menu.
- From the **File menu** choose the **Properties** command.
- From the **Settings menu** choose the **Model** command.
- From the **Settings menu** choose the **Submodel** command.

Depending if a *submodel* was selected or a *main model*, various tabs are visible:



The model properties dialog.

Description

In this tab you can enter the name of the submodel and enter various fields to classify a model. The most important elements are:

- **Name**: Enter a unique name for the model.
- **Help Page**: Enter the location of a help page that should be displayed when you click *F1* or *Help*.

Library

In this tab you can find information on location of the model and some important properties.

- *Version*: The version of 20-sim that was used to create the model.
- *Library Path*: If the model was stored in a known library, the path is shown.
- *Library File*: The relative location of the submodel in the library or the absolute location of the model submodel on your computer.
- *Created*: The date of creation.
- *Main Model / Submodel*: Shows if the model is a main model or submodel.
- *Allow model updates*: Allow the Check for Model Updates command to check if there is an update available for this model.
- *Replace parameters when this model is used to update another model*: If you drag and drop a model from the library on top of an existing model, it will be replaced. Select this option if you want the original parameters to be replaced as well.

Background Image

In this tab you can specify a background image for the model. If the model is a graphical model, in the background the image is shown.

- *None*: default, no image shown.
- *Inherit From Parent*: Use the background that was defined one model higher in the model hierarchy.
- *Static Image*: Specify the image file to be used.
- *Script File:* You can specify a script file if you want to use a dynamic background.
- *Scaling*: Scaling of the background image.

Processing

This tab is only available for main models. 20-sim can operate in two modes: *Debug Mode* and *Fast Mode*. In this tab you can define settings for both modes.

Warnings/Errors

- *Model contains algebraic variables*: See the section on algebraic loops.
- *Model contains algebraic loops*: See the section on algebraic loops.
- *Algebraic variables solved*: See the section on algebraic loops.
- *Model contains constraint variables*: See the section on constraints.
- *Constraint variables solved*: See the section on constraints.
- *Model contains dependant states*: The model contains differential equations which could not be solved.
- *Dependent states are transformed*: The model contains differential equations which were solved.
- *Output is not used*: Output signals that are not used for connections to other models.
- *Input is not used*: Input signals that are not used for connections to other models (to make such models simulate, a zero input value is applied to all not-connected inputs)
- *Port is not used*: Ports that are not connected.
- *Parameter is not used*: Parameters that are defined but not used.
- *Matrix is assigned a scalar*: A matrix is assigned to a scalar.
- *Variable is not used*: Variables that are defined but not used.
- *Variable Multiple set*: Variables that are assigned a value more than once.
- *Variable is never given a value*: Variables that are never assigned a value.
- *Variable is set but not used*: The variable is assigned a value but never used.
- *Unit Conversion when SI disabled*: Not relevant yet.
- *Unit Conversion when SI enabled*: A unit conversion was found.
- *Unit missing for variable when SI disabled*: Not relevant yet.
- *Quantities Mismatch*: Variables with different quantities are compared.
- *Unit is unknown*: A unit has been used that is not available in the Quantities and Units file.
- *Equations interpreted as code*: Equations can only be valid when interpreted as sequential code. For example when variables are assigned more then once.
- *Possible loss of data at type conversion*: A type conversion has been found that may lead to loss of precision digits.
- *Type conversion found*: A type conversion has been found.

- *Conversion of booleans found*: A type conversions involving booleans has been found.

Model optimization

- *Transform Dependents States*: Try to solve differential equations.
- *Solve algebraic variables*: Try to analytically solve algebraic loops.
- *Remove Redundant Equations*: Try to remove equations that do not influence model behavior.
- *Optimize Equation Structure*: Try to separate equations into an input section, a dynamic section and an output section.
- *Optimize Static Expressions*: Move expressions with constant output throughout the simulation to the static part of the model equations.
- *Optimize Duplicate Expressions*: Calculate and expression only once and use the result everywhere.
- *Optimize Divisions*: Rewrite divisions as multiplications whenever possible.

Simulator

This tab is only available for main models. You can set the simulator properties here.

- *Initialize variables at start of simulation*: You can choose to set any variable that was not given a proper value, to zero at the start of a simulation. You can also choose to set the value to NaN to make detection more easy.
- *Calculate Hold Instruction During Initialization*: Calculate an output for hold functions during the initialization of the a simulation.

5.2.11 General Properties

The general properties of 20-sim models are shown in the *Options* dialog.

- From the **Settings menu** choose the **Options** command.
- From the **Tools menu** choose the **Options** command.

Editor

- **Fonts**: Enter the default fonts used in graphical models (Editor) and equation models (Equation Editor).
- **Syntax Highlighting Threshold**: Select the number of characters that should be submitted for color sytax highlighting. If this number is too large, the editor may become very slow.

Plots

You can choose the default settings for a simulation plot in this tab.

- **Default Line Thickness**: Enter the default plot line thickness.

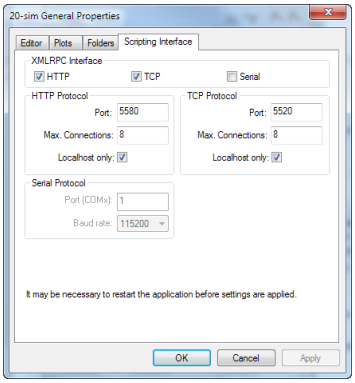
Folders

You can choose the location of libraries and files in the *Folders tab*.

- *Library* Folders: Enter the library paths and corresponding library names here. The libraries are shown in the Library tab.
- *C-code Folders*: C-code can be generated for various targets. For each target a file targets.ini defines how the C-code should be generated. You can enter the locations of ini-files here.
- *Matlab-Code folders*: 20-sim models can be exported to Matlab. Similar to C-code generation, a file targets.ini defines how the code should be generated. You can enter the locations of ini-files here.
- *Model Template Folders:* You can enter the location of model templates.

Scripting Interface

20-sim uses the XMLRPC protocol to communicate with external software and run scripts. By default, 20-sim will only accept scripting connections from your local computer (**Localhost only** option is enabled by default). The defaults setting are shown below.



The Scripting Interface.

Communication can be done by:

- HTTP: XMLRPC protocol.
- TCP: Optimized for performance binary communication protocol.
- Serial: Serial communication using the RS232 port of your computer.

In most cases the default settings should be fine. However, for each of these protocols, settings may be changed. Ask your system administrator for details.

5.2.12 Check Energetic Behavior

If you select the **Check Energetic Behaviour** command of the **Model** menu, 20-sim will check the complete model and generate additional variables. The additional variables will only be generated for submodels with powerports:

- power for every power port.
- the net power that flows into the submodel (total sum of all the powers).
- the net energy of the submodel (integrated net power)

This command is the same as Check Complete Model command but will generate extra variables:

- *Sumodelname**port.power*: the net power flow of the port.
- *Sumodelname**summated_port_power*: the net power flow into the model.
- *Sumodelname**summated_port_energy*: the net energy of the model.

You can inspect these variables during simulation in the Variable Chooser.

5.2.13 Analyze Causality

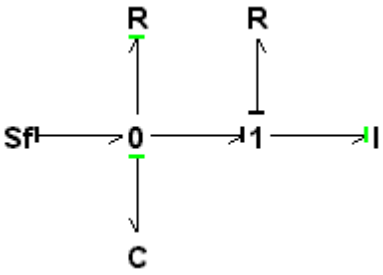
Causal analysis is the procedure to get the model equations correct form. For Bond Graph models this means that the direction of the efforts and flows of the bonds have to be determined. The result of the analysis is displayed by causal strokes (denoted by **|**). For Iconic Diagrams this means that the direction of the across and through variables of the connections have to be determined. The result of the analysis is displayed by causal arrows (denoted by **->**).

To perform causal analysis you have to:

1. From the **Model** menu select the **Analyze Causality** command. Now causality will be assigned in the complete model.

Bond Graphs

- Causal strokes are shown in Bond Graphs automatically.
- Setting Causality can also be done by hand. Just select a bond and choose Properties from the right mouse menu. A menu pops up in which you can set causality by hand. The corresponding causal stroke is displayed in green (denoted by |).

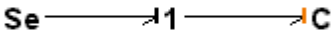


User defined causality (denoted by |).

- If 20-sim fails to perform a causal analysis of the model (a **causal conflict**), the corresponding bond is displayed red.

A causal conflict (denoted by the red bond).

- Some submodels have a preferred causality. If assignment of the preferred causality is not possible (because of other constraints), the corresponding causal stroke is displayed in orange (denoted by |).

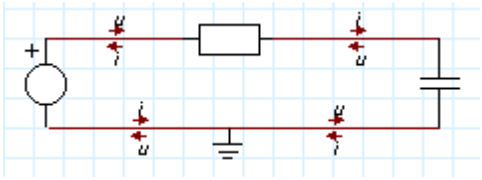


Preferred causality not assigned (denoted by |).

- To see the order in which automatic causality assignment has been performed, choose **Causality Info** from the **View** menu.

Iconic Diagrams

- Causality in Iconic Diagrams is only shown when you select the **Causality Info** command of the **View Menu**.



Causal Information shown by arrows.

- If 20-sim fails to perform a causal analysis of the model (a **causal conflict**), the corresponding connection is displayed red.

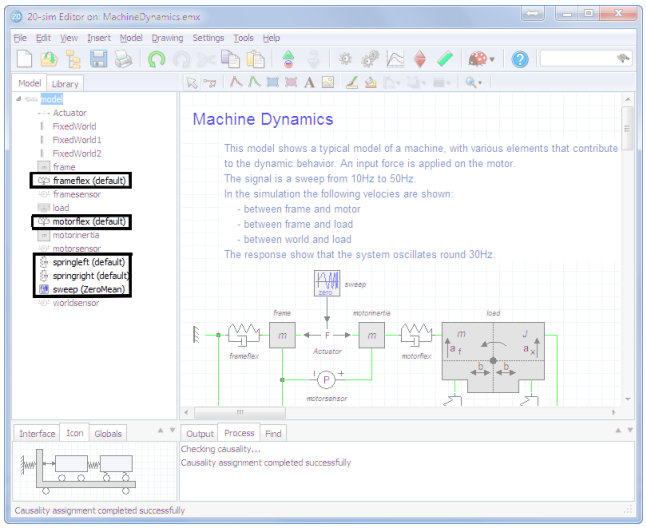
5.2.14 Implementations

A model may have various implementations. It means that you can combine several versions of a model and store them in one file. If you drag and drop a model with implementations, 20-sim will ask which one to use.

Example

A fine example of a model with various implementations is the model *MachineDynamics.emx* from the *Examples\1D mechanics* library. This example model has several submodels with implementations:

- SpringDampers
- SignalGenerator-Sweep



The selected implementation is shown between brackets in the Model Browser.

When you drag and drop a model with implementation into the *Graphical Editor* you will be asked which implementation to use. In the Model Browser, the chosen implementation is shown between brackets (*Default* and *ZeroMean* in the picture above). If you hover the mouse above a submodel with an implementation, you also see the chosen implementation between brackets. While modeling, you may decide to change the implementation. This can easily be done by selecting the submodel and clicking the right mouse menu. Choose *Implementation* and select the implementation that you want to use.

Construction

You can make submodels with implementations yourself:

1. **Select** a **submodel**.

2. From the **right mouse** menu choose **Edit Implementation** and **Add New**.
3. Enter the **name** of the implementation and click **OK**.
4. **Store** the submodel using the **Save Submodel** command of the **File** menu.

Changes

You can change the implementation:

1. **Select** the submodel.
2. From the right mouse menu choose **Edit Implementation** and **select** the **implementation** that you want to change.
3. Make the **changes** that you want in the submodel.
4. **Store** the submodel using the **Save Submodel** command of the **File** menu.

5.2.15 Continuous-Time and Discrete-Time Models

In 20-sim you can model continuous-time systems, discrete-time systems and combinations of continuous-time systems and discrete-time systems (hybrid) systems. By default, models in 20-sim are continuous-time, and simulated using continuous-time integration methods. However, the program will automatically identify discrete-time parts in a model and simulate them at a fixed rate.

Continuous-Time

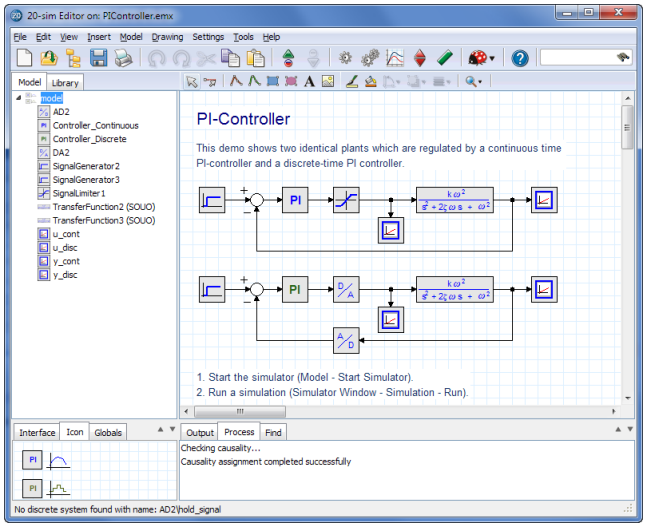
Continuous-time models describe real-world processes. To simulate a continuous-time model, the model will be calculated many steps per second to get a result that is a good representation of the real-world process. If you model a car suspension system, about 100 steps per second might be sufficient to give a good representation of the behaviour of the car. If you model an electronics circuit, 100.000 steps per second might be required. The default integration method in 20-sim will automatically choose the required steps per second.

Discrete-Time

Discrete-time models describe processes that run on computers at a fixed rate. To simulated a discrete-time model, you only have to indicate the rate at which the model should be calculated. This is called the sample rate or sample frequency. The sample rate of a discrete-time part can be set in the Simulator.

Mixed Models

In 20-sim you can model in continuous-time and discrete-time. Mixed models containing discrete-time parts and continuous time parts are supported as well. The example below shows a continuous time model of a PI-controlled system at the top. At the bottom the same system is shown with a discrete time controller and a continuous-time system. The discrete-time part is indicated by green input and output lines.



Discrete-time parts of a model are indicated by green inputs and outputs.

Identification of Discrete-Time Parts

In 20-sim every model is continuous-time by default. Discrete-time parts of a model are identified by the occurrence of special functions:

- sample
- hold
- next
- previous
- sampletime

Normally these functions are hidden in the equations that describe a submodel, but 20-sim will automatically recognize that the input or output of such a function should be discrete-time. If the output is discrete-time it is propagated to the next function, which output is then also "*tagged*" as discrete-time, and so on until the whole discrete-time part of a model is identified. A discrete time part of a model is indicated by green inputs and outputs (see previous picture).

Connection

Continuous-time parts of a model can be connected to discrete-time parts, by using the functions:

- sample: The sample function has a discrete-time input and continuous-time output.
- hold: The hold function has a continuous-time input and a discrete-time output.

You can find these functions for example in the following library models:

- DA.emx (Discrete to Analog Converter)

- AD.emx (Analog to Digital Converter)
- Encoder.emx (Optical Encode)

Forbidden Functions

In a discrete part of a model, certain functions are not allowed, because they are specifically meant for the use in continuous-time models:

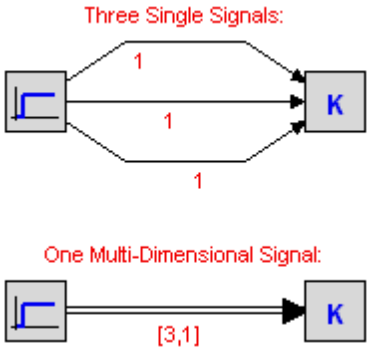
- algebraic
- constraint
- ddt
- dly
- event
- eventdown
- eventup
- frequencyevent
- int
- limint
- tdelay
- timeevent

If you use a library model that contains such a function in a discrete-time part, an error message will be given.

5.2.16 Working with Multi-Dimensional Models

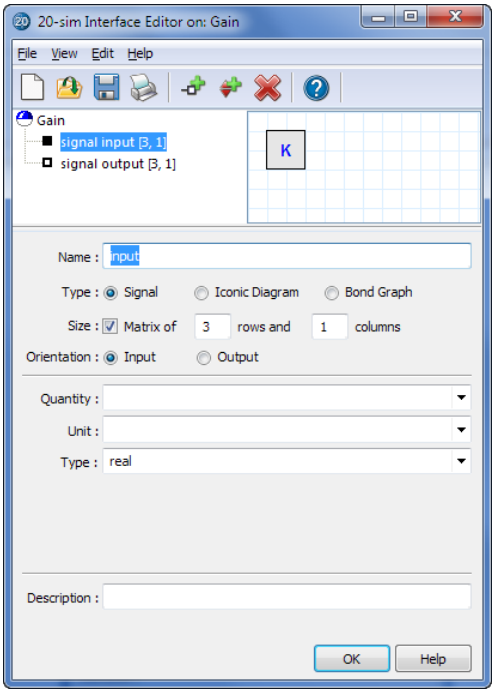
Multi-Dimensional models are models that have connections which are of a size [n,m] with either n or m larger than one. Here *n* is the numbers of rows and m is the numbers of columns. More information on this matrix notation can be found in the topic on matrices and vectors.

The advantage of multidimensional models is shown in the example below. On top two submodels are connected by three (single) signals. The same submodels are also shown below, connected by one multi-dimensional signal of size [3,1]. If these submodels have to be used a lot, multi-dimensional signals or multi-signals, are easier to use. Multi-dimensional signals, bonds and connections are shown in 20-sim by double lines.



The Secret: Multi-Dimensional Ports

The secret of multi-dimensional models is hidden in the model interface. If you open the Interface Editor to inspect an interface, you will notice that each port (signal, iconic diagram or bond graph) has a default **size** (dimension) of 1 (1 Row, 1 Column). By increasing the number of rows and/or columns this size can be increased.



Definition of Multi-ports: increase the number of rows or columns.

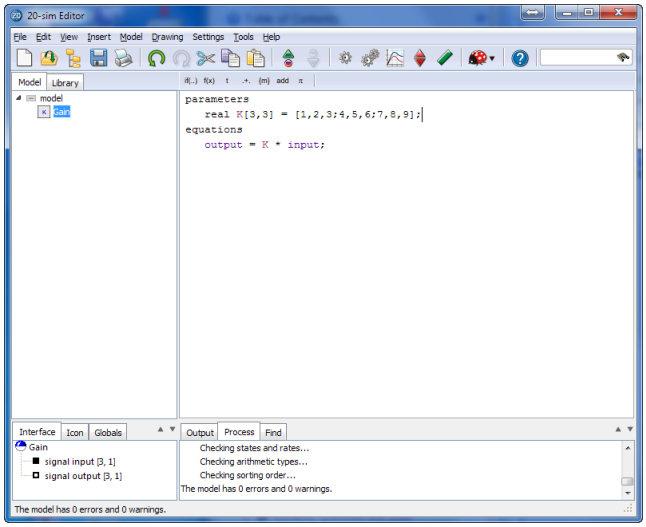
The Trick: Automatic Connection

When two submodels are connected, 20-sim will automatically check for the port-sizes and create a connection:

1. If the port-sizes are unequal, an error message will be generated.
2. If both ports have size one, a single signal, bond or connection will be drawn.
3. If both ports have a size larger then one, a multi-signal, multi-bond or multi-connection will be drawn.

The Finishing Touch: Matrices and Vectors

Behind each multi-dimensional submodel, there are equations in the end. To combine multi-dimensional ports with equations, you can use matrices and vectors. 20-sim has a large connection of matrix and vector functions and operators, to help you define any possible matrix equation. For example the gain model in the picture on top, could have been described as shown below.



Use the matrix and vector notation to define the model equations.

5.2.17 Exporting Models

There are several methods to export 20-sim models.

Export to previous versions of 20-sim

You can save a model file using the **Save** command or the **Save As** command from the **File** menu. If you want to save part of you model, use the **Save Submodel** command of the *File* menu. A *Save* dialog will be opened in which you can enter the model name and location. At the bottom of the dialog, you can use the *Save As Type* box to select the file type:

Model files (*.emx)	Standard 20-sim 4.x format
Packed files (*.emz)	Zipped file including all linked files (e.g. datafiles, dll's, bitmaps etc.)
20-sim 3.6 files (*.emx	Export to 20-sim 3.6 format
Sidops text files (*.txt)	Save model as text file.

Export to zip file

You can save a model and all externally linked files into one zip-file, using the **Pack** command of the *File* menu. With the **Unpack** command of the **File** menu, you can open these zip-files.

Export to Clipboard

From the **File** menu click **Export** and **To File** to export a selected model as a drawing to the clipboard (*Windows Enhanced Metafile* format).

Export to File

From the **File** menu click **Export** and **To File** to export a selected model as a drawing to a file (*Windows Enhanced Metafile* format).

Export to HTML

From the **File** menu click **Export** and **To HTML** to export the model, simulations etc. to an HTML file for the use in a word processor.

Export to Bitmap

From the **File** menu click **Export** and **To Bitmap** to export a selected model as a drawing to a file (*png* format).

Export to Matlab (m-file)

To export 20-sim models to Matlab, from the *Editor* you have to select the **File** menu and then click **Export to Matlab/Simulink** command. This will open a *Matlab-Code Generation* dialog. You can choose to export a complete model or a submodel. In both cases, three m-files are generated:

- ModelName.m: A file containing the model equations and special 20-sim functions. The model equations are directly translated from the equations that are shown with the Show Equations command of the Model menu.
- ModelName_run.m: An example file showing you how to run a simulation with the exported model in 20-sim.
- ModelName_print.m: This file contains a function that is used in the run file.

Export to Simulink (m-file)

20-sim models are exported to Simulink S-functions. S-functions are block diagram elements of which the internal description can be an *m-file* or a *dll-file*. 20-sim can export both types. M-files can be opened from within Matlab and are therefore more accessible. *Dll-files* are compiled out of C-code, which makes them inaccessible, but a lot faster. Simulink does not support powerports. Therefore the powerports in a 20-sim model are translated to input and output ports.

To export 20-sim models to Simulink with an m-file description, from the *Editor* select the *File* menu and then click *Export to Matlab/Simulink* command. This will open a *Matlab-Code Generation* dialog. You can choose to export a complete model or a submodel. In both cases, two files are generated:

- ModelName_.mdl: The exported 20-sim model based on a m-file that describes the model.
- ModelName.m: An m-file that contains the model equations and special 20-sim functions. The model equations are directly translated from the equations that are shown with the Show Equations command of the *Model* menu.

Export to Simulink (dll-file)

To export 20-sim models to Simulink with an *dll-file* description, from the *Simulator* you have to select the **Tools** menu and then click the **C-code generation** command. This will open the *C-code Generation* dialog where you can choose to export to a Simulink S-function.

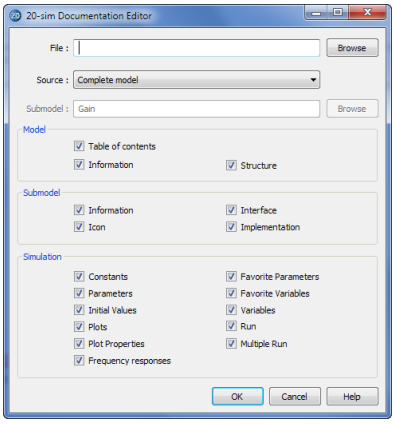
If you select the *OK* button, ANSI C-Code will be generated and compiled into a dll-function. Some files will be generated of which two are needed in Simulink:

- ModelName_.mdl: The exported 20-sim model based on a dll-file that describes the model.
- ModelName.dll: A dll-file that contains the model equations and special 20-sim functions. The model equations are directly translated from the equations that are shown with the Show Equations command of the Model menu.

5.2.18 Documentation Editor

The *Documentation Editor* can be open from the *File* menu in the 20-sim *Editor* by clicking *Document.* The *Document Editor* can be used to generate an html document that can be used for Word Processors and Presentation software.

The *Document Editor* will generate an html document that is fully hyperlinked. From a table of contents you can click to go to the desired part of the document. You can also click in pictures to go to the desired submodels.



The 20-sim Documentation Editor.

General Items

- *File*: Select the output file.
- *Source*: Select the whole model or parts of it.
- *Submodel*: If documentation of a part of the model should be documented, select the desired submodel.

Model

- *Table of Contents*: Select this option to start the document with a table of contents.
- *Information*: Information on the creation date, user, file location etc.
- *Structure*: The model hierarchy (list of all submodels, their submodels etc.).

SubModel

- *Information*: Information on the creation date, user, file location etc.
- *Icon*: The picture of the submodel icon.
- *Interface*: The inputs, outputs and ports.
- *Implementation*: The model implementation with equations at the lowest level.

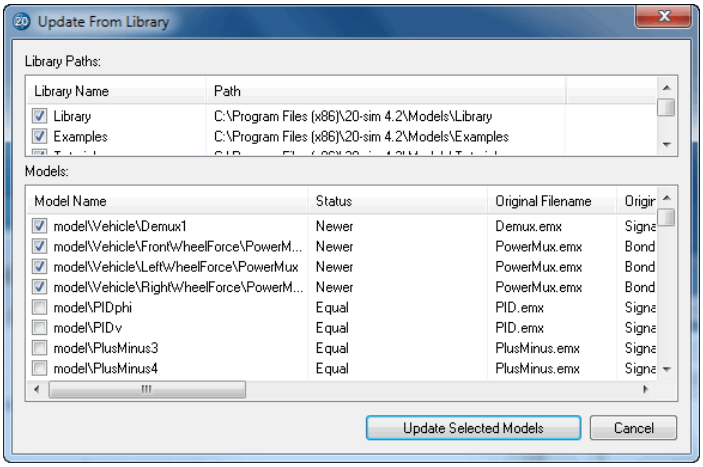
Simulation

- *Constants*: List of the constants that are used in the model.
- *Parameters*: List of the parameters that are used in the model.
- *Initial Values*: List of the starting values of the states of the model.
- *Plots*: Show pictures of the plots.
- *Plot Properties*: Show the plot settings.
- *Frequency Response*: Show the frequency responses.
- *Favorite Parameters*: List of the favorite parameters that are used in the model.
- *Favorite Variables*: List of the favorite variables.
- *Variables*: List of the variables.

- *Run*: list of the run settings.
- *Multiple Run*: List of the multiple run settings.

5.2.19 Check for Model Updates

Most 20-sim models will be built using predefined submodels from a library. You can use the *Check for Model Updates* command of the *Models* menu to see if there are updates of these submodels. A window will open, showing you all the predefined library submodels.



Items

- *Library Paths*: This part shows the libraries that are included in the search for predefined submodels. You can select or deselect these libraries. You can add your own libraries in the General Properties window.
- *Models*: This part shows all the submodels of which a match has been found in the libraries.
  - *Model Name*: The local name of the submodel.
  - *Status*: Shows if the match is *older*, *equal* or *newer*.
  - *Original Filename*: Original name of the submodel file.
  - Original Directory: Original location of the submodel file.
  - Model Time Stamp: Time when the used model was saved.
  - Library Time Stamp: Time when the library model was saved.

Note

- The *Check for Model Updates* command will only check the time stamp of a model. If you have changed a submodel, without saving it, the time stamp is not altered.
- To prevent models from being included in the updates check, switch off the Allow model updates option of the Model properties.

- If you are not sure if a submodel should be changed, open the new library submodel in a separate *Editor* first.

5.2.20 Backgrounds

Graphical models may be shown with a background images. The background image can be defined for the main model and all submodels:

1. In the **Editor** from the **Settings menu** choose the **Model** - **Background Image**.

or you can choose to set the background image for a specific submodel:

2. Select a submodel and from the right mouse menu choose **Properties - Background Image**.

In this tab you can specify a background image for the model. If the model is a graphical model, in the background the image is shown in the Editor. The background is always shown when a model is printed to file. The following options can be chosen:

- *None*: default, no image shown.
- *Inherit From Parent*: Use the background that was defined one model higher in the model hierarchy.
- *Static Image*: Specify the image file to be used.
- *Script File:* You can specify an xml file with a script if you want to use a dynamic background.
- *Scaling*: Set the scaling of the background image.

Static & Dynamic

Static backgrounds are just images that you can choose. Dynamic Background are .svg files in which you can enter 20-sim tokens (e.g. the model name, date, creator etc.). The tokens are replaced by 20-sim and then the .svg file used in 20-sim.

To get a clear understanding of this mechanism, have a look at the example model *Examples\Tips and Tricks\Background images\BackgroundImage.emx.* You can open it from the model library. In this model backgrounds are explained.

- In the same folder where the model is stored, a background image named *background.svg* can be found. This an image file which contains 20-sim specific tokens like %SUBMODELNAME% and %DESCRIPTION%. These tokens will be automatically replaced by 20-sim. Have a look at the example model to see the list of available tokens.
- In the same folder where the model is stored, an .xml file named *dynamic_background.xml* can be found. This is a script file that replaces the tokens.

To create your own dynamic backgrounds:

1. Create an .svg image with the tokens that you want to use. Store it in the folder where your model is located.
2. Copy the .xml file *dynamic_background.xml* to your own .xml file and make it refer to the new .svg file. Store it in the folder where your model is located.

3. In the 20-sim model open the *Model Properties*. In the *Background Image* tab, select *Script File* and enter the name of your .xml file.

5.3 Compiling

5.3.1 Compiling Models

When you enter a model and open the Simulator or when you select the **Check Complete Model** command of the **Model** menu, the model will be compiled. During compilation, 20-sim will perform a number of tasks.

Error Checking

- *Type checking*: 20-sim will search for illegal type conversion and possible loss of data.
- *Unit Checking*: Many models in 20-sim use units and quantities. 20-sim will check for a proper use of these units and quantities.

Optimizing Model Structure

- *Analyze causality*: determine the causal order of equations.
- *Integral Form*: Equations are changed to integral form as much as possible.
- *Solving differential equations*: Differential equations are solved directly if possible.
- *Solving Algebraic Loops*: Algebraic loops are solved to their analytical solution, where possible.
- *Optimizing Equation Structure*: Some equations have to be calculated only once during each simulation step, because they do not influence the model dynamics. Depending on their relation to the model dynamics (needed for or a result of) these equations are calculated before the model dynamics (input equations) or after the model dynamics (output equations)
- *Remove redundant equations*: A lot of model equations are just assignments like var1 = var2, where one variable is already known. 20-sim reduces these equations from the model and maps the unknown variables to known variables.

After these tasks a complete equation model is created, which can be inspected using the **Show Equations** command of the **Model** menu. After the complete equation model has been created the following tasks are applied:

Compiling

- *Find the correct order of execution*: All equations will split up into their most simple form and rewritten into the correct order of execution.
- *Interpreter Code*: Create interpreter like code out of the equations. This is low level code that can be understood by the simulation algorithms.
- *Machine Code*: When you have the Built-In compiler option selected, the interpreter code is compiled into platform specific 32-bit machine code. This code uses the full power of native Pentium and 486 instructions. The result is a dramatic increase of simulation speed: 100% to 400 %, depending on the kind of model used!

Simulation Code

The resulting code, either interpreter code or machine code, is used in the Simulator to perform simulation runs with and therefore also know as simulation code. As we have learned, the simulation code can be quite different from the original model equations. To denote the function of each original model variable in the simulation code, 20-sim uses the following names:

- independent rate
- independent state
- dependent rate
- dependent state
- algebraic loop in
- algebraic loop out
- interesting variable
- hidden variable

5.3.2 Causal Form

Equations within 20-sim may be entered in random form. During compilation, 20-sim will automatically try to rewrite equations into a causal form and set them in a correct order. I.e. a form where all output variables are written as function of input variables. Consider for example the following model:

```
variables
    real u,z;
equations
    u = sin(time);
    u = cos(z);
```

Here the (input) variable u is given by the equation *u = sin(time)*. Consequently the (output) variable z should be written as a function of u. This is exactly what 20-sim will try to do while compiling the model into simulation code. I.e. the function cos will be inverted and the model will be rewritten to:

```
variables
    real u,z;
equations
    u = sin(time);
    z = arccos(u);
```

Some functions cannot be inverted. Consequently not all equations can be rewritten. 20-sim will report this to the user, during model checking.

Indifferent Causality

For some models the causal form is not known beforehand. For example the iconic diagram model that describes a damper can be written as:

```
parameters
    real d;
equations
    p.F = d*p.v;
```

Here *p.F* denotes the force and *p.v* denotes the velocity of the powerport *p*. There is no preferred input (force or velocity). This can be explicitly stated, by giving the powerport *p* an indifferent causality. During compilation 20-sim will determine whether *p.F* or *p.v* is the input variable and consequently rewrite the equations.

Preferred Causality

For some models there is a preferred causal form. For example the iconic diagram model that describes a spring can be written as:

```
parameters
    real k;
equations
    p.F = (1/k)*int(p.v);
```

Here *p.F* denotes the force and *p.v* denotes the velocity of the powerport *p*. The equation is written in integral form which is preferred. Consequently the preferred input is the velocity. Should the force be the input, the equation must be rewritten to a differential form, which is leads to less efficient simulation. This can be explicitly stated, by giving the powerport *p* an preferred causality. During compilation 20-sim will try to keep the model in this preferred form. If this is not possible the equations will be rewritten to the less preferred form.

Preferred causality has a lower priority than fixed causality. During compilation 20-sim will first assign all models with a fixed causality and then all models with preferred causality.

Likes Causality

For some models there is a causal form which is liked more than the other. For example the iconic diagram model that describes a parasitic volume can be written as:

```
effortincausality(p) then
        p.phi = 0;
else
        volume_ratio = int(p.phi/V);
        p.p = B * volume_ratio + p_initial;
end;
```

Here *p.phi* denotes the volume flow and *p.p* denotes the pressure of the powerport *p*. The equation is written in integral form which is liked. Consequently the preferred output is the pressure. Should the pressure be the input, the equation gives a zero flow as output. During compilation 20-sim will first try to keep all models the liked form. If this is not possible the equations will be rewritten to the other form.

Liked causality has a lower priority than preferred causality. During compilation 20-sim will first assign all models with a preferred causality and then all models with likes causality.

Fixed Causality

For some models there is only one causal form. For example a simple iconic diagram model that describes coulomb friction can be written as:

```
parameters
    real Rc;
equations
    p.F = Rc*abs(p.v);
```

Here *p.F* denotes the force and *p.v* denotes the velocity of the powerport *p*. The equation cannot be rewritten to a form where *p.F* is the input. This can be explicitly stated, by giving the powerport *p* a fixed causality. During compilation 20-sim will try to keep the model in this fixed form. If this is not possible an error message will be generated.

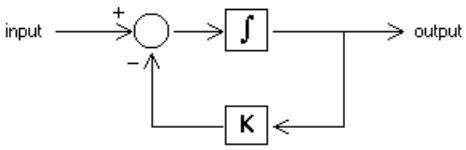
Fixed causality has a lower priority than a likes causality. During compilation 20-sim will first assign all models with a likes causality and then all models with fixed causality.

Setting Causality

For some models, the equations are too complex to analyze causality. To help 20-sim, using the right causality, you can set causality for every port in the Interface Editor.

5.3.3 Integral Form

Consider the following first order linear model:



This model can be described by the dynamic equation:

*output = int(input - K*output)*

Now look at the following model:

This model can be described by the dynamic equation:

output = (input - ddt(output))/K

Note that we can rewrite this equation as:

*ddt(output) = input - K*output*

or

*output = int(input - K*output)*

This is the same equation as the previous model! Apparently, both models are the same! Both models can therefore be described by the dynamic equations:

*ddt(output) = input - K*output*
*output = int(input - K*output)*

We call the first equation the *differential form* (no integrals). The second equation is called the *integral form* (no derivatives). In 20-sim, models can be entered in integral form as well as the differential form.

Solving Differential Equations

During compilation, 20-sim will automatically try to rewrite equations into the integral form, since this leads to more efficient simulation. Sometimes an integral form cannot be found. Then algorithms will be applied to solve the differential directly. For example an equation like:

*output = ddt(sin(a*time))*

will be replaced by the following equation (applying the chain rule and using the known derivative for the sine function):

*output = a*cos(a*time)*

Sometimes a differential cannot be solved directly. Then only the Backward-Differentiation Methods can be used for simulation.

Simulation Code

After compilation simulation code is generated. The equation in integral form:

*output = int(input - K*output)*

will be written as:

independent state = output
*independent rate = input - K*output*

and can be handled by all integration methods. The equation in differential form:

*ddt(output) = input - K*output*

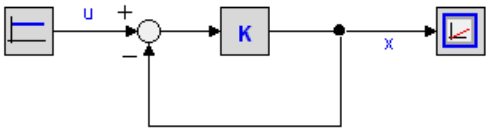
will be written as:

*dependent rate = input - K*output*
dependent state = output

and can only be handled by the Backward-Differentiation Methods.

5.3.4 Algebraic Loops

An algebraic loop in a model is a loop consisting of elements without "memory like" functions. To calculate the variables in this loop, the variable values themselves are needed. Consider the following example of an algebraic loop in an amplifier with negative feedback:



Standard derivation of a simulation model would yield:

$$x = K*(u-x)$$

The variable x depends on its own value and must be solved by **iteration**. In 20-sim every simulation algorithm is accompanied by an iteration routine. Fortunately 20-sim is able to solve many algebraic loops at equation level. For this model this leads to the analytic solution the system:

$$x = K*u/(1+K)$$

Simulating algebraic loops

Although 20-sim contains a sophisticated algorithms to find analytic solutions, the occurrence of unbreakable loops can not always be prevented. The occurrence of algebraic loops may lead to an increase of simulation time, or even stop simulation when iteration fails.

The best solution for these problems is to have a critical look at the model and change the order of calculations in a model. Possible solutions are:

- Algebraic Loops occur when the order of calculations is arbitrary. When an algebraic loop occurs in an equation model or in a set of equation models, you may change the order of calculation by rewriting the equations. The calculation order in bond graph models can be changed by introducing hand-defined causality.
- Introduce 'parasitic' energy storage elements (e.g. a small mass, a small capacitor etc.) to break an algebraic loop. These elements introduce however, large poles in the state equations, which might increase the simulation time considerably.
- Delete elements in the algebraic loop which are not relevant for the model's simulation output (e.g. small dampers, very stiff springs etc.). Care should however be taken, since correct deletions are not always possible and require considerable modeling skill and intuition.
- Combine dual elements. Sometimes elements of the same type can be combined by adding the parameter values (e.g. combining a mass m1 and a mass m2 to a mass m1 + m2). This will in most cases decrease the amount of algebraic loops.

5.3.5 Order of Execution

Equations within 20-sim may be entered in random form. During compilation, 20-sim will automatically try to rewrite equations into a correct order of execution. I.e. a form where all output variables are written as function of input variables and output variables of previous lines. Consider for example the following equations:

```
variables
    real u,z;
equations
    z = sin(u);
    u = cos(time);
```

Here the (input) variable *z* is given as a function of *u*. Consequently u should be calculated first. This is exactly what 20-sim will try to do while compiling the model into simulation code. I.e. the equations will be executed as:

```
u = cos(time);
z = sin(u);
```

Code Blocks

Equations in a **code block** are not reordered. A code block is a set of equations inside a statement. Suppose we have the following equations:

```
if condition then
    code block 1
    ...
    ...
else
    code block 2
    ...
    ...
end;
```

To prevent incorrect executions of the if-statement, the equations of the code blocks will not be separated. Inside a code-block, equations can are not rewritten into an new order of execution. E.g. the following equations:

```
if time > 10 then
    z = sin(u);
    a = z^2;
    u = cos(time);
end;
```

Will be **not be reordered** and therefore not correctly executed! To get correct code, enter code blocks in a correct order, e.g.:

```
if time > 10 then
      u = cos(time);
      z = sin(u);
      a = z^2;
end;
```

Prevent Order Changes

To make all equations a code block you can use the code section. E.g.

```
parameters
      real y = 8;
variables
      real x1, x2, x3;
code
      x1 = time;
      x2 = sin(time);
      x3 = y*x1;
```

Integration Steps

Some integration algorithms do more calculations before generating the next output value. These calculations are called minor steps, the output generation is called a major step. During a minor step, all model equations are executed. In most cases you will not notice this because only the results of the major step are shown in the simulator. There are however, exceptions. The next topic will discuss this in more detail.

5.3.6 Integration Steps

Some integration methods do more calculations before generating the next output value. These calculations are called **minor steps**, the output generation is called a **major step**.

integration method	minor steps
Euler	0
Backward Euler	variable
Adams-Bashford 2	0
Runge Kutta 2	1
Runge-Kutta 4	3
Runge Kutta Dormand Prince 8	variable
Runge-Kutta-Fehlberg	variable
Vode Adams	variable

Backward Differentiation Formula (BDF)	variable
Modified Backward Differentiation Formula (MBDF)	variable

Execution of Equations

During a minor step, all model equations are executed. In most cases you will not notice this because only the results of the major step are shown in the simulator. There is one exception. If you use equations like

```
variables
        real y;
initialequations
        y = 0;
equations
        y = y + 1;
```

you will find that the value of *y* depends on the integration method that is used. The reason is obvious once your realize that the equations are executed during minor steps. If you use a *Runge Kutta 2* integration method, there is one minor step during which *y* is increased and one major step during which *y* is increased again!

Major

To prevent an equation from being calculated during minor steps, you can us the predefined variable **major**. This variable will return the boolean *false* when calculations are performed for a minor step and will return *true* when calculations are performed during a major step. The *major* variable is used for example in the library model *Amplitude Sensor*:

```
parameters
        real initial = 0.0;
variables
        real prev, peak;
initialequations
        peak = initial;
        prev = 0;
        output = initial;
code
        if major then
            peak = max([abs(input), peak]);
            if (input > 0 and prev < 0) or (input < 0 and prev > 0) then
                output = peak;
                peak = 0;
            end;
            prev = input;
        end;
```

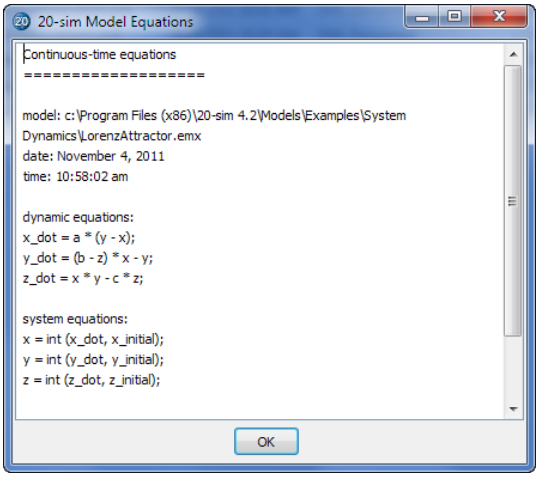
Note that the initial values are set in the *initialequations section*, because we do no want the variables to be set to zero at every integration step. Instead of an *equations section*, a *code section* is used to prevent 20-sim from rewriting the equations in a different order.

5.3.7 Show Equations

During processing a complete set of equations is generated of each model. To inspect these equations or copy them for use in other programs, you have to:

1. From the **Model menu** select the **Check Complete Model** command.
2. From the **Model menu** select the **Show Equations** command.

Now a window is popped up showing all the model equations.

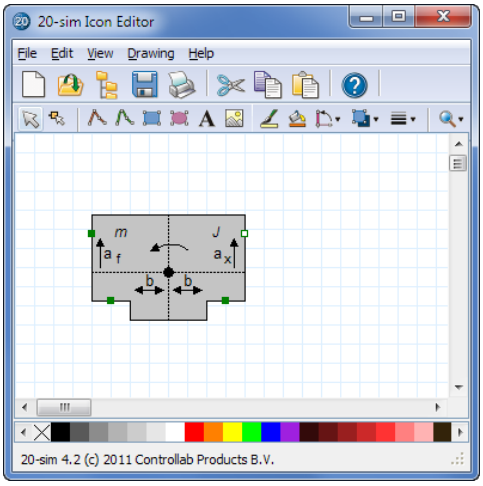


The model equations after compiling.

5.4 Icon Editor

5.4.1 Icon Editor

The *Icon Editor* is a vector oriented drawing editor that can be used to create custom made icons for any 20-sim model. You can also use the *Icon Editor* to change the appearance of existing models.



The 20-sim Icon Editor can be used to create custom made model icons.

Open

1. In the **Editor** select the **model** that you want to give a new appearance.
2. From the **Tools menu** select **Icon Editor**,
3. or from the **right mouse** menu select **Edit Icon**,
4. or at the bottom left of the Editor click the **Icon tab** and in the Icon tab **double click.**

Use

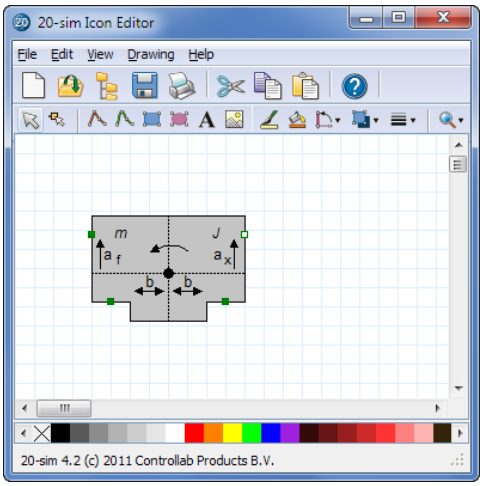
1. Use the **taskbar buttons** or the menu commands to insert objects and edit these objects.
2. If you are satisfied with the icon, insert the **terminals**.
3. From the Icon Editor **menu**, select **File** and **Update**.
4. **Close** the Icon Editor.

5.4.2 Terminals

Most submodels will contain ports (i.e an input signal, an output signal or a power port). If you are creating submodel icons, you want to indicate the connection points of these ports. These connection points are called *terminals*. In the *Icon Editor* you can define the position of the terminals.

1. In the taskbar click the **Terminal button** .
2. Click with the **mouse** pointer on the position where you want to **insert a terminal**.

If you have multiple inputs, outputs or ports, a menu will open to ask you which input, output or port should be selected.

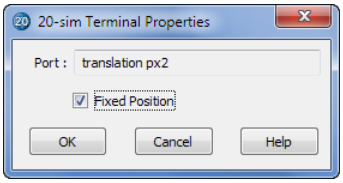


The inputs, outputs and ports are indicate by small rectangles.

The terminals are visible in the drawing as small rectangles.

3. You can **select a terminal** and from the **right mouse** menu click **Properties**.

This will open the *Terminal Properties* showing the name of the input, output or port and the option to fixate the position. With a fixed position, the connections will start or end exactly at the position of the terminal. Without a fixed position, the connections will point to the center of the icon and start or end at the border of the icon.



You can give a terminal a fixed position.

5.5 Global Relations Editor

5.5.1 Introduction

If you want to share parameter values or variable values between submodels, you can use signals from one submodel to the others. This is not always desired and it will increase the effort that you have to make in connecting submodels.

Using globals is an alternative for sharing values. By defining a parameter or variable global, we can instruct 20-sim that a parameter or variable value is shared between submodels.

Using Globals

If you want to share parameters or variables between submodels, you have to:

1. Add the **keyword global** to definition of the parameter or variable.
2. Define the scope of the globals, i.e. which models share the same parameters and variables?

5.5.2 Global Parameters and Variables

Global parameters and variables are defined using the keyword ***global***. In equation models, this keyword is added after the definition of the data type in the Equation Editor:

```
parameters
      real global par1 = 100 {Hz};
      real global par2 ;
variables
      real global var1;
      real global var2;
..
..
```

Parameters can only be assigned a value once. The same goes for variables. Only one submodel must defined their value using an equation. If parameters or variables are assigned more than once, 20-sim will generate an error.

Example

The best way to describe the use of global parameters and variables is by an example model. You can find this model in the 20-sim demonstration library (*Demo\Tips And Tricks\Global Parameters and Variables.emx*). The model exists of three submodels. In the first submodel two global parameters (*f* and *A*) are assigned and one variable (*sinewave*). The variable is assigned a value in the equations section and connected with the output (*out1*).

```
submodel1

parameters
    real global f = 100 {Hz};
    real global A = 1 {none};
variables
    real global sinewave;
equations
    sinewave = A*sin(f*2*pi*time);
    out1 = sinewave;
```

In the second submodel, the two global parameters are use to connect them with two outputs (*out1* and *out2).*

```
submodel 2

parameters
    real global f {Hz};
    real global A {none};
variables
    real global sinewave;
equations
    out1 = f;
    out2 = A;
```

In the third submodel, the global variable is connected with an output (*out1).*

```
submodel 3

variables
    real global sinewave;
equations
    out1 = sinewave;
```

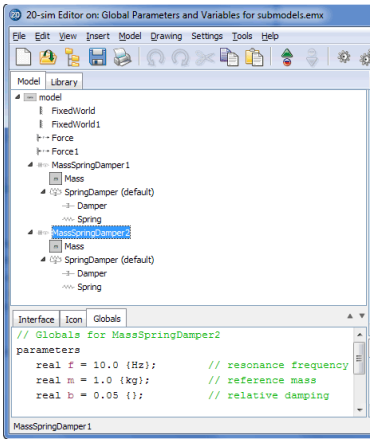
To prevent conflicts global parameters may be assigned a default value only once. The same goes for global variables: they may be assigned a value in the equation section only once.

5.5.3 Scope

Global parameters and variables are defined using the *keyword* ***global***. In equation models, this keyword is added after the definition of the data type in the Equation Editor. In 20-sim we can define which submodels share the same parameter or variable. This is called the ***scope*** of this parameter or variable.

Part of the Model

You can set the scope by using the Global Relations Editor. In this editor you can enter parameters or variables. All global parameters or variables **that are defined in submodels lower in the hierarchy**, share the same value.



Setting the scope of global parameters with the Global Relations Editor.

The image above shows an example. In the model hierarchy we see the submodel *MassSpringDamper2* selected. The Globals Relations Editor, shows three parameters: *f*, *m* and *b*. If these parameters exists in the submodels lower in the hierarchy (*Mass*, *SpringDamper*, *Damper*, *Spring*), and use the keyword global, they share the same value. Submodels higher in the hierarchy, can never share these parameter values.

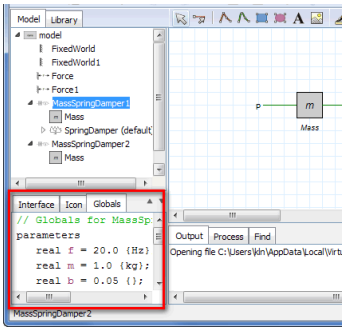
Whole Model

If no parameter or variable is defined using the Global Relations Editor, the scope is the whole model. I.e. global parameters and variables are shared across the whole model.

5.5.4 Global Relations Editor

The *Global Relations Editor* is used to restrict the scope of global parameters and variables. You can find the *Global Relations Editor* at the lower left of the *Editor* by selecting the Globals tab.

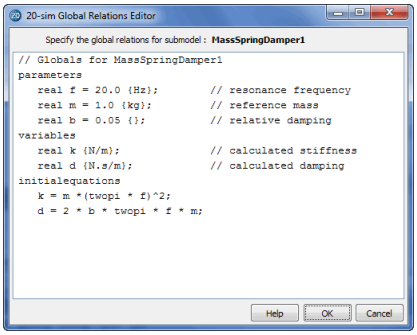
1. In the **Editor**, go to the **Model Hierarchy** at the left and **select** the **submodel**, you want to add global parameters or variables for.
2. Select the **Globals tab** at the lower left of the Editor.



Select the Globals tab, put your mouse pointer on top and click Edit Global Relations.

3. Put your **mouse pointer** in the middle of the tab. From the **right mouse menu** select **Edit Global Relations**.

This will open the Global Relations Editor. You can use this editor to enter parameters, variables and equations.



Use the Global Relations Editor to enter parameters, variables and equations.

You can enter parameters, variables and equations here just like in the Equations Editor. The parameters and variables will only be useful if they are used in submodels lower in the hierarchy as globals.

Example

The best way to describe the use of the Global Relations Editor is by an example model. You can find this model in the 20-sim demonstration library (*Demo\Tips And Tricks \Global Parameters and Variables for submodels.emx*). This model shows two mass-spring-dampers with shared parameters and variables.

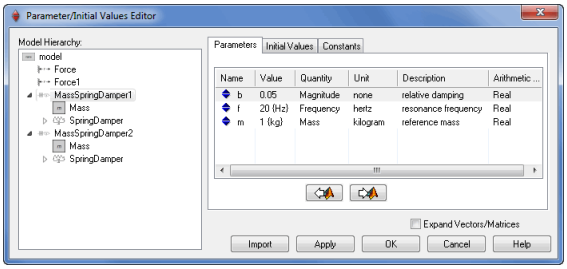
1. In the **Editor**, go to the **Model Hierarchy** at the left and **select** the submodel *MassSpringDamper1*.

2. Select the **Globals tab** at the lower left of the Editor.

3. Put your **mouse pointer** in the middle of the tab. From the **right mouse menu** select **Edit Global Relations**.

Now you will see the Global Relations Editor, as shown in the previous figure. Three parameters are defined: *f*, *m* and *b*. The parameter *m* is use in the submodel *Mass* lower in the hierarchy. The variable *k* is used in the submodel *Spring* lower in the hierarchy. The parameter *d* is use in the submodel *Damper* lower in the hierarchy.

4. Now open the *Parameters Editor*.



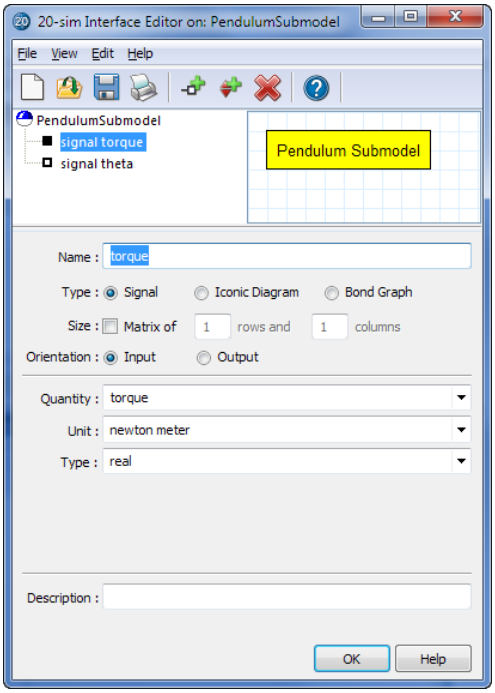
Parameters that are shared can be set at the top model.

You will see that it contains the parameters *f*, *m* and *b* at the level of the model MassSpringDamper1. This allows us to duplicate this model and use a second mass-spring-damper model with different parameters.

5.6 Interface Editor

5.6.1 Interface Editor

The *Interface Editor* can be used to define the interface, i.e. the input signals, output signals and power ports of a submodel.



The 20-sim Interface Editor can be used to define the model interface.

Open

1. In the **Editor** select the model of which you want to change the interface.
2. From the **Tools menu** select **Interface Editor**,
3. or from the **right mouse** menu select **Edit Interface**,
4. or at the bottom right of the Editor click the **Interface tab** and in the Interface tab **double click.**

Use

1. Use the **Edit menu** to add ports to the list.

2. Select a port from the list and change its **properties** (make it an input signal, output signals and power ports etc.)
3. From the **Interface Editor menu**, select **File** and **Update**.
4. **Close** the Interface Editor.

5.6.2 Port Properties

From the *Edit* menu select *Add Port*. A new port will now be added to the list. If you select the port in the list you can change its properties

- *Name*: Enter the name of the port.
- *Type*: Select the port-type: Signal, Iconic Diagram or Bond Graph.

Signal

- *Size*: If the signal is multi-dimensional, enter the number of rows and columns.
- *Orientation*: Choose input signal or output signal.
- Quantity: If the signal represents a physical quantity, you can choose the quantity from a predefined list.
- *Unit*: If the signal represents a physical quantity, you can choose the corresponding unit from a predefined list.
- *Type:* The default type of a signal is a real number. Choose out of *real*, *integer* or *boolean.*

Iconic Diagram

- *Size*: If the port is multi-dimensional, enter the number of rows and columns.
- *Orientation*: Choose an input or output orientation for the port.
- *Domain*: Choose the physical domain of the port. The most general domain is power. Depending on your choice you will see the two variables (Across and Through) associated with the port.
- *Causality*: Select the causality of the port.
- Separate High/Low Terminals: You may choose this option for models with a velocity difference, voltage difference etc.
- Any Number of Terminals: You may choose this option for models with a variable number of connections.

Bond Graph Port

- *Size*: If the port is multi-dimensional, enter the number of rows and columns.
- *Orientation*: Choose an input or output orientation for the port.
- *Domain*: Choose the physical domain of the port. The most general domain is power. Depending on your choice you will see the two variables (Effort and Flow) associated with the port.
- *Causality*: Select the causality of the port.

5.6.3 Orientation

In the *Interface Editor* you can set the orientation of a ports. In the *Editor* this orientation can be made visible by setting the **Orientation Info** command from the **View** menu. Depending on the type of port the orientation has a different meaning.

- *Signals*: For signals the orientation indicates the direction of the associated variable : input or output.
- *Bond Graphs*: The orientation of a bond graph port is the direction of the half arrow at the end of a bond. It indicates the direction of the positive power flow.
- *Iconic Diagrams*: The orientation of an iconic diagram port also indicates the direction of the positive power flow.

If you select the top of the list in the Interface Editor, you can choose the *Port Relations tab* to set multiple port relations:

- *Equal* / *Not equal*: The equal and not equal restrictions can be used to specify that a port p1 must have the same or opposite orientation as port p2.

5.6.4 Causality

A powerport is always characterized by two variables. One of these variables should be an input and one of these variables should be an output. The choice of input and output variable is called causality. In the *Interface Editor* you can set the causality of the ports of a submodel. The following causal forms can be chosen:

1. Fixed out: One of the two variables is always an output variable.
2. Preferred out: One of the two variables is preferably an output variable.
3. Likes: Same as preferred out but with lower priority: one of the two variables is preferably an output variable
4. Indifferent: A powerport with an indifferent causality restriction can have both variables as output.

During processing, the assignment of the causal forms goes with the rank in the list. First all elements with a *fixed out* assignment are set, then the elements with a preferred out assignment, then with a likes assignment and finally with an indifferent assignment.

If you select the top of the list in the *Interface Editor*, you can choose the *Port Relations tab* to set multiple port restrictions:

- *Equal* / *Not equal*: The equal and not equal restrictions can be used to specify that a powerport p1 must have the same or opposite causality as powerport p2. This is denoted as p1 equal p2 and p1 notequal p2 respectively.
- *One_out*: The restriction One_out should only be used for bond graph ports. It is used to indicate that at a one or zero junction, only one bond may have an effort out causality.
- *One_in*: The constraint restriction One_in should only be used for bond graph ports. It is used to indicate that at a one or zero junction, only one bond may have an effort in causality.

In the *Editor* this causality can be made visible by setting the **Causality Info** command from the **View** menu.

5.6.5 Ports with more than one Terminal

By default, an iconic diagram port is a port where power can be exchanged between a component and its environment in terms of an *across* variable and a *through* variable. Such a port is represented by one terminal (connection point). However, there are two special cases where it is desirable to define an iconic diagram port that has more than one terminal.

Separate High / Low Terminals

Consider, as an example, an mechanical spring. The power that flows into such a component is uniquely determined by the velocity difference between the two terminals of the component and the common force that acts upon both ends. Because of this one could say that there is one port, whose power is determined by one *across* value (the velocity difference) and one *through* value (the common force), but is represented by two terminals. To support this, 20-sim allows you to define a special type of iconic diagram port by indicating that is has *Separate High / Low Terminals*. The two terminals of the connection are named *high* and *low*. If the port is named *p*, the formal equations are:

fixed in orientation:

p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v

fixed out orientation:

p.F = p_high.F = p_low.F
p.v = - p_high.v + p_low.v

In 20-sim these equations are automatically derived.

Any Number of Terminals

Consider, as an example, a mass. A characteristic of this component is that you can connect many springs and dampers to it. Implicitly, one expects that the net force (i.e., the summation of the forces applied by the connected components) will be applied to the mass, and that it will have a single velocity. To support this, 20-sim allows you to define a special type of iconic diagram port by indicating the kind to be *Any Number of Terminals*. The terminals of the connection are named *1*, *2*, *3* etc.. If the port is named p, the formal equations are:

fixed in orientation:

p.v = p1.v = p2.v = p3.v = ...
*p.F = sign1*p1.F + sign1*p2.F + sign1*p3.F + ...*

fixed out orientation:

p.v = p1.v = p2.v = p3.v = ...
*p.F = -sign1*p1.F - sign1*p2.F - sign1*p3.F - ....*

with

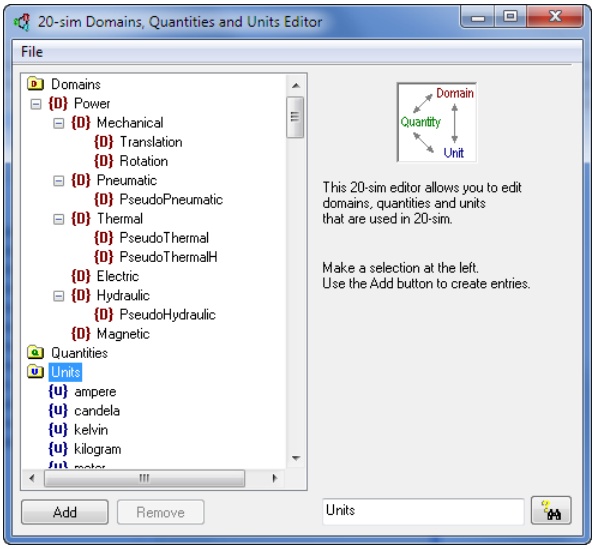
sign = 1 when p1 has a fixed in orientation etc.
sign = -1 when p2 has a fixed out orientation etc.

In 20-sim these equations are automatically derived.

5.7 Domains, Quantities and Units

5.7.1 Domains, Quantities and Units Editor

At various modeling levels you can use information on physical domains, quantities and units. This information is stored in the file *QuantitiesAndUnits.ini* in the 20-sim bin directory. To edit this information you can use the *Domains, Quantities and Units Editor*. When opened, the editor automatically reads the file *QuantitiesAndUnits.ini*. When closed the editor asks you where to store the file. Note that any changes made become effective after 20-sim has been restarted.



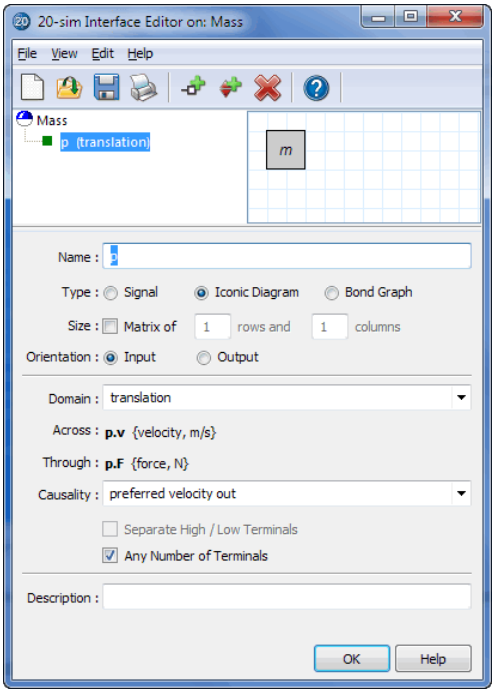
The Domains, Quantities and Units Editor.

You can start the *Domains, Quantities and Units Editor* by selecting the **Units Editor** command from the **Tools** menu. In the editor a tree-like structure is shown indicating the available Domains, Quantities and Units.

1. You can select every items in the tree for editing.
2. Use the **Add** or **Delete** button to add new items or delete existing item.
3. Use the **Find button** to find items in the tree.

5.7.2 Domains

In the Interface Editor every powertport (Iconic Diagram or Bond Graph) can be assigned a physical domain. 20-sim will use this information to prevent that ports of different physical domains may be connected. More popular: in this way you can never connect a translation model to an electric component.



Use the Interface Editor to assign a physical domain to a port.

In the *Interface Editor* you can select the physical domain that you want to assign to a port. The editor will show the corresponding port variables and the notation that should be used in the model equations.

The tables below show the available domains and their respective port variables.

power domain	**across**	**through**
Power	across	through
Mechanical	velocity	force
Translation	velocity	force
Rotation	angular velocity	torque
Pneumatic	pressure	air flow

Thermal	temperature	entropy flow
Electric	voltage	current
Hydraulic	pressure	volume flow
Magnetic	magnetomotoric force	magnetic flux

For the power domains the variables multiply to **power**. There are also some domains defined where the variables do not multiply to power. These are called **pseudo domains**. The following pseudo domains are defined.

pseudo domain	**across**	**through**
Pseudo Pneumatic	pressure	mass flow
PseudoThermal	temperature	heat flow
PseudoThermalH	temperature	enthalpy flow
PseudoHydraulic	pressure	mass flow

You can enter new (pseudo) domains in the Domains, Quantities and Units Editor. If the port is defined as a bond graph port, effort and flow are used to indicate the domain variables. The corresponding tables are shown below.

power domain	**effort**	**flow**
Power	effort	flow
Mechanical	force	velocity
Translation	force	velocity
Rotation	torque	am
Pneumatic	pressure	angular velocity
Thermal	temperature	air flow
Electric	voltage	entropy flow
Hydraulic	pressure	current
Magnetic	magnetomotoric force	volume flow
		magnetic flux

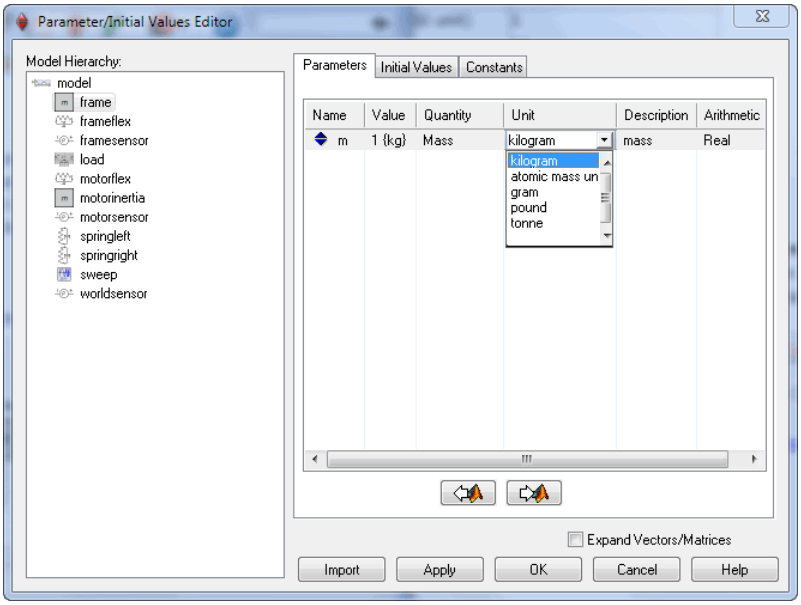
pseudo domain	**effort**	**flow**
Pseudo Pneumatic	pressure	mass flow
PseudoThermal	temperature	heat flow
PseudoThermalH	temperature	enthalpy flow
PseudoHydraulic	pressure	mass flow

5.7.3 Quantities and Units

In 20-sim, to every constant, parameter and variable a quantity may be assigned. Well known quantities are for example position, velocity, time and energy. 20-sim will use these quantities to check the validity of (basic) equations. The value of a quantity can be expressed in a specific unit. The quantity time can for example be expressed in the unit second or in the unit minute:

quantity	**unit**	**multiplication to SI unit**
Time	second (SI unit)	1
	minute	60
	hour	3600
	day	31556926.08
Position	meter (SI unit)	1
	foot	0.3048
	inch	0.0254
	yard	0.9144

In 20-sim a quantity can be expressed in every desired unit. To prevent **conversion** problems, internal calculations are always performed in standard SI-units. Therefore non-SI units are stored with a **multiplication** factor. When a variable has a unit, the unit can also have common engineering **prefixes**. For example, a parameter m {kg} with a value of 0.1 can be selected in the *Parameters / Initial Values Editor*. If you click on the unit, you can select other units to enter the mass.



You can use the units in the Parameters / Initial Values Editor to select a desired unit.

You can enter new quantities and units in the Domains, Quantities and Units Editor.

5.7.4 Editing Domains

You can edit existing domains or add new domains in the *Domains, Quantities and Units Editor*. In the Domains section you can add new domains or edit existing domains. Each domain has several edit fields:

- *Domain name*: If you click on a selected domain in the tree, you can edit the name.
- *Description*: Enter a description of the domain.
- *Quantities*: You can enter quantities of these variables in the quantities section.
- *Mechanical Domain*: Select this button if your domain is part of the mechanical domain.
- *Pseudo Domain*: Select this button if your domain is a pseudo domain.
- *Color*: Double-click the color box to change the color of a domain.

In some domains the integrated effort/flow variables and differentiated effort/flow variables are of a specific quantity. You can enter these quantities in the according edit fields. For some domains these values have no meaning and you can leave the edit fields blank.

5.7.5 Editing Quantities

You can edit existing quantities or add new quantities in the *Domains, Quantities and Units Editor*. In the Quantities section you can add new quantities or edit existing quantities. Each quantity has several edit fields:

- *Quantity name*: If you click on a selected quantity in the tree, you can edit the name.
- *Description*: Enter a description of the quantity.
- *Variable name*: This name is used for display purposes. For example if one of the variables of a port p has the quantity length it will be denoted as p.x.
- *SI Symbol*: Enter the official SI-symbol here. Enter combinations as nominator/ denominator (e.g. Electric Resistance: kg.m2/A2.s3)
- *Units*: Every quantity can be expressed in several unique units (i.e. a unit can be assigned to one quantity only). You can enter new units here or edit existing units that belong to the selected quantity.
- *Unit Symbol*: For display purposes every unit has a specific symbol. It is shown here in a read-only field.

A quantity can have an alias name. Enter this alias as a subentry of the main quantity. This alias name refers to the main quantity and therefore to the same associated units.

5.7.6 Editing Units

You can edit existing units or add new units in the *Domains, Quantities and Units Editor*. In the Units section you can add new units or edit existing units. Each unit has several edit fields:

- *Unit name*: If you click on a selected unit in the tree, you can edit the name.
- *Description*: Enter a description of the unit.
- *Quantity*: Choose the quantity to which the unit should be associated. A unit can be associated to one quantity only.
- *Symbol*: For display purposes every unit has a specific symbol.
- *Multiplications*: A quantity can be expressed in several units. To prevent conversion problems, internal calculations are always performed in standard SI-units. Therefore non-SI units are stored with a multiplication factor. You can enter the multiplication factor here.
- *Allow Symbol Prefixes*: Some units can be expressed with symbol prefixes. Select the check box to allow the use of prefixes. 20-sim supports the following prefixes:

prefix	**symbol**	**value**
femto	f	10^{-15}
pico	p	10^{-12}
nano	n	10^{-9}
micro	u	10^{-6}

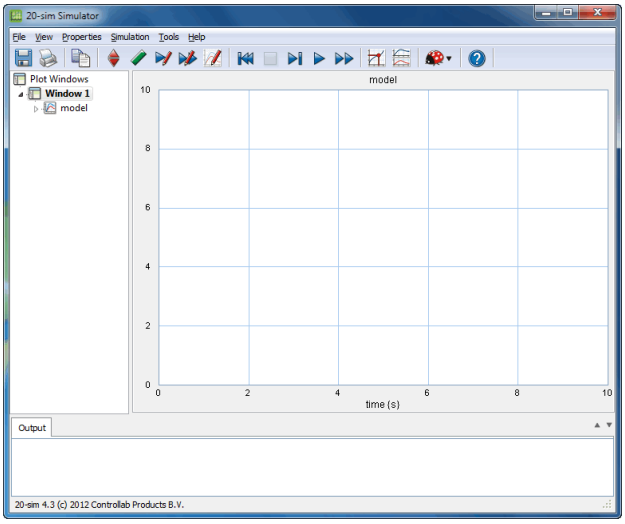
milli	m	10^{-3}
kilo	k	10^{3}
mega	M	10^{6}
giga	G	10^{9}
tera	T	10^{12}

6 Simulator

6.1 Introduction

6.1.1 Simulator

20-sim consists of two main windows and many tools. The first window is the *Editor* and the second is the *Simulator*. The *Editor* is used to enter and edit models. The *Simulator* is used to simulate and analyze the models. The Simulator is opened in the *Editor* by clicking the *Start Simulator* button from the *Model* menu.



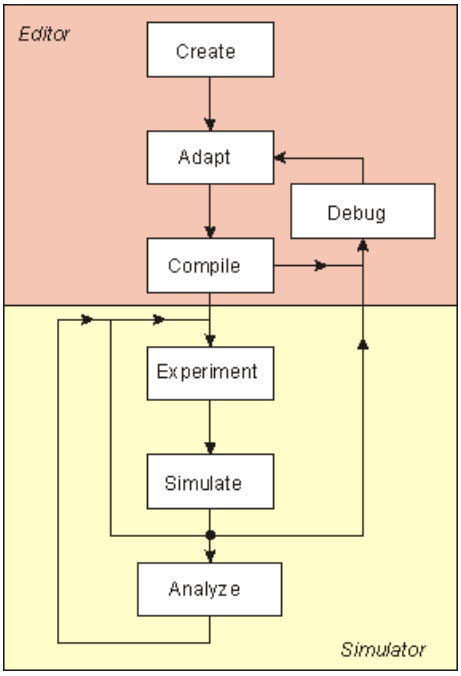
The Simulator is used to carry out simulations, show the results and analyze them.

The *Simulator* consists of three parts:

- *Simulator Tree*: This is the tree at the left. Here plots and plot windows can be added.
- *Simulation Plot*: This is the graph in the middle. Here the simulation results are shown.
- *Output tab:* The *Output tab* shows warnings and messages.

Simulation

A modeling and simulation session in 20-sim can be visualized by the diagram shown below.



Modeling and Simulation in 20-sim.

Compiling/Debugging

After the model has been successfully entered, it must be checked and compiled to generate simulation code. This is done in 20-sim automatically when you open the *Simulator* or when you check a model. In some cases the model contains errors which have to be solved by adapting the model. Errors in 20-sim are presented in the *Process tab* at the bottom of the *Editor.*

Settings

After a successful compilation of a model that is created in the Editor you can open the Simulator. In the Simulator you have to enter specific settings:

- Parameters / Initial Values.
- Run Properties.
- Plot Properties.

Running

After the settings have been entered you can run the simulation.

Plotting

Simulation results can be shown in 20-sim as:

- Numerical Plots:
  - Time Domain (default)
  - Frequency Domain (FFT Analysis & Linearization)

- Animation in a Graphical Model
- 3D Animation

Debugging

If you experience problems during simulation you have to resolve them in a debugging session.

Analysis

After a successful simulation you can analyze your model using various tools:

- Parameter Sweeps
- Curve Fitting
- Optimization
- Sensitivity Analysis
- Monte Carlo Analysis
- Variation Analysis
- FFT Analysis
- Linearization

Message Log

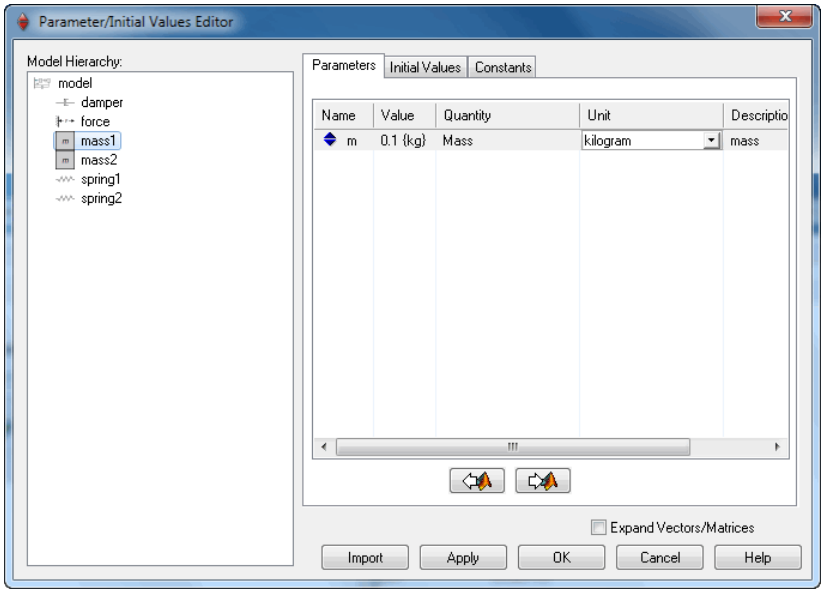
The bottom of the Simulator shows a tab named *Output*. This tab contains the *Message Log* which shows all the messages, warnings and errors, that are generated during a simulation.

6.2 Running a Simulation

6.2.1 Parameters and Initial Values

The *Parameters/Initial Values Editor* is used to edit the constants, parameters and initial values of a model. Parameters and initial values can be changed during simulation, constants cannot be changed. To open the Parameters/Initial Values Editor, you have to:

1. From the **Properties** menu select the **Parameters** command.



The Parameters/Initial Values Editor.

The parameter items (Name, Value, Quantity, Unit, Description, Kind, Arithmetic Type, Value) of the *parameters* and *initial values* are listed in right part of the window. The list of parameters and initial values that is displayed depends on the selected submodel in the left part of this window.

Items

- *Model Hierarchy*: The left part displays the complete model hierarchy. At each level of the hierarchy, submodels can be selected. The right part of the window then shows the parameters and initial values of that submodel. To see all parameters and initial values, select the top level of the hierarchy, i.e. the main model listed on top.
- *Parameters and Initial Values*: The order of the parameters or initial values in the list depends on the order in which the submodels were originally entered in the Editor. Click on the name box to make the order to alphabetical. Drag the name box to change the width of the parameters list.

- *Value*: You can change the default value of a parameter or initial value in the value list. Select the parameter or initial value (mouse) and enter the desired value. Use the Enter key to quickly run through a list of parameters.
- *Unit*: If available, select the desired Unit from the drop down list.
- *From Matlab / To Matlab*: Use the buttons to import or exports parameter values from or to Matlab.
- *Import*: Use this button to import the parameter values from another model.

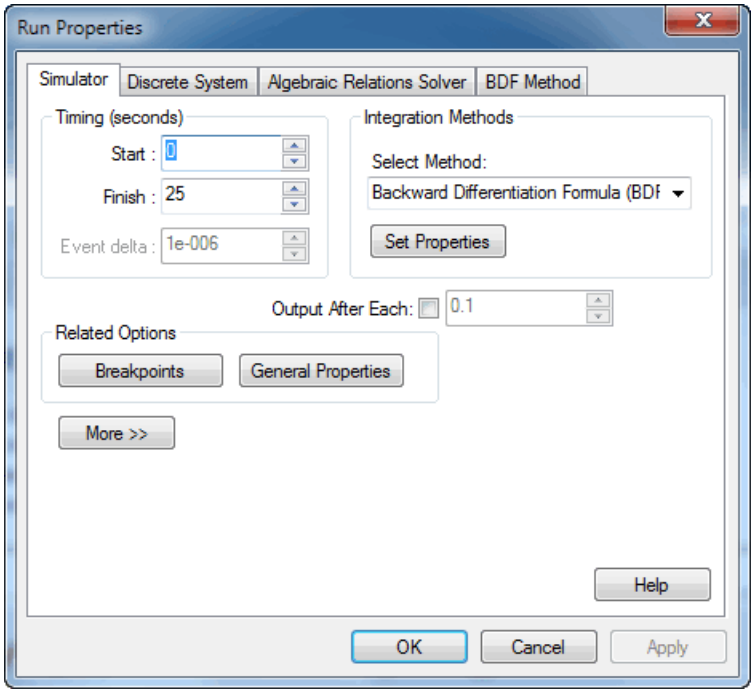
Note

- All equations in 20-sim are calculated using standard SI-units. You can however enter parameters, using whatever Unit you like. In the *Parameters/Initial Values Editor*, you can change the unit using the little drop down list next to the *Value* box.
- The description of a parameter can be specified in the parameters part of the equation description of a model.
- To see a global parameter, select the top level of the hierarchy, i.e. the main model listed on top.
- Hidden parameters are not visible in the Variables Chooser.

6.2.2 Run Properties

The *Run Properties Editor* can be used to choose the *integration method* used for simulation. An integration method is the algorithm that calculates the model equations during each simulation and generates the output values for the simulation plots. The default integration method is the Backward Differentiation Formula (BDF). To open the Run Properties Editor, you have to:

1. From the **Properties** menu select the **Run** command.



The Run Properties Editor.

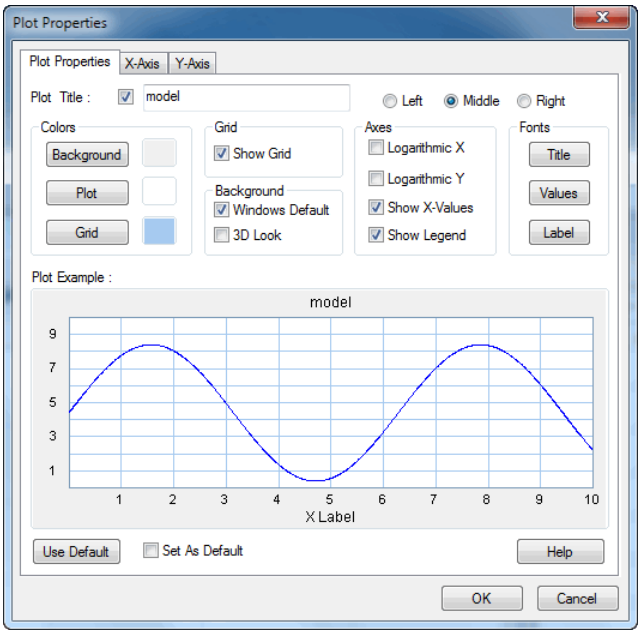
Initially, the fields in the Run Properties Editor contain default values if no experiment was loaded. Two or more tabs can be selected:

- *Simulator*: This tab shows the general simulation settings.
- *Discrete Systems*: This tab is only visible when discrete systems are part of your model. It contains the settings for the sample frequency.
- *Algebraic Relations Solver*: This tab is only visible when your model contains algebraic loops that could not be solved during processing.
- *Integration Method*: The last tab contains the specific settings for the selected integration method.

6.2.3 Plot Properties

The Plot Properties Editor can be used to enter the plot specifications. To open the Plot Properties Editor, you have to:

1. From the **Properties** menu select the **Plot** command.



The Plot Properties Editor.

The editor shows three tabs which can be used to set the properties of the plot and the x and y variables.

Plot Properties Tab

The *Plot Properties Tab* shows the general settings that accompany a plot. A preview is shown in the lower part of the tab.

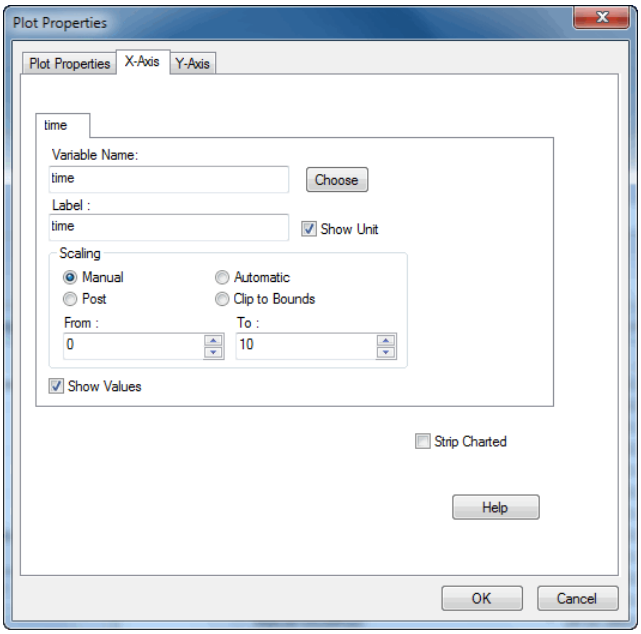
- *Plot Title*: Select this option to add a title to the plot. In the text box you can type the desired title. Use the radio buttons to display the title to the left, in the middle or to the right of the plot.
- *Colors*: Click the Background, Plot and Grid buttons to select the background, plot and grid colors.
- *Grid*: Select this option to show the grid.
- *Background*
  - *Windows Default*: Select this option to get the default Windows background.
  - *3D Look*: Select this option to get a plot with 3D Look (sunken edges).
- *Axes*:

- Select the options *Logarithmic X* and *Logarithmic Y* to show logarithmic plots.
- *Show X-Values*: Use this option to show or hide the X-axis values.
- *Show Legend*: Use this option to show to show a legend.

- *Fonts*: Click the *Title*, *Values* and *Labels* buttons to select the font options for the title, values and labels.

X-Axis Tab

The *X-Axis Tab* can be used to specify the variable plotted along the X-axis.



The X-Axis tab of the Plot Properties Editor.

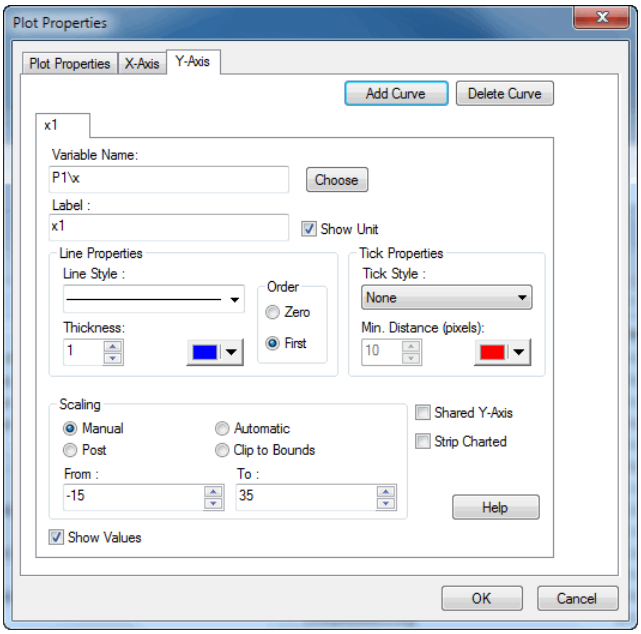
- *Variable Name*: Either time or one of the dynamic variables of the model can be chosen as the X-axis variable. Click the button on the right (Choose) to open the Variable Chooser and select the desired variable.
- *Label*: Enter the label that should be printed below the X-axis.
- *Show Unit*: Select this option to display the unit of the chosen variable.
- *Scaling*
  - *Manual*: Fix the scale of the X-axis by selecting a minimum and maximum value in the From and To boxes.
  - *Post*: During the simulation run, the scale for the variable along the axis is fixed. When the run is finished or has been interrupted by the user, the minimal and maximal scale values are automatically updated and the plot is redrawn to make sure all calculated points are displayed.

- *Automatic*: When a variable reaches the end of the scale, during simulation, the the minimal and maximal scale values are automatically updated and the plot is redrawn to make sure all calculated points are displayed.
- *Clip to Bounds*: When a variable reaches the end of the scale, during simulation, the the minimal and maximal scale values are automatically updated and the plot is redrawn to make sure all calculated points are displayed. When the run is finished or has been interrupted by the user, the minimal and maximal scale values are automatically updated to show the plot at maximum scope.

- *Show Values*: Use this option to show or hide the X-axis values.

Y-Axis Tab

The *Y-Axis Tab* is used to specify the curves that should be shown in the plot. For each selected curve, a tab is shown, displaying the settings of that curve.



The Y-Axis tab of the Plot Properties Editor.

Curves

- *Add Curve*: Add a new tab to specify a new curve to be displayed in the plot.
- *Delete Curve*: Delete the selected tab. As a result, the curve specified in the tab will not be displayed in the plot, and the settings are lost.
- *Variable Name*: Use this box to connect a variable to a curve. One of the dynamic variables of the model can be selected. Click the button on the right (Choose) to open the Variable Chooser and select the desired variable.

- *Label*: Enter the name of the curve. This label is also printed next to the the Y-axis.
- *Show Unit*: Select this option to display the unit of the chosen variable.
- *Line Properties*
  - *Line Style*: Select the desired line style of the curve.
  - *Thickness*: Select the desired line thickness of the curve.
  - *Color*: Select the desired line color of the curve.
  - *Order*: In the plot a line is drawn from plot point to plot point. You can choose the line to be of two orders of interpolation:
    - Zero: the line consists only of horizontal and vertical parts.
    - First: the line consists of straight parts.
- *Tick Style Properties*
  - *Tick Style*: Each calculated point of a curve, can be displayed by a special marker. Select here the desired marker.
  - *Min. Distance (pixels)*: To prevent markers from being drawn on top of each other, you can set the minimum distance between markers. Points that are to close to a previous marker are not drawn with a marker.
  - *Color*: Select the marker.
- *Scaling:* Scaling can be selected for each curve individually or combined for all curves.
  - *Shared Y Axes*: Select this option to use one Y-axis scale for all curves.
  - *Strip Charted*: Select this option to display each curve in a separate plot.
  - *Manual*: Fix the scale of the variable by selecting a minimum and maximum value in the From and To boxes.
  - *Post*: During the simulation run, the scale for the variable along the axis is fixed. When the run is finished or has been interrupted by the user, the minimal and maximal scale values are automatically updated and the plot is redrawn to make sure all calculated points are displayed.
  - *Automatic*: When a variable reaches the end of the scale, during simulation, the the minimal and maximal scale values are automatically updated and the plot is redrawn to make sure all calculated points are displayed.
  - *Clip to Bounds*: When a variable reaches the end of the scale, during simulation, the the minimal and maximal scale values are automatically updated and the plot is redrawn to make sure all calculated points are displayed. When the run is finished or has been interrupted by the user, the minimal and maximal scale values are automatically updated to show the plot at maximum scope.
  - *Show Values*: Select or de-select this option to show or hide the name and scale of a curve.

- *Separate X-Axis*: If you want another x-axis than specified in the X-axis tab, click the Choose button to select the desired variable.

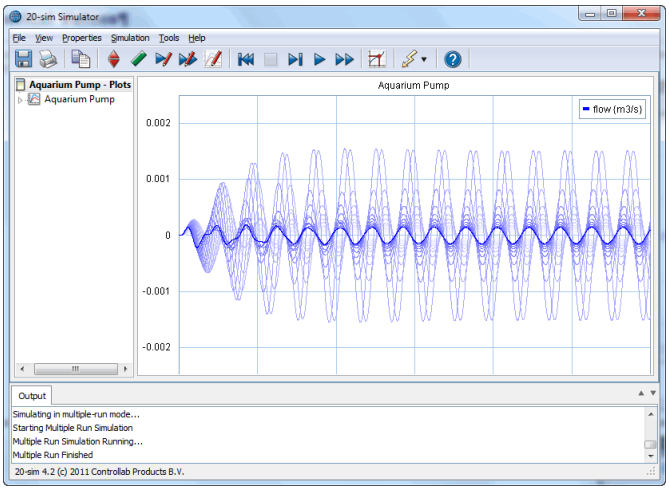
6.2.4 Running a Simulation

After specification of the Parameters/Initial Values, the Run Properties and Plot Properties you can start a simulation run by choosing the appropriate command from the *Simulation* menu.

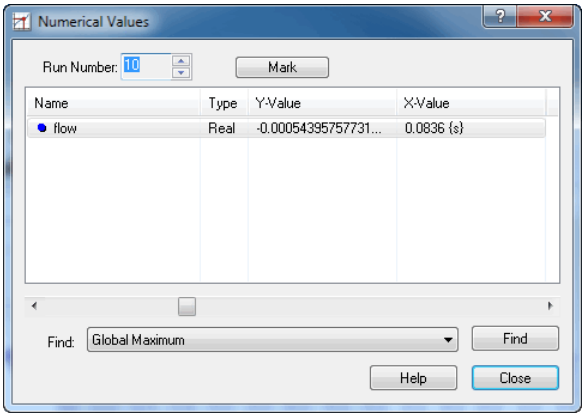
- *Run*: Select the Run command to start or continue a simulation run. Once a run is started, the Stop command becomes active. The run can be interrupted by selecting this command.
- *Stop*: Use the Stop command to interrupt a simulation. Once a simulation has stopped it can be continued by selecting the Start Simulation command.
- *Clear*
  - *All Runs*: Select the Clear All Runs command to clear all runs that were performed, from the graphical screen and from the computer memory.
  - *Last Run*: Select the Clear Last Run command to clear the run that was performed last, from the graphical screen and from memory.
  - *Previous Runs*: Select the Clear Previous Runs command to clear all runs, but the last, from the graphical screen and from memory.
- *One Step*: Use the One Step command to calculate one new simulation point. Using this command repeatedly, allows you to run a simulation step by step.
- *Brute Force*: Using the Brute Force command, curves are displayed after complete calculation of a simulation run. This saves time (re)drawing curves during calculations and therefore considerably speeds up simulation.
- *Multiple Run*: Select the Multiple Run command to start multiple runs needed for the *Time Domain Toolbox*. Runs can be interrupted by selecting the Stop command.
- *Replay*
  - *3D Animation*: Any 3D Animation that was performed during a simulation run, can be quickly replayed using this option.
  - *Real Time 3D Animation*: Any 3D animation that was performed during a simulation run, can be replayed in real time (i.e. frames are skipped if necessary) using this option.
  - *Read Datafile*: Use the Read Data File command to read previously stored simulation runs from file.
- *Copy States*: With the Copy States command the current independent state variables can be imported as new initial values. Current state variables are the state variables at the end of a simulation run or at the point where a simulation run was interrupted.

6.2.5 Numerical Values

In every simulation plot you can inspect the numerical values by selecting the *Numerical Values* command of the *View* menu. The *Numerical Values* window, can also be popped up by double clicking the left mouse button, while pointing the mouse at a curve in the plot.



The Numerical Values window will pop up and display the numerical values of all plotted variables as a function of time. The last variable shown is always the X-axis variable (usually time).



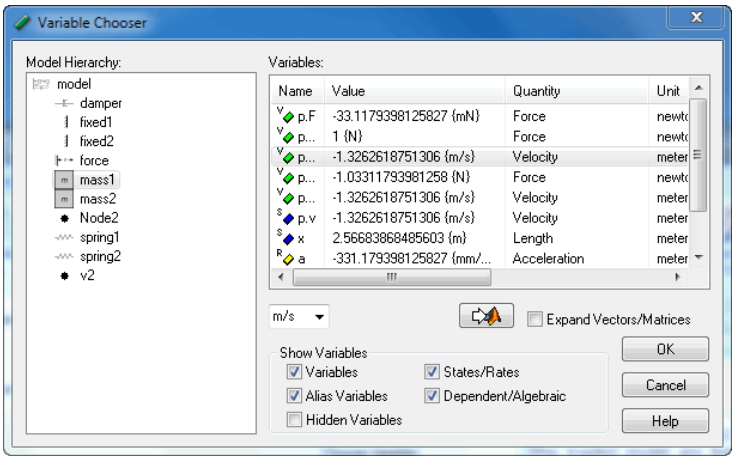
The Numerical Values window.

- *Run Number*: The run number indicates the run of which the numerical values are shown. The highest number indicates the last run simulated. Lower numbers are used for previous runs. To select another run number, type in the number of interest or point in the plot at the curve of interest. In the simulation plot, previous runs are shown with fading colors.

- *List*: The list shows all variables that are shown in the plot and their corresponding X-values and Y-values. The variable that is currently selected is pointed out by an arrow in the plot.
- *Scroll-bar*: Use the scroll-bar to change the selected x-axis variable value (usually time).
- *Find*: With the Find button, the global maximum etc. of the selected variable can be found. The following functions can be selected:
  - *Global Maximum*
  - *Global Minimum*
  - *Next Local Maximum*
  - *Next Local Minimum*
  - *Previous Local Maximum*
  - *Previous Local Minimum*
- *Mark*: Click the *Mark* button to store he set of numerical values. If you drag the scroll-bar to find a new set of values, the relative changes will be displayed as well as a first order derivative.

6.2.6 Variable Chooser

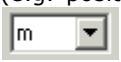
The Variable Chooser can be used to select and inspect any dynamic variable of the loaded model. The names, kinds, types, values, quantities, units and descriptions of the variables of the loaded model are listed in right part of the window. The list of dynamic variables displayed, depends on the selected (sub)model in the left part of this window. The left part displays the complete model hierarchy.



The Variable Chooser shows all variables of a model.

Items

- *Model Hierarchy*: The model hierarchy shows several keys:

  - *Model*: At each level of the hierarchy, submodels can be selected and inspected. The right part of the window then shows the variables that are part of that submodel. To see all variables including time, select the top level of the hierarchy, i.e. the main model listed on top. Click on one of the variable items (Name, Value, Quantity, Unit, Description, Kind, Arithmetic Type, Value) to order the variables alphabetically.
  - *Input Probes/Output Probes*: In the Frequency Response dialog pairs of input variables and output variables can be selected for linearization. If input variables have been selected they are shown under the input probes key. If output variables have been selected they are shown under the output probes key.
  - *Favorites*: variables that are used a lot can be added to the favorites key for easy use.
- *Variables*: Select the desired variable from the list (use your mouse pointer), select the desired Unit from the drop down list and click OK to close the window.
- *Minimum and Maximum values*: If you run a simulation in *Debug Mode* , the minimum and maximum value of every variable is shown.
- *Unit Selection*: If a quantity was assigned to the selected variable with multiple units (e.g. position -> m , mm , inch , etc.) you can switch units by the unit selection 

at the bottom of the variable list.
- *Expand Vectors/Matrices*: Select complete vectors and matrices or only their elements.
- *To Matlab*: Use the button to export a variable value to Matlab. You will be asked to enter a Matlab variable. Data will be exchanged between this Matlab variable and the variable that is selected in 20-sim.
- *Show Variables*: Select the specific kind of variables (Variables, States/Rates, Dependent/Algebraic) that should be displayed in the Variables list.

Actions

If you select a variable in the list and use the right mouse menu several actions can be performed:

- *Add to Favorites*: add the selected variable to the favorites list.
- *Add to Input Probes*: Add the variable to the list of inputs for linearization. This list is used in the Frequency Response dialog.
- *Add to Output Probes*: Add the variable to the list of outputs for linearization. This list is used in the Frequency Response dialog.
- *Copy*: copy the variable value to clipboard.
- *Copy* with Unit: copy the variable value and corresponding value to clipboard.
- *Copy Specified*: Specify the items that should be copied and then copy to clipboard.

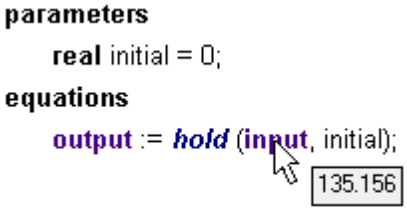
Tips

1. All equations in 20-sim are calculated using standard SI-units. You can however display the results, using whatever Unit you like. In the *Variable Chooser*, you can change the unit using the little drop down list just above the *Show Variables* section.
2. To see global variables, select the top level of the hierarchy, i.e. the main model listed on top.
3. To quickly find some variables out of the list, add the keyword interesting in the equation description. In the Variables list, de-select States, Rates, Algebraic Loops, Dependent States and Dependent Rates. Then only the interesting variables are shown.
4. When using large models, opening the Variable Chooser may take some time. Use the General Properties window to set the maximum number of variables that should be shown.
5. To hide variables from the list, add the keyword hidden in the equation description. Hidden variables are not visible in the *Variable Chooser*.

6.2.7 Debugging

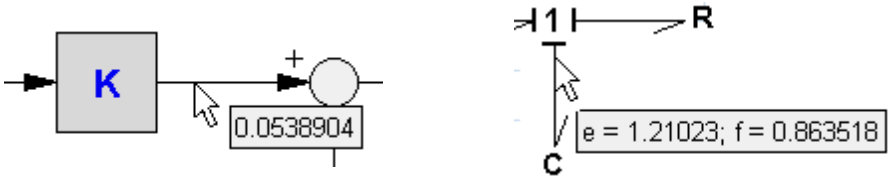
20-sim has various tool to help you to pinpoint and resolve problems that may occur during a simulation.

- *Switch to Debug Mode*: 20-sim is operating in *Debug Mode* when you see the *Debug Mode button* . If your *Simulator* or *Editor* is in *Fast Mode* click on the button to change to *Debug Mode*.
- *Recompile your model*: In the editor select *Check Complete Model* from the *Model* menu. This will recompile your model in *Debug* mode and show a maximum of warnings. If any warning looks suspicious, solve this first.
- *Breakpoints*: Set breakpoints to monitor time or conditions between variables.
- *Run to the breakpoint*: Use the *Run* command from the *Simulation* menu to run the simulation to a breakpoint. As soon as the breakpoint has been reached, the simulation will halt. The *Output tab* at the bottom of the Simulator will display which breakpoint was active.
- *Simulate Step-by-Step*: Use the command One Step from the *Simulation* menu to simulate do one simulation step for the complete model.
- *Inspect the Results*: After each step, open the Variable Chooser to inspect the various variables of the model. You can also switch to the *Editor* to inspect results at model level:
  - *Submodels*: Select a specific submodel and choose *parameters* or *variables* to inspect the parameters and variables of that submodel.
  - *Equation submodels*: Put the mouse pointer on top of an equation and point to a variable or parameter of interest. A box will appear, showing the numerical value associated with that variable or parameter.



Use the mouse pointer to inspect numerical values.

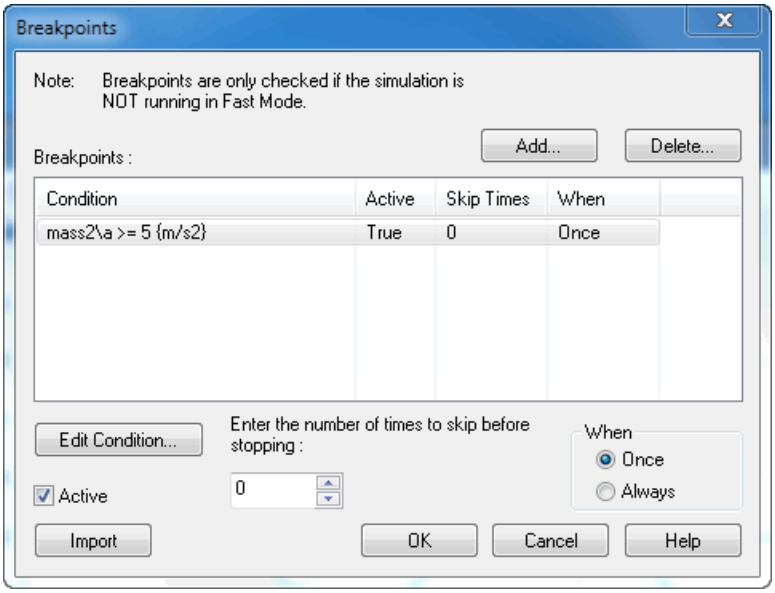
  - Graphical submodels: Put the mouse pointer on top of a signal, bond or connection. A box will appear, showing the numerical value associated with that signal, bond or connection.



Use the mouse pointer to inspect numerical values.

6.2.8 Breakpoints

20-sim allows you define stops in a simulation called breakpoints. You can set breakpoints using the Breakpoint Editor. This editor can be opened by selecting the *Breakpoints* command from the *Properties* menu in the Simulator.



Setting Breakpoints.

You can run a simulation from breakpoint to breakpoint, using the *Continue* command from the *Simulation* menu.

Breakpoints are useful when problems occur during simulation. By defining an appropriate breakpoint and using the *Run* command, you can run a simulation until the point where the problem occurs. You can continue the simulation using single steps or just running it again.

Items

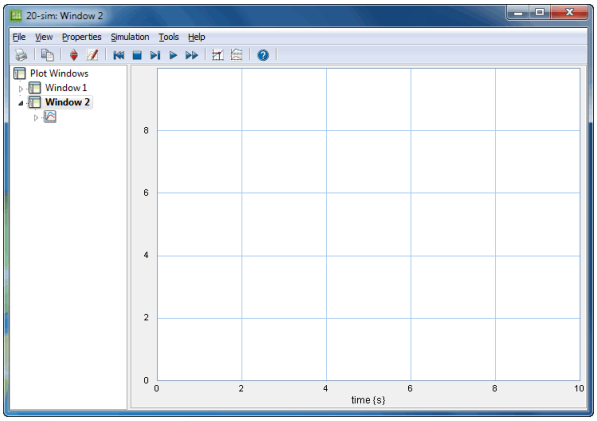
- *Add/Delete*: Use the Add and Delete buttons to add or delete breakpoints. When you add a breakpoint, an editor will be opened to enter the breakpoint conditions.
- *Breakpoints*: The Breakpoints list shows the entered breakpoints.
- *Enter the ... skip before stopping*: A breakpoint may occur more than once before it should stop the simulation run. Enter here the number of times to skip the breakpoint.
- *When*: A breakpoint may occur more than once during a simulation run. With this option you can select the breakpoint to be active only once or always.
- *Active*: Use this option to make a breakpoint effective or not.
- *Edit Condition*: When a breakpoint is selected in the breakpoints list, you can edit it, using the Edit Condition button. This will open an editor which helps you to enter the breakpoint condition.
- *Import*: Import breakpoints from another model.

Note

Breakpoints are only active when 20-sim is operating in *Debug Mode!*

6.2.9 New Simulation Plot

You can open additional plots by selecting the *New Plot Window* command from the *View* menu of the simulator.

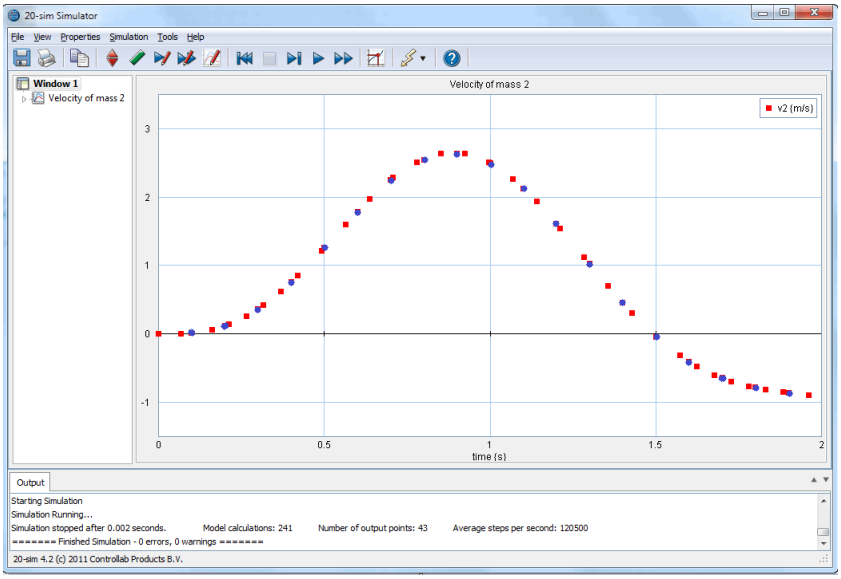


You can open additional plots in 20-sim.

Every plot will have a menu bar and buttons to allow you to enter the settings for the plot and running a simulation.

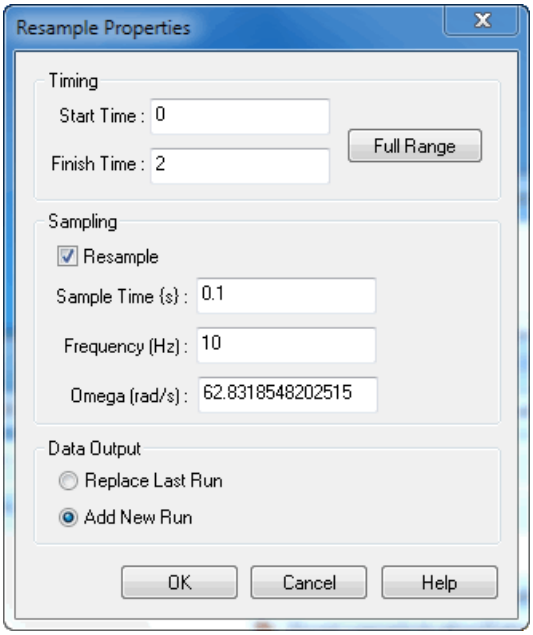
6.2.10 Resample Curves

Select the *Resample Curves* command from the *View* menu to sample a simulation run with a fixed sample time. This is useful if you want to use your simulation results in other programs that need data with a fixed time step. An example is given below where a curve (red dots) is shown that is simulated with a variable step integration method. Resampling with a sample time of 0.1 s will result in a curve with only the blue dots.



Use the Resample Curves command to get equidistant timing in the plot.

Selection of the *Resample Curves* command of the *View* menu pops up the *Resample Curves dialog*. You can enter the amount of data that you want to resample and the sample time or frequency.



The Resample Curves dialog.

Note

To get simulation data with a fixed time step, using a fixed step integration method is not enough, because this data may contain additional steps generated by time or state events. **Only the Resample Curves command will guarantee fixed time step data!**

6.2.11 Copy States

To get the same model behavior, compared to the end of the previous run, we have to copy the states of a model to the initial values.

When you use the **Copy States** command of the **Simulation** menu, 20-sim overwrites the initial values of the functions ddt, int, limint and resint with their current output states. This can be done at the end of a simulation run or at any point where a simulation run was interrupted.

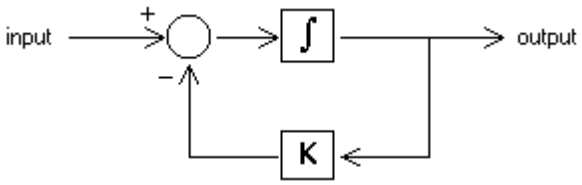
Tip

For some models, behavior at some operating point or "steady state" is of interest. Use the **Copy States** command to create simulation that directly starts at working level:

1. Run a simulation from t = 0 [s] until the operating point is reached. Use the **Copy States** command to store the states as new initial values. Now save the experiment using some special name.
2. Now each time you open this experiment, simulations directly start at operating point.

States

Consider the following first order linear model:



This model can be described by the dynamic equation:

output = int(0) + int(f(input,output))

with:

int(0) = the initial value of the integral
*f(input,output) = input - K*output*

In more general terms this equation can also be described as:

state = int(0) + int(rate)
rate = f(input, state)

When we start to simulate this model at t = 0 [s] the value of the integral is zero. The state is therefore equal to the initial value int(0):

state(0) = int(0)

At the end of a simulation, the integral may be unequal to zero. The state is therefore equal to:

state(end) = int(0) + int(rate(end)).

In other words, the value of the state is *characteristic* for the behavior of the model. To start a new simulation with the *same behavior*, we have to change the initial value to:

int(0) = state(end)

This is valid for all dynamic models: The states of a model are characteristic for its behavior. To get the same behavior, compared to the end of a previous run, we have to copy the states to the initial values.

6.2.12 Exporting Simulations

There are several methods to export simulation results:

1. Export Plot to Clipboard or File.
2. Export Experiment To Clipboard.
3. Export Data to File.

Export to Clipboard or File

From the *File* menu, you can choose *Export Plot to MetaFile* or *Export Plot to Clipboard* to export a drawing of the simulation plot.

Export High Resolution Plot to Clipboard or File

Simulation plots can be very dense. To limit the file size, by default not all details are exported to clipboard or to file. If you need a detailed plot, from the *File* menu, you can choose *Export High Resolution Plot to Clipboard or Bitmap.* Another option is to use the right mouse menu, when the mouse on top of a plot and select *High Resolution Export*.

Export Experiment To Clipboard.

From the *File* menu, you can choose *Export Experiment to Clipboard* to export all plot settings. A dialog will be opened asking you to specify the settings that should be copied. You can select:

- Parameters
- Initial Values
- Plot Specifications
- Run Specifications
- Multiple Run Specifications
- Export Data Specifications
- BreakPoints

Export Data to File

From the *File* menu, you can choose *Save Data To File* to export the plotted values to a data file. A file dialog window will pop up asking you to enter a filename. You can choose from the following file formats:

- Comma Separated File (*.csv): This is the default format and can be used for exporting data to spreadheet programs such as Microsoft Excel or Open Office Calc.
- 20-sim data file (*.n): This is the standard 20-sim format.
- Data Files (*.dat): The standard ANSI format for data files.

Tip

You can use the *Export Data To File* command to store simulation results of time consuming simulations. For later use you can use the *Import From Data File* command to read the result from file.

6.2.13 Importing Simulations

There are several methods to import simulation results:

1. Import experiment.
2. Import simulation data.

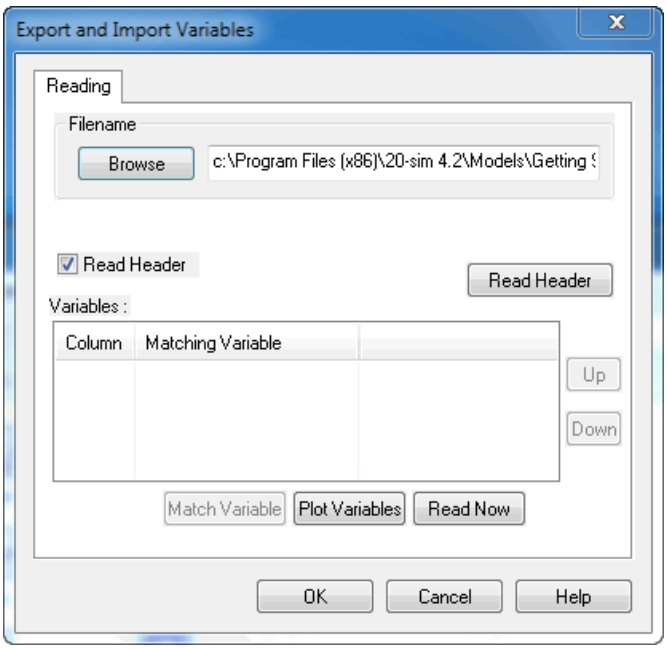
Export Plot Settings

From the *File* menu, you can choose *Import Experiment* to import the simulation settings from another model. The setting contain all the necessary information that you need to define a simulation:

- parameters
- run porperties
- plot properties
- ...

Import Data File

From the *File* menu, you can choose *Import Data From File* to import a previously saved simulation plot. A window will pop up allowing you to specify the filename and variables that should be plotted.



Import data from file.

If the file contains a header describing the variables inside, they will be shown in the variables dialog.

- *Filename*: You can select an existing filename using the browse button or type one. Data can be read as text file or as binary format.
- *Read Header*: Select this button if you want extra information (variable names etc.) to be read from the data file.
- *Variables*: The variables box shows the number of columns that were found in the data file. Each column must be matched with a variable that is shown in the plot. Click with the mouse on a column number to select it. Then click the Match Variable button. This opens a Variable Chooser in which you can select a variable. When a header was used when storing the data file, this matching is done automatically.
- *Match Variable*: Use this button to select a matching variable for each column in the variables box.
- *Plot Variables*: Use the Delete button to delete a selected variable
- *Plot Variables*: Use this button to put the same variables as shown in the plot into the columns of the variables box.
- *Up/Down*: Use the Up and Down buttons to change the order of the variables shown in the list.

Tip

You can use the *Export Data To File* command to store simulation results of time consuming simulations. For later use you can use the *Import From Data File* command to read the result from file.

6.2.14 Full Screen Mode

You can show simulation plots and 3D Animations in full screen mode.

1. Put your mouse on top of a plot or animation.
2. From the **right mouse menu**, select **Toggle Full Screen**.

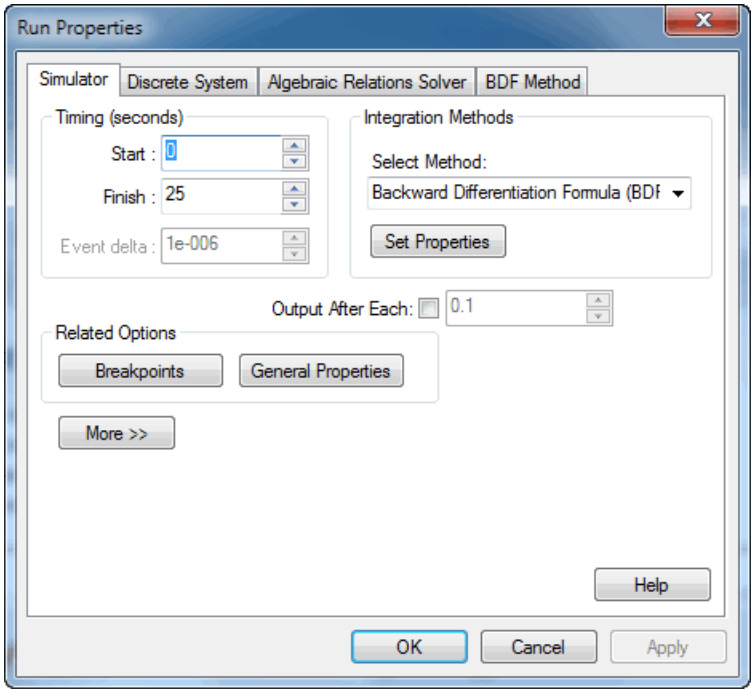
To remove the full screen mode:

1. From the **right mouse menu**, select **Toggle Full Screen** or press the **Escape** key.

6.3 Run Properties

6.3.1 Simulator Tab

The Simulator tab is the first tab of the *Run Properties Editor*. This tab shows the general simulation settings:



The 20-sim Run Properties.

Items

- *Timing (seconds)*:
  - *Start*: the start time of a simulation run (default: 0)
  - *Finish*: the finish time of a simulation run (default: 10)
  - *Event delta*: the accuracy for spotting state events. This option can only be selected if a model contains state events.
- *Integration Methods*: 8 numerical integration methods are available, each with its own parameters. The parameters can be changed by clicking the Set Properties button or selecting the tab on top of the editor.

  - Euler
  - Backward Euler

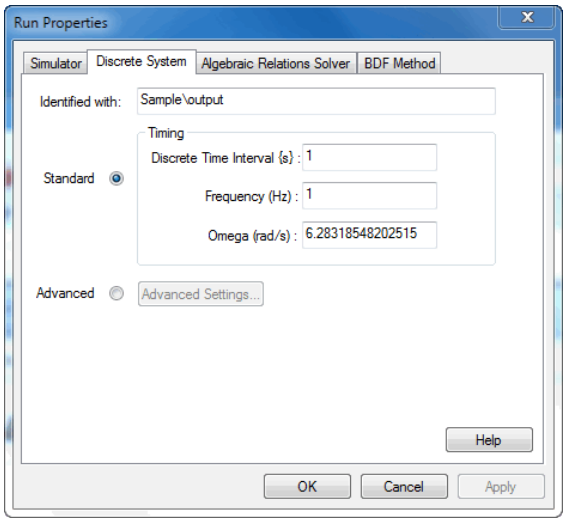
  - Adams-Bashford 2
  - Runge Kutta 2
  - Runge-Kutta 4
  - Runge Kutta Dormand Prince 8
  - Runge-Kutta-Fehlberg
  - Vode Adams
  - Backward Differentiation Formula (BDF)
  - Modified Backward Differentiation Formula (MBDF)
- *Output After Each*: This is the plot interval. Use this option to generate output only after each equidistant time interval has passed. For each interval, the exact simulation point is used (if available) or the nearest point is used (if not available, for example when using variable step algorithms).
- *Related Options*
  - *BreakPoints*: Click this button to open the Breakpoints Editor.
  - *General Properties*: Click this button to open the General Properties Editor.
  - *More*: Click this button to set advanced options:
    - *Endless*: Select this option if you do not want the simulation to stop.
    - *Attempting Real-Time simulation*: Select this option if you want the simulation to run as fast as the real time. You can choose the option *Catch up with lost time* and set the *Maximum allowed lost time*.

Tips

- Use *Attempting Real-Time simulation* when you have keyboard input or joystick input in your model.

6.3.2 Discrete System Tab

The Run Properties Editor can be used to edit the sample frequency of a discrete-time system in the model. 20-sim automatically detects the existence of discrete systems in a model. For each discrete system, one extra tab (Discrete System) is shown in the *Run Properties Editor.*



The Discrete Systems tab.

Items

- *Identified with*: Each discrete system in a model is assigned a unique identifier. 20-sim detects discrete systems by looking for discrete functions and variables:

  1. sample
  2. hold
  3. next
  4. previous
  5. sampletime

  When a function has been found, the elements of the corresponding discrete system are identified by propagation of the equations. In order of appearance in the model, the discrete system is assigned a unique identifier:

  1. sample -> the variable that is assigned to the output of the sample function.
  2. hold -> the variable that is assigned to the output of the hold function.
  3. previous -> the discrete state corresponding with the previous function.

4. next -> the discrete state corresponding with the next function.
5. sampletime -> the variable that is equal to the sampletime.

- *Timing (Standard)*: Each discrete system has a default sample frequency of 1 Hz. You can change this frequency to any desired value:
  - *Discrete Time Interval (s)*: The sample interval (in s) of the discrete system
  - *Frequency (Hz)*: The sample frequency (in Hz) of the discrete system.
- *Timing (Advanced)***:** For discrete-time control loops that interact with physical continuous-time systems through sensors (analog to digital) and actuators (digital to analog) the specific timing is important. Click the Advanced button to specify this timing.

6.3.3 Advanced Discrete System Settings

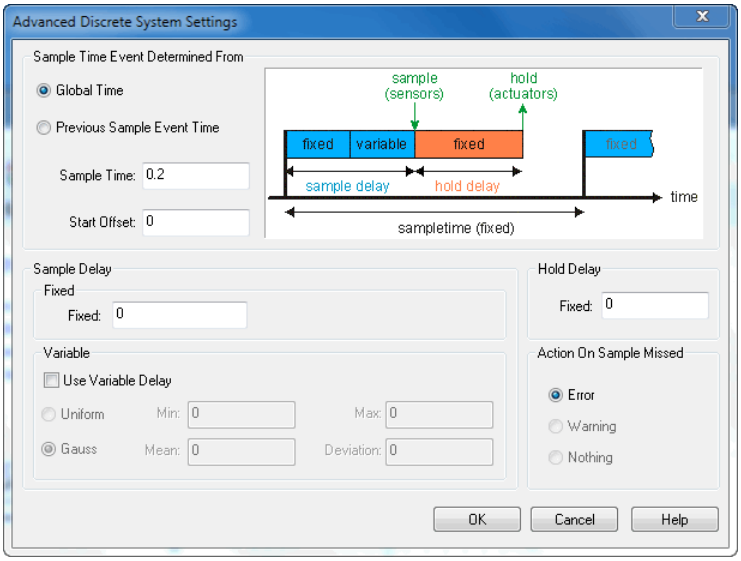
The *Advanced Discrete System Settings* dialog can be used to set the timing for discrete-time control loops that interact with physical continuous-time systems through sensors (analog to digital) and actuators (digital to analog). You can open the dialog by:

1. In the Simulator from the **Properties** menu, click **Run** to open the Run Properties Editor.
2. Select the **Discrete System tab** and click the **Advanced Settings** button.

In the *Advanced Discrete System Settings* dialog you can specify the timing of control loops with a fixed *sampletime* (click *Global Time*) and control loops with a variable *sampletime* (click *Previous Sample Event Time*).

Fixed Sampletime

When you click the **Global Time** option the following window appears.



Advanced settings for discrete systems.

The window shows the timing diagram for a fixed *sampletime* control loop. The loops starts with an clock interrupt from a timer. It will take some time before the interrupt is handled and generally some calculations are performed before the sensor signals (*sample*) are read. This delay is represented by the *sample delay* which is divided in a fixed time and a variable time part. The control loop will proceed and perform the necessary calculations to generate the actuator output (*hold*). These calculations will take some time, which is represented by the fixed time *hold* delay. After this action the control loops will come to a halt until the next clock interrupt is generated.

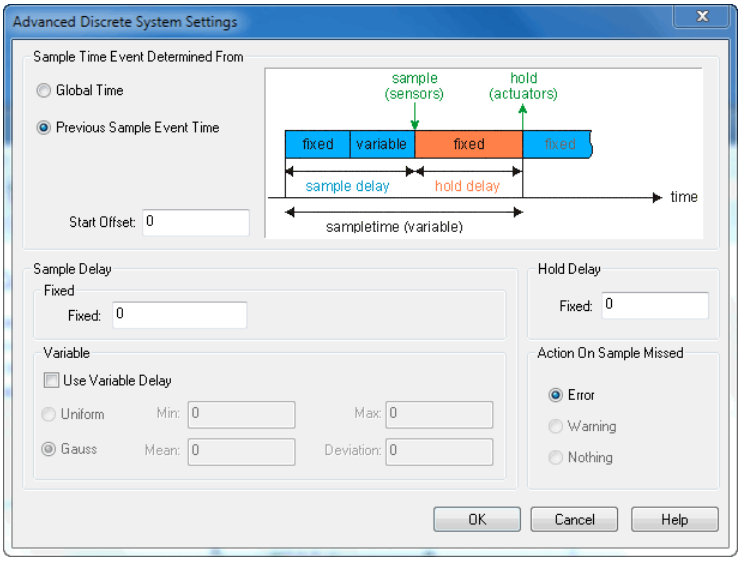
If the control loops is properly designed it will know the task it is performing when a clock interrupt is getting in. E.g. if it is still calculating the actuator outputs when clock interrupt is getting in, an error will be generated, specifying that the sampletime is too small to perform all necessary calculations.

Items

- *Sample Delay*:
  - *Fixed*: Enter the fixed sample delay here.
  - *Variable*: Enter the variable delay here. You can chose from two distributions:
    - *Uniform*: The delay is uniformly distributed between the given minimum and maximum.
    - *Gaussian*: The delay has a given mean and standard deviation.
- *Hold Delay*: Enter the fixed hold delay here.
- *Action on Sample Missed*: An error will be generated when the total delay is larger than the sampletime or when the total delay is negative.

Variable Sampletime

When you click the **Previous Sample Event Time** option the following window appears.



Variable sample time.

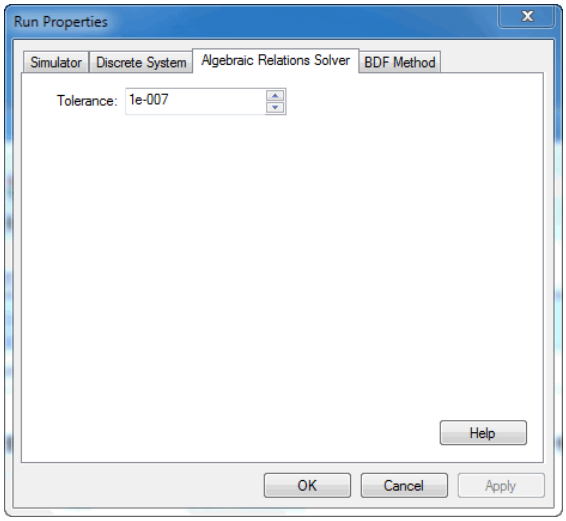
The window above shows the timing diagram for a variable sampletime control loop. It performs the same tasks as a fixed sampletime control loop, but will start the next loop as soon as the previous loop was finished. The variable sampletime control loop does not use a clock interrupt!

Items

- *Sample Delay*:
  - *Fixed*: Enter the fixed sample delay here.
  - *Variable*: Enter the variable delay here. You can chose from two distributions:
    - *Uniform*: The delay is uniformly distributed between the given minimum and maximum.
    - *Gaussian*: The delay has a given mean and standard deviation.
- *Hold Delay*: Enter the fixed hold delay here.
- *Action on Sample Missed*: Only an error will be generated when the total delay is negative.

6.3.4 Algebraic Relations Solver Tab

The *Algebraic Relations Solver tab* of the Run Properties Editor is visible when your model contains algebraic loops that could not be solved during processing. In the tab, you can specify the tolerance of the algebraic solver algorithm. For each model evaluation, this solver tries to solve the algebraic loop relations, while standard integration algorithms, such as Euler, can be use to evaluate the model equations.



The Algebraic Relations Solver tab.

6.3.5 Euler

This is the explicit Euler method. It is a single-step explicit method, which needs one model calculation per time step. The size of the time step is fixed. This simulation algorithm requires 1 parameter to be specified:

- *Step Size*: the step time of the integration algorithm (default: 0.01).

6.3.6 Backward Euler

This is a combination of the implicit Backward Euler method and the explicit Forward Euler method. It is a single-step implicit method, which needs one model calculation per time step. With the parameter alpha the combination is defined:

alpha	combination	
-1	100% Forward Euler	explicit
0	50% Forward Euler, 50 % Backward Euler	explicit
1	100% Backward Euler	implicit

- *Step Size*: the step time of the integration algorithm (default: 0.01).
- *Relative Tolerance*: Relative tolerance for the rootfinding in the Backward Euler method.
- *Alpha*: Choose between Forward Euler (alpha = -1), Trapezoidal (alpha = 0) and Backward Euler (alpha = 1).

6.3.7 Adams-Bashford

This is the trapezoidal rule. It is a single-step explicit method, which needs only two model calculations per time step. The size of the time step is fixed. This simulation algorithm requires 1 parameter to be specified:

- **Step Size**: the step time of the integration algorithm (default: 0.1).

6.3.8 Runge Kutta 2

This is an explicit single-step second-order derivatives method which needs 2 model calculations per time step. It is the simple form of the classical Runge-Kutta method. The size of the time step is fixed. This simulation algorithm requires 1 parameter to be specified:

- *Step Size*: the step time of the integration algorithm (default: 0.01).

6.3.9 Runge-Kutta 4

This is an explicit single-step fourth-order derivatives method which needs 4 model calculations per time step. It is the classical Runge-Kutta method. The size of the time step is fixed. This simulation algorithm requires 1 parameter to be specified:

- *Step Size*: the step time of the integration algorithm (default: 0.01).

6.3.10 Runge-Kutta-Fehlberg

This is an explicit variable-step 4/5-order derivatives method, primarily designed to solve non-stiff and mildly stiff differential equations. Because the method has very low overhead costs, it will usually result in the least expensive integration when solving problems requiring a modest amount of accuracy and having equations that are not costly to evaluate. This simulation algorithm has 4 parameters:

- *Integration Error* (required)
  - *Absolute*: The absolute integration error, valid for every state variable (default: 1e-6).
  - *Relative*: The relative integration error, valid for every state variable (default: 1e-6).
- *Step Size* (not required)
  - *Initial*: The step size for the first simulation step (default: 0.01).
  - *Maximum*: The maximum size of a simulation step (default: 1).

6.3.11 Runge Kutta Dormand Prince 8

This is an explicit variable-step 8-order derivatives method, primarily designed to solve non-stiff and stiff differential equations. This simulation algorithm has 4 parameters:

- *Integration Error* (required)
  - *Absolute*: The absolute integration error, valid for every state variable (default: 1e-6).
  - *Relative*: The relative integration error, valid for every state variable (default: 1e-6).
- *Step Size* (not required)
  - *Initial*: The step size for the first simulation step (default: 0.01).
  - *Maximum*: The maximum size of a simulation step (default: 1).

6.3.12 Vode Adams

This is the explicit variable-step stiff integration algorithm of Cohen and Hindmarsh. This method is specially suited for explicit models that combine high and low frequency vibrations with little damping (so called "stiff" models). This simulation algorithm has 4 parameters:

- *Integration Error* (required)
  - *Absolute*: The absolute integration error, valid for every state variable (default: 1e-6).
  - *Relative*: The relative integration error, valid for every state variable (default: 1e-6).
- *Maximum Step Size* (not required): The maximum size of a simulation step (default: 1).
- *Multistep Method* (required): The variable step integration method that is used. A BDF method is default and best suited for stiff systems.
- *Iteration Type* (required): The method used for iteration loops (algebraic relations). The Newton method is default and best suited for stiff systems.

6.3.13 Backward Differentiation Formula (BDF)

This is an implicit multi-step variable-order first-order derivatives method. It is Gear's backward differentiation formula, which is up to fifth order, so 1 to 5 model calculations are needed per time step. The code of the Dassl library is used. It is suitable for models having derivative causalities and/or algebraic loops. This simulation algorithm has 4 parameters:

- *Integration Error* (required)
  - *Absolute*: The absolute integration error, valid for every state variable (default: 1e-5).
  - *Relative*: The relative integration error, valid for every state variable (default: 1e-5).

- *Step Size* (not required)
  - *Initial*: The step size for the first simulation step (default: 0.1).
  - *Maximum*: The maximum size of a simulation step (default: 1).

Note

- Undamped models are not simulated properly with this method.
- The default values are suitable for problems with time constants of order of magnitude of 0.1 to 1. Initial Step Size and Maximal Step Size influence the behavior of the method and should be tuned with care.

6.3.14 Modified Backward Differentiation Formula (MBDF)

This is an implicit multi-step variable-order first-order derivatives method. It is Gear's backward differentiation formula, which is up to fifth order, so 1 to 5 model calculations are needed per time step. The code of the Dassl library is used. It is suitable for models having derivative causalities and/or algebraic loops. The MBDF allows the use of constraint variables when proper initial values are used.

This simulation algorithm has 4 parameters:

- *Integration Error* (required)
  - *Absolute*: The absolute integration error, valid for every state variable (default: 1e-5).
  - *Relative*: The relative integration error, valid for every state variable (default: 1e-5).
- *Step Size* (not required)
  - *Initial*: The step size for the first simulation step (default: 0.1).
  - *Maximum*: The maximum size of a simulation step (default: 1).

Note

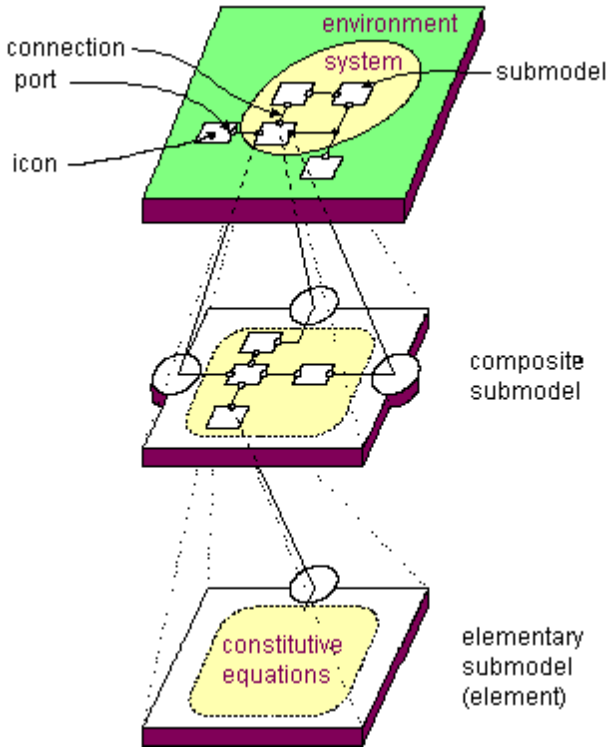
- This is the only method that supports the use of the constraint function.
- The MeBDFi method cannot handle contraints that do not influence the model. Therefore 20-sim will check at the beginning of the simulation if contraints have been properly defined. The contraints that do not influence the model are set to zero automatically. If you do not want to use this check, deselect the **Smart Contraint Solving** option.
- Undamped models are not simulated properly with this method.
- The default values are suitable for problems with time constants of order of magnitude of 0.1 to 1. Initial Step Size and Maximal Step Size influence the behavior of the method and should be tuned with care.

7 Language Reference

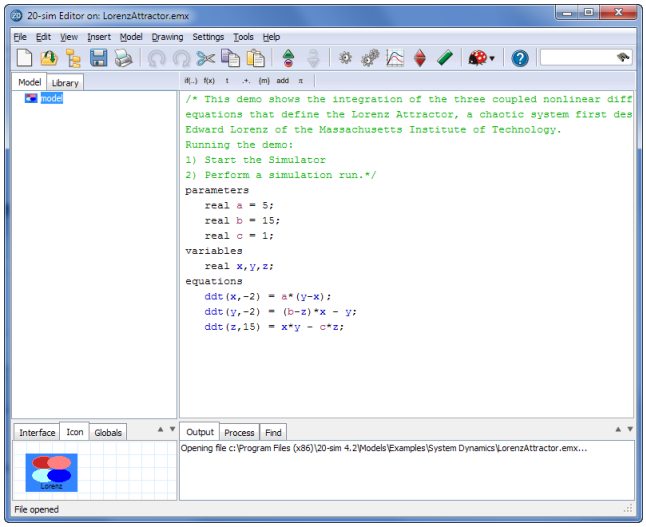
7.1 Introduction

7.1.1 Language Reference

Every (sub)model has an implementation. This can be a composition of lower level submodels, which themselves are composed of lower level submodels etc. At the bottom of this hierarchy the submodels consist of a set of mathematical equations (equation submodel). These submodels are therefore known as *equation submodels*.



All equations used in 20-sim are described in the language *SIDOPS+*. A simple equation model written in *SIDOPS+* is shown below:



A simple equation model in 20-sim.

You will find out that in most cases the *SIDOPS+* language is equal to standard mathematical notation. Regardless of your modeling background, you can learn quickly to build your own equation models. 20-sim comes with over 80 built-in functions that do most of the work for you. To learn more about equation models, have a look at the following sections:

1. Model Layout
2. Keywords (Constants, Parameters, Variables, Equations)
3. Data Types
4. Operators
5. Functions
6. Statements

7.1.2 Model Layout

The general layout of an equation model is:

constants
    // enter your constants here, for a description see constants
parameters
    // enter your parameters here, for a description see parameters
variables

```
        // enter your variables here, for a description see variables
initialequations
        // enter your initial equations here, for a description see initialequations
code
        // enter your equations here, for a description see code
equations
        // enter your equations here, for a description see equations
finalequations
        // enter your final equations here, for a description see finalequations
```

Only the *equations* section is required. The other sections are optional.

7.1.3 Equations

The body of an equation model consists of the keywords *initialequations*, *code*, *equations* and *finalequations*, each followed by one or more equations. Equations are relations between variables indicated by an equal (=) sign:

```
y = x + 1;
z + y = sin(time*2*pi*f);
g = ddt(g + z, g_initial);
```

An equation can be combination of operators, functions, powerports variables, inputs, outputs, constants, parameters and variables. Equations in 20-sim follow the standard mathematical notation.

Rules

The following rules must be obeyed when specifying equations:

1. An equation may span more than one line but must always be finished by a semicolon (;).
2. Parentheses may be used to indicate grouping as in ordinary mathematical notation, e.g.
   `u = sin(time*f*2*3.1415 + (a + b)/c );`
4. Extensions like .e or .f are used to indicate the effort and the flow variables of powerports. These extensions are not allowed for any other type of identifier (i.e. inputs, outputs, constants, parameters or variables).
5. 20-sim has some predefined variables which have a special meaning.
6. Each variable must be assigned a value exactly once.
7. Errors are shown automatically after checking in the Process tab. Double clicking on the error text will make 20-sim jump to the error, which is then colored red.

Execution

1. Equations within 20-sim may be entered in random form. During compilation, 20-sim will automatically try to rewrite equations into a correct order of execution.

2. Some integration algorithms do more calculations before generating the next output value. These calculations are called minor steps, the output generation is called a major step. During a minor step, all model equations are executed. In most cases you will not notice this because only the results of the major step are shown in the simulator, but in some cases this may cause unwanted results.

7.1.4 Equation Sections

In 20-sim equations can be divided in four sections: *initialequations*, *equations, code* and *finalequations*.

Initialequations

The *initialequations* section should contain all equations that should calculated before the simulation run. The equations are calculated once. The equations in this section form a code block, i.e. the equations themselves and the order of the equations are not rewritten during compilation and executed sequentially. The results of the initialequations section are not shown in the simulator plots and can therefore not be inspected in the Numerical Values window!

Example

The *initialequations section* is used in the library model *Spring Damper (stiffness)*:

```
parameters
      real k = 10.0 {N/m};
      real m = 1.0 {kg};
      real b = 0.05 {};
variables
      real x {m};
      real d  {damping,N.s/m};
initialequations
      d = 2*b*sqrt(k*m);
equations
      x = int (p.v);
      p.F = k * x + d*p.v;
```

As you can see in the *initialequations section* the damping *d* is calculated out of the relative damping *b*, mass *m* and stiffness *k*.

Equations

The equations section contains all standard equations. They do not have to be entered in a sequential form. They are rewritten during compilation to get the correct order of execution. The resulting code is calculated every simulation step. You can see the use of the *equations section* in the example above.

Code

The *code* section contains a code block. A code block contains all equations that must be calculated sequentially, i.e. the equations themselves and the order of the equations are not rewritten during compilation. The resulting sequential code is calculated every simulation step.

Example

The *code section* is used in the library model *Spring Damper (stiffness)*:

```
parameters
      real initial = 0.0;
variables
      real prev, peak;
initialequations
      peak = initial;
      prev = 0;
      output = initial;
code
      if major then
                  peak = max([abs(input), peak]);
                  if (input > 0 and prev < 0) or (input < 0 and prev > 0) then
                        output = peak;
                        peak = 0;
                  end;
                  prev = input;
      end;
```

Finalequations

The finalequations section should contain all equations that should be calculated after a simulation run. The equations are calculated once. The equations in this section form a code block, i.e. the equations themselves and the order of the equations are not rewritten during compilation and executed sequentially. The results of the finalequations section are not shown in the simulator plots and can therefore not be inspected in the Numerical Values window!

Example

The *finalequations section* is used in the library model *DoMatlab-Final*:

```
parameters
      string command = '';
finalequations
      // send the command to the workspace
      doMatlab (command);
```

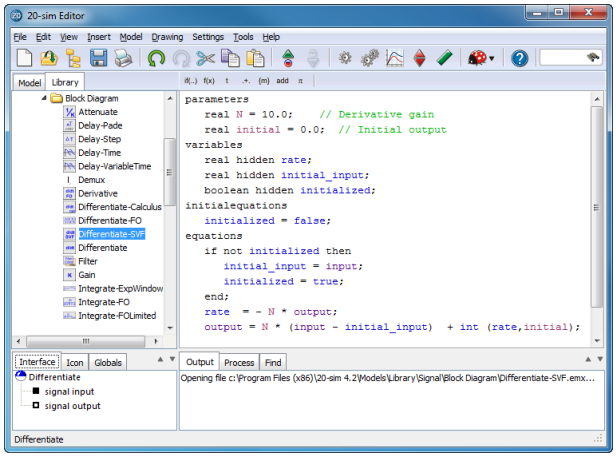
As you see you can give the parameter *command* any value, which is then exported to Matlab at the end of the simulation run.

7.1.5 Using Port Names

To use port variables in equations, you must use the port names that are defined in the *Interface Editor*.

Signal Ports

For signal ports, you can directly use the port names. The example below, shows the library model *Differentiate-SVF.emx*. In the *Interface Editor*, two ports are defined with the names *input* and *output*. In the equations these names are used.

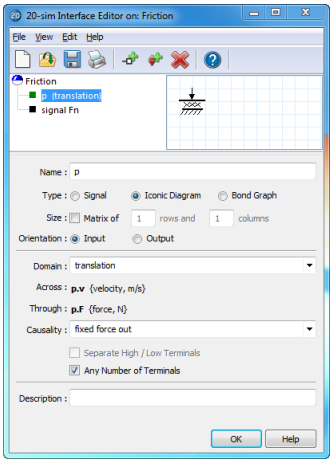


The Interface tab shows the contents of the Interface Editor.

Power ports

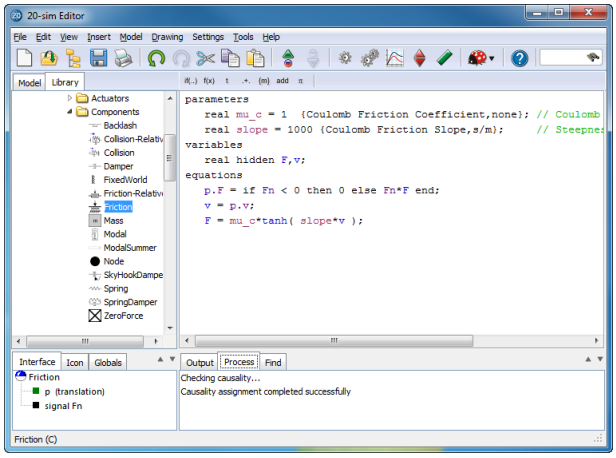
Power ports describe the flow of power and are always characterized by two variables. For bond graph ports the extensions *.e* and *.f* are used to indicate these variables. For iconic diagram ports many aliases are known. You can use these variables in equations by typing the port name followed by a dot and the extension.

You can always find the correct extensions by opening the *Interface Editor* and selecting a port. The example below, shows the library model *Friction.emx*. In the Interface Editor, one power port is defined with the name *p*. As shown, the correct description of both variables is *p.v* and *p.F*.



Double clicking in the Interface tab will open the Interface Editor.

The equation description of this model (see below) shows the use of the variables *p.v* and *p.F*.



In the equations the port variables p.F and p.v are used.

7.1.6 Reserved Words

While entering equations, the 20-sim Editor will automatically detect reserved words. You can recognize reserved words, because they will be shown with a different color:

element	**color**
keywords	**black**
data types	**black**
functions	***blue***
statements	**black**
predefined variables	**blue**

20-sim knows the following reserved words:

- , * , .* , ./ , .^ , / , ^ , + , < , <= , <> , == , > , >= , abs , adjoint , Adjoint , algebraic , and , antisym , arccos , arccosh , arcsin , arcsinh , arctan , arctanh , atan2 , boolean , by , ceil , code , collect , columns , constants , constraint , cos , cosh , cross , data , ddt , delta , det , diag , direct , discrete , div , dll , dlldynamic , dly , do , doMatlab , effortincausality , else , end , equal , equations , event , eventdown , eventup , exp , exp10 , exp2 , export , externals , eye , false , finalequations , first , floor , flowincausality , for , frequencyevent, from , fromMatlab , gauss , global , hidden , hold , homogeneous , if , import , initialequations , inner , int , integer , interesting , inverse , inverseH, limint , limit , log , log10 , log2 , major , max , min , mod , msum , mul , next , nonlinear , norm , norminf , not , or , parameters , previous , ramp , ran , random , real , repeat , resint , resint , return , round , rows , sample , sampletime , sign , sin , sinh , skew , sqr , sqrt , step , stopsimulation , string , sum , sym , table , tan , tanh , tdelay , then , tilde , time , timeevent , to , toMatlab , trace , transpose , true , trunc , type , types , until, variables , warning , while , xor

7.1.7 White Space

Like C and other programming languages, 20-sim does not recognize white space (spaces and tabs, carriage returns) except in a string. We recommend that you use white space to make your model easier to follow. For example, the following model is dense and difficult to read:

```
variables
    real y,u;
equations
    y = step(1);
    u = -step(1);
    if time > 5 then y = -ramp(1); u = ramp(1);
    else y = ramp(1); u = -ramp(1); end;
```

Adding white space with indentation makes the same code much easier to read:

```
variables
    real y;
    real u;
equations
    y = step(1);
    u = -step(1);

    if time > 5 then
            y = -ramp(1);
            u = ramp(1);
    else
                y = ramp(1);
                u = -ramp(1);
    end;
```

7.1.8 Writing Comments

20-sim gives you several options to create comments in a model. You can use each option add explanatory text to your model or to exclude ("comment out") certain parts of your model for testing and debugging purposes.

Block Quotes ("...")

You can enclose a block of text between quotes ("). This method makes it easy to write a comment over multiple lines:

```
"This is a line of sample code that shows you how to
use the 20-sim comments"
```

Programming (/*...*/)

You can enclose a block of text between the constructs /* and */. This method is used often in programming and also allows to write a comment over multiple lines:

```
/*This is a line of sample code that shows you how to
use the 20-sim comments*/
```

Single Line Comment (//...)

A option often used in programming is to insert // into a line. The compiler ignores everything to the right of the double slashes on that line only.

```
// This is a line of sample code showing the
// 20-sim comments.
```

You can begin comments anywhere in a model. 20-sim will automatically recognize comments and color it green.

7.1.9 State and Time Events

Normally an integration algorithm uses a fixed time step or a variable step, depending only on the model dynamics. In some cases, the algorithm is forced to perform calculations at a specific time.

Time Events

The most simple form of forced calculation is calculation at a specific time. This is called a *time event*. The following functions cause time events.

function	description
timeevent	Forces the integration algorithm perform a calculation at a specific time.
frequencyevent	Forces the integration algorithm perform a calculation at a specific frequency.
sample , hold, next, previous	These functions introduce a discrete system in a model. A discrete system forces the integration algorithm to perform calculations every sampletime.

State Events

Sometimes the integration algorithm must iterate to find the next calculation step. This is called a *state event*. The following functions cause state events.

function	description
event	Forces the integration algorithm to find the exact point where the input crosses zero.
eventup	Forces the integration algorithm to find the exact point where the input crosses zero from a negative to a positive value.
eventdown	Forces the integration algorithm to find the exact point where the input crosses zero from a positive to a negative value.
limint	Forces the integration algorithm to find the exact point where the output become larger or smaller than the minimum and maximum parameters.
resint	Forces the integration algorithm to find the exact point where the input becomes zero.

To find the exact point where simulation should be performed iteration is used. The accuracy of the iteration can be set by the event delta parameter in the Run Properties window.

7.2 Keywords

7.2.1 Keywords

Before the actual equations, three sections can be entered denoted by the keywords *constants*, *parameters* and *variables*. This is shown in the example below:

```
constants
    real basefrequency = 60;
    real pi = 3.14159265;
parameters
    real amplitude = 1;
variables
    real plot;
equations
    plot = amplitude * sin (basefrequency*2*pi * time);
```

The order of appearance of these sections must always be: (1) constants, (2) parameters and (3) variables. The sections are optional. If you do not use constants parameters or variables, these sections may be omitted.

7.2.2 Constants

Constants are symbolic representations of numerical quantities that do not change during or in between simulation runs. Constants must be defined in 20-sim, using the constants keyword. After the keyword, lines with constant declarations can be entered. Every line must be finished by a semicolon (;). An example is given below:

```
constants
    real V = 1 {volume, m3};            // the volume of barrel 1
    real hidden D = 0.5 {length, m};    // the length of part 1
    ...
```

Types

20-sim currently supports four types of constants: *boolean*, *integer*, *real* and *string*. These types must be specified explicitly. The use of the type real is shown in the example above.

Hidden

To prevent users from inspecting constants, the keyword *hidden* can be applied. This keyword should follow the data type.

Constant names

Constant names can consist of characters and underscores. A constant name must always start with a character. Constant names are case-sensitive. No reserved words may be used for constants. If a reserved word is used, an error message will pop-up while checking the model. In the example above the names *V* and *D* are used.

Constant Values

It is advised to assign every constant a value. Otherwise 20-sim will assign a default value (0, false or an empty string), which can easily lead to misinterpretations. Unlike parameter values, constant values cannot be overridden in the Simulator.

Quantity and Unit

If a constant is known to represent a physical value, you can define the corresponding quantity and unit. 20-sim will use this information to do a unit check on equations. The use of quantities and units is optional. In the example above the quantities volume and *length* are used with the units *m3* and *m*.

Example

```
constants
    real pi = 3.14159265359;
    real two pi = 6.28318530718;
    integer i = 10 {length,m};          // vehicle length
    integer j = 1 {time,s};

constants
    string filename = 'data.txt';
    integer i = 1;
    real state = 0.0;
    boolean t = true, f = false;
```

Tip

- You can easily enter constants by using the Add button

in the *Equation Editor Taskbar*.
- You can easily select quantities and units by using the Units button

of the *Equation Editor Taskbar*.

7.2.3 Predefined Constants

Some constants in 20-sim have a predefined value. These constants do not have to be declared in the *constants* section of a model. You can add your own predefined constants in the file *Constants.ini* in the 20-sim bin-directory.

Standard

pi	3.14159265358979323846264338327 95
exp1	2.71828182845904523536028747135266
twopi	6.28318530717958647692528676655901
sqrt2	1.41421356237309504880168872420969
log2	0.69314718055994530941723212145817 66

Physics

g_n	9.80665 {m/s2}	Standard acceleration of gravity
G	66.7259e-12 {N.m2/kg2}	Newtonian constant of gravitation
c_0	299.792458e6 {m/s}	Speed of light (vacuum)
mu_0	1.25663706143591729538505 73533118e-6 {H/m}	Permeability (vacuum)
epsilon_0	8.854188e-12 {F/m}	Permittivity (vacuum)
h	6.6260755e-34 {J.s}	Planck constant
m_p	2.17671e-08 {kg}	Planck mass
l_p	1.61605e-35 {m}	Planck length
t_p	5.39056e-44 {s}	Planck time
e	1.60217733e-19 {C}	Elementary charge
R_H	25812.8056 {Ohm}	Quantized Hall resistance
mu_B	9.2740154e-24 {J/T}	Bohr magneton
mu_N	5.0507866e-27 {J/T}	Nuclear magneton
Ry	10973731.534 {1/m}	Rydberg constant
a_0	5.29177249e-11 {m}	Bohr radius
N_A	6.0221367e+23 {1/mol}	Avogadro constant
m_u	1.6605402e-27 {kg}	Atomic mass constant
F	96485.309 {C/mol}	Faraday constant
k	1.380658e-23 {J/K}	Boltzmann constant
sigma	5.67051e-08 {W/m2.K4}	Stefan-Boltzmann constant
b	0.002897756 {m.K}	Wien constant
R	8.31451 {J/mol.K}	Molar gas constant
V_m	0.0224141 {m3/mol}	Molar volume (ideal gas)
n_0	2.686763e+25 {1/m3}	Loschmidt constant
m_e	9.1093897e-31 {kg}	Electron mass
m_p	1.6726231e-27 {kg}	Proton mass
m_n	1.6749286e-27 {kg}	Neutron mass

Conversion Constants

eV	1.60217733e-19 {J};	Electron volt
u	1.6605402e-27 {kg}	Atomic mass unit
atm	101325 {Pa}	Standard atmosphere
deg	0.0174532925199432957692369076848861 {rad}	Angular degree

7.2.4 Parameters

Parameters are symbolic representations of numerical quantities which can only be changed after simulation has been stopped. They can therefore be changed in between of simulation runs, or after the user has (temporarily) stopped the simulation.

Parameters must be defined in 20-sim, using the parameters keyword. After the keyword, lines with parameter declarations can be entered. Every line must be finished by a semicolon (;) and may be followed by comment with a description of the parameter. An example is given below:

```
parameters
    real V = 1 {volume, m3};              // the volume of barrel 1
    real hidden D = 0.5 {length, m};      // the length of part 1
    ...
```

Types

20-sim currently supports four types of parameters: *boolean*, integer, real and string. These types must be specified explicitly. The use of the type real is shown in the example above.

Hidden

To prevent users from inspecting parameters or decreasing the list of parameters, the keyword *hidden* can be applied. This keyword should follow the data type.

Global

To use the same parameter in other submodels, the keyword *global* can be applied. This keyword should follow the data type.

Parameter Names

Parameter names can consist of characters and underscores. A parameter name must always start with a character. Parameter names are case-sensitive. No reserved words may be used for parameters. If a reserved word is used, an error message will pop-up while checking the model. In the example above the names *V* and *D* are used.

Parameter Values

Parameters can be assigned *default* values. These values can be overridden in the Simulator using the Parameters command.

Quantity and Unit

If a parameter is known to represent a physical value, you can define the corresponding quantity and unit. 20-sim will use this information to do a unit check on equations. The use of quantities and units is optional. In the example above the quantities *volume* and *length* are used with the units *m3* and *m*.

Comment

If comment is added at the end of the parameter declaration, it is shown in the Variable Chooser to facilitate the selection between variables.

Example

```
parameters
    real p1 = 12, p2, state = 1.1;
    integer i = 10 {length,m};          // vehicle length
    integer j = 1 {time,s};

parameters
    string filename = 'data.txt';   // This comment appears as a description in
the Parameters/Initial values editor.
    integer i = 1;                  // Just a parameter
    real state;                         // Initial velocity [m/s]
    boolean t = true;               // T = true
```

Tip

- You can easily enter parameters by using the Add button

in the *Equation Editor Taskbar*.
- You can easily select quantities and units by using the Units button

of the *Equation Editor Taskbar*.

7.2.5 Variables

Variables are symbolic representations of numerical quantities that may change during simulation runs. Variables must be defined in 20-sim, using the variables keyword. After the keyword, lines with variable declarations can be entered. Every line must be finished by a semicolon (;). An example is given below:

```
variables
    real interesting V {volume, m3};        // the volume of barrel 1
    real hidden D {length, m};              // the length of part 1
    ...
```

Types

20-sim currently supports four types of variables: *boolean*, *integer*, *real* and *string*. These types must be specified explicitly. The use of the type real is shown in the example above.

Interesting

By adding the keyword *interesting* after the variable type, a variable will be given special focus in the Variable Chooser. The Variable Chooser shows the complete list of variables of a model and is used to select variables for plotting. The keyword *interesting* is shown in the variable list, making it easier to find variables of special interest.

Hidden

To prevent users from inspecting variables or decreasing the list of variables, the keyword *hidden* can be applied. This keyword should follow the data type.

Global

To use the same variable in other submodels, the keyword *global* can be applied. This keyword should follow the data type.

Variable Names

Variable names can consist of characters and underscores. A variable name must always start with a character. Variable names are case-sensitive. No reserved words may be used for variables. If a reserved word is used, an error message will pop-up while checking the model.

Variable Values

Variables cannot be assigned default values.

Quantity and Unit

If a variable is known to represent a physical value, you can define the corresponding quantity and unit. 20-sim will use this information to do a unit check on equations. The use of quantities and units is optional. In the example above the quantities *volume* and *length* are used with the units *m3* and *m*.

Comment

If comment is added at the end of the variable declaration, it is shown in the *Variable Chooser* to facilitate the selection between variables.

Predefined Variables

20-sim has some predefined variables which have a special meaning.

Example

```
variables
    real interesting p1,p2,state;
    integer i;
    integer j;

variables
    string filename;
    integer interesting i;
    real interesting state;
    boolean t,f;
```

7.2.6 Predefined Variables

Some variables in 20-sim have a predefined value. These variables do not have to be declared in the *variables* section of a model.

time

The variable *time* is equal to the simulated time. It can for example be used in functions such as:

```
u = sin(time*f*2*3.1415);
```

sampletime

The variable *sampletime* is equal to the sampletime. It is only meaningful when it is used in an equation model that is part of a discrete loop in your model. 20-sim will automatically detect, which loop the model belongs to and assign the proper value to the variable sampletime.

random

The variable *random* will yield a random variable uniformly distributed in the interval [-1,1]. E.g.:

```
u = random;
```

true

The variable *true* will return the boolean value true.

false

The variable *false* will return the boolean value false.

major

Some integration algorithms, will perform model calculations at one or more intermediate points before calculating the next output value. To prevent equations being calculated, at these intermediate steps, you can use the variable *major*. This variable will return the boolean *false* when calculations are performed for an intermediate step and will return *true* when calculations are performed for the output values.

Example

```
....
if major then
    sim_steps = sim_steps + 1;        // this equation is not evaluated at
intermediate points
end;
....
```

7.2.7 Global Parameters and Variables

Global parameters and variables are defined using the keyword ***global***. In equation models, this keyword is added after the definition of the data type in the Equation Editor:

```
parameters
      real global par1 = 100 {Hz};
      real global par2 ;
variables
      real global var1;
      real global var2;
..
..
```

Parameters can only be assigned a value once. The same goes for variables. Only one submodel must defined their value using an equation. If parameters or variables are assigned more than once, 20-sim will generate an error.

Example

The best way to describe the use of global parameters and variables is by an example model. You can find this model in the 20-sim demonstration library (*Demo\Tips And Tricks\Global Parameters and Variables.emx*). The model exists of three submodels. In the first submodel two global parameters (*f* and *A*) are assigned and one variable (*sinewave*). The variable is assigned a value in the equations section and connected with the output (*out1*).

```
submodel1

parameters
      real global f = 100 {Hz};
      real global A = 1 {none};
variables
      real global sinewave;
equations
      sinewave = A*sin(f*2*pi*time);
      out1 = sinewave;
```

In the second submodel, the two global parameters are use to connect them with two outputs (*out1* and *out2).*

```
submodel 2

parameters
      real global f {Hz};
      real global A {none};
variables
      real global sinewave;
equations
      out1 = f;
      out2 = A;
```

In the third submodel, the global variable is connected with an output (*out1).*

```
submodel 3

variables
    real global sinewave;
equations
    out1 = sinewave;
```

To prevent conflicts global parameters may be assigned a default value only once. The same goes for global variables: they may be assigned a value in the equation section only once.

7.3 Types

7.3.1 Data Types

For all parameters, variables, initial conditions etc. 20-sim supports the following data types:

1. Boolean
2. Integer
3. Real
4. String

The declaration of data types can be done in the *constants*, *parameters* and *variables* sections of the model. E.g.:

```
parameters
    real a = 1;
    integer B[2,2];
variables
    string c;
    boolean yesno;
```

Declaration of data types in equations is not allowed. E.g.:

```
real v = sin(t*a + b); // This is not allowed!
```

Hidden

To prevent users from inspecting variables, parameters or constants (e.g. for encrypted models) or decreasing the list of parameters and variables, the keyword *hidden* can be applied. This keyword should follow the data type. For example:

```
parameters
    real hidden a = 1;
    integer hidden B[2,2];
variables
    string hidden c;
    boolean hidden yesno;
```

Interesting

To allow for a quick selection of variables in the Variable Chooser, the keyword interesting can be applied for variables. This keyword should follow the data type. For example:

```
parameters
    real a = 1;
    integer B[2,2];
variables
    string interesting c;
    boolean interesting yesno;
```

Interesting variables are shown, even when all options are deselected in the Variable Chooser. This makes it possible to quickly select variables out of a large list.

7.3.2 Boolean

Either *true* (1) or *false* (0). Variables of this type should not be used to store any other values. Like C++, 20-sim evaluates non-zero values as true; only the value of zero is evaluated as false. Normally, the value of one is used to indicate true.

Examples

```
boolean a;
boolean b = true;
boolean c = false;
boolean D[2,2] = [true,true;false;false];
a = b and c;
```

Limitations

When used in standard functions, booleans are treated as reals with value 0 or 1. An equation like:

```
boolean a = true;
real c;
c = sin(a);
```

would yield c equal to 0.8415. Try to avoid these constructions, since they lead to confusions.

7.3.3 Integer

Signed four-byte integer. It can hold any value between -2147483648 and +2147483647. Integer is the recommended data type for the control of program loops (e.g. in for-next loops) and element numbering.

Example

```
parameters
    integer i = 1,j = 2;
    real B[2,2] = [1,2;3,4]
    real b;
equations
    b = B[i,j];
    for i := 1 to 10 do ....end;
```

Limitations

When used in standard functions, integers are treated as reals. An equation like:

```
parameters
    integer c;
equations
    c = 10*sin(time);
```

would yield c equal to 0.8415. Try to avoid these constructions, since they lead to confusions. Us the round function instead:

```
parameters
    integer c;
equations
    c = round(10*sin(time));
```

7.3.4 Real

Signed 64 bit floating point number. The maximum representable number is 1.7976931348623158e+308. The minimum positive value unequal to zero is 2.2250738585072014e-308. If reals exceed this range during simulation, the numerical values window will display the values 1.#INF (positive infinite) or -1.#INF (negative infinite).

Example

```
real a = 0.0;
real b = 1.04e-31,c = -3198.023e-64;
real B[2,2] = [1.2,22.4;3.0e3,-4.023e-6];
```

Limitations

In practice, during simulation, real should not exceed 1e-100 .. 1e100 to prevent instability in the integration algorithms.

7.3.5 String

A string is an array of characters. There is no limit to the amount of characters a String may contain. Simple string concatenation (e.g. concatenation that can be done during processing) is allowed. Strings must always be entered between single quotes ('). A single quote can be escaped with a second quote, so if a single quote should be part of the string, it needs to be typed twice.

Example

```
parameters
    string c = 'C:\data.txt';
    string b = 'This is a string with a '' single quote';
variables
    real d;
    string name;
equations
    if time > 10 then stopsimulation (b); end;
    d = table(c,1);
    name = c + ' // file';
    toMatlab (d, name)
```

Limitations

Strings may only be used as input for some special functions (e.g. table).

7.4 Functions

7.4.1 Functions

In 20-sim you can use all kind of functions to describe your equations. They are classified according to the list below.

1. Arithmetic
2. Discrete
3. Event
4. Expansion
5. Extern
6. Matrix
7. Port
8. Source
9. Trigonometric

7.4.2 Arithmetic

Absolute Value

Syntax

Y = abs(X);

Description

Returns the absolute value of X or the elements of X.

Examples

X	abs(X)
1	1
2.1	2.1
-2.6	2.6
-1e-102	1e-102
[1.1,2.7;-3.6,-4]	[1.1,2.7;3.6,4]

Limitations

- X and Y must have the same size.
- For scalars, the abs function can also be written as: Y = |X|;

Algebraic

Syntax

y = algebraic(x);

Description

y becomes equal to x. This function forces the integration algorithm to find a solution by using the algebraic loop solver. Normally when algebraic loops occur in a model, 20-sim will itself will break the algebraic loop using the algebraic loop solver internally. You can use this function to break an algebraic loop at a user-defined position.

Examples

```
u = sin(time);
a = algebraic(y);
y = K*(u - a);
```

Note

Be very careful with this function. In most cases the symbolic engine of 20-sim will find and exact solution which will lead to much faster simulation!

Ceil

Syntax

Y = ceil(X);

Description

Rounds X or the elements of X to the nearest integers greater than or equal to X.

Examples

X	ceil(X)
1	1
2.1	3
-2.6	-2
3.4	4
[1.1,-2.7;3.5,-4.1]	[2,-2;4,-4]

Limitations

X and Y must have the same size.

Constraint

Syntax

y = constraint(x);

Description

This function iteratively assigns a value to y such that x approaches zero within a given error margin. It only works in combination with the MBDF simulation method! The function is very useful for entering constraints in physical systems and inverse dynamics.

Examples

This example makes a velocity *vm* equal to a reference velocity *vs*:

```
vs = sin(time);              // reference velocity
v = vm - vs;                 // If v = 0 then the mass follows the reference velocity
F = constraint(v);       // Make F have a value that yields a zero velocity v
vm = (1/m)*int(F);       // mass
```

DDT

Syntax

y = ddt(x,init);

Description

Returns the derivative value of x with respect to the time. The initial value of y is equal to the value of init.

Method

20-sim will always try to rewrite equations in such a way that only integrals are used (integral form). This is done automatically and means all integration methods can be applied. Sometimes an integral form cannot be found. Then only the Backward-Differentiation Method can be used for simulation.

Examples

equation	integral form	integration
ddt(x,0) = u - k*x;	x = int(u - k*x);	All methods
x = ddt(sin(time),0);	not possible!	Only BDF

Limitations

When no integral form can be found, the use of the derivative function can introduce noisy signals or may even cause signals to exceed the data range.

Derivative

Syntax

y = derivative(x,base);

Description

Returns the derivative value of x with respect to base. E.g:

```
variables
    real x, y, z;
equations
x = sin (time);
y = derivative (x, time);
z = derivative (x^2, x);
```

Limitations

20-sim will use its symbolic engine to solve the derivative and rewrite it to a closed form. If no solution is found, and error message is generated.

Unit Delay

Syntax

y = dly(x,init);

Description

y is equal to the value of x, one simulation-time step delayed. The initial value of y is equal to init

Example

*x = sin(time*0.3);*
y = dly(x, 1.0);

Limitations

x,y and init must be scalars.

Exponential

Syntax

Y = exp(X);

Description

Returns the exponential function of X or the elements of X.

$y = e^x;$

Examples

X	exp(x)
1	*2.718282 (= e^1)*
2.1	*8.166170 (= $e^{2.1}$)*
-2.6	*0.074274 (= $e^{-2.6}$)*
[1,2.1;-2.6,1]	[2.718282,8.166170;0.074274,2.718282] *(= [e^1,$e^{2.1}$;$e^{-2.6}$,e^1])*

Limitations

X and Y must have the same size.

Exponential (Base 10)

Syntax

Y = exp10(X);

Description

Returns the exponential function (base 10) of X or the elements of X.

$y = 10^x;$

Examples

X	exp10(x)

1	*10 (= 10^{1})*
2.1	*125.89 (= $10^{2.1}$)*
-2.6	*0.00251 (= $10^{-2.6}$)*
[1,2.1;-2.6,1]	[10,125.89;0.00251,10] *(= [10^{1},$10^{2.1}$;$10^{-2.6}$,10^{1}])*

Limitations

X and Y must have the same size.

Exponential (Base 2)

Syntax

Y = exp2(X);

Description

Returns the exponential function (base 2) of X or the elements of X.

$y = 2^{x}$;

Examples

X	**exp2(x)**
1	*2 (= 2^{1})*
2.1	*4.8709 (= $2^{2.1}$)*
-2.6	*0.16494 (= $2^{-2.6}$)*
[1,2.1;-2.6,1]	[2,4.8709;0.16494,2] *(= [2^{1},$2^{2.1}$;$2^{-2.6}$,2^{1}])*

Limitations

X and Y must have the same size.

Floor

Syntax

Y = floor(X);

Description

Rounds X or the elements of X to the nearest integers less than or equal to X.

Examples

X	**floor(X)**

1	1
2.1	2
-2.6	-3
3.4	3
[1.1,-2.7;3.5,-4.1]	*[1,-3;3,-5]*

Limitations

X and Y must have the same size.

initialvalue

Syntax

Y = initialvalue(X);

Description

Returns the initial value of X or the elements of X. Value should be assigned to X first.

Examples

X	**initialvalue(X)**
1	1
sin(time)	0
[cos(time); sin(time)]	[1;0]

Limitations

X and Y must have the same size.

Integral

Syntax

y = int(x,init);

Description

Returns the integral of x with respect to the time. The initial value of y is equal to the value of init. This initial value is optional. If no value is entered, a default initial value of 0 is used.

Examples

```
y = int(x,0);
z = int( sin(time*w + p) - cos(time*x + z) , b);
```

Limited Integration

Syntax

y = limint(x,min,max,init);

Description

Returns the integral of x with respect to the time. The initial value of y is equal to the value of init. The output of this integral is limited between a maximum and minimum bound given by the parameters min and max.

This function forces the integration algorithm to find the exact point where the gets saturated. If the integral is in saturated condition and the input changes sign, the output wanders away from the bounds immediately.

The limitedIntegrator block is used for the prevention of wind-up in PI and PID controllers in control applications. It is also used in kinematics, electrical circuits, process control, and fluid dynamics.

Examples

```
x = 10*sin(time*10);
y = limint(x,-1,1,0);
```

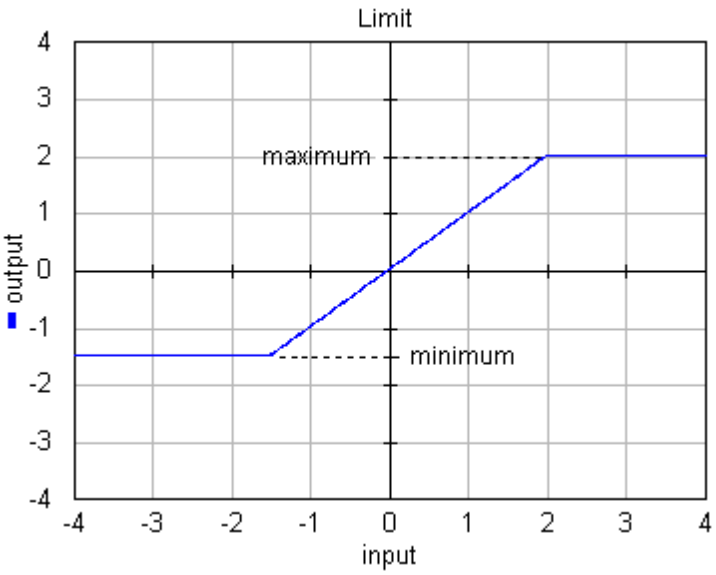
Limit

Syntax

y = limit(x,a,b);

Description

This function limits the signal x between a minimum of a and a maximum of b.



Examples

*x = 10*sin(time);*
y = limit(x,-5,5);

Limitations

a,b,x and y must be scalars. a must be smaller than b.

Natural Logarithm

Syntax

y = log(x);

Description

Returns the natural logarithm of X or the elements of X.

Examples

X	**log(X)**
1	*0*
2.1	*0.6931*
4	*1.3863*
[1,2.1;4,1]	*[0,0.6931;1.3863,0]*
-2.6	*not allowed!*

Limitations

X and Y must have the same size. X or the elements of X must be larger than zero.

Base 10 Logarithm

Syntax

Y = log10(X);

Description

Returns the base 10 logarithm of X or the elements of X.

Examples

X	log10(X)
1	*0*
50	*1.6990*
100	*2*
[1,50;100,1]	*[0,1.6990;2,0]*
-2.6	*not allowed!*

Limitations

X and Y must have the same size. X or the elements of X must be larger than zero.

Base 2 Logarithm

Syntax

Y = log2(X);

Description

Returns the base 2 logarithm of X or the elements of X.

Examples

X	log2(X)
1	*0*
2.1	*1.0704*
4	*2*
[1,2.1;4,1]	*[0,1.0704;2,0]*
-2.6	*not allowed!*

Limitations

X and Y must have the same size. X or the elements of X must be larger than zero.

Resetable Integration

Syntax

y = resint(x,newoutp,reset,init);

Description

Returns the integral of x with respect to the time. The initial value of *y* is equal to the value of *init*. The output of this integral is reset to the value of *newoutp* when the Boolean argument *reset* is *TRUE*.

Examples

```
x = 1;
reset = event(sin(time));
y = resint(x,0,reset,0);
```

or

```
x = 1;
reset = if y > 10 then true else false end;
y = resint(x,0,reset,0);
```

Round

Syntax

Y = round(X);

Description

Rounds X or the elements of X to the nearest integer.

Examples

X	round(X)
1	*1*
2.1	*2*
-2.6	*-3*
3.7	*4*
[1.1,-2.7;3.5,-4.1]	*[1,-3;4,-4]*

Limitations

X and Y must have the same size.

Sign

Syntax

Y = sign(X);

Description

Returns the sign of X or the elements of X.

X = < 0 : Y = -1
X = 0: Y = 0
X > 0: Y = 1

Examples

X	sign(X)
1	*1*
2.1	*1*
-2.6	*-1*
0	*0*
[1,2.1;-2.6,0]	*[1,1;-1,0]*

Limitations

X and Y must have the same size.

Square

Syntax

Y = sqr(X);

Description

Returns the square of X or the elements of X.

Examples

X	sqr(X)
1	*1 (= 1^2)*
2.1	*4.41 (= 2.1^2)*
-2.6	*6.76 (= -2.6^2)*
0	*0 (= 0^2)*
[1,2.1;-2.6,0]	[1,4.41;6.76,0] *(= [1^2,2.1^2;-2.6^2,0^2])*

Limitations

x and y must be scalars.

Square Root

Syntax

X = sqrt(Y);

Description

Returns the square root of X or the elements of X.

Examples

X	**sqrt(X)**
1	*1*
2.1	*1.44913*
4	*2*
[1,2.1;4,1]	*[1,1.44913;2,1]*
-1	*not allowed!*

Limitations

X and Y must have the same size. X or the elements of X must be larger than or equal to zero.

Time Delay

Syntax

y = tdelay(x,delaytime);

Description

Returns the (continuous) signal *x* delayed for an absolute time (given by *delaytime*). The initial value of *y* for *time = 0* to *delaytime* is equal to zero. This block is intended to model a continuous delay in a continuous simulation. Use the Unit Delay block to model a digital delay.

Examples

```
x = 10*sin(time*10);
y = tdelay(x,2);
```

Limitations

x,*y* and *delaytime* must be scalars. delaytime must be a **positive** constant value.

Trunc

Syntax

Y = trunc(X);

Description

Rounds X or the elements of X towards zero (i.e. the function trunc removes the fraction).

Examples

X	trunc(X)
1	*1*
2.1	*2*
-2.6	*-2*
3.4	*3*
[1.1,-2.7;3.5,-4.1]	*[1,-2;3,-4]*

Limitations

X and Y must have the same size.

7.4.3 Discrete

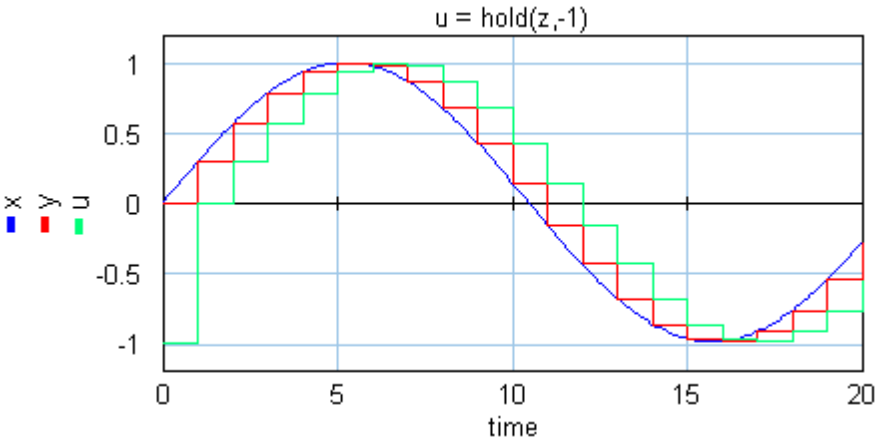
Hold

Syntax

y = hold(x,init);

Description

The hold function implements a zero order hold function operating at a specified sampling rate. It provides a mechanism for creating a continuous output signal y out of a discrete input signal x. The initial value of y is equal to the value of init.



Examples

```
x = sin(time*0.3);
y = sample(x);
z = previous(y);
u = hold(z,-1);
```

Limitations

x and y must be scalars. x be a discrete signal. y is a continuous signal. 20-sim will automatically detect the existence of discrete signals. Each chain of discrete signals will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose *Properties*, *Simulation* and *Discrete System*).

Next

Syntax

```
y = next(x,init);
```

Description

The next function allows you to make constructs like:

$$x(k+1) = x(k) - 0.1 + u(k)$$

where k is the kth sample. Using the *next* function, this can be entered in 20-sim as:

```
next(x,0) = x - 0.1 + u;
```

Limitations

x and y will become discrete signals. 20-sim will automatically detect the existence of discrete signals. Each chain of discrete signals will be assigned a specific sample time. You can set this sample time to any desired value in the Simulator (choose *Properties*, *Simulation* and *Discrete System*).

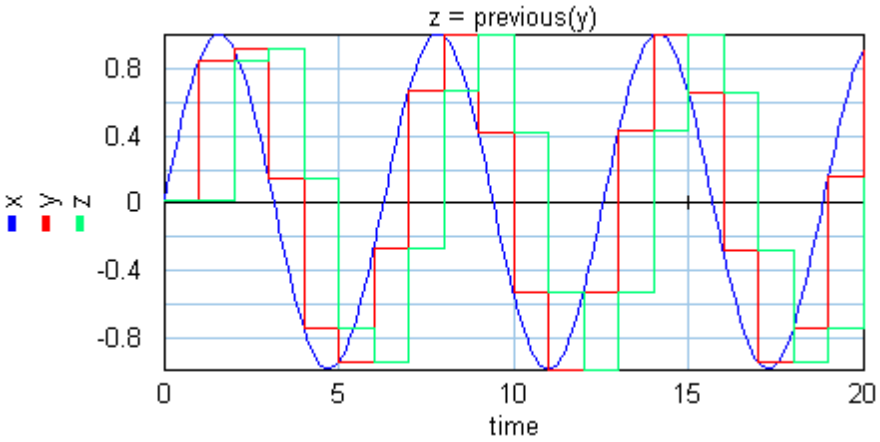
Previous

Syntax

y = previous(x,init);

Description

The previous function delays the input signal x and holds it for one sample interval. It is equivalent to the z-1 discrete time operator.



Examples

*x = sin(time*0.3);*
y = sample(x);
z = previous(y);
u = hold(z);

Limitations

x and *y* will become discrete signals. 20-sim will automatically detect the existence of discrete signals. Each chain of discrete signals will be assigned a specific sample time. You can set this sample time to any desired value in the Simulator (choose *Properties*, *Simulation* and *Discrete System*).

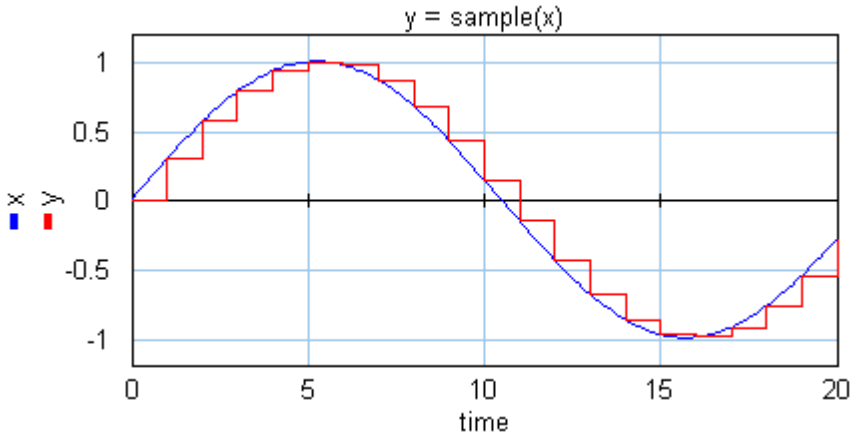
Sample

Syntax

y = sample(x,init);

Description

The sample function implements a sample and hold function operating at a specified sampling rate. It provides a mechanism for discretizing a continuous input signal. The initial value of y is equal to the value of init. This initial value is optional. If no value is entered, a default initial value of 0 is used.



Examples

```
x = sin(time*0.3);
y = sample(x);
z = previous(y);
u = hold(z);
```

Limitations

- x and y must be scalars. x represents a continuous signal and y represents a discrete signal.
- y will become a discrete signals. 20-sim will automatically detect the existence of discrete signals. Each chain of discrete signals will be assigned a specific sample time. You can set this sample time to any desired value in the Simulator (choose *Properties*, *Simulation* and *Discrete System*).

7.4.4 Event

Event

Syntax

```
y = event(x);
```

Description

y becomes true when *x* is zero. This function forces the integration algorithm to find the exact point where the input crosses zero. This is called a state event.

Examples

```
x = sin(time*omega +phi);
y = event(x);
```

Limitations

y must be a boolean, *x* must be an expression yielding a scalar.

Eventdown

Syntax

y = eventdown(x);

Description

y becomes true when *x* is zero. This function forces the integration algorithm to find the exact point where the input crosses zero from a positive to a negative value. This is called a state event.

Examples

```
x = sin(time*omega +phi);
y = eventdown(x);
```

Limitations

y must be a boolean, *x* must be an expression yielding a scalar.

Eventup

Syntax

y = event(x);

Description

y becomes true when x is zero. This function forces the integration algorithm to find the exact point where the input crosses zero from a negative to a positive value. This is called a state event.

Examples

```
x = sin(time*omega +phi);
y = eventup(x);
```

Limitations

y must be a boolean, *x* must be an expression yielding a scalar.

Frequencyevent

Syntax

y = frequencyevent(p,o);

Description

y becomes true every time when *p* [s] have passed. The function starts after an offset of *o* [s]. The offset parameter is optional. This function is a time event function.

Examples

```
parameters
    real period = 0.1 {s};
    real offset = 0.005 {s};
variables
    boolean y,z;
equations
    y = frequencyevent(1);                  // y = true every second
    z = frequencyevent(period,offset);      // z = true at 0.005 s, 0.105 s, 0.205 s
etc.
```

Limitations

y must be a boolean, *p* and *o* must be a parameter or a constant.

Timeevent

Syntax

y = timeevent(x);

Description

y becomes true when time is equal to *x*. This functions forces the integration algorithm to do a calculation when time *y* becomes true. This is a time event function.

Examples

The example code below forces the integration algorithm, to do a calculation at the start of the step:

```
x = step(start_time);
y = timeevent(start_time);
```

Limitations

y must be a boolean, *x* must be an expression yielding a scalar.

7.4.5 Expansion

Equal

Syntax

equal([x1,x2, ...,xn]);

Description

This function is used to make all elements of a matrix equal. It returns the equations x2 = x1, x3 = x1, .. , xn = x1. During processing the equations will be automatically reshaped into a proper causal form. This function is created for use in library models that use a matrix with unknown size which results from the collect function.

Examples

Suppose we have m bonds connected to a submodel, collected in a port p. We could then use the function:

collect(p.e);

During processing this equation will be expanded as:

[p.e1;p.e2;...;p.n];

If we use the equal function:

equal(collect(p.e));

this results in:

equal([p.e1;p.e2;...;p.n]);

which will be expanded in:

p.e2 = p.e1;
p.e3 = p.e1;
..
p.en = p.e1

Limitations

This function is designed for a special class of models and should be used by experienced users only!

Mul

Syntax

y = mul([x1,x2, ...,xn]);

Description

This function is used for multiplication of variables of which one may be unknown. It returns the equation y = x1 * x2 * .. * xn. During processing this equation will be automatically reshaped into a causal form.

This function is created for use in library models that have unknown inputs such as *MultiplyDivide* model. Try to avoid the use of this function!

Examples

equations	**after reshaping (processing)**
x1 = 1;	*x1 = 1;*
x2 = 2;	*x2 = 2;*
1 = mul([x1,x2,x3]);	*x3 = (1/x1)/x2;*
y = sin(time);	*y = sin(time);*
x2 = 2;	*x2 = 2;*
y = mul([x1,x2]);	*x1 = y/x2;*

Limitations

This function is designed for a special class of models and should be used by experienced users only! y, x1, x2, ..,xn must be scalars. n may be 1 or higher1

Sum

Syntax

y = sum([x1,x2, ...,xn]);

Description

This function is used for summation of variables of which one may be unknown. It returns the equation y = x1 + x2 + .. xn. During processing this equation will be automatically reshaped into a causal form.

This function is created for use in library models that have unknown inputs such as *PlusMinus* model. Try to avoid the use of this function!

Examples

equations	**after reshaping (processing)**
x1 = 1;	*x1 = 1;*
x2 = 2;	*x2 = 2;*
0 = sum(x1,x2,x3);	*x3 = x1 + x2;*

```
y = sin(time);
x2 = 2;
y = sum(x1,x2);
```

```
y = sin(time);
x2 = 2;
x1 = y - x2;
```

Limitations

- This function is designed for a special class of models and should be used by experienced users only! y, x1, x2, ..,xn must be scalars. n may be 1 or higher.
- Use the function msum if you want to use a standard summation of multiple variables.

7.4.6 External

data

Syntax

y = data(filename,column);

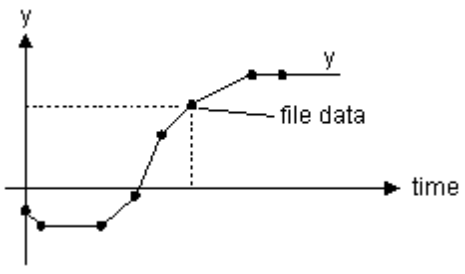
Description

This function generates an output by linear interpolation of data read from file. The data on file is stored in columns. The first column contains the time values (t) and the second column contains the corresponding output values (y).

- The time data of the first column needs to be monotonically increasing.
- Discontinuities are allowed, by providing the same time point twice in the table.
- Values outside of the range, are computed by linear extrapolation of the last two points.

The first argument (*filename*) of this function must be a parameter of the type string and is used as a reference to a filename. This reference can be specified using the complete path or relative to the model directory. The second argument (*column*) of this function must be a parameter of the type integer denoting a whole number. It is used to denote the column number.

table	(example):
0	-0.5
0.5	-1
2.5	-1
3.5	0
4.5	1
5.5	1.75
7	2.5



8 2.5

The input file must be an ASCII (text) file or a Comma Separated Values (.csv) file and should consist at least two columns of data. The first column (number 0) should always contain the time values. The other columns (number 1, 2, 3, etc.) should contain the corresponding data values. The parameter *column* is used to specify which column is used for as output data.

The various values must be separated by a space, a tab or a comma. Each new set of time and data values must start on a new line. No comment or other text may be part of the file. The filename of the input file can be specified using the complete path (e.g. `c:\data\data.tbl`). When no path is given, the file is assumed to be in the same directory as your model.

The first row may contain names for the columns. Names should be indicated by quotation marks (").

Example

```
parameters
    string filename = 'data.txt';
    integer col = 2;
variables
    real y;
equations
    y = data(filename,col);
```

Example data.txt file with header:

```
"time","x","y"
0.0, 1, 2
0.1, 5, 10
0.2, 6, 11
```

Limitations

y must be a scalar. *from_file* must be string parameter. *x* must be an integer parameter.

DLL

Syntax

Y = dll(filename,functionname,X);

Description

Given a function (*functionname*) of a dll (*filename*), the dll function returns the output values (*X*) for a given input (*Y*).

Users can write their own source code using a native compiler such as Visual C++ or Borland C++. With these compilers it is possible to create DLL's with user defined functions that have an input- output relation which can be embedded in simulation code.

Examples

```
parameters
    string filename = 'example.dll';
    string function = 'myFunction';
variables
    real x[2],y[2];
equations
    x = [ramp(1);ramp(2)];
    y = dll(filename,function,x);
```

Limitations

filename and *functionname* must be string parameters. Note that the size of Y and X (scalars or matrices) must correspond with the size that is expected for the given dll-function.

Search Order

20-sim uses the following search order for the dll-file:

1. The bin directory of 20-sim (usually C:\Program Files\20-sim 4.5\bin).
2. The current directory.
3. The Windows system directory.
4. The Windows directory.
5. The directories that are listed in the PATH environment variable.

The simplest place to put the dll-file is the bin directory, but it is always possible to give a complete path for the dll filename.

Code generation notice

20-sim allows for code generation of the 20-sim model. In case of a DLL call, 20-sim cannot generate the complete code for a DLL call since it only knows the DLL file name, function name and arguments and not the internal DLL code.

Therefore, in the generated code, the *dll(dll_name, function_name, in)* will be generated as:

*%dll_name%_%function_name%(double *inarr, int inputs, double *outarr, int outputs, int major)*

Where *%dll_name%* and *%function_name%* are replaced with the actual DLL and function name. You have to add the C-code implementation of this function yourself to the generated code before you can compile it.

DLLdynamic

Syntax

Y = dlldynamic(filename,functionname,X);

Description

Given a function (functionname) of a dll (filename), the dll function returns the output values (X) for a given input (Y).

Users can write their own source code using a native compiler such as Visual C++ or Borland C++. With these compilers it is possible to create DLL's with user defined functions that have an input- output relation which can be embedded in simulation code.

Examples

```
parameters
    string dllName = 'demoDynamicDll.dll';
    string functionName = 'SFunctionCalculate';
variables
    real x[2],y[2];
equations
    x = [ramp(1);ramp(2)];
    y = dlldynamic(dllName,functionName,x);
```

Limitations

The *filename* and *functionname* must be string parameters. Note that the size of Y and X (scalars or matrices) must correspond with the size that is expected for the given dll-function.

Search Order

20-sim uses the following search order for the dll-file:

1. The bin directory of 20-sim (usually C:\Program Files\20-sim 4.5\bin).
2. The current directory.
3. The Windows system directory.
4. The Windows directory.
5. The directories that are listed in the PATH environment variable.

The simplest place to put the dlldynamic-file is the bin directory, but it is always possible to give a complete path for the dll filename.

Writing Static DLL's

Almost every compiler that can build Windows applications, can also build dll-files containing dll-functions. For two compilers an example will be given how to build the **static dll's** (i.e. dll functions that do not contain internal model states) in combination with 20-sim. The two compilers are: Visual C++ and Borland C++ .

When the simulator has to call the user defined dll for the first time the dll is linked and the appropriate function is called. At the end of the simulation run the dll will be disconnected from the simulator. In general the dll-function called by the simulator has the following syntax in 20-sim:

Y = dll(filename,functionname,X);

For example (equation model):

```
parameters
    string dll_name = 'example.dll';
    string function_name = 'myFunction';
variables
    real in[2],out[2];
equations
    in = [ramp(1),ramp(2)];
    out = dll(dll_name, function_name, in);
```

The name of the function is the actual name which was given to the corresponding string parameter. The return value of the function determines whether the function was successful or not. A return value of 0 means success, every other value error. When a nonzero value is returned the simulation stops after finishing its current simulation step. The parameters given to the function correspond directly to the 20-sim parameters: double pointers for the inputs and outputs point to an array of doubles.

Function arguments

The user-function in the dll must have certain arguments. The function prototype is like this:

```
  int myFunction(double *inarr, int inputs, double *outarr, int outputs,
int major)
```

where

- *inarr*: pointer to an input array of doubles. The size of this array is given by the second argument.
- *inputs*: size of the input array of doubles.
- *outarr*: pointer to an output array of doubles. The size of this array is given by the fourth argument.
- *outputs*: size of the output array of doubles.
- *major*: boolean which is 1 if the integration method is performing a major integration step, and 0 in the other cases. For example Runge-Kutta 4 method only has one in four model evaluations a major step.

Initializing the DLL-file

When the dll is linked to the simulator it is often useful to perform some initializations. There are several ways to perform these initializations.

Method 1

When the simulator has attached the dll-file it automatically searches for a function with the name ‘int Initialize()’. If this function is found it is called. The return value is checked for success, 1 means success, 0 means error. At the end of the simulation run just before the dll-file is detached the simulator searches for a function called ‘int Terminate()’. In this function the necessary termination action can be performed like cleaning up allocated memory.

At the start of every run the simulator also searches a function with the name 'int InitializeRun()'. If it is found it is called the same way the Initialize function was called. At the end the same happens with the function 'int TerminateRun()'. This is very useful in multiple run simulations, in this case the dll-file is linked only once with the simulator, and only once the Initialize and Terminate functions are called. But for every subsequent run the InitializeRun and TerminateRun functions are called so every run can be initialized and terminated gracefully. In this case in the Terminate function collection of multiple run data could be collected and saved for example.

So suppose you have a multiple run with 2 runs, the following functions (if they exist) will be called in the dll-file in this order:

```
Initialize()
InitializeRun()
TerminateRun()
Terminate()
```

Method 2

Using the DLLEntryPoint function. This function is automatically called when the library is linked or detached. Both in Visual C++ and in Borland C++ this function has the same syntax. However when using visual C++ as the compiler, our simulator which is build with Visual C++ does not seem to be able to call this function. So the framework as given below only works with a Borland C++ dll.

```
// Every dll has an entry point LibMain || DllEntryPoint
// and an exit point WEP (windows exit point).
BOOL WINAPI DllEntryPoint(HINSTANCE hinstDll, DWORD fdwRreason, LPVOID
plvReserved)
{
  if (fdwRreason == DLL_PROCESS_ATTACH)
  {
    return 1; // Indicate that the dll was linked successfully.
  }
  if (fdwRreason == DLL_PROCESS_DETACH)
  {
    return 1; // Indicate that the dll was detached successfully.
  }
  return 0;
}
```

Method 3

When using an object oriented language it is possible to have a global variable which is an instance of a class. When linking the dll-file to the simulator the constructor function of this global variable will be called automatically. In this constructor the initialization can be performed. When the dll-file is detached the destructor of the global variable is called. Some care should be taken with this method of terminating. E.g. if the user is asked for a filename using a FileDialog this sometimes causes the complete application to crash. The reason for this is that some of the dll-files contents was already destructed and that some functionality may not work anymore resulting in an application error. The solution for this problem is tot use method 1 of initialization and destruction.

Frame work for a Visual C++ dll-file implementation.

```
#include <windows.h>
#define DllExport __declspec( dllexport )
extern "C"
{
  DllExport int myFunction(double *inarr, int inputs, double *outarr,
int outputs, int major)
  {
    ...       // function body
    return 0; // return successful
  }
  DllExport int Initialize()
  {
    ...       // do some initializations here.
    return 0; // Indicate that the dll was initialized successfully.
  }
  DllExport int Terminate()
  {
    ...       // do some cleaning here
    return 0; // Indicate that the dll was terminated successfully.
  }
}
```

Frame work for a Borland C++ dll-file implementation.

```
#include <windows.h>
extern "C"
{
  int _export myFunction(double *inarr, int inputs, double *outarr, int
outputs, int major)
  {
    ...       // function body
    return 0; // return successful
  }
  int _export Initialize()
  {
    ...       // do some initializations here.
    return 0; // Indicate that the dll was initialized successfully.
  }
  int _export Terminate()
  {
    ...       // do some cleaning here
```

```
    return 0; // Indicate that the dll was terminated successfully.
  }
}

// Every dll has an entry point LibMain || DllEntryPoint
// and an exit point WEP.
BOOL WINAPI DllEntryPoint(HINSTANCE hinstDll, DWORD fdwRreason, LPVOID
plvReserved)
{
  if (fdwRreason == DLL_PROCESS_ATTACH)
  {
    ...       // do some initializations here.
    return 1; // Indicate that the dll was initialized successfully.
  }
  if (fdwRreason == DLL_PROCESS_DETACH)
  {
    ...       // do some cleaning here
    return 1; // Indicate that the dll was initialized successfully.
  }
  return 0;
}
```

Note

Make sure that Borland C++ does not generate underscores in front of the exported function names! This can probably be found in the compiler settings.

Working example

Here is a complete working example of how to use a dll-function from within the simulator. This code will compile with Visual C++ or gcc. See the Borland C++ framework how to write this code in Borland C++.

```
/* Example C++ DLL for 20-sim */
#include <windows.h>
#include <math.h>
#include <fstream>
#include <stdio.h>

#define DLLEXPORT __declspec( dllexport )

using namespace std;

ofstream outputStream;
#ifdef _MSC_VER
#define snprintf _snprintf
#endif

extern "C"
{
  char g_lasterrormessage[255]; /* Used to store a DLL error message
```

```
                                    for transfer to 20-sim */
  const char* g_modelpath;

  /**
  * This is an example of a function that can be called in your 20-sim
  * model using the dll() Sidops call
  *
  * @param inarr   This double array contains all inputs that will be
  *                send from 20-sim to the other side
  * @param inputs  20-sim tells the dll how many elements inarr[]
  *                contains.
  * @param outarr  Result from this dll function that will be returned
  *                to 20-sim
  * @param outputs 20-sim tells the dll how many elements it expects
  *                (and allocated) in outarr.
  * @param major   1=major integration step, 0=minor step (e.g. an
  *                intermediate integration method step in Runge Kutta
  *                4)
  **/
  DLLEXPORT int myFunction(double *inarr, int inputs, double *outarr,
int outputs, int major)
  {
    // Check the sizes of our input and output arrays
    if (inputs != outputs)
    {
      snprintf(g_lasterrormessage, 255, "%s: expects that the number of
inputs is equal to the number of outputs.", __FUNCTION__);
      return 0; // Failure
    }

    for (int i = 0; i < inputs; i++)
    {
      outarr[i] = cos(inarr[i]);
      outputStream << inarr[i] << " " << outarr[i] << " ";
    }
    outputStream << endl;
    return 1; // Success
  }

  /****** Initialization and cleanup ******/

  /* Note 1:
   *   The Initialize(), InitializeRun(), Terminate() and TerminateRun()
   *   functions are optional.
   *   Implement them when you need to initialize something before the
   *   actual experiment is started and to cleanup/reset your DLL
   *   functionality for a next run.
   * Note 2:
   *   When these functions are implemented, the "continue run"
   *   functionality in 20-sim is disabled.
```

```
 */

/**
 * Initialize() [optional]
 *
 * This function is called by the 20-sim simulator BEFORE starting the
 * simulation experiment (and only once in a multiple run experiment)
 * to initialize the dll properly.
 */
DLLEXPORT int Initialize()
{
  outputStream.open("c:\\temp\\data.log");
  return 0; // Indicate that the dll was initialized successfully.
}
/**
 * InitializeRun() [optional]
 *
 * This function is called by the 20-sim simulator BEFORE starting the
 * simulation experiment (and only once in a multiple run experiment)
 * to initialize the dll properly.
 */
DLLEXPORT int InitializeRun()
{
  /* Clear lasterrormessage before every run. */
  snprintf(g_lasterrormessage, 255, "");

  return 0; /* Indicate that the dll was initialized successfully. */
}

/**
 * TerminateRun() [optional]
 *
 * This function is called by 20-sim after each finished run
 */
DLLEXPORT int TerminateRun()
{
  /* Cleanup / reset your DLL here for the next run
   * (e.g. in a multiple run experiment)
   */
  return 0; // Indicate that the dll was terminated successfully.
}

/**
 * Terminate() [optional]
 *
 * This function is called by 20-sim on a DLL unload
 */
DLLEXPORT int Terminate()
{
  outputStream.close();
```

```
    return 0; // Indicate that the DLL was terminated successfully.
  }

  /**
   * LastErrorMessage() [optional]
   * Used by 20-sim to fetch a string with the last error that occurred
   * within the DLL
   * @return A char pointer to a string indicating the error message
   */
  DLLEXPORT char* LastErrorMessage()
  {
    return g_lasterrormessage;
  }

  /**
  * RegisterModelPath() [optional]
  * This function is called by 20-sim before Initialize() to learn the
  * DLL where the model is located. It can be used e.g. to find data
  * files stored in the same folder as the model.
  *
  * @param modelPath char pointer to the model directory
  * @return 0 if parameter is set successfully, 1 if not successful.
  */
  DLLEXPORT int RegisterModelPath(const char * modelPath)
  {
    g_modelpath = modelPath;
  }
}
```

Usage within 20-sim

Suppose the dll has been created as "example.dll". With the following code this model can be tested:

```
parameters
    string dll_name = 'example.dll';
    string function_name = 'myFunction';
variables
    real in[2],out[2];
equations
    in = [ramp(1),ramp(2)];
    out = dll(dll_name, function_name, in);
```

Note that the general function "dll" is used. The arguments of this function, dll_name and function_name, are parameters which are used to denote the dll that should be used and the function of that dll that should be called. You can load this model from the Demonstration Models Library:

1. Open the *Editor*.
2. From the demo library open the model *DllFunction.emx* (choose *File* and *Open*)

3. Start the simulator (*Model* menu and *Start Simulator*).
4. Start a simulation run (select *Run* from the *Simulation* menu).

Code generation notice

20-sim allows for code generation of the 20-sim model. In case of a DLL call 20-sim cannot generate the complete code for a DLL call since it only knows the DLL file name, function name and arguments and not the internal DLL code.

Therefore, in the generated code, the dll(dll_name, function_name, in) will be generated as:

```
%dll_name%_%function_name % (double *inarr, int inputs, double *outarr, int outpu
```

Where *%dll_name%* and *%function_name%* are replaced with the actual DLL and function name. You have to add the C-code implementation of this function yourself to the generated code before you can compile it.

Writing Dynamic DLL's

Almost every compiler that can build Windows applications, can also build dll-files containing dll-functions. For the Visual C++ compiler an example will be given how to build the dynamic dll's (i.e. dll functions that may contain internal model states) in combination with 20-sim.

The functions in the DLL are called in a specific sequence. For communication with the functions and the Simulator Kernel information is passed in a structure. This structure looks like this:

```
struct SimulatorSFunctionStruct
{
    double versionNumber;
    int nrInputs;
    int nrOutputs;
    int nrIndepStates;
    int nrDepStates;
    int nrAlgLoops;
    double simulationStartTime;
    double simulationFinishTime;
    double simulationCurrentTime;
    BOOL major;
    BOOL initialOutputCalculation;
};
```

1. Initialize

At the start of the simulation the function

int Initialize()

is called. When this function is not present it is not called. Any initializations of data structures can be performed here.

2. SFunctionInit

During initialization of the Simulator Kernel the function

*int SFunctionInit(SimulatorSFunctionStruct *s)*

is called. Return value is 0 means error. every other value succes Argument is a pointer to the simstructure. On initialization the following fields should be filed in:

nrIndepStates
nrDepStates
nrAlgLoop

The following fields already have valid values:

simulationStartTime	giving the start time of the simulation
simulationFinishTime	giving the finish time of the simulation
simulationCurrentTime	giving the current time (actually the start time at the moment of initialization)

3. Initial values for the states:

*int SFunctionGetInitialStates(double *initialIndepStates,*
*double *initialDepRates,*
*double *initialAlgloopIn,*
*SimulatorSFunctionStruct *simStruct);*

Return value is 0 means error. every other value succes. The initial value for the independent states, dependent rates and algebraic loop variables can be specified by the DLL in this function. It is just called before the initial output calculation function in step 3. If all the initial values are zero, nothing has to be specified.

4. Initial Output Calculation

It is possible that the DLL-function can give an initial output. A separate function is called so that the DLL can calculate it's initial output values. The boolean initialOutputCalculation in the simulatorSFunction structure is used. Just the sFunction is called. as in point 5.

5. SFunction calling:

Here all the fields of the SimulatorSFunctionStruct are input for the function. The inputArray, stateArray, outputArray and rateArray are always given as arguments of the function. Dependent on the number of dependent states and algebraic loop variables more arguments can be given as shown in the functions below (sFunctionName is the name defined by the parameter name specified by the user):

Return value is 0 means error. every other value succes

case: no dependent states, no algebraic loop variables

*int sFunctionName(double *inputArray,*
*double *stateArray,*
*double *outputArray,*
*double *rateArray,*
*SimulatorSFunctionStruct *simStruct);*

case: dependent states, no algebraic loop variables
*int sFunctionName(double *inputArray,*
*double *stateArray,*
*double *dependentRateArray,*
*double *outputArray,*
*double *rateArray,*
*double *dependentStateArray,*
*SimulatorSFunctionStruct *simStruct);*

case: no dependent states, algebraic loop variables
*int sFunctionName(double *inputArray,*
*double *stateArray,*
*double *algLoopInArray,*
*double *outputArray,*
*double *rateArray,*
*double *algLoopOutrray,*
*SimulatorSFunctionStruct *simStruct);*

case: dependent states, algebraic loop variables
*int sFunctionName(double *inputArray,*
*double *stateArray,*
*double *dependentRateArray,*
*double *algLoopInArray,*
*double *outputArray,*
*double *rateArray,*
*double *dependentStateArray,*
*double *algLoopOutrray,*
*SimulatorSFunctionStruct *simStruct);*

the boolean major in the SimulatorSFunctionStruct determines whether the evaluation of the model is done at the time output is generated (major == TRUE) or that the model is evaluated because of determining model characteristics. For example Runge-Kutta4 integration method uses three minor steps before taking a major step where output is generated. Higher order methods can have different number of minor steps before a major step is taken.

6. Termination
At the end of the simulation the function:

int Terminate()

is called. When this function is not present it is not called. Any terminations of data structures can be performed here.

Frame work for a Visual C++ dll-file implementation.

```
#include <windows.h>
#define DllExport __declspec( dllexport )
extern "C"
{
DllExport int dllfunction(double *inarr, int inputs, double *outarr, int outputs, int major)
{
... // function body
return 0; // return succesfull
}
DllExport int Initialize()
{
... // do some initializations here.
return 0; // Indicate that the dll was initialized successfully.
}
DllExport int Terminate()
{
... // do some cleaning here
return 0; // Indicate that the dll was terminated successfully.
}
}
```

Frame work for a Borland C++ dll-file implementation.

```
#include <windows.h>
extern "C"
{
int _export dllfunction(double *inarr, int inputs, double *outarr, int outputs, int major)
{
... // function body
return 0; // return succesfull
}
int _export Initialize()
{
... // do some initializations here.
return 0; // Indicate that the dll was initialized successfully.
}
int _export Terminate()
```

```
{
... // do some cleaning here
return 0; // Indicate that the dll was terminated successfully.
}

}
// Every dll has an entry point LibMain || DllEntryPoint
// and an exit point WEP.
BOOL WINAPI DllEntryPoint( HINSTANCE hinstDll,DWORD fdwRreason,LPVOID
plvReserved)
{
if( fdwRreason == DLL_PROCESS_ATTACH )
{
... // do some initializations here.
return 1; // Indicate that the dll was initialized successfully.
}
if( fdwRreason == DLL_PROCESS_DETACH )
{
... // do some cleaning here
return 1; // Indicate that the dll was initialized successfully.
}
return 0;
}
```

Working example

Here is a complete working example of how to use a dll-function from within the simulator. This code will compile with Visual C++. See the Borland C++ framework how to write this code in Borland C++.

```
#include <windows.h>
#include "SimulatorSFunctionStruct.h"

/
**************************************************************************
*******
 * in this source file we are gonna describe a linear system which is defined by
 * the following transfer function description:

  34
Y = ---------------- * U
  s^2 + 6s + 34

or A, B, C, D system:
```

```
A = [ 0, -3.4;
  10, -6];
B = [ -3.4;
  0];
C = [0, -1];
D = 0
which has two poles on (-3 + 5i) and (-3 -5i)

  steady state = 1

   ************************************************************************
*******/

#define DllExport __declspec( dllexport )

extern "C"
{

// called at begin of the simulation run
DllExport int Initialize()
{
// you can perform your own initialization here.

// success
return 0;
}

// called at end of the simulation run
DllExport int Terminate()
{
// do some cleaning here

// success
return 0;
}

DllExport int SFunctionInit (SimulatorSFunctionStruct *s)
{
// tell our caller what kind of dll we are
s->nrIndepStates = 2;
s->nrDepStates = 0;
s->nrAlgLoops = 0;

// dubious information, since 20-sim itself does not check and need this info
```

```
s->nrInputs = 1;
s->nrOutputs = 1;

// return 1, which means TRUE
return 1;
}

DllExport int SFunctionGetInitialStates (double *x0, double *xd0, double *xa0,
SimulatorSFunctionStruct *s)
{
// fill in the x0 array here. Since we specified no Dependent states, and No algebraic
loop variables
// the xd0 and xa0 may not be used.

// initial value is zero.
x0[0] = 0;
x0[1] = 0;

// return 1, which means TRUE
return 1;
}

DllExport int SFunctionCalculate(double *u, double *x, double *y, double *dx,
SimulatorSFunctionStruct *s)
{
// we could check the SimulatorSFunctionStruct here if we are in an initialization state
and/or we are
// in a major integration step.
#if 0
if( s->initialialOutputCalculation )
; // do something

// possibly do some explicit action when we are in a major step.
if( s->major == TRUE )
; // do something
#endif
dx[0] = -3.4 * x[1] -3.4 * u[0];
dx[1] = 10 * x[0] -6 * x[1];


y[0] = -x[1];

// return 1, which means TRUE
return 1;
```

```
}

}// extern "C"
BOOL APIENTRY DllMain( HANDLE hModule,
  DWORD ul_reason_for_call,
  LPVOID lpReserved
  )
{
  return TRUE;
}
```

Use within 20-sim

Suppose the dll has been created as "demoDynamicDll.dll". With the following code this model can be tested:

```
parameters
        string dllName = 'demoDynamicDll.dll';
        string functionName = 'SFunctionCalculate';
equations
        output = dlldynamic (dllName, functionName, input);
```

Note that the general function "dlldynamic" is used. The arguments of this function, dllName and functionName, are parameters which are used to denote the dll that should be used and the function of that dll that should be called. You can load this model from the Demonstration Models Library:

1. Open the Editor.
2. From the demo library open the model *DllFunction.emx* (choose *File* and *Open*)
3. Start the simulator (*Model* menu and *Start Simulator*).
4. Start a simulation run (select *Run* from the *Simulation* menu).

Table

Syntax

y = table(file_name,x);

Description

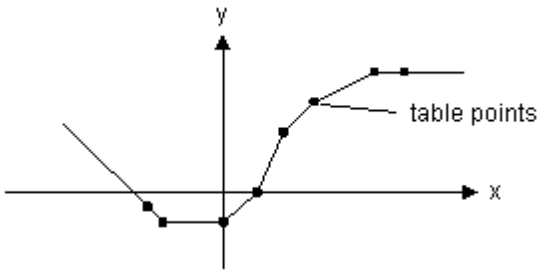
This model uses a one-dimensional table with data points to calculate the output *y = f(x)* as a function of the input x. The output y is calculated using linear interpolation between the table data points.

- The input data x of the first column needs to be monotonically increasing.
- Discontinuities are allowed, by providing the same input point twice in the table.
- Values outside of the table range, are computed by linear extrapolation of the last two points.

The first argument *(file_name)* of this function must be a parameter of the type string and is used as a reference to a filename. This reference can be specified using the complete path or relative to the model directory. The second argument *(x)* should be a real variable.

table (example):

-2.5	-0.5
-2	-1
0	-1
1	0
2	2
3	3.0
5	4
6	4.0



A table must be stored as an ASCII (text) file or a Comma Separated Values (.csv) file and should consist two columns of data. The first column consists of the *x*-values and the second column of the corresponding *y*-values. Each line of the file may only contain one *x*- and one corresponding *y*-value. The two values must be separated by a space or a tab. Each new set of x- and y-values must start on a new line. No comment or other text may be part of the table-file.

Example

```
parameters
    string filename = 'data.txt';
variables
    real y;
    real x;
equations
    x = ramp(1);
    y = table(filename,x);
```

Example data.txt format:

```
-2.5, -0.5
-2.0, -1.0
0.0, -1.0
1.0, 0.0
2.0, 2.0
3.0, 3.0
5.0, 4.0
6.0, 4.0
```

Limitations

x and y must be scalars. from_file must be string parameter.

7.4.7 Matrix

Adjoint

Syntax

A = Adjoint(H);
A = Adjoint(R,P);

Description

Returns the adjoint matrix [6,6] of a homogeneous matrix [4,4]:

$$H = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix} \rightarrow$$

$$Adjoint(H) = \begin{bmatrix} R & 0 \\ skew(P)*R & R \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 & 0 & 0 \\ r_{21} & r_{22} & r_{23} & 0 & 0 & 0 \\ r_{31} & r_{32} & r_{33} & 0 & 0 & 0 \\ -p_3 \cdot r_{21} + p_2 \cdot r_{31} & -p_3 \cdot r_{22} + p_2 \cdot r_{32} & -p_3 \cdot r_{23} + p_2 \cdot r_{33} & r_{11} & r_{12} & r_{13} \\ p_3 \cdot r_{11} - p_1 \cdot r_{31} & p_3 \cdot r_{12} - p_1 \cdot r_{32} & p_3 \cdot r_{13} - p_1 \cdot r_{33} & r_{21} & r_{22} & r_{23} \\ -p_2 \cdot r_{11} + p_1 \cdot r_{21} & -p_2 \cdot r_{12} + p_1 \cdot r_{22} & -p_2 \cdot r_{13} + p_1 \cdot r_{23} & r_{31} & r_{32} & r_{33} \end{bmatrix}$$

or returns the adjoint matrix [6,6] of a rotation matrix R [3,3] and position vector P [3]:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \quad P = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \rightarrow$$

$$Adjoint(R,p) = \begin{bmatrix} R & 0 \\ skew(P)*R & R \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 & 0 & 0 \\ r_{21} & r_{22} & r_{23} & 0 & 0 & 0 \\ r_{31} & r_{32} & r_{33} & 0 & 0 & 0 \\ -p_3 \cdot r_{21} + p_2 \cdot r_{31} & -p_3 \cdot r_{22} + p_2 \cdot r_{32} & -p_3 \cdot r_{23} + p_2 \cdot r_{33} & r_{11} & r_{12} & r_{13} \\ p_3 \cdot r_{11} - p_1 \cdot r_{31} & p_3 \cdot r_{12} - p_1 \cdot r_{32} & p_3 \cdot r_{13} - p_1 \cdot r_{33} & r_{21} & r_{22} & r_{23} \\ -p_2 \cdot r_{11} + p_1 \cdot r_{21} & -p_2 \cdot r_{12} + p_1 \cdot r_{22} & -p_2 \cdot r_{13} + p_1 \cdot r_{23} & r_{31} & r_{32} & r_{33} \end{bmatrix}$$

The Adjoint function is useful for dynamics described by twists and wrenches.

Examples

```
 R = [cos(alpha),-sin(alpha),0 ; sin(alpha),cos(alpha),0 ; 0,0,0];
P = [p1 ; p2 ; p3];
A = Adjoint(R,P);
```

Limitations

H must be of size [4,4], R must always be of size[3,3] , P must have the size [3,1] and A must have size [6,6].

adjoint

Syntax

```
Y = adjoint(T);
Y = adjoint(W,V);
```

Description

Returns the adjoint matrix [6,6] of a twist or wrench vector *T* [6,1]:

$$T = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} W \\ V \end{bmatrix} \rightarrow$$

$$adjoint(T) = \begin{bmatrix} skew(W) & 0 \\ skew(V) & skew(W) \end{bmatrix} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & 0 & 0 & 0 \\ \omega_3 & 0 & -\omega_1 & 0 & 0 & 0 \\ -\omega_2 & \omega_1 & 0 & 0 & 0 & 0 \\ 0 & -v_3 & v_2 & 0 & -\omega_3 & \omega_2 \\ v_3 & 0 & v_1 & \omega_3 & 0 & -\omega_1 \\ -v_2 & v_1 & 0 & -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

or returns the adjoint matrix [6,6] of an angular velocity vector *W* [3,1] and a velocity vector *V* [3,1]:

$$W = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \quad , \quad V = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \rightarrow$$

$$adjoint(W,V) = \begin{bmatrix} skew(W) & 0 \\ skew(V) & skew(W) \end{bmatrix} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & 0 & 0 & 0 \\ \omega_3 & 0 & -\omega_1 & 0 & 0 & 0 \\ -\omega_2 & \omega_1 & 0 & 0 & 0 & 0 \\ 0 & -v_3 & v_2 & 0 & -\omega_3 & \omega_2 \\ v_3 & 0 & v_1 & \omega_3 & 0 & -\omega_1 \\ -v_2 & v_1 & 0 & -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

The adjoint function is useful for dynamics described by twists and wrenches.

Examples

```
W = [w1 ; w2 ; w3];
V = [v1 ; v2 ; v3];
Y = adjoint(W,V);
```

Limitations

Y must be of size [6,6], *W* must always be of size[3,1] , *V* must have the size [3,1] and *T* must have size [6,1].

Antisym

Syntax

```
y = antisym(A);
```

Description

Returns an anti-symmetric matrix. This function is equal to:

```
y = (A - transpose(A))/2;
```

Examples

```
A = [1,2;3,4];
Y = antisym(A);
```

Limitations

A must be a square matrix. Y and A must be of the same size.

Columns

Syntax

y = columns(A);

Description

Returns the number of columns of A.

Examples

```
A = [1,2;3,4;5,6];
y = columns(A);
```

Limitations

y must be a scalar.

Cross

Syntax

Y = cross(A,B);

Description

This functions returns the vector cross product of the vectors A and B. That is Y = A × B or:

```
 Y[1] = A[2]*B[3] - A[3]*B[2]
Y[2] = A[3]*B[1] - A[1]*B[3]
Y[3] = A[1]*B[2] - A[2]*B[1]
```

Examples

```
parameters
      real A[3] = [1;2;3];
      real B[3] = [4;5;6];
variables
      real Y[3];
equations
Y = Cross(A,B); // Y = [-3;6;-3]
```

Limitations

Y, A and B must be column vectors of size 3.

Determinant

Syntax

y = det(A);
or
Y = |A|;

Description

Returns the determinant of the square matrix A.

Examples

```
A = [1,2;3,ramp(1)];
y = det(A);
```

Limitations

A must be a square matrix, y must be a scalar.

Diag

Syntax

```
Y = diag(x);
```

Description

Fills the diagonal elements of the matrix Y with the elements of the columvector x.

Example

```
variables
    real x[3];
    real Y[3,3];
equations
    x = [1;2;3];
    Y = diag(x);
```

Limitations

Y must be a matrix of size [n,n]. x must be a column vector of size n.

Eye

Syntax

```
Y = eye(n);
```

Description

Returns the the n-by-n identity matrix.

Example

```
Y = eye(3);
```

Limitations

Y must be a matrix of size [n,n]. n must be an integer value.

Homogeneous

Syntax

```
H = homogeneous(R,P);
```

Description

Returns the homogenous matrix [4,4] of a rotation matrix R[3,3] and position vector P[3]:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad , \quad P = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \quad \rightarrow \quad H = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_1 \\ r_{21} & r_{22} & r_{23} & p_2 \\ r_{31} & r_{32} & r_{33} & p_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The homogeneous matrix gives a full coordinate transformation from one reference frame to another. It can be used for 3D dynamics.

Examples

R = [cos(alpha),-sin(alpha),0 ; sin(alpha),cos(alpha),0 ; ; 0,0,0];
P = [p1 ; p2 ; p3];
H = homogeneous(R,P);

Limitations

R must always be of size[3,3] , P must have the size [3,1] and H must have size [4,4].

Inner

Syntax

y = inner(A,B);

Description

Returns the scalar product of the column vectors A and B. A and B must be vectors of the same length. Inner(A,B) is the same as transpose(A)*B.

Example

A[3] = [1;2;3];
B[3,1] = [4;5;6];
y = inner(A,B);

Limitations

A and B must be vectors of the same size [n,1] or [n]. y must be a scalar.

Inverse

Syntax

Y = inverse(A);

Description

Returns the inverse of square non-singular matrix A.

Examples

A = [1,0;0,1+ramp(1)];
Y = inverse(A);

Limitations

A must be a non-singular square matrix. Y must have the same size as A.

InverseH

Syntax

Y = inverseH(H);
Y = inverseH(R,P);

Description

Returns the inverse of a homogeneous matrix H [4,4]. The homogeneous matrix can be entered directly or by a rotation matrix R[3,3] and position vector P [3,1].

This function uses the special nature of a homogeneous matrix. I.e the inverse can be computed directly instead of the numerical approach of the standard inverse function.

Examples

```
H = [cos(alpha),-sin(alpha),0,p1 ; sin(alpha),cos(alpha),0,p2 ; 0,0,0,p3 ; 0,0,0,1];
Y = inverseH(H);
 or
R = [cos(alpha),-sin(alpha),0 ; sin(alpha),cos(alpha),0 ; 0,0,0];
P = [p1 ; p2 ; p3];
Y = inverseH(R,P);
```

Limitations

H must be a homogeneous matrix of size [4,4], R must always be of size[3,3] , P must have the size [3,1] and A must have size [6,6].

Linsolve

Syntax

x = linsolve(A,b [,method]);

Description

This function solves the equation

*A*x = b;*

Where A is a square matrix of arbitrary size n and x and b are vectors of size n.
Returns the inverse of square non-singular matrix A.
The last argument is a string that specifies the desired method to use.

method	description
lu	LU decomposition (default method)
qr	QR factorization
cholesky	Cholesky factorization
gmres	Generalized Minimum Residual

The gmres method allows further steering of the method by specifying method parameters:

method parameter	description
tol	Set the desired tolerance to use
maxiter	Set the maximum number of iterations to use

ortho	Set the method of Gramm-Schmidt orthogonalization: 1 modified Gramm-Schmidt 2 iterative Gramm-Schmidt 3 classical Gramm-Schmidt 4 iterative classical Gramm-Schmidt

The lu, qr and gmres methods allow a non-square matrix A to be entered yielding a pseudoinverse.

Examples

```
A = [1,2;0,1+ramp(1)];
b = [1;5];
x = linsolve(A,b);
x2 = linsolve (A, b, 'qr');
x3 = linsolve (A, b, 'gmres tol=1e-8 ortho=4');
```

Limitations

A must be a non-singular square matrix.

Note

The following equations

```
A*x = b;
x = inverse(A)*b;
x = (A^-1)*b;
```

all lead in the calculation of the inverse of A and will give the same result as

```
x = linsolve(A,b);
```

For the inverse calculation, Cramers rule is used. This is a method which is fast for small matrix sizes. The linsolve function with the gmres method is superior to Cramers rule for larger matrix sizes.

Max

Syntax

```
y = max(A);
```

Description

Returns the value of the largest matrix element of A.

Examples

```
A = [1,2;3,ramp(1)];
y = max(A);
```

Limitations

y must be a scalar.

Min

Syntax

y = min(A);

Description

Returns the value of the smallest matrix element of A.

Examples

A = [1,2;3,ramp(1)];
y = min(A);

Limitations

y must be a scalar.

Msum

Syntax

y = msum(A);

Description

Returns the sum of the matrix elements of A.

Examples

A = [1,2;3,4;5,6];
y = msum(A);

Limitations

y must be a scalar.

Norm

Syntax

y = norm(A);
or
Y = ||A||;

Description

Returns the square root of the sum of the squared matrix elements of A:

$$y = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} A[i,j]^2}$$

Examples

A = [1,2;3,ramp(1)];
y = norm(A);

Limitations

y must be a scalar.

Norminf

Syntax

y = norminf(A);

Description

Returns the largest row sum of the absolute values of the matrix A:

$$\max\left\{\sum_{j=1}^{n}\left|a_{ij}\right|, 1 \le i \le m\right\}$$

Examples

A = [1,2;3,ramp(1)];
y = norminf(A);

Limitations

y must be a scalar.

Rows

Syntax

y = rows(A);

Description

Returns the number of rows of A.

Examples

A = [1,2;3,4;5,6];
y = rows(A);

Limitations

y must be a scalar.

Skew

Syntax

Y = skew(X);

Description

Returns the vector product:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}$$

Examples

$$X = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 0 & -3 & 2 \\ 3 & 0 & -1 \\ -2 & 1 & 0 \end{bmatrix}$$

Limitations

X must be a vector of size n. Y must be a matrix of size [n,n].

Sym

Syntax

y = sym(A);

Description

Returns a symmetric matrix. This function is equal to:
y = (A + transpose(A))/2;

Examples

A = [1,2;3,4];
Y = sym(A);

Limitations

A must be a square matrix. Y and A must be of the same size.

Tilde

Syntax

H = tilde(T);
H = tilde(W,V);

Description

Returns the twist matrix [4,4] of a twist or wrench vector T [6,1]:

$$T = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad , \quad \rightarrow \quad tilde(T) = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & v_1 \\ \omega_3 & 0 & -\omega_1 & v_2 \\ -\omega_2 & \omega_1 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Or returns the twist matrix [4,4] of an angular velocity vector W [3,1] and a velocity vector V [3,1]:

$$W = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \quad , \quad V = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad , \quad \rightarrow \quad tilde(W,V) = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & v_1 \\ \omega_3 & 0 & -\omega_1 & v_2 \\ -\omega_2 & \omega_1 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The tilde function is useful for dynamics described by twists and wrenches.

Examples

```
W = [w1 ; w2 ; w3];
V = [v1 ; v2 ; v3];
H = tilde(W,V);
```

Limitations

H must be of size [4,4], W must always be of size[3,1] , V must have the size [3,1] and T must have size [6,1].

Trace

Syntax

```
y = trace(A);
```

Description

Returns the sum of the diagonal elements of the matrix A.

Examples

```
A = [1,2,3;4,ramp(1),6];
y = trace(A);
```

Limitations

y must be a scalar.

Transpose

Syntax

Y = transpose(A);

Description

Returns the transpose of the matrix A.

Examples

```
A = [1,2;3,ramp(1)];
Y = transpose(A);
```

Limitations

With the A matrix of size[n,m] , Y must have the size [m,n].

7.4.8 Port

Collect

Syntax

```
y = collect(p.e);
y = collect(p.f);
y = collect(m);
```

(with p.e and p.f powerportvariables and m a signal)

Description

Some models in 20-sim can have an unknown amount of signals or bonds connected. They are treated as an array with unknown length.

For example the PlusMinus model can have n signals that must be added and m signals that must be subtracted. 20-sim treats them as an array plus and an array minus with:

plus = [plus1;plus2;..;plusn] and minus = [minus1;minus2;..;minusm]

To assign an array of unknown length, the *collect* function is used. It creates an array with port variables or signals. This function is created for use in library models that have an unknown amount of bonds or signals connected. **Try to avoid the use of this function!**

Examples

Suppose we have m bonds connected to a submodel, collected in a port p. We could then use the equation:

Y = collect(p.e);

During processing, this equation will be rewritten as:

Y = [p.e1;p.e2;...;p.n];

Suppose we have a submodel with n input signals connected. We could then write the equation:

y = sum(collect(input));

During processing, this equation will be rewritten as:

y = sum([output1;output2;...;outputn]);

which is of course equal to:

y = output1 + output2 + .. + outputn;

Limitations

This function is designed for a special class of models and should be used by experienced users only! The function is only valid when used in a submodel.

Direct

Syntax

y = direct(p.e);
y = direct(p.f);

(with p.e and p.f powerportvariables)

Description

This function is equal to the collect function, but utilizes the direction of the bonds connected to the submodel to sign the powerportvariables. This function is created specially for the bond graph submodels *OneJunction.emx* and *ZeroJunction.emx*. Try to avoid the use of this function!

Examples

Suppose we have 4 bonds connected to a 0-junction, collected in a port p. p1 and p4 are pointing towards the junction and p2 and p3 are pointing from the 0-junction. We could then use the equation:

Y = direct(p.f);

During processing, this equation will be rewritten as:

Y = [p.f1;-p.f2;-p.f3,p.f4];

Which can of course be used in combination with the sum function:

sum(Y) = 0;

To get the 0-junction equation:

p.f1 - p.f2 - p.f3 + p.f4 = 0;

Limitations

This function is designed for a special class of models and should be used by experienced users only! The function is only valid when used in a submodel.

First

Syntax

y = first(p.e);
y = first(p.f);
y = first(m);

(with p.e and p.f powerportvariables and m a signal)

Description

Some models in 20-sim can have an unknown amount of signals or bonds connected. They are treated as an array with unknown length. For example the PlusMinus model can have n signals that must be added and m signals that must be subtracted. 20-sim treats them as an array *plus* and an array *minus* with:

plus = [plus1;plus2;..;plusn] and minus = [minus1;minus2;..;minusm]

The function *first* returns the value of the first element of an array of unknown length. This function is created for use in library models that have an unknown amount of bonds or signals connected. **Try to avoid the use of this function!**

Examples

Suppose we have m bonds connected to a submodel, collected in a port p. We could then use the equation:

y = first(p.e);

During processing, this equation will be rewritten as:

y = p.e1;

Limitations

This function is designed for a special class of models and should be used by experienced users only! The function is only valid when used in a submodel.

7.4.9 Source

Gaussian Noise

Syntax

y = gauss(x,s);

Description

Returns gaussian noise with a variance x and seed s. The seed parameter is optional. When omitted, the default value (0) is used.

Examples

```
x = 20;
y = gauss(x);
z = gauss(x,450);
```

Limitations

x and y must be scalars.The seed s must be a number in the region <0,65535>.

Ramp

Syntax

y = ramp(x);

Description

Returns a ramp signal with start time x:
time < x: y = 0
time >= x: y is time - x

Examples

```
x = 20, amplitude = 10;
y = amplitude*ramp(x);
```

Limitations

x and y must be scalars.

Random Noise

Syntax

y = ran(x,s);

Description

Returns uniformly distributed noise in the interval [-x,x] with seed s. The seed parameter is optional. When omitted, the default value (0) is used.

Examples

x = 20;
y = ran(x);
z = ran(x,450);

Limitations

x and y must be scalars. The seed s must be a number in the region <0,65000>.

Random Seed

20-sim generates a sequence of random numbers for each simulation differently depending upon the value of the seed parameter. The random noise function and gaussian noise function are affected by this. The default value of the seed is 0. The maximum value is 65535.

default value (0)

When the seed value is 0 (default value), 20-sim generates a new sequence of random numbers for each simulation and for each new random function. E.g. when two random functions with default seed value (0) are used in one model, they will generate different sequences of random numbers during a simulation.

other values (>0)

When the seed value is chosen larger than zero, 20-sim generates the same sequence of random numbers for each simulation. Moreover 20-sim will generate the same sequence of random numbers for each random function that uses the same seed parameter (>0). E.g. when two random functions with seed value 50, are used in one model, they will generate the same sequence of random numbers during a simulation.

Step

Syntax

y = step(x);

Description

Returns a step signal with start time x:
time < x: y = 0
time >= x: y = 1

Examples

x = 20, amplitude = 10;
*y = amplitude*step(x);*

Limitations

x and y must be scalars.

7.4.10 Trigonometric

Arcsin

Syntax

Y = arcsin(X);

Description

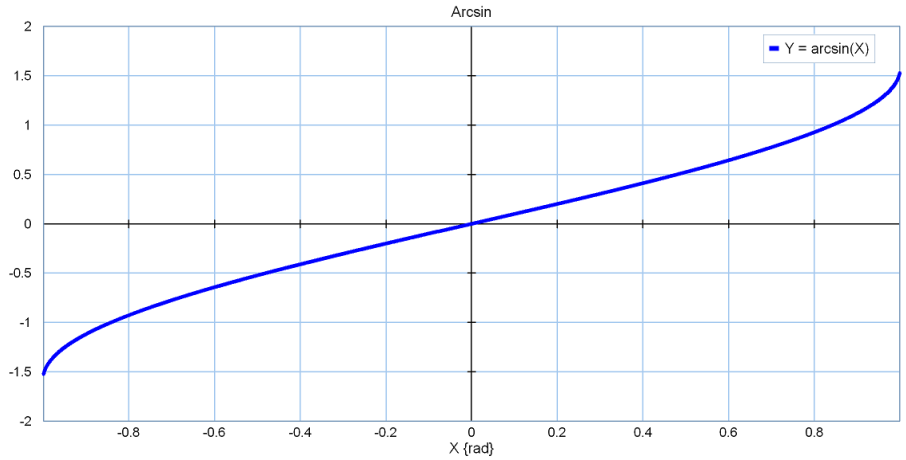
Returns the arcsine of the elements of X.

Examples

```
b = cos(time);
a = arcsin(b);
X = [0.5*cos(time);0.75*cos(time)];
Y = arcsin(X);
```

Limitations

Y and X must have the same size. The elements of X must be in the range [-1 ,1].



Arccos

Syntax

Y = arccos(X);

Description

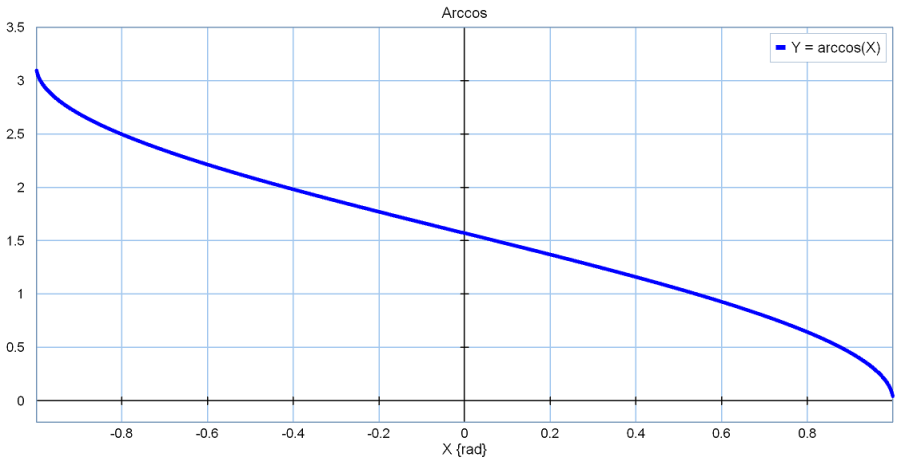
Returns the arccosine of the elements of X.

Examples

```
b = sin(time);
a = arccos(b);
X = [0.5*sin(time);0.75*sin(time)];
Y = arccos(X);
```

Limitations

Y and X must have the same size. The elements of X must be in the range [-1 ,1].



Arccosh

Syntax

```
Y = arccosh(X);
```

Description

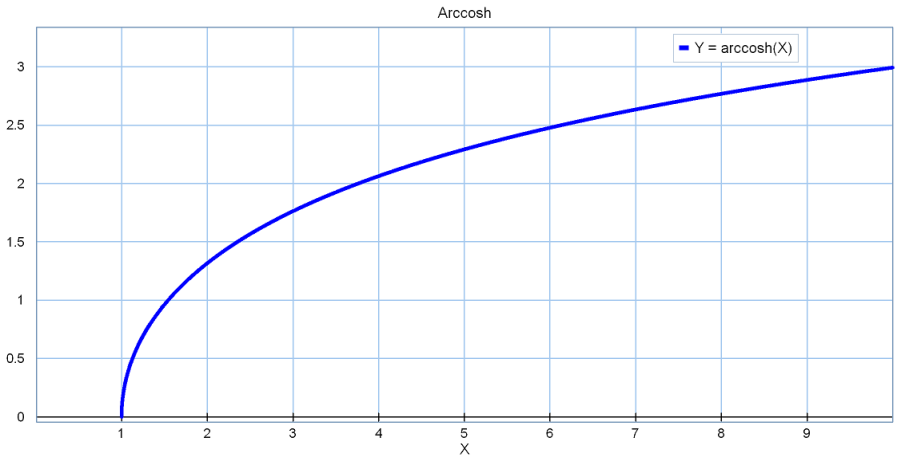
Returns the hyperbolic arccosine of the elements of X.

Examples

```
y = arccosh(time + 1.0);
```

Limitations

Y and X must have the same size. The elements of X must be 1 or bigger.



Arcsinh

Syntax

Y = arcsinh(X);

Description

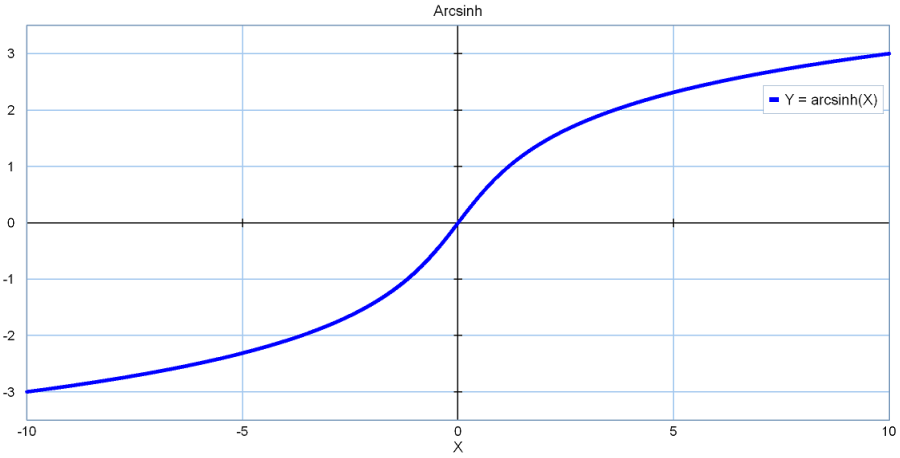
Returns the hyperbolic arcsine of the elements of X.

Examples

```
a = arcsinh(time);
X = [0.5*cos(time);0.75*cos(time)];
Y = arcsinh(X);
```

Limitations

Y and X must have the same size.



Arctan

Syntax

Y = arctan(X);

Description

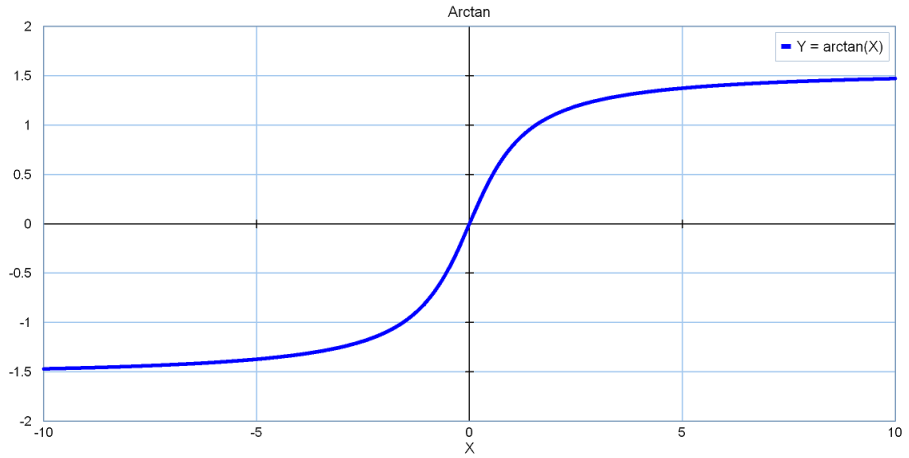
Returns the arctangent of the elements of X.

Examples

a = arctan(time);
*X = [ramp(1);-0.5*ramp(9)];*
Y = arctan(X);

Limitations

Y and X must have the same size.



Arctanh

Syntax

Y = arctanh(X);

Description

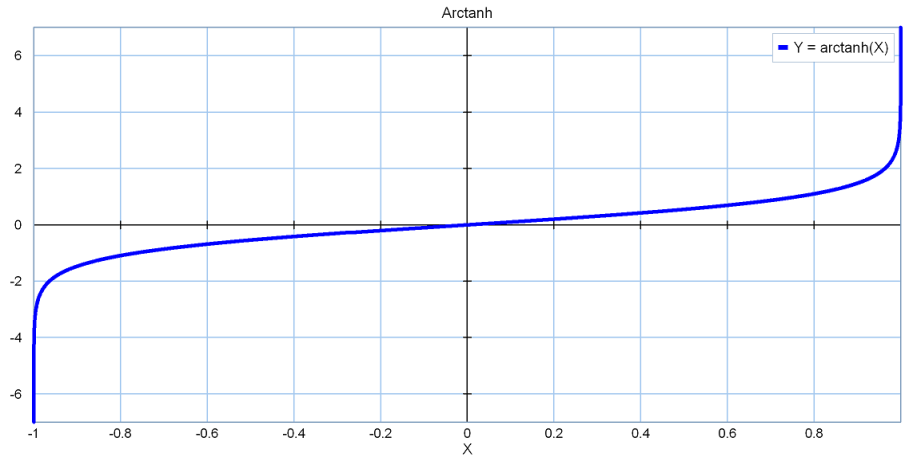
Returns the hyperbolic arctangent of the elements of X.

Examples

b = sin(time);
a = arctan(b);
*X = [sin(time);-0.5*cos(time)];*
Y = arctanh(X);

Limitations

Y and X must have the same size. The elements of X must be in the range [-1 ,1].



Atan2

Syntax

R = atan2(Y,X);

Description

This is the four quadrant arctangent of the elements of Y and X. Unlike arctangent function, atan2 does distinguish between diametrically opposite directions.

Examples

X, Y	arctan(Y/X);	atan2(Y, X);
1,1	π/4	π/4
-1,-1	π/4	−3π/4
0,1	-	π/2
0,0	-	set to 0 instead of undefined.

```
y = sin(time);
x = cos(time);
r = atan2(y,x);

X = [ramp(0);ramp(1)];
Y = [1;1];
R = atan2(Y,X);
```

Limitations

r, x and y must have the same size. It produces results in the range (−π, π]. Atan2(0, 0) is defined as 0 instead of undefined.

Cos

Syntax

Y = cos(X);

Description

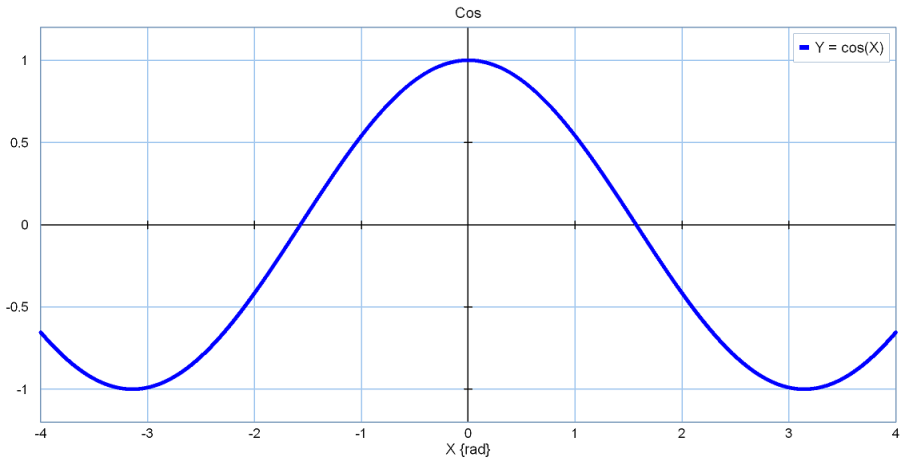
Returns the cosine of the elements of X.

Examples

```
b = ramp(2.5);
a = cos(b);
X = [ramp(0);ramp(1)];
Y = cos(X);
```

Limitations

Y and X must have the same size.



Cosh

Syntax

Y = cosh(X);

Description

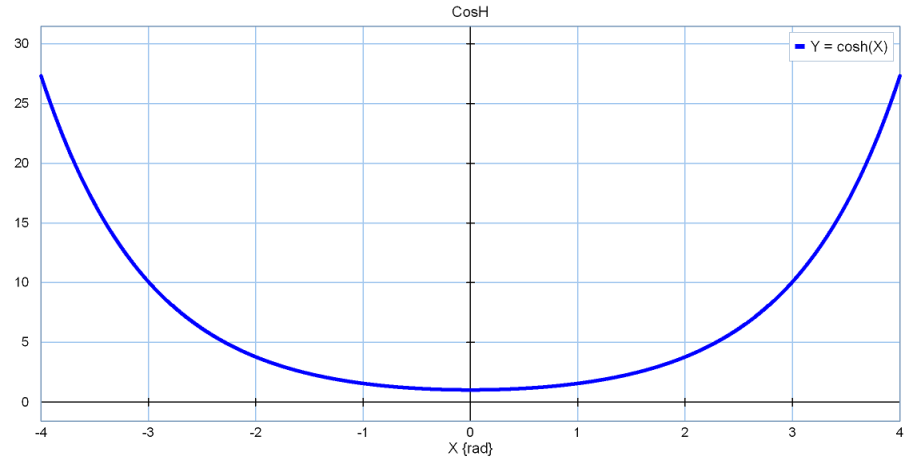
Returns the hyperbolic cosine of the elements of X.

Examples

```
b = ramp(2.5);
a = cosh(b);
X = [ramp(0);ramp(1)];
Y = cosh(X);
```

Limitations

Y and X must have the same size.



Sin

Syntax

Y = sin(X);

Description

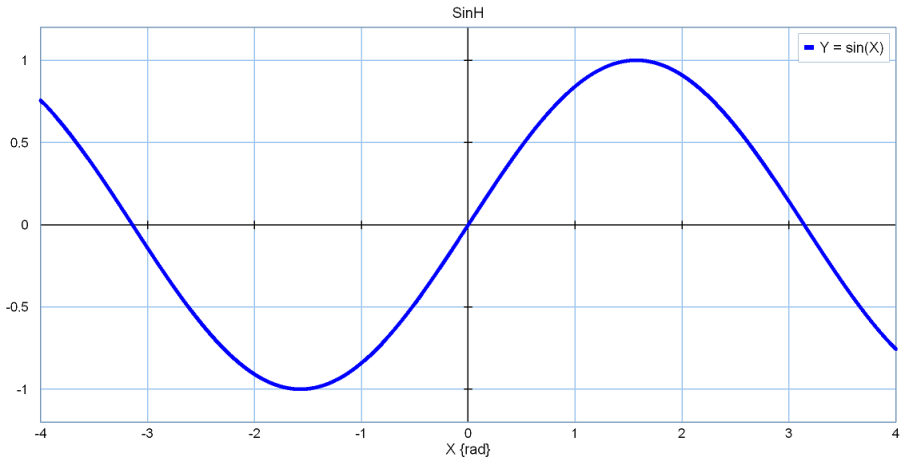
Returns the sine of the elements of X.

Examples

```
b = ramp(2.5);
a = sin(b);
X = [ramp(0);ramp(1)];
Y = sin(X);
```

Limitations

Y and X must have the same size.



Sinh

Syntax

Y = sinh(X);

Description

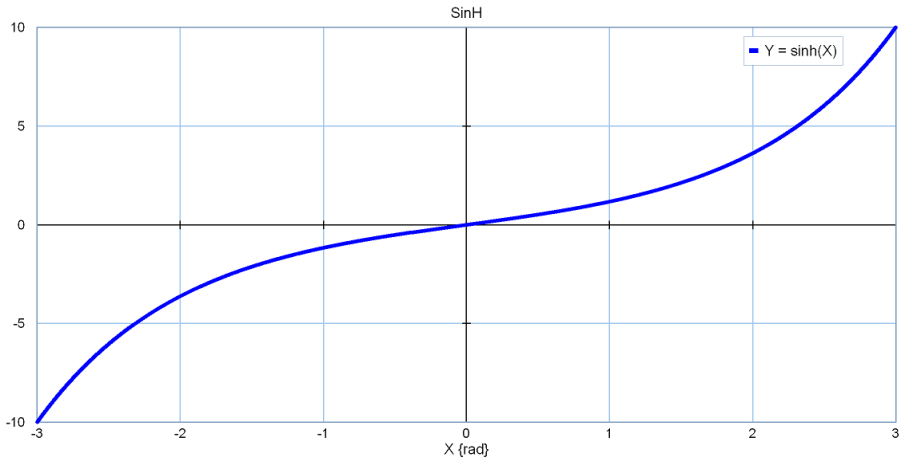
Returns the hyperbolic sine of the elements of X.

Examples

```
b = ramp(2.5);
a = sinh(b);
X = [ramp(0);ramp(1)];
Y = sinh(X);
```

Limitations

Y and X must have the same size.



Tanh

Syntax

Y = tanh(X);

Description

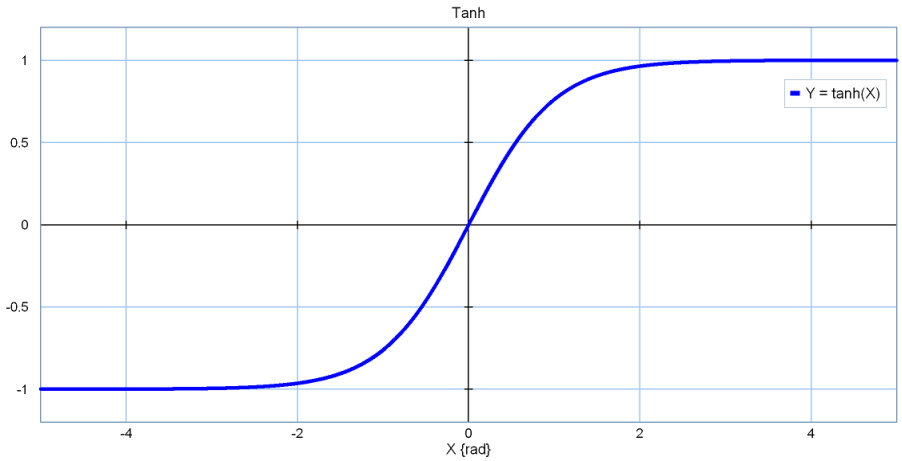
Returns the hyperbolic tangent of the elements of X.

Examples

```
b = ramp(2.5);
a = tan(b);
X = [ramp(0);ramp(1)];
Y = tanh(X);
```

Limitations

Y and X must have the same size.



Tan

Syntax

Y = tan(X);

Description

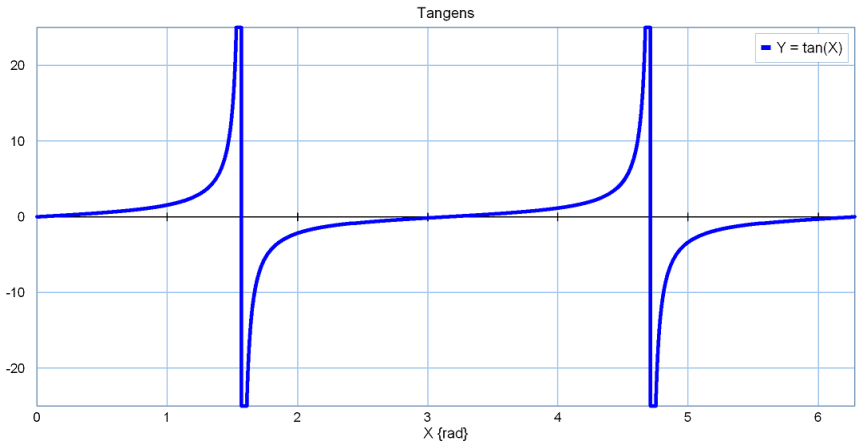
Returns the tangent of the elements of X.

Examples

```
b = ramp(2.5);
a = tan(b);
X = [ramp(0);ramp(1)];
Y = tan(X);
```

Limitations

Y and X must have the same size.



7.5 Operators

7.5.1 Operators

In 20-sim you can use the following operators in equations:

Operator	***Description***
*	Multiplication
+	Addition
-	Subtraction
.*	ArrayMultiplication
./	ArrayDivision
.^	ArrayPower
/	Division
div	Integer division
% or mod	Modulus operator
^	Power
and	Boolean and
or	Boolean or
xor	Boolean exclusive or
<	Less than
<=	Less than or equal
<>	Not equal
==	Equal
>=	Larger than or equal
>	Larger than
-	Prefix Minus Sign
+	Prefix Plus Sign
not	Prefix Boolean Not
| .. |	Absolute / Determinant / Norm
bitand	Bitwise AND
bitor	Bitwise OR
bitxor	Bitwise XOR
bitcmp	Bitwise complement
bitset	Bitwise set
bitget	Bitwise get
swapbytes	Swap bytes
bitshift	Bitwise shift

bitclear	Bitwise clear

7.5.2 Arithmetic

Absolute

Absolute / Determinant / Norm (|..|)

Syntax

|A| or *||A||*

Description

The upright stroke (|) is not a real operator. Depending on what's between the strokes, the following functions are applied:

| scalar | -> abs(scalar)
| matrix | -> det(matrix)
| vector | -> abs(vector)
|| matrix || -> norm(matrix)

Multiplication

Syntax

*A * B*

Description

A * B multiplies A with B. For nonscalar A and B, the number of columns of A must equal the number of rows of B. If C = A*B then the elements of C can be calculated out of the matrix elements of A and B as:

$$C[1,j] = \sum_{k=1}^{n} A[1,k]B[k,j]$$

A scalar can be multiplied with a matrix of any size.

Examples

A	B	A * B
5	*3*	*15*
-4.5	*5.4*	*-24.3*
2	*[1,2;3,4]*	*[2,4;6,8]*
[1,-2;-3,4]	*4*	*[4,-8;-12,16]*
[1,2,3;4,5,6]	*[1,2;3,4;5,6]*	*[22,28;49,64]*
[1,2;3,4;5,6]	*[1,2,3;4,5,6]*	*not allowed!*

Limitations

For nonscalar A and B, the number of columns of A must equal the number of rows of B.

Addition

Syntax

A + B

Description

A + B adds A and B. A and B must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size.

Examples

A	B	A + B
1	*3*	*4*
2.1	*-3.2*	*-1.1*
[1,2;3,4]	*[1,1;5,5]*	*[2,3;8,9]*
2	*[1,2;3,4]*	*[3,4;5,6]*
[1,2;3,4]	*4*	*[5,6;7,8]*
[1,2;3,4]	*[5,6;7,8;9,10]*	*not allowed!*

Limitations

If A and B are matrices, they must have the same size.

Subtraction

Syntax

A - B

Description

A - B subtracts B from A. A and B must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size.

Examples

A	**B**	**A - B**
5	*3*	*2*
-4.2	*5.5*	*-9.7*
[4,5;6,7]	*[1,1;5,5]*	*[3,4;1,2]*
[4,5;6,7]	*4*	*[0,1;2,3]*

4	*[4,5;6,7]*	*not allowed!*

Array Multiplication

Syntax

A . B*

Description

A .* B multiplies the elements of A by the elements of B. A.*B is the array or matrix with elements A(i,j)*B(i.j).

Examples

A	B	A .* B
[1,-2,-4,2]	*[1,2,1,4]*	*[1,-4,-4,8]*
[2,0;0,2]	*[1,2;3,4]*	*[2,0,0,8]*
[2,0;0,2]	*1*	*not allowed!*

Limitations

A and B must have the same size.

Array Division

Syntax

A ./ B

Description

A ./ B divides the elements of A by the elements of B. A./B is the matrix with elements A(i,j)/B(i.j).

Examples

A	B	A ./ B
[1,-2,-4,2]	*[1,2,1,4]*	*[1,-1,-4,0.5]*
[2,0;0,2]	*[1,2;3,4]*	*[2,0,0,0.5]*
[2,0;0,2]	*1*	*not allowed!*

Limitations

A and B must have the same size.

Array Power

Syntax

A .^ B

Description

A .^ B raises the elements of A to the power of the elements of B. A.^B is the matrix with elements A(i,j)^B(i.j).

Examples

A	B	A .^ B
[1,-2,-4,2]	*[1,2,1,4]*	*[1,4,-4,16]*
[2,2;3,2]	*[1,0;3,4]*	*[2,1;27,16]*
[2,0;0,2]	*1*	*not allowed!*

Limitations

A and B must have the same size.

Division

Syntax

A / B

Description

A / B divides A by B. For a nonscalar B, A/B equals A*inverse(B). For a nonscalar A, each element of A is divided by the scalar B.

Examples

A	B	A / B
15	*3*	*5*
-40.5	*5*	*-8.1*
2	*[1,2;3,4]*	*[-4,2;3,-1]*
[1,-2;-3,4]	*4*	*[0.25,-0.5;0.75,1]*
[2,0;0,2]	*[1,2;3,4]*	*[-4,2,3,-1]*
1	*0*	*not allowed!*
1	*[1,2,3;4,5,6]*	*not allowed!*

Limitations

If A and B are matrices, they must have the same size. If B is a matrix and it becomes singular, simulation is stopped.

Integer Division

Syntax

a div b

Description

a div b divides the scalar a by the scalar b and rounds the output toward zero.

Examples

a	b	a div b
10	*5*	*2*
9.9	*5*	*1*
-8	*2*	*-4*
-7.9	*2*	*-3*
-7.9	*-2*	*3*

Limitations

a and b must be scalars.

Modulus Operator

Syntax

a mod b

Description

a mod b returns the signed remainder after division:
a mod b = a - trunc(a / b) *b

Examples

a	b	a mod b
10	*5*	*0*
9.9	*5*	*4.9*
-8	*2*	*0*
-7.9	*2*	*-1.9*
-7.9	*-2*	*-1.9*

Limitations

a and b must be scalars.

Power

Syntax

A ^ b

Description

A ^ B raises A to the power B. For a nonscalar A and integer B, A^B is computed by repeated multiplication of A. If B is negative A is inverted first. If B is zero, A^B is the identity matrix. For other values of B, it is rounded to its nearest integer value. Making B a nonscalar is not allowed.

Examples

A	b	A ^ b
5	*3*	*125*
4	*-2.1*	*0.054409*
5.5	*0*	*1*
3	*2.1*	*10.04511*
[2,0;0,2]	*4*	*[16,0;0,16]*
[1,2,3,4]	*-2.1*	*[5.5,-2.5;-3.75,1.75]*
[1,2,3,4]	*0*	*[1,0;0,1]*
[1,2,3,4]	*[1,2;3,4]*	*not allowed!*

Limitations

b must be a scalar. For a matrix A, b is rounded to the nearest integer value value.

7.5.3 Binary

bitand

Syntax

a bitand b

Description

a bitand b performs a bitwise AND between *a* and *b*.

Example

In 20-sim all variables are stored as doubles. This means a decimal value of 30 is 4 bytes as shown in the figure below. Because the binary representation of 30 is equal to 11110, only the last byte is filled with non zero bits. Similar the binary representation of 9 is 1001. The binary value of *30 bitand 9* is equal to 01000, which has a decimal value of 8.

operation	A		B		C		D		Decimal
a	0000	0000	0000	0000	0000	0000	0001	1110	30
b	0000	0000	0000	0000	0000	0000	0000	1001	9
a bitand b	0000	0000	0000	0000	0000	0000	0000	1000	8
a bitor b	0000	0000	0000	0000	0000	0000	0001	1111	31
a bitxor b	0000	0000	0000	0000	0000	0000	0001	0111	23
a bitcmp b	0000	0000	0000	0000	0000	0001	1110	0001	481
a bitset 1	0000	0000	0000	0000	0000	0000	0001	1111	1
a bitget 5	0000	0000	0000	0000	0000	0000	0001	1110	1
swapbytes a	0001	1110	0000	0000	0000	0000	0000	0000	503316480
a bitshift 1	0000	0000	0000	0000	0000	0000	0011	1100	60
a bitclear 3	0000	0000	0000	0000	0000	0000	0001	1010	26

Limitations

a and b must be integers.

bitclear

Syntax

a bitclear b

Description

a bitclear b clears the bth bit of *a*.

Example

In 20-sim all variables are stored as doubles. This means a decimal value of 30 is 4 bytes as shown in the figure below. Because the binary representation of 30 is equal to 11110, only the last byte is filled with non zero bits. The operation *30 bitclear 3* clears the 3rd bit of 11110 which gives 11010. This number has a decimal value of 26.

operation	A		B		C		D		Decimal
a	0000	0000	0000	0000	0000	0000	0001	1110	30
b	0000	0000	0000	0000	0000	0000	0000	1001	9
a bitand b	0000	0000	0000	0000	0000	0000	0000	1000	8
a bitor b	0000	0000	0000	0000	0000	0000	0001	1111	31
a bitxor b	0000	0000	0000	0000	0000	0000	0001	0111	23
a bitcmp b	0000	0000	0000	0000	0000	0001	1110	0001	481
a bitset 1	0000	0000	0000	0000	0000	0000	0001	1111	1
a bitget 5	0000	0000	0000	0000	0000	0000	0001	1110	1
swapbytes a	0001	1110	0000	0000	0000	0000	0000	0000	503316480
a bitshift 1	0000	0000	0000	0000	0000	0000	0011	1100	60
a bitclear 3	0000	0000	0000	0000	0000	0000	0001	1010	26

Limitations

a and b must be integers.

bitcmp

Syntax

a bitcmp b

Description

a bitcmp b performs the *b* bit complement of *a*.

Example

In 20-sim all variables are stored as doubles. This means a decimal value of 30 is 4 bytes as shown in the figure below. Because the binary representation of 30 is equal to 11110, only the last byte is filled with non zero bits. The operator *30 bitcmp 9* performs the 9 bit complement of 30. This gives the binary number 111100001 which is equal to 481.

operation	A		B		C		D		Decimal
a	0000	0000	0000	0000	0000	0000	0001	1110	30
b	0000	0000	0000	0000	0000	0000	0000	1001	9
a bitand b	0000	0000	0000	0000	0000	0000	0000	1000	8
a bitor b	0000	0000	0000	0000	0000	0000	0001	1111	31
a bitxor b	0000	0000	0000	0000	0000	0000	0001	0111	23
a bitcmp b	0000	0000	0000	0000	0000	0001	1110	0001	481
a bitset 1	0000	0000	0000	0000	0000	0000	0001	1111	1
a bitget 5	0000	0000	0000	0000	0000	0000	0001	1110	1
swapbytes a	0001	1110	0000	0000	0000	0000	0000	0000	503316480
a bitshift 1	0000	0000	0000	0000	0000	0000	0011	1100	60
a bitclear 3	0000	0000	0000	0000	0000	0000	0001	1010	26

Limitations

a and b must be integers.

bitget

Syntax

a bitget b

Description

a bitget *b* gets the bth bit of *a*.

Example

In 20-sim all variables are stored as doubles. This means a decimal value of 30 is 4 bytes as shown in the figure below. Because the binary representation of 30 is equal to 11110, only the last byte is filled with non zero bits. The operation *30 bitget 5* gets the 5th bit of 11110 which is 1.

operation	A		B		C		D		Decimal
a	0000	0000	0000	0000	0000	0000	0001	1110	30
b	0000	0000	0000	0000	0000	0000	0000	1001	9
a bitand b	0000	0000	0000	0000	0000	0000	0000	1000	8
a bitor b	0000	0000	0000	0000	0000	0000	0001	1111	31
a bitxor b	0000	0000	0000	0000	0000	0000	0001	0111	23
a bitcmp b	0000	0000	0000	0000	0000	0001	1110	0001	481
a bitset 1	0000	0000	0000	0000	0000	0000	0001	1111	1
a bitget 5	0000	0000	0000	0000	0000	0000	0001	1110	1
swapbytes a	0001	1110	0000	0000	0000	0000	0000	0000	503316480
a bitshift 1	0000	0000	0000	0000	0000	0000	0011	1100	60
a bitclear 3	0000	0000	0000	0000	0000	0000	0001	1010	26

Limitations

a and b must be integers.

bitor

Syntax

a bitor b

Description

a bitor b performs a bitwise OR between *a* and *b*.

Example

In 20-sim all variables are stored as doubles. This means a decimal value of 30 is 4 bytes as shown in the figure below. Because the binary representation of 30 is equal to 11110, only the last byte is filled with non zero bits. Similar the binary representation of 9 is 1001. The binary value of *30 bitor 9* is equal to 11111, which has a decimal value of 31.

operation	A		B		C		D		Decimal
a	0000	0000	0000	0000	0000	0000	0001	1110	30
b	0000	0000	0000	0000	0000	0000	0000	1001	9
a bitand b	0000	0000	0000	0000	0000	0000	0000	1000	8
a bitor b	0000	0000	0000	0000	0000	0000	0001	1111	31
a bitxor b	0000	0000	0000	0000	0000	0000	0001	0111	23
a bitcmp b	0000	0000	0000	0000	0000	0001	1110	0001	481
a bitset 1	0000	0000	0000	0000	0000	0000	0001	1111	1
a bitget 5	0000	0000	0000	0000	0000	0000	0001	1110	1
swapbytes a	0001	1110	0000	0000	0000	0000	0000	0000	503316480
a bitshift 1	0000	0000	0000	0000	0000	0000	0011	1100	60
a bitclear 3	0000	0000	0000	0000	0000	0000	0001	1010	26

Limitations

a and b must be integers.

bitset

Syntax

a bitset b

Description

a bitset b sets the bth bit of *a*.

Example

In 20-sim all variables are stored as doubles. This means a decimal value of 30 is 4 bytes as shown in the figure below. Because the binary representation of 30 is equal to 11110, only the last byte is filled with non zero bits. The operation *30 bitset 1* sets the last bit of 11110 to one which gives 11111. This number has a decimal value of 31.

operation	A		B		C		D		Decimal
a	0000	0000	0000	0000	0000	0000	0001	1110	30
b	0000	0000	0000	0000	0000	0000	0000	1001	9
a bitand b	0000	0000	0000	0000	0000	0000	0000	1000	8
a bitor b	0000	0000	0000	0000	0000	0000	0001	1111	31
a bitxor b	0000	0000	0000	0000	0000	0000	0001	0111	23
a bitcmp b	0000	0000	0000	0000	0000	0001	1110	0001	481
a bitset 1	0000	0000	0000	0000	0000	0000	0001	1111	1
a bitget 5	0000	0000	0000	0000	0000	0000	0001	1110	1
swapbytes a	0001	1110	0000	0000	0000	0000	0000	0000	503316480
a bitshift 1	0000	0000	0000	0000	0000	0000	0011	1100	60
a bitclear 3	0000	0000	0000	0000	0000	0000	0001	1010	26

Limitations

a and b must be integers.

bitshift

Syntax

a bitshift b

Description

a bitshift b shifts the bits of *a* with *b* places.

Example

In 20-sim all variables are stored as doubles. This means a decimal value of 30 is 4 bytes as shown in the figure below. Because the binary representation of 30 is equal to 11110, only the last byte is filled with non zero bits. The operation *30 bitshift 1* shifts the bits of 11110 with one place which gives 111100. This number has a decimal value of 60.

operation	A		B		C		D		Decimal
a	0000	0000	0000	0000	0000	0000	0001	1110	30
b	0000	0000	0000	0000	0000	0000	0000	1001	9
a bitand b	0000	0000	0000	0000	0000	0000	0000	1000	8
a bitor b	0000	0000	0000	0000	0000	0000	0001	1111	31
a bitxor b	0000	0000	0000	0000	0000	0000	0001	0111	23
a bitcmp b	0000	0000	0000	0000	0000	0001	1110	0001	481
a bitset 1	0000	0000	0000	0000	0000	0000	0001	1111	1
a bitget 5	0000	0000	0000	0000	0000	0000	0001	1110	1
swapbytes a	0001	1110	0000	0000	0000	0000	0000	0000	503316480
a bitshift 1	0000	0000	0000	0000	0000	0000	0011	1100	60
a bitclear 3	0000	0000	0000	0000	0000	0000	0001	1010	26

Limitations

a and b must be integers.

bitxor

Syntax

a bitxor b

Description

a bitxor b performs a bitwise XOR between *a* and *b*.

Example

In 20-sim all variables are stored as doubles. This means a decimal value of 30 is 4 bytes as shown in the figure below. Because the binary representation of 30 is equal to 11110, only the last byte is filled with non zero bits. Similar the binary representation of 9 is 1001. The binary value of *30 bitxor 9* is equal to 10111, which has a decimal value of 23.

operation	A		B		C		D		Decimal
a	0000	0000	0000	0000	0000	0000	0001	1110	30
b	0000	0000	0000	0000	0000	0000	0000	1001	9
a bitand b	0000	0000	0000	0000	0000	0000	0000	1000	8
a bitor b	0000	0000	0000	0000	0000	0000	0001	1111	31
a bitxor b	0000	0000	0000	0000	0000	0000	0001	0111	23
a bitcmp b	0000	0000	0000	0000	0000	0001	1110	0001	481
a bitset 1	0000	0000	0000	0000	0000	0000	0001	1111	1
a bitget 5	0000	0000	0000	0000	0000	0000	0001	1110	1
swapbytes a	0001	1110	0000	0000	0000	0000	0000	0000	503316480
a bitshift 1	0000	0000	0000	0000	0000	0000	0011	1100	60
a bitclear 3	0000	0000	0000	0000	0000	0000	0001	1010	26

Limitations

a and b must be integers.

7.5.4 Boolean

And

Syntax

a and b

Description

a and b performs a logical AND between a and b. If a or b are not a boolean, 0 represents a logical false and any nonzero value represents a logical true. The truth table for this operator is shown below.

Table

a	***b***	***a and b***
false	*false*	*false*
false	*true*	*false*
true	*false*	*false*
true	*true*	*true*

Limitations

a and b must be scalars.

Or

Syntax

a or b

Description

a or b performs a logical OR between a and b. If a or b are not booleans, 0 represents a logical false and any nonzero value represents a logical true. The truth table for this operator is shown below.

Table

a	b	a or b
false	*false*	*false*
false	*true*	*true*
true	*false*	*true*
true	*true*	*true*

Limitations

a and b must be scalars.

Xor

Syntax

a xor b

Description

a xor b performs a logical Exclusive OR between a and b. If a or b are not booleans, 0 represents a logical false and any nonzero value represents a logical true. The truth table for this operator is shown below.

Table

a	b	a xor b
false	*false*	*false*
false	*true*	*true*
true	*false*	*true*
true	*true*	*false*

Limitations

a and b must be scalars.

7.5.5 Comparison

Less than

Syntax

a < b

Description

The < operator compares two arguments and yields a boolean. If a is less than b, then a < b yields true. If a is equal or larger than b, then a < b yields false.

Example

a	b	a < b
1	*2*	*true*
1.01e2	*101.0*	*false*
31	*1*	*false*
-12	*-11*	*true*

Limitations

a and b must be scalars.

Less than or Equal

Syntax

a <= b

Description

The <= operator compares two arguments and yields a boolean. If a is less than or equal to b, then a <= b yields true. If a is larger than b, then a <= b yields false.

Example

a	b	a < b
1	*2*	*true*
1.01e2	*101.0*	*true*
31	*1*	*false*
-12	*-11*	*true*

Limitations

a and b must be scalars.

Not Equal

Syntax

a <> b

Description

The <> operator compares two arguments and yields a boolean. If a is equal to b, then a <> b yields false. If a is not equal to b, then a <> b yields true.

Example

a	b	a <> b
1	*1*	*false*
1.01e2	*101.0*	*false*
31	*1*	*true*
true	*false*	*true*

Limitations

a and b must be scalars.

Equal

Syntax

a == b

Description

The == operator compares two arguments and yields a boolean. If a is equal to b, then a==b yields true. If a is not equal to b, then a==b yields false. It is typically used in an if-statement.

Example

a	b	a == b
1	*1*	*true*
1.01e2	*101.0*	*true*
31	*1*	*false*
true	*false*	*false*

Note

The normal equal sign (=) should only be used for equations. The equal sign (=) makes the left side of an equation equal to the right side. This is different to the equal statement (==) which compares two arguments and yields a boolean.

Larger Than

Syntax

a > b

Description

The > operator compares two arguments and yields a boolean. If a is less than or equal to b, then a > b yields false. If a larger than b, then a > b yields false.

Example

a	b	a > b
1	*2*	*false*
1.01e2	*101.0*	*false*
31	*1*	*true*
-12	*-11*	*false*

Limitations

a and b must be scalars.

Larger Than or Equal

Syntax

a >= b

Description

The >= operator compares two arguments and yields a boolean. If a is less than b, then a > b yields false. If a larger than or equal to b, then a > b yields true.

Example

a	b	a >= b
1	*2*	*false*
1.01e2	*101.0*	*true*
31	*1*	*true*
-12	*-11*	*false*

Limitations
a and b must be scalars.

7.5.6 Prefix

Prefix Plus Sign

Syntax

Y = +A

Description
The plus sign may be used as a prefix for scalars, vectors and matrices.

Examples

A	**+A**
5	*5*
[4,5;-6,7.1]	*[4;5;-6;7.1]*

Prefix Minus Sign

Syntax

Y = -B

Description
The minus sign may be used as a prefix for scalars, vectors and matrices.

Examples

A	**-A**
5	*-5*
[4,5;-6,7.1]	*[-4;-5;6;-7.1]*

Boolean Not

Syntax

y = not a

Description
The boolean prefix not performs an inversion of a.

Table

a	***not a***
false	*true*
true	*false*

Limitations

a must be a boolean.

swapbytes

Syntax

swapbytes a

Description

swapbytes a swaps the bytes of a.

Example

In 20-sim all variables are stored as doubles. This means a decimal value of 30 is 4 bytes as shown in the figure below. Because the binary representation of 30 is equal to 11110, only the last byte is filled with non zero bits. The operation *swapbytes 30* swaps byte D to A, C to B, B to C and A to D (see figure below) which gives a decimal value of 503316480.

operation	A		B		C		D		Decimal
a	0000	0000	0000	0000	0000	0000	0001	1110	30
b	0000	0000	0000	0000	0000	0000	0000	1001	9
a bitand b	0000	0000	0000	0000	0000	0000	0000	1000	8
a bitor b	0000	0000	0000	0000	0000	0000	0001	1111	31
a bitxor b	0000	0000	0000	0000	0000	0000	0001	0111	23
a bitcmp b	0000	0000	0000	0000	0000	0001	1110	0001	481
a bitset 1	0000	0000	0000	0000	0000	0000	0001	1111	1
a bitget 5	0000	0000	0000	0000	0000	0000	0001	1110	1
swapbytes a	0001	1110	0000	0000	0000	0000	0000	0000	503316480
a bitshift 1	0000	0000	0000	0000	0000	0000	0011	1100	60
a bitclear 3	0000	0000	0000	0000	0000	0000	0001	1010	26

Limitations

a and b must be integers.

7.6 Statements

7.6.1 Statements

Statements are important to guide the flow of information in equation models. 20-sim supports the following statements:

1. For to Do
2. If Then
3. If Then Else
4. If Then Else (expression)
5. Repeat Until
6. Switch Case
7. Stopsimulation
8. Warning
9. While Do
10. toMatlab
11. fromMatlab
12. doMatlab
13. Effortincausality
14. Flowincausality

7.6.2 If Then

Syntax

if condition then
equation;
....
end;

Description

The simplest if statement evaluates a condition and performs the specified equation(s) if the condition is true. If the condition is not true, 20-sim ignores the equation(s).

Examples

```
b = false;
y = ramp(1);
u = -ramp(1);
if time > 5 then
      y = -ramp(1); // equations to be executed if condition is true,
      u = ramp(1); // these are not executed if condition is false
end;
if time == 10 then // note the use of the equal operator ==
      b = true; // equations to be executed if condition is true,
end;
```

Limitations

1. The output of the condition must be a boolean.
2. If needed, always use the equal operator (==) in the condition.

Note

1. Take care when using event functions in if-then-else statements. In if-then-else statements only the equations of the true parts are evaluated, so event functions may not always be triggered!
2. Equations within an if statement have to be written in the correct order of execution, i.e. they are not rewritten into a causal form but executed sequentially.

7.6.3 If Then Else

Syntax

```
if condition then
      equation;
      ...
else
      equation;
      ...
end;
```

Description

The if-then-else statement evaluates a condition and performs the first set of equation(s) if the condition is true and the second set of equation(s) if the condition is false. If the condition is true, 20-sim ignores the second set of equation(s). If the condition is false, 20-sim ignores the first set of equation(s).

Nesting

If-then-else statements may be nested as many times as desired. Do not forget to included the word "end;" to finish every nested statement!

Examples

```
y = step(1);
u = -step(1);
if time > 5 then
    y = -ramp(1); // equations to be executed if condition is true,
    u = ramp(1); // these are not executed if condition is false
else
    y = ramp(1); // equations to be executed if condition is false,
    u = -ramp(1); // these are not executed if condition is true
end;

//Nesting
    if time < 5 then
        a = 1;
        b = sin(time*1);
    else
        if time == 5 then
                a = 2;
                b = sin(time*2);
        else
                a = 3
                b=sin(time*3);
        end;
    end;
```

Limitations

1. The output of the condition must be a boolean.
2. If needed, always use the equal operator (==) in the condition.

Note

1. Take care when using event functions in if-then-else statements. In if-then-else statements only the equations of the true parts are evaluated, so event functions may not always be triggered!
2. There is also an if-then-else expression.
3. Equations within an if statement have to be written in the correct order of execution, i.e. they are not rewritten into a causal form but executed sequentially.

7.6.4 If Then Else (expression)

Syntax

```
y = if condition then
      expression
else
      expression
end;
```

Description

The if-then-else expression evaluates a condition and makes y equal to the first expression if the condition is true and makes y equal to the second expression if the condition is true. If the condition is true, 20-sim ignores the second expression. If the condition is false, 20-sim ignores the first expression. Because the complete construct forms one expression, only one semicolon is used at the end.

Nesting

If-then-else expression may be nested. Take care that **no semicolons** are used, because no matter how large the construct is, it is still one expression!

Examples

```
y = if time > 5 then
                -ramp(1) // statement to be executed if condition is true,
      else
                ramp(1) // statement to be executed if condition is false,
      end;

// Nesting:
x = if time < 1 then
                1
      else
                if time < 2 then
                      2
                else
                      if time == 2 then
                            1
                      else
                            0
                      end
                end
      end;
```

Limitations

1. The output of the condition must be a boolean.
2. If needed, always use the equal operator (==) in the condition.

Note

1. Take care when using event functions in if-then-else expressions. In if-then-else expressions only the equations of the true parts are evaluated, so event functions may not always be triggered!

2. There is also an if-then-else statement.

3. Equations within an if statement have to be written in the correct order of execution, i.e. they are not rewritten into a causal form but executed sequentially.

7.6.5 For To Do

Syntax

```
for variable = start to stop by step do
    equation;
    equation;
    ....
end;
```

Description

The for statement is designed to execute equation(s) a fixed number of times. It begins with the keyword **for** and an expression that specifies the number of times the equations within the for statement should be executed. The for statement ends with the keyword **end**. The default increment in a for statement is one (1), but you can use the keyword **by** to adjust the increment.

The equations within the for statement are always executed once. The for statement stops when the variable has exceeded the stop value.

Examples

```
variables
    real i,y[10];
equations
    for i = 1 to 2 do
            // executed 2 times: y[1] and y[2]
            y[i] = ran (1);
    end;
    for i = 2 to 4.1 by 2 do
            // executed 3 times: y[2], y[4] and y[6]
            y[i] = ran (1);
    end;
```

Limitations

1. The start, stop and step values must be scalars. The step value is optional. There are no limitations to the number of equations that can be used within the for statement.

2. Equations within an for statement have to be written in the correct order of execution, i.e. they are not rewritten into a causal form but executed sequentially.

7.6.6 While Do

Syntax

```
while condition do
    equation;
    equation;
    ....
end;
```

Description

The while statement is designed to execute equation(s) repeatedly, as long as the condition is true. The condition is evaluated before the equations are executed.

It is the user's responsibility to guarantee that the execution of the while statement finishes by making the condition become false at a certain time.

Examples

```
variables
    real i,y[10];
equations
    i = 1;
    while i < 3 do
            // executed 2 times: y[1] and y[2]
            y[i] = ran (1);
            // i is incremented to guarantee a stop of the loop
            i = i+1;
    end;
```

Limitations

1. The output of the condition must be a boolean. If the boolean does not become false, the loop never ends! There are no limitations to the number of equations that can be used within the while statement.

2. Equations within an while statement have to be written in the correct order of execution, i.e. they are not rewritten into a causal form but executed sequentially.

7.6.7 Repeat Until

Syntax

```
repeat
    equation;
    equation;
    ....
until condition;
```

Description

The repeat statement is designed to execute equation(s) repeatedly, as long as the condition is true. The condition is evaluated after the equations are executed.

It is the user's responsibility to guarantee that the execution of the repeat statement finishes by making the condition become false at a certain time.

Examples

```
variables
    real i,y[10];
equations
    i = 1;
    repeat
            // executed 2 times: y[1] and y[2]
            y[i] = ran (1);
            // i is incremented to guarantee a stop of the loop
            i = i+1;
    until i > 2;
```

Limitations

1. The output of the condition must be a boolean. If the boolean does not become false, the loop never ends! There are no limitations to the number of equations that can be used within the repeat statement.
2. If needed, always use the equal operator (==) in the condition.
3. Equations within an repeat statement have to be written in the correct order of execution, i.e. they are not rewritten into a causal form but executed sequentially.

7.6.8 Switch Case

Syntax

```
switch variable
    case condition1 do
            equations;
    case condition2 do
            equations;
    default do
            equations;
end;
```

Description

The switch case statement is designed to execute equation(s) based on the condition of a variable. Only the equations of the valid branch are executed. If non of the branches is valid, the equations of the default branch are executed.

Limitations

1. The switch variable should not be changed inside one of the branches.
2. Equations within a switch case statement have to be written in the correct order of execution, i.e. they are not rewritten into a causal form but executed sequentially. Therefore it is advised only to use this statement in a code section.

7.6.9 Stopsimulation

Syntax

stopsimulation ('string');

Description

The stopsimulation function stops the simulation as soon as it comes to action. It is useful when used in combination with an if statement. As soon as the simulation stops, the log window will appear showing the string that is used in the function.

- **Normal stop**: If the string starts with 'Error', the Simulator will stop and the Editor will jump to the the model containing the stopsimulation function.
- **Silent stop**: If the string does not start with 'Error' the Simulation will jump to the *finalequations* section, execute them and stop the simulation.

Examples

In the example below the Simulator will stop the simulation at time > 6 and show a message 'time > 6' in the output tab. The *finalequations* are evaluated and number has a value of 2:

```
variables
    integer number;
initialequations
    number = 1;
equations
    if time > 6 then
        stopsimulation ('time > 6' );
    end;
finalequations
    number = 2;
```

In the example below the Simulator will stop the simulation at time > 6 and show a message 'time > 6' in the output tab. The Editor will jump to this model showing an error message. The *finalequations* are not evaluated and number has a value of 1:

```
variables
    integer number;
initialequations
    number = 1;
equations
    if time > 6 then
        stopsimulation ('Error time > 6' );
    end;
finalequations
    number = 2;
```

7.6.10 Warning

Syntax

warning ('string',boolean);

Description

The warning function displays a message in the the log window. You can choose to display the string once (boolean = true) or every simulation step (boolean = false). It is advised to use the warning function in combination with an if-then statement or event-function.

Examples

```
warning('this message will be displayed every simulationstep!',false);
if time > 1
    warning('this message will be displayed every simulationstep after time > 1!',false);
end;
if x > 100 then
    warning('this message will be displayed only once after x > 100',true);
end;
if event(y) then
    warning('this message will be displayed when y is exactly 0',false);
end;
```

7.6.11 toMatlab

Syntax

toMatlab(<20-sim variable>)
toMatlab(<20-sim variable>, 'matlab variable name')

Description

The **toMatlab** statement allows you to pass a variable from 20-sim to Matlab.

The first argument is the name of the 20-sim variable to send to Matlab (source).
This 20-sim variable can be a scalar variable, a vector or a matrix.
The second (optional) argument *'matlab variable name'* is the name of the corresponding matlab variable (destination).

When the *'matlab variable name'* argument is omitted, 20-sim will use the full hierarchical name of the 20-sim variable.
A 20-sim variable named *Controller\x* becomes in Matlab: *Controller_x*

The corresponding **fromMatlab** statement allows you to retrieve a variable from Matlab.

Example

```
// this example shows how a variable can be transferred to matlab and from matlab during simulation
variables
        real x;
        real y;
        real vector[2];
        real matrix[2,2];
```

```
// at the start of the simulation
initialequations
        // create an empty array a
        doMatlab ('a = []; ');
        // create a variable
        doMatlab ('b=0; ');
        // create a vector
        doMatlab ('vector=[1 2];');
        // create a matrix
        doMatlab ('matrix=[1 2; 3 4];');

// during simulation
equations
        // calculate x
        x = sin (time);
        // send it to Matlab
        toMatlab (x,'x');
        // and add it to array a
        doMatlab ('a = [ a x ]; ');
        // in Matlab add 2 to x
        doMatlab ('b = x + 2;');
        // read matlab 'b' into 20-sim variable 'y'
        fromMatlab(y, 'b');
        // read matlab 'vector' into 20-sim variable 'vector'
        fromMatlab(vector, 'vector');
        // read matlab 'matrix' into 20-sim variable 'matrix'
        fromMatlab(matrix, 'matrix');

// at the end of the simulation
finalequations
        // plot the resulting array in Matlab
        doMatlab (' plot (a); ');
```

7.6.12 domatlab

Syntax

doMatlab('string')

Description

The **doMatlab** statement allows you to pass a command line string to Matlab.

Example

```
// this example shows how a variable can be transferred to matlab and from matlab
during simulation
variables
        real x;
        real y;
        real vector[2];
        real matrix[2,2];

// at the start of the simulation
initialequations
```

```
        // create an empty array a
        doMatlab ('a = []; ');
        // create a variable
        doMatlab ('b=0; ');
        // create a vector
        doMatlab ('vector=[1 2];');
        // create a matrix
        doMatlab ('matrix=[1 2; 3 4];');

// during simulation
equations
        // calculate x
        x = sin (time);
        // send it to Matlab
        toMatlab (x,'x');
        // and add it to array a
        doMatlab ('a = [ a x ]; ');
        // in Matlab add 2 to x
        doMatlab ('b = x + 2;');
        // read matlab 'b' into 20-sim variable 'y'
        fromMatlab(y, 'b');
        // read matlab 'vector' into 20-sim variable 'vector'
        fromMatlab(vector, 'vector');
        // read matlab 'matrix' into 20-sim variable 'matrix'
        fromMatlab(matrix, 'matrix');

// at the end of the simulation
finalequations
        // plot the resulting array in Matlab
        doMatlab (' plot (a); ');
```

7.6.13 frommatlab

Syntax

fromMatlab(<20-sim variable>)
fromMatlab(<20-sim variable>, 'matlab variable name')

Description

The **fromMatlab** statement allows you to retrieve a variable from Matlab and store it into a 20-sim variable.
The first argument is the name of the 20-sim variable to use as storage (destination).
This 20-sim variable can be a scalar variable, a vector or a matrix and should match the size used at the Matlab side.
The second (optional) argument *'matlab variable name'* is the name of the Matlab variable to retrieve (source).

When the *'matlab variable name'* argument is omitted, 20-sim will use the full hierarchical name of the 20-sim variable.
A 20-sim variable named *Controller\x* requests from Matlab: *Controller_x*

The corresponding **toMatlab** statement allows you to pass a variable from 20-sim to Matlab.

Example

```
// this example shows how a variable can be transferred to matlab and from matlab during simulation
variables
        real x;
        real y;
        real vector[2];
        real matrix[2,2];

// at the start of the simulation
initialequations
        // create an empty array a
        doMatlab ('a = []; ');
        // create a variable
        doMatlab ('b=0; ');
        // create a vector
        doMatlab ('vector=[1 2];');
        // create a matrix
        doMatlab ('matrix=[1 2; 3 4];');

// during simulation
equations
        // calculate x
        x = sin (time);
        // send it to Matlab
        toMatlab (x,'x');
        // and add it to array a
        doMatlab ('a = [ a x ]; ');
        // in Matlab add 2 to x
        doMatlab ('b = x + 2;');
        // read matlab 'b' into 20-sim variable 'y'
        fromMatlab(y, 'b');
        // read matlab 'vector' into 20-sim variable 'vector'
        fromMatlab(vector, 'vector');
        // read matlab 'matrix' into 20-sim variable 'matrix'
        fromMatlab(matrix, 'matrix');

// at the end of the simulation
finalequations
        // plot the resulting array in Matlab
        doMatlab (' plot (a); ');
```

7.6.14 Effortincausality

Syntax

effortincausality (portname) then
equation;
equation;
...
else
equation;
equation;
...
end;

Description

The effortincausality statement evaluates if a port has causality effort in and performs the first set of equation(s) if the condition is true and the second set of equation(s) if the condition is false. If the condition is true, 20-sim ignores the second set of equation(s). If the condition is false, 20-sim ignores the first set of equation(s).

Depending on the domain you are working in, effort and flow correspond to particular variables. The table below shows the variables for the domains that are currently supported in 20-sim.

Domain	**effort (e)**	**flow (f)**
power	effort *e*	flow *f*
mechanical (translation)	force *F* [N]	velocity *v* [m/s]
mechanical (rotation)	torque *T* [Nm]	angular velocity *omega* [rad/s]
pneumatic	pressure *p* [Pa]	volume flow *phi* [m^3/s]
thermal	temperature *T* [K]	entropy flow *dS* [J/Ks]
electric	voltage *u* [V]	current *i* [A]
hydraulic	pressure *p* [Pa]	volume flow *phi* [m^3/s]
magnetic	current *i* [A]	voltage *u* [V]
pseudothermal	temperature *T* [K]	heat flow *dQ* [W]

Examples

```
effortincausality p then
    R = if p.u > 0 then Ron else Roff end; // executed if causality of port p is effort (u) in
else
    R = if p.i > 0 then Ron else Roff end; // executed if causality of port p is flow (i) in
end;
p.i = p.u/R;
```

Note

1. Take care when using event functions in effortincausality statements. In this statement only the equations of the true parts are evaluated, so event functions may not always be triggered!
2. There is also a flowincausality statement.
3. Equations within an effortincausality have to be written in the correct order of execution, i.e. they are not rewritten into a causal form but executed sequentially.

7.6.15 Flowincausality

Syntax

```
flowincausality (portname) then
    equation;
    equation;
    ...
else
    equation;
    equation;
    ...
end;
```

Description

The effortincausality statement evaluates if a port has causality flow in and performs the first set of equation(s) if the condition is true and the second set of equation(s) if the condition is false. If the condition is true, 20-sim ignores the second set of equation(s). If the condition is false, 20-sim ignores the first set of equation(s).

Depending on the domain you are working in, effort and flow correspond to particular variables. The table below shows the variables for the domains that are currently supported in 20-sim.

Domain	**effort (e)**	**flow (f)**
power	effort *e*	flow *f*
mechanical (translation)	force *F* [N]	velocity *v* [m/s]

mechanical (rotation)	torque *T* [Nm]	angular velocity *omega* [rad/s]
pneumatic	pressure *p* [Pa]	volume flow *phi* [m^3/s]
thermal	temperature *T* [K]	entropy flow *dS* [J/Ks]
electric	voltage *u* [V]	current *i* [A]
hydraulic	pressure *p* [Pa]	volume flow *phi* [m^3/s]
magnetic	current *i* [A]	voltage *u* [V]
pseudothermal	temperature *T* [K]	heat flow *dQ* [W]

Examples

```
flowincausality p then
R = if p.i > 0 then Ron else Roff end; // executed if causality of port p is flow (i) in
else
R = if p.u > 0 then Ron else Roff end; // executed if causality of port p is effort (u)
in
end;
p.u = p.i*R;
```

Note

1. Take care when using event functions in effortincausality statements. In this statement only the equations of the true parts are evaluated, so event functions may not always be triggered!
2. There is also an effortincausality statement.
3. Equations within an effortincausality have to be written in the correct order of execution, i.e. they are not rewritten into a causal form but executed sequentially.

7.7 Matrices and Vectors

7.7.1 Declaration

Matrices and column vectors are declared in the *constants*, *parameters* and *variables* sections of a model. Declaration of matrices and column vectors is equal to to the declaration of scalars, the size however has to be specified explicitly. For example:

```
parameters
    real A[3,3], B[3,3];
    integer C[2,2] = [1,3;5,7];
    real K[4] = [1;2;3;4];
variables
    real K[4],L[4],D[4,5];
    integer M[2,2],N[2,2];
```

Matrices and Vectors can only be of the datatype real or integer.

7.7.2 Notation

Matrix sizes in 20-sim are always denoted by [n,m] where n is the numbers of rows and m is the numbers of columns. Elements can separately be denoted with commas (,) distinguishing between row elements and semicolons (;) distinguishing between rows.

matrix size	**matrix form**	**20-sim notation**
[4,1]	1 3 5 7	[1;3;5;7]
[1,4]	1 3 5 7	[1,3,5,7]
[2,3]	1 2 3 4 5 6	[1,2,3;4,5,6]
[3,2]	1 2 3 4 5 6	[1,2;3,4;5,6]

Column vectors are matrices with only one column. Consequently the column size may be omitted in the notation.

column size	***column form***	***20-sim notation***
[4]	1 3 5 7	[1;3;5;7]

7.7.3 Use

Matrices and Vectors can be used in equations just like scalars. If possible, the element notation can be left out. **If equations get ambiguous, element notation must be used!**

Whole matrix or vector

```
K = 1; // Make all elements of K equal to 1.
M = C; // Make matrix M equal to matrix C (sizes have to be equal).
N = D*inverse(L); // Make matrix N equal to the matrix product of D
// and the inverse of L (sizes of N, D and L have to be equal).
```

Matrix and vector elements

```
N = [sin(time),cos(time);cos(time),-sin(time)]; // Make elements of N equal to functions.
L[4] = time; // Make element 4 of columned L equal to time.
D[2,5] = A[2,2]*B[1,1]; // Declare one element
```

Multiple elements (ranges)

To prevent multiple equations for assigning matrix elements, ranges can be assigned using a colon. E.g. 1:5 means element 1 to 5, 7:8 means element 7 and 8. Backward counting ranges (like 10:1) are not allowed!

```
D[2,1:5] = A[1,1:5]; // D[2,1] = A[1,1], ... , D[2,5] = A[1,5]
variables
    real v[3],p[6,6];
equations
    v = p[4:6,6]; // v[1] = p[4,6], ... , v[3] = p[6,6]
```

Operators

Some scalar operators can also be used for matrices and vectors. Depending on the specific operator, the meaning may differ for scalars, vectors and matrices.

Operator	Description
*	Multiplication
+	Addition
-	Subtraction
.*	ArrayMultiplication
./	ArrayDivision
.^	ArrayPower
/	Division
^	Power
-	Prefix Minus Sign
+	Prefix Plus Sign
| .. |	Absolute / Determinant / Norm

Functions

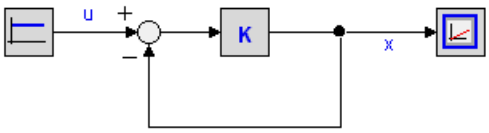
A lot of special matrix and vector functions are supported in 20-sim:

Function	*Description*
	Adjoint
	adjoint
	Antisym
	Columns
	Cross
	Determinant
	Diag
	Eye
	Homogeneous
	Inner
	Inverse
	InverseH
	Linsolve
	Max
	Min
	Msum
	Norm
	Norminf
	Rows
	Skew
	Sym
	Tilde
	Trace
	Transpose

7.8 Advanced Topics

7.8.1 Algebraic Loops

An algebraic loop in a model is a loop consisting of elements without "memory like" functions. To calculate the variables in this loop, the variable values themselves are needed. Consider the following example of an algebraic loop in an amplifier with negative feedback:



Standard derivation of a simulation model would yield:

$$x = K*(u-x)$$

The variable x depends on its own value and must be solved by **iteration**. In 20-sim every simulation algorithm is accompanied by an iteration routine. Fortunately 20-sim is able to solve many algebraic loops at equation level. For this model this leads to the analytic solution the system:

$$x = K*u/(1+K)$$

Simulating algebraic loops

Although 20-sim contains a sophisticated algorithms to find analytic solutions, the occurrence of unbreakable loops can not always be prevented. The occurrence of algebraic loops may lead to an increase of simulation time, or even stop simulation when iteration fails.

The best solution for these problems is to have a critical look at the model and change the order of calculations in a model. Possible solutions are:

- Algebraic Loops occur when the order of calculations is arbitrary. When an algebraic loop occurs in an equation model or in a set of equation models, you may change the order of calculation by rewriting the equations. The calculation order in bond graph models can be changed by introducing hand-defined causality.
- Introduce 'parasitic' energy storage elements (e.g. a small mass, a small capacitor etc.) to break an algebraic loop. These elements introduce however, large poles in the state equations, which might increase the simulation time considerably.
- Delete elements in the algebraic loop which are not relevant for the model's simulation output (e.g. small dampers, very stiff springs etc.). Care should however be taken, since correct deletions are not always possible and require considerable modeling skill and intuition.
- Combine dual elements. Sometimes elements of the same type can be combined by adding the parameter values (e.g. combining a mass m1 and a mass m2 to a mass m1 + m2). This will in most cases decrease the amount of algebraic loops.

7.8.2 Causal Form

Equations within 20-sim may be entered in random form. During compilation, 20-sim will automatically try to rewrite equations into a causal form and set them in a correct order. I.e. a form where all output variables are written as function of input variables. Consider for example the following model:

```
variables
    real u,z;
equations
    u = sin(time);
    u = cos(z);
```

Here the (input) variable u is given by the equation *u = sin(time)*. Consequently the (output) variable z should be written as a function of u. This is exactly what 20-sim will try to do while compiling the model into simulation code. I.e. the function cos will be inverted and the model will be rewritten to:

```
variables
    real u,z;
equations
    u = sin(time);
    z = arccos(u);
```

Some functions cannot be inverted. Consequently not all equations can be rewritten. 20-sim will report this to the user, during model checking.

Indifferent Causality

For some models the causal form is not known beforehand. For example the iconic diagram model that describes a damper can be written as:

```
parameters
    real d;
equations
    p.F = d*p.v;
```

Here *p.F* denotes the force and *p.v* denotes the velocity of the powerport *p*. There is no preferred input (force or velocity). This can be explicitly stated, by giving the powerport *p* an indifferent causality. During compilation 20-sim will determine whether *p.F* or *p.v* is the input variable and consequently rewrite the equations.

Preferred Causality

For some models there is a preferred causal form. For example the iconic diagram model that describes a spring can be written as:

```
parameters
    real k;
equations
    p.F = (1/k)*int(p.v);
```

Here *p.F* denotes the force and *p.v* denotes the velocity of the powerport *p*. The equation is written in integral form which is preferred. Consequently the preferred input is the velocity. Should the force be the input, the equation must be rewritten to a differential form, which is leads to less efficient simulation. This can be explicitly stated, by giving the powerport *p* an preferred causality. During compilation 20-sim will try to keep the model in this preferred form. If this is not possible the equations will be rewritten to the less preferred form.

Preferred causality has a lower priority than fixed causality. During compilation 20-sim will first assign all models with a fixed causality and then all models with preferred causality.

Likes Causality

For some models there is a causal form which is liked more than the other. For example the iconic diagram model that describes a parasitic volume can be written as:

```
effortincausality(p) then
        p.phi = 0;
else
        volume_ratio = int(p.phi/V);
        p.p = B * volume_ratio + p_initial;
end;
```

Here *p.phi* denotes the volume flow and *p.p* denotes the pressure of the powerport *p*. The equation is written in integral form which is liked. Consequently the preferred output is the pressure. Should the pressure be the input, the equation gives a zero flow as output. During compilation 20-sim will first try to keep all models the liked form. If this is not possible the equations will be rewritten to the other form.

Liked causality has a lower priority than preferred causality. During compilation 20-sim will first assign all models with a preferred causality and then all models with likes causality.

Fixed Causality

For some models there is only one causal form. For example a simple iconic diagram model that describes coulomb friction can be written as:

```
parameters
    real Rc;
equations
    p.F = Rc*abs(p.v);
```

Here *p.F* denotes the force and *p.v* denotes the velocity of the powerport *p*. The equation cannot be rewritten to a form where *p.F* is the input. This can be explicitly stated, by giving the powerport *p* a fixed causality. During compilation 20-sim will try to keep the model in this fixed form. If this is not possible an error message will be generated.

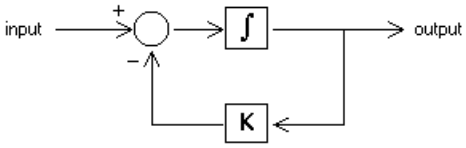
Fixed causality has a lower priority than a likes causality. During compilation 20-sim will first assign all models with a likes causality and then all models with fixed causality.

Setting Causality

For some models, the equations are too complex to analyze causality. To help 20-sim, using the right causality, you can set causality for every port in the Interface Editor.

7.8.3 Integral Form

Consider the following first order linear model:



This model can be described by the dynamic equation:

*output = int(input - K*output)*

Now look at the following model:

This model can be described by the dynamic equation:

output = (input - ddt(output))/K

Note that we can rewrite this equation as:

*ddt(output) = input - K*output*

or

*output = int(input - K*output)*

This is the same equation as the previous model! Apparently, both models are the same! Both models can therefore be described by the dynamic equations:

*ddt(output) = input - K*output*
*output = int(input - K*output)*

We call the first equation the *differential form* (no integrals). The second equation is called the *integral form* (no derivatives). In 20-sim, models can be entered in integral form as well as the differential form.

Solving Differential Equations

During compilation, 20-sim will automatically try to rewrite equations into the integral form, since this leads to more efficient simulation. Sometimes an integral form cannot be found. Then algorithms will be applied to solve the differential directly. For example an equation like:

*output = ddt(sin(a*time))*

will be replaced by the following equation (applying the chain rule and using the known derivative for the sine function):

```
output = a*cos(a*time)
```

Sometimes a differential cannot be solved directly. Then only the Backward-Differentiation Methods can be used for simulation.

Simulation Code

After compilation simulation code is generated. The equation in integral form:

```
output = int( input - K*output )
```

will be written as:

```
independent state = output
independent rate = input - K*output
```

and can be handled by all integration methods. The equation in differential form:

```
ddt(output) = input - K*output
```

will be written as:

```
dependent rate = input - K*output
dependent state = output
```

and can only be handled by the Backward-Differentiation Methods.

7.8.4 Order of Execution

Equations within 20-sim may be entered in random form. During compilation, 20-sim will automatically try to rewrite equations into a correct order of execution. I.e. a form where all output variables are written as function of input variables and output variables of previous lines. Consider for example the following equations:

```
variables
    real u,z;
equations
    z = sin(u);
    u = cos(time);
```

Here the (input) variable *z* is given as a function of *u*. Consequently u should be calculated first. This is exactly what 20-sim will try to do while compiling the model into simulation code. I.e. the equations will be executed as:

```
u = cos(time);
z = sin(u);
```

Code Blocks

Equations in a **code block** are not reordered. A code block is a set of equations inside a statement. Suppose we have the following equations:

```
if condition then
    code block 1
    ...
    ...
else
    code block 2
    ...
    ...
end;
```

To prevent incorrect executions of the if-statement, the equations of the code blocks will not be separated. Inside a code-block, equations can are not rewritten into an new order of execution. E.g. the following equations:

```
if time > 10 then
    z = sin(u);
    a = z^2;
    u = cos(time);
end;
```

Will be **not be reordered** and therefore not correctly executed! To get correct code, enter code blocks in a correct order, e.g.:

```
if time > 10 then
    u = cos(time);
    z = sin(u);
    a = z^2;
end;
```

Prevent Order Changes

To make all equations a code block you can use the code section. E.g.

```
parameters
    real y = 8;
variables
    real x1, x2, x3;
code
    x1 = time;
    x2 = sin(time);
    x3 = y*x1;
```

Integration Steps

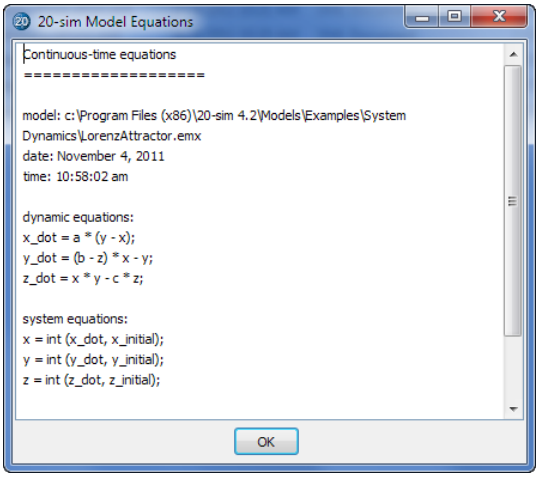
Some integration algorithms do more calculations before generating the next output value. These calculations are called minor steps, the output generation is called a major step. During a minor step, all model equations are executed. In most cases you will not notice this because only the results of the major step are shown in the simulator. There are however, exceptions. The next topic will discuss this in more detail.

7.8.5 Show Equations

During processing a complete set of equations is generated of each model. To inspect these equations or copy them for use in other programs, you have to:

1. From the **Model menu** select the **Check Complete Model** command.
2. From the **Model menu** select the **Show Equations** command.

Now a window is popped up showing all the model equations.



The model equations after compiling.

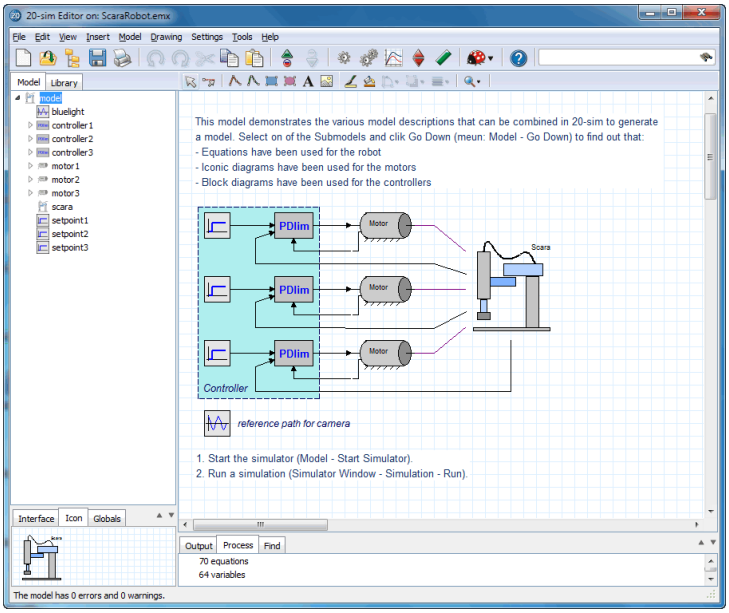
8 Toolboxes

8.1 3D Mechanics Toolbox

8.1.1 Introduction

Introduction

With the 3D Mechanics Editor you can create 3D dynamic models of mechanical structures. The resulting 3D mechanical model can be connected with other 20-sim components. In the example shown below *(Getting Started\3D Mechanics Toolbox \ScaraRobotSimulation.emx)*, you see a robot model that has been generated with the 3D Mechanics Editor. The model has two actuated rotation joints and one actuated translation joint. The rotation joints are directly coupled to gearboxes. The translation joint is coupled via a spindle.



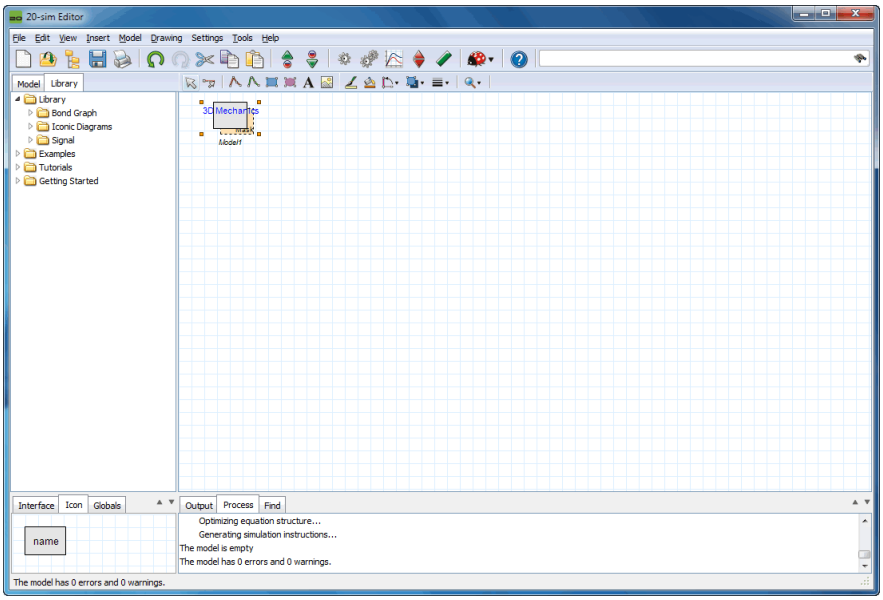
In 20-sim you can use plots to show the simulation results and use the 3D Animation Toolbox to show an animation of the 3D model.

Opening The 3D Mechanics Editor

You can open the *3D Mechanics Editor* by:

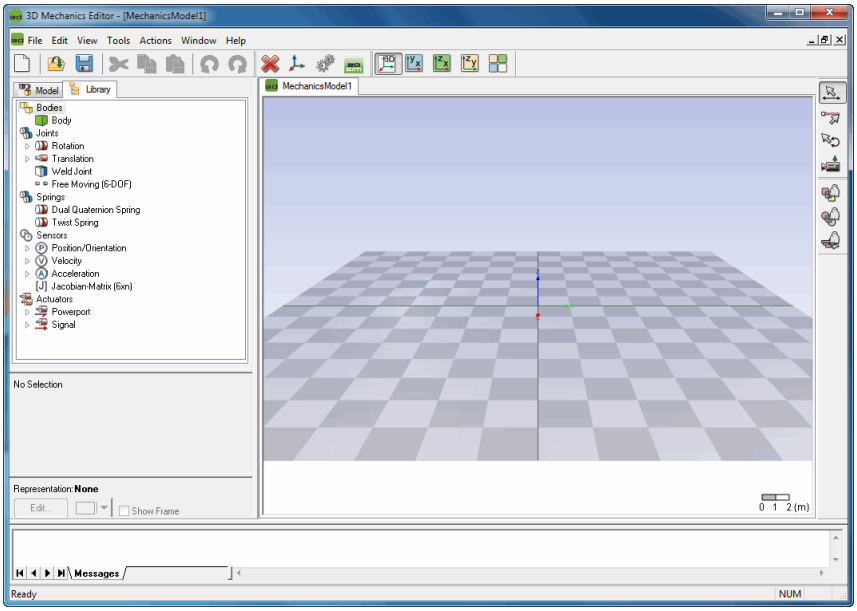
1. In the 20-sim Editor from the Tools menu select 3D Mechanics Toolbox and 3D Mechanics Editor.

Now a masked model is inserted in the *Editor* and the *3D Mechanics Editor* will be opened.



If you select a 3D Mechanics model and select Go Down, the 3D Mechanics Editor will be automatically opened.

The first time you start 20-sim 3D Mechanics Editor the following screen will appear.

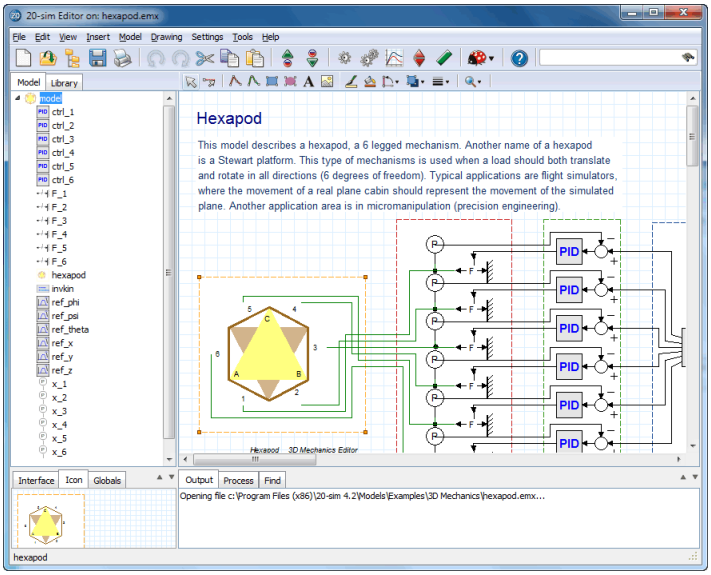


The 3D Mechanics Editor

Migrating from Older Versions

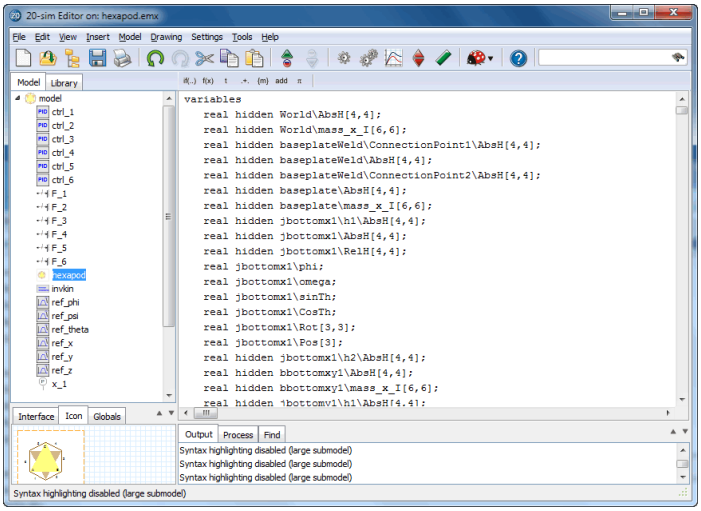
This topic is only important for users who are migrating from older version of 20-sim and want to use previously constructed 3D Mechanics models. Since 20-sim 4.1, the *3D Mechanics Editor* operates on masked models. I.e. you can open the *3D Mechanics Editor* by selecting a *3D Mechanics model* and clicking the *Go Down* command. If you import a 20-sim 4.0 model, you will notice that this option does not work. We will explain the actions you have to perform to do make this trick work on older models.

1. Open the 20-sim model that contains a submodel that was generated by the 3D Mechanics Editor (make sure you have stored an emergency copy).



The star shaped model was created using the 3D Mechanics Editor of 20-sim 4.0.

2. Select the model that was generated by the 3D Mechanics Editor and select Go Down.

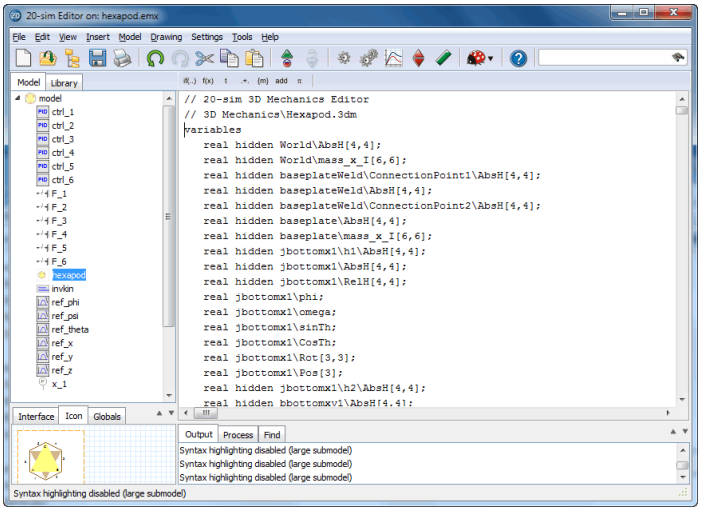


The automatically generated equations of a 3D Mechanical Model.

3. At the top of the model enter the following code:

```
// 20-sim 3D Mechanics Editor
// C:\Program Files\20-sim 4.5\Models\Examples\3D Mechanics\hexapod.3dm
```

The first line tells 20-sim which editor should be opened. The second line shows the location of the corresponding 3D Mechanics configuration file. Enter the path and name of your own file here.

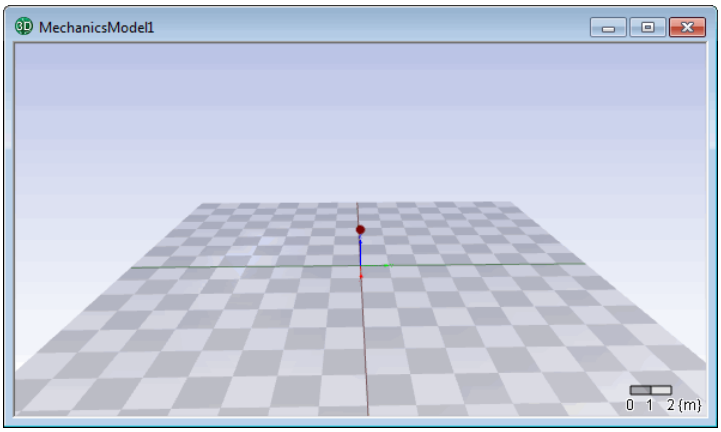


The comment at the top tells 20-sim which editor should be opened and the corresponding configuration file.

The next time you select the model and select Go Down the 3D Mechanics Editor will be opened with the proper 3D Mechanics model.

Edit Window

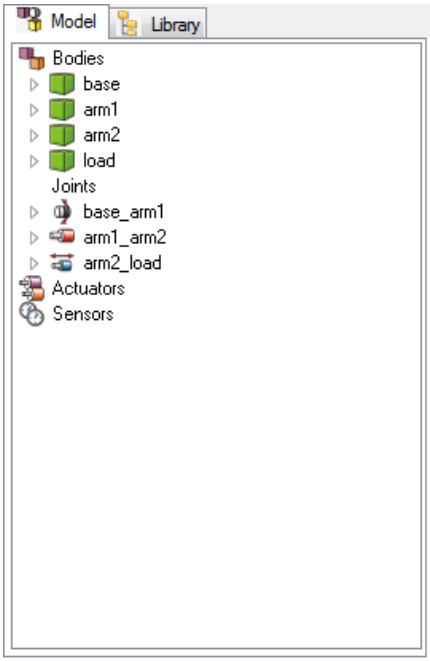
In the center of the 3D Mechanics Editor the **Edit window** is visible. At the start a reference frame and a base plane is shown. In the Edit window, you can insert components and assemble you model.



In the Edit Window you can assemble your model.

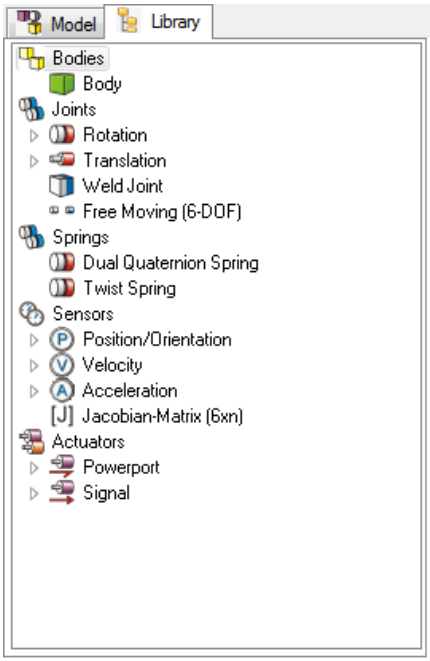
Model Tab

One of the two tabs at the left of the 3D Mechanics Editor is the **Model tab**. The Model tab contains a tree of all the components of in your current model. This tree can be used to select components and edit their properties.



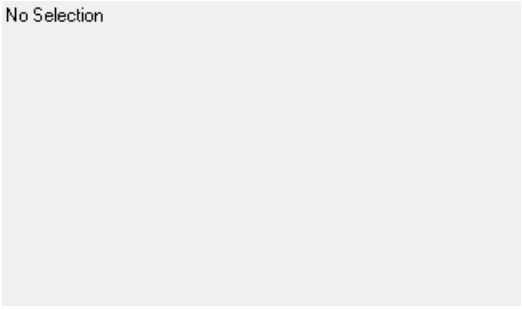
Library Tab

One of the two tabs at the left of the 3D Mechanics Editor is the **Library tab**. The Library tab contains predefined components, which you can use to construct your model with. The components in the library tree can be dragged into the Edit window.

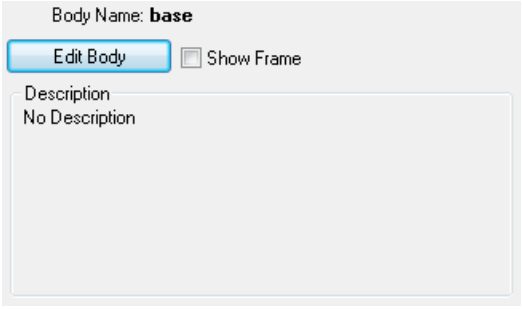


Selection Properties

Below the Bodies/Library tabs the *properties* of the selected component are shown. When no components are selected the selection properties are empty.



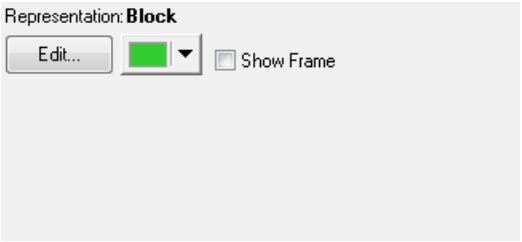
When a component is selected, the edit button will be enabled. You can click the button to change the component properties.



3D Representation

Below the Selection Properties, the 3D representation of the selected component is shown. When no components, the 3D Representation is empty.

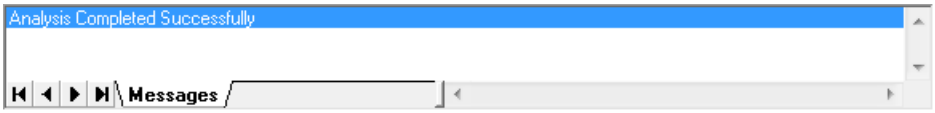
When a component is selected, the edit button will be enabled. You can click the button to change the 3D Representation of the component.



A component may be composed of several representations. Using the *Add Representation* and *Delete Representation* commands of the *Actions* menu you can add more representations or remove representations.

Output

In the Output part of the 3D Mechanics Editor, messages will be shown during analysis of the model.



Edit Modes

At the right of the 3D Mechanics Editor you will find the **Edit Mode** buttons. You can select them to enter several edit modes.

	Translation Mode	In this mode you can translate components.
	Connection Mode	In this mode you can make connections between components.
	Rotation Mode	In this mode you can rotate components.
	Camera Mode	In this mode you can change the camera position and orientation. I.e. the way you look at your model.

Ghost Modes

At the right of the 3D Mechanics Editor you will find the **Ghost Mode** buttons. You can use these buttons to make components transparent. This is useful when components are hidden inside other components.

	Ghost Mode for Bodies	Make every body transparent.

	Ghost Mode for Joints	Make every joint transparent.
	Ghost Mode for Sensors/ Actuators	Make every sensor/actuator transparent.

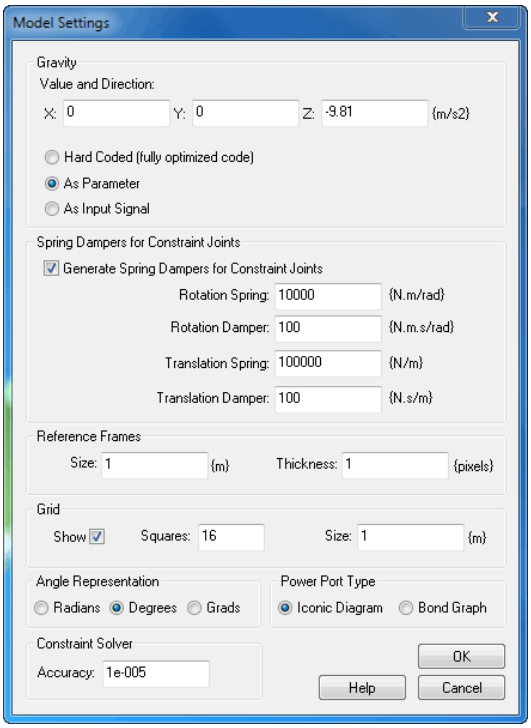
View Modes

In the button bar of the 3D Mechanics Editor you will find the *View Mode* buttons. You can select them to select different views.

	3D View	Default view.
	X-Y plane	View perpendicular to the x-y plane. You can only move objects along the x-y plane.
	X-Z plane	View perpendicular to the x-z plane. You can only move objects along the x-z plane.
	Y-Z plane	View perpendicular to the y-z plane. You can only move objects along the y-z plane.
	All	A combination of the all the views.

Model Settings

The *model Settings* command of the *Tools* menu will open the *Model Settings* dialog. In this dialog you can change some general settings such as the size of the reference frames or the amount of gravity.



The Model Setting Dialog.

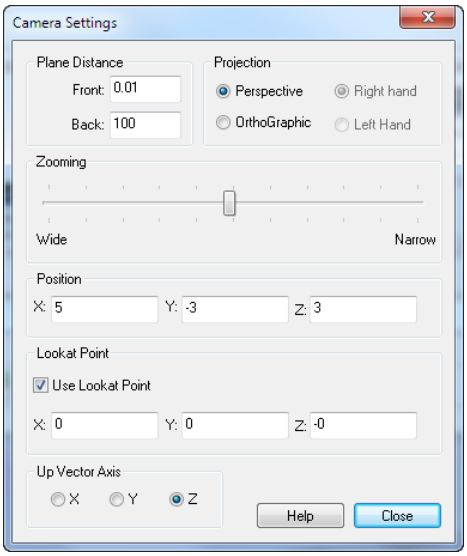
Items

- Gravity: You can set the gravitational acceleration to any desired value.
  - Hard Coded: choose this option if you want to optimize for speed.
  - As Parameter: Use a gravity parameter, allowing you to change the value before the start of a simulation run.
  - As Input Signal: get the gravity value from an input signal, allowing you to change the value during simulation.
- Spring Dampers for Constraint Joints: Choose this option if you want to use spring dampers for closed kinematic chains.
  - Rotation Spring: Set the desired stiffness for the rotational spring.
  - Rotation Damper: Set the desired rotational damping.
  - Rotation Spring: Set the desired stiffness for the translation spring.
  - Translation Damper: Set the desired translation damping.

- Reference Frames: Choose the size and thickness of the reference frames.
- Grid: Set the grid.
- Angle Representation: 20-sim uses SI-units for the model equations (e.g. radians). You can choose to show them in non-SI-units like degrees or grads.
- Power Port Type: Choose Iconic Diagram power ports or bond Graph power ports.
- Constraint Solver: set the accuracy of the constraint solver.

Camera Settings

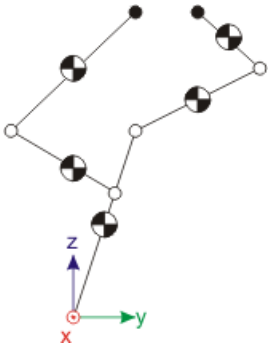
At the start of a model session, in the Edit window, a default camera is used. You can change camera settings by clicking **Camera Settings** from the **Tools** menu.



In the Camera Settings dialog you can change the range (front and back plane) and zooming of the camera, use a perspective view or orthographic, select the camera position and the point to look at.

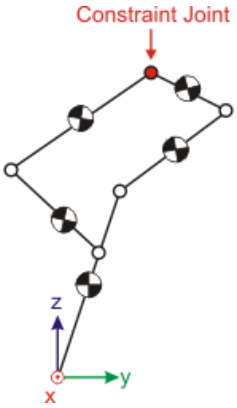
Spring Dampers and Constraints

In the 3D Mechanics Editor you can connect bodies with joints. If no path from any body through a joint to another body and so on never returns to the starting body, we have an open chain mechanism. Open chain mechanisms are easy to simulate with al known integration methods.



Open Chain Mechanism.

If a path from any body through a joint to another body and so on returns to the starting body we have a closed chain mechanism. Closed chain mechanisms are generally more difficult to simulate.



Closed Chain Mechanism.

20-sim uses a special feature to simulate closed chain mechanisms. At an arbitrary place a joint is removed to create an open chain mechanism. Then the open chain is closed again using a *spring damper* joint or a *constraint* joint. You can choose which joint to use in the Model Settings Dialog.

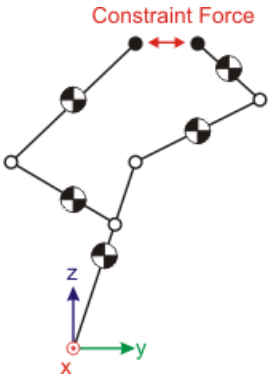
Spring Damper Joint

A spring damper joint is a joint where all rigid connections are replaced by spring dampers. If weak spring dampers are used a certain amount of (unwanted) displacement will be possible. By increasing the stiffness and damping, the displacement can be decreased. You can set the spring stiffness and damping in the Model Settings Dialog.

Constraint Joint

The constraint joint works exactly like the original joint but uses constraint forces to do the job. Constraint forces can only be simulated with special integration methods. In 20-sim the Modified Backward Differentiation Method is able to calculate constraint forces.

The concept of constraint forces is quite easy to understand if we look at the open chain mechanism below. At the tips of the mechanism a force is applied. If we apply a force that keeps the tips at the same position at each time step, it will effectively work as a rotation joint. That is exactly what constraint forces do. At every simulation step, an iterative procedure is run to find exactly that force that keeps the position offset at zero.



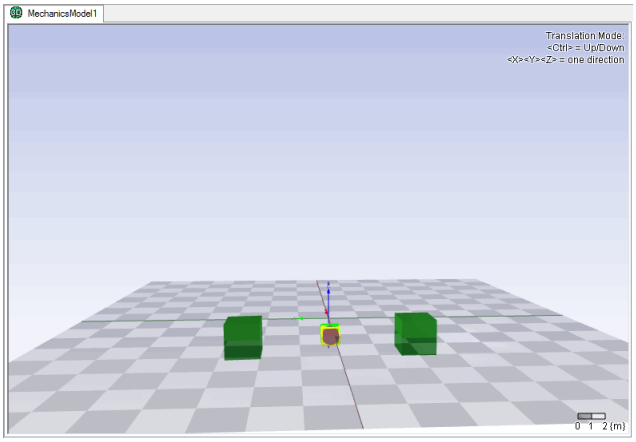
Constraint Force.

This iterative procedure is not standard for most integration methods. In 20-sim only the Modified Backward Differentiation Method supports it. In the 3D Mechanics Editor you do not have to worry about closed chain mechanisms. If necessary, standard joints are replaced automatically by constraint joints. You only have to realize that simulation of a mechanism with constraint joints in 20-sim should be performed using the Modified Backward Differentiation Method.

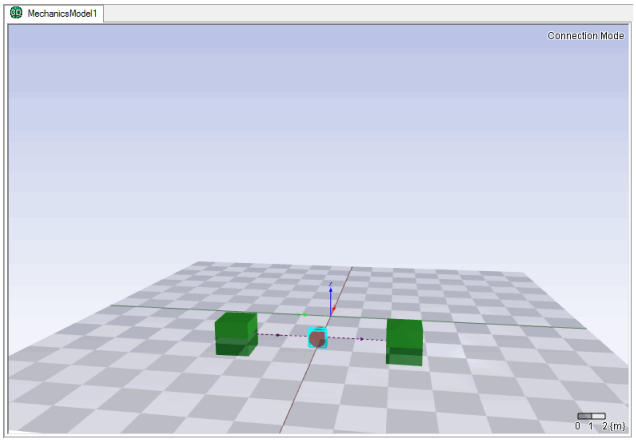
8.1.2 Working Order

3D Mechanics Editor

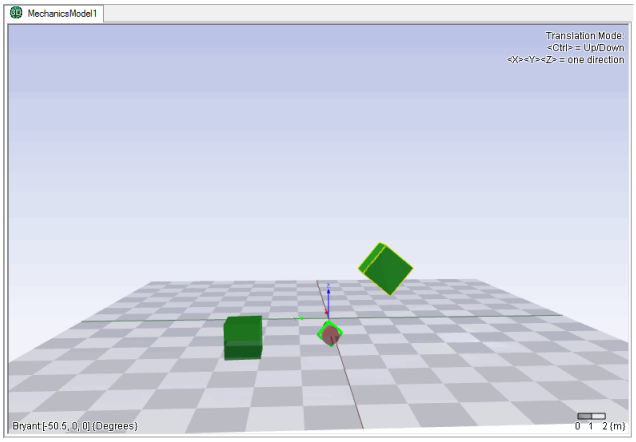
From the Library tab you can drag and drop components to the Edit window. If you put the editor in translation mode you can select the components and use the mouse to put them in the correct position.



A standard model will at least consist of some bodies and joints. The bodies have to be connected with the joints to define the degrees of freedom of the model. The connecting of joints and bodies can be done in the connection mode. By clicking on a body and then on a joint a connection is defined.



If all connections are defined, you can test the possible motions of your model. Return to the translation mode and click on a body. You will see arrows that indicate the possible motion of the connected joint. By dragging the mouse you can make the body move.



When the model is ready you can check it. When no errors are found you can update it to 20-sim.

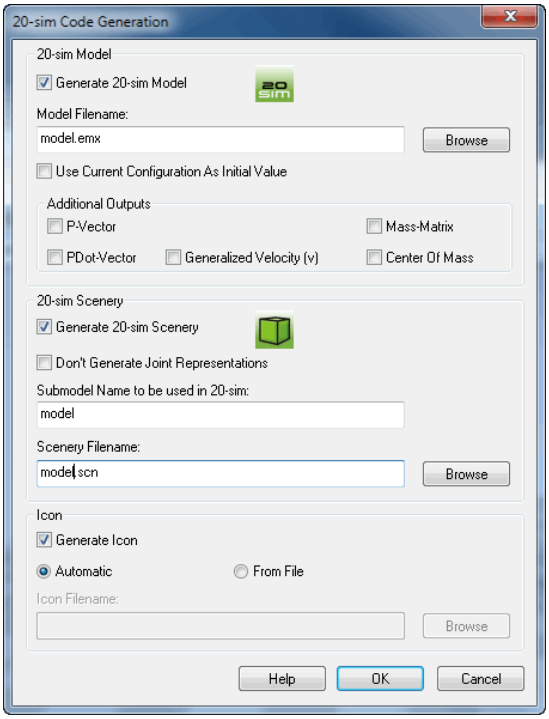
Generating a 20-sim Model

Checking

When your 3D dynamic model has been created and assembled, you can check for errors by selecting the *Check Model* command from the *Actions* menu. If everything is OK the Output will show the message "Analysis Completed Successfully".

Generating 20-sim Model

After the model has been checked, you can export it as a 20-sim submodel by selecting the *Generate 20-sim Model* command from the Actions menu.



20-sim Model

As you may know from the Joint Properties dialog, all joints have an initial value. During analysis of your model, some joints may have been twisted. You can use the current (twisted) configuration to generate a 20-sim model for, or reset it to its initial values first.

Additional Outputs

For advanced users, some additional outputs may be useful. First time users are advised not to select these outputs.

Output Filename

Use the *Browse* button to select a folder and a filename for the generated 20-sim model.

20-sim Scenery

In 20-sim you can use the 3D Animation Toolbox to show an animation of the mechanical structure. You can export the structure to a scenery file, which can be loaded in the 3D Animation Toolbox.

- Don't Generate Joint Representations: Each joint is represented in the 3D Animation as autonomous component with variables to denote the potion and orientation. For large models you can save memory space by not exporting the joints.
- Submodel Name to be used in 20-sim: The 3D Animation will use 20-sim submodel names to identify the movements of object. You have to enter the submodel name, that you will use for the 3D dynamic model in 20-sim.

- Scenery Filename: Use the Browse button to select a folder and a filename for the generated scenery file.

8.1.3 Library

Library

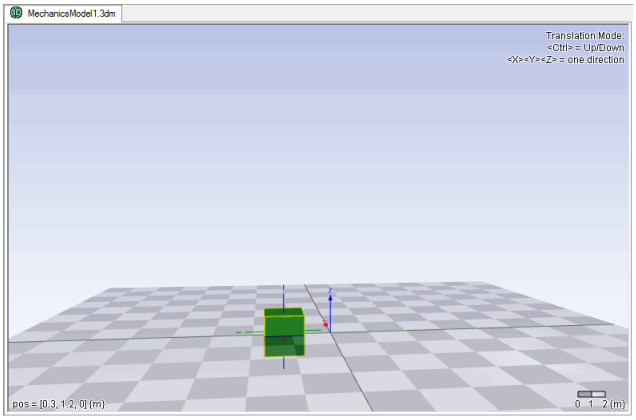
The Library Tab shows all components that can be used to create your 3D Dynamic model.

- Bodies
- Joints
- Sensors
- Actuators

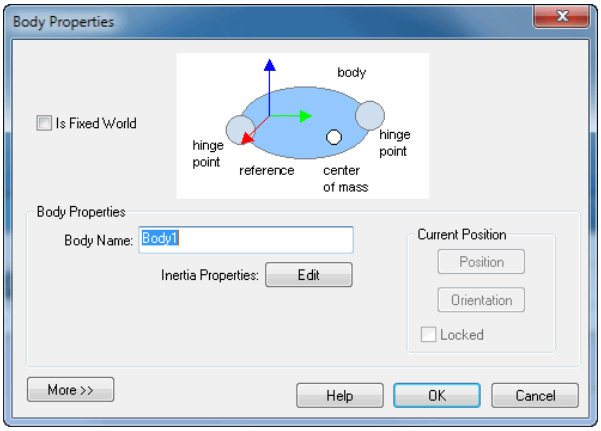
Bodies

Position

When you drag and drop a body into the Edit window, its position (i.e. the position of its reference frame) will be defined with respect to the base reference frame. You can change the position with the mouse pointer. From the **View** menu click *Mode and Translation* or click the button to go to translation mode. If you put the mouse pointer on top of the body, you can change its position by pressing the left mouse button. If you pressing the *Ctrl-button* at the same time, the body will move up and down.



You can also set the position manually by double clicking the body. This will open the Body Properties dialog.



Using this dialog you can:

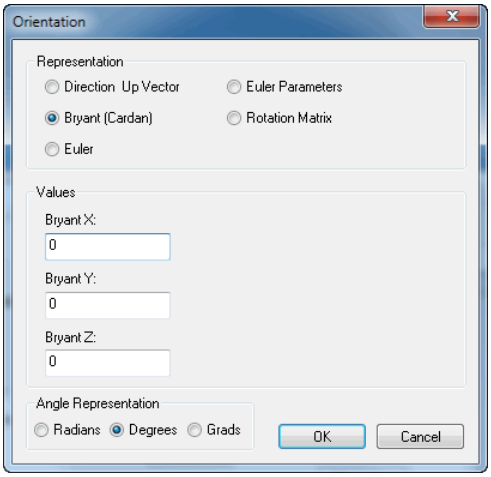
- change the name of the body
- set the inertia parameters
- change the position
- change the orientation
- make it a fixed body
- set more options

Fixed Body

The purpose of building a 3D model in most cases is to construct some kind of mechanism using bodies and joints. The joints make the bodies move in a particular way, which can be inspected in the 3D Editor. Every body in 3D mechanics editor is floating by default, unless specifically set to be attached to the fixed world, or connected to another body by means of a joint. Any body can be set to be a fixed body by checking the **Is Fixed World** option in the **Body Properties dialog.** Changing the position of a fixed body in a mechanism will change the base position of the mechanism. Dragging the other bodies will only make the mechanism move, but not change the base position. If you want to model a floating mechanism (e.g. a satellite in space), all bodies should remain floating.

Orientation

To set the orientation of a body, the same procedure can be followed as for the position of a body. If you set the Editor in Rotation mode ( ), you can change the orientation of the body with the mouse pointer. You can also use the Body Properties dialog to set the orientation precisely by clicking the orientation button.



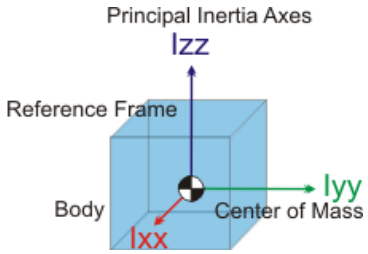
The orientation of a body is specified by the orientation of its reference frame. You can define this orientation using various representations.

- Direction Up Vector: Using the Direction Up vector method you have to specify the Y-axis (Direction) and Z-axis (Up) of the body reference frame.
- Bryant angles: Using Bryant angles we start with the initial body reference frame. First we rotate around the X-axis of the body reference frame, then we rotate around the Y-axis of the body reference frame en finally around the Z-axis of the body reference frame. Bryant angles are also known as X-Y-Z relative Euler angles or Cardan angles.
- X-Y-X Euler: Using X-Y-X Euler angles we start with the initial body reference frame. First we rotate around the X-axis of the body reference frame, then we rotate around the Y-axis of the body reference frame en finally around the X-axis of the body reference frame.
- Euler Parameters: Using Euler parameters we start with the initial body reference frame and rotate this frame around a vector K [X,Y,Z] about an angle theta.
- Rotation Matrix: Using a 3x3 rotation matrix, you can enter 9 elements that will transform the initial orientation of the body reference frame to the new orientation.

Inertia Properties

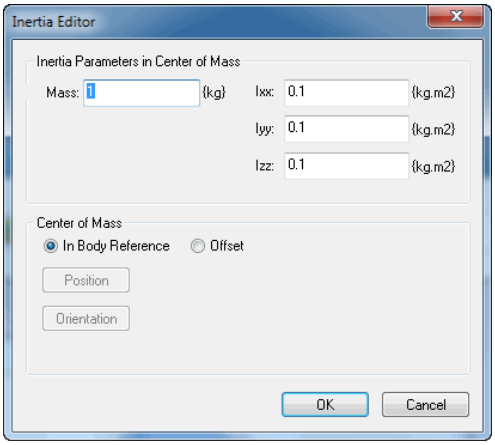
Each body has a mass and rotational inertia. The mass indicates the resistance of the body to a change in position. Given a fixed force, a body with a large mass will accelerate more slowly. This property is the same for all directions. Therefore the mass can be indicated by a single parameter.

The rotational inertia indicates the resistance of a body to a rotation. Given a fixed torque, a body wit a large rotational inertia will start to rotate more slowly. The rotational inertia is not the same for all orientations. Generally, for any body three axes can be identified that indicate a stable rotation. The rotational inertia for these three principal axes of rotation is sufficient to describe the motion of a body. The connection of these three principal axes of rotation is generally known as the center of mass.

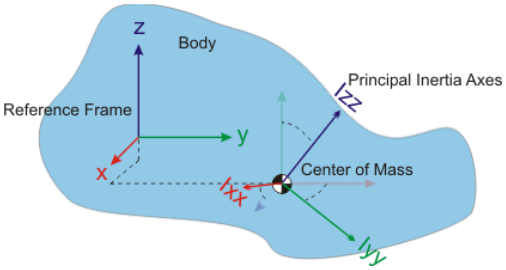


Any body, indifferent of its shape, which has the same mass, the same principal axes of rotation and the same rotational inertia, will show the same dynamic behavior. That is why you can change the representation of a body to any size, it will not change its dynamic properties.

If you open the **Body Properties Dialog** you can click the **Edit** button to change the inertia properties. You can set mass and the rotational inertias for each of the principle inertia axis.



The center of mass does not necessarily have to coincide with the body reference frame and the orientation of the principal inertia axes do not have to coincide with the orientation of the body reference frame.



You can enter the offset position of the center of mass and the orientation of the principal inertia axes, by clicking the offset button.

More Options

If you click the More button in the *Body Properties dialog*, options are shown which form a shortcut for the normal way of building models. They are meant for experienced users.

First a list of connection points are shown and some buttons to edit these points. Connection points are used to indicate at which point a joint or other element is connected with the body. Connection Points are explained in more detail in the next section.

At the bottom two options are shown. If you choose the Generate Power Interaction Port option, a power port will be added to the body, which allows you to insert forces and torques onto the body, from other 20-sim submodels.

The Output Absolute H-matrix option, will create an 4x4 output signal that gives the position and orientation of the body, using an H-matrix. This signal can be connected to other 20-sim submodels.

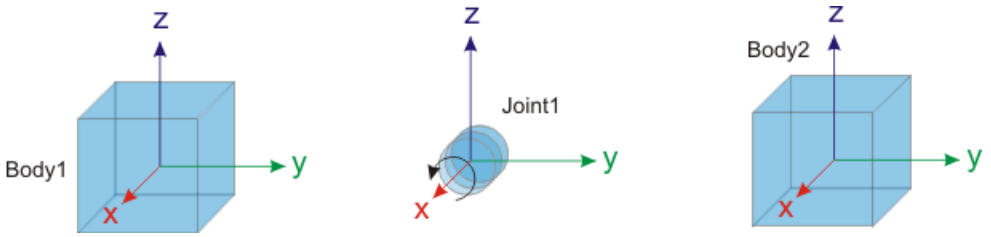
Joints

Properties

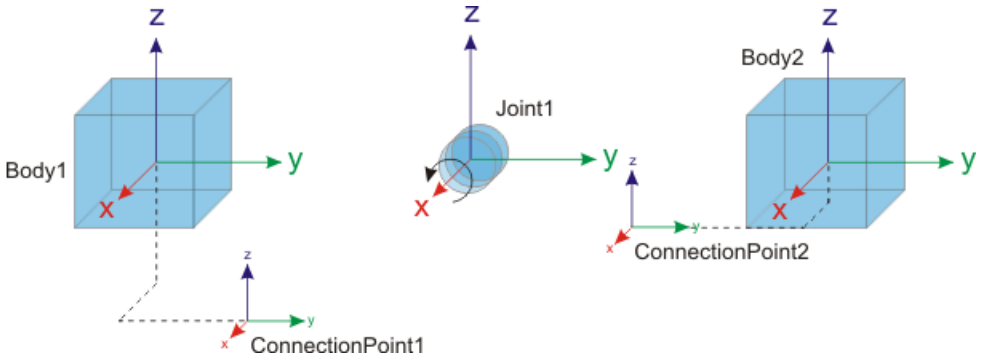
In order to make two bodies move with respect to each other, we have to connect them by joints. Joints do not have any mass or rotational inertia. They only constrain the motion of bodies. The position of a joint is indicated by its reference frame. Because the joint position and orientation is completely determined by the bodies that are connected, you do not have to specify any position or orientation parameters. These are automatically derived when you make connections.

Connections

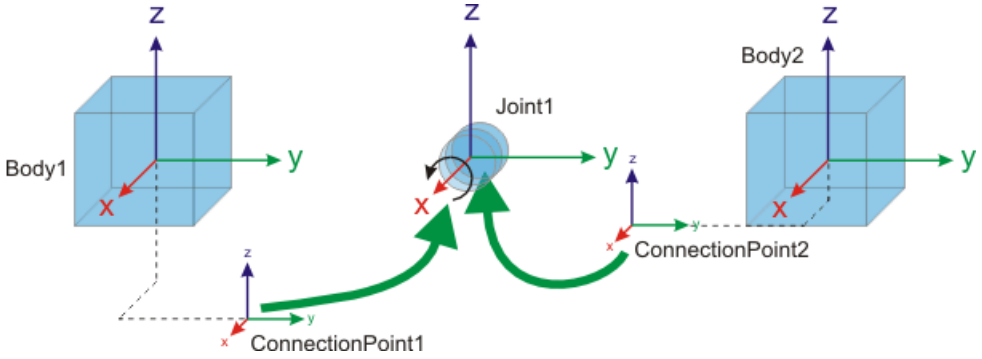
Making connections is not difficult when you understand how frames work. We will explain this with a simple example. Suppose we have two bodies and a rotational joint. The initial positions of the bodies and joints are indicated by their reference frames.



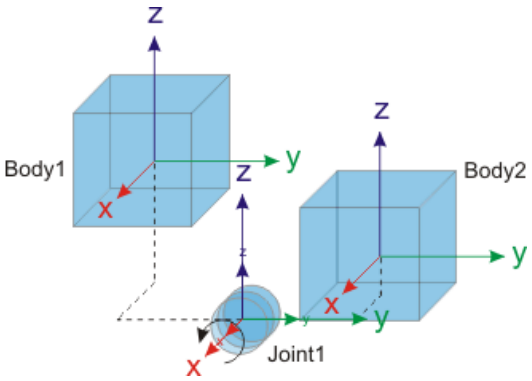
Now we have to define for each of the two bodies the offset to the joint. This can be done using connection points. A connection point is in fact a new frame with an offset from the body reference frame. As you can see, in the picture below, we have defined ConnectionPoint1 with an offset of [x = 1, y = 1, z = -1] with respect to the Body1 and we have defined ConnectionPoint2 with an offset of [x = 1, y = -1, z = -0.3] with respect to Body2.



Now the bodies and the joint can be assembled by changing the body positions in such a way that the frames of the connection points coincide with the reference frame of the joint exactly.

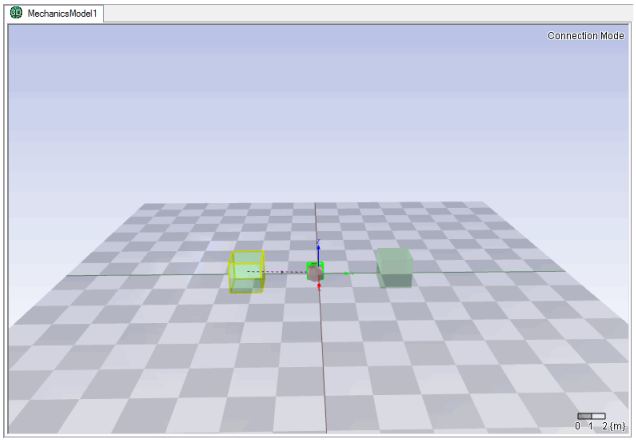


Suppose that Body1 is set as the reference body. Now the whole assembly will be moved until Body1 has its original position.

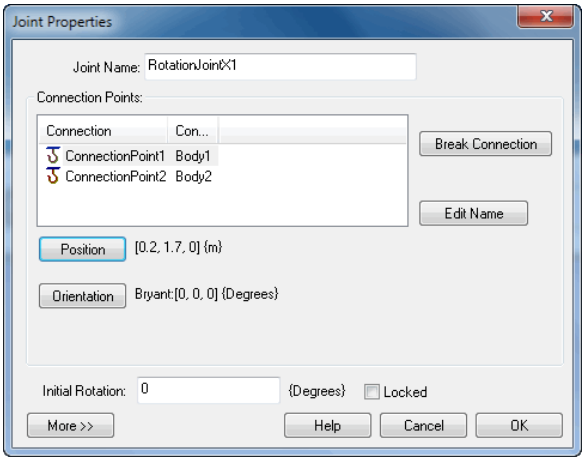


Creating Connections

In the 3D Mechanics Editor most of the assembly of joints and bodies is automated. You simply have to make connections between joints and bodies. To make a connection you have to drag and drop bodies and a joint to the Edit window. Then change to Connection mode ( ). First click a body and then the joint it should be connected to.



The *Create Connection dialog* will pop to ask you for the location of the connection point. In the connection dialog you can set the offset (both the position as well as rotation) from the body reference frame to the connection point.



The Joint Properties Dialog.

After the dialog is closed the body will automatically assemble with the joint. When two bodies are assembled with a joint, you can view their constrained motions.

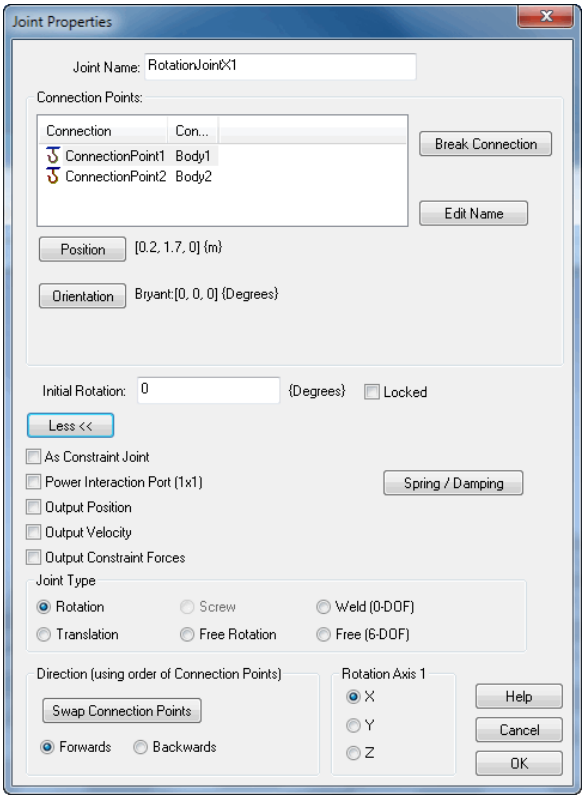
Joint Properties

When you want to change the Connection Points after the connection has been made, you have to double clicking the joint. This will open the *Joint Properties Dialog.*

You can change the name, position, orientation, and the rotation axis of the connection points. You can also change the joint type and the initial rotation or translation of the joint. If you lock the joints, it will not move when you want to show the possible motions of the system.

Advanced Properties

At the bottom of the Joint Properties you can click the *More* button to see the advanced properties.

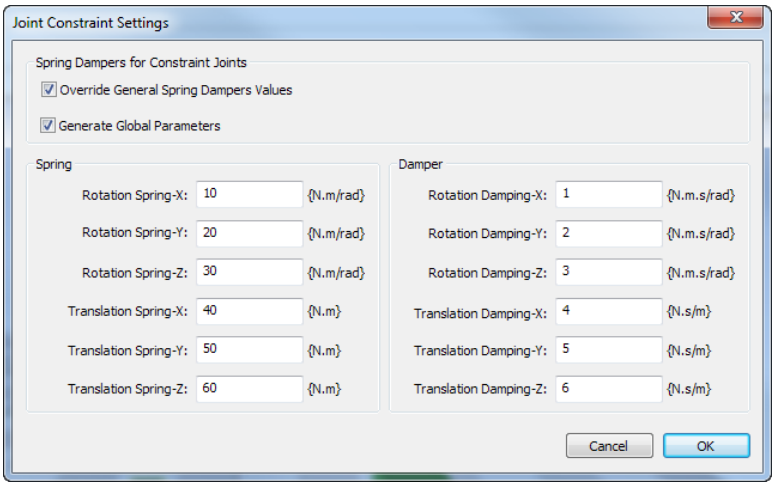


The advanced properties of the Joint Properties Dialog.

- As Constraint Joint: Calculate the joint using constraints.
- Power Interaction Port: add a power port to the joint to make it an actuated joint.
- Output Position: Create an output signal that gives the joint position.
- Output Velocity: Create an output signal that gives the joint velocity.
- Output Constraint Forces: Create an output signal that gives the forces and torques that act on the joint.

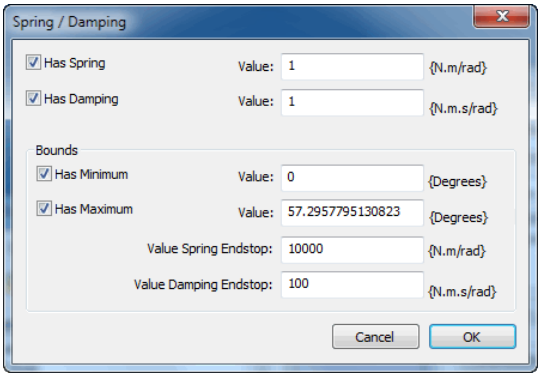
Constraint Settings

If you click the *More* button to see the *Advanced Joint Properties*, you can click the *As Constraint Joint* selection box. This box allows you to manually choose to make this joint a constraint joint. A button will be visible showing the *Constraint Settings*. if you click this button a dialog is shown allowing your to override the constraint by a spring damper. In this way you can set each individual joint by its own spring damper constants.



Spring Damper

If you click the *More* button to see the *Advanced Joint Properties*, you can click the *Spring Damper* button. This will allow you to add damping to a joint or a spring and define end-stops.



- *Has Spring*: choose this option to define a spring parallel to the joint.
- *Has Damping*: select this option to define the damping (linear with the velocity).
- *Has Minimum*: set an endstop at the minimum joint rotation/extension.
- *Has Minimum*: set an endstop at the maximum joint rotation/extension.

Joint Types

Several types of joints are available in the library:

Rotation Joints (Non-Actuated)

joint	description
XYZ-Rotation	Joints which rotate around the principal axes.
Hinge XYZ-Rotation	Same as XYZ-Rotation but with a different representation.
Balljoint XYZ-connection	Balljoints with orientations of the principal axes.

Rotation Joints (Actuated)

joint	description
XYZ-Rotation	Joints which rotate around the principal axes.
Hinge XYZ-Rotation	Same as XYZ-Rotation but with a different representation.

Translation Joints (Non-Actuated)

joint	description
XYZ-Rotation	Joints which rotate around the principal axes.
Hinge XYZ-Rotation	Same as XYZ-Rotation but with a different representation.
Balljoint XYZ-connection	Balljoints with orientations of the principal axes.

Translation Joints (Actuated)

joint	description
XYZ-Translation	Joints which rotate around the principal axes.

joint	description
XYZ-Translation	Joints which rotate around the principal axes.

Other Joints

joint	description
Weld joint	A joint that welds two bodies together.
Free Moving Joint	Opposite of the weld joint. In practice this joint will only be used to give a body a fixed starting position with respect to another body.

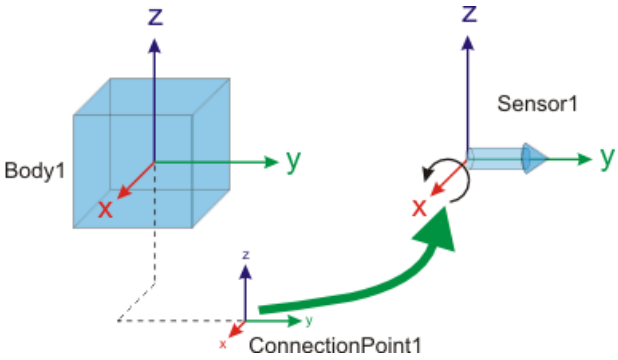
Compliance	Joint with compliance in all directions. Only valid for high stiffness and small deflections.

Sensors

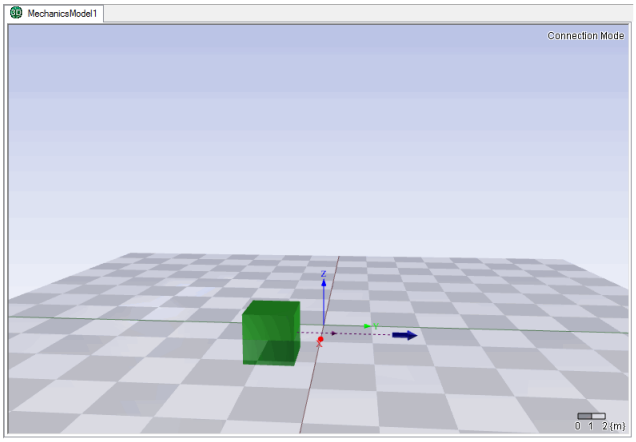
Sensors are components that indicate the position or velocity of bodies as output signals.

Connecting Sensors

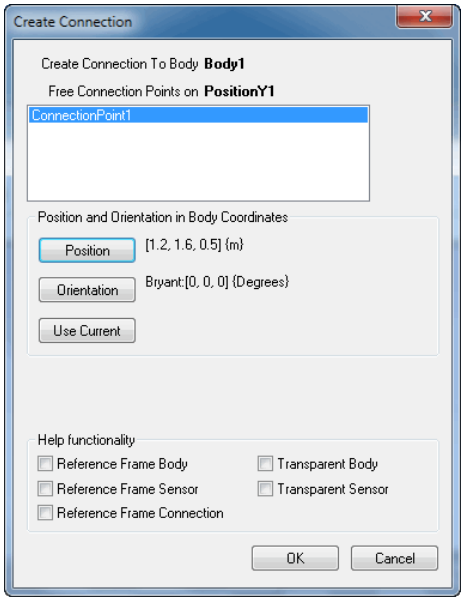
Sensors have to be connected with bodies, just like joints.



The position of the sensor is indicated by its connection point. The connection point is shown automatically when you make a connection between a body and a sensor. First change to **Connection mode** ( ), then click the body and then click the sensor.



The *Create Connection* dialog will pop to ask you for the location of the connection point. In the connection dialog you can set the offset (both the position as well as rotation) from the body reference frame to the connection point.



Sensor Types

Various sensor types are available in the library.

Position/Orientation

sensor	output signal
Position-X	A signal with the x-position of the body.
Position-Y	A signal with the y-position of the body.
Position-Z	A signal with the z-position of the body.
Position (3x1)	A signal (size 3) with the x-, y- and z-position of the body.
H-matrix (4x4)	A 4x4 signal with the H-matrix.
Rotation (3x3)	A 3x3 signal with the rotation-matrix.

Velocity

sensor	output signal
Velocity-X	A signal with the x-velocity of the body.
Velocity-Y	A signal with the y-velocity of the body.
Velocity-Z	A signal with the z-velocity of the body.
Velocity (3x1)	A signal (size 3) with the x-, y- and z-velocity of the body.

Omega-X	A signal with the rotational velocity in x-direction of the body.
Omega-Y	A signal with the rotational velocity in y-direction of the body.
Omega-Z	A signal with the rotational velocity in z-direction of the body.
Omega (3x1)	A signal (size 3) with all three rotational velocities.

Acceleration

sensor	output signal
Acceleration-X	A signal with the x-acceleration of the body.
Acceleration-Y	A signal with the y-acceleration of the body.
Acceleration-Z	A signal with the z-acceleration of the body.
Acceleration (3x1)	A signal (size 3) with the x-, y- and z-acceleration of the body.
Rotational Acceleration-X	A signal with the rotational acceleration in x-direction of the body.
Rotational Acceleration-Y	A signal with the rotational acceleration in y-direction of the body.
Rotational Acceleration-Z	A signal with the rotational acceleration in z-direction of the body.
Rotational Acceleration (3x1)	A signal (size 3) with all three rotational accelerations.

Jacobian Matrix

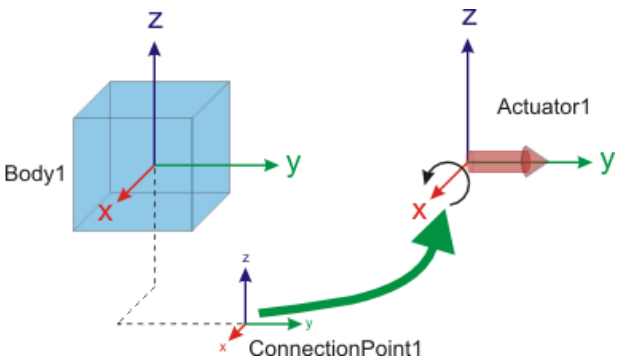
This yields the Jacobian Matrix of the connected mass.

Actuators

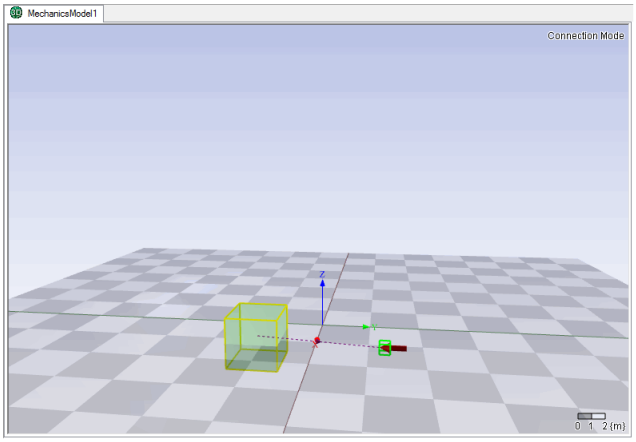
Actuators are components that impose a force or torque on bodies, where the force or torque is given by a power port or input signal.

Connecting Sensors

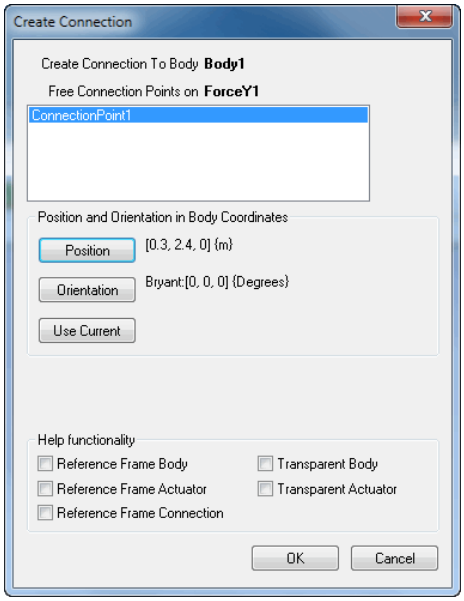
Actuators have to be connected with bodies, just like joints.



The position of the sensor is indicated by its connection point. The connection point is shown automatically when you make a connection between a body and a sensor. First change to *Connection mode* ( ), then click the body and then click the sensor.

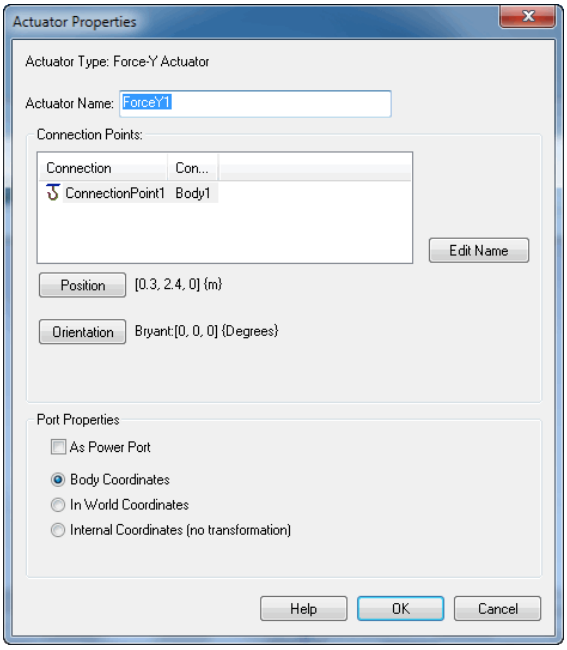


The *Create Connection dialog* will pop to ask you for the location of the connection point. In the connection dialog you can set the offset (both the position as well as rotation) from the body reference frame to the connection point.



Local or Global Coordinates

For each actuator, you can change its settings, by double clicking it. This will open the **Actuator Properties dialog**.

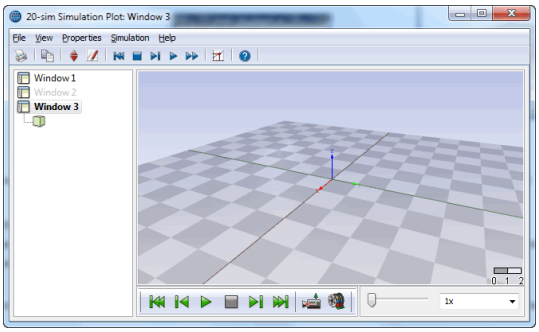


8.2 Animation Toolbox

8.2.1 3D Animation

3D Animation Window

In the *Simulator* choose *Tools and Animation* Toolbox and *3D Animation* to open a *3D Animation* window.



The 3D Animation window.

Editing

- To start editing, from the *Properties* menu select *3D Properties* . This will open 3D Properties window where you build your animation.

Running

- You can run an animation by clicking the green run button . You can also select the *Run* command from the *Replay* menu.
- At the top right of the window you see the Speed box where you can set the replay speed. The default speed is to replay an animation at real time but you can set it to play an animation faster or slower.
- If you have multiple camera's in your animation, you can switch camera, by clicking the camera button .

Ray Trace

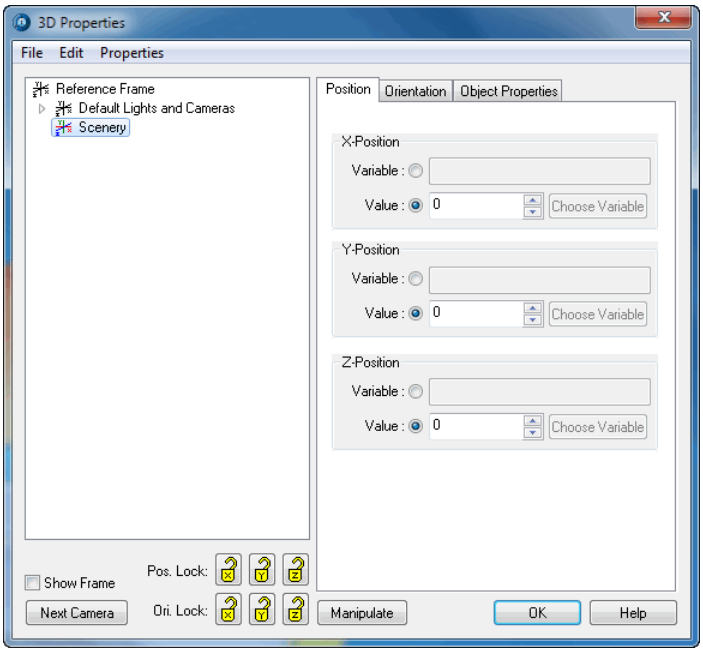
- Ray Tracing will add shadows and mirror images to a 3D Animation. From the *File* menu, select *RayTrace* to apply Ray Tracing to the current window.

Creating Movies

- From the *File* menu select *Create Movie* to create movies from your animation.

3D Properties

The *3D Properties* window allows you to define the various objects that should be shown in a 3D Animation. The 3D Properties window can be opened, by *double clicking* the mouse pointer in the *Animation* window or using the *Properties* menu.



The 3D Animation Properties window at start-up.

Menu

The menu of the 3D Animation Properties consists of three parts.

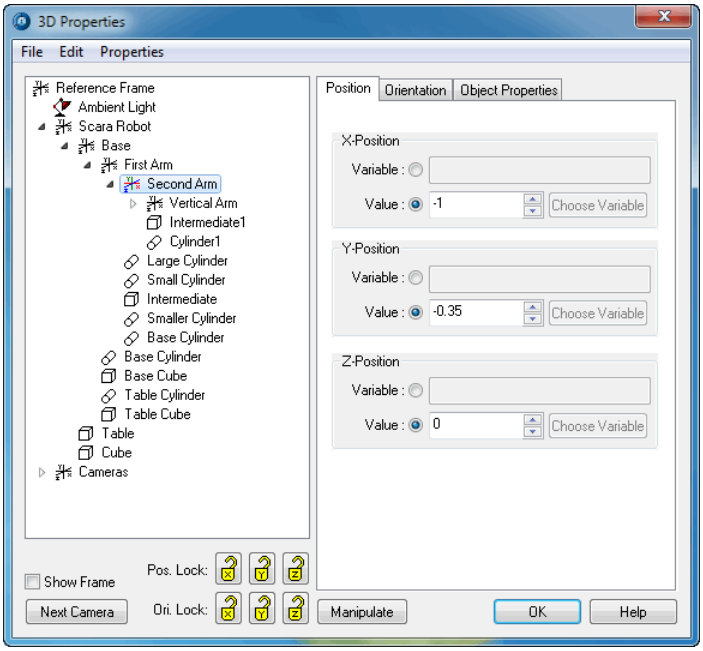
- The *File* menu allows you to store and load scenes (parts of the object tree) and create movies.
- The *Edit* menu allows you to create and edit objects.
- The *Properties* menu allows you to set the General Properties.

Lights, Cameras and Objects

Two lights and five cameras are standard available when you start a new animation. You can delete or change these lights and cameras or add new additional lights and cameras. Inserting new objects can be done with the *Edit* menu by selecting the *Insert Object* command.

Object Tree

The left side of the 3D Animation Properties window shows tree-like listing of the objects that are inserted. All objects are defined with respect to a reference frame. This can be done hierarchically by inserting new reference frames.



The 3D Animation Properties window of the model ScaraRobot.emx.

Object Attributes

On the right side of the 3D Animation Properties window shows the attributes of a selected object (e.g. position, orientation, scaling etc.). Objects can be selected, by clicking the mouse pointer on top of the object in the objects tree.

Show Frame

Each object has an object attached reference frame. This frame can be shown by selecting the object and clicking the **Show Frame** option. This is useful for manipulating objects.

Next Camera

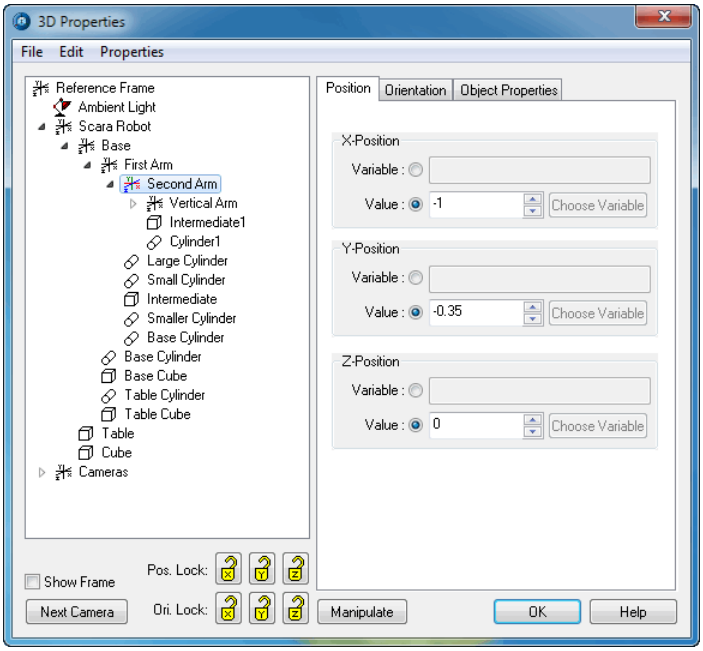
Click the Next Camera button to seen the next camera view.

Manipulate

Click the Manipulate button to move an object with use of your mouse.

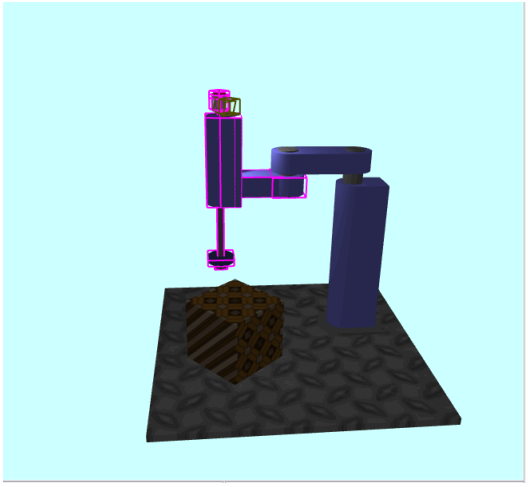
Selecting Objects

An object can be selected by pointing with the mouse in the object tree.



Select an object in the 3D Animation Properties window.

This selection is also shown in the 3D Animation window as a *pink box* around the object.

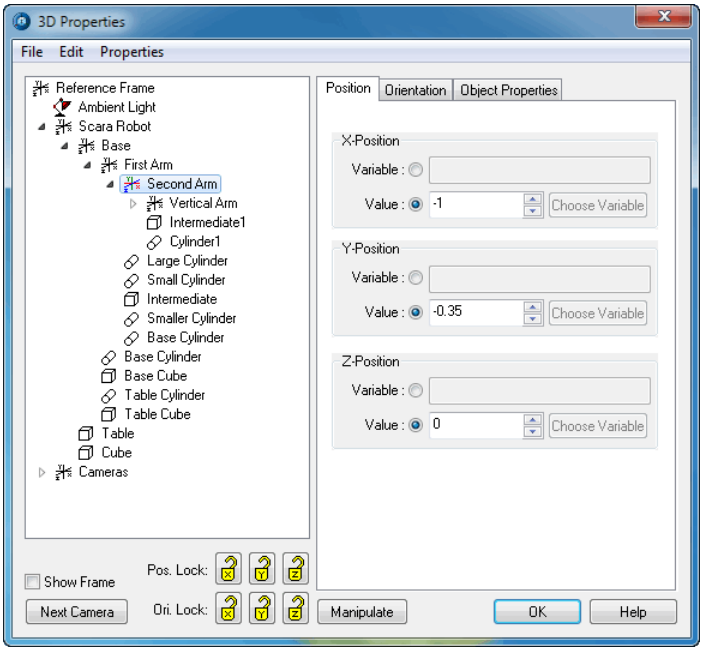


Indicating the selected object.

It is also possible to *select* an object with the mouse by *clicking* in the 3D Animation window. The selection will automatically jump to the object in the tree of the *3D Animation Properties* window.

Moving Objects

To move an object, you have select it.



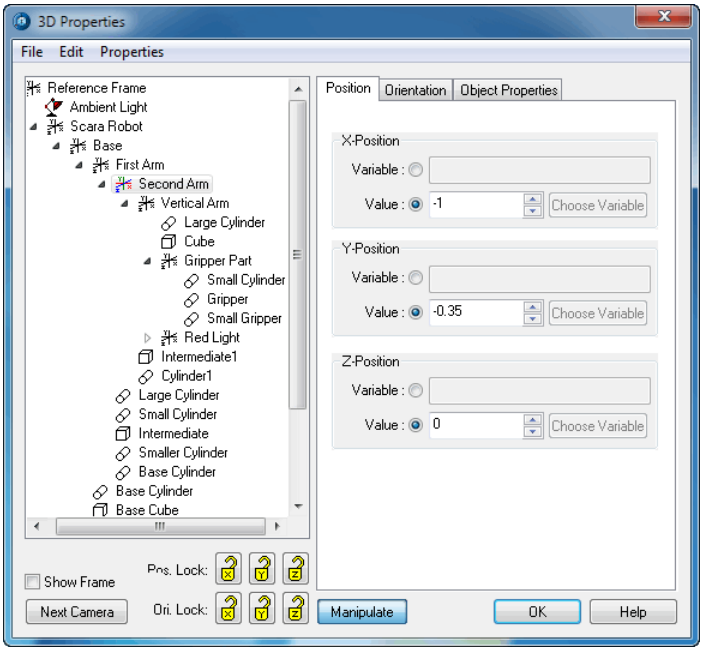
Select an object in the 3D Animation Properties window.

Attributes

The right side of the 3D Animation Properties window shows the attributes of a selected object (e.g. position, orientation, scaling etc.). You can change the position or orientation to move an object.

Manipulate

You can move an object directly by pushing the *Manipulate* button.



Manipulate an object directly.

A set of *locks* will be shown below the Manipulate button. You can use them to lock one or more directions and orientations. After this you must put the 3D Animation window on front (click on it with the mouse) and select one of the following keys:

Keys	**Action**
right arrow	move right (positive x-direction)
left arrow	move left (negative x-direction)
up arrow	move up (positive z-direction)
down arrow	move down (negative z-direction)
Ctrl + up arrow	move forward (positive y-direction)
Ctrl + down arrow	move backward (negative y-direction)
Alt+ right arrow	yaw right (positive z-axis rotation)
Alt+ left arrow	yaw left (negative z-axis rotation)
Alt+ up arrow	roll forward (negative x-axis rotation)
Alt + down arrow	roll back (positive x-axis rotation)
Ctrl+ Alt + right arrow	pitch right (positive y-axis rotation)
Ctrl+ Alt + left arrow	pitch left (negative y-axis rotation)

Keys above + Shift	Smaller steps (more accurate)
Ctrl + A	Next Camera

Scenes

In the 3D Animation Properties you can load and store scenes. Scenes are simply part of an object tree or a complete object tree. This option can be used to obtain scenes from other experiments or to create pre-defined environments.

Save Scene

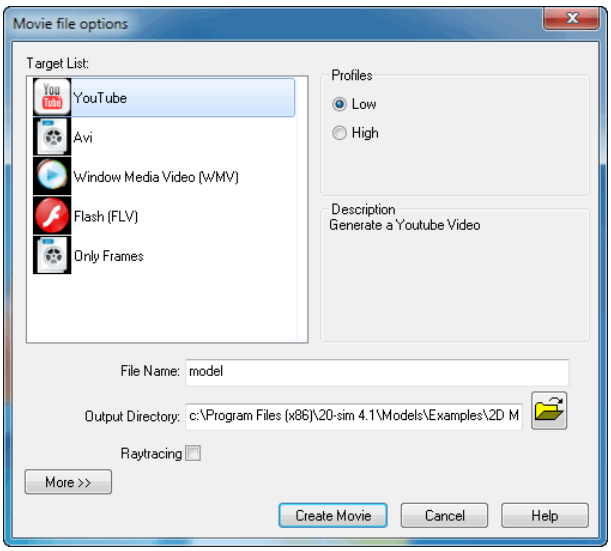
The *Save Scene* command of the *File* menu, allows you to save a scene on file. Only the selected object and all object below will be saved.

Load Scene

The *Load Scene* command of the *File* menu, allows you to open a scene from file and insert it under the selected object of the object tree.

Movies

With a few mouse clicks you can create movies from an animation. *Right click* on the 3D animation window and select *Create Movie*.



Target List

You can create movies for various purposes:

- **YouTube**: Movies in mpg-format for placement on YouTube.

- **Avi**: Movies in avi format for use in movie editing software.
- **Windows Media Video**: Movies for the use in Microsoft programs (e.g. Powerpoint).
- **Flash**: Movies in flv format for the use on Internet sites.
- **Only Frames**: For the use in advanced movie editing programs that can handle collections of pictures

Profiles

You can create movies with three profiles (low, medium, high). The corresponding sizes are shown in the table below.

	low	medium	high
YouTube	640 * 480	-	1280*720
Avi	640 * 480	1024*768	1920*1080
Windows Media Video	640 * 480	1024*768	1920*1080
Flash	640 * 360	1024*576	-
Only Frames	640 * 480	1024*768	1920*1080

Ray tracing will add shadows and mirror images to a 3D Animation. However ray tracing cost a lot of processing power. If you select the option Ray Tracing, the quality of your movie will be improved, but the time to create it will increase tremendously.

More

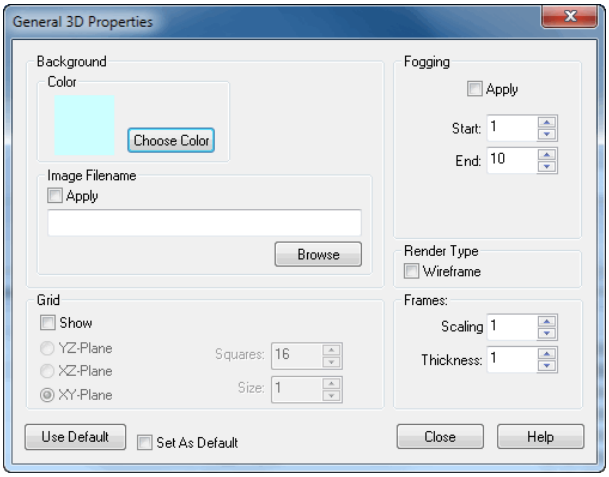
Choose the *More* button if the standard profiles are not up to your demands. You can set the size of a movie, the frame rate and speed up or slow down the speed.

Create Movie

Click the *Create Movie* button to create a movie. The movie will be stored in the output directory that you have specified using the filename that you have given.

General Properties

Using the *General Properties* command of the 3D Properties window helps you to select several options.

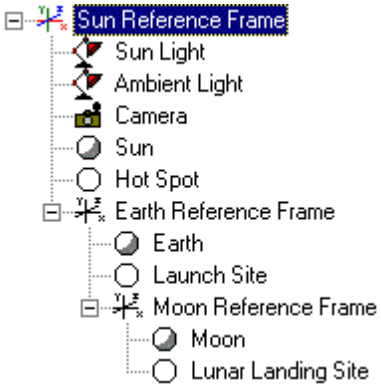


- *Background Color*: Change the background color to any desired color you like.
- *Background Image*: Use a bitmap image for the background.
- *Wireframe:* You can choose to show the animation as Wireframe.
- *Fogging:* Fog in any desired color can be added to the animation. The fog is minimal at a start plane and maximal at an end plane. You can set the distance of these planes.
- *Grid:* You can choose to show a grid.
- *Frames:* Set the frame scaling and thickness.

Objects

Reference Frames

All objects in a 3D Animation are be defined with respect to the **top reference frame**. The top reference frame is the reference frame on top of the objects tree. If you have multiple objects that move relative to the top reference frame but have a fixed position and orientation to with respect to each other, it is useful to introduce an **additional reference frame**. Define all the objects with fixed position and orientation with respect to an additional reference frame and give this frame a movement with respect to the top reference frame.



Relative reference frames.

In the objects tree frame **hierarchy** is shown by **indentation**. In the example above, the Earth Reference Frame describes the earth motion with respect to the Sun Reference frame. The Moon Reference Frame describes the moon motion with respect to the Earth Reference Frame.

Each Reference Frame has a specific position, orientation and scaling. The scaling is defined with respect to the frame one level higher in the hierarchy. In the example above, the sphere and circle describing the Earth and the Launch Site can directly be copied to form the Moon and the Lunar Landing Site. Only the scaling of the Moon Reference Frame has to be set to correct values.

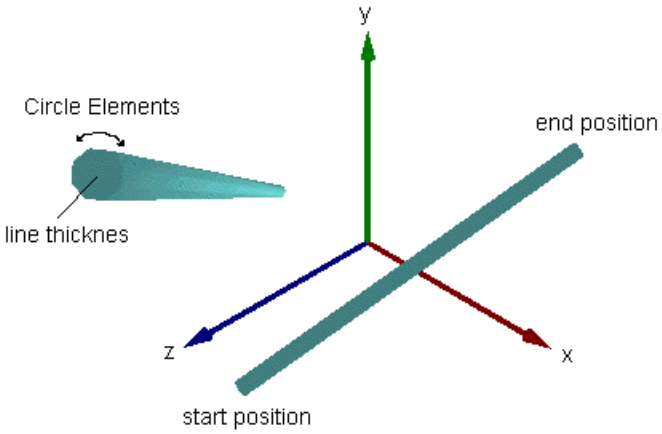
Properties (Specific)

- *Override Group Color*: You can choose to give all objects that are defined in a frame the same color.

Properties (General)

- position
- orientation
- scaling
- duplication

Line



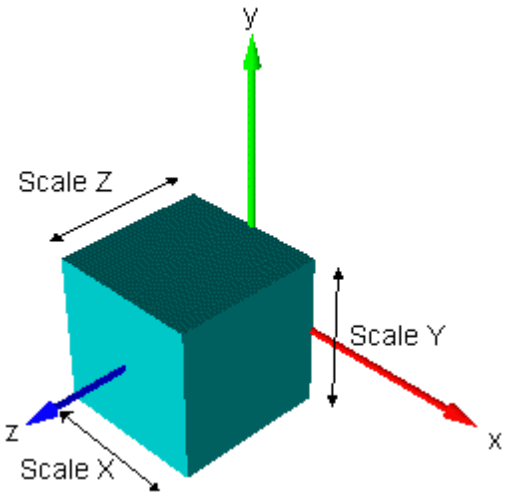
Properties (Specific)

You can set the start position and end position of the line, enter the line thickness and set the number of circle elements.

Properties (General)

- color
- mesh
- texture
- material
- duplication

Cube



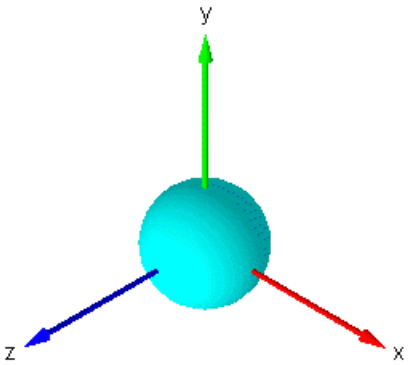
Properties (Specific)

You can set the position of the origin and the rib length and choose to show the inside.

Properties (General)

- position
- orientation
- scaling
- color
- mesh
- texture
- material
- duplication

Sphere



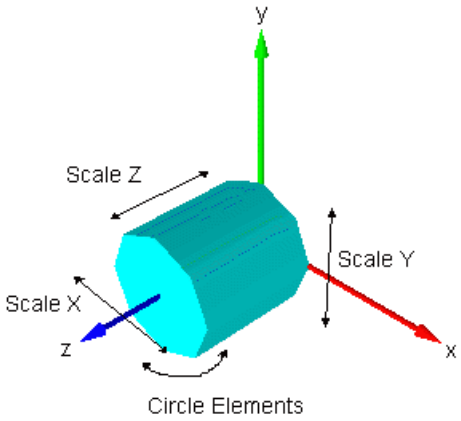
Properties (Specific)

You can choose to **show the inside and outside**. The number of segments that are used to draw the circle is fixed.

Properties (General)

- position
- orientation
- scaling
- color
- mesh
- texture
- material
- duplication

Cylinder



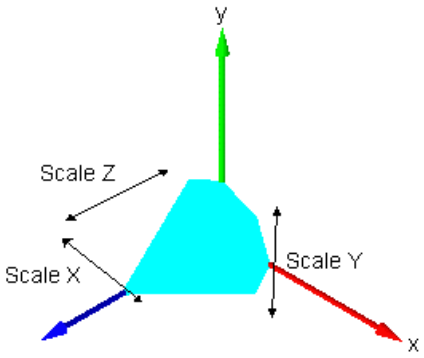
Properties (Specific)

You can set the position of the **origin**, choose to **show the inside and outside**, set the number of circle elements. A cylinder can be shown with **open or closed sides**.

Properties (General)

- position
- orientation
- scaling
- color
- mesh
- texture
- material
- duplication

Cone



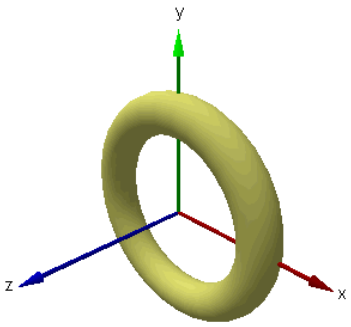
Properties (Specific)

You can set the number of circle elements and choose to show the cone with **an open or closed side**.

Properties (General)

- position
- orientation
- scaling
- color
- mesh
- texture
- material
- duplication

Torus



Properties (Specific)

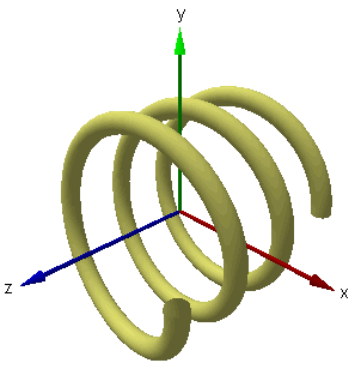
A Torus has some specific options:

- The *Torus Radius* denotes the length from the origin to the center of the tube.
- The *Tube Radius* denotes the tub thickness.
- With the *Number of Segments* you can set the total number of segments for one revolution.
- The *Points Per Segment* are the number of circle elements for each segment.

Properties (General)

- position
- orientation
- scaling
- color
- mesh
- texture
- material
- duplication

Spiral



Properties (Specific)

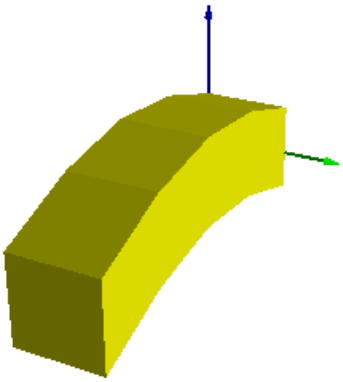
- The *Height* is the the length of the spiral.
- The *Spiral Radius* is the length from origin to the center of the tube.
- The *Tube Radius* denotes the tub thickness.
- The *Number of Turns* is the number of revolutions from bottom to top of the spiral. This may be a non-integer value.
- The *Number of Segments* is the total number of segments from bottom to the top of the spiral.
- The *Points Per Segment* are the number of circle elements for each segment.

- You can set the position of the origin.

Properties (General)

- position
- orientation
- scaling
- color
- mesh
- texture
- material
- duplication

Beam



Properties (Specific)

- *Dimensions*: You can choose the *Segment Length*, *the Width* and *Height* of the beam.
- *Variable:* The beam offset (in the direction of the height) is given by a vector. The number of segments is equal to the number of vector elements. You can choose the vector as a variable from your model.

Properties (General)

- position
- orientation
- scaling
- color
- mesh
- material
- duplication

3D-files

You can import object from files, using the 3D-files object. As soon as you insert a 3D-files object, 20-sim will open a file browser. Three formats are supported

- *STL-files (*.stl):* The STL file format or stereolithography format is another widely accepted interchange format between CAD packages.
- *DXF-files (*.dxf):* This is AutoCAD's drawing interchange format. The DXF file format was originally designed by AutoDesk® to represent 3D models built with AutoCAD®. Now they are generally accepted as an interchange format between CAD packages.
- *Lightwave Object files (*.lwo):* LightWave is a software package used for rendering 3D images, both animated and static.

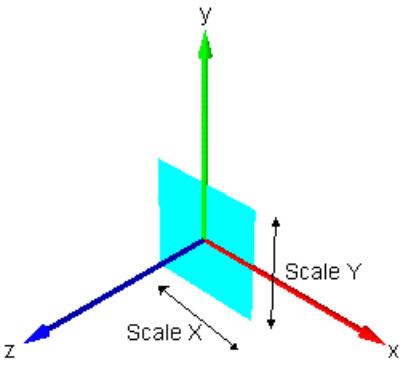
Properties (Specific)

- Use the Browse button to load another file.
- Set the **Override Color** check box to give the object any desired color.
- You can set the position of the **origin**.

Properties (General)

- position
- orientation
- scaling
- color
- mesh
- texture
- material
- duplication

Square



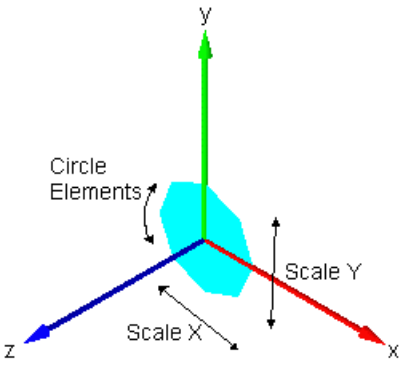
Properties (Specific)

- You can set the position of the *origin*.

Properties (General)

- position
- orientation
- scaling
- color
- mesh
- texture
- material
- duplication

Circle



Properties (Specific)

- You can set the number of circle elements.

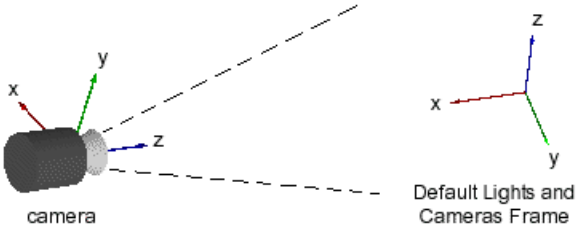
Properties (General)

- position
- orientation
- scaling
- color
- mesh
- texture
- material
- duplication

Camera

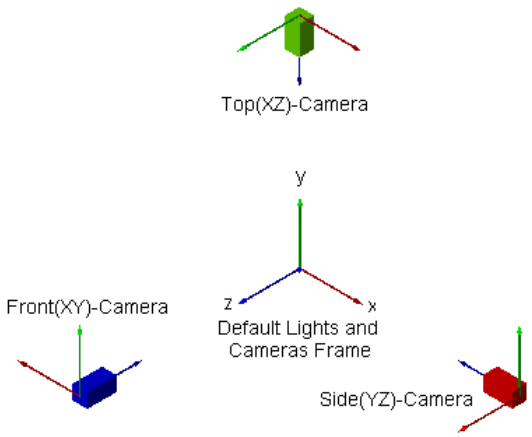
All objects in a 3D Animation are defined with respect to the *Default Lights and Cameras* frame. This frame coincides with the top *Reference Frame*. When you start up a 3D animation 5 default cameras are available showing several views of the origin of this frame.

Two cameras show the *Default Lights and Cameras* frame from a position of $x = 5$, $y = 5$ and $z = 10$. The cameras are pointed to the center of origin of the frame. The *Camera Looking At Origin* will keep looking at the origin, even if its position is changed. The *Perspective Camera* will not keep looking at the origin but keep its original orientation when its position is changed.



The Camera Looking At Origin and Perspective Camera.

Three *Front* cameras are positioned on top of each of the axes of the *Default Lights and Cameras* frame. The front cameras do not show perspective. I.e. they can be used for 2D-Animation.



Properties (Specific)

Camera Properties

- *Plane Distance*: A camera only shows objects that lie between a front plane and a back plane. You can set the distance of these planes (seen from the camera).
- *Projection*: You can show all objects in perspective (objects further away look smaller) or Orthographic (objects further away are shown equally large). You can choose to show right hand frames or left-hand frames.
- Use the slider bar to set the *Zooming*.

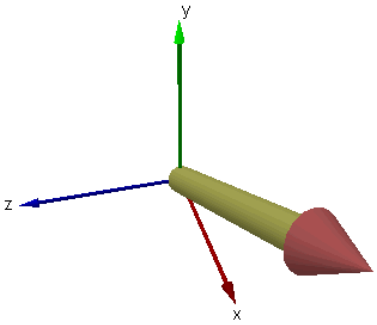
LookAtProperties

- When you enable the *Look at Position*, the camera will always look at a specific position, regardless the position of the camera itself
- You can make the look at position *static* or connect it to *variables* of your model.
- Use the *Always Look up Vector* to let the camera have a fixed angle of view.

Properties (General)

- position
- orientation

Vector



Properties (Specific)

The Vector object is a combination of a Cylinder object and a Cone object. The length of the Vector can be constant or determined by the value of the variable, the offset and the multiplication factor. You can set the vector diameter, the cone diameter and the cone height and set the number of circle elements. The color of the Cone and the Cylinder can be specified separately by the two color property pages. The first page is for the Cylinder and the second page is for the Cone.

Properties (General)

- position
- orientation
- mesh
- texture
- material
- duplication

Ambient Light

An Ambient Light source illuminates everything with a predefined color, regardless of the orientation, position, and surface characteristics of the objects available. It is constructed by making a spot light shine everywhere.

Spot Light

A Spotlight emits a cone of light of a certain color. The cone starts at a given position and the direction is given by the orientation. Only objects within the cone are illuminated.

Color

- Ambient color: This is the main color of point light. For a spot light this option is turned off.
- Diffuse color: This is color of point light when it shines on matt surfaces.
- Specular color: This is color of point light when it is reflected in shiny surfaces.

Direction

You can select the shape of the light bundle.

- Parallel: Select this option to make the light a parallel bundle (e.g. the sun light). For a spot light this option is turned off.
- Spot Exponent [Wide-Narrow]: Select this option to make the light intensity equal in all directions (Wide) or more intensive in the center beam of the spot light (Narrow)
- Spot Angle [0..90]: Select the angle of the cone from 0 degrees (very thin bundle) to 90 degrees (light in all directions).

Attenuation

Light attenuation defines how the intensity of the light weakens as it travels away from the source. The attenuation is computed based on the following formula:

$$att = \frac{1}{k_0 + k_1 d + k_2 d^2}$$

where *k0* is the constant attenuation, *k1* is the linear attenuation, *k2* is the quadratic attenuation and *d* is the distance from the light's position to an object.

Directional Light

A *Directional Light* illuminate all objects with light of equal intensity, as if it were at an infinite distance from the objects. The directional source is commonly used to simulate distant light sources, such as the sun. It is constructed by making a spot light shine in parallel bundles.

Properties

Position

Most objects in a 20-sim Animation can be positioned with respect to the frame one level up in the object tree. This position can be fixed (using fixed values) or change (using model variables) during simulation.

Orientation

All objects in a 3D Animation have an orientation. This orientation can be made visible by showing the object attached reference frame. This orientation can be fixed (using fixed values) or changed (using model variables) during simulation.

The orientation of an object is defined as the transformation of the frame one level up the object tree {A} to the object attached reference frame {B}. Several methods can be used in 20-sim to define this transformation.

DirectX

Using the DirectX method you have to specify the Y-axis and Z-axis of the frame {B} in coordinates of frame {A}.

Bryant angles

Start with the frame {B} coincident with a known frame {A}. First rotate {B} about the X-axis of {B} by an angle of X (rad), then rotate about the Y-axis of {B} by an angle of Y (rad) and then rotate {B} about the Z-axis of {B} by an angle of Z (rad). **Note:** Bryant angles are also known as X-Y-Z Euler angles or Cardan angles.

Z-X-Z-Euler

Start with the frame {B} coincident with a known frame {A}. First rotate {B} about the Z-axis of {B} by an angle of X (rad), then rotate about the X-axis of {B} by an angle of Y (rad) and then rotate {B} about the Z-axis of {B} by an angle of Z (rad).

Euler Parameters

Start with a frame {A}. Rotate this frame around a vector K [X,Y,Z]T about an angle theta to get to the frame {B}.

Scaling

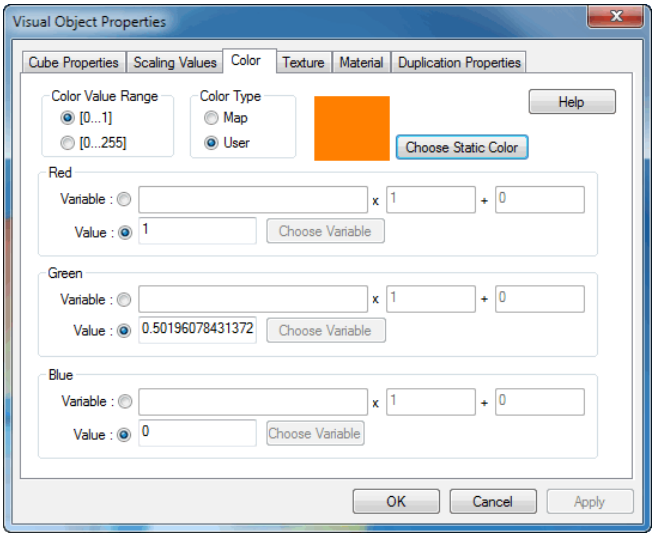
Most objects in a 20-sim Animation can be scaled in size. This scaling can be fixed (using fixed values) or change (using model variables) during simulation.

Color

Most objects in a 20-sim Animation can be shown with a preset color. The colors can be user defined or defined with respect to a color map.

User Defined

The additive primary colors are red (R), green (G), and blue (B). By mixing these colors in different percentages, any other color can be created. When blue and green are mixed, the resulting color is cyan. When blue and red are mixed, the resulting color is magenta. If all three primary colors are mixed together, the resulting color is white.

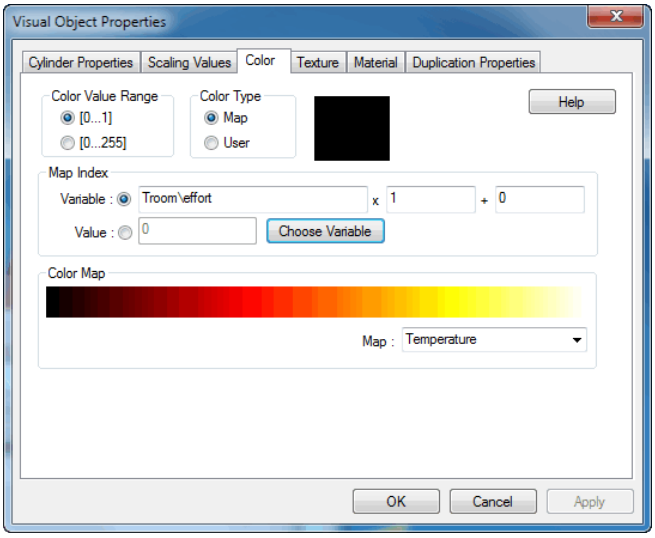


The color tab.

Color ranges can be set from 0 to 1 or 0 to 255. Constant values or model variables can be chosen for each primary color. Use the multiplication and addition terms for scaling and offset.

Color Map

The color that is chosen is defined by a color map. Color ranges can be set from 0 to 1 or 0 to 255. A constant value or a model variable can be chosen. Use the multiplication and addition term for scaling and offset.

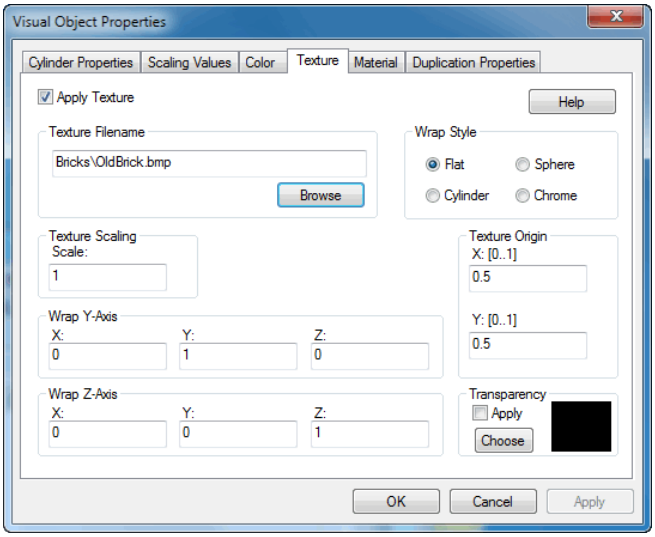


Mesh

The rendering option defined in the General Properties can be overridden for each object. The options are: WireFrame, Unlit Flat, Flat and Gouraud.

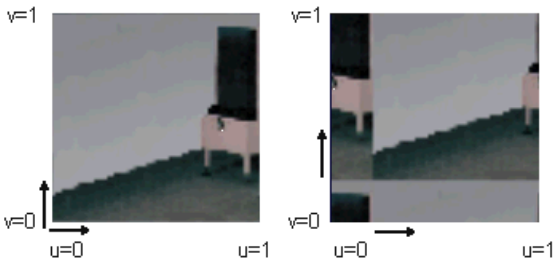
Texture

Using the *Texture* attribute, you can wrap a bitmap file around an object. Using the browse button, you can easily select the desired bitmap file (bmp, jpg, gif, tif, etc.). Using the *transparency* option you can select a color in the bitmap that should be transparent.



Texture Origin

A texture is defined in generalized coordinates (u,v). The bottom right of the texture has coordinates (0,0) and the top left (1,1). The offset parameters define how much offset is used to project the bitmap. This is shown in the figure below. At the left an offset of (0,0) is shown and at the right an offset (0.2,0.2) is shown.



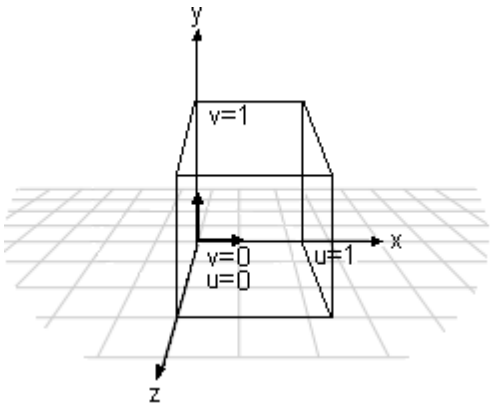
Depending on the type of texture wrapping (Flat, Cylinder, Sphere or Chrome) the three wrapping vectors () have a different meaning. This is explained in the next sections. To facilitate the explanation in the pictures below the origin vector equals (0,0,0), the Y-vector equals (0,1,0) and the Z-vector equals (0,0,1).

Vectors

Using three vectors (Origin, Y-Axis and Z-Axis) and Offset parameters (X, Y) you can define how the texture should be wrapped around an object. The vectors are defined with reference to the object attached reference frame. The object attached reference frame can be made visible by selecting Show Frame. The vectors use generalized coordinates. I.e. the length, width and height of the object are defined as 1 and the vectors are scaled accordingly.

Flat

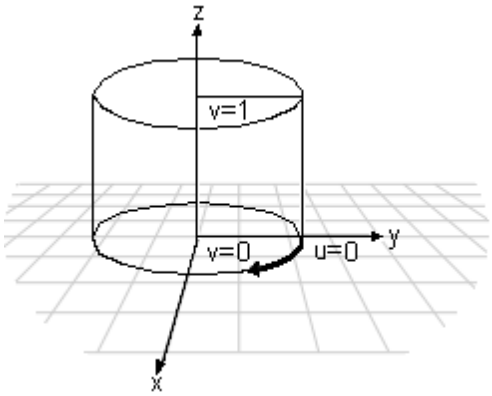
The flat wrap conforms to the faces of an object as if the texture were a piece of rubber that was stretched over the object. For a flat texture wrap, the effects of the various vectors are shown in the following illustration.



- The *Origin* vector defines the base from which the texture wrapping is performed.
- The *Y-vector* equals the v axis of the texture wrap.
- The *Z-vector* is always perpendicular to the texture wrap.

Cylindrical

The cylindrical wrap treats the texture as if it were a piece of paper that is wrapped around a cylinder so that the left edge is joined to the right edge. The object is then placed in the middle of the cylinder and the texture is deformed inward onto the surface of the object. For a cylindrical texture wrap, the effects of the various vectors are shown in the following illustration.



- The *Origin* vector defines the base from which the texture wrapping is performed.
- The *Y-vector* specifies the point on the outside of the cylinder where *u* equals 0.
- The *Z-vector* specifies the axis of the cylinder.

Spherical

For a spherical wrap, the u-coordinate is derived from the angle that the vector [*x y* 0] makes with the x-axis (as in the cylindrical map) and the v-coordinate from the angle that the vector [*x y z*] makes with the z-axis. Note that this mapping causes distortion of the texture at the z-axis.

- The *Origin* vector defines the base from which the texture wrapping is performed.
- The *Y-vector* specifies the point on the outside of the sphere where *u* equals 0.
- The *Z-vector* specifies the point on the outside of the sphere where *v* equals 0.

Chrome

A chrome wrap allocates texture coordinates so that the texture appears to be reflected onto the objects. The chrome wrap takes the reference frame position and uses the vertex normals in the mesh to calculate reflected vectors. The texture u- and v-coordinates are then calculated from the intersection of these reflected vectors with an imaginary sphere that surrounds the mesh. This gives the effect of the mesh reflecting whatever is wrapped on the sphere.

Material

A material defines how a surface reflects light. A material has two components: an emissive property (background color) and a specular property (reflected light), whose brightness is determined by a power setting.

Specular Highlights

The value of the power determines the sharpness of the reflected highlights, with a value of 5 giving a metallic appearance and higher values giving a more plastic appearance.

Emissive Color

The emissive color is the background color of the object. If you choose the color white the object looks like a light bulb. If you choose the color black to object does not radiate any background color.

Closed Shapes

Some objects in a 20-sim Animation can be shown as a closed shape (sides closed) or open shape (sides open).

Circle Elements

All objects in a 20-sim Animation are build out of a set of planar surface elements. Curved objects have to be build up out of a lot of surface elements to give the impression of a smooth surface. For some objects you can set the amount of surface elements used to create circular surfaces.

Attenuation

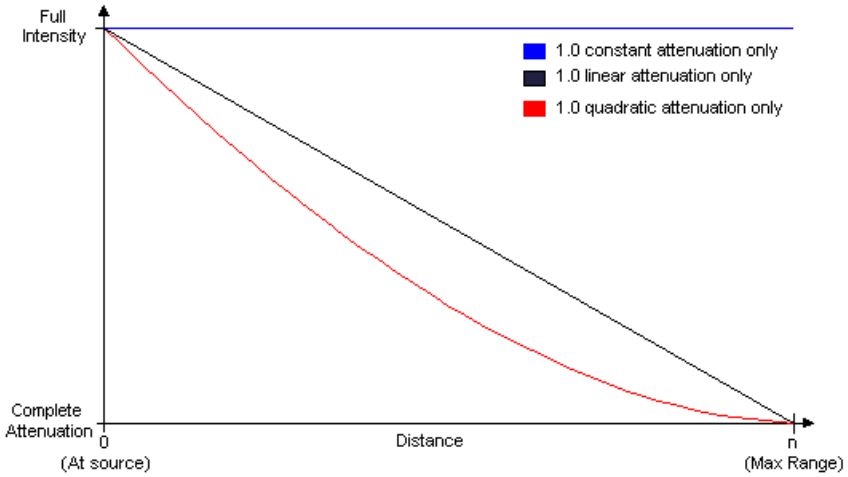
The Animation Engine in 20-sim is based on the Microsoft® Direct3D. Direct3D uses the following formula to normalize the distance from a light source to the object surface into a value from 0.0 to 1.0, inclusive:

$$D_n = \frac{R - D}{R}$$

In the preceding formula, *Dn* is the normalized distance, *R* is the light's range, and *D* is the distance, in world space, from the light source to the object surface. A normalized distance is 1.0 at the light's source, and 0.0 at the light's range.
With the normalized distance in hand, Direct3D then applies the following formula to calculate light attenuation over distance for point lights and spotlights:

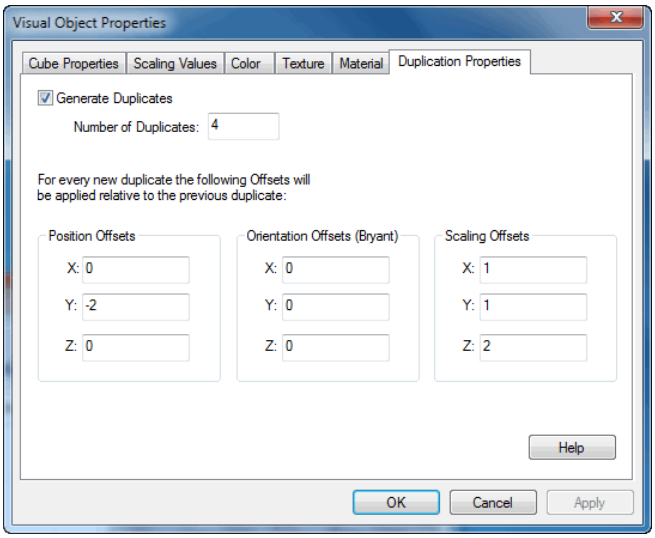
$$A = Attenutation_{constant} + D_n \cdot Attenutation_{linear} + {D_n}^2 \cdot Attenutation_{quadratic}$$

In this attenuation formula, *A* is the calculated total attenuation and Dn is the normalized distance from the light source to the object. The constant, linear and quadratic attenuation factors act as coefficients in the formula. You can produce a wide variety of attenuation curves by making simple adjustments to them. Most applications will set the linear attenuation factor to 1.0 and set the remaining factors to 0.0 to produce a light that steadily falls off over distance. Similarly, you could apply a constant attenuation factor of 1.0 by itself to make a light that doesn't attenuate (but will still be limited by range). The following illustration shows the three most common attenuation curves.



Duplication

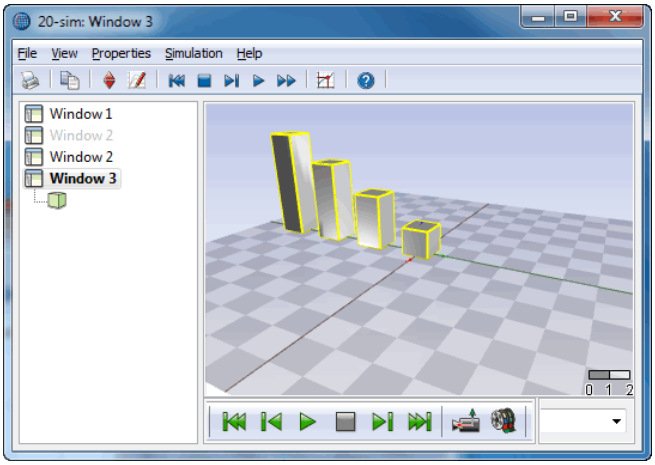
All objects in a 20-sim Animation can be duplicated. This property is useful when drawing a lot of the same objects.



You can choose several options:

- *Number of Duplicates*: Select the number of duplicates of the object.
- Position Offsets: Select the position offset of the first duplicate from the original object. This offset is repeated for every next duplicate.
- *Orientation Offset*: Select the orientation offset of the first duplicate from the original object. This offset is repeated for every next duplicate.
- *Scaling Offset*: Select the scaling of the first duplicate compared to the original object. This scaling is repeated for every next duplicate.

The example below shows a cube that is duplicated according to the specifications given in the figure above. As shown, the first duplicate is a little larger than the original object. The second duplicate is a little larger than the first duplicate etc.

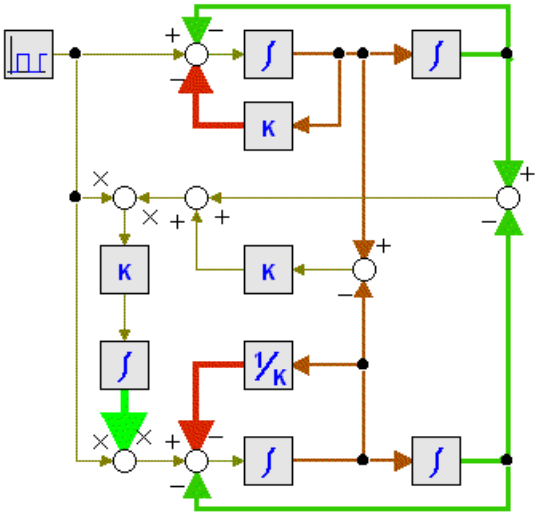


8.2.2 Graph Animation

Graph Animation

To Animate the results of a simulation in the graphical model select the *Graph Animation* command from the *Tools* menu (Simulator). During simulation, the thickness and color of bonds and signals will correspond with the values they carry.

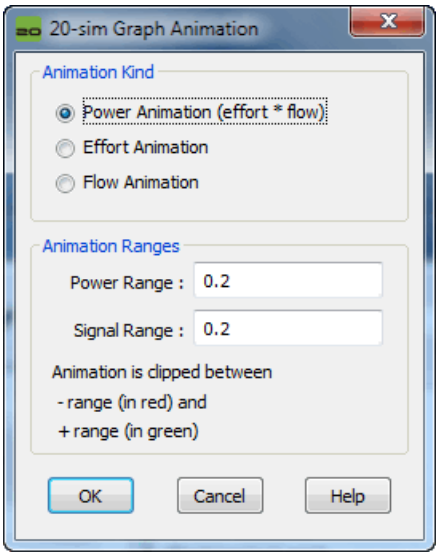
The color of a bond or signal always corresponds with the sign, red for negative values and green for positive values. The thickness of signals corresponds with the value. For signals that is obvious but for bonds or connections carry two variables: effort and flow (also know as across and through). For bonds and connections you can choose to let the thickness correspond with the effort, flow or power (effort*flow), depending on the Animation command that was chosen.



You can start Animation in the Simulator:

1. Run a simulation until you are satisfied with the results.
2. From the **Tools** menu, select **Animation Toolbox** then **Graph Animation**.
3. A menu pops up. Choose the options you like and click **OK** to start Animation.

Graph Animation Menu



Items

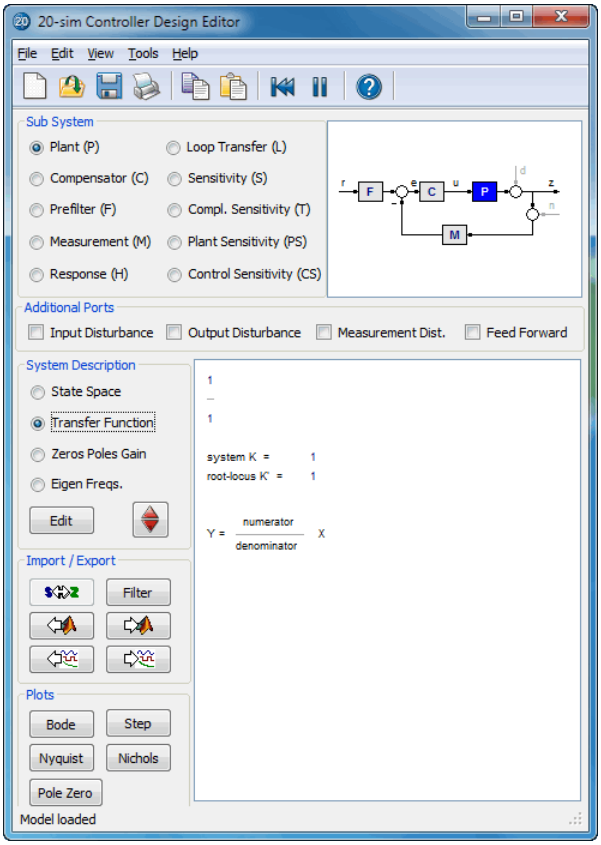
- *Animation Kind*: For signals, the thickness always corresponds with their value. For bonds and connections you can choose. Select here to what the thickness of bonds and connections should represent: effort, flow or power (effort*flow).
- *Animation Ranges*: The thickness of bonds and signals can vary from 1 to 5 pixels. Select here the range that should correspond with a thickness of 5 pixels.

8.3 Control Toolbox

8.3.1 Controller Design Editor

Controller Design Editor

The 20-sim *Controller Design Editor* allows you to enter or edit linear time-invariant models of a plant with control loop. The control loop is separated in a compensator, prefilter and measurement. You can open the *Controller Design Editor* from the *Tools* menu of the 20-sim Editor or by clicking Go Down on a ControlledLinearSystem model that was inserted in the Editor.



Sub System

- You can choose to show a single element in the loop:
  - **Plant (P), Compensator (C), Prefilter (F) or Measurement (M)**
- or view the response of the system:
  - **Response (H), Loop Transfer (L), Sensitivity (S), Compl. Sensitivity (T), Plant Sensitivity (PS) or Control Sensititvity (CS).**

The corresponding transfer function is show in the editor.

Additional Ports

Select one of the radio buttons **Input Disturbance**, **Output Disturbance**, **Measurement Dist**. or **Feedforward**. The first three options allow you to use external disturbance in your model. The last option allows you to use an external feedforward controller (e.g. a B-Spline network). The selected inputs are shown in the picture.

System Description

- Pressing the ***Edit*** button will open a dialog for editing the linear system. Each description (State Space, Transfer Function, Zeros Poles Gain or Eigen Frequency) has a special dialog and can be used to specify continuous-time and discrete-time systems.
- In the 20-sim Simulator, out of an existing (non-linear) model a symbolic linear-system can be derived by means of linearization. This means that the relevant model parameters are preserved in the Linear System and the parameters button can be used to change parameters.

Import/Export

- Pressing the **s<->z** button will transform a continuous-time linear system into a discrete-time linear system and back.
- Pressing the **Filter** button opens the Filter Editor, where a filter can be designed. This filter can then be combined with the current linear system or replace the current linear system. If the linear system is a discrete-time system, the designed analog filter is automatically transferred into its digital equivalent.
- Pressing the **From Matlab** button and the **To Matlab** button allows for an instant exchange of the linear system with the Matlab workspace. Information is transferred numerically (no parameter relations are preserved)
- The button is only active when a Linear System has been imported through linearization. Clicking the button will import parameters from the simulation.
- The button is only active when a Linear System has been imported through linearization. Clicking the button will export the current parameters to the simulation.

Plots

You can inspect the time- and frequency responses of the selected transfer function using:

- Step Response
- Bode Plot
- Nyquist Plot
- Nichols Chart
- Poles and Zeros (including root locus)

Output

- Clicking the **20-sim** button will export the linear system as a new 20-sim submodel.
- Clicking the **Matlab** button will export the 20-sim Linear System to a Matlab m-file. If the Linear System is symbolic, all parameter relations are preserved.

State Space Models

State space models use linear differential equations (continuous) or difference equations (discrete) to describe system dynamics. They are of the form:

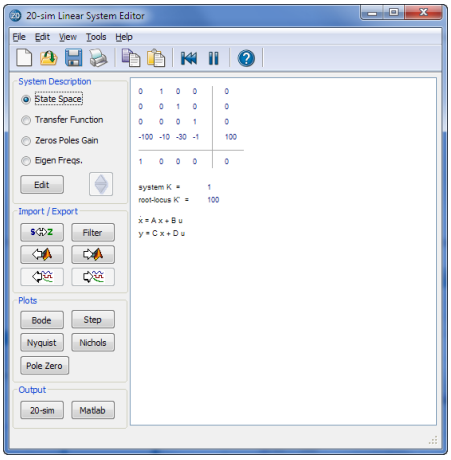
$$\frac{dx}{dt} = A \cdot x + B \cdot u \qquad x_{n+1} = A \cdot x_n + B \cdot u_n$$
$$y = C \cdot x + D \cdot u \qquad y_n = C \cdot x_n + D \cdot u_n$$

(continuous) (discrete)

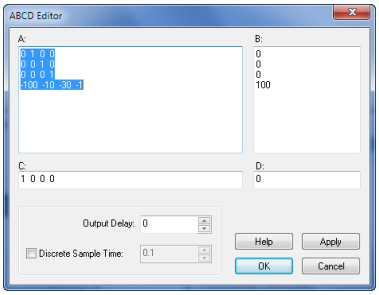
with for example:

$$A = \begin{bmatrix} 0 & 0.5 \\ -1 & -1 \end{bmatrix} \quad B = \begin{bmatrix} -0.1 \\ 0.1 \end{bmatrix}$$
$$C = \begin{bmatrix} 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 \end{bmatrix}$$

where x is the state vector and u and y are the input and output vectors. These models may arise from the equation of physics, from state space identification or as the result of linearization.



You can enter state space models by selecting the *State Space* button and clicking the *Edit* button.



This opens an editor in which you can enter the A, B, C and D matrices. Depending on the selection of the Discrete Linear System check box (and Sample Time) the system is a continuous-time or discrete-time system. You can enter the matrix elements in the white space areas. Separate column elements with Spaces or Commas. Enter new rows by clicking the Enter key (new line) or using a semicolon. Brackets (e.g. [...]) may be used to denote the a matrix or vector.

The A-matrix, shown in the figure above can for example be entered as:

0 1
-0.5 -1

or

0,1;-0.5,-1

or

[0,1;
-0.5,-1]

Output Delay

To inspect the effects of time delay in your model, you can add output delay. The result will be visible in the various plots that you can show of a linear system. The unit of the output delay is seconds.

Discrete Sample Time

If you want to transfer a linear system from continuous time to discrete time directly (i.e. replace the s by a z), select Discrete Sample time and fill in the sample time value. You can also transfer back directly by deselecting Discrete Sample time.

Commands

- *Help*: Open the help file.
- *Apply:* Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero).
- *OK*: Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero) and close the editor.
- *Cancel*: Do not apply any changes to the system and close the editor.

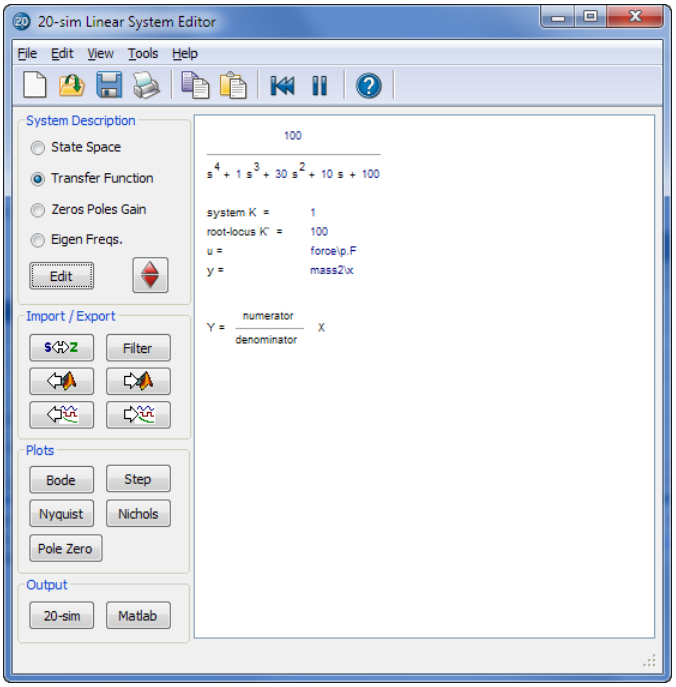
Transfer Functions

A continuous time or discrete time SISO transfer function:

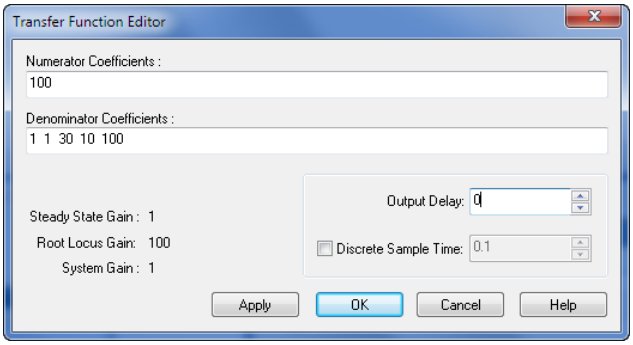
$$h(s) = \frac{n(s)}{d(s)} \qquad h(z) = \frac{n(z)}{d(z)}$$

(continuous) *(discrete)*

is characterized by its numerator *n* and denominator *d*, both polynomials of the variable *s* or *z*.



You can enter transfer functions by selecting the *Transfer* Function button and clicking the *Edit* button.



This opens an editor in which you can enter the coefficients of the numerator and denominator polynomials. You can enter the elements in the white space areas. Separate the elements with Spaces. Polynomials should be entered in descending powers of s or z. The Steady State Gain, Root Locus Gain and System Gain are parameters, that are automatically derived from the transfer function.

In the editor shown above the transfer function

$$\frac{0.1 \cdot s + 0.1}{s^2 + s + 0.5}$$

was given. You can enter the coefficients as:

0.1 0.1

and

1 1 0.5

or

0.1,0.1

and

1,1,0.5

Output Delay

To inspect the effects of time delay in your model, you can add output delay. The result will be visible in the various plots that you can show of a linear system. The unit of the output delay is seconds.

Discrete Sample Time

If you want to transfer a linear system from continuous time to discrete time directly (i.e. replace the s by a z), select Discrete Sample time and fill in the sample time value. You can also transfer back directly by deselecting Discrete Sample time.

Commands

- *Help*: Open the help file.
- *Apply*: Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero).
- *OK*: Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero) and close the editor.
- *Cancel*: Do not apply any changes to the system and close the editor.

Zeros and Poles

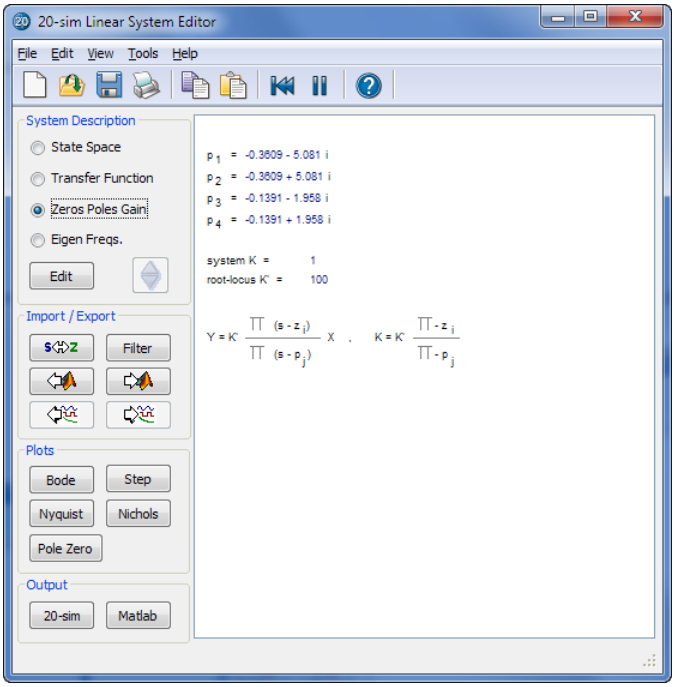
A continuous-time SISO transfer function can described by the transfer function:

$$h(s) = \frac{n(s)}{d(s)} = \frac{n_i \cdot s^i + n_{i-1} \cdot s^{i-1} + .. + n_0 \cdot s^0}{d_j \cdot s^j + d_{j-1} \cdot s^{j-1} + .. + d_0 \cdot s^0}$$

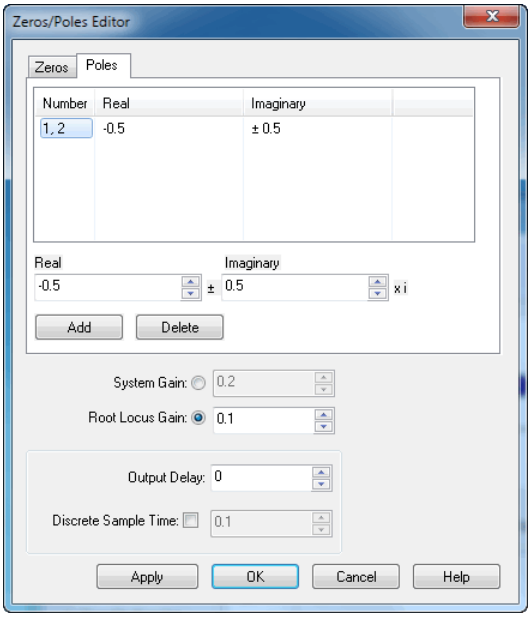
This transfer function can be rewritten in pole zero notation with

$$h(s) = K_{RL} \frac{(s - z_i)(s - z_{i-1}) \ldots (s - z_0)}{(s - p_j)(s - p_{j-1}) \ldots (s - p_0)}$$

where *pi .. p1* are the poles and *zi .. z1* are the zeros and *KRL* is the Root Locus Gain of the system. The same can be done for a discrete-time SISO transfer function.



You can enter zeros and poles by selecting the *Zeros & Poles* button and clicking the *Edit* button.



This opens an editor in which you can enter the real and imaginary parts of the zeros and poles as well as the Root Locus Gain. If preferred, you can also enter the System Gain. Note that zeros and poles always have conjugate when the imaginary part is non-zero. I.e. when you enter a pole with imaginary part 0.5 an extra pole is added with imaginary part -0.5.

Output Delay

To inspect the effects of time delay in your model, you can add output delay. The result will be visible in the various plots that you can show of a linear system. The unit of the output delay is seconds.

Discrete Sample Time

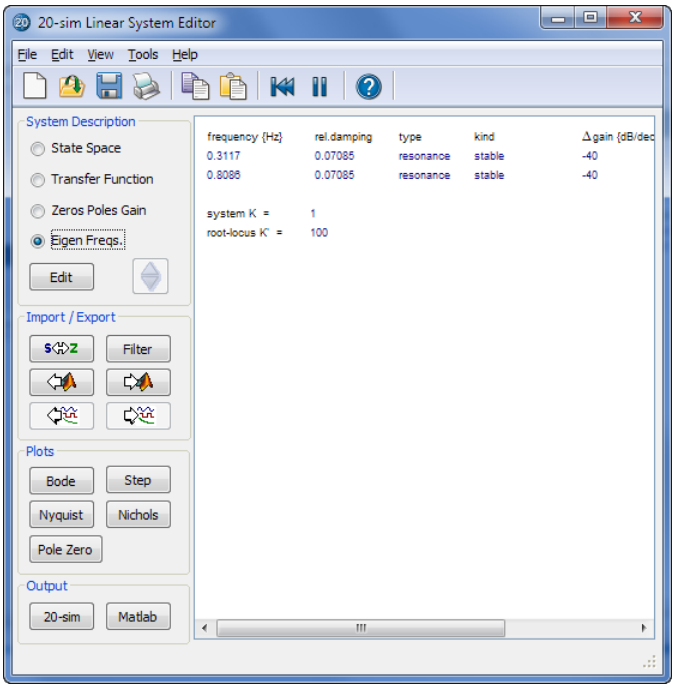
If you want to transfer a linear system from continuous time to discrete time directly (i.e. replace the s by a z), select Discrete Sample time and fill in the sample time value. You can also transfer back directly by deselecting Discrete Sample time.

Commands

- *Add/Delete:* Add or delete selected poles or zeros.
- *Help*: Open the help file.
- *Apply:* Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero).
- *OK*: Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero) and close the editor.
- *Cancel:* Do not apply any changes to the system and close the editor.

Eigen Frequencies

The Eigen Frequencies view is closely related to the pole zero notation and bode plot. It shows the resonance frequencies and anti-resonance frequencies that result from the given poles and zeros as well as some characteristic parameters from the bode plot.

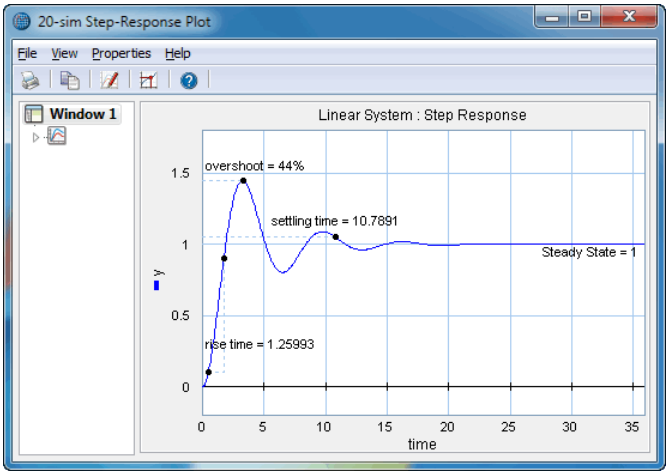


You can enter transfer functions by selecting the *Eigen Freqs.* button and clicking the *Edit* button.

Step Response

The *Step Response* command calculates the systems response y on a unit step input:

u = 0 (time < 0)
u = 1 (time >= 0)



20-sim automatically generates an appropriate range for the time response, based on the system dynamics. With the Plot Properties command (right mouse menu), you can change this horizon and recalculate the step response (click the Step command again).

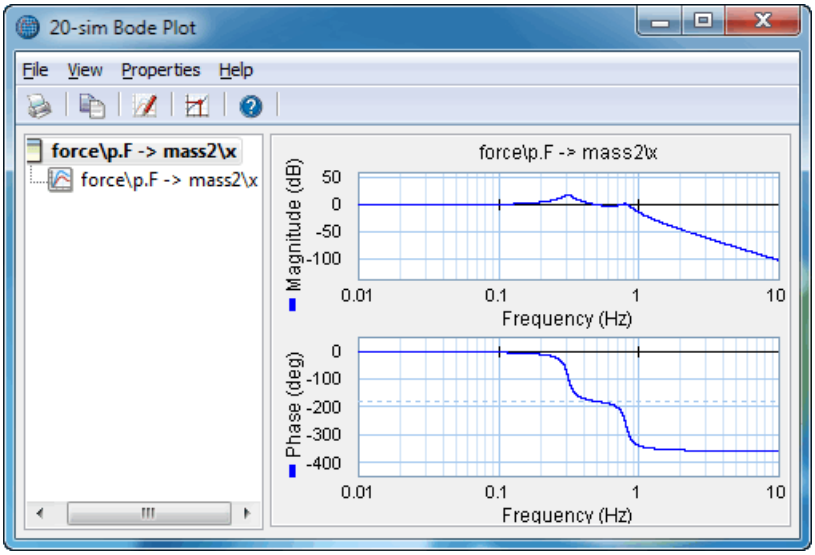
Plot Options

Using the right mouse menu, you can use several options:

- *Plot Properties*: Set the general plot properties for the step response and specify the curve properties.
- *Numerical Values*: Inspect numerical values.
- *Step Characteristics*: Display rise time, overshoot, settling time and the steady state value of the step response.
- *Copy to Clipboard*: Copy the plot to the windows clipboard.
- *Print*: Print the plot.
- *Zoom in*: Show a detail of the plot.
- *Zoom out*: Show a larger portion of the plot.
- *Zoom Normal*: Show the complete plot.

Bode Plots

Bode Plots show the amplitude and phase of a linear system as function of the frequency. Bode plots can be shown for every 20-sim model through Linearization. During Linearization you are asked to enter the input variable and output variable for which linearization should be performed. After that the linear system is calculated and shown in the Linear System Editor. From the Linear System Editor you can generate a bode plot. These actions can also be predefined using the Frequency Response command of the *Properties* menu.



20-sim automatically generates a range of logarithmically displayed frequencies, based on the system dynamics. With the **Plot Properties** command (**right mouse menu**), you can change this horizon and recalculate the bode response (click the Bode command again).

The magnitude part of the plot can be displayed in dB or in absolute values. The phase part can be displayed in radians or degrees. The frequency can be displayed in radians per second or in Hz.

Plot Options

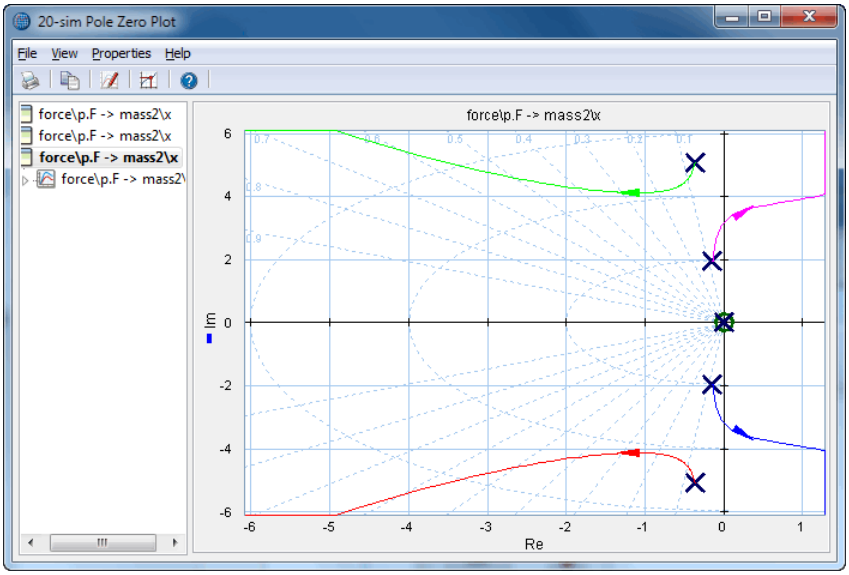
Using the toolbar or the right mouse menu, you can use various options:

- *Plot Properties:* Set the plot properties.
- *Numerical Values:* Inspect numerical values.
- *Magnitude (dB):* Display magnitude in decibels
- *Magnitude (-):* Display magnitude in absolute values.
- *Phase (rad):* Display phase in radians.
- *Phase (deg):* Display phase in degrees.
- *Frequency (rad/sec):* Display frequency in radians per second.

- *Frequency (Hz):* Display frequency in Hz.
- *Phase/Gain Margins:* Display the gain and phase margins.
- *Modulus Margin:* Display the Modulus Margin.
- *Unwrap Phase*: Display the phase plot as a continuous plot by selecting this option or as a folded plot between -180 and 180 deg. by deselecting this option.
- *Peak Response:* Display the peak response.
- *Copy to Clipboard*: Copy the plot to the windows clipboard.
- *Print:* Print the plot.
- *Zoom in:* Show a detail of the plot.
- *Zoom out*: Show a larger portion of the plot.
- *Zoom Normal:* Show the complete plot.

Pole Zero Diagram

The **Pole Zero** command plot the poles and zeros of a system and computes the root locus plot.



20-sim automatically generates a range of real and imaginary parts, based on the system dynamics. With the **Plot Properties** command (right mouse menu), you can change this horizon and recalculate the pole zero diagram (click the **Pole Zero** command again).

Root Locus

You can show the rootlocus plot by selecting **Root Locus** from the right mouse menu. Inspect the root locus gain by selecting **Numerical Values** from the right mouse menu.

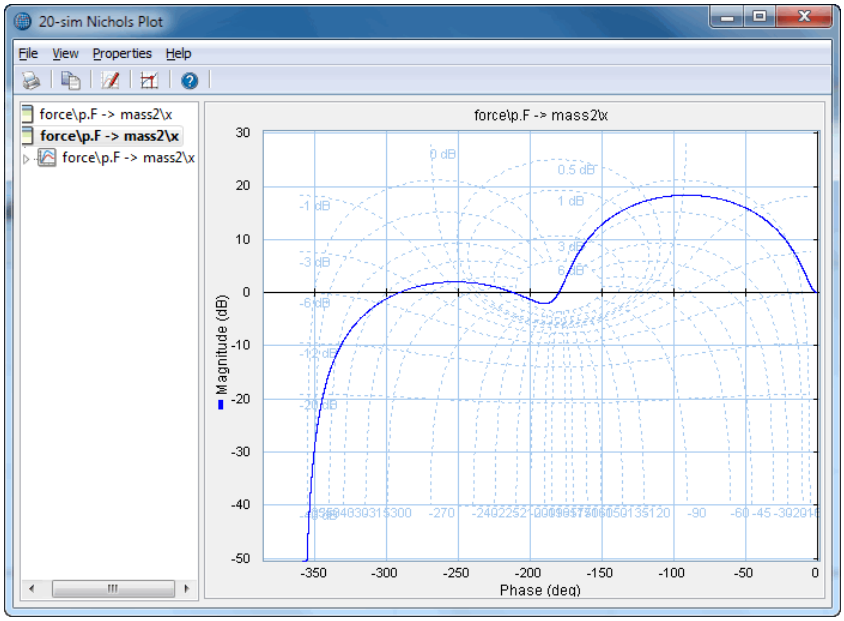
Plot Options

Using the right mouse menu, you can use several options:

- *Plot Properties*: Set the plot properties.
- *Numerical Values*: Inspect numerical values.
- *Root Locus:* Show or hide the root locus plot.
- *Copy to Clipboard:* Copy the plot to the windows clipboard.
- *Print*: Print the plot.
- *Zoom in:* Show a detail of the plot.
- *Zoom out:* Show a larger portion of the plot.
- *Zoom Normal:* Show the complete plot.

Nichols Chart

The *Nichols* command computes the Nichols chart of a system.



20-sim automatically generates a range of logarithmically displayed frequencies, based on the system dynamics. With the **Plot Properties** command (right mouse menu), you can change this horizon and recalculate the Nichols chart (click the **Nichols** command again). The magnitude part of the plot can be displayed in decibels (dB) or in absolute values. The phase part can be displayed in radians or degrees.

Plot Options

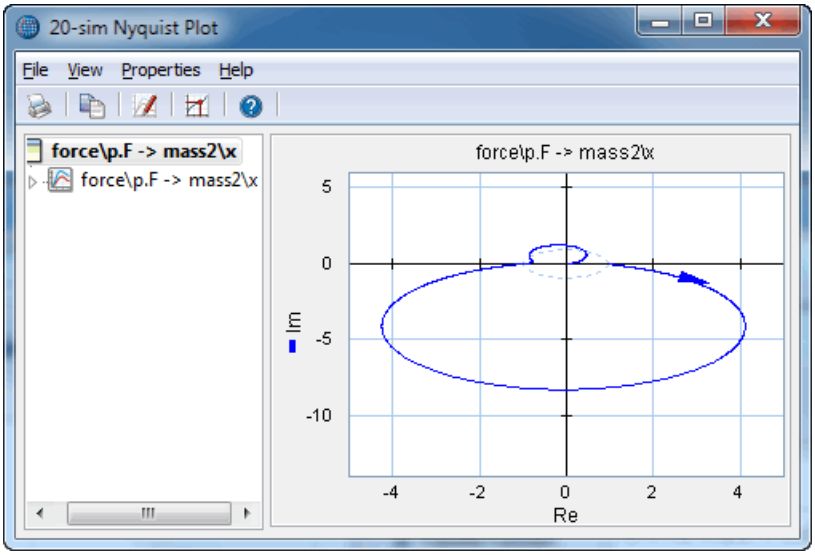
Using the right mouse menu, you can use several options:

- *Plot Properties:* Set the plot properties.

- *Numerical Values:* Inspect numerical values.
- *Magnitude (dB):* Display magnitude in decibels
- *Magnitude (-)*: Display magnitude in absolute values.
- *Phase (rad)*: Display phase in radians.
- *Phase (deg)*: Display phase in degrees.
- *Frequency (rad/sec):* Display frequency in radians per second.
- *Frequency (Hz):* Display frequency in Hz.
- *Phase/Gain Margins:* Display the gain and phase margins.
- *Modulus Margin:* Display the Modulus Margin.
- *Copy to Clipboard*: Copy the plot to the windows clipboard.
- *Print*: Print the plot.
- *Zoom in*: Show a detail of the plot.
- *Zoom out:* Show a larger portion of the plot.
- *Zoom Normal*: Show the complete plot.

Nyquist Diagram

The **Nyquist** command computes the Nyquist plot of a system.



20-sim automatically generates a range of real and imaginary parts, based on the system dynamics. With the **Plot Properties** command (right mouse menu), you can change this horizon and recalculate the Nyquist diagram (click the **Nyquist** command again).

Plot Options

Using the right mouse menu, you can use several options:

- *Plot Properties:* Set the plot properties.
- *Numerical Values:* Inspect numerical values.
- *Phase/Gain Margins*: Display the gain and phase margins.
- *Modulus Margin:* Display the Modulus Margin.
- *Copy to Clipboard:* Copy the plot to the windows clipboard.
- *Print:* Print the plot.
- *Zoom in:* Show a detail of the plot.
- *Zoom out:* Show a larger portion of the plot.
- *Zoom Normal*: Show the complete plot.

8.3.2 MLP Network Editor

Neural Networks

Human brains consist of billions of neurons that continually process information. Each neuron is like a tiny computer of limited capability that processes input information from other neurons into output information to other neurons. Connected together, these neurons form the most intelligent system known.

For some years now, researchers have been developing models that mimic the activity of neurons to produce a form of artificial intelligence. These "Neural Networks" are formed from tens or hundreds of simulated neurons connected together in much the same way as the brain's neurons. Just like the neurons of the brain, artificial neurons can change their response to a given set of inputs, or "learn".

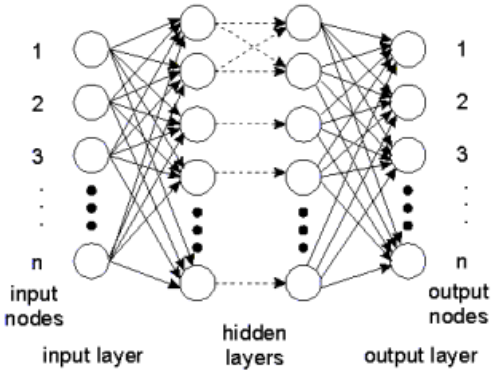
In the past many learning algorithms have been developed, mostly with limited success. The introduction of the backpropagation paradigm, however, appeared a turning point. It is an extremely effective learning tool that can be applied to a wide variety of problems.

The backpropagation paradigm require supervised training. This means neural networks must be trained by repeatedly presenting examples to the network. Each example includes both inputs (information you would use to make a decision) and desired outputs (the resulting decision, prediction, or response). Based on a calculation on the inputs, the desired outputs and the networks own response to the inputs, the backpropagation paradigms tries to adapt the response of each neuron to achieve an improved neural network behavior. This trial-and-error process continues until the network reaches a specified level of accuracy.

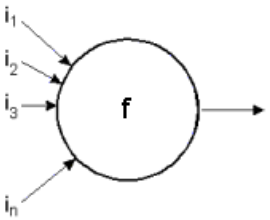
Once a network is trained and tested it's neurons can be "frozen". This means the neurons ability to learn" or adapt is stopped. The network can then be used to perform the task it was trained for.

BackPropagation Networks

In backpropagation type neural networks, the neurons or "nodes" are grouped in layers. We can distinct three groups of layers in a backpropagation type neural network: one input layer, one or more hidden layers and one output layer. The nodes between two adjacent layers are interconnected. Fully connected networks occur when each node of a layer is connected with each node of the adjacent layer.



As the figure shows, information (i.e. input signals) enters the network through the input layer. The sole purpose of the input layer is to distribute the information to the hidden layers. The nodes of the hidden layers do the actual processing. The processed information is captured by the nodes of the output layer, and transported as output signals to the world outside.



$y = f (w1*i1 + w2*i2 + w3*i3 + ... + wn*in)$

The nodes of the hidden layers process information by applying factors (weights) to each input. This is shown in the figure above. The sum of the weighted input information (S) is applied to an output function. The result is distributed to the nodes of the next layer.

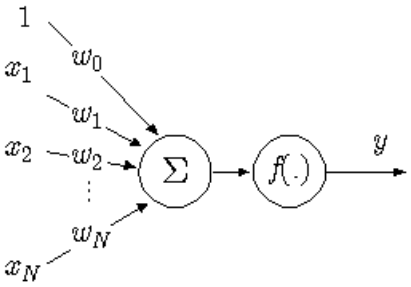
When the network is in being trained, the weights of each node are adapted according to the backproagation paradigm. When the network is in operation, the weights are constant.

Depending on the kind neural network, various topologies can be discerned. The 20-sim Neural Network Editor supports two well-known networks:

1. Adaptive B-spline networks
2. Multi-layer Perceptron networks

Introduction to MLP Networks

The basic element of the Multi Layer Perceptron (MLP) neural network, is the artificial neuron. An artificial neuron, is a unit that performs a simple mathematical operation on its inputs. In the figure below, the neuron is graphically presented.

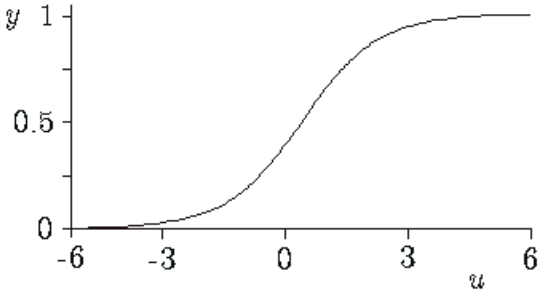


The input, **x**, of the neuron consists of the variables x1 ... xn and a bias term, known as the **momentum constant**, which is equal to 1. Each of the input values is multiplied by a weight, wi , after which the results are added. On the result, a simple mathematical function, f (x), is performed. This function is also known as the **activation function**. The calculations the neuron performs are thus given by:

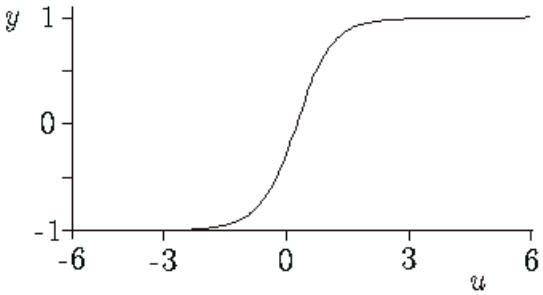
$$y = f (w0 + x1*w1 + ... + xn*wn)$$

Numerous choices for the functions exist. Frequently used implementations are the Sigmoid functions:

$$f(u) = 1 / (1 + e\text{-}u)$$



*f (u) = c1 * tanh (c2 * u)*



(in the picture c1 = c2 = 1)

MLP Network

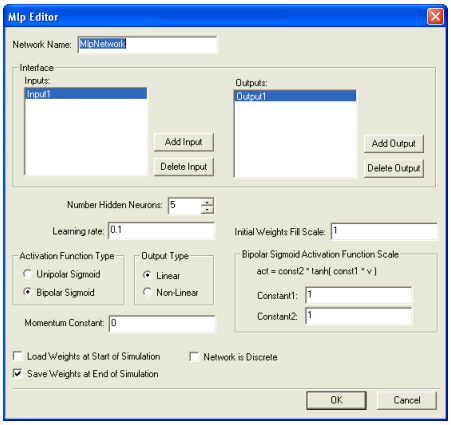
An MLP network, as any type of back-propagation network can consist of many neurons, which are ordered in layers. The neurons in the hidden layers do the actual processing, while the neurons in the input and output layer merely distribute and collect the signals. Although many hidden layers can be used, it has been shown that an MLP with one hidden layer can approximate any continuous function. Therefore in 20-sim, the MLP networks only have one hidden layer.

Training the MLP network

The MLP network is trained by adapting the weights. During training the network output is compared with a desired output. The error between these two signals is used to adapt the weights. This rate of adaptation is controlled by the **learning rate**. A high learning rate will make the network adapt its weights quickly, but will make it potentially unstable. Setting the learning rate to zero, will make the network keep its weights constant.

How to use the MLP Editor

The MLP Editor can be used to define a 20-sim Multi Layer Perceptron (MLP) network. The editor is opened when you try to edit this submodel (using the Go Down command).



General Settings

- *Network Name:* this is the local name of the submodel representing the MLP network (this option is not yet supported).
- *Number Hidden Neurons:* The MLP network specified by this editor has one layer of hidden neurons. You can specify here the number of neurons that should be used.
- *Learning Rate*: Specify the learning rate that should be used during training. If you do not want the network to learn, enter a zero value here.
- *Initial Weights Fill Scale*: To start proper learning of the network, all initial weights will be given a random value (unequal to zero!) between -scale and +scale. You can enter the scale factor here.
- *Activation Function Type*: Select one of the following activation functions
  - Unipolar Sigmoid: *act = 1/(1 + exp(-v))*
  - Bipolar Sigmoid: *act = const2*tanh(const1*v)*
- *Bipolar Sigmoid Activation Function Scale*: If a Bipolar Sigmoid was chosen as the activation function, you can enter here the values of const1 and const2.
- *Momentum Constant:* Select the momentum constant here.
- *Load Weights at Start of Simulation*: Select this option, if you want to use predefined weights stored on file (saved in a previous run). Before each simulation run, you will be asked to enter the filename of this weights file.
- *Save Weights at Start of Simulation*: Select this option, if you want to store the weights on file. After each simulation run, you will be asked to enter the filename of this weights file.
- *Network is Discrete:* Select this option if the B-spline network is connected with discrete-time models. Deselect this option if the network is connected with continuous models.

Inputs

- Add Input: Add a new input to the network. You will we prompted to give a specific input name. This name will be shown in the Inputs list.
- Delete Input: Delete the input selected in the Inputs list.

Outputs

- Add Output: Add a new output to the network. You will we prompted to give a specific output name. This name will be shown in the Outputs list.
- Delete Output: Delete the output selected in the Outputs list.

8.3.3 B-Spline Network Editor

Neural Networks

Human brains consist of billions of neurons that continually process information. Each neuron is like a tiny computer of limited capability that processes input information from other neurons into output information to other neurons. Connected together, these neurons form the most intelligent system known.

For some years now, researchers have been developing models that mimic the activity of neurons to produce a form of artificial intelligence. These "Neural Networks" are formed from tens or hundreds of simulated neurons connected together in much the same way as the brain's neurons. Just like the neurons of the brain, artificial neurons can change their response to a given set of inputs, or "learn".

In the past many learning algorithms have been developed, mostly with limited success. The introduction of the backpropagation paradigm, however, appeared a turning point. It is an extremely effective learning tool that can be applied to a wide variety of problems.

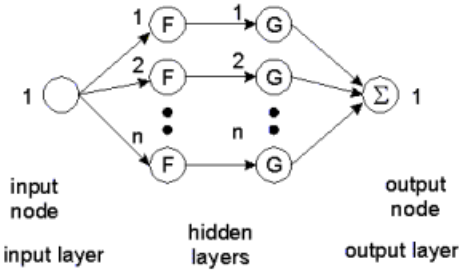
The backpropagation paradigm require supervised training. This means neural networks must be trained by repeatedly presenting examples to the network. Each example includes both inputs (information you would use to make a decision) and desired outputs (the resulting decision, prediction, or response). Based on a calculation on the inputs, the desired outputs and the networks own response to the inputs, the backpropagation paradigms tries to adapt the response of each neuron to achieve an improved neural network behavior. This trial-and-error process continues until the network reaches a specified level of accuracy.

Once a network is trained and tested it's neurons can be "frozen". This means the neurons ability to learn" or adapt is stopped. The network can then be used to perform the task it was trained for.

Introduction to B-Spline Networks

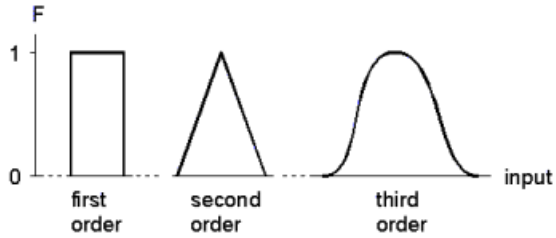
One-dimensional B-spline network

An adaptive B-spline network can be used to relate k inputs and a single output y on a restricted domain of the input space. The following network shows a realization with one input:



The network is shown with two hidden layers. Some authors prefer to show a B-spline network with only one hidden layer. For a proper understanding of multi-dimensional B-spline networks we prefer to show the network with two hidden layers.

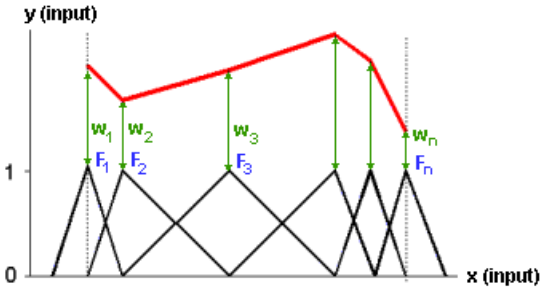
The first hidden layer distributes the inputs over several nodes. In the figure above input 1 is distributed over n nodes. Each node of this layer has only one input. To this input a "basis function" F is applied. As basis functions, B-splines are used of any desired order. An N-th order B-spline function consists of pieces of (N-1)th order polynomials, such that the resulting function is (N-1) times differentiable. The figure below shows examples of B-spline functions of different order. A spline function differs from zero on a finite interval.



The second hidden layer also consists of n nodes. Each node of this layer has only one input. To this input a function G is applied which is merely a multiplication by a weight:

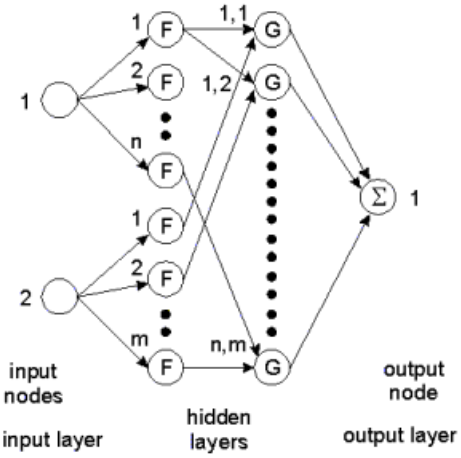
$G = w * i$

The output node sums the results of all second hidden layer nodes. When the spline functions of the various nodes are properly spaced, every one dimensional function can be approximated. This is shown in the figure below where the various splines (F1 to Fn) combined with the various weights (w1 to wn), together form an output function.



Two-dimensional B-spline network

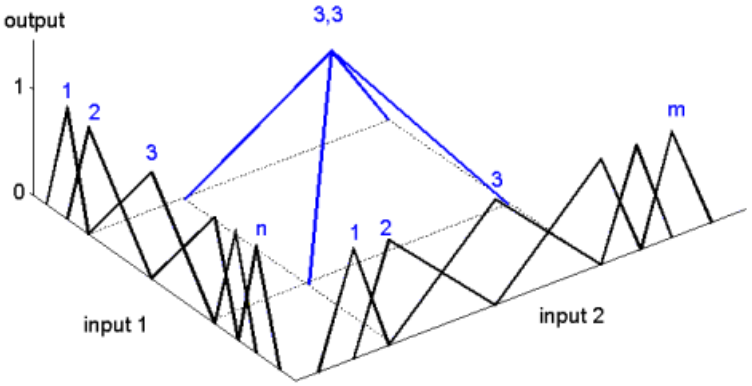
Two-dimensional B-spline networks have two input nodes. The first hidden layer, as with the one-dimensional network, consists of nodes, to which a basis function F is applied. This is shown in the figure below. To the first input a group of n nodes are applied, and to the second input a group of m nodes are applied.



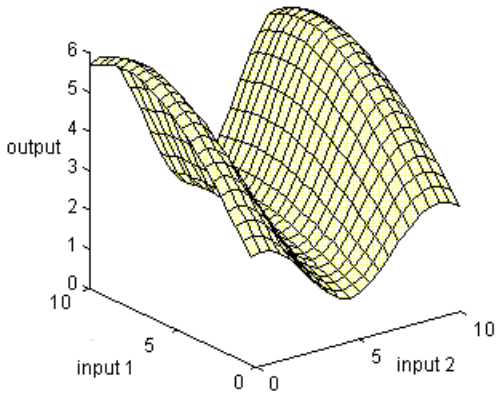
The second hidden layer now consists of nodes which each have two inputs. For every combination of a node from one group and and a node from the second group, a node exists. To each node of the second hidden layer, a function G is applied which is merely a multiplication of the two inputs multiplied by a weight:

$$G = w * i1 * i2$$

Again the output node sums the results of all second hidden layer nodes. When the spline functions of the various nodes are properly spaced, every two dimensional function can be approximated. The figure below shows the spacing of various spline functions of the two inputs and one output of a node of the second hidden layer.



The output node sums the results of all second hidden layer nodes. When the spline functions of the various nodes are properly spaced, every two dimensional function can be approximated. This is shown in the figure below.



More-dimensional B-spline networks

In a same way more dimensional B-spline networks can be created, using more inputs. The 20-sim B-spline Editor supports networks with up to 256 inputs and one output.

Training the B-spline network

The B-spline network is trained by adapting the weights. During training the network output is compared with a desired output. The error between these two signals is used to adapt the weights. This rate of adaptation is controlled by the **learning rate**. A high learning rate will make the network adapt its weights quickly, but will make it potentially unstable. Setting the learning rate to zero, will make the network keep its weights constant. Learning of B-spline networks can either be done after each sample (**learn at each sample**), or after series of samples (**learn after leaving spline**).

Learn at each Sample

When learning at each sample, after every sample the weights are adapted according to:

$$\Delta w_j = \gamma \cdot (y_d - y) \cdot F_j(x)$$

with

Δw_j : adaptation of the weight w_j

γ : learning rate

$F_j(x)$: B-spline function number j

x : input

y_d : desired output

y : network output

Given a certain input (x), only a limited number of splines Fi (x) are nonzero. Therefore only a few weights are adapted each sample.

Learn after Leaving Spline

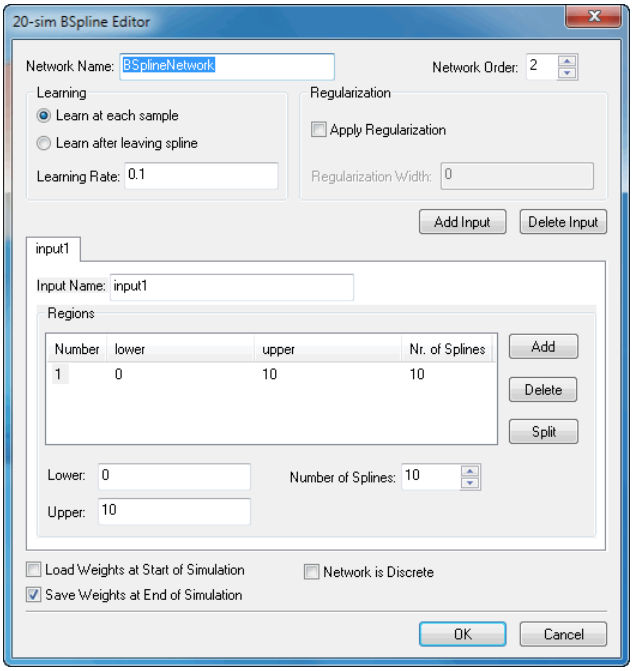
When learning after leaving a spline, the network will keep track of the input (x) and the corresponding nonzero splines Fi (x). For each nonzero spline, a sample will be stored consisting of the input (x), the output (y) and desired output (yd). Only after the input has left the region where a spline is nonzero, its weight is updated according to:

$$\Delta w_j = \gamma \cdot \frac{\sum_{i=1}^{n} (y_{d,i} - y_i) \cdot F_j(x_i)}{\sum_{i=1}^{n} F_j(x_i)}$$

Here n is the number of samples that have been taken.

How to use the BSpline Editor

The B-Spline Editor can be used to define a 20-sim B-Spline network. The editor is opened when you try to edit this submodel (using the Go Down command).



General Settings

- *Network Name*: this is the local name of the submodel representing the B-Spline network (this option is not yet supported).
- *Order*: Enter the order of the B-Splines that must be used.
- *Learn at Each Sample*: Select this option if you want the network to adapt the weights after every simulation step
- *Learn after leaving Spline:* Select this option if you want the network to adapt the weights only leaving the spline region.
- *Learning Rate:* Specify the learning rate that should be used during training. If you do not want the network to learn, enter a zero value here.
- *Apply Regularization*: This option is not yet supported.
- *Load Weights at Start of Simulation:* Select this option, if you want to use predefined weights stored on file (saved in a previous run). Before each simulation run, you will be asked to enter the filename of this weights file.
- *Save Weights at End of Simulation*: Select this option, if you want to store the weights on file. After each simulation run, you will be asked to enter the filename of this weights file.

- *Network is Discrete*: Select this option if the B-spline network is connected with discrete-time models. Deselect this option if the network is connected with continuous models.

Input Settings

The tabs shown in the editor is used to specify the inputs of the network. Each tab shows the specific settings of that input.

- *Add Input:* Add a new input to the network. A new tab will be added with default settings.
- *Delete Input*: Delete the input defined in the selected tab (the tab that is in front).

Each input is associated with a certain number of splines. To give the user more flexibility, each input can be divided in certain regions that each have their own density of splines.

- *Add:* Add a new region of splines.
- *Delete*: Delete the selected region of splines.
- *Split:* Split the selected region into two regions.

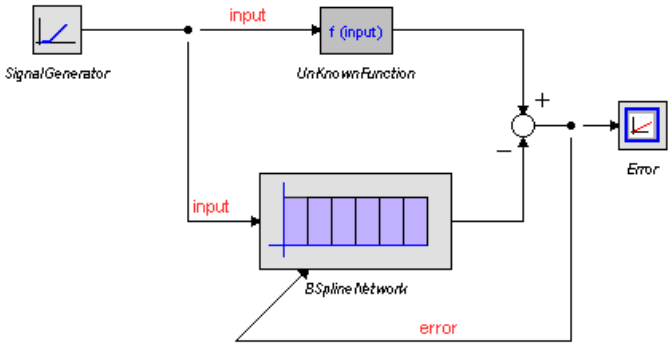
Each region can be edited. Just select it from the Regions list and use the following options:

- *Upper / Lower*: The upper and lower bound of a region.
- *Number of Splines:* The number of splines. The more splines you choose, the more accurate will the network output be, for this region of inputs.

Running a 1-D B-Spline Network

Now we have trained the B-spline network and saved the weights. We can run the network with every desired input. We only have to keep in mind that the network was defined for input signals between 0 and 10.

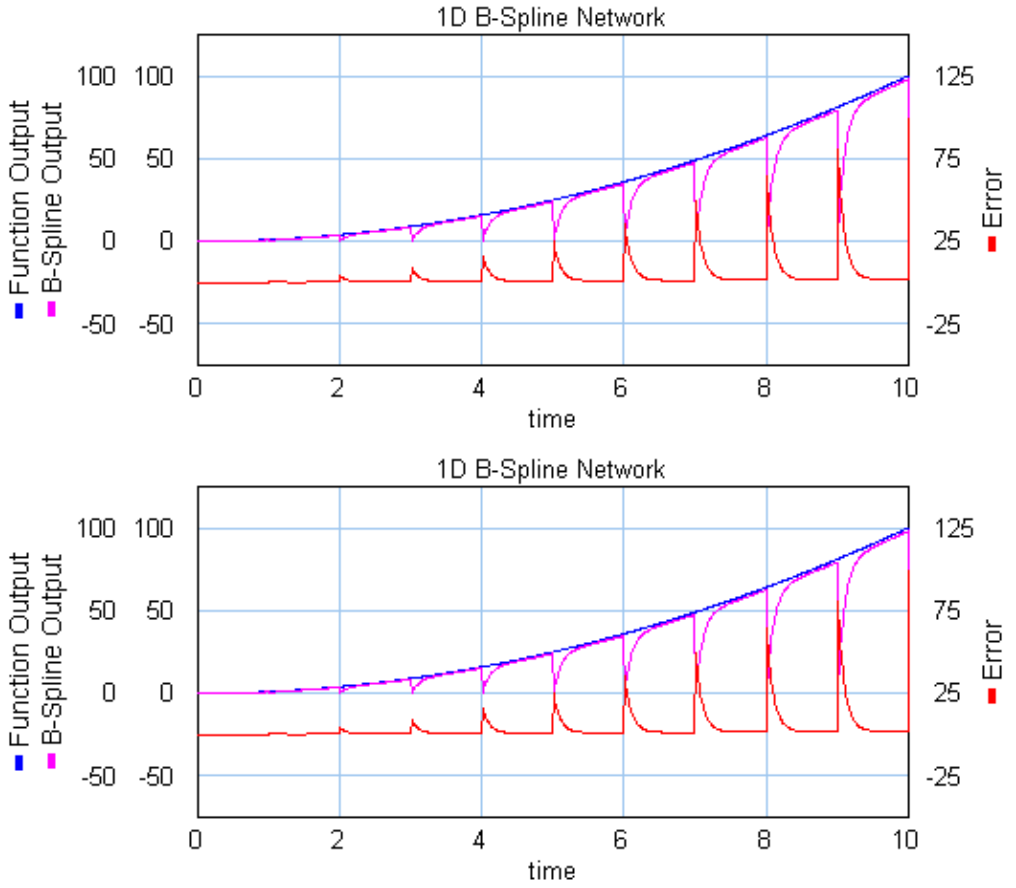
1. In the Editor from the **File** menu select **Open**.
2. In the model library choose Examples\Control\Neural Networks
3. Select the model 1DBSplineNetwork-Run.emx. Now the predefined model will be opened.



As you can see, this is exactly the same model as used for training, but now a ramp input is used.

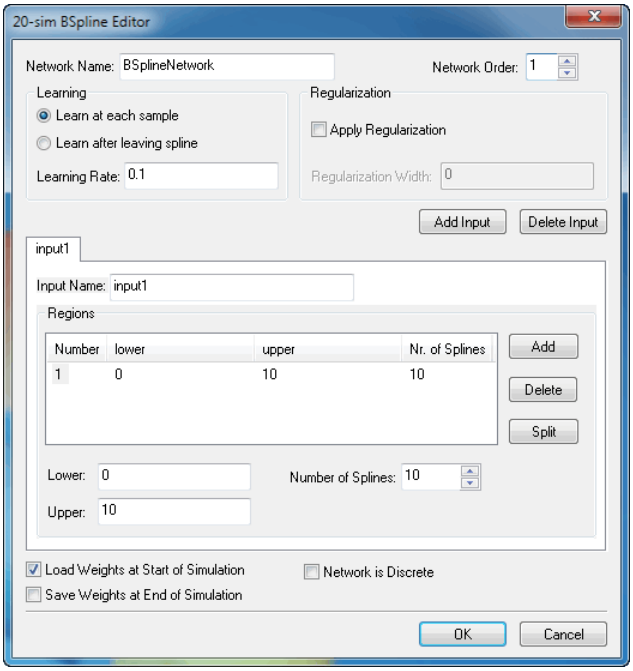
4. From the menu select **Model** and **Start Simulator**. Now a Simulator will be opened with the predefined experiment loaded.

5. From Simulator menu click **Simulation** and **Run**. Do **not save** the weights file at the end of the run! The results will look like:



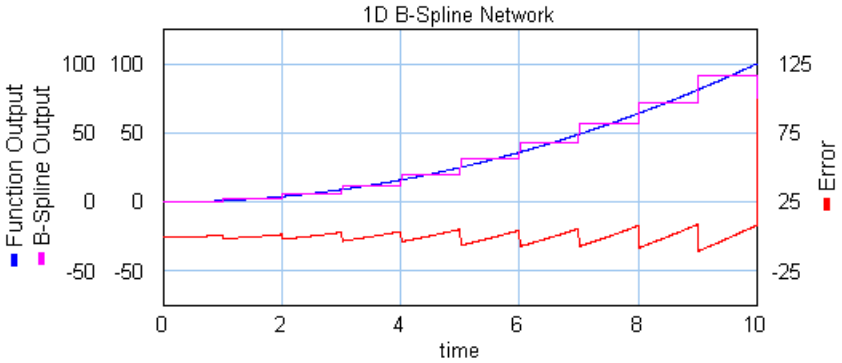
As you can see, the B-Spline Network still has its training settings and still prompts for a weight file at the end of the simulation. Now we are going to change the settings of the B-Spline Network.

6. Open the B-Spline Editor (select the model, Go Down). To create a normal run with the network, we have to set the learning rate to zero, and load the weights file before simulation. Change the settings of the network, until it looks like:



7. Click **OK** to close the B-Spline Editor. From the Editor menu click **Model** and **Check Complete Model**. This will implement the changed settings.

8. Return to the Simulator and click **Simulation** and **Run**. Before the run, an Open dialog will asking you to enter the name of the Weights File. This is a file that contains all the weights of the trained B-Spline network. Use the file that you saved during training, or the predefined weights file *1DBSplineNetwork.wgt.* The simulation results will look like:



As you can see the function is approximated by the B-Spline network by 10 first order splines.

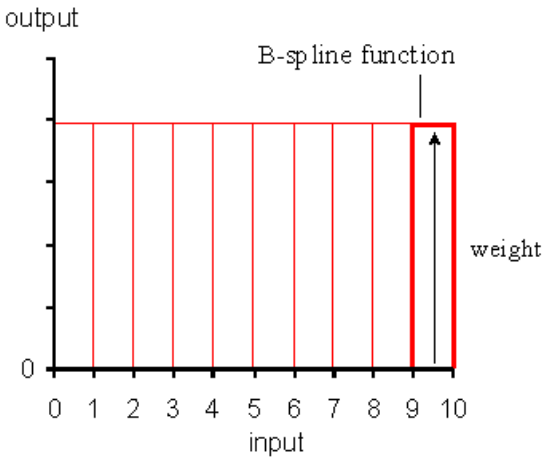
Training a 1-D B-Spline Network

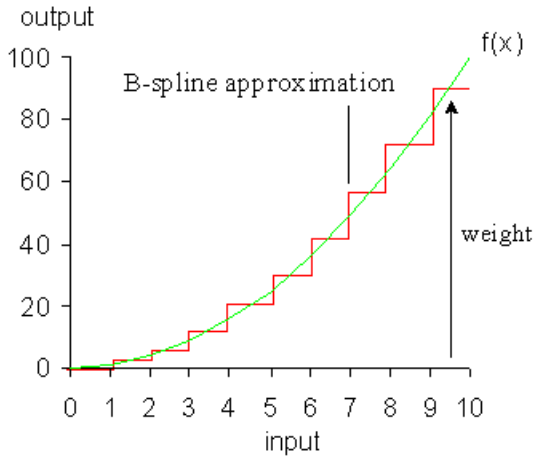
Non Linear Function

In this lesson we will use a B-spline network with one input to approximate the simple non-linear function:

F = input2

The input, x , of this function ranges from 0 to 10. It will be approximated by a B-Spline network with one input. As is shown in the figure below (left), 10 first order B-splines are used. The tenth B-spline is indicate with fat lines. The output of the B-spline network is a combination of all ten B-Splines. In other words, the non-linear function will be approximated by 10 intervals of constant value.

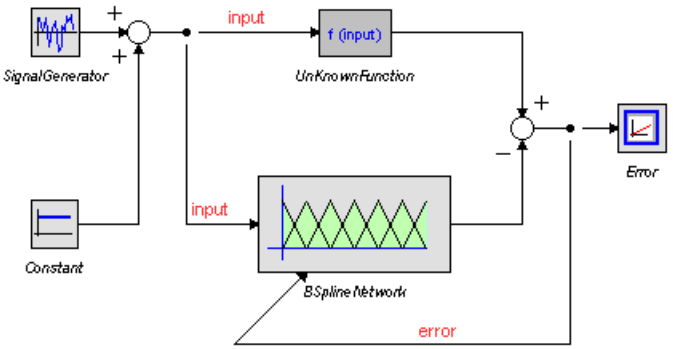




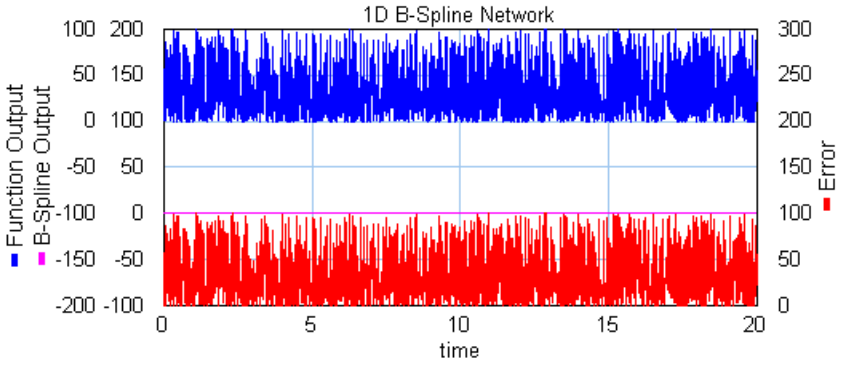
At the start the B-spline has undefined weights. They can either be chosen at random values or all set to zero. A normal session should therefore start with training the network to adjust the weights to give a best approximation of the non-linear function. With these weights we can run the network with any desired input signal.

Defining the B-Spline Network

We will use the model below to train the B-spline network. It has already been prepared for you.



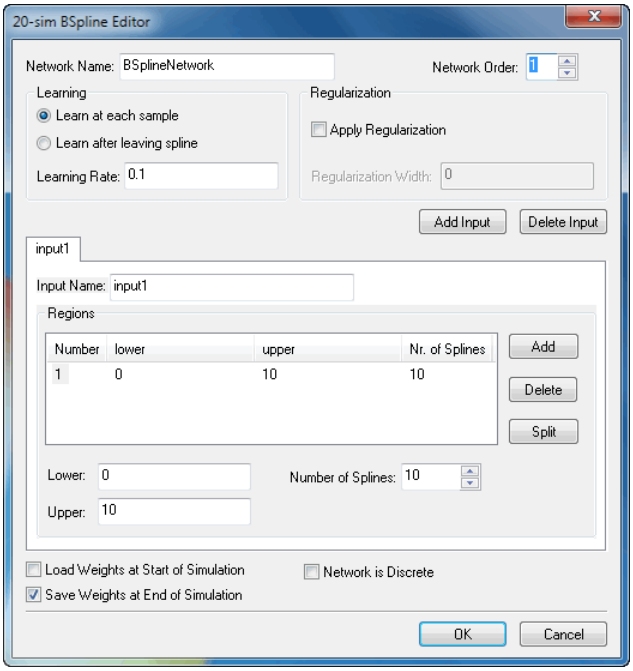
1. In the Editor from the **File** menu select **Open**.
2. In the model library choose Examples\Control\Neural Networks
3. Select the model *1DBSplineNetwork-Train.emx*. Now the predefined model will be opened.
4. From the menu select **Model** and **Start Simulator**. Now a Simulator will be opened with the predefined experiment loaded.
5. From Simulator menu click **Simulation** and **Run**. The results will look like:



As you can see, the B-Spline output is zero and there is no learning. Now we are going to change the settings of the B-Spline Network.

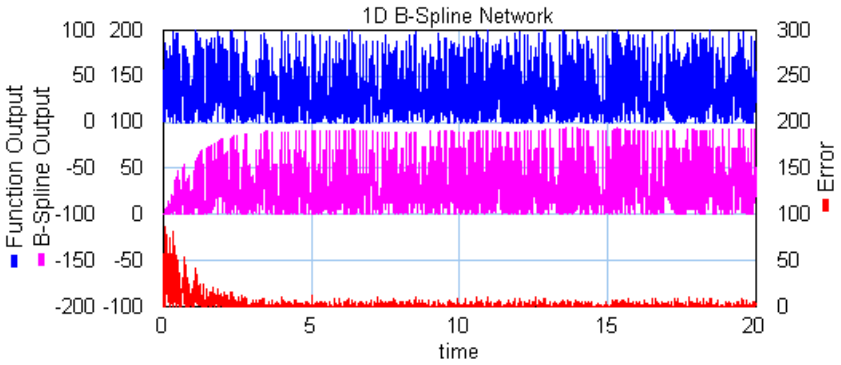
Training the Network

6. Return to the Editor. Select the BSplineNetwork model and click **GoDown** from the **Model** menu. Now the B-Spline Editor will pop-up. Change the settings (Network Order, Learning Rate, Input Upper Limit, Save Weights) until it looks like:



7. Click **OK** to close the B-Spline Editor. Not that the icon of the BSplineNetwork model has changed to correspond with the first order splines that are used. From the Editor menu click **Model** and **Check Complete Model**. This will implement the changes in settings.

8. Return to the Simulator and click **Simulation** and **Run**. The results will look like:



As you can see the error rapidly descent, due to the network learning. After the Simulation Run, Save dialog opens, asking you to enter a name for the Weights File. This is a file that contains all the weights of the trained B-Spline network. We can use these weights for normal runs with the network.

9. Enter file name (do not use the predefined weights file!), e.g. *test.wgt* and click **Save**.

10. Close the **Simulator.**

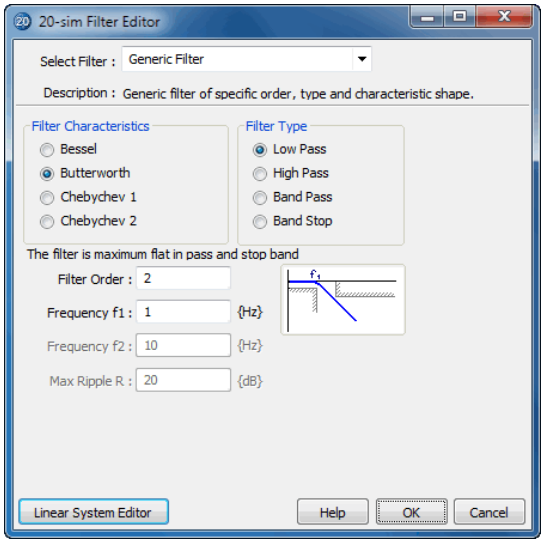
Now all is prepared to perform a normal run with the Network.

8.3.4 Filter Editor

Filter Editor

The Filter Editor of 20-sim allows you to create continuous-time filters and controllers. The result is a linear system presented in the Linear System Editor of 20-sim. The filter can also be applied to an existing linear system.

You can open the Controller Design Editor from the Tools menu of the 20-sim Editor or by clicking Go Down on a Filter model.



Procedure

1. Select a filter or controller of your choice from the list of available filters and controllers.

Filters

- Generic Filter
- Lead Filter
- Lag Filter
- Low Pass First Order Filter
- Low Pass Second Order Filter

- High Pass First Order Filter
- High Pass Second Order Filter
- Notch Filter
- Universal Notch Filter

Controllers

- P Controller
- I Controller
- D Controller
- PI Controller
- PD Controller
- PID Controller
- PID-1 Controller
- PID-2 Controller
- PID Compensator

A short description of the selected filter will be given in the description field. Depending on the filter that was chosen three or more parameters will be shown.

2. The second step is to fill in the desired parameters.
3. Choose the desired output:
4. Select the **Linear System Editor** button to export the filter to the Linear System Editor.
5. Select the **OK** button to store the filter model on file or update the filter model in the Editor.

Controllers

P-Controller

The P Controller can be used to create an additional gain for a linear system.

- Gain *Kp*:The controller gain *Kp* determines the amount of gain the controller adds for each frequency.

Transfer function of the controller:

$$H_P(s) = K_p$$

I-Controller

The I Controller can be used to create an almost ideal integrator (when the frequency goes to zero).

- Gain *Ki*: The filter gain *Ki* determines the frequency gain of the controller at frequency w = 1 [rad/s].
- Frequency *fi*: The integration behavior of the controller stops at frequency *fi* in Hz. A decrease of this frequency corresponds to a more pure integration behavior.

Transfer function of the controller

$$H_I(s) = \frac{K_i}{2\pi f_i + s}$$

D-Controller

The D Controller can be used to create an almost ideal derivative (when the frequency goes to infinity).

- Gain *Kd*: The controller gain *Kd* determines the frequency gain of the controller at frequency w = 1 [rad/s].
- Frequency *fd*: The differentiation behavior of the controller stops at frequency *fd* in Hz. An increase of this frequency corresponds to a more pure differentiation behavior, reducing the tameness of the controller.

Transfer function of the controller

$$H_D(s) = K_d \frac{2\pi f_d s}{s + 2\pi f_d}$$

PD-Controller

The PD Controller consists of a differentiator with proportional gain.

- Gain *Kp:* The controller gain *Kp* determines the low frequency gain of the controller.
- Gain *Kd:* The controller gain *Kd* determines the gain of the controller at frequency w = 1 [rad/s].
- Frequency *fd*: The differentiation behavior of the controller stops at frequency *fd* in Hz. An increase of this frequency corresponds to a more pure differentiation behavior, reducing the tameness of the controller.

Transfer function of the controller:

$$H_{PD} = H_P + H_D$$

that equals

$$H_{PD} = \frac{(K_p + K_d 2\pi f_d)s + K_p 2\pi f_d}{s + 2\pi f_d}$$

PI-Controller

The PI Controller consists of an integrator with a proportional gain.

- Gain *Kp*: The controller gain *Kp* determines the high frequency gain of the controller.
- Gain *Ki*: The controller gain *Ki* determines the frequency gain of the controller at frequency w = 1 [rad/s].
- Frequency *fi*: The integration behavior of the controller stops after frequency *fi* in Hz. A decrease of this frequency corresponds to a more pure integration behavior.

Transfer function of the controller:

$$H_{PI} = H_P + H_I$$

that equals

$$H_{PI} = \frac{K_p s + K_p 2\pi f_i + K_i}{s + 2\pi f_i}$$

PID1-Controller

The PID-1 Controller consists of an integrator and differentiator with proportional gain. It resembles a PID Controller, but has different input parameters.

- Gain *Kp*: The controller gain *Kp* determines the frequency gain of the controller in the pass band (where integration and differentiation are of no interest).
- Length *Li*: The length *Li* determines the frequency at which integration starts, that is *fi / Li* in Hz. A decrease of this frequency corresponds to a more pure integration behavior.
- Length *Ld*: The length *Ld* determines the frequency at which differentiation stops, that is *fd* * *Ld* in Hz. An increase of this frequency corresponds to a more pure differentiation behavior, reducing the tameness of the controller.
- Frequency *fi*: The integration behavior of the controller stops at frequency *fi* in Hz.
- Frequency *fd*: The differentiation behavior of the controller starts at frequency *fd* in Hz.

Transfer function of the controller:

$$H_{PID} = K_p \left(1 + \frac{L_i - 1}{1 + \frac{L_i s}{2\pi f_i}} + \frac{(L_d - 1)s}{L_d 2\pi f_d + s} \right)$$

that equals

$$H_{PID} = \frac{K_p L_d L_i s^2 + K_p(-2\pi f_i + L_i L_d 2\pi f_d + (L_i + L_d)2\pi f_i)s + K_p L_i L_d 4\pi^2 f_i f_d}{L_i s^2 + (2\pi f_i + L_i L_d 2\pi f_d)s + L_i L_d 4\pi^2 f_i f_d}$$

PID2-Controller

The PID-2 Controller consists of an integrator and differentiator with proportional gain. It resembles a serial PID Controller, but has different input parameters.

- Gain *Kp*: The controller gain *Kp* determines the frequency gain of the controller in the pass band (where integration and differentiation are of no interest).
- Length *Li*: The length *Li* determines the frequency at which integration starts, that is *fi / Li* in Hz. A decrease of this frequency corresponds to a more pure integration behavior.
- Length *Ld*: The length *Ld* determines the frequency at which differentiation stops, that is *fd * Ld* in Hz. An increase of this frequency corresponds to a more pure differentiation behavior, reducing the tameness of the controller.
- Frequency *fi*: The integration behavior of the controller stops at frequency *fi* in Hz.
- Frequency *fd*: The differentiation behavior of the controller starts at frequency *fd* in Hz.

Transfer function of the controller:

$$H_{PID} = K_p L_i \left(\frac{\frac{s}{2\pi f_i} + 1}{\frac{s}{2\pi f_i / L_i} + 1} \right) \left(\frac{\frac{s}{2\pi f_d} + 1}{\frac{s}{2\pi f_d * L_d} + 1} \right)$$

that equals

$$H_{PID} = \frac{K_p L_d L_i s^2 + K_p L_i L_d 2\pi (f_i + f_d)s + K_p L_i L_d 4\pi^2 f_i f_d}{L_i s^2 + (2\pi f_i + L_i L_d 2\pi f_d)s + L_i L_d 4\pi^2 f_i f_d}$$

PID-Compensator

The PID Compensator consists of an integrator and differentiator with proportional gain (PID Controller) together with a high-frequency roll-off filter.

- Gain *K*: The controller gain *K* determines the frequency gain of the controller in the pass band (where integration and differentiation are of no interest).
- Integration *tau_i:* The integration time constant *tau i* determines the frequency *fi = 1 / tau_i [rad/s]* after which the integrating behavior stops.
- Differentiation *tau_d:* The differentiation time constant tau d determines the frequency *fd = 1 / tau_d [rad/s]* after which the differentiation behavior starts.
- HF roll-off *tau_h*: The filter time constant *tau h* determines the frequency *fh = 1 / tau_h [rad/s]* of the high-frequency roll-off filter. Frequencies higher than *fh* are filtered out.
- Tameness *beta*: The tameness factor beta influences the differentiating behavior. After frequency *fdb = fd / beta* the differentiation behavior stops. Decreasing beta increases this frequency *fdb*, resulting in a more pure differentiating behavior and therefore in a reduce of differentiation tameness.

Transfer function of the controller:

$$H_{PIDH} = H_{PI} \cdot H_D \cdot H_H$$

that equals

$$H_{PID} = K\left(\frac{\tau_i s + 1}{\tau_i s}\right) \cdot \left(\frac{\tau_d s + 1}{\tau_d \beta s + 1}\right) \cdot \left(\frac{1}{\tau_h s + 1}\right)$$

PID-Controller

The PID Controller consists of an integrator and differentiator with proportional gain.

- Gain *Kp*: The controller gain *Kp* determines the frequency gain of the controller in the pass band (where integration and differentiation are of no interest).
- Gain *Ki*: The controller gain *Ki* determines the shape of the frequency response at low frequencies.
- Gain *Kd*: The controller gain *Kp* determines the shape of the frequency response at high frequencies.
- Frequency *fi*: The integration behavior of the controller starts at frequency *fi* in Hz. A decrease of this frequency corresponds to a more pure integration behavior.
- Frequency *fd*: The differentiation behavior of the controller stops at frequency *fd* in Hz. An increase of this frequency corresponds to a more pure differentiation behavior, reducing the tameness of the controller.

Transfer function of the controller:

$$H_{PID} = H_P + H_I + H_D$$

that equals

$$H_{PID} = \frac{(K_p + K_d 2\pi f_d)s^2 + (K_p 2\pi f_i + K_d 4\pi^2 f_i f_d + K_p 2\pi f_d + K_i)s + K_p 4\pi^2 f_i f_d + K_i 2\pi f_d}{s^2 + (2\pi f_i + 2\pi f_d)s + 4\pi^2 f_i f_d}$$

Filters

Generic Filter

The Generic Filter can be used to design a filter of a specific order, type and characteristic shape.

Filter Characteristics

The filter can have a Bessel, Butterworth or Chebychev characteristic shape. Each characteristic has its own advantages and disadvantages; it depends on the purpose of the filter what to choose here:

A *Bessel* filter is maximal flat in both the pass band and the stop band. It is used when the phase response should be nearly linear throughout the pass band, i.e. the group delay is almost constant throughout the pass band. This preserves the wave shape of filtered signals in the pass band. A disadvantage is the low roll-off steepness. Often a high filter order is required to obtain the desired roll-off.

A *Butterworth* filter is maximally flat in the pass band and monotonic overall. As a consequence of this smoothness, the roll-off steepness is rather low. As with Bessel filters, a rather high filter order is required to obtain the desired roll-off.

A *Chebychev* 1 filter has a ripple in the pass band and is monotonic in the stop band. This filter rolls off faster than Bessel, Butterworth and Chebychev 2 filters at the same filter order. A disadvantage is the greater deviation and discontinuous phase response in the pass band.

A *Chebychev 2* filter is monotonic in the pass band and has a ripple in the stop band. This filter rolls off faster than Bessel and Butterworth filters at the same filter order. Not as fast as a Chebychev 1 filter, but it is free of a pass band ripple. A disadvantage is the ripple and discontinuous phase response in the stop band.

Filter Type

- A *low pass* filter will filter out high frequencies and let low frequencies pass the filter.
- A *high pass* filter does the opposite; it will filter out low frequencies and let high frequencies pass the filter.
- A *band pass* filter will filter out low and high frequencies and let middle frequencies pass the filter.
- A *band stop* filter does the opposite; it will filter out the middle frequencies and let low and high frequencies pass the filter.

Filter Order

The filter order specifies how steep the filter will roll off at a specified filter frequency. The higher the order, the steeper the roll off will be. Note that the filter order also has a large influence on the phase response of the filter.

Frequency f1

For a low pass filter *f1* is the frequency in Hz where the roll off starts. For a high pass filter it is the frequency where the roll off ends. For a band pass or band stop filter f1 is the start frequency of the band.

Frequency f2

For a band pass or band stop filter *f2* is the stop frequency in Hz of the band. It is disabled for low pass and high pass filters.

Max. Ripple R

Only available when a Chebychev 1 filter is selected. *R* specifies the maximum allowed ripple in dB for the filter in the pass band.

Stop Band R

Only available when a Chebychev 2 filter is selected. *R* specifies the distance of the stop band to the unity gain axis in dB. This is the amount of attenuation in dB of the filter in the stop band (the absolute value is used).

High-Pass First Order Filter

The high pass filter can be used to filter out undesired low frequencies.

- Gain *K*: The filter gain *K* determines the high frequency gain of the filter.
- Frequency *f*: The filter roll off stops at frequency *f* in Hz. Higher frequencies will pass the filter, lower frequencies will be filtered out.

Transfer function of the filter:

$$H(s) = K\frac{s}{s + 2\pi f}$$

High-Pass Second Order Filter

The high pass filter can be used to filter out undesired low frequencies.

- Gain *K*: The filter gain *K* determines the high frequency gain of the filter.
- Frequency *f*: The filter roll off stops at frequency *f* in Hz. Higher frequencies will pass the filter, lower frequencies will be filtered out.
- Quality *Q*: The filter quality indicates how steep the phase response will increase up to the frequency f. A quality *Q* > 0.707 results in a peak in the magnitude response at frequency f.

Transfer function of the filter:

$$H(s) = K\frac{s^2}{s^2 + 2\pi f\frac{s}{Q} + (2\pi f)^2}$$

Lag Filter

The Lag Filter can be used to add a lag in the phase response of a system.

- Gain *K*: The filter gain *K* determines the low frequency gain of the filter.
- Frequency *f1*: The lag in the phase response of the filter starts at frequency *f1* in Hz. This corresponds to an integrating behavior in the magnitude response from frequency *f1*.
- Frequency *f2*: The lag in the phase response of the filter stops at frequency *f2* in Hz. This corresponds to a constant attenuation in the magnitude response of the filter after frequency *f2*.

Transfer function of the filter

$$H_{lag}(s) = K\left(\frac{\frac{s}{2\pi f_2}+1}{\frac{s}{2\pi f_1}+1}\right)$$

Lead Filter

The Lead Filter can be used to add a lead in the phase response of a linear system.

- Gain *K*: The filter gain *K* determines the low frequency gain of the filter.
- Frequency *f1*: The lead in the phase response of the filter starts at frequency *f1* in Hz. This corresponds to a differentiating behavior in the magnitude response from frequency *f1*.
- Frequency *f2*: The lead in the phase response of the filter stops at frequency *f2* in Hz. This corresponds to a constant gain in the magnitude response of the filter after frequency *f2.*

Transfer function of the controller

$$H_{lead}(s) = K\left(\frac{\frac{s}{2\pi f_1}+1}{\frac{s}{2\pi f_2}+1}\right)$$

Low-Pass First Order Filter

The low pass filter can be used to filter out undesired high frequencies.

- Gain *K*: The filter gain *K* determines the low frequency gain of the filter.
- Frequency *f*: The filter roll off starts at frequency *f* in Hz. Lower frequencies will pass the filter, higher frequencies will be filtered out.

Transfer function of the filter

$$H(s) = \frac{K}{\frac{s}{2\pi f} + 1}$$

Low-Pass Second Order Filter

The low pass filter can be used to filter out undesired high frequencies.

- Gain *K*: The filter gain *K* determines the low frequency gain of the filter.
- Frequency *f*: The filter roll off starts at frequency *f* in Hz. Lower frequencies will pass the filter, higher frequencies will be filtered out.
- Quality *Q*: The filter quality indicates how steep the phase response will decrease at the frequency *f*. A quality *Q* > 0.707 results in a peak in the magnitude response at frequency *f*.

Transfer function of the filter

$$H(s) = \frac{K(2\pi f)^2}{s^2 + 2\pi f \frac{s}{Q} + (2\pi f)^2}$$

Notch Filter

The notch filter can be used to filter out an undesired frequency.

- Gain *K*: The filter gain *K* determines the frequency gain of the filter.
- Frequency *f*: The filter attenuates at frequency *f* in Hz. Lower and higher frequencies will pass the filter.
- Quality *Q*: The filter quality indicates how steep the magnitude response will be at the frequency *f*.

Transfer function of the filter:

$$H(s) = K\frac{s^2 + (2\pi f)^2}{s^2 + 2\pi f \frac{s}{Q} + (2\pi f)^2}$$

Universal Notch Filter

The universal notch filter can be used to filter out an undesired frequencies.

- Gain *K*: The filter gain *K* determines the low frequency gain of the filter.
- Frequency *fz*: The filter attenuates at the zero frequency *fz* in Hz. Lower and higher frequencies will pass the filter. A lead in the phase response will start at this zero frequency *fz*.
- Frequency *fp:* The filter amplifies at the pole frequency *fp* in Hz. The phase lead will stop at this pole frequency *fp*.
- Damping *dz*: The filter damping of the zero *dz* indicates how steep the magnitude response will be at the frequency *fz*. More damping will result in a less steep behavior.
- Damping *dp*: The filter damping of the pole *dp* indicates how steep the magnitude response will be at the frequency *fp*. Again, more damping will result in a less steep behavior.

Transfer function of the filter:

$$H(s) = K \frac{\frac{1}{(2\pi f_z)^2} s^2 + \frac{2d_z}{2\pi f_z} s + 1}{\frac{1}{(2\pi f_p)^2} s^2 + \frac{2d_p}{2\pi f_p} s + 1}$$

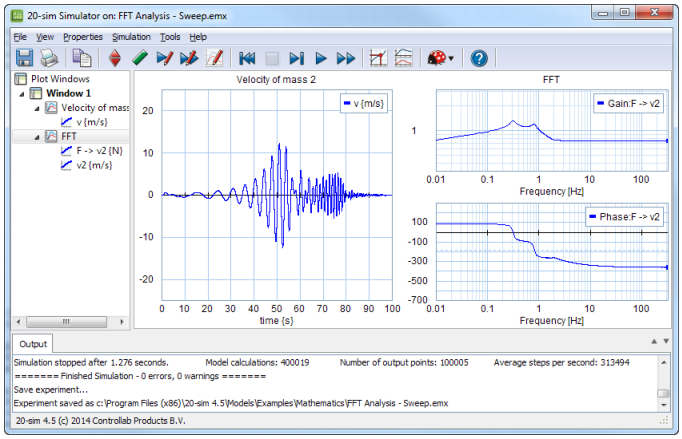
8.4 Frequency Domain Toolbox

8.4.1 FFT Analysis

FFT Analysis

In FFT-Analysis, the Fast Fourier Transform is used to calculate the frequency contents of simulation signal. Three representations are supported:

- Amplitude and Phase: This representation shows the amplitude and phase of the signal as function of the frequency, similar to a bode plot. If you want show the transfer function of a system, you have to choose the option *Pair-wise transfer function*, which is explained below in the FFT Settings.
- Frequency Plot: This representation shows the frequency contents of the signal. Summation of the peak values yields the average power of the original time signal.
- Power Spectral *Density*: This representation shows the frequency contents of the signal. Integration over the frequency range yields the average power of the original time signal.



FFT plot showing the power spectral density.

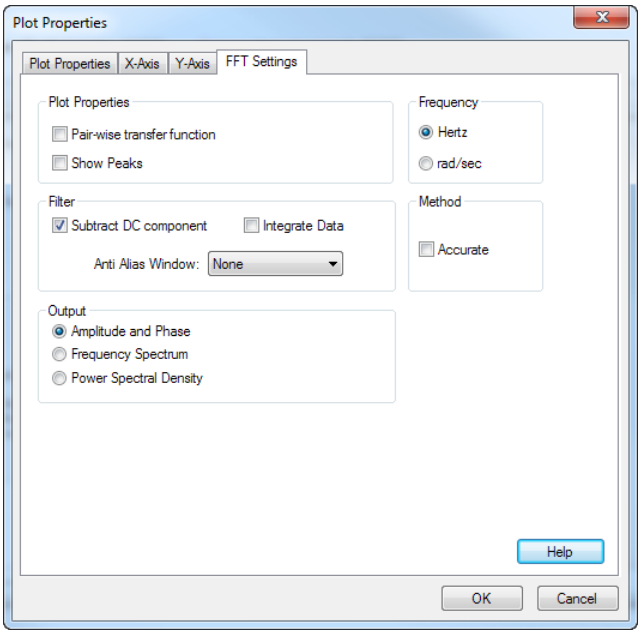
FFT analysis can be performed in the Simulator by choosing the **FFT Analysis** command from the **Tools** menu. You can also choose to add an **FFT plot** in the Simulator tree. In both cases a new FFT plot is shown.

Plot Properties

You can define the FFT plot by opening the plot properties (from the menu choose **Properties** - **Plot**). Four tabs are shown: *Plot Properties*, *X-Axis*, *Y-Axis* and *FFT Settings*.

- The Plot Properties tab shows the general settings of the plot, similar to every standard plot.
- The X-axis tab shows the X-Axis variable of the FFT input. The default value is time and should not be changed.
- The Y-axis tab shows the variables that are selected to be shown as FFT curves, similar to every standard plot.
- The FFT-Settings tab shows the options that you can choose for the FFT plot.

FFT Settings



Items

- *Plot Properties*
  - *Pair-wise transfer function*: Shows an FFT with the frequency contents of the first plot variable subtracted from the second plot variable. Select this option if you want to show the FFT of a transfer function. The first plot variable is the input of the transfer function and the second plot variable is the output of the transfer function.
  - *Show Peaks:* Choose this option to show the peak values in the FFT plot.
- *Filter*
  - *Subtract DC-component*: If your signal contains a DC-component, the resulting Spectral Density plot will show a high peak at 0 Hz, which may obscure all other frequencies. Choose Subtract DC-component, to remove this peak from the Spectral Density plot.
  - *Integrate Data*: You can integrate the FFT data to inspect the energy contents of each frequency. This option is useful for analyzing resonance frequencies. Integrated FFT-data will usually show a sudden increase exactly at these frequencies.
- *Anti-Alias Window*: You can choose a Hamming, Hahn or Quadratic window to prevent aliasing.
- *Frequency*: Show the results in Herz or radians per second.
- *Output:* select the desired representation of the frequency data:
  - *Amplitude and Phase*

- *Frequency Spectrum*
- *Power Spectral Density*

- *Status Window*: The bottom part of the window shows the number of simulation points and the chosen frequency range.

Quality

The quality of an FFT plot depends on the contents of the time domain simulation:

- The number of simulation points per second, determine the frequency range of the FFT. If you want to increase the frequency range to higher frequencies, choose a smaller step size or a smaller maximum step size of the integration method.
- The quality of the FFT depends on the total time of the simulation. Increase the finish time of the simulation to get a better FFT plot.
- Make sure that you system is excited properly. If you give an input signal that does not contain higher frequencies, the FFT plot will not show them.

FFT Window

The Fast Fourier Transform is an approximation of the standard Fourier Transform, using a time limited set of data. The begin and end parts of this limited data set may lead to aliasing effects (i.e. yielding not existing frequency peaks). Especially when using small data sets aliasing may lead to unwanted results.

To reduce aliasing, the data set can be preprocessed using special windows. These windows reduce the values at the begin and end of the data set and thus reduce the aliasing effects. Given an input array A[i] and an output array B[i], with = 1,2,...,N, 20-sim supports the following windows:

- *None*: no preprocessing A == B
- *Hamming Window*: B(i) = (0.54 + 0.46*cos(pi*(i-1)/(N-1)))*A(i), i = 1,2,...,N.
- *Hann Window*: B(i) = 0.5*(1 + cos(pi*(i-1)/(N-1)))*A(i), i = 1,2,...,N.
- *Quadratic Window*: B(i) = (1 - 2*((i-1)/(N-1))**2)*(1 - (i-1)/(N-1))*A(i), i = 1,2,..., (N-1)/2+1; B(i) = 2*(1 - ((i-1)/(N-1))**3)*A(i), i = (N-1)/2+2,...,N.

8.4.2 Model Linearization

Linearize

In the *Simulator*, select the **Tools** menu and then click the **Linearize Model** to start linearization. Linearization will generate a linear, state-space description (linear system) out of any 20-sim model:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$
$$x(0)$$

Where:

x = state vector
x(0) = state vector initial value
dx/dt = state vector derivative
y = output
u = input

And:

A = system matrix
B = input gain matrix
C = output gain matrix
D = direct link gain matrix

This linear system shows an input-output behavior that is closely related to the input-output behavior of the original model.

The result is shown in the Linear System Editor. In the Linear System Editor you can choose out of several plot options to show the response of the linear system, such as step responses and Bode plots.

Working Point

Non-linear models can show variable behaviour. Think for example of an arm. When you stretch it and push against a wall, it is very stiff. When your arm is bent and you push it against a wall, it will be more compliant. Depending on which state it is in, linearization of a non-linear system will therefore result in different linear systems.

To get the correct linear system, we have to bring the non-linear system into the correct state. This can be done by giving it the proper inputs and simulate the system for a while until the desired state is reached. This is called the working point. When the working point is reached, the simulation must be stopped to perform linearization.

When linearization is performed without simulation, the working point is the initial state of the non-linear system.

Symbolic and Numeric

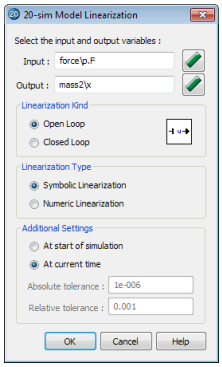
In the 20-sim Simulator, out of an existing (non-linear) model, a symbolic linear-system can be derived. This means that states, rates and matrix elements are related to the parameters of the original model:

As a result you will get a linear system with the original model parameters that can be changed at will, without having to linearize again.

Symbolic linearization is not possible if the original model contains functions that cannot be differentiated. For such models linearization has to be performed numerically. The resulting linear system has to be recalculated through linearization for different model parameters.

Procedure

1. Run a simulation until the working point is reached.
2. Stop the simulation at operating point an select the **Linearize** command from the **Tools** menu.
3. Enter your options in the Linearization Dialog.



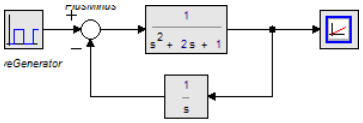
The Linearization Toolbox.

Items

- *Input*: Use the button to select the input (u).
- *Output*: Use the button to select the output (y).
- *Linearization Kind*: Choose between open loop and closed loop linearization.
- *Linearize Type*: Choose between Symbolic and Numeric linearization.
- *Additional Settings:* When you are performing numeric linearization, select *At current time* when you want to linearize at the operating point where the simulation was stopped. Select *At start of simulation* when you want to linearize at the start of the simulation.
- *Tolerances:* 20-sim calculates a numerical linear state space model by small deviations of the complete model from the chosen setpoint. The deviations are based on the given absolute and relative tolerances. For most models the default values are sufficient. For stiff models you can change these values by hand.
- *OK*: click the OK button to start Linearization.

Linearization Explained

During linearization you are asked to enter an input signal and an output signal of your (non-linear) model. During linearization, 20-sim will derive a corresponding linear system. It is important to understand that 20-sim will **cut** the model at the input. This will be explained with the example model below. It shows a linear system with feedback.

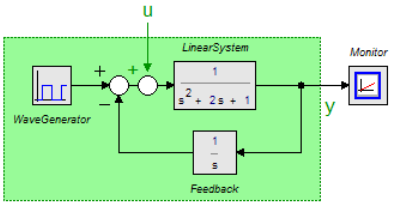


Closed Loop

Suppose we would choose the following settings:

input u: WaveGenerator\output
output y: Monitor\input
kind: Closed Loop

This means that 20-sim will **add an input u** and linearize the model between this variable and the variable *Monitor\output* (equal to y in the figure below). 20-sim will thus linearize the system between u en y and (in this example) and yield the **closed loop system**.

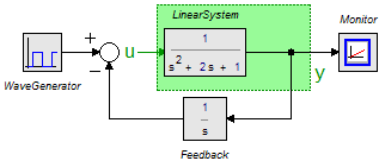


Open Loop

Suppose we would choose the following settings:

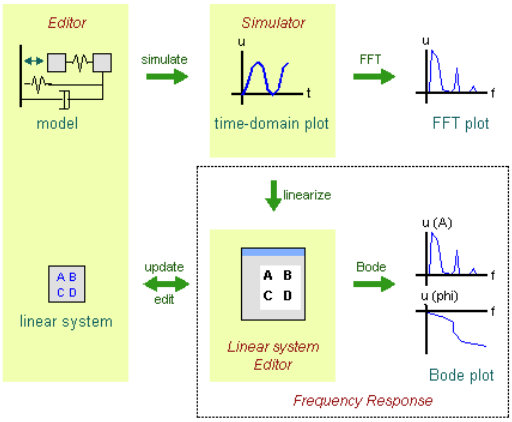
input u: PlusMinus\output
output y: Monitor\input
kind: Open Loop

This means that 20-sim will **cut** the model at the variable *PlusMinus\output* (equal to u in the figure below) and linearize the model between this variable and the variable *Monitor \input* (equal to y in the figure below). 20-sim will thus linearize the system between u en y and (in this example) and yield the **open loop system**.



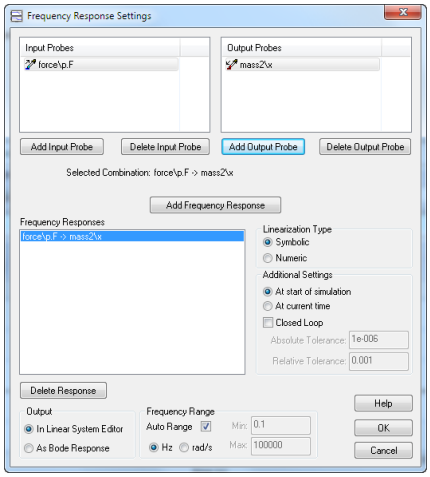
Frequency Response

Next to time domain analysis through the standard time domain plots, 20-sim can also show results in the frequency domain through FFT analysis and Bode Plots. The picture below shows the various actions and resulting plots.



Bode plots originate from linear systems theory. A bode plot shows the amplitude and phase response (as a function of the frequency) of a linear system. In 20-sim linear systems can be derived in the Simulator out of any linear or non-linear model (model linearization). The resulting linear system is shown in the Linear System Editor. In this editor you can generate Bode plots. The generation of Bode plots can be automated in the Frequency Response dialog. In this dialog you can define the linearization settings and desired linearization output (Linear System Editor or Bode plot).
Procedure

1. Select the **Frequency Response** from the command from the **Properties** menu.
2. Enter your options in the Frequency Response Dialog.



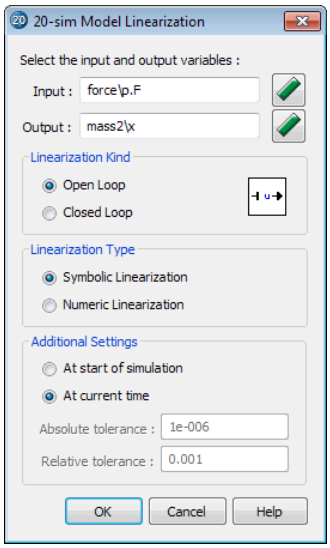
The Frequency Response dialog.

Items

- *Input Probes*: Enter possible input signals for linearization here.
- *Output Probes*: Enter possible output signals for linearization here.
- *Frequency Response:* With the Add Frequency Response you can add new input/ output pairs. For every input/output pair linearization will be performed with a corresponding linear system as result.
- *Symbolic/Numeric:* Select the desired linearization method.
- *Closed Loop:* By default all linearization is open loop. Select the *Closed Loop* option to preform closed loop linearization.
- *At start of simulation/At current time*: Select the time of the linearization.
- *Tolerances:* For numeric linearization you can enter the absolute and relative tolerances.
- *Output*: Select the desired output of the linearization.
- *Frequency Range*: Select the desired frequency range.
- *OK*: click the OK button to close the dialog.

Linearization Tolerances

When using numerical linearization, absolute and relative tolerances can be set. In this section is explained how 20-sim uses these tolerances to derive a linear system for the model equations.



Algorithm

Suppose you have chosen the following tolerance values.

alpha = absolute tolerance (e.g. 1e-6)
beta = relative tolerance (e.g. 1e-3)

Suppose we have the following model:

*ddt (x) = 4 * x + 2 * u; // with u = 1*
*y = 1 * x + 3 * u; // with x = 2*

During the linearization procedure (suppose linearization from u to y) 20-sim will vary the input variable and state variables. First we will show the variation for the input u:

*u' = (1 + beta)*u + alpha*
*u'' = (1 - beta)*u - alpha*
*delta_u = beta*u + alpha*

this will yield

x_dot' = 10.002002 and y' = 5.003003
x_dot'' = 9.997998 en y'' = 4.996997

Out of this 20-sim will calculate the B and D vectors of the state-space ABCD representation:

*B = (x_dot' - x_dot'') / (2 * delta_u) = 2*
*D = (y' - y'') / (2 * delta) = 3*

The variation of state variables is done accordingly:

*x' = (1 + beta)*x + alpha*
*x'' = (1 - beta)*x - alpha*
*delta_x = beta*x + alpha*

this will yield

x_dot' = 10.008004 en y' = 5.002001
x_dot'' = 9.991996 en y'' = 4.997999

Out of this 20-sim will calculate the A matrix and C vector of the state-space ABCD representation:

*A = (x_dot' - x_dot'') / 2*delta_x = 4*
*C = (y' - y'') / 2*delta_x = 1*

The example system is linear, so the corresponding ABCD representation will give an equal system. The shown algorithm works equivalent for non-linear models. Suppose we have the following non-linear model:

*ddt (x) = 4 * sin(x) + 2 * u; // with u = 1*
*y = 1 * x + 3 * u; // with x = 2*

This will yield:

B = 2, D = 3, A = -1.668445, C = 1;

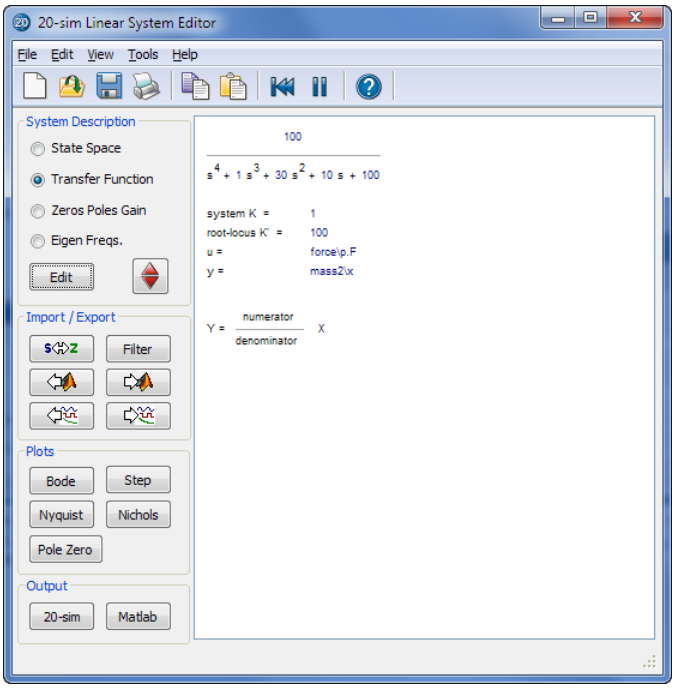
Tolerance Values

The absolute tolerance is necessary only if the input or state is zero. In that case the contribution of the relative tolerance is zero an will not give a contribution to the algorithm. There is however a problem with the absolute tolerance. Suppose the input = 1e-12 and the state = 1e+6 (ill-scaled model). For the input an absolute tolerance of 1e-6 will yield a far too large deviation, while the same absolute tolerance is negligible compared to the state. Only if both the state and input are nonzero the absolute tolerance can be made non-zero to yield good results.

8.4.3 Linear System Editor

Linear System Editor

The Linear System Editor of 20-sim allows you to enter or edit linear time- invariant models. It supports continuous and discrete-time single input / single output (SISO) systems with and without time delay. You can open the Linear System Editor from the Tools menu of the 20-sim Editor or by clicking Go Down on the Linear System model. A linear system can also be the result of a linearization operation in the 20-sim Simulator.



Menu

Some menu items are of particular importance.

- File: You can use the file menu to open, import and export linear systems.
- Edit - Tolerance: You can specify the tolerance for transforming between state-space and transfer functions.
- Edit - Reduce System: You can reduce the system order by pole-zero cancelation. The distance between poles and zeros to match for cancelation is given by the **Reduction Tolerance**.
- View: the view menu can be used for holding, clearing and refreshing plots.

System Description

- Pressing the **Edit** button will open a dialog for editing the linear system. Each description (State Space, Transfer Function, Zeros Poles Gain or Eigen Frequency) has a special dialog and can be used to specify continuous-time and discrete-time systems.
- In the 20-sim Simulator, out of an existing (non-linear) model a symbolic linear-system can be derived by means of linearization. This means that the relevant model parameters are preserved in the Linear System and the parameters button can be used to change parameters.

Import/Export

- Pressing the **s<->z** button will transform a continuous-time linear system into a discrete-time linear system and back.
- Pressing the **Filter** button opens the Filter Editor, where a filter can be designed. This filter can then be combined with the current linear system or replace the current linear system. If the linear system is a discrete-time system, the designed analog filter is automatically transferred into its digital equivalent.
- Pressing the **From Matlab** button and the **To Matlab** button allows for an instant exchange of the linear system with the Matlab workspace. Information is transferred numerically (no parameter relations are preserved)
- The button is only active when a Linear System has been imported through linearization. Clicking the button will import parameters from the simulation.
- The button is only active when a Linear System has been imported through linearization. Clicking the button will export the current parameters to the simulation.

Plots

You can inspect the time- and frequency responses of the linear system using:

- Step Response
- Bode Plot
- Nyquist Plot
- Nichols Chart
- Poles and Zeros (including root locus)

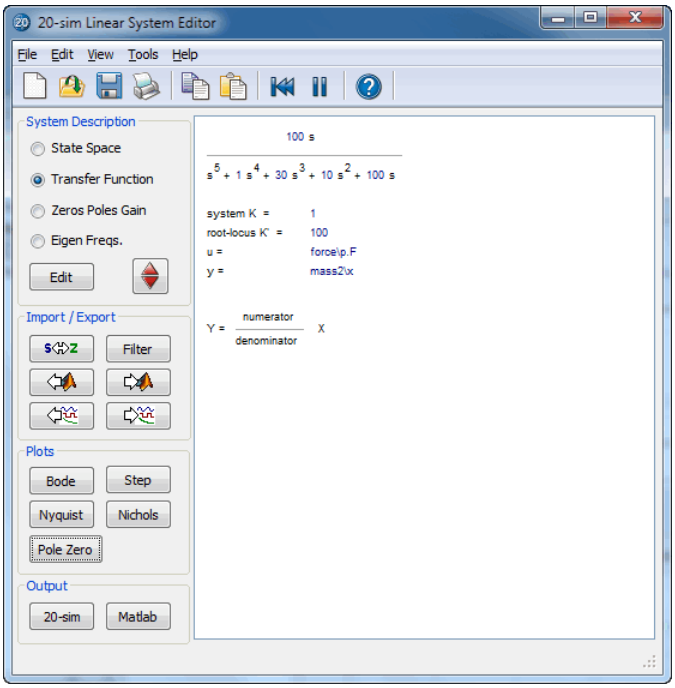
Output

- Clicking the **20-sim** button will export the linear system as a new 20-sim submodel.
- Clicking the **Matlab** button will export the 20-sim Linear System to a Matlab m-file. If the Linear System is symbolic, all parameter relations are preserved.

Symbolic Linear Systems

In the 20-sim Simulator, out of an existing (non-linear) model a symbolic linear-system can be derived by means of linearization. This means that the relevant model parameters are preserved in the Linear System.

The relation between the original model parameters and the linear system elements is always shown in the white square of Linear System editor, just below the system description. An example is shown in the first figure below:



As you can see, the some elements of the A, B, C and D matrix are related to the spring constant and masses of the original model.

Editing

A symbolic linear system can be edited by changing the original model parameters. Click the Edit Parameters button and a Parameters Editor pops-up. Change the desired parameters, close the Parameters Editor and the linear system will updated.

Updating Parameters

When the parameters have been changed in the Linear System Editor you can update the parameters in the Simulator by clicking the **Update to Simulator** button . Click the **Update from Simulator** button to do it the other way.

Exporting

You can export a symbolic Linear System to Matlab, with preservation of the parameter relations, by clicking the *Matlab* button or selecting *Export to Matlab* command from the *File* menu. This will generate an m-file that you can use in Matlab.

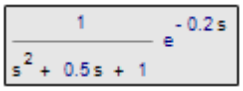
Continuous and Discrete Linear Systems

The Linear System Editor works on continuous time as well as discrete time system. Pressing the **s<->z** button will transform a continuous-time linear system into a discrete-time linear system and back. You can choose between a Bilinear transformation (Tustin), a Forward Euler transformation and a Backward Difference transformation. When a continuous-time system is transferred, the user is asked to specify the sample time of the discrete-time system.

If you want to transfer a linear system directly (i.e. replace the z by and s or vice versa), click the **Edit** button and select or deselect the Discrete Sample time.

Output Delay

In the Linear System Editor you can add an output delay. Open the editor and click the *Edit* button to edit the system. Now a editor will open where you can set the output delay. The effects of this output delay is shown in the various plots that you can show of a linear system. The unit of the output delay is seconds.



The output delay is shown as an exponential added to the transfer function.

Editor

State Space Models

State space models use linear differential equations (continuous) or difference equations (discrete) to describe system dynamics. They are of the form:

$$\frac{dx}{dt} = A \cdot x + B \cdot u \qquad x_{n+1} = A \cdot x_n + B \cdot u_n$$

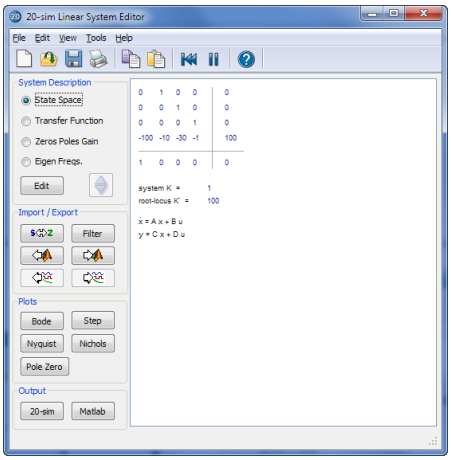
$$y = C \cdot x + D \cdot u \qquad y_n = C \cdot x_n + D \cdot u_n$$

(continuous) (discrete)

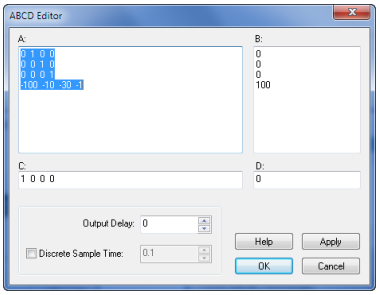
with for example:

$$A = \begin{bmatrix} 0 & 0.5 \\ -1 & -1 \end{bmatrix} \quad B = \begin{bmatrix} -0.1 \\ 0.1 \end{bmatrix}$$
$$C = \begin{bmatrix} 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 0 \end{bmatrix}$$

where x is the state vector and u and y are the input and output vectors. These models may arise from the equation of physics, from state space identification or as the result of linearization.



You can enter state space models by selecting the *State Space* button and clicking the *Edit* button.



This opens an editor in which you can enter the A, B, C and D matrices. Depending on the selection of the Discrete Linear System check box (and Sample Time) the system is a continuous-time or discrete-time system. You can enter the matrix elements in the white space areas. Separate column elements with Spaces or Commas. Enter new rows by clicking the Enter key (new line) or using a semicolon. Brackets (e.g. [...]) may be used to denote the a matrix or vector.

The A-matrix, shown in the figure above can for example be entered as:

0 1
-0.5 -1

or

0,1;-0.5,-1

or

[0,1;
-0.5,-1]

Output Delay

To inspect the effects of time delay in your model, you can add output delay. The result will be visible in the various plots that you can show of a linear system. The unit of the output delay is seconds.

Discrete Sample Time

If you want to transfer a linear system from continuous time to discrete time directly (i.e. replace the s by a z), select Discrete Sample time and fill in the sample time value. You can also transfer back directly by deselecting Discrete Sample time.

Commands

- *Help*: Open the help file.
- *Apply:* Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero).
- *OK*: Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero) and close the editor.
- *Cancel*: Do not apply any changes to the system and close the editor.

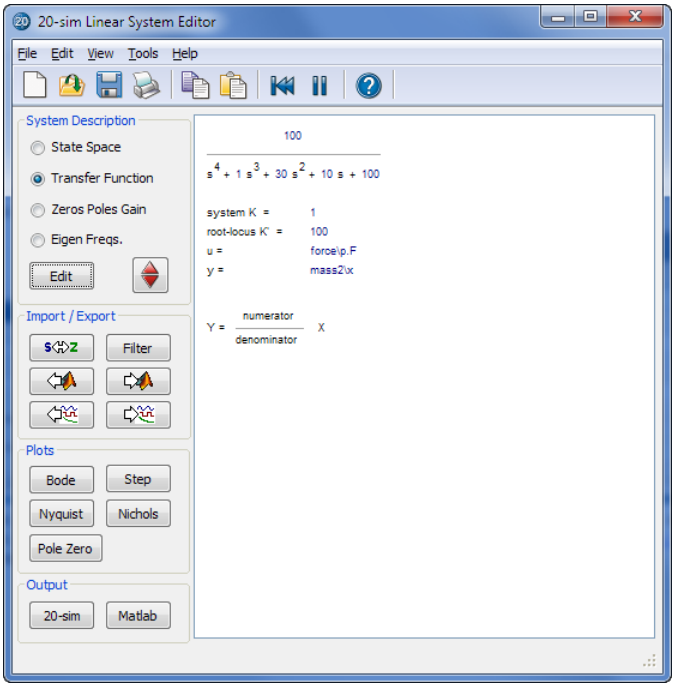
Transfer Functions

A continuous time or discrete time SISO transfer function:

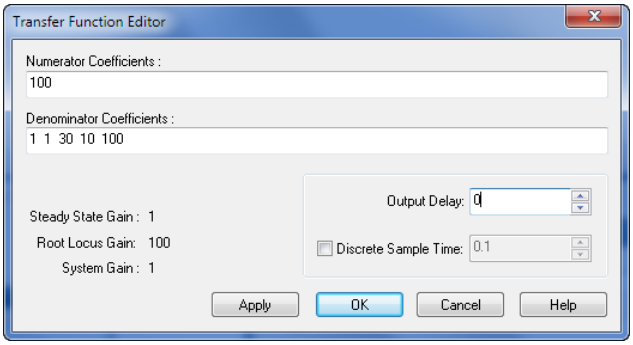
$$h(s) = \frac{n(s)}{d(s)} \qquad h(z) = \frac{n(z)}{d(z)}$$

(continuous) *(discrete)*

is characterized by its numerator *n* and denominator *d*, both polynomials of the variable *s* or *z*.



You can enter transfer functions by selecting the *Transfer* Function button and clicking the *Edit* button.



This opens an editor in which you can enter the coefficients of the numerator and denominator polynomials. You can enter the elements in the white space areas. Separate the elements with Spaces. Polynomials should be entered in descending powers of s or z. The Steady State Gain, Root Locus Gain and System Gain are parameters, that are automatically derived from the transfer function.

In the editor shown above the transfer function

$$\frac{0.1 \cdot s + 0.1}{s^2 + s + 0.5}$$

was given. You can enter the coefficients as:

0.1 0.1

and

1 1 0.5

or

0.1,0.1

and

1,1,0.5

Output Delay

To inspect the effects of time delay in your model, you can add output delay. The result will be visible in the various plots that you can show of a linear system. The unit of the output delay is seconds.

Discrete Sample Time

If you want to transfer a linear system from continuous time to discrete time directly (i.e. replace the s by a z), select Discrete Sample time and fill in the sample time value. You can also transfer back directly by deselecting Discrete Sample time.

Commands

- *Help*: Open the help file.
- *Apply*: Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero).
- *OK*: Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero) and close the editor.
- *Cancel*: Do not apply any changes to the system and close the editor.

Gains

Steady State Gain

Given the transfer function:

$$h(s) = \frac{n(s)}{d(s)} = \frac{n_i \cdot s^i + n_{i-1} \cdot s^{i-1} + .. + n_0 \cdot s^0}{d_j \cdot s^j + d_{j-1} \cdot s^{j-1} + .. + d_0 \cdot s^0}$$

the **Steady State Gain** is defined as:

$$K_{SS} = \lim_{s \to 0} h(s)$$

Note that the steady state gain can be zero or infinite depending on the element values of the numerator and denominator!

Root Locus Gain

Given an system described by the transfer function:

$$h(s) = \frac{n(s)}{d(s)} = \frac{n_i \cdot s^i + n_{i-1} \cdot s^{i-1} + .. + n_0 \cdot s^0}{d_j \cdot s^j + d_{j-1} \cdot s^{j-1} + .. + d_0 \cdot s^0}$$

This transfer function can be rewritten in pole zero notation with

$$h(s) = K_{RL} \frac{(s - z_i)(s - z_{i-1}) \ldots (s - z_0)}{(s - p_j)(s - p_{j-1}) \ldots (s - p_0)}$$

where *pi .. p1* are the poles and *zi .. z1* are the zeros of the system. The gain *KRL* is known as the *Root Locus Gain.* Note that it can easily be derived from the transfer function as:

$$K_{RL} = \frac{first\ element\ of\ n(s)}{first\ element\ of\ d(s)} = \frac{n_i}{d_j}$$

System Gain

Given the transfer function:

$$h(s) = \frac{n(s)}{d(s)} = \frac{n_i \cdot s^i + n_{i-1} \cdot s^{i-1} + .. + n_0 \cdot s^0}{d_j \cdot s^j + d_{j-1} \cdot s^{j-1} + .. + d_0 \cdot s^0}$$

If *n0* and *d0* are unequal to zero, this transfer function can be rewritten in pole zero notation with:

$$h(s) = K_S \frac{(z'_i s + 1)(z'_{i-1} s + 1) \ldots (z'_0 s + 1)}{(p'_j s + 1)(p'_{j-1} s + 1) \ldots (p'_0 s + 1)}$$

The gain *KS* is known as the *System Gain*. If *n0* is zero and *n1* is nonzero an equivalent notation can be found with an extra s multiplied:

$$h(s) = K_S \frac{(z'_i s + 1)(z'_{i-1} s + 1) \ldots (z'_1 s + 1) \cdot s}{(p'_j s + 1)(p'_{j-1} s + 1) \ldots (p'_0 s + 1)}$$

If more numerator element are zero, extra multiplications with s are added. The same goes for denominator elements equal to zero. In general the System Gain can be derived from the transfer function as:

$$K_S = \frac{last\ nonzero\ element\ of\ n(s)}{last\ nonzero\ element\ of\ d(s)}$$

Relating the Gains

The elements of the System Gain are related to the poles and zeros of the Root Locus Gain as:

$$z_i' = \frac{-1}{z_i} \quad , \quad p'z_j = \frac{-1}{p_j} \quad etc.$$

If *z0* and *p0* are unequal to zero, the following equation holds:

$$K_{RL} = K_S \frac{z_i \cdot z_{i-1} \cdot \ldots \cdot z_0}{p_j \cdot p_{j-1} \cdot \ldots \cdot p_0}$$

The Root Locus Gain and the Steady State Gain are related as:

$$K_{SS} = K_{RL} \frac{-z_i \cdot -z_{i-1} \cdot \ldots \cdot -z_0}{-p_j \cdot -p_{j-1} \cdot \ldots \cdot -p_0}$$

Zeros and Poles

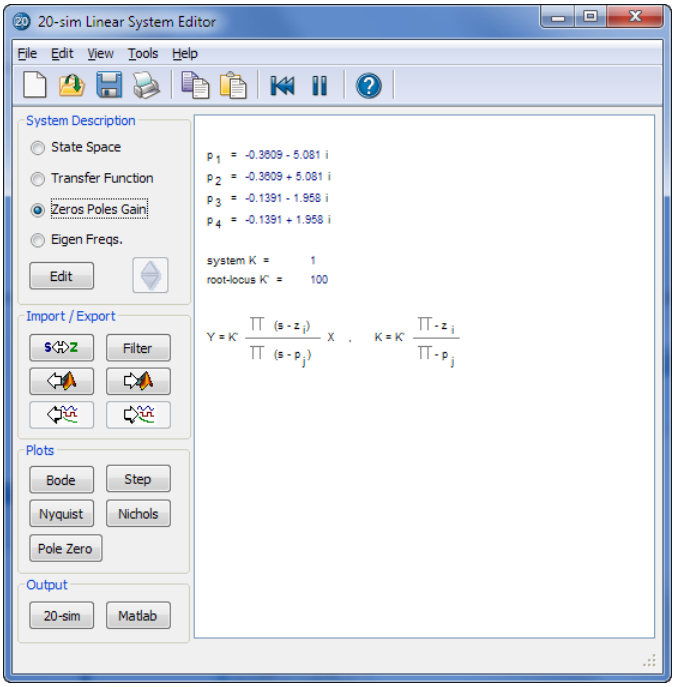
A continuous-time SISO transfer function can described by the transfer function:

$$h(s) = \frac{n(s)}{d(s)} = \frac{n_i \cdot s^i + n_{i-1} \cdot s^{i-1} + .. + n_0 \cdot s^0}{d_j \cdot s^j + d_{j-1} \cdot s^{j-1} + .. + d_0 \cdot s^0}$$

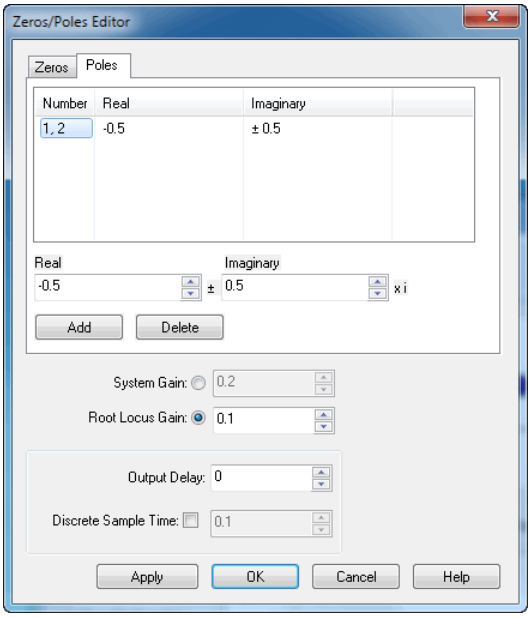
This transfer function can be rewritten in pole zero notation with

$$h(s) = K_{RL} \frac{(s - z_i)(s - z_{i-1}) \ldots (s - z_0)}{(s - p_j)(s - p_{j-1}) \ldots (s - p_0)}$$

where *pi .. p1* are the poles and *zi .. z1* are the zeros and *KRL* is the Root Locus Gain of the system. The same can be done for a discrete-time SISO transfer function.



You can enter zeros and poles by selecting the *Zeros & Poles* button and clicking the *Edit* button.



This opens an editor in which you can enter the real and imaginary parts of the zeros and poles as well as the Root Locus Gain. If preferred, you can also enter the System Gain. Note that zeros and poles always have conjugate when the imaginary part is non-zero. I.e. when you enter a pole with imaginary part 0.5 an extra pole is added with imaginary part -0.5.

Output Delay

To inspect the effects of time delay in your model, you can add output delay. The result will be visible in the various plots that you can show of a linear system. The unit of the output delay is seconds.

Discrete Sample Time

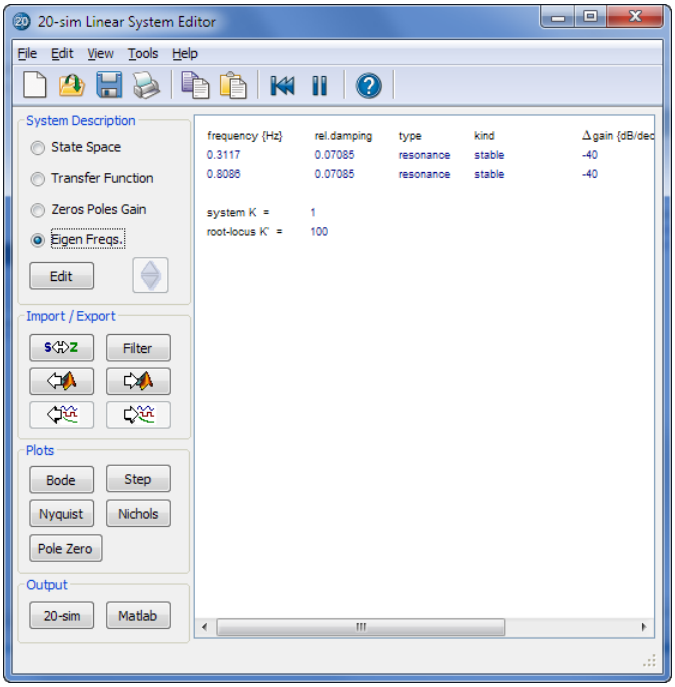
If you want to transfer a linear system from continuous time to discrete time directly (i.e. replace the s by a z), select Discrete Sample time and fill in the sample time value. You can also transfer back directly by deselecting Discrete Sample time.

Commands

- *Add/Delete:* Add or delete selected poles or zeros.
- *Help*: Open the help file.
- *Apply:* Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero).
- *OK*: Apply the current changes of the system, recalculate each plot that is active (step, Bode, Nyquist, Nichols, pole zero) and close the editor.
- *Cancel:* Do not apply any changes to the system and close the editor.

Eigen Frequencies

The Eigen Frequencies view is closely related to the pole zero notation and bode plot. It shows the resonance frequencies and anti-resonance frequencies that result from the given poles and zeros as well as some characteristic parameters from the bode plot.

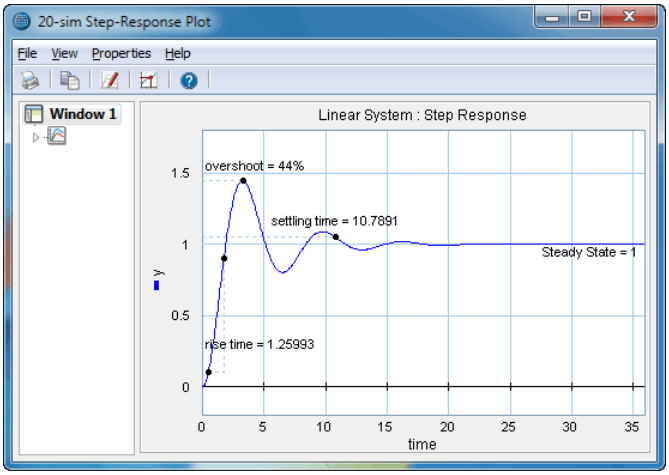


You can enter transfer functions by selecting the *Eigen Freqs.* button and clicking the *Edit* button.

Step Response

The *Step Response* command calculates the systems response y on a unit step input:

u = 0 (time < 0)
u = 1 (time >= 0)



20-sim automatically generates an appropriate range for the time response, based on the system dynamics. With the Plot Properties command (right mouse menu), you can change this horizon and recalculate the step response (click the Step command again).

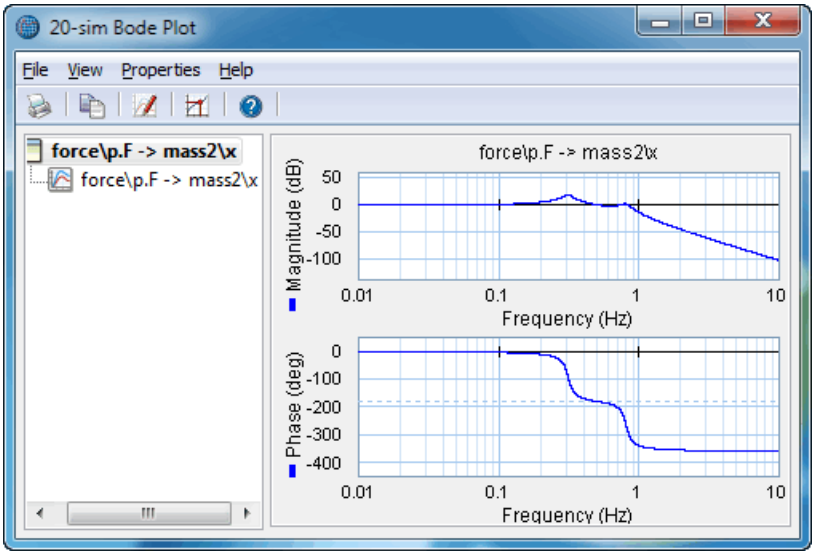
Plot Options

Using the right mouse menu, you can use several options:

- *Plot Properties*: Set the general plot properties for the step response and specify the curve properties.
- *Numerical Values*: Inspect numerical values.
- *Step Characteristics*: Display rise time, overshoot, settling time and the steady state value of the step response.
- *Copy to Clipboard*: Copy the plot to the windows clipboard.
- *Print*: Print the plot.
- *Zoom in*: Show a detail of the plot.
- *Zoom out*: Show a larger portion of the plot.
- *Zoom Normal*: Show the complete plot.

Bode Plots

Bode Plots show the amplitude and phase of a linear system as function of the frequency. Bode plots can be shown for every 20-sim model through Linearization. During Linearization you are asked to enter the input variable and output variable for which linearization should be performed. After that the linear system is calculated and shown in the Linear System Editor. From the Linear System Editor you can generate a bode plot. These actions can also be predefined using the Frequency Response command of the *Properties* menu.



20-sim automatically generates a range of logarithmically displayed frequencies, based on the system dynamics. With the **Plot Properties** command (**right mouse menu**), you can change this horizon and recalculate the bode response (click the Bode command again).

The magnitude part of the plot can be displayed in dB or in absolute values. The phase part can be displayed in radians or degrees. The frequency can be displayed in radians per second or in Hz.

Plot Options

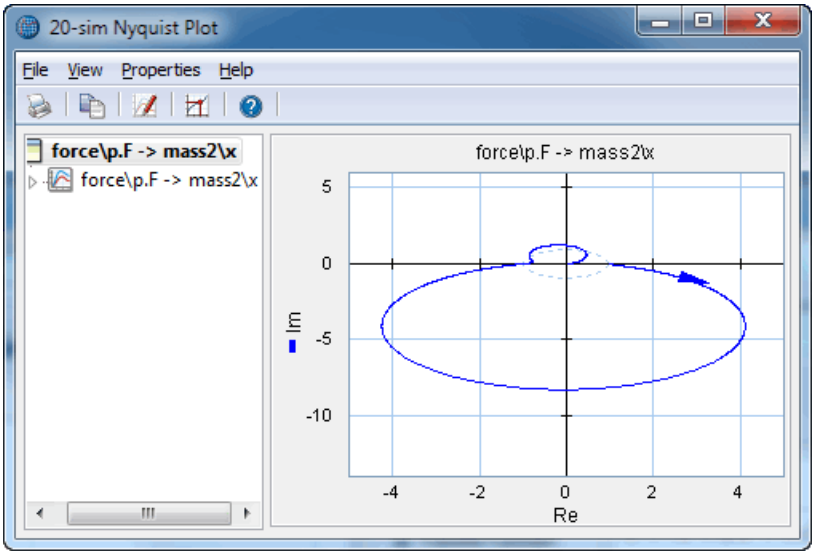
Using the toolbar or the right mouse menu, you can use various options:

- *Plot Properties:* Set the plot properties.
- *Numerical Values:* Inspect numerical values.
- *Magnitude (dB):* Display magnitude in decibels
- *Magnitude (-):* Display magnitude in absolute values.
- *Phase (rad):* Display phase in radians.
- *Phase (deg):* Display phase in degrees.
- *Frequency (rad/sec):* Display frequency in radians per second.
- *Frequency (Hz):* Display frequency in Hz.

- *Phase/Gain Margins:* Display the gain and phase margins.
- *Modulus Margin:* Display the Modulus Margin.
- *Unwrap Phase*: Display the phase plot as a continuous plot by selecting this option or as a folded plot between -180 and 180 deg. by deselecting this option.
- *Peak Response:* Display the peak response.
- *Copy to Clipboard*: Copy the plot to the windows clipboard.
- *Print:* Print the plot.
- *Zoom in:* Show a detail of the plot.
- *Zoom out*: Show a larger portion of the plot.
- *Zoom Normal:* Show the complete plot.

Nyquist Diagram

The **Nyquist** command computes the Nyquist plot of a system.



20-sim automatically generates a range of real and imaginary parts, based on the system dynamics. With the **Plot Properties** command (right mouse menu), you can change this horizon and recalculate the Nyquist diagram (click the **Nyquist** command again).

Plot Options

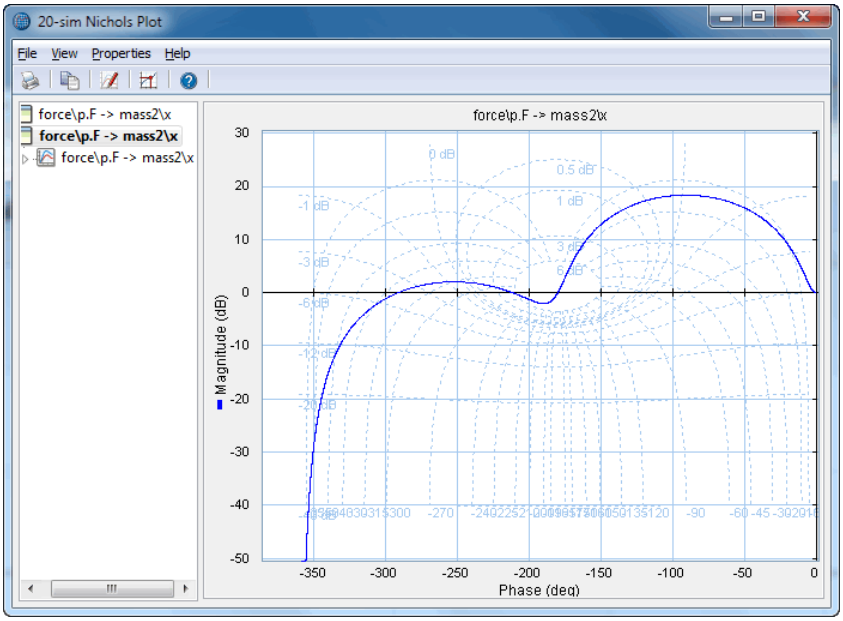
Using the right mouse menu, you can use several options:

- *Plot Properties:* Set the plot properties.
- *Numerical Values:* Inspect numerical values.
- *Phase/Gain Margins*: Display the gain and phase margins.
- *Modulus Margin:* Display the Modulus Margin.

- *Copy to Clipboard:* Copy the plot to the windows clipboard.
- *Print:* Print the plot.
- *Zoom in:* Show a detail of the plot.
- *Zoom out:* Show a larger portion of the plot.
- *Zoom Normal*: Show the complete plot.

Nichols Chart

The *Nichols* command computes the Nichols chart of a system.



20-sim automatically generates a range of logarithmically displayed frequencies, based on the system dynamics. With the **Plot Properties** command (right mouse menu), you can change this horizon and recalculate the Nichols chart (click the **Nichols** command again). The magnitude part of the plot can be displayed in decibels (dB) or in absolute values. The phase part can be displayed in radians or degrees.

Plot Options

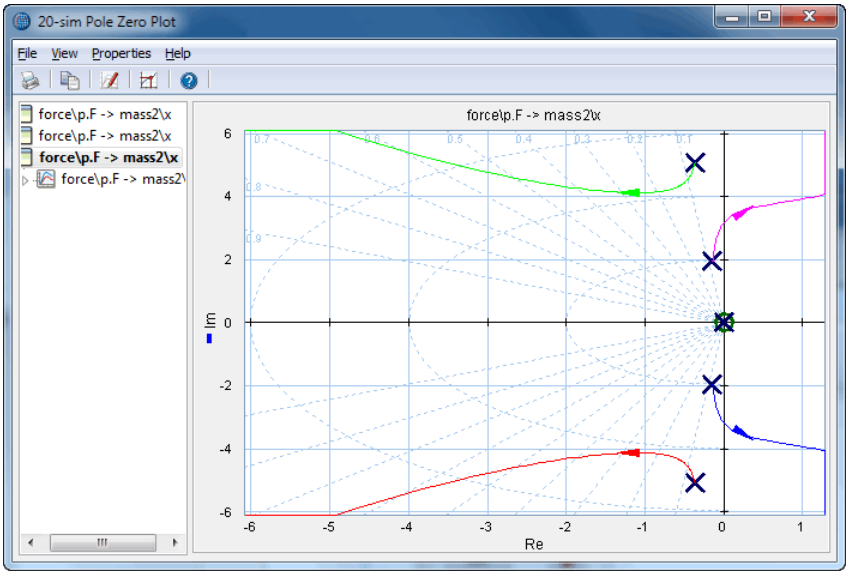
Using the right mouse menu, you can use several options:

- *Plot Properties:* Set the plot properties.
- *Numerical Values:* Inspect numerical values.
- *Magnitude (dB):* Display magnitude in decibels
- *Magnitude (-)*: Display magnitude in absolute values.
- *Phase (rad)*: Display phase in radians.
- *Phase (deg)*: Display phase in degrees.

- *Frequency (rad/sec):* Display frequency in radians per second.
- *Frequency (Hz):* Display frequency in Hz.
- *Phase/Gain Margins:* Display the gain and phase margins.
- *Modulus Margin:* Display the Modulus Margin.
- *Copy to Clipboard*: Copy the plot to the windows clipboard.
- *Print*: Print the plot.
- *Zoom in*: Show a detail of the plot.
- *Zoom out:* Show a larger portion of the plot.
- *Zoom Normal*: Show the complete plot.

Pole Zero Diagram

The **Pole Zero** command plot the poles and zeros of a system and computes the root locus plot.



20-sim automatically generates a range of real and imaginary parts, based on the system dynamics. With the **Plot Properties** command (right mouse menu), you can change this horizon and recalculate the pole zero diagram (click the **Pole Zero** command again).

Root Locus

You can show the rootlocus plot by selecting **Root Locus** from the right mouse menu. Inspect the root locus gain by selecting **Numerical Values** from the right mouse menu.

Plot Options

Using the right mouse menu, you can use several options:

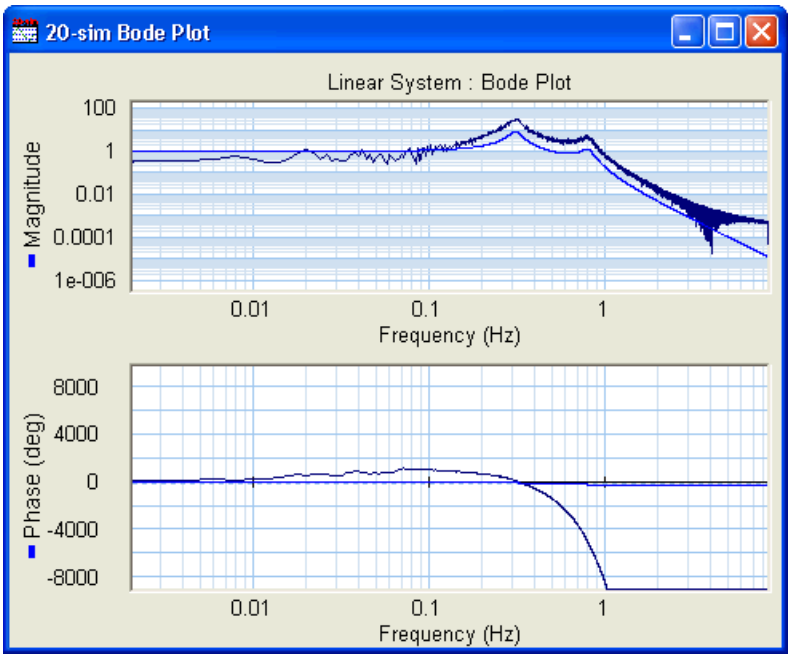
- *Plot Properties*: Set the plot properties.

- *Numerical Values*: Inspect numerical values.
- *Root Locus:* Show or hide the root locus plot.
- *Copy to Clipboard:* Copy the plot to the windows clipboard.
- *Print*: Print the plot.
- *Zoom in:* Show a detail of the plot.
- *Zoom out:* Show a larger portion of the plot.
- *Zoom Normal:* Show the complete plot.

Import Data

You can import data from measurements or other software tools to compare it with the response of your model. 20-sim accepts data in two formats:

- *Gain-Phase:* The data should be stored in a text-file with three columns. The first column should contain the frequency and the second and third column, the corresponding gain and phase data.
- *Real-Imag*: The data should be stored in a text-file with three columns. The first column should contain the frequency and the second and third column, the corresponding real and imaginary data.



Import Data

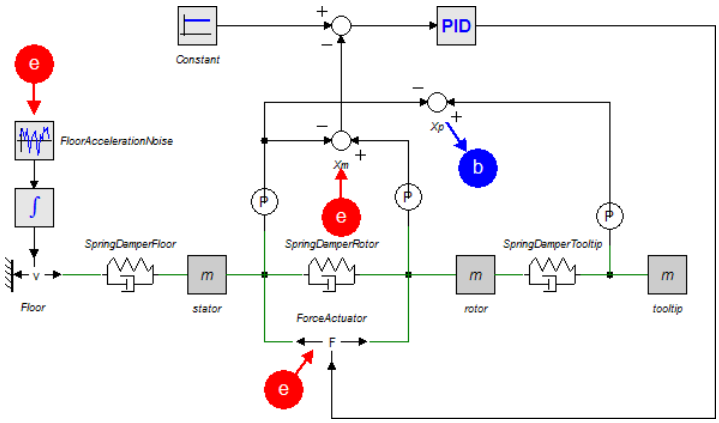
1. Open the Linear System Editor with your linear system.
2. Open the plot (e.g. Bode, Nyquist) in which you want to show the data.
3. Set the Magnitude (- or dB), Phase (rad or deg) and Frequency (rad/s or Hz) according to the data you want to import.
4. In the Linear System Editor from the **File** menu select **Import.**
5. Select Import **Gain-Phase** or **Import Real-Imag**.

Now a file dialog opens helping you to select the file to import. When you have selected a file and closed the dialog, the data should be visible in your plot. An example is shown in the figure above.

8.4.4 Dynamic Error Budgeting

Dynamic Error Budgeting

The performance of precision machines is mostly limited by the disturbances that are injected in these machines. These disturbances are often stochastic in nature. Dynamic Error Budgeting is a method whereby the effect of these disturbances on the final performance can be calculated. The advantage of this method is that it enables the designer to enter the contributions of the individual disturbances and view and optimize the overall machine performance.



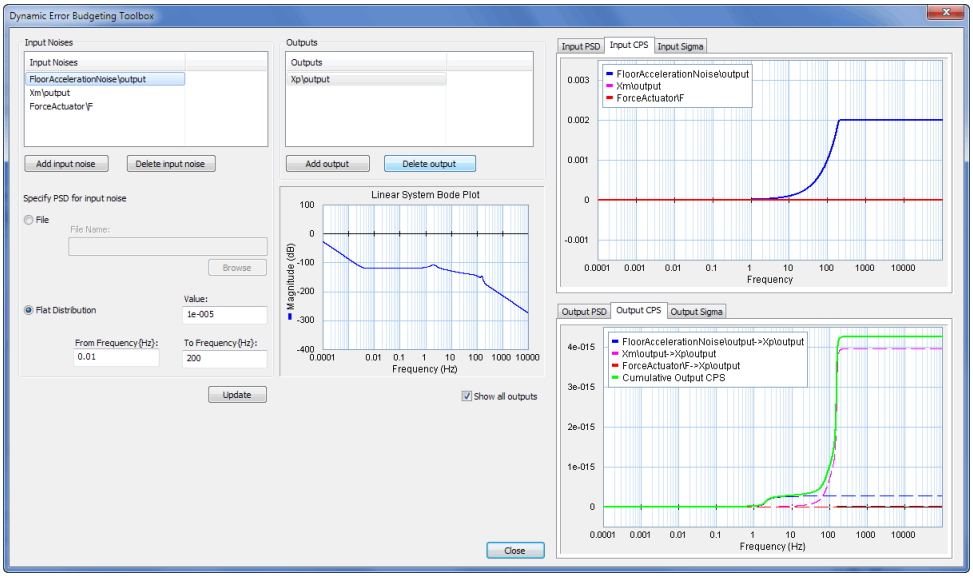
The Dynamic Error Budgeting toolbox shows the total error as a result of injected disturbances.

Running the Toolbox

1. Open the example model *Dynamic Error Budgetting* from the *Getting Started/ Frequency Domain Toolbox.*

You can open the Dynamic Error Budgeting tool in the *Simulator*:

2. From the **Tools** menu select **Frequency Domain Toolbox** and **Dynamic Error Budgeting**.



The Dynamic Error Budgeting tool.

The tool allows you to enter disturbances (as power spectral density) in the *Input Noises* section.

3. For each disturbance you have to select a corresponding variable by clicking the **Add input noise** button.

Each disturbance is effectively a summation to the chosen variable, just like closed loop linearization. You can inspect each disturbance in the graph on the top right.

Next you have to select an output variable, where the result of the disturbances is calculated.

4. Select the output by clicking the **Add Output** button.

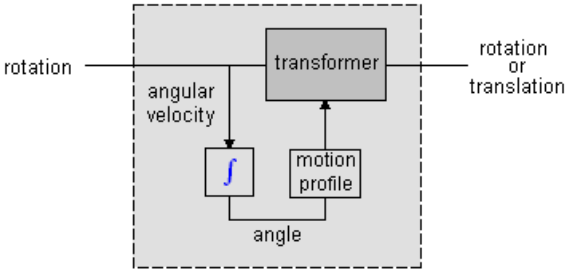
In the graph on the bottom right you can see the resulting error at the selected output as a result of the disturbances. The error is given in the form of a power spectral density (PSD) and cumulative power spectral density (CPS). The square root of the final value of the CPS is equal to the standard deviation of the output error. The standard deviations are shown in the *Output Sigma* tab.

8.5 Mechatronics Toolbox

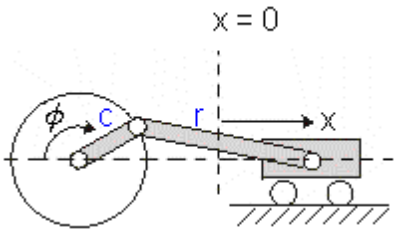
8.5.1 Cam Wizard

How to use the Cam Wizard

Cams and mechanisms are all based on the same principle. The motion of an input axis is transformed to an output axis or translation. The transformation is a function of the input angle. This function is called the **cam motion profile**.



The 20-sim cam wizard helps you to generate cam and mechanism models. You can use various types of motion profiles which are continuous in velocity, acceleration or even in jerk! An example of a mechanism that can be generated by the Cam wizard is the crank-rod mechanism, where a rotary input motion is transformed to a translating output motion.



The models that are generated by the 20-sim Cam wizard are fully dynamic! This means:

- The models describe the output velocity as function of the input velocity but also the input torque as function of the output load.
- The speed of the input axis does not have to be constant nor does the output load.
- No inertia, stiffness or other dynamic behavior is included, but this can be easily incorporated by coupling elements from the 20-sim library (inertias, springs etc.) to the input or output of the model.

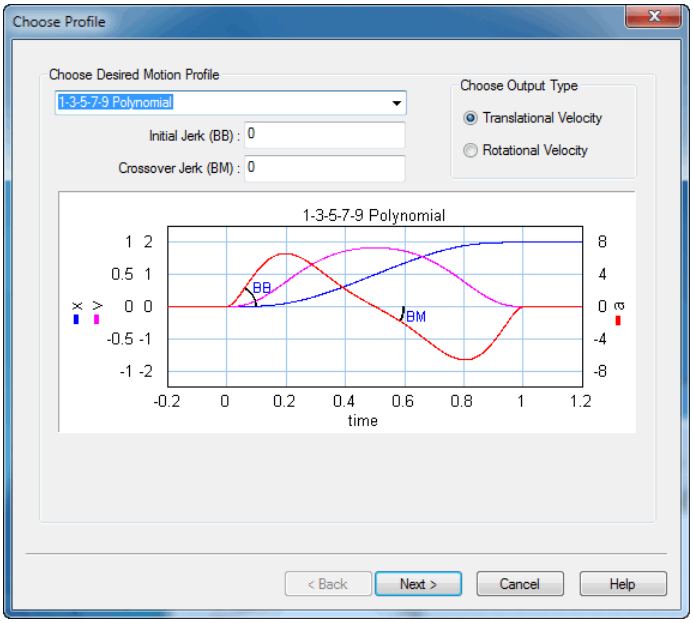
To generate a cam or mechanism model in 20-sim, follow the next steps.

1. Open the model library and drag the model *CamManipulator.emx* to the editor.

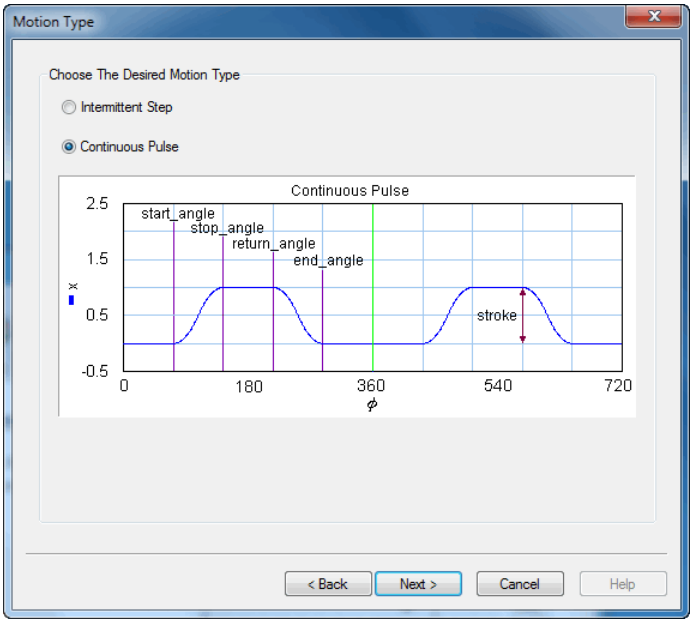
2. Make an input connection (rotation) and an output connection (rotation/translation) to the model.

3. Select the camwizard model and click Go Down. Now the wizard will be opened.

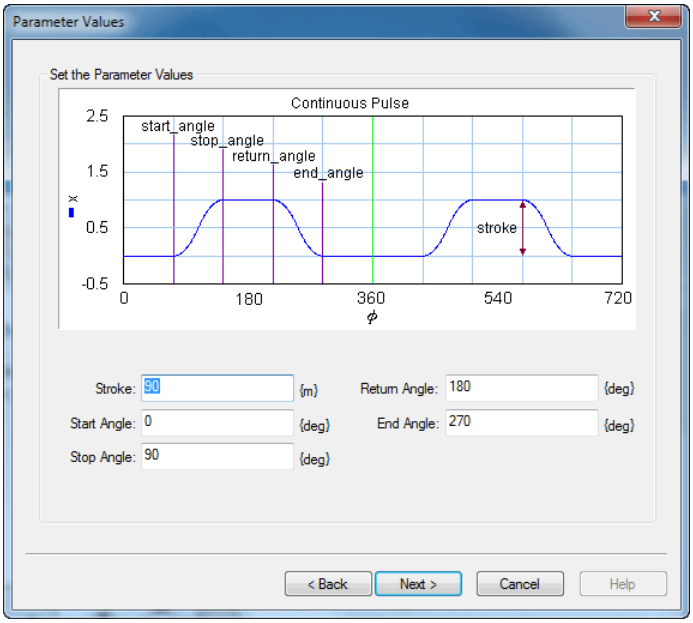
4. Choose the desired motion profile, the profile parameters and type of output (translation/rotation):



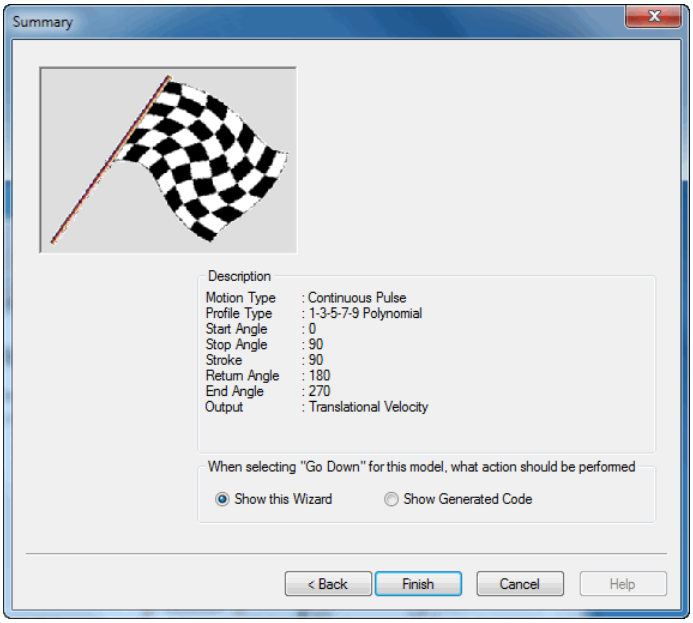
5. Select the type of cam motion profile:



6. Enter the parameters values (Stroke, Start Angle Stop Angle). If you have chosen a continuous pulse as output, you must also enter the values for the Return Angle and End Angle:



7. In the last page, the resulting values are shown:



Note

1. If you have changed the settings of this wizard, you have to process the model first (from the *Model* menu, choose *Check Complete Model*) before the changes become effective.

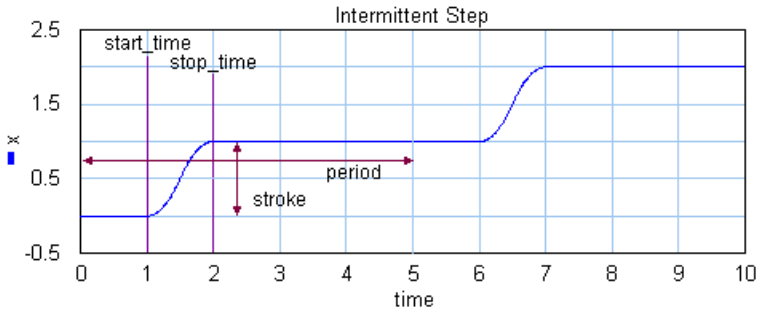
2. You can change parameter values during simulation (from the **Properties** menu select Parameters).

3. This is a masked model that uses the dll-file MotionProfile.dll to open the wizard. This dll-function must be stored in the bin directory of 20-sim. To see the SIDOPS code of a masked model press the *shift-key*, while clicking the *Go Down* command.

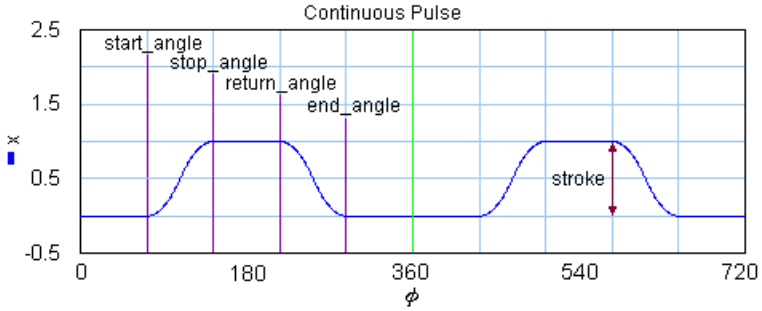
Cam Motion Profiles

Motion Types

Two types of motion profiles can be generated by the Cam wizard: **intermittent steps** and **continuous pulses**. The intermittent step motion does not return to its starting position, but gradually moves further away . It is characterized by the parameters stroke, start_angle and stop_angle.



The **continuous pulse** motion output returns at the end of each cycle returns to the starting position. It is characterized by the parameters stroke, *start_angle, stop_angle, return_angle* and *end_angle.*



Motion Profiles

In mechanical engineering an important part of design is the reduction of vibrations. Every time a construction is in motion, vibrations are induced. When cams or mechanisms are used, the amount of vibrations, depend on the kind of motion that is induced. E.g. a sudden step change will induce violent vibrations in comparison with with a fluent motion. An important parameter to characterize motions, is the order:

1. *Zero Order:* Motions that are discontinuous in the position (e.g. a step).
2. *First Order:* Motions that are continuous in the position but discontinuous in the velocity (e.g. a ramp).
3. *Second Order*: Motions that are continuous in the position and velocity but discontinuous in the acceleration.
4. *Third Order*: Motions that are continuous in the position, velocity and acceleration.

5. *Fourth Order:* Motions that are continuous in the position, velocity, acceleration and jerk.

The specific shape of a motion profile can have a significant influence on the dynamic behavior. Some profiles minimize the maximum velocity, some profile minimize acceleration, while other profiles tend to make a tradeoff between the maximum velocity and acceleration. If we take a standard motion with stroke 1 and motion time 1 sec., the following table can be generated:

profile	order	v_{max}	a_{max}	y(0)	y(1/2)
Ramp	1	1	infinite	infinite	0
Crank-Rod[2]	2	> 1.57	> 4.93	infinite	< -15.5
Trapezoidal	2	2	4	infinite	-infinite
Partial Trapezoidal[1]	2	1.67	4.17	infinite	0
Geneva Mechanism	2	2.41	8.49	infinite	-118.5
Sine	2	1.57	4.93	infinite	-15.5
Cubic	3	2	4	32	-32
Partial Cubic[3]	3	1.67	5.55	55.6	0
Cycloidal	3	2	6.28	39.5	-39.5
Modified Sine	3	1.76	5.53	69.4	-23.2
Modified Trapezoidal	3	2	4.88	61.4	-61.4
MSC50	3	1.26	9.20	173.6	0
MSC%[4]	3	1.37	7.20	113.1	0
3-4-5 Polynomial	3	1.88	5.77	60	-30
1-3-5-7-9 Polynomial[5]	3/4	2.05	10.25	BB	BM

1: Parameter CV = 20%

2: For lw >> lc the Crank Rod profile equals the sine profile, for lw > lc performance deteriorates.

3: Parameter CV = 20%, CA = 20%

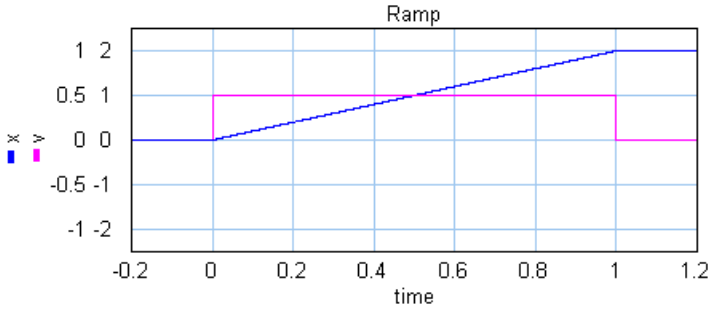
4: Parameter n = 30%, alpha = 10%

5: Parameter BB = 30%, BM = 10%

Here y(0) is the initial jerk (derivative of acceleration) and y(1/2) the crossover (halftime) jerk.

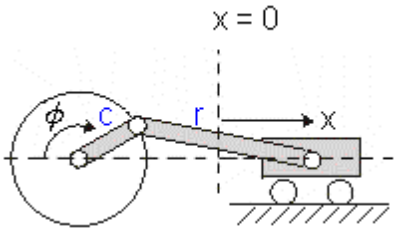
Ramp

The ramp profile is a first order profile. It has a constant velocity and acceleration peaks at the start and end of the motion.

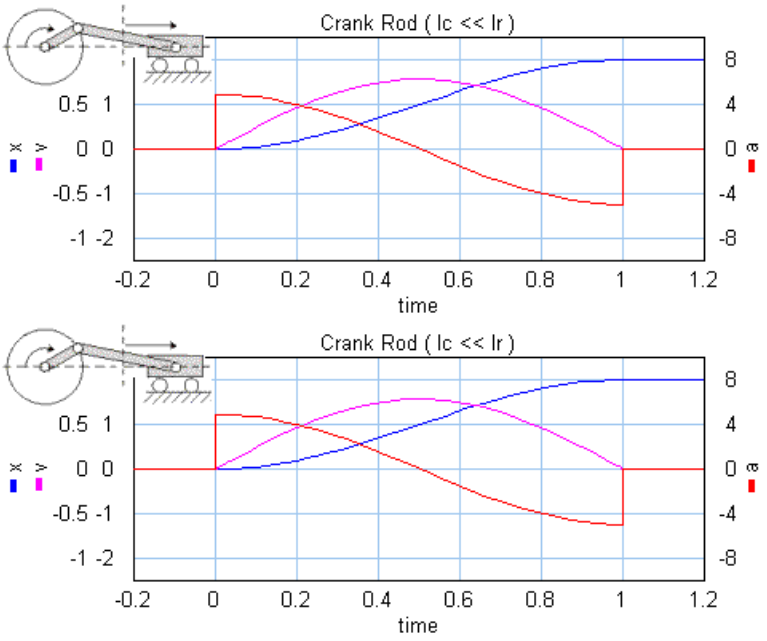


Crank Rod Mechanism

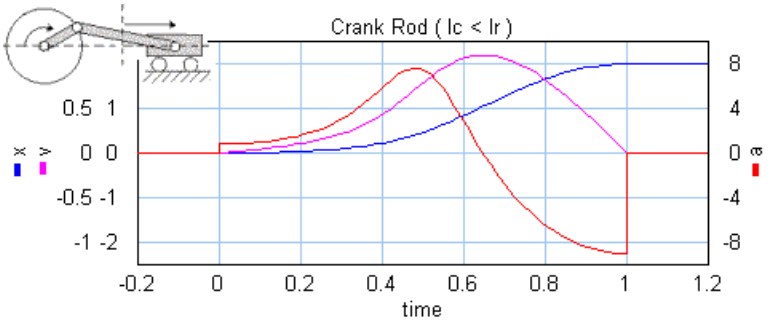
A Crank Rod mechanism converts a rotary motion into a repeating translation and



If the rod is much longer than the crank, the resulting motion profile will approach the sine profile. Below the motion profile is shown for half a rotation of the crank (angle from 0 to 180 degrees in 1 sec.).



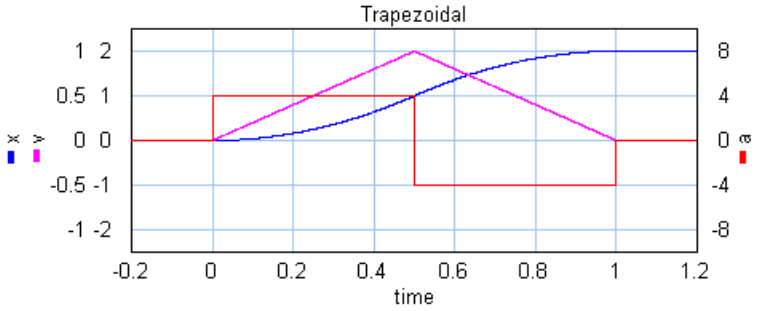
If the rod gets smaller, the maximum velocity and acceleration increase.



Note: A real crank-rod mechanism will not suddenly stop at half a rotation and therefore not show a discontinuous acceleration.

Trapezoidal

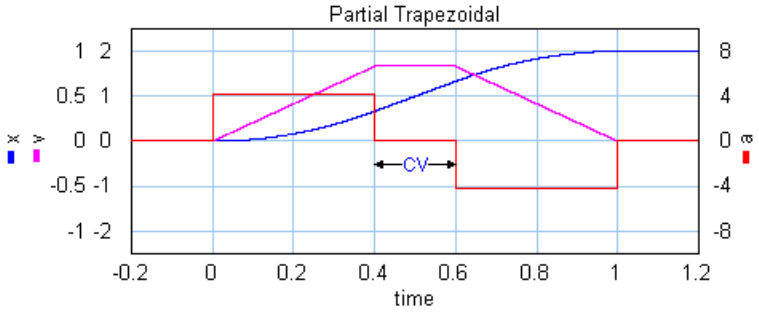
The trapezoidal profile is a second order profile. It has a constant acceleration at the start of the motion and a constant deceleration at the end of the motion.



This profile is most widely used in early servo systems. Because of the discontinuity in the acceleration this motion profile can still induce a lot of vibrations. Therefore in modern servo systems third order profiles are preferred.

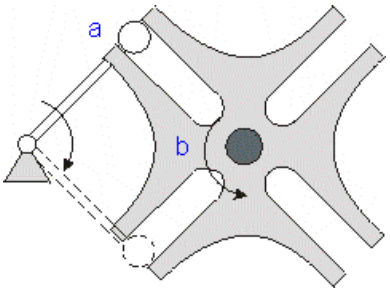
Partial Trapezoidal

The partial trapezoidal profile is a second order profile. It has is equal to the trapezoidal motion, but has a constant velocity part, during a fraction **CV** (%) of the motion.

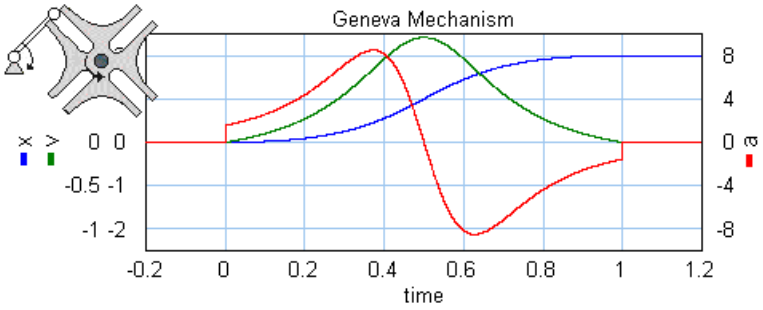


Geneva Mechanism

A Geneva mechanism is an old motion profile generation mechanism. A continuously rotating crankshaft (a) generates an intermittent motion of a second shaft (b). The result is a second order profile.



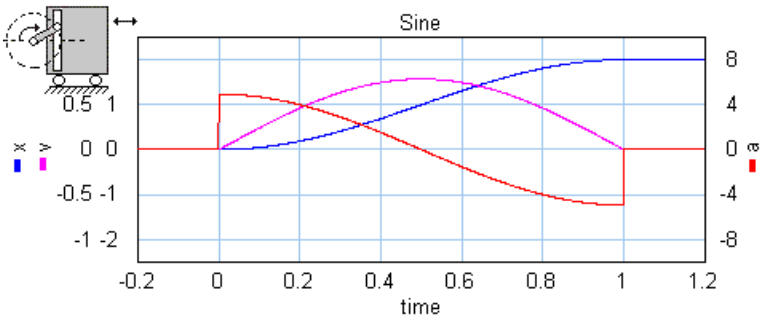
The motion profile generator yields the motion of the second shaft (b). It is shown below:



Sine

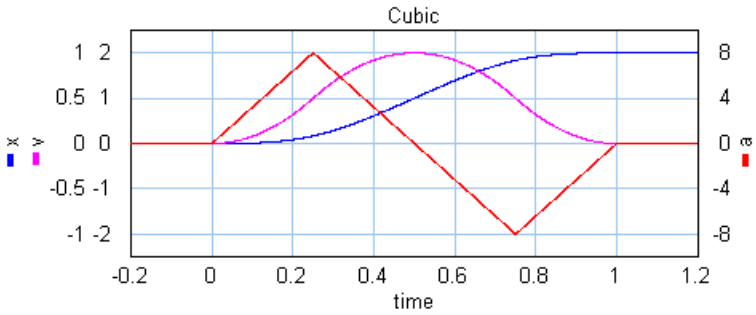
The sine profile is a second order profile of which the displacement can be described as:

*x = 0.5 - cos (p * t) / 2.*



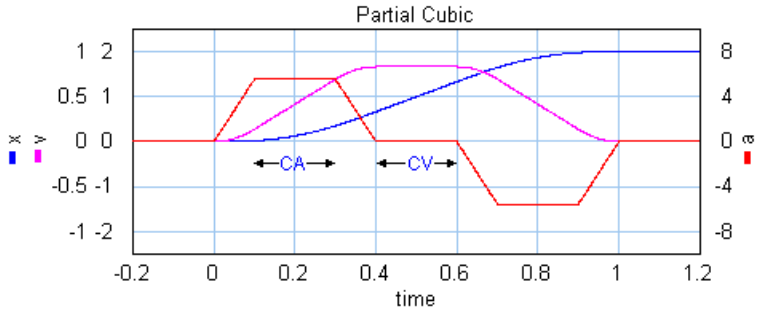
Cubic

The cubic profile is a third order profile of which the acceleration is constantly increasing and decreasing.



Partial Cubic

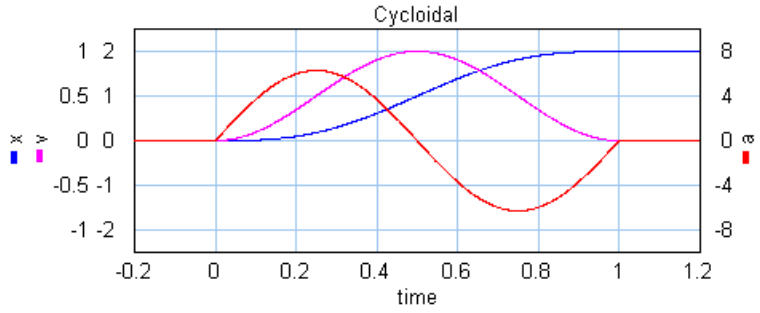
The partial cubic profile (3rd order) is a modified cubic profile with a constant velocity during a fraction **CV** (%) of the motion and a constant acceleration during a fraction **CA** (%) of the motion.



Cycloidal

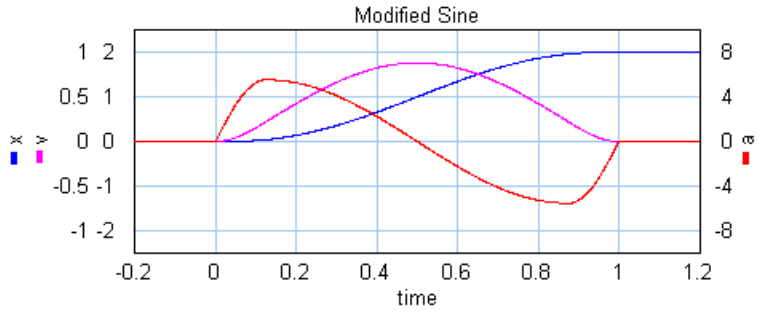
The cycloidal profile is a third order profile, of which the velocity can be described as:

v = stroke(1-cos(t*a))*



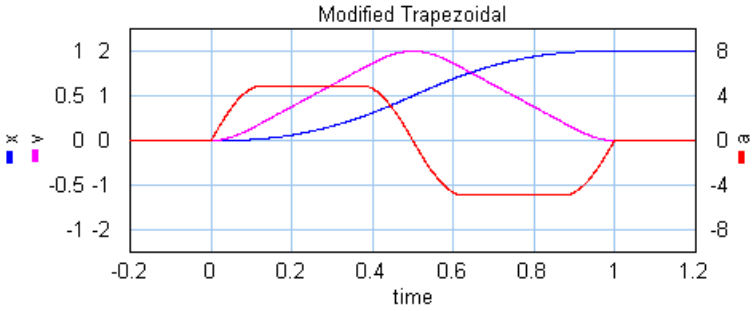
Modified Sine

The modified sine profile is also a third order profile. It is a modification of the cycloidal profile to get a lower maximum velocity and a lower maximum acceleration.



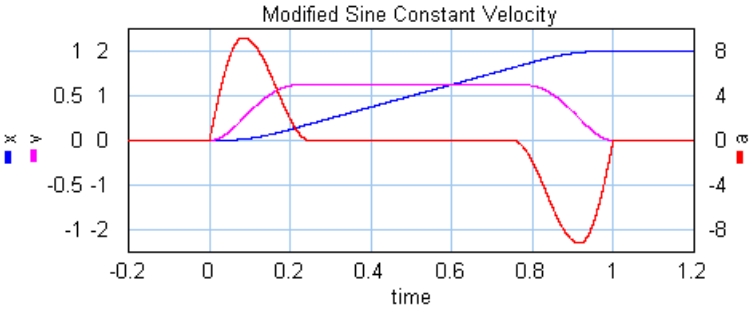
Modified Trapezoidal

The modified trapezoidal profile is a modification of the trapezoidal profile (to make it a third order profile). This profile yields a very low maximum acceleration.



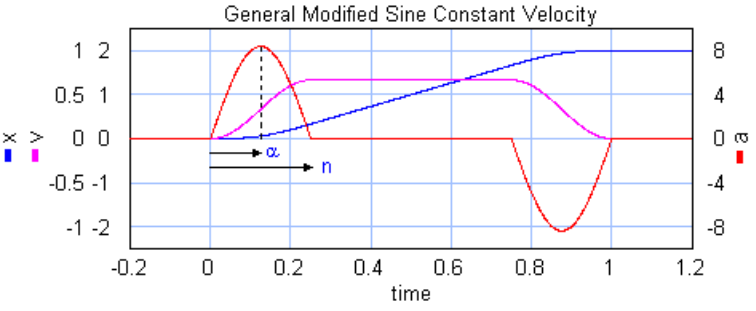
Modified Sine with Constant Velocity (MSC50)

The modified sine with constant velocity profile (3rd order) is a modification of the modified sine profile. It has a constant velocity during 50% of the motion.



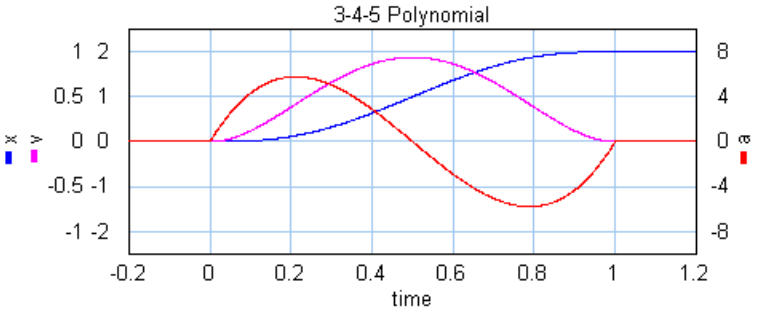
General Modified Sine with Constant Velocity (MSC%)

The modified sine with constant velocity profile (3rd order) is a modification of the modified sine profile. It has a constant velocity during a user definable part of the of the motion. The non-zero acceleration part is defined by two parameters which are both defined as a fraction of the motion. The first parameters **alpha** (%) defines the start of the acceleration and the second parameters **n** (%) defines the end of the acceleration.



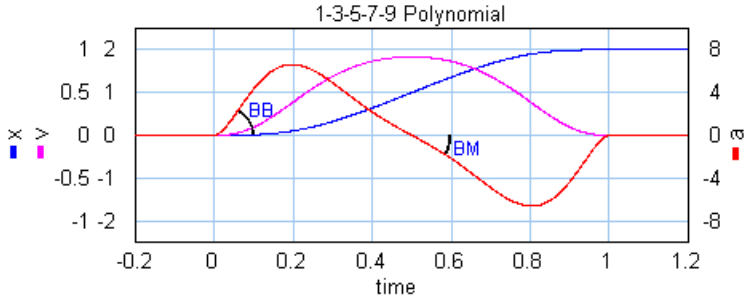
3-4-5 Polynomial

The is a third profile described by a 5th order polynomial.



1-3-5-7-9 Polynomial

The is a third/fourth order profile described by a 9th order polynomial. The profile is characterized by two parameters that denote the initial jerk, **BB**, and the crossover jerk, **BM**. If BB is chosen zero, this a a fourth order profile. If BB is chosen non-zero, this is a third order profile.



8.5.2 Motion Profile Wizard

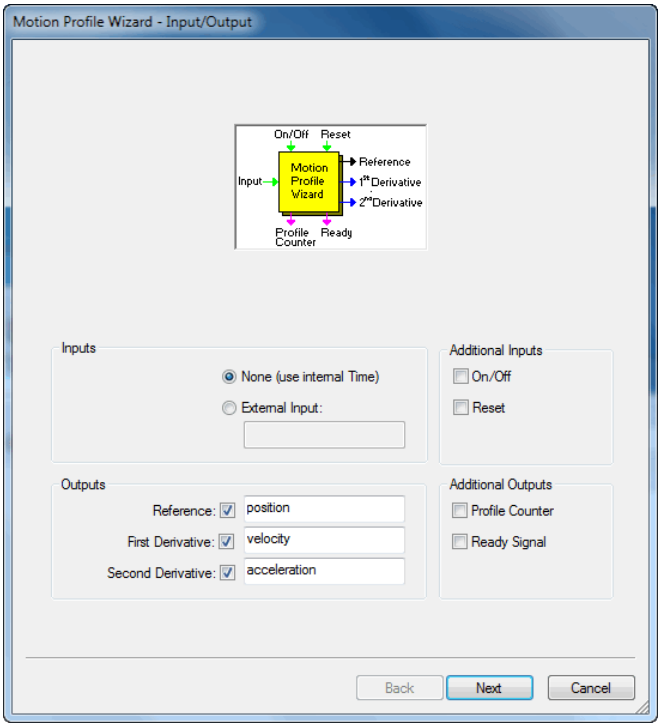
Motion Profile Wizard

The 20-sim Motion Profile Wizard helps you to define motion profiles. To generate such a profile, follow the next steps.

1. Open the 20-sim model library an go to the section Signal\Sources.
2. Drag and drop the model MotionProfileWizard.emx to the editor.
3. Select the model and click **Go Down.**

This is a masked model. Clicking Go Down will open the *Motion Profile Wizard.*

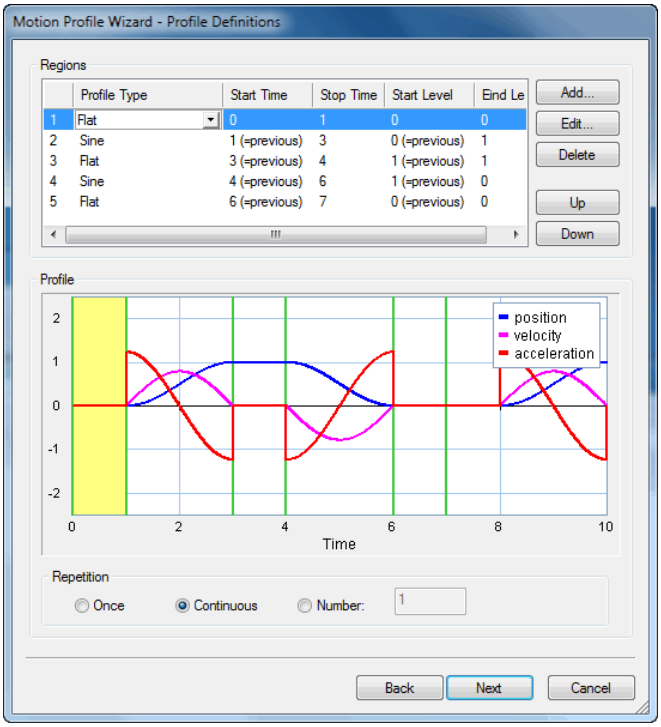
4. Select the inputs and outputs:



A number of inputs and outputs are available:

- *External Input:* Normally the time will be used for the x-axis of the profile. You can however use an external input as x-axis variable.
- *On/Off*: Use this input signal to start (onoff = TRUE) and stop (onoff = FALSE) the profile.
- *Reset*: When the reset signal gets high (reset = TRUE) the profile will be started.
- *Reference*: The motion profile reference signal, e.g. position.
- First Derivative: The first derivative of the motion profile reference signal, e.g. velocity.
- Second Derivative: The second derivative of the motion profile reference signal, e.g. velocity.
- Profile Counter: Gives the number of profiles that have been generated.
- Ready Signal: Gets high (ready = TRUE) when the profile is completely generated.

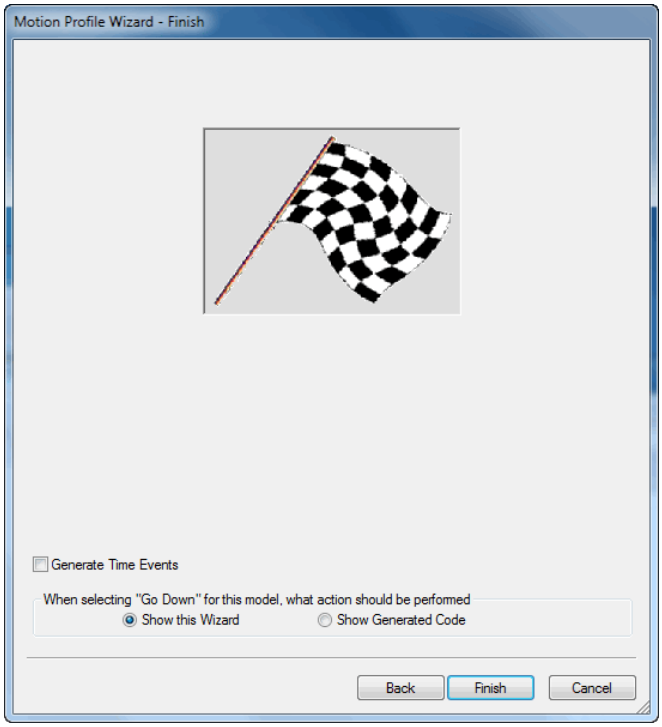
5. Click **Next** to enter the desired profile.
6. You can enter as many profiles as needed to design the complete motion. You can choose out of a number of predefined profiles.



7. Choose the number of times the motion should be repeated.

8. Click **Next** to go to the last page.

In the last page, you can choose to generate Time Events and what to show the next time (the wizard or the generated code). Use the default settings if you are not sure what to use.



9. Click **Next** to go to close the Wizard.
10. In the Editor from the **Model** menu Check Complete Model to make the changes effective.

Motion Profile Wizard (Old Style)

If you select the model *MotionProfile.emx* and click **Go Down**, the old style Motion Profile Wizard will open. This wizard is kept in 20-sim to allow you to run old models. The wizard has been replaced by a new wizard which is far more powerful.

Description

The 20-sim Motion Profile Wizard helps you to define motion profiles. To generate such a profile, follow the next steps.

1. Open the 20-sim model library an go to the section Signal\Sources.
2. Drag and drop the model MotionProfile.emx to the editor.
3. Select the model and click Go Down. Now the wizard will be opened.
4. Select the type of motion.
5. Choose the desired profile and output signals.

6. Enter the profile parameters.

7. Close the wizard.

During simulation the parameter values are used that you have selected in the wizard. These parameters are available in the *Parameters* Editor where you can change them as you would do with any other model. If you have changed the settings of this wizard, you have to process the model first (from the *Model* menu, choose *Check Complete Model*) before the changes become effective.

Motion Profiles

You can choose out of a number of predefined motion profiles in the Motion Profile Wizard.

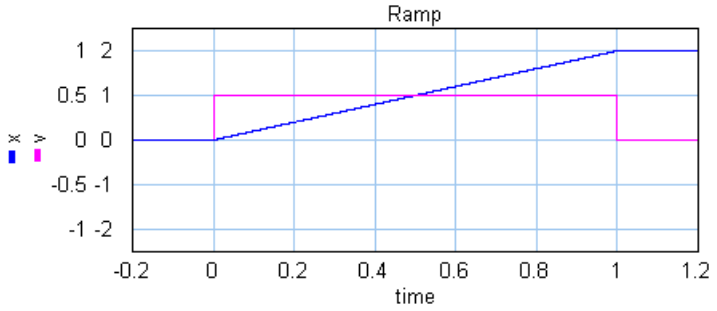
- Flat
- Ramp
- Trapezoidal
- Partial Trapezoidal
- Geneva Mechanism
- Sine
- Cubic
- Partial Cubic
- Cycloidal
- Standard Modified Sine
- Modified Trapezoidal
- Modified Sine with Constant Velocity (MSC50)
- General Modified Sine with Constant Velocity (MSC%)
- 3-4-5 Polynomial
- 1-3-5-7-9 Polynomial

Flat

The flat profile is a zero order profile. It has a constant position, zero velocity and acceleration.

Ramp

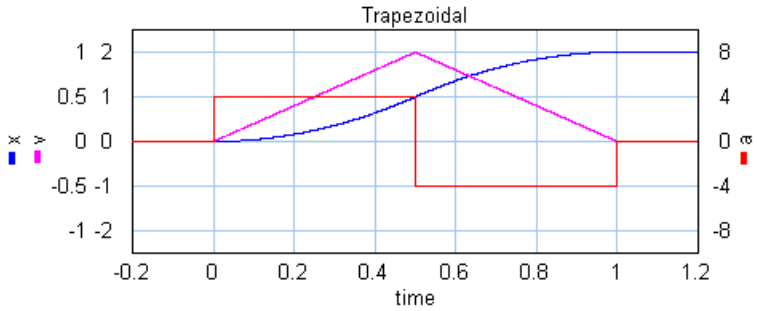
The ramp profile is a first order profile. It has a constant velocity and acceleration peaks at the start and end of the motion.



The ramp profile.

Trapezoidal

The trapezoidal profile is a second order profile. It has a constant acceleration at the start of the motion and a constant deceleration at the end of the motion.

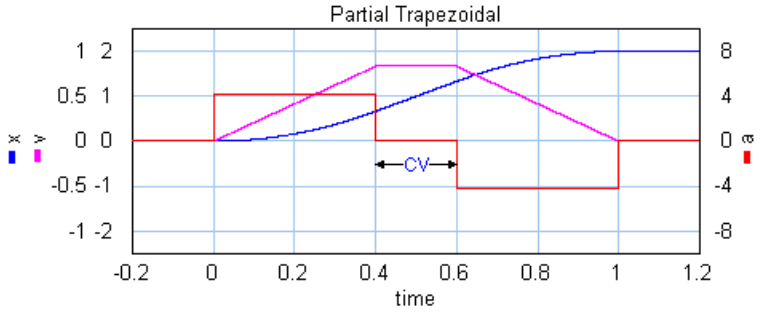


The trapezoidal profile.

This profile is most widely used in early servo systems. Because of the discontinuity in the acceleration this motion profile can still induce a lot of vibrations. Therefore in modern servo systems third order profiles are preferred.

Partial Trapezoidal

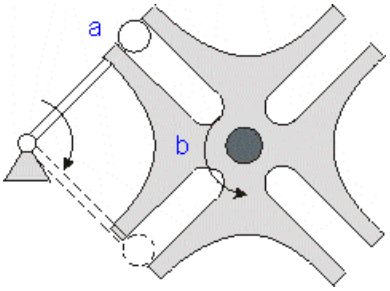
The partial trapezoidal profile is a second order profile. It has is equal to the trapezoidal motion, but has a constant velocity part, during a fraction **CV** (%) of the motion.



The partial trapezoidal profile.

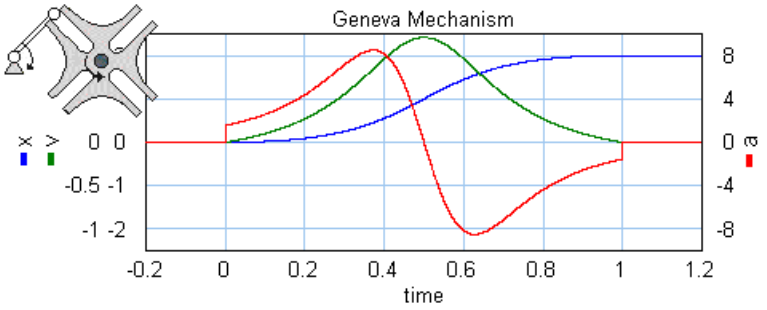
Geneva Mechanism

A Geneva mechanism is an old motion profile generation mechanism. A continuously rotating crankshaft (a) generates an intermittent motion of a second shaft (b). The result is a second order profile.



A Geneva mechanism.

The motion profile generator yields the motion of the second shaft (b). It is shown below:

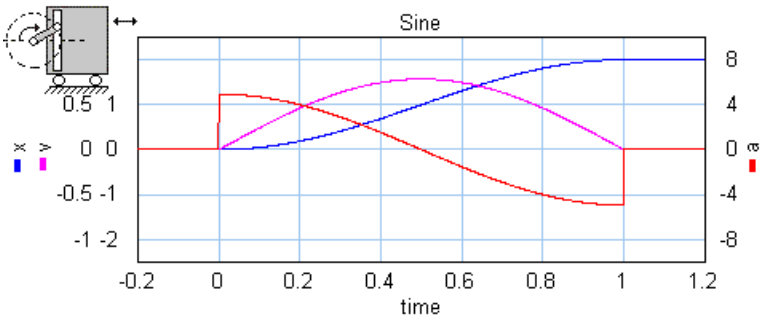


The Geneva mechanism profile.

Sine

The sine profile is a second order profile of which the displacement can be described as:

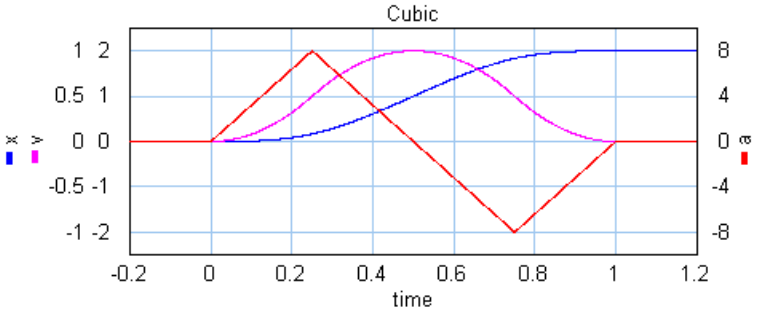
$$x = 0.5 - \cos(p * t) / 2.$$



The sine profile.

Cubic

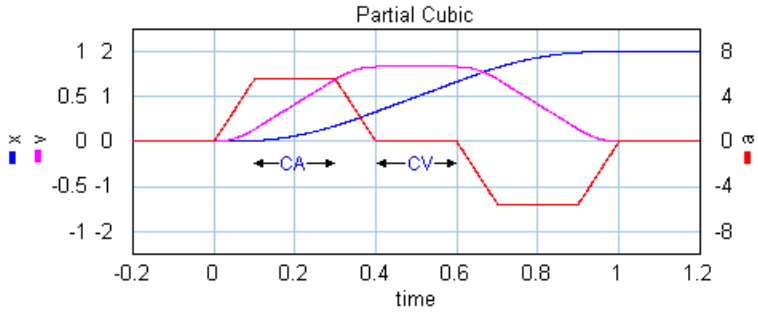
The cubic profile is a third order profile of which the acceleration is constantly increasing and decreasing.



The cubic profile.

Partial Cubic

The partial cubic profile (3rd order) is a modified cubic profile with a constant velocity during a fraction **CV** (%) of the motion and a constant acceleration during a fraction **CA** (%) of the motion.

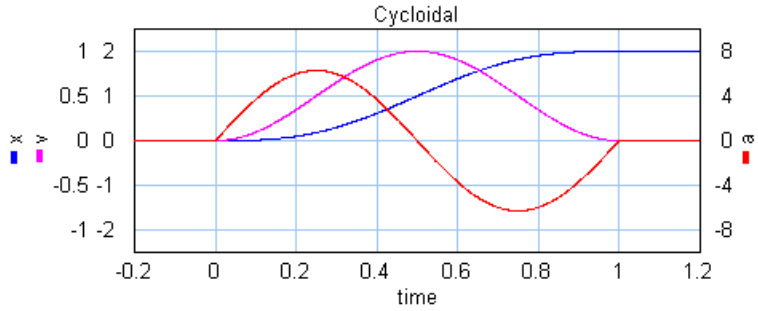


The partial cubic profile.

Cycloidal

The cycloidal profile is a third order profile, of which the velocity can be described as:

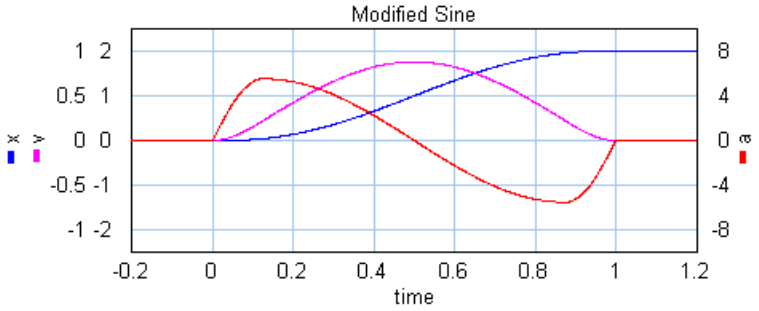
$$v = stroke*(1-cos(t*a))$$



The cycloidal profile.

Modified Sine

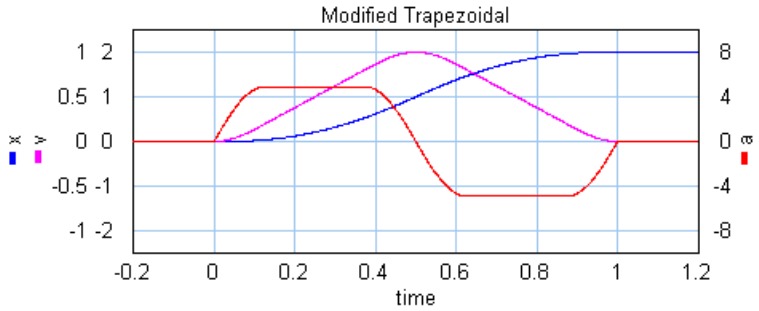
The modified sine profile is also a third order profile. It is a modification of the cycloidal profile to get a lower maximum velocity and a lower maximum acceleration.



The modified sine profile.

Modified Trapezoidal

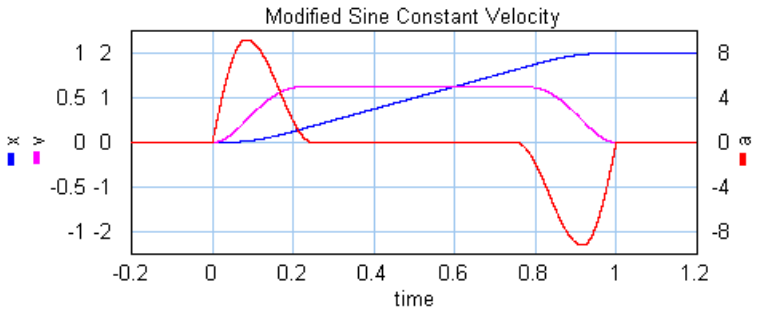
The modified trapezoidal profile is a modification of the trapezoidal profile (to make it a third order profile). This profile yields a very low maximum acceleration.



The modified trapezoidal profile.

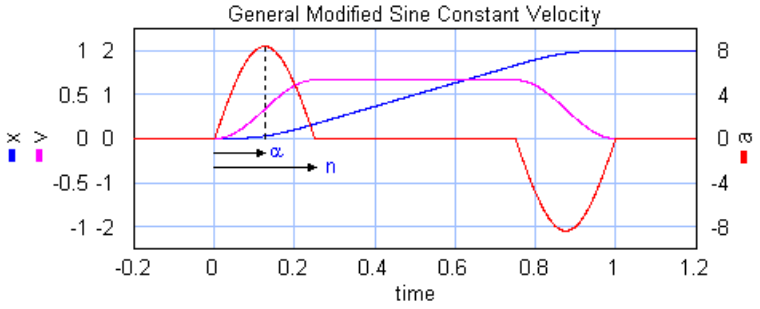
Modified Sine with Constant Velocity (MSC50)

The modified sine with constant velocity profile (3rd order) is a modification of the modified sine profile. It has a constant velocity during 50% of the motion.



The modified sine with constant velocity profile.

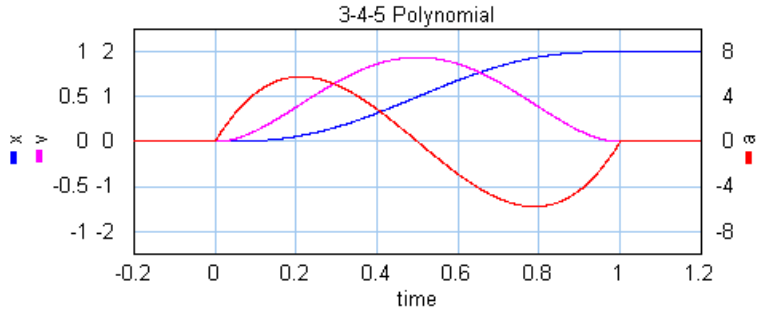
General Modified Sine with Constant Velocity (MSC%)
The modified sine with constant velocity profile (3rd order) is a modification of the modified sine profile. It has a constant velocity during a user definable part of the of the motion. The non-zero acceleration part is defined by two parameters which are both defined as a fraction of the motion. The first parameters **alpha** (%) defines the start of the acceleration and the second parameters **n** (%) defines the end of the acceleration.



The general modified sine with constant velocity profile.

3-4-5 Polynomial

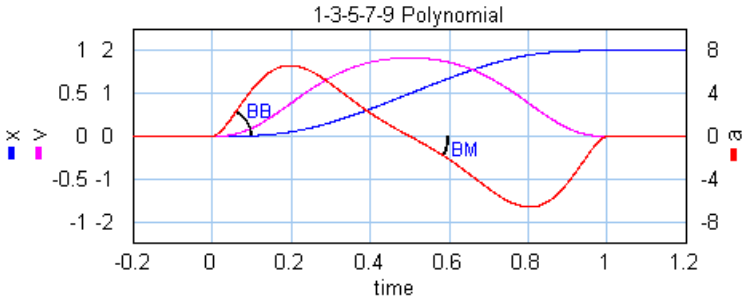
The is a third profile described by a 5th order polynomial.



The 1-3-5 polynomial profile.

1-3-5-7-9 Polynomial

The is a third/fourth order profile described by a 9th order polynomial. The profile is characterized by two parameters that denote the initial jerk, **BB,** and the crossover jerk, **BM**. If BB is chosen zero, this a a fourth order profile. If BB is chosen non-zero, this is a third order profile.



The 1-3-5-7-9 polynomial profile.

To keep the polynomial consistent, i.e. the polynomial does not change when the amplitude is changed, the initial jerk (**BB**) and the crossover jerk (**BM**) are defined for a standard motion with:

- *stroke = 1*
- *start_time = 0*
- *stop_time = 1*

Comparison of Profiles

The specific shape of a motion profile can have a significant influence on the dynamic behavior. Some profiles minimize the maximum velocity, some profile minimize acceleration, while other profiles tend to make a tradeoff between the maximum velocity and acceleration. If we take a standard motion with stroke 1 and motion time 1 sec., the following table can be generated:

profile	order	vmax	amax	j(0)	j(1/2)
Flat	0	0	0	0	0
Ramp	1	1	infinite	infinite	0
Trapezoidal	2	2	4	infinite	-infinite
Partial Trapezoidal1	2	1.67	4.17	infinite	0
Geneva Mechanism	2	2.41	8.49	infinite	-118.5
Sine	2	1.57	4.93	infinite	-15.5
Cubic	3	2	4	32	-32
Partial Cubic3	3	1.67	5.55	55.6	0
Cycloidal	3	2	6.28	39.5	-39.5
Modified Sine	3	1.76	5.53	69.4	-23.2
Modified Trapezoidal	3	2	4.88	61.4	-61.4
MSC50	3	1.26	9.20	173.6	0
MSC%4	3	1.37	7.20	113.1	0
3-4-5 Polynomial	3	1.88	5.77	60	-30

1-3-5-7-9 Polynomial5	3/4	2.05	10.25	BB	BM

1: Parameter CV = 20%

2: For lw >> lc the Crank Rod profile equals the sine profile, for lw > lc performance deteriorates.

3: Parameter CV = 20%, CA = 20%

4: Parameter n = 30%, alpha = 10%

5: Parameter BB = 30%, BM = 10%

Here j(0) is the initial jerk yerk (derivative of acceleration) and j(1/2) the crossover (halftime) jerk.

8.5.3 Servo Motor Editor

Servo Motor Editor

Introduction

The 20-sim Servo Motor Editor is a tool to generate dynamic models of servo motors for the use in 20-sim. These models describe the complete dynamic behaviour of servo motors, including the electrical, mechanical and thermal behaviour. The following classes of motors are supported in the editor:

1. Brush DC
2. Brushless DC (trapezoidal EMC and square wave currents)
3. AC synchronous (sinusoidal EMC and sinusoidal currents)
4. AC synchronous linear (sinusoidal EMC and sinusoidal currents)

The dynamic models are generated automatically from data files containing commercially available motors, but you can also enter your own motor parameters. The following motors are available on data files:

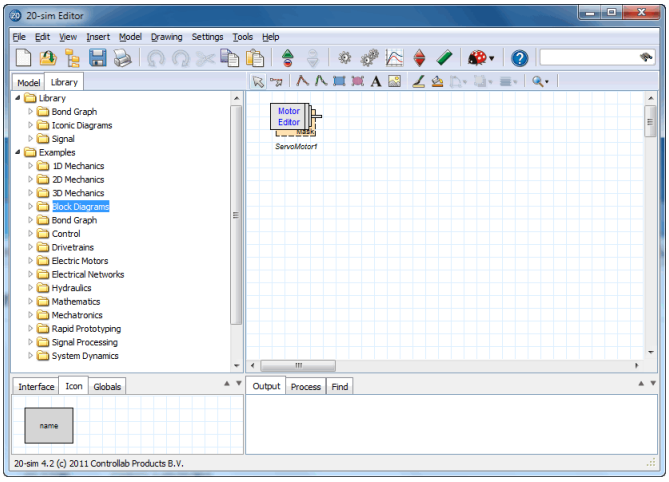
1. The complete Maxon 2005 / 2006 program.
2. The complete Tecnotion 2006 program.
3. The Faulhaber 2006 program.

In the first part of these help files, the use of the 20-sim Servo Motor Editor is explained. The Editor uses a data table with motor parameters of commercial servo motors. You can select any motor from the table, inspect the corresponding torque speed curve and let the Servo Motor Editor generate a 20-sim dynamic model from the parameters.

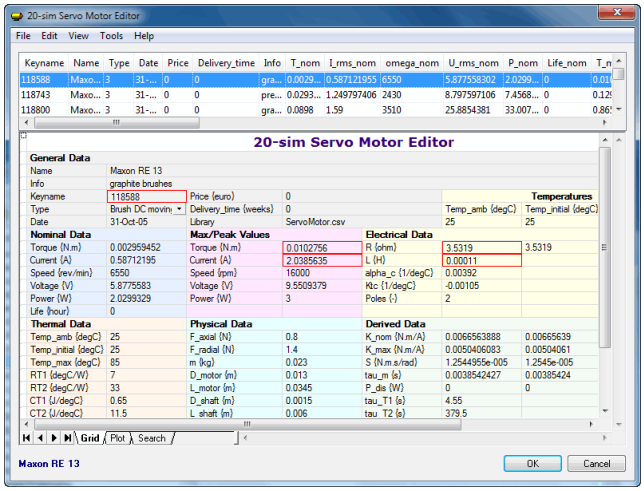
The second part of these help files, explains the operating principles of the motors. A good indication of the performance of permanent magnet motors can be given by the torque speed plot. Various curves of the torque speed plot and how this can be used to choose the proper motor for a given task. An important part of the torque speed plot is the maximum continuous torque. It is the maximum torque that a motor can deliver without overheating. This curve is based on a thermal model of the motor, which is also explained.

Use

The 20-sim Servo Motor Editor is part of the Mechatronics Toolbox of 20-sim. You can open the editor by selecting the **Servo Motor Editor** command (**Tools** - **Mechatronics** Toolbox). If you have a valid license, a servo motor model will be inserted:



Normally the editor will be opened automatically. If this does not happen, force the editor to open by clicking the Go Down button. This Editor looks like:



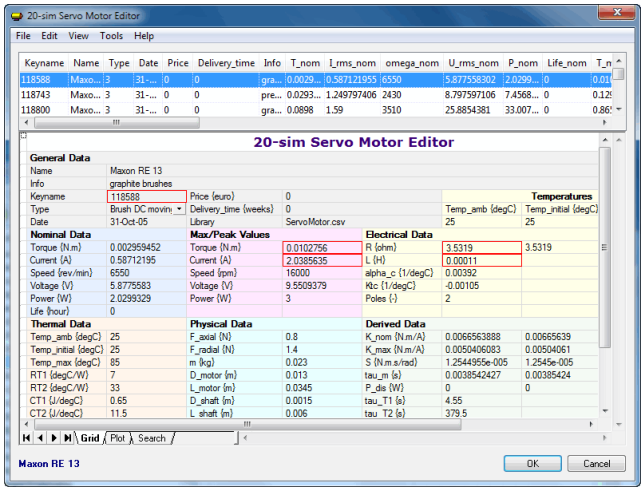
The top of the editor shows a list of motors. You can select one of the motors from the list by selecting it with your mouse pointer. The selected motor is shown with the blue line. The parameters of the selected motor are shown below the list.

To see the torque speed plot of the selected motor, click the *Plot* tab at the bottom of the editor. To find a motor based on specific searching conditions, click the *Search* tab. By clicking the *OK* button, a dynamic model will be created automatically, based on the selected motor in the list.

Using the Edit menu, new motors can be added to the list or deleted. By double clicking the mouse pointer on a parameter value, it can be changed.

Data Files

If you open the Servo motor Editor, default data from file *ServoMotor.csv* is shown. You can open other data files by using the file menu.



Currently the following data files are available:

1. *ServoMotor.csv*: A selection of various motors to show the capabilities of the Servo Motor Editor.
2. *Maxon 2006.cse:* The complete list of Maxon motors, program 2005 / 2006.
3. *Tecnotion 2006.cse*: The complete list of Tecnotion motors, program 2006.
4. *Faulhaber 2006.cse.*: Most of the Faulhaber motors, program 2006.

Parameters

Unfortunately not all motor suppliers use the same parameters in the data sheets. Therefore the Servo Motor Editor uses the most common parameters as described in this section. All parameters are defined using a motor temperature and ambient temperature of 25° C.

General Data

This general section contains the unique identifier of the motor (Keyname) and some additional data, which can be filled in freely.

Name	The name of the motor
Info	Additional Info
Keyname	A unique name (no duplicates allowed in the motor list) to identify the motor.
Type	The motor type: 1. AC Synchronous

2. Brush DC Iron
3. Brush DC moving coil
4. Brush DC Disc Rotor
5. AC Synchronous Linear
6. Brushless DC

In the Editor, you can select from a drop down list. In the data file (see next section, the corresponding numbers are used).

Date	Date at which the motor info was entered
Price	The price of the motor
Delivery Time	Delivery Time
Library	The name of the data file that is used

Nominal Data

Many data sheets describe the nominal operating point of a motor. The nominal operation point is characterized by a fixed current that is supplied to a motor with an initial temperature of 25º C that will result in a heating up of the coils to the exactly the maximum temperature. Because a current alone is not enough to describe the state of a motor, the operating point is always given at a certain speed. Although the other parameters (torque and power) can be derived from the current and speed they are usually printed in most data sheets.

Torque [N.m]or Force[N]	Torque or force at nominal operation.
Current [A] or [Arms]	Current at nominal operation.
Speed [rpm]or [m/s]	Speed at nominal operation.
Voltage [V] or [Vrms]	Voltage at nominal operation.
Power [W]	The output power at nominal operation.
Life [hour]	Expected Lifetime of the motor under normal operation.

As described, the definition of current and voltage depends on the selected motor:

motor	**current**	**voltage**
1. AC Synchronous	rms phase current (A)	rms phase to phase voltage (V)
2. Brush DC Iron	current (A)	voltage (V)
3. Brush DC moving coil	current (A)	voltage (V)
4. Brush DC Disc Rotor	current (A)	voltage (V)
5. AC Synchronous Linear	rms phase current (A)	rms phase to phase voltage (V)

6. Brushless DC	peak current (A)	peak phase to phase voltage (V)

Max/Peak Values

The maximum parameters indicate the maximum performance of the motor and are used to show the limits in the torque speed plot as explained in chapter 3. The parameters do not indicate an operating point on the torque speed curve. The maximum parameters are preferably measured at a temperature of 25º C.

Torque [N.m]or Force[N]	The maximum short-time or torque or force (also known as stall torque).
Current [A] or [Arms]	The maximum short-time current.
Speed [rpm]or [m/s]	The maximum speed.
Voltage [V] or [Vrms]	Maximum short-time voltage.
Power [W]	The maximum short time power.

The currents and voltages are defined the same as for the nominal parameters.

Electrical Data

R [ohm]	The resistance at 25° C.
L [H]	The terminal inductance.
alpha_c [1/degC]	The temperature dependency of the coil resistance.
ktc [1/degC]	The temperature dependency of the magnets.
poles [-] or pitch[m]	The number of poles (always an even number) or pole pitch (distance N-N poles).

As described, the definition of resistance and inductance depends on the selected motor:

motor	**resistance**	**inductance**
1. AC Synchronous	phase to phase resistance (ohm)	phase to phase inductance (H)
2. Brush DC Iron	resistance (ohm)	inductance (H)
3. Brush DC moving coil	resistance (ohm)	inductance (H)
4. Brush DC Disc Rotor	resistance (ohm)	inductance (H)
5. AC Synchronous Linear	phase to phase resistance (ohm)	phase to phase inductance (H)
6. Brushless DC	phase to phase resistance (ohm)	phase to phase inductance (H)

Mechanical Data

The mechanical data describes the motor inertia and losses.

J_rotor [kg.m2] or m_motor [kg]	The rotor inertia or the the moving mass of the motor.
d [N.m.s/rad]or [N.s/m]	Mechanical damping and eddy current losses.
T_fric [N.m] or [N]	Mechanical friction and hysteresis losses.
T_cog [N.m] or [N]	The amplitude of the cogging torque.

Thermal Data

The thermal data is used for the thermal model as explained in chapter 4. Zero values for the thermal resistances are replace by a small value (1e-2) to prevent division by zero.

Temp_amb [degC]	The ambient temperature.
Temp_initial [degC]	The initial temperature of the coils and housing.
Temp_max [degC]	The maximum coil temperature.
RT1 [degC/W]	The thermal resistance between rotor and stator.
RT2 [degC/W]	The thermal resistance between stator and environment.
CT1 [J/degC]	Thermal capacity of the rotor.
CT2 [J/degC]	Thermal capacity of the stator.

Physical Data

This section describes the motor dimensions and maximal load.

F_axial [N]	The maximum axial force applied to the motor shaft.
F_radial [N]	The maximum radial force applied to the motor shaft.
m [kg]	The motor weight.
D_motor [m]	The motor diameter.
L_motor [m]	The motor length.
D_shaft [m]	The shaft diameter.
L_shaft [m]	The shaft length.

Derived Data

These are values that can be found in many data sheets. They are useful for selection criteria when searching for motors.

K_nom [N.m/A], [N.m/Arms] or [N/Arms]	The torque constant or force constant at 25° C and nominal currents.
K_max [N.m/A], [N.m/Arms] or [N/Arms]	The torque constant or force constant at 25° C and at peak currents.
S [N.m.s/rad] or [N.s/m]	The steepness.
tau_m [s]	The mechanical time constant (J_rotor/S or m_motor/S).
P_dis [W]	The maximal continuous dissipation.
tau_T1 [s]	The thermal time constant of the rotor (RT1* CT1).
tau_T2 [s]	The thermal time constant of the stator (RT2* CT2).

Additional Data

20-sim model []	Not implemented yet.

How to use the Parameters

When you enter a new motor, not all parameters have to be filled in. In this section the critical and non critical parameters are listed.

Critical

Only a few parameters are critical. I.e. they are necessary to generate a 20-sim dynamic model.

Keyname	A unique name to identify the motor.
Type	The motor type.
Maximum Torque or Force	The maximum short-time or torque or force.
Maximum Current	The maximum short-time current.
R	The resistance at 25° C.
L	The terminal inductance.
J_rotor or m_motor	The rotor inertia or the the moving mass of the motor.

Desired

Some parameters are desired. I.e. they add more detail to the model:

Nominal Torque or Force	Torque or force at nominal operation.
Nominal Current	Current at nominal operation.
Nominal Speed	Speed at nominal operation.
alpha_c	The temperature dependency of the coil resistance.
ktc	The temperature dependency of the magnets.
poles or pitch	The number of poles (always an even number) or pole pitch (distance N-N poles).
d	Mechanical damping and eddy current losses.
T_fric	Mechanical friction and hysteresis losses.
T_cog	The amplitude of the cogging torque.
Temp_amb	The ambient temperature.
Temp_initial	The initial temperature of the coils and housing.
Temp_max	The maximum coil temperature.
RT1	The thermal resistance between rotor and stator.
RT2	The thermal resistance between stator and environment.
CT1	Thermal capacity of the rotor.
CT2	Thermal capacity of the stator.

If you do not know the values of these parameters, fill in zero. 20-sim will automatically change some into small numbers to prevent division by zero. Because the nominal torque and current are used to derive the torque constant at nominal operation, they will be replaced by the maximum current and torque if you fill in a zero value.

Torque Speed Plot

Some parameters will add more detail to the torque speed plot:

Speed	The maximum speed.
Voltage	Maximum short-time voltage.
Power	The maximum short time power.

Additional

The remaining parameters are not used in the torque speed plot or the 20-sim model. They are useful for selection criteria when searching motors. If you do not know the values of these parameters, fill in zero.

Dynamic Model

If you have selected a motor, clicking the OK button will close the editor and generate a dynamic model. The model will be filled with the parameters from the selected motor and is ready for the use in a simulation. The model contains a number of variables that may be useful to show in a simulation plot.

General Part

R	resistance at simulated (fluctuating) temperature
K	torque constant at simulated (fluctuating) temperature
u_rms_ff	effective terminal (phase-phase) voltage
u_tt_ff	maximum terminal (phase-phase)
i_tt	maximum phase current
phi	shaft angle

Thermal part

Temp_coil	coil temperature
Temp_housing	housing temperature

Torque speed plot

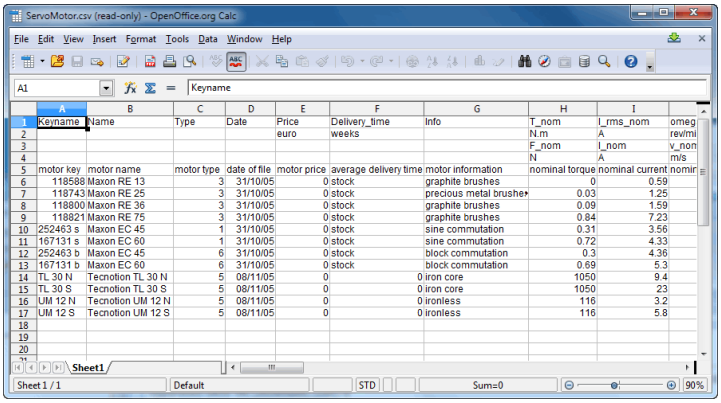
omega_range	speed ranging from zero to over 5% of maximum speed
T_range	torque ranging from zero to over 5% of maximum torque
Torquemax	maximum torque
Speedmax	maximum speed
T_max_power	torque at maximum power
T_max_current	torque at maximum current
T_max_outputpower	line of maximum output torque
T_max_voltage	torque at maximum voltage
T_max_efficiency	line of torque at maximum efficiency
T_100	maximum allowable continuous torque, (i.e. a 100% duty cycle)

T_50	maximum allowable torque for a 50% duty cycle
T_25	maximum allowable torque for a 25% duty cycle
T_10	maximum allowable torque for a 10% duty cycle
T_abs	absolute output (load) torque
omega_abs	absolute velocity

Creating your own data files

The motor parameters are visible in the Grid tab. These parameters are stored in a coma separated data file (extension .csv) or in an encrypted coma separated data file (extension .cse).

By default, the file `ServoMotor.csv` is shown. Using the File menu you can store and open csv files. The csv files can be edited with the *Servo Motor Editor* or an external spreadsheet program like OpenOffice. In *OpenOffice Calc* the file looks like:



The top 5 rows contain the description of the motor parameters and the corresponding units. The first and second row are used for rotation motors and the third and fourth row are used for linear (translation) motors. The fifth row contains the parameter names.

The other rows contain the motor parameters. The meaning of these parameters is described in the previous sections. Please use the following guidelines when editing data files:

1. Never change the first five lines
2. Do no use duplicate names in the first column (keyname)
3. Do use commas

4. Do no use strange formatting (in Excel use the General format)
5. Do no enter extra columns
6. Do net enter other information in additional rows or columns. The Servo Motor Editor will try to read all cells and halt if the content is not according the specifications.
7. Do no add comment.
8. The last column entry (20-sim) should contain one white space character.
9. Store as comma separated file.

Data Files

Maxon 2005 / 2006

File

20-sim 3.6\Tools\Servo Motor Dynamics\Maxon 2006.cse

Date

November 2005

More information

www.maxonmotors.com

Description

The data file Maxon 2006.cse contains the complete set of Maxon motors, program 2005 / 2006. Most of the data has been extracted from the Maxon data sheets. The tables below shows the 20-sim names and the corresponding data sheet names. Some parameters were not available in the data sheets and have been given by the Maxon motor company. These parameters are indicated as *Given by Maxon*. Some parameters were calculated. These are indicated as *Calculated.*

DC motors

Servo Motor Editor	Maxon Data Sheets
General Data	
Name	Motor Series
Keyname	Maxon Order Number
Info	Additional Information
Type	DC
Date	21 October 2005
Price	-
Delivery Time	stock / standard / special
Library	Maxon 2006.cse
Nominal Data	
Torque	Max. continuous torque
Current	Max. continuous current
Speed	No load speed divided by 2
Voltage	Calculated

Power	Calculated
Max/Peak Data	
Torque	Stall torque
Current	Starting current
Power	Assigned Power Rating
Voltage	Calculated*
Speed	Max. permissible speed
Thermal Data	
Temp_amb	Set to 25
Temp_initial	Set to 25
Temp_max	Maximum rotor temperature
RT1	Thermal resistance rotor-housing
RT2	Thermal resistance housing-ambient
CT1	Calculated
CT2	Given by Maxon
Physical Data	
F_axial	Maximum Axial load
F_Radial	Maximum radial load
m	Weight of motor
D_motor	Diameter of motor
L_motor	Length of motor
D_shaft	Diameter shaft
L_shaft	Length Shaft
Electrical Data	
R	Terminal resistance
L	Terminal inductance
alpha_c	Given by Maxon
Ktc	Given by Maxon
Poles	Number of poles
Mechanical Data	
J_rotor	Rotor inertia
d	Given by Maxon
T_fric	Given by Maxon
T_cog	0
Derived Data	
K_nom	Calculated
K_max	Calculated
S_nom	Calculated
P_dis	Calculated

* In the data sheets Maxon only gives a nominal voltage. According to Maxon higher operating voltages are permissible provided that other limits are not exceeded. Therefore the maximum voltage is chosen equal to the voltage that is needed to run a motor with zero load at maximum speed.

EC motors with block commutation

Maxon EC motors can be driven by block commutation and sine commutation. The Maxon data sheets give the parameters for block commutation. To distinguish between block commutation and sine commutation the character **b** or **s** is added to the keyname.

Servo Motor Editor	**Maxon Data Sheets**
General Data	
Name	Motor Series
Keyname	Maxon Order Number + b
Info	Additional Information
Type	Brushless DC
Date	21 October 2005
Price	-
Delivery Time	stock / standard / special
Library	Maxon 2006.csv
Nominal Data	
Torque	Calculated
Current	Max. continuous current
Speed	Nominal speed
Voltage	Calculated
Power	Calculated
Max/Peak Data	
Torque	Stall torque
Current	Calculated
Power	Assigned Power Rating
Voltage	Calculated*
Speed	Max. permissible speed
Thermal Data	
Temp_amb	Set to 25
Temp_initial	Set to 25
Temp_max	Maximum rotor temperature
RT1	Thermal resistance rotor-housing
RT2	Thermal resistance housing-ambient
CT1	Calculated
CT2	Calculated
Physical Data	
F_axial	Maximum Axial load
F_Radial	Maximum radial load
m	Weight of motor
D_motor	Diameter of motor
L_motor	Length of motor
D_shaft	Diameter shaft
L_shaft	Length Shaft
Electrical Data	
R	Terminal resistance phase to phase
L	Terminal inductance phase to phase
alpha_c	Given by Maxon
Ktc	Given by Maxon
Poles	Number of poles
Mechanical Data	
J_rotor	Rotor inertia
d	Given by Maxon
T_fric	Given by Maxon
T_cog	0
Derived Data	
K_nom	Calculated

K_max	Calculated
S_nom	Calculated
P_dis	Calculated

* In the data sheets Maxon only gives a nominal voltage. According to Maxon higher operating voltages are permissible provided that other limits are not exceeded. Therefore the maximum voltage is chosen equal to the voltage that is needed to run a motor with zero load at maximum speed.

EC motors with sine commutation

Maxon EC motors can be driven by block commutation and sine commutation. The Maxon data sheets give the parameters for block commutation. The parameters for sine commutation can be calculated from the block commutation parameters. To distinguish between block commutation and sine commutation the character **b** or **s** is added to the keyname.

Servo Motor Editor	**Maxon Data Sheets**
General Data	
Name	Motor Series
Keyname	Maxon Order Number + s
Info	Additional Information
Type	AC Synchronous
Date	21 October 2005
Price	-
Delivery Time	stock / standard / special
Library	Maxon 2006.csv
Nominal Data	
Torque	Calculated
Current	Calculated
Speed	Nominal speed
Voltage	Calculated
Power	Calculated
Max/Peak Data	
Torque	Calculated
Current	Calculated
Power	Assigned Power Rating
Voltage	Calculated*
Speed	Calculated
Thermal Data	
Temp_amb	Set to 25
Temp_initial	Set to 25
Temp_max	Maximum rotor temperature
RT1	Thermal resistance rotor-housing
RT2	Thermal resistance housing-ambient
CT1	Calculated
CT2	Calculated
Physical Data	
F_axial	Maximum Axial load
F_Radial	Maximum radial load
m	Weight of motor
D_motor	Diameter of motor

L_motor	Length of motor
D_shaft	Diameter shaft
L_shaft	Length Shaft
Electrical Data	
R	Terminal resistance phase to phase
L	Terminal inductance phase to phase
alpha_c	Given by Maxon
Ktc	Given by Maxon
Poles	Number of poles
Mechanical Data	
J_rotor	Rotor inertia
d	Given by Maxon
T_fric	Given by Maxon
T_cog	0
Derived Data	
K_nom	Calculated
K_max	Calculated
S_nom	Calculated
P_dis	Calculated

* In the data sheets Maxon only gives a nominal voltage. According to Maxon higher operating voltages are permissible provided that other limits are not exceeded. Therefore the maximum voltage is chosen equal to the voltage that is needed to run a motor with zero load at maximum speed.

Tecnotion 2006

File

20-sim 3.6\Tools\Servo Motor Dynamics\Tecnotion 2006.cse

Date

November 2006

More information

www.tecnotion.com

Description

The data file Tecnotion 2006.cse contains the complete set of Tecnotion motors, program 2006. Most of the data has been extracted from the Tecnotion data sheets. The tables below shows the 20-sim names and the corresponding data sheet names. Some parameters were not available in the data sheets and have been given by the Tecnotion motor company. These parameters are indicated as *Given by Tecnotion*. Some parameters were calculated. These are indicated as *Calculated*.

Important: The nominal data and the default value of the thermal resistance RT2 are calculated for **air cooling**. The value for RT2 can be changed using a slider. **If water cooling is used, change the value of RT2 to its minimum** value.

Servo Motor Editor	**Tecnotion Data Sheets**
General Data	

Name	Motor Series
Keyname	Order Number
Info	Additional Information
Type	AC Synchronous Linear
Date	21 October 2005
Price	-
Delivery Time	-
Library	Tecnotion 2006.cse

Nominal Data

Force	Continuous Force (air cooled*)
Current	Continuous Current (air cooled*)
Speed	0 m/s
Voltage	Calculated
Power	-

Max/Peak Data

Force	Ultimate Force / Peak Force
Current	Ultimate Current / Peak Current
Power	Calculated
Voltage	Max. voltage ph-ph
Speed	Maximum speed

Thermal Data

Temp_amb	Set to 25
Temp_initial	Set to 25
Temp_max	Maximum Temperature
RT1	Thermal Resistance coil - housing
RT2	Calculated*
CT1	Given by Tecnotion
CT2	Given by Tecnotion

Physical Data

F_axial	-
F_Radial	-
m	-
D_motor	-
L_motor	-
D_shaft	-
L_shaft	-
	-

Electrical Data

R	2 * Resistance per phase
L	2 * Induction per phase
alpha_c	0.004
Ktc	0
Pitch	Magnet Pitch NN

Mechanical Data

m_motor	Weight of Coilunit
d	Given by Tecnotion
F_fric	-
F_cog	0

Derived Data

K_nom	Calculated
K_max	Calculated
S_nom	Calculated
P_dis	Calculated

* Note: The nominal data and the default value of the thermal resistance RT2 are calculated for **air cooling**. The value for RT2 can be changed using a slider. **If water cooling is used, change the value of RT2 to its minimum** value.

Faulhaber 2006

File

20-sim 3.6\Tools\Servo Motor Dynamics\Faulhaber 2006.cse

Date

February 2006

More information

www.faulhaber-group.com or www.minimotor.ch

Description

The data file Faulhaber 2006.cse contains the most of the Faulhaber servo motors, program 2006. Most of the data has been extracted from the Faulhaber data sheets. The tables below shows the 20-sim names and the corresponding data sheet names. Some parameters were not available in the data sheets and have been given by Faulhaber. These parameters are indicated as *Given by Faulhaber*. Some parameters were calculated. These are indicated as *Calculated*.

DC motors

Servo Motor Editor	**Faulhaber Data Sheets**
General Data	
Name	Motor Name
Keyname	Faulhaber Order Number
Info	Additional Information
Type	DC
Date	23 February 2006
Price	-
Delivery Time	-
Library	Faulhaber 2006.cse
Nominal Data	
Torque	Calculated
Current	Current up to
Speed	No load speed divided by 2
Voltage	Calculated
Power	Calculated
Max/Peak Data	
Torque	Stall torque
Current	Calculated

Power	Output Power
Voltage	Nominal Voltage
Speed	No-load speed
Thermal Data	
Temp_amb	Set to 25
Temp_initial	Set to 25
Temp_max	Maximum coil temperature
RT1	Thermal resistance rotor – housing Rth1
RT2	Thermal resistance housing-ambient Rth1
CT1	Calculated
CT2	Calculated
Physical Data	
F_axial	Maximum Axial load
F_Radial	Maximum radial load
m	Weight of motor
D_motor	Diameter of motor
L_motor	Length of motor
D_shaft	Diameter shaft
L_shaft	Length Shaft
Electrical Data	
R	Terminal resistance
L	Rotor Inductance
alpha_c	Temperature coefficient of resistance of copper
Ktc	Given by Faulhaber
Poles	Given by Faulhaber
Mechanical Data	
J_rotor	Rotor inertia
d	Friction torque, dynamic
T_fric	Friction torque, static
T_cog	0
Derived Data	
K_nom	Calculated
K_max	Calculated
S_nom	Calculated
P_dis	Calculated

DC Brushless motors with block commutation

Servo Motor Editor	**Faulhaber Data Sheets**
General Data	
Name	Motor Name
Keyname	Faulhaber Order Number
Info	Additional Information
Type	DC Brushless
Date	23 February 2006
Price	-
Delivery Time	-
Library	Faulhaber 2006.cse

Nominal Data

Torque	Torque up to
Current	Current up to
Speed	At speed
Voltage	Calculated
Power	Calculated

Max/Peak Data

Torque	Stall torque
Current	Calculated
Power	Output Power
Voltage	Nominal Voltage
Speed	No-load speed

Thermal Data

Temp_amb	Set to 25
Temp_initial	Set to 25
Temp_max	Maximum coil temperature
RT1	Thermal resistance rotor – housing Rth1
RT2	Thermal resistance housing-ambient Rth1
CT1	Calculated
CT2	Calculated

Physical Data

F_axial	Maximum Axial load
F_Radial	Maximum radial load
m	Weight of motor
D_motor	Diameter of motor
L_motor	Length of motor
D_shaft	Diameter shaft
L_shaft	Length Shaft

Electrical Data

R	Terminal resistance
L	Rotor Inductance
alpha_c	Temperature coefficient of resistance of copper
Ktc	Given by Faulhaber
Poles	Given by Faulhaber

Mechanical Data

J_rotor	Rotor inertia
d	Friction torque, dynamic
T_fric	Friction torque, static
T_cog	0

Derived Data

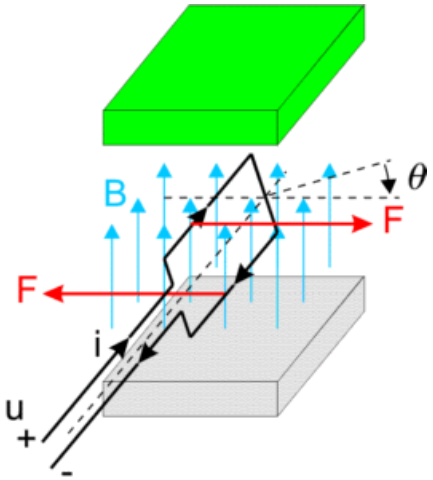
K_nom	Calculated
K_max	Calculated
S_nom	Calculated
P_dis	Calculated

Theory

Basic Principles

Permanent Magnet Motors

When permanent magnets are used in a motor, a magnetic field will be present with a magnetic flux density B. Suppose an electric coil is placed in this magnetic field. When an electric current i is forced to flow through this coil, a force acting on the coil will occur.



The resulting torque can be found as:

$$T = K_{tc}(\theta) \cdot i$$

where *Ktc(q)* denotes the transfer of current to torque. It is a function of the angle, shape of the coil, magnetic field density, current distribution etc. When the current is kept equal to zero and we start to rotate the coil, a voltage will be induced. This voltage is called the electromotive force (EMF) and can be found as:

$$EMF = K_{ec}(\theta) \cdot \omega$$

where *Kec(q)* denotes the transfer of speed to voltage. It is also function of the angle, shape of the coil, magnetic field density, current distribution etc. For an ideal coil the input power should be equal to the output power

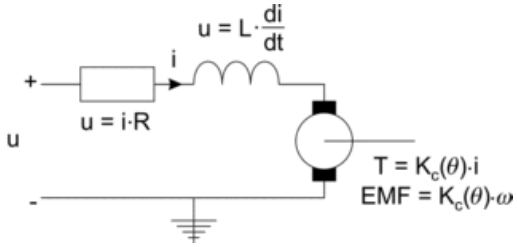
$$P_{in} = EMF \cdot i = K_{ec}(\theta) \cdot \omega \cdot \frac{T}{K_{tc}(\theta)} = \omega \cdot T = P_{out}$$

and thus

$$K_{tc}(\theta) = K_{ec}(\theta) = K_c(\theta)$$

In most literature the *Kc(q)* is calculated out of the magnetic field and coil distribution. Here it is simply assumed to be given and is called the **torque function**.

In normal operation a coil will have a resistance and inductance. A basic coil model is thus equal to:



By a proper design of the motor, i.e. geometry of the coil windings, magnets etc., the torque function *Kc(q)* can be given a particular shape. Then by proper manipulation of the current a positive torque can be created during the whole rotation of the coil. All permanent magnet motors are based on this principle.

Brush DC Motors

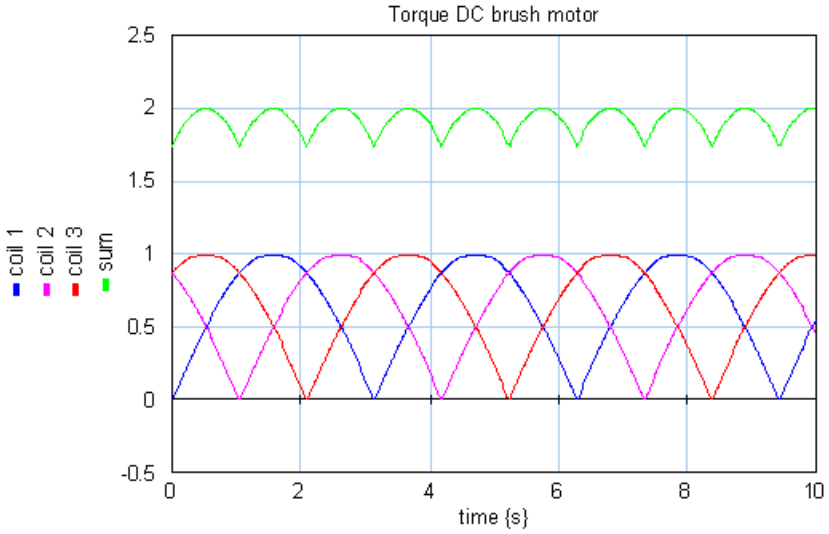
In DC motors, the torque function *Kc(q)* is sinusoidal:

$$K_c(\theta) = \hat{K}_c \cdot sin(\theta)$$

In DC motors a constant current is supplied. Through a process called commutation, the coil is connected the other way around (i.e the current changes sign) when the coil function crosses zero. The resulting torque will then always be positive but will vary between zero and maximum:

$$T = i(\theta) \cdot \hat{K}_c \cdot sin(\theta) = \hat{i} \cdot \hat{K}_c \cdot abs(sin(\theta))$$

By combining coils, the resulting torque variation or *ripple* will decrease. The graph below shows the resulting torque when three coils are used.



The more coils are used the more the torque ripple decreases.

number of coils	torque ripple (top-top)
3	14%
5	5%
7	2.5%
9	1.5%
11	1%

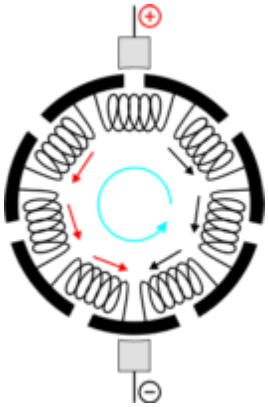
If we neglect the torque ripple, we get the common DC motor equations:

$$T = K \cdot i$$

$$EMF = K \cdot \omega$$

Commutation

In practical motors, the rotor is equipped with a commutator. A commutator consists of insulated collector bars that are connected with the coils. Each coil is connected with one end to a collector and the next end to the neighboring collector. A pair of brushes is used to connect an outside current source to the collectors.



Every rotation the current changes sign twice, resulting in a continuous positive torque addition of every coil.

Limits

Commutation is the largest limiting factor in DC motor performance. Sparks between the brushes and the collectors are the main causes for brush wear. There are several causes for sparking, which limit the maximum speed, current, voltage and power of a brush DC motor.

When the brush leaves a collector, the current has to be reversed. The time that is needed to reverse the current depends directly on the current amplitude and thus on the generated torque. Because the time that is available depends on the motor speed, the result is a limit on the power that can be generated.

Another limitation of the motor speed is due to imperfect dimensions of the collectors. Differences in height of a few microns may cause the brushes to jump at high speeds.

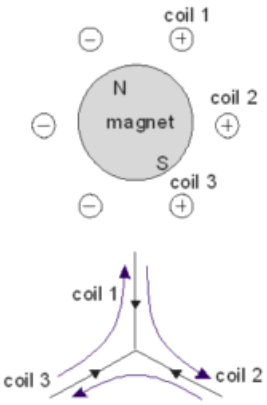
Just a small part of the contact area of a brush and collector is actually used for the current transfer. As a result highly localized, short time "hot spots" develop. When the currents get greater than permitted, local temperatures occur that lead to material evaporation.

Above a voltage of 15 to 18V experience shows that the air will experience ionization phenomena, which leads to arcing. This arc extends from one collector bar to another. When enough voltage is applied, the arc will extend several collector bars and reach the other brush, leading to serious damage. The maximum motor voltage is therefore the result of multiplying the number of collector bars between the brushes and the maximum collector bar voltage.

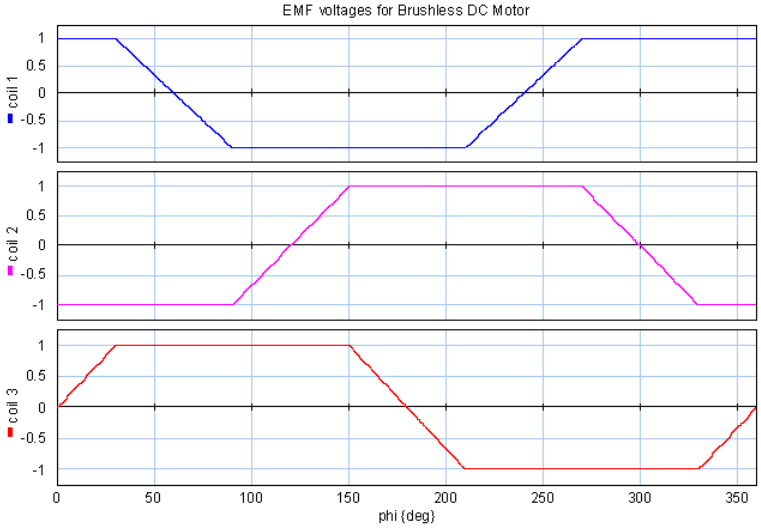
Permanent magnets can lose their strength when operating above temperatures of 60 ºC and suppressed by a high external magnetic field. Such a field can occur when large current changes occur. Therefore the maximum current of a motor should be limited.

Brushless DC Motors

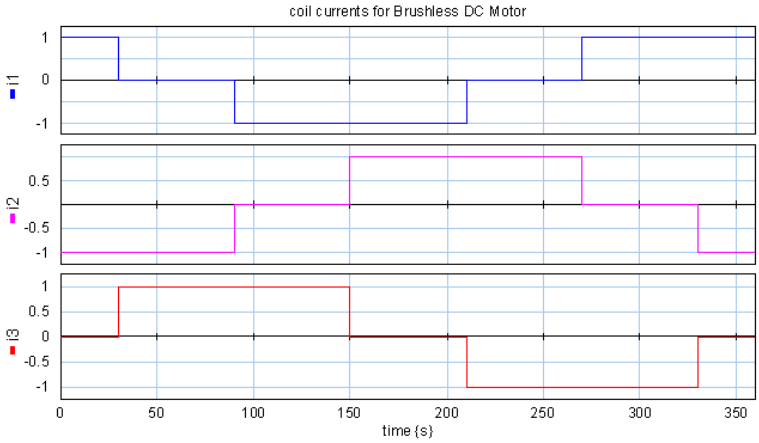
Given motor with three coils, where the coils are mounted in the stator with a spatial displacement of 120°. The coils are connected in a star-formation as shown in the figure below.



In Brushless DC motors, the coils and magnets are designed so that a trapezoidal torque function *Kc(q)* is found as shown below:



To get a constant torque each coil current is a block wave as shown in the figure below. Although the current is thus alternating, the term DC is used to distinguish this type of motor from motors that used a sinusoidal current. The block wave current is provided through "electronic commutation". The current is generated by a three phase amplifier that gets the exact switching points from a motor controller.



As can easily be seen from the graphs, the resulting torque is constant and twice the current times the amplitude of the coil function:

$$T = 2 \cdot \hat{i}_c \cdot \hat{K}_c$$

To get an equivalent of the coil model we can write this equation as:

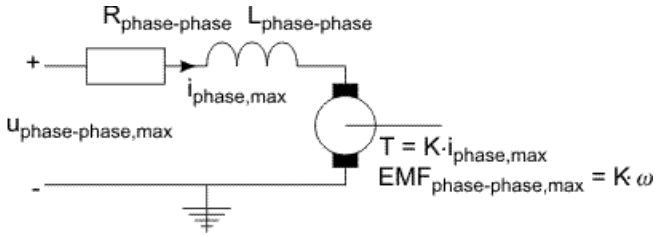
$$T = K \cdot i_{phase,max}$$

We have to realize that here i is not a constant current anymore but the maximum coil current! Because the coil current is the current of one of the three phases, in literature i is mostly written as the **maximum phase current**. It is the maximum current that can be measured at each of the three terminals of the motor.

In a derivation that goes beyond the scope of this introduction (see Compter, 2004), it can be shown that the electromotive force can be found as:

$$EMF_{phase\text{-}phase,max} = K \cdot \omega$$

Here the electromotive force is defined as the maximum induced voltage between two phases. The resulting motor model is shown below:



The voltage of the motor model is commonly known as the **maximum phase to phase voltage**. It is the maximum voltage that can be measured between any pair of two terminals. The **phase to phase resistance** and **phase to phase inductance** are defined equivalent as the resistance and inductance that can be measured between any pair of two terminals.

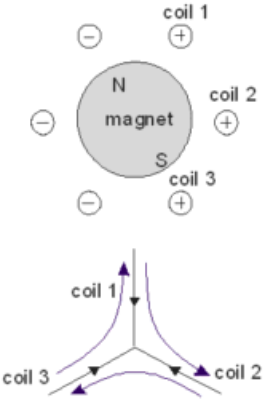
Limits

Brushless DC motors have no mechanical commutation, and therefore do not suffer from the limits caused by mechanical commutation. Consequently Brushless DC motors can run at much higher speeds and are not limited by a maximum voltage. The speed of Brushless DC motors is only limited by the bearings.

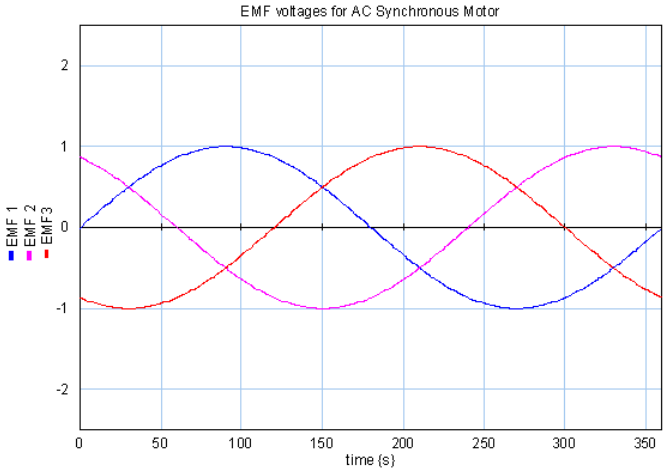
Just like brush DC motors the magnets impose an upper limit for the current and torque in a Brushless DC motor.

AC synchronous motor

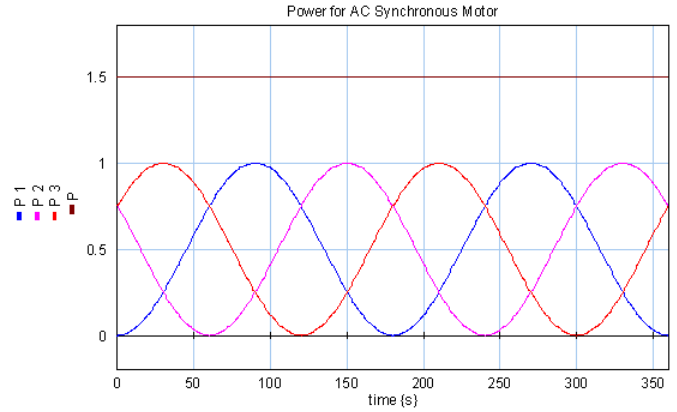
AC synchronous motors use the same principle as Brushless DC-motors. The coils are also mounted in the stator with a spatial displacement of 120°.



The coils and magnets are designed so that a sinusoidal torque function *Kc(q)* is found as shown below:



To get a constant torque each coil current is a sinusoidal wave as shown in the figure below.



As can be seen from the graph, the resulting torque is constant and one and a half times the current times the amplitude of the coil function:

$$T_a = \frac{3}{2} \cdot \hat{K}_c \cdot \hat{i}_c$$

To get an equivalent of the coil model we can write this equation as:

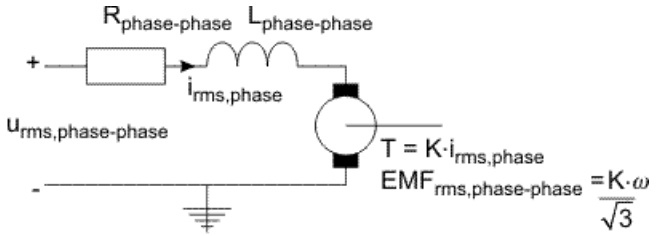
$$T = K \cdot i_{rms,phase}$$

We have to realize that here i is not a constant current anymore but the root mean square coil current! Because the coil current is the current of one phase in literature i is mostly denoted as the **rms phase current.** It is the rms current that can be measured at each of the three terminals of the motor.

In a derivation that goes beyond the scope of this introduction (see Compter, 2004), it can be shown that the electromotive force can be found as:

$$EMF_{rms,phase-phase} = \frac{K}{\sqrt{3}} \cdot \omega$$

Here the electromotive force is defined as the induced rms voltage between two phases. The resulting motor model is shown below:



The voltage of the motor model is mostly denoted as the **rms phase to phase voltage**. It is the rms voltage that can be measured between any pair of two terminals. The **phase to phase resistance** and **phase to phase inductance** are defined equivalent as the resistance and inductance that could be measured between any pair of two terminals.

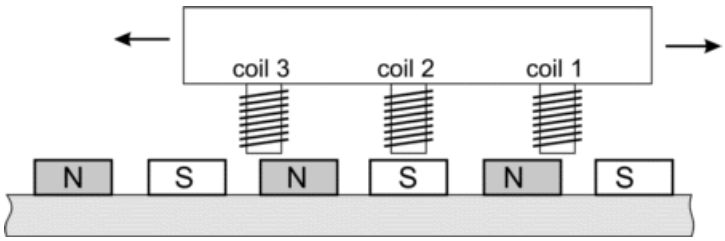
Limits

AC synchronous motors have no mechanical commutation, and therefore do not suffer from the limits caused by mechanical commutation. Consequently AC synchronous motors can run at much higher speeds and are not limited by a maximum voltage. The speed of AC synchronous motors is only limited by the bearings.

Just like brush DC motors the magnets impose an upper limit for the current and torque in a AC synchronous motor.

Linear Motors

The working principle of linear motors is exactly the same as for rotary motors. Magnets are place on a flat surface, as shown in the figure below. A carriage holding the coils can run over the magnets.



When the carriage runs over the magnets an electromotive voltage will be induced:

$$u = K_c(x) \cdot v$$

with *Kc(x)* the **force function** of the coil. It is a function of the carriage position, shape of the coil, magnetic field density, current distribution etc. The force function in a linear motor is equivalent to the torque function of a rotary motor. When a current runs through the coil a force results:

$$F = K_c(x) \cdot i$$

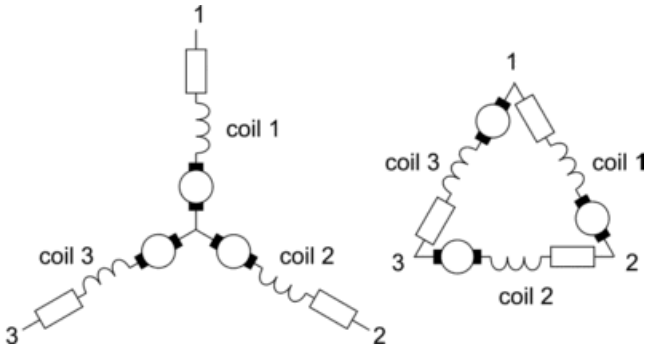
The principle for linear motor is the same as the principle of rotary motors. By a proper design of the motor, i.e. geometry of the coil windings, magnets etc., the force function Kc(x) can be given a particular shape. By proper manipulation of the current a positive force can be created during the movement of the carriage. Because mechanical commutation is not very useful for linear movements, linear motors use electronic commutation. Like their rotary counterparts, linear motors can be driven by block shape currents (Brushless DC Linear) and sinusoidal currents (AC Synchronous Linear).

Limits

The limits for linear motors are equal to the Brushless DC and AC synchronous motors when torque is replaced by force and angular speed by linear speed.

Star and Delta Networks

Brushless DC motors and AC synchronous motors use three phase currents to connect to three groups of coils. The connection can be made use a star network or a delta network. Compared to star networks, in a delta network less voltage and more current is needed to produce the same amount of torque. For amplifiers the maximum current is directly related to the costs. That is why delta networks are only used in special cases where higher speeds than normal are required.



In 20-sim the electrical resistance and inductance are measured between the terminals (e.g. between 1 and 2 or 2 and 3 or 3 an 1). This is known as the **terminal resistance** and **inductance** or as the **phase to phase resistance** and **inductance**. There are two reasons for using terminal values. Most motor suppliers give the terminal values in their data sheet and by using terminal values the same dynamic model can be used for star networks and delta networks.

If the motor supplier gives coil values instead of terminal values, you have to calculate them by hand. The relation between the terminal values and the coil values is:

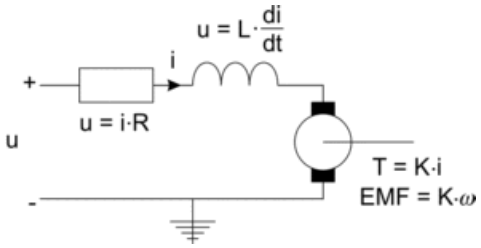
$$R_Y = 2 \cdot R_{coil} \quad R_\Delta = \frac{2}{3} \cdot R_{coil}$$

$$L_Y = 2 \cdot R_{coil} \quad L_\Delta = \frac{2}{3} \cdot R_{coil}$$

Torque Speed Plot

General Model

The DC motor model can serve as a prototype model for all permanent magnet motors.



Although the models for DC Brushless and AC synchronous motors are slightly different, the DC motor model is still very useful to explain their operation. In this chapter the DC motor model will be used to derive the torque speed plot of a motor and show the use of the torque speed plot for motor selection.

The working principle for all permanent magnet motors is the same: get a constant torque by manipulation of the coil function and the current. The relation between the torque and current is:

$$T = K_T \cdot i$$

The constant *KT* is generally known as the **torque constant**. The relation between the induced current and the rotational speed is:

$$EMF = K_e \cdot \omega$$

The constant *Ke* is generally known as the **voltage constant**. For DC motors the torque constant and voltage constant have the same value. Therefore the subscripts T and e are usually omitted. The total voltage can be described as:

$$u = i \cdot R + L \cdot \frac{di}{dt} + K \cdot \omega$$

If we assume the current is changing only very slowly, the current derivative can be neglected:

$$i = \frac{u - K \cdot \omega}{R}$$

With the torque equation:

$$T = K \cdot i$$

this leads to

$$T = \frac{K \cdot (u - K \cdot \omega)}{R}$$

The torque at zero speed (**stall torque**) is found as

$$T_{stall} = \frac{K \cdot u}{R}$$

which enables us to write the torque as:

$$T = T_{stall} - \frac{K^2}{R} \cdot \omega = T_{stall} - S \cdot \omega$$

where S is commonly know as the **steepness**. The steepness has many definitions:

$$S = \frac{T^2}{i^2 \cdot R} = \frac{K^2}{R} = \frac{\Delta T}{\Delta \omega}$$

The speed where the current is zero (no load speed) is equal to

$$\omega_0 = \frac{u}{K}$$

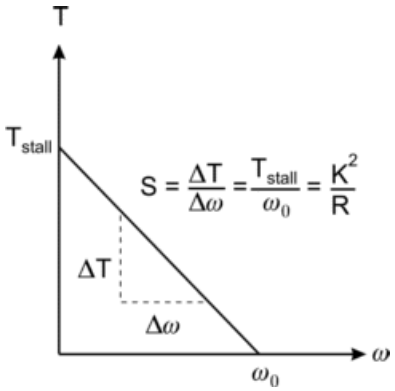
The relation between stall torque and no load speed is obvious:

$$T_{stall} = S \cdot \omega_0$$

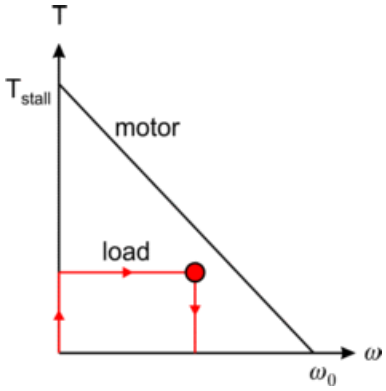
which leads to the general torque equation:

$$T = S \cdot (\omega_0 - \omega)$$

This equation can be displayed graphically and is commonly known as the torque speed plot. The graph shows the generated torque as a function of the speed. For constant voltages a straight line from the stall torque to the no load speed is found. The slope of the line is equal to the steepness S.



The torque speed plot is a useful representation of a motor because we can draw the desired load torque in this plot and to see if the motor is able to produce this torque. In the figure below the load curve is shown in red. It is a constant torque applied to a load, which makes it accelerate until a certain velocity is reached and the torque gets zero.



The red dot indicates the point of maximum load power. This point is important because it indicates the maximum power that the motor has to deliver.

Limits

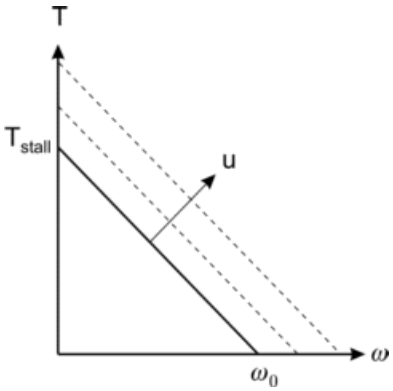
To find out if a motor is sufficient, we have to plot the motor limits in the torque speed plot. In the previous chapter we have described the following limits:

Limit	Brush DC
torque / current	**V**
speed	**V**
power	**V**

voltage	**V**

Maximum Voltage

For higher voltages, a higher stall torque and no load speed is found but the steepness does not change.

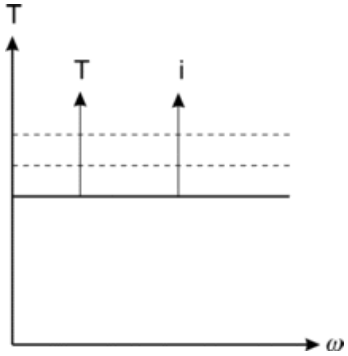


Maximum Torque / Maximum Current

In many data sheets a maximum torque is given. This is directly connected with a maximum current limit through the equation:

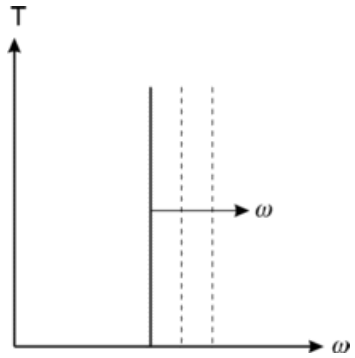
$$i = \frac{T}{K}$$

So both the maximum current and the maximum torque appear as horizontal lines in the torque speed plot.



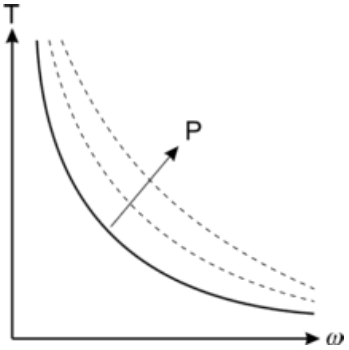
Maximum Velocity

The maximum velocity can be directly indicated in the torque speed plot as a vertical line.



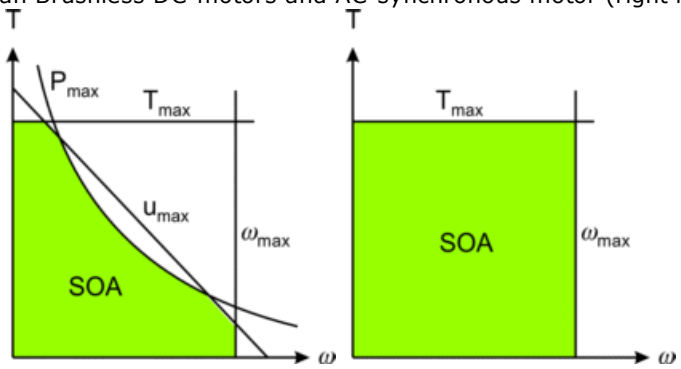
Maximum Power

The line of maximum power can be indicated as a parabola in the torque speed plot.



Safe Operating Area

If the limits are plotted in the torque speed plot, an Area of Safe Operation is found. For a proper motor, the torque speed curve of the load should be inside the **Safe Operating Area (SOA).** The figures below show the safe operating areas for the various motor types. Brush DC-motors (left figure) have a safe operating area that is more limited than Brushless DC motors and AC synchronous motor (right figure).



In practice motor suppliers of Brushless DC motors and AC synchronous motors may impose limits on the maximum voltage and power. This is mostly done to prevent heating up the motor too quickly during continuous operation, or material limits. This makes the Safe Operating Area of these motors look like that of a DC motor. Compared to DC motors, Brushless DC motors and AC synchronous motors will always allow higher torques and speeds because of the absence of mechanical commutation.

Losses

Every motor will experience losses, which results in a temperature rise. The maximum allowed temperature is limited by the magnets, the oil in the bearings and the mechanical behaviour of the brushes. Analysis of the cause of losses is therefore important for the correct choice of a motor.

Electric Dissipation

All motor coils have resistance and will therefore generate heat. The heat flow generated is equal to:

$$dQ = \frac{1}{\Delta T}\int_0^{\Delta T} i^2 \cdot R\,dt = \frac{R}{\Delta T}\int_0^{\Delta T} i^2\,dt$$

which can also be written as:

$$dQ = i_{rms}^2 \cdot R$$

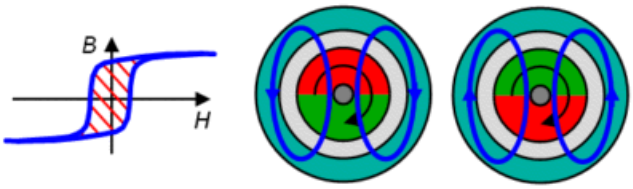
with *irms* the **root mean square** current:

$$i_{rms} = \sqrt{\frac{1}{\Delta T}\int_0^{\Delta T} i^2\,dt}$$

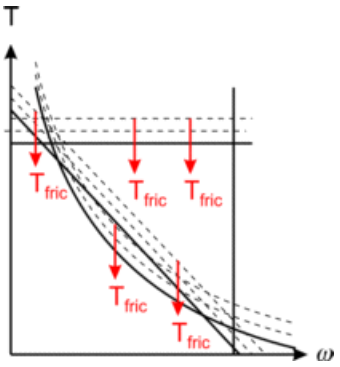
For constant DC currents the root mean square value is equal to the current itself. For sinusoidal currents, the root mean square value is equal to the amplitude of the current times the square root of 2.

Hysteresis losses

In all motors except from hollow rotor motors, the magnets move with respect to the iron core of the motor. This results in a reversing magnetic field in the iron which leads to hysteresis.

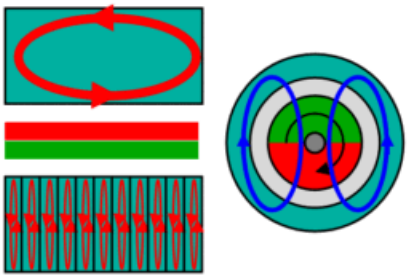


Each rotation produces an energy loss proportional to the area of the hysteresis curve (B/H). The power loss due to hysteresis is proportional to the motor speed and can therefore be represented as a friction torque. Due to this friction torque the motor can produce less output torque. This friction torque can therefore be represented in the torque speed plot as a drop down of all curves.

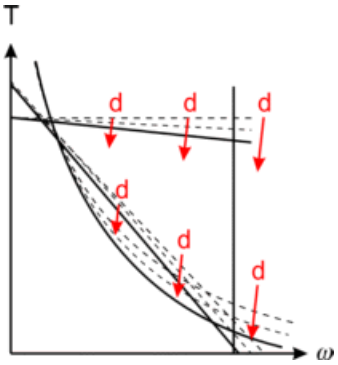


Eddy Current losses

A reversing magnetic field in iron also leads to an induce voltage. The resistance of the iron determines the resulting currents in the iron. These currents are named eddy currents.



Eddy currents heat up the iron because of its resistance. If a massive iron return is used large eddy currents will be induced with high power losses. Therefore in most motors laminated is applied, which reduce eddy currents by decreasing the path length and thus lead to smaller small power losses. The power loss due to eddy currents is proportional to the square of the motor speed and can therefore be modeled as mechanical damping. Damping will decrease the performance of the motor. It can be represented in the torque speed plot as a falling down of curves that increases with the speed.



Cogging

In a permanent magnet motors cogging torque manifests itself by the tendency of the rotor to align in a number of stable positions when unexcited. The cogging torque can be described as:

$$T_{cog} = \hat{T}_{cog} \cdot \sin\left(\frac{p \cdot \theta}{2}\right)$$

where q is the motor angle and p the number of poles. Although cogging does not consume power and is not visible in the torque speed plot, under dynamic conditions it may cause undesirable speed pulsation and also may induce vibrations and acoustic noise.

Mechanical Losses

Mechanical losses in motors are caused by bearings, brushes and air friction. Only when an air fan is mounted air friction becomes significant. The effects of brushes and bearings can be represented by coulomb friction and damping.

Temperature

As a result of the heat flow, the motor will warm up leading to two major effects. First of all the coil resistance will increase. Because all coils are made of copper, the temperature dependency of the coil resistance can be written as:

$$R_T = R_{25} \cdot \left(1 + 0.004 \cdot \left(T_{coil} - 25\right)\right)$$

where *R25* is the coil resistance at room temperature (25 degrees centigrade) and *Tcoil* the actual coil temperature in degrees centigrade. As a result of the increase of motor temperature, the performance of the magnets will decrease. The result is a decrease of the torque constant :

$$K_T = K_{25} \cdot \left(1 + ktc \cdot \left(T_{magnet} - 25\right)\right)$$

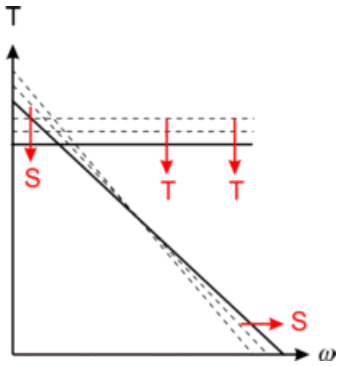
where *K25* is the torque constant at room temperature (25 degrees centigrade), *Tmagnet* the actual magnet temperature in degrees centigrade and *ktc* the temperature dependency of the used magnet.

Magnets	ktc
Ferrites	-0.002
$Nd_2Fe_{14}B$	-0.0013
SmCo	-0.0005
AlNiCo	-0.0002

Because the steepness is directly dependent of the torque constant and motor resistance

$$S = \frac{K^2}{R}$$

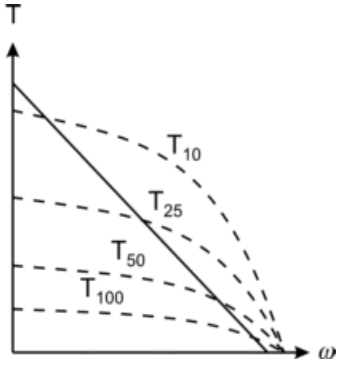
an increase of temperature will lead to a decrease of the steepness. The results of a temperature increase can therefore be shown in the torque speed plot indirectly through the decrease of the maximum torque and steepness.



Thermal Duty Cycle

Motor losses result in an increase of the temperature. During short term operation the temperature increase will be limited. During long-term operation, the generated heat and the amount of heat that can be removed via the air and motor mounting determine the temperature increase.

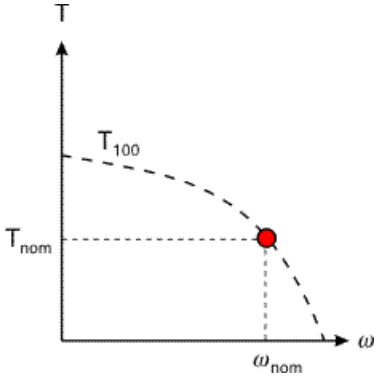
In most motors, the largest portion of the generated heat is cause by electric dissipation. The generated heat is therefore directly proportional to the current and thus the torque. Imagine a motor in long-term operation generating a constant torque. A small torque will lead to only a moderate increase of the motor temperature, but there will be a certain torque where the generated heat leads to an increase of the maximum motor temperature. This is called the **maximum continuous torque** or the **100% Thermal Duty Cycle (TDC)**. The line of the maximum continuous torque can be shown in the torque speed plot and is often denoted as T100. For lower speeds the T100 curve will be straight because the heat generation mainly depends on depends on electric dissipation. For higher speeds the T100 curve will decrease because eddy current losses start to become significant.



If the motor is in continuous operation, but a constant torque is just delivered 50% of the time, a higher torque is permissible. This torque is denoted as T50 and sometimes called the 50% thermal duty cycle. In a similar way T25 and T10 are defined.

Nominal Operating Point

Many motor manufacturers give in their data sheets a nominal or continuous current at a certain speed. This indicates an operating point on the T100 curve. The nominal or continuous current is the current that continuously be continuously be supplied to the motor at the given speed, which will result in a heating up of the coils to the maximum temperature.



The nominal operating point is always based on a certain ambient temperature. In the 20-sim Servo Motor Editor an ambient temperature of 25 ºC is used.

The choice of the speed for the nominal operation point is arbitrary. Some motor suppliers use the speed that results in a maximum power output at the motor shaft. Some motor suppliers use a speed that is half of the maximum allowed speed.

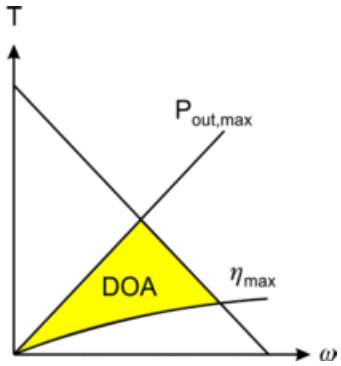
Maximum Power and Maximum Efficiency

The purpose of a motor is to deliver power to a load. Given a certain voltage u, the delivered power is equal to the torque times the speed:

$$P = S \cdot \left(\frac{u}{K} - \omega \right) \cdot \omega$$

The maximum delivered power *Pout,max* for a given voltage can be found by differentiating this function and setting it to zero. The result will be a specific torque and speed. If this process is repeated for other voltages, a line of points where maximum power is delivered will result.

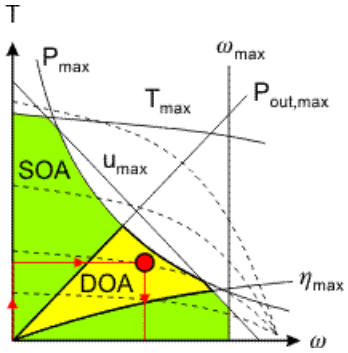
The efficiency *h* of a motor is defined as the net output power divided by the input power. For a given voltage, the maximum efficiency can be found just like the maximum power and will lead to a certain torque and speed. By repeating this procedure for other voltages, a line of points where maximum efficiency is obtained will result.



The resulting lines of maximum power and maximum efficiency can be shown in the torque speed plot. Choosing a good motor is always a compromise between efficiency and output power. A good choice is to choose a motor that has the major part of the load curve inside the lines of maximum power and maximum efficiency. This area is called the Desired Operating Area.

Choosing a Motor

If we combine the results of the previous sections, a torque speed plot results as shown below. The arrows indicate the load curve with the circle as point of maximum load power. The **Safe Operating Area** shows the limits of operation due to motor specific parameters such as the maximum torque and speed.



The lines of maximum output power and maximum efficiency are the edges of the **Desired Operating Area**.The procedure for choosing a proper motor is:

1. Determine the load curve and the point of maximum load power.
2. Determine the duty cycle *d* (the percentage of the time that torque is required) and estimate the corresponding *Td* line with help of the shown *T100, T50 , T25* and *T10* lines.
3. Select a motor such that the following demands are satisfied:

- The load curve is completely beneath the Td line (no overheating).
- The complete load curve is in the Save Operating Area.
- The point of maximum load power is in the Desired Operating Area.
- The point of maximum load power is as close to maximum voltage line as possible (we want to choose the smallest motor that can do the job).
- The load curve is as much in the Desired Operating Area as possible.

Thermal Behaviour

DC Motors

Every motor generates heat, even when at zero speed, and this will lead to an increase of the motor temperature. Especially when running in continuous operation, there is a good change that a motor, which is perfectly capable of generating the needed output power, will heat up beyond its thermal limit and break down.

As explained in the previous section the thermal limit is indicated in the torque speed curve by the maximum continuous torque line. To generate such a line, a thermal model is used. This thermal model is also part of the dynamic model that is used in 20-sim. This section describes the thermal model of permanent magnets motors.

In standard brush DC motors the coils are part of the rotor. To create a thermal model, various components will be identified first.

Heat Generation

Heat is generated in the motor at every place where energy is lost: coils, bearings, iron etc. Because the main source of energy loss is in the coils due to the coil resistance, we start at the coils. The coils act as a heat source and generate a heat flow equal to:

$$dQ = i^2 \cdot R$$

Thermal Capacity

The rotor package, consisting of the coils and possibly iron and other material, will heat up due to the generated heat. The temperature increase depends on the generated heat *dQ*, the heat capacity of the rotor *Ccoil* and the transfer of heat to the stationary housing of the motor *dQcoil-housing:*

$$T_{coil} = \frac{1}{C_{coil}} \cdot \int dQ - dQ_{coil-housing}\, dt$$

Thermal Resistance

The heat flow from the coils to the housing *dQcoil-housing* depends on the temperature difference between the rotor and the housing and the thermal resistance *Rcoil-housing* :

$$dQ_{coil\text{-}housing} = \frac{\left(T_{coil} - T_{housing}\right)}{R_{coil\text{-}housing}}$$

Thermal Capacity

Like the rotor, the housing will also heat up. The temperature increase depends on the heat coming from the rotor *dQrotor-stator*, the heat capacity of the housing Chousing and the transfer of heat to the environment *dQhousing-amb*:

$$T_{stator} = \frac{1}{C_{housing}} \cdot \int dQ_{coil-housing} - dQ_{housing-amb}\, dt$$

Thermal Resistance

The heat flow from the stator to the environment stator *dQhousing-amb* depends on the temperature difference between the housing and the environment and the thermal resistance *Rhousing-amb:*

$$dQ_{housing-amb} = \frac{\left(T_{housing} - T_{amb}\right)}{R_{housing-amb}}$$

The thermal resistance *Rhousing-amb* depends on the motor mounting and external cooling. In most data sheets, a value for the thermal resistance can be found based on standard motor mounting.

Environment

The environment is supposed to have a fixed temperature *Tamb*.

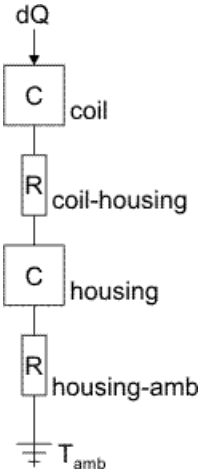
Other Motors

The same components as found for the DC brush motor can be found for other motor types. We only have to realize that for Brushless DC motors and AC Synchronous motors the coils are mounted on the housing. This means the thermal resistance Rcoil-housing is relatively small and excess heat can be transferred relatively easy. Therefore these motors are very robust for overheating.

In AC synchronous linear motors, the coils are mounted are part of a platform, on which the load can be attached. To get the same thermal model we consider the platform to be the housing. Depending on the type of motor the housing can have a medium thermal capacity (iron core motors) or a very small thermal capacity (ironless motors). Because the all excess heat has to be removed through the air, linear motors will heat up easily. That is why sometimes forced cooling (water) is applied.

Thermal Model

If we combine all components we get the complete thermal model. It can be represented as shown in the figure below.



The values for thermal capacities and resistances can be found in most data sheets. These values are, however based on general conditions. The value of the thermal resistance *Rhousing-amb* can vary considerably, based on the specific mounting of the motor and use of passive or active cooling. Most motor suppliers will indicate the how the thermal parameters were derived and how the motor should be mounted to meet the general conditions.

8.6 Real-Time Toolbox

8.6.1 Introduction

Real Time Toolbox

The Real Time toolbox of 20-sim allows you to create C-code out of any 20-sim model for the use in real-time applications:

1. Create ANSI-C code for use in a real-time environment.
2. Create Matlab / Simulink S-functions for the use in the Matlab Real-Time Workshop.

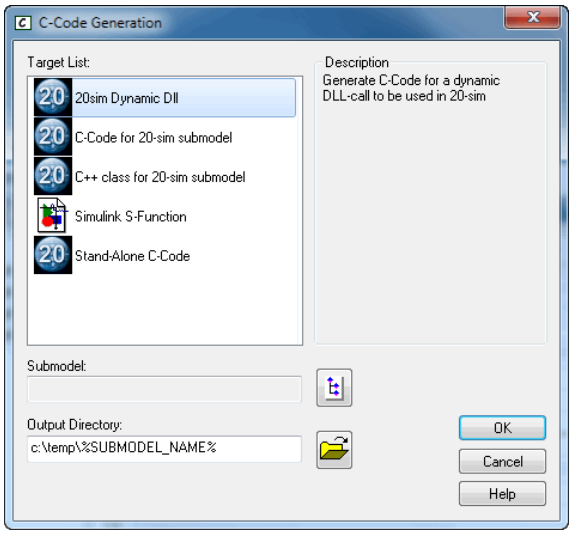
The generation of Matlab / Simulink code is can be performed using the Export command of the Editor File menu. You can also use the C-code generation command of the Simulator **Tools** menu.

The generation of C-code can be done with the C-code generation command of the Simulator **Tools** menu.

8.6.2 C-Code Generation

C-Code Generation

20-sim has a C-Code Generator which automatically converts a complete 20-sim model or submodel into C-Code. The application can be used to generate Matlab™/Simulink™ S-Functions, to generate Stand-Alone executables or to generate input/output functions for use in other C and C++ programs. The ANSI C-Code Generator can be opened from the Simulator (**Tools** menu, **C-Code Generation** command).



- Depending on the target selected, 20-sim will generate specific C-code.
- You can add more targets by adding paths in the General Properties.

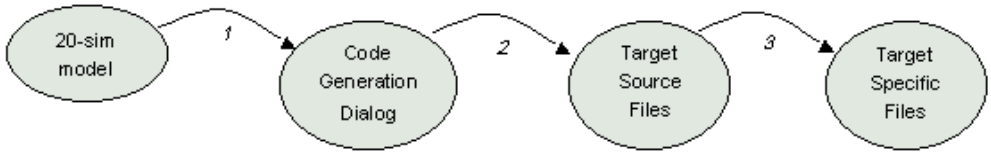
Generating ANSI C-Code

Overview

The modeling and simulation package 20-sim is capable of generating ANSI C-Code for several targets. Possible targets are Stand-Alone C, a C-Function, or a Simulink S-Function. Since 20-sim generates well-documented ANSI C-Code, it is easy for a user to make modifications or extensions to the generated code by hand. However it is also possible to fully define your own code target using C-Code templates, and to make modifications in these templates, so the resulting source code is tailor made for your application.

General Structure

The code generation process of 20-sim typically has the following structure:



Step 1

1. The process starts with a model and simulation in 20-sim.
2. In the 20-sim Simulator, the user selects the **C-Code Generation** command from the **Tools** menu.
3. 20-sim collects information about the current model.
4. A target configuration file is read that specifies the different code generation targets that exist.
5. The **C-code generation dialog** is opened, allowing the user to specify the target to use.

Step 2

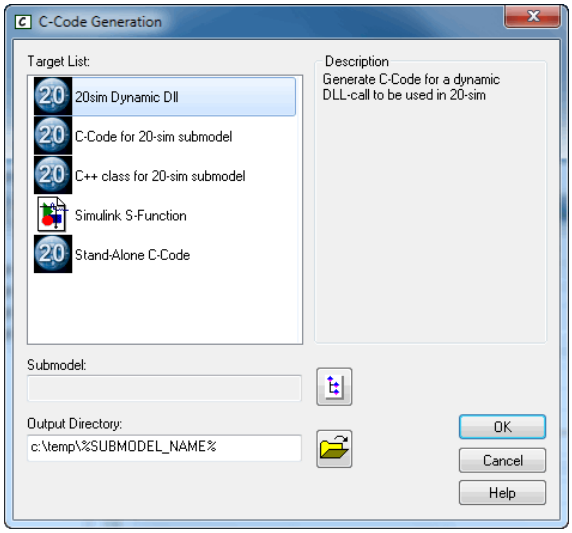
1. The selected target refers to specific template files (source files) in a target specific subdirectory.
2. 20-sim adapts these template files with the model information and generated code for the model.

Step 3

1. The resulting files are placed in a destination directory.
2. These files are now specific for the target and the model.
3. If desired, 20-sim can call additional commands for further processing (f.i. make, run etc).

Code Generation Dialog

After 20-sim reads the target configuration file, the code generation dialog is shown with the information of this file:



The target list shows the names of the available targets. When a selection is made, the description of the selected target is given at the right. A submodel can be selected when the target requires this. The output directory can be overruled.

Targets

The user can create own targets. All targets are defined in the target definition file `Targets.ini`. This file has a typical INI-file structure and starts with a section that simply enumerates which targets exist. Each target then has its own section that holds the remaining information, like a short description of the target, the name of the template directory, the names of the template source files, additional commands to perform, etc. The following keywords may appear in a target section:

Templates

Each target can point to a set of template files that are used to create C-Code with. Own template files can be created by the user, with the help of the most powerful feature of 20-sim C-Code generation: the use of tokens! A token is a placeholder for model-dependent information. For instance in the C-Code Generation Dialog, the target destination directory contains the name of the selected submodel (e.g. c:\ temp\% SUBMODEL_NAME%) by default. Since this information is not yet known when the targets.ini file is created, a specific token that refers to this name is used instead. A complete description of predefined tokens is described in the section about Available 20-sim Tokens.

Generation Result

After the tokens in the target template files are replaced, the resulting target and model specific files will be placed in the destination directory. If commands are specified in the `Targets.ini` file, these commands are performed. This allows the user to call scripts automatically. E.g. for automatic compilation, linking and running of the code in a certain target environment.

Target.Ini File

The user can create own targets. All targets are defined in the target definition file `Targets.ini`. This file has a typical INI-file structure and starts with a section that simply enumerates which targets exist. Each target then has its own section that holds the remaining information, like a short description of the target, the name of the template directory, the names of the template source files, additional commands to perform, etc. The following keywords may appear in a target section:

targetName

="string"
The name that will appear in the 20-sim C-Code Generation Dialog.

iconFile

="string"
The name of an icon file (.ico) that contains an icon to appear in the 20-sim C-Code Generation Dialog.

description

="string"
The string that will appear in the description field in the 20-sim C-Code Generation Dialog.

templateDirectory

="string"
Here the path name where the template files for the c-code can be found can be specified. The default name is the target name in the CCode directory of 20-sim. If no full path is specified, the Ccode directory in 20-sim is taken as a starting point.

templateFiles

`=filename1; filename2; filename3...`
A list of files, semicolon-separated, that specify the files that are generated in the targetDirectory.

targetDirectory

="string"
This holds the default target directory where the files will be generated. This directory name will appear in the 20-sim dialog box when C-Code is generated and can be overruled by the user.

submodelSelection

=TRUE (default)
=FALSE
Determines whether C-Code is generated for the complete 20-sim model, or that a submodel selection is required.

preCommand

="string"
A command which will be executed in the target directory before that the C-Code will be generated.

postCommand

="string"
A command which will be executed in the target directory after that the C-Code has been generated. For example a "make" command can be given to automatically compile the generated code for the specific target.

newLineCharacter

=0	CRLF (0x0d0a = DOS Standard)
=1	CR (0x0d = Macintosh Standard)
=2	LF (0x0a = Unix Standard)

Enter a number for the kind of newline character that should be used.

%KEYWORD%

=value
This (re)defines the keyword "KEYWORD" and gives it the contents "value". Own keywords can be defined in this manner as well.
example:

```
%XX_TIME%=someTime
```

will redefine the time variable with the value "someTime"

Example

A valid Targets.ini file (defining three targets) may look like below. The file may also contain specific 20-sim tokens (%MODEL_NAME%) that are described later on.

```
; Possible targets for 20-sim C-Code Generation
;
[targets]
StandAloneC
CFunction
20simDLL
Simulink


; Generate Stand-Alone C-Code for the complete 20-sim model
;
[StandAloneC]
targetName="Stand-Alone C-Code"
iconFile="20sim.ico"
description="Use this target when testing the complete 20-sim model as a single process."
SubmodelSelection=FALSE
templateDirectory="StandAloneC"
```

```
templateFiles=xxfuncs.c;xxfuncs.h;xxinteg.c;xxinteg.h;xxinverse.c
templateFiles=xxmain.c;xxmatrix.c;xxmatrix.h;xxmodel.c;xxmodel.h
templateFiles=xxtypes.h;%MODEL_NAME%.dsp;%MODEL_NAME%.dsw
targetDirectory="c:\temp\%MODEL_NAME%"


; Generate C-Code for a selected Submodel
;
[CFunction]
targetName="C-Code for 20-sim submodel"
iconFile="20sim.ico"
description="This is the C-Code as it was generated for a submodel in 20-sim version
3.1"
templateDirectory="CFunction"
templateFiles=xxfuncs.c;xxfuncs.h;xxinteg.c;xxinteg.h;xxinverse.c
templateFiles=xxmain.c;xxmatrix.c;xxmatrix.h;xxmodel.c;xxmodel.h
templateFiles=xxsubmod.c;xxsubmod.h;xxtypes.h;%SUBMODEL_NAME%.dsp;%
SUBMODEL_NAM
E%.dsw
targetDirectory="c:\temp\%SUBMODEL_NAME%"

; Generate C-Code for a dynamic DLL-call to be used in 20-sim
;
[20simDLL]
targetName="20sim Dynamic Dll"
iconFile="20sim.ico"
description="Generate C-Code for a dynamic DLL-call to be used in 20-sim"
templateDirectory="20simDLL"
templateFiles=xxfuncs.c;xxfuncs.h;xxinverse.c
templateFiles=xxmatrix.c;xxmatrix.h;xxmodel.c;xxmodel.h
templateFiles=xxtypes.h;%SUBMODEL_NAME%.c
templateFiles=%SUBMODEL_NAME%.dsw;%SUBMODEL_NAME%.dsp;%
SUBMODEL_NAME%.emx
targetDirectory="c:\temp\%SUBMODEL_NAME%"

; Generate C-Code for a Simulatink S-Function
;
[Simulink]
targetName="Simulink S-Function"
iconFile="mdl.ico"
description="This generates C-Code for a submodel to be used in Matlab/Simulink"
templateDirectory="Simulink"
templateFiles=%SUBMODEL_NAME%.c;xxinverse.c;xxmatrix.c;xxmatrix.h
templateFiles=xxmexfcs.c;xxmextps.h;xxtypes.h;%SUBMODEL_NAME%_.mdl
targetDirectory=c:\temp\%SUBMODEL_NAME%
postCommand=mex %SUBMODEL_NAME%.c
```

Available 20-sim Tokens

Both the target configuration file and the code generation dialog revealed the most important part of the 20-sim code generation process, the use of tokens!

A token is a placeholder for model-dependent information. For instance in the code generation dialog, the target destination directory contains the name of the selected submodel by default. Since this information is not yet known when the `targets.ini` file is created, a specific token that refers to this name is used instead.

The idea is to create targets from template source files that contain tokens instead of actual model-dependent information (like equations, names, parameters, inputs etc). The files that are specified in a target section will be scanned for these tokens and tokens will be replaced by the corresponding model-dependent information.

Predefined variable names

The following variables should be declared in the C-Code template, because 20-sim uses these names in the generation of the equations:

c: constant array
P: parameters array
V: variables array
s: states array
R: rates array
M: matrix array
U: unnamed variables array
F: favorite variables array
f: favorite parameters array

Note that these parameters are case-sensitive!

Model Data

The following tokens will be replaced by numbers indicating the number of model parameters etc., typically used in memory allocation parts and loops.

%	The number of constants in the model
NUMBER_CONSTANTS	The number of parameters in the model
%	The number of variables in the model
%	The number of states in the model
NUMBER_PARAMETERS	The number of inputs of the model
%	The number of outputs of the model
%NUMBER_VARIABLES	The number of matrices used in the model
%	The number of unnamed variables in the model
%NUMBER_STATES%	The number of favorite parameters in the model
%NUMBER_INPUTS%	The number of favorite variables in the model
%NUMBER_OUTPUTS%	The size of the largest workarray necessary in calculating
%NUMBER_MATRICES	some matrix functions
%	

%NUMBER_UNNAMED
%
%
NUMBER_FAVORITE_PA
RAMETERS%
%
NUMBER_FAVORITE_VA
RIABLES%
%WORK_ARRAY_SIZE
%

The following tokens are already reserved for future use:

%NUMBER_DEPSTATES	The number of dependent states in the model
%	The number of algebraic loop variables pairs in the model
%NUMBER_ALGLOOPS	The number of constraint variable pairs in the model
%	The number of import variables in the model
%	The number of export variables in the model
NUMBER_CONSTRAINT	
S%	
%NUMBER_IMPORTS%	
%NUMBER_EXPORTS%	

Example

Variable name arrays

If you want to use parameters etc. that are specific to a model you can use arrays of names using the following tokens:

%CONSTANT_NAMES%	All the constants used in the model.
%PARAMETER_NAMES	All the parameters used in the model.
%	All the variables used in the model.
%VARIABLE_NAMES%	All the states used in the model.
%STATE_NAMES%	All the rates used in the model.
%RATE_NAMES%	All the dependent states used in the model.
%DEPSTATE_NAMES%	All the dependent rates used in the model.
%DEPRATE_NAMES%	All the algebraic loop variables used in the model.
%ALGLOOP_NAMES%	All the constraint variables used in the model.
%CONSTRAINT_NAMES	All the inputs used in the model.
%	All the outputs used in the model.
%INPUT_NAMES%	All the matrices used in the model.
%OUTPUT_NAMES%	All the favorite parameters used in the model.

%MATRIX_NAMES%	All the favorite variables used in the model.
%FAVORITE_PARAMETER_NAMES%	
%FAVORITE_VARIABLE_NAMES%	

Example

Initialization code

An important part of a model are the parameters, initial values states etc. To match these with the generated model code and use the correct values the following tokens can be used:

%INITIALIZE_CONSTANTS%
%INITIALIZE_PARAMETERS%
%INITIALIZE_MATRICES%
%INITIALIZE_STATES%
%INITIALIZE_DEPSTATES%
%INITIALIZE_ALGLOOPS%
%INITIALIZE_CONSTRAINTS%
%INITIALIZE_INPUTS%
%INITIALIZE_OUTPUTS%
%INITIALIZE_FAVORITE_PARS%
%INITIALIZE_FAVORITE_VARS%

Example

Equations

The following tokens can be used to place 20-sim simulation model equations into the C-Code:

%INITIAL_EQUATIONS%	Equations that should be calculated once for initialization of the model
%STATIC_EQUATIONS%	Equations that should be calculated once
%INPUT_EQUATIONS%	Equations that should be calculated once at the beginning of every simulation step
%DYNAMIC_EQUATIONS%	Equations that calculates the dynamic part of the model, calculates the rates
%OUTPUT_EQUATIONS%	Equations that should be calculated once at the end of every simulation step

%OUTPUT2_EQUATIONS%	Reduced set of equations that should be calculated at the end of every simulation step
%FINAL_EQUATIONS%	Equations that should be calculated once for termination of the model

Inputs/Outputs

To match model inputs and model outputs to for example sensor signals and actuator signals you can use:

%INPUT_TO_VARIABLE_EQUATIONS%
%VARIABLE_TO_OUTPUT_EQUATIONS%
%ALIAS_EQUATIONS%
%FAVORITE_PARS_EQUATIONS%
%FAVORITE_VARS_EQUATIONS%

Example

Additional tokens

%INTEGRATION_METHOD_NAME%	A string representing the name of the selected integration method, for now only Euler, RungeKutta4 and Discrete are available.
%XX_TIME%	The name of the simulation time variable (combined with %VARPREFIX%)
%XX_INITIALIZE%	The name of the variable that indicates the initialization phase(combined with %VARPREFIX%).
%START_TIME%	Start time of the simulation (floating point notation).
%FINISH_TIME%	Finish time of the simulation (floating point notation).
%TIME_STEP_SIZE%	Step size of the simulation (floating point notation).
%MODEL_IS_DISCRETE%	Either TRUE or FALSE, depending if the model is discrete in time or continuous in time.

Experiment Tokens

Tokens which hold information about the model, experiment, user and system:

%FILE_NAME%	The file name of the generated C-Code.
%MODEL_FILE%	The complete path and file name of the original 20-sim model.
%MODEL_NAME%	The name of the 20-sim model out of which C-Code was generated.
%SUBMODEL_NAME%	The name of the 20-sim submodel out of which C-Code was generated.
%EXPERIMENT_NAME%	The experiment name accompanying the 20-sim (sub)model.

%GENERATION_TIME	The time of C-Code generation.
%	The date of C-Code generation.
%GENERATION_DATE	The 20-sim version and build number.
%	The directory where the generated C-Code files are stored.
%GENERATION_BUILD	The user name given in the 20-sim license.
%	The company name given in the 20-sim license.
%GENERATION_DIR%	The directory where 20-sim is installed.
%USER_NAME%	
%COMPANY_NAME%	
%20SIM_DIR%	

8.7 Time Domain Toolbox

8.7.1 Time Domain Toolbox

The Time Domain Toolbox contains powerful tools to inspect the behaviour of your model using time domain simulation.

- *Parameter sweep*: Perform a number of simulation runs, while changing model parameters.
- *Optimization*: Optimize a given result by changing parameters.
- *Curve Fitting:* Fit your model to a given result by changing parameters.
- Tolerance Analysis
  - *Sensitivity:* Change parameters by a given percentage and monitor results.
  - *Monte Carlo*: Change parameters statistically and monitor results.
  - *Variation Analysis*: Find the statistical range of parameters to yield a given result.
- *External DLL*: Let the multiple run be controlled by a user-defined function in a DLL.

Procedure

1. From the **Toolbox** menu select the **Time Domain Toolbox** and then the tool that you want to use.
2. This will open the *Multiple Run Wizard*. Follow the instructions and after all settings have been entered, close the wizard.
3. Select the **Multiple Run** command from the **Simulation** menu to perform the specified analysis.
4. After simulation is completed a special window will open to showing the results. If it is not available, you can always open this window, by selecting the **Multiple Run Results** command of the **View** Menu.

Multiple Run Wizard

All tools of the Time Domain Toolbox are started use the Multiple Run Wizard. You can open this Wizard directly by selecting the **Multiple Run** command from the **Properties** menu.

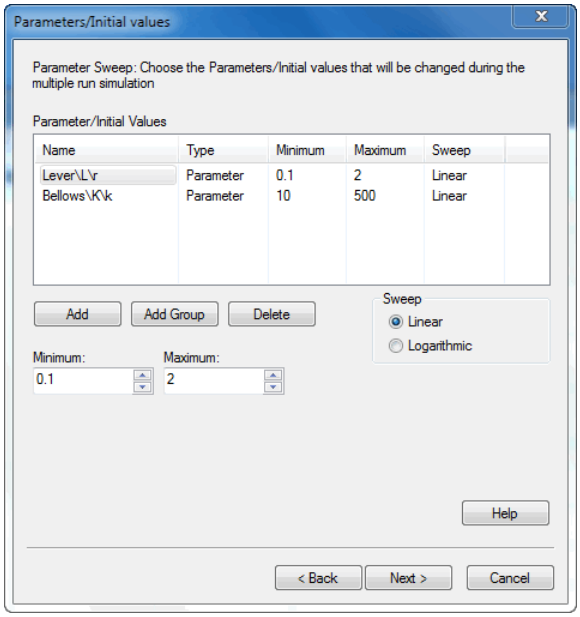
8.7.2 Parameter Sweep

Using the Parameter Sweep option of the *Time Domain Toolbox*, you can perform a predefined number of simulation runs with variation of parameters / initial values.

Parameters / Initial Values

1. From the **Tools** menu select the **Time Domain Toolbox** and then **Parameter Sweep**.

A window will open, asking you which parameters / initial values should be changed during the multiple simulations:

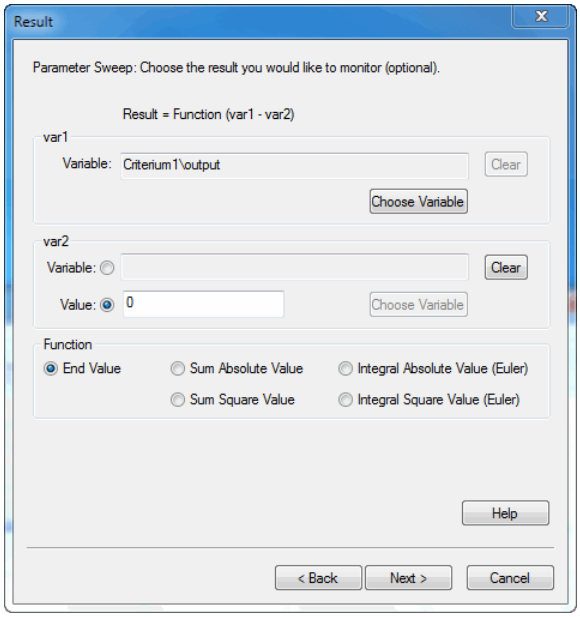


2. Add the parameters / initial values of your choice:

3. Click the **Next** button.

Result

A window opens asking you which result (optional) should be shown after the simulations:

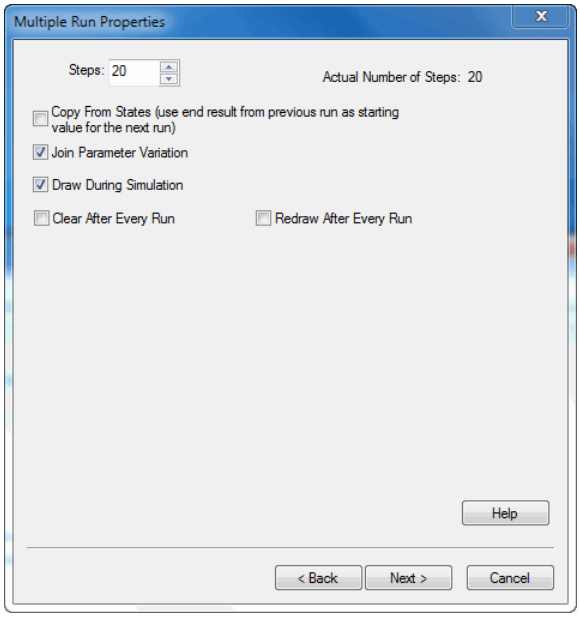


4. If you do not want to use a result after simulation, enter nothing here. If you like to present a result after the simulation, enter it here using the various options of the window.

5. Click the **Next** button.

Run options

A window will open, asking some simulation run options:



You can select the following items:

- *Steps*: Number of simulations runs.
- *Copy from States*: Use the result of the previous simulation runs as the starting value
- numerical output of the parameter sweep.
- *Join Parameter Variation*: Change all parameters simultaneously.
- *Clear After Every Run:* Clear the simulation plot when a new simulation run starts.
- *Redraw After Every Run*: Update the scaling after every run.

6. Select the desired run options and click the **Next** button.

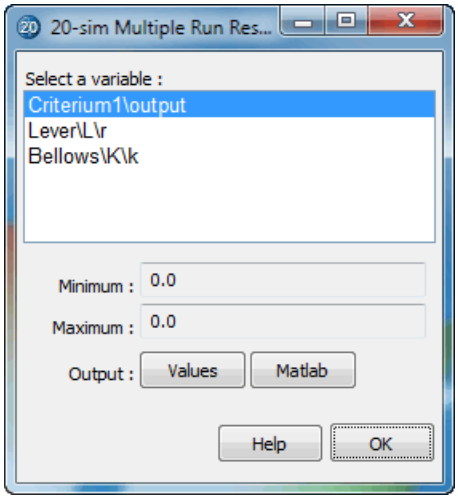
A window will open with a summary of the chosen options. If your are not satisfied you can use the **Back** button to go to a previous window and change settings.

7. Click the **Finish** button to close the **Multiple Run Wizard**.

8. From the **Simulation** menu click the **Multiple Run** command to perform the Parameter Sweep.

Parameter Sweep Results

After simulation has been done, a Multiple Run Results window will open, showing the minimum and maximum value of the result and chosen parameters:



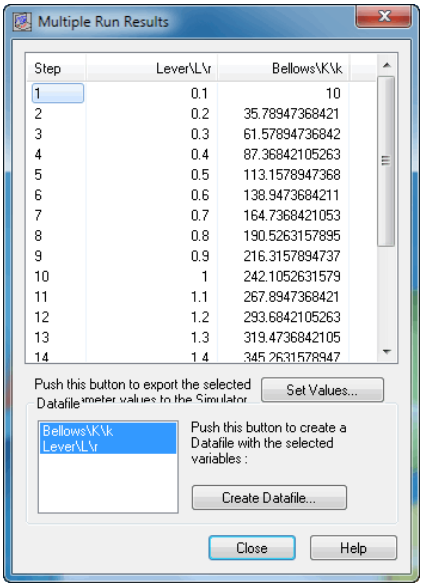
You can select the following items:

- *Values*: Choose this button to open the list that shows the numerical output of the parameter sweep.
- *Matlab*: Export the results to Matlab.
- *OK:* Close the Multiple Run Results Window.

You can always re-open the Multiple Run Results window, by selecting the **Multiple Run Results** command of the **View** Menu.

9. By choosing the Values button, a new window is opened, showing the numerical results of the parameter sweep.

In the **Results** list of this window the subsequent runs are shown and (if available) the value of the chosen result:



You can select the following items:

- *Set Values*: Choose the values of the selected run, as new parameter values of your model.
- *Create Datafile*: Use this button to store the values of the selected variables on file. Use the control-key and shift-key to make multiple selections.

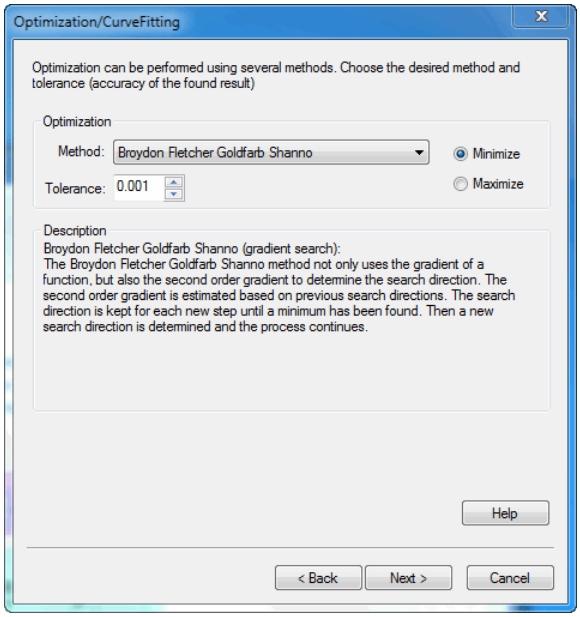
8.7.3 Optimization

Using the Optimization option of the *Time Domain Toolbox*, you can optimize a given result using variation of parameters / initial values.

Optimization Method

1. From the **Tools** menu select the **Time Domain Toolbox** and then Optimization.

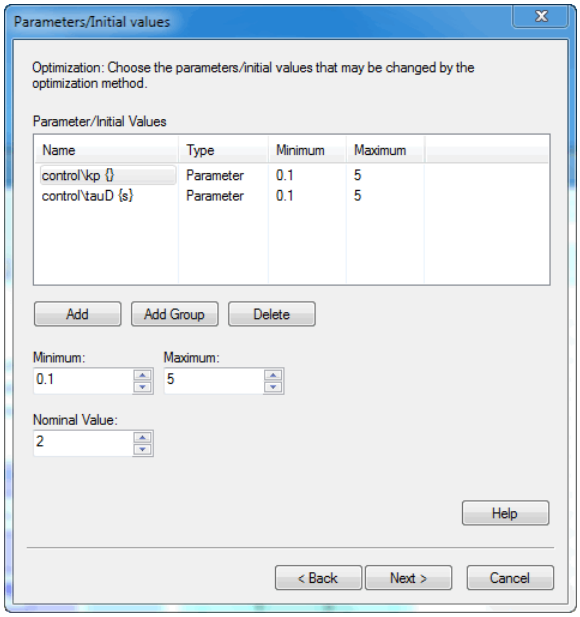
A window opens asking you which optimization method should used:



2. Select the Optimization method that should be used.
3. Select the Tolerance (accuracy of the found optimum) that should be used and click **Next.**

Parameters / Initial Values

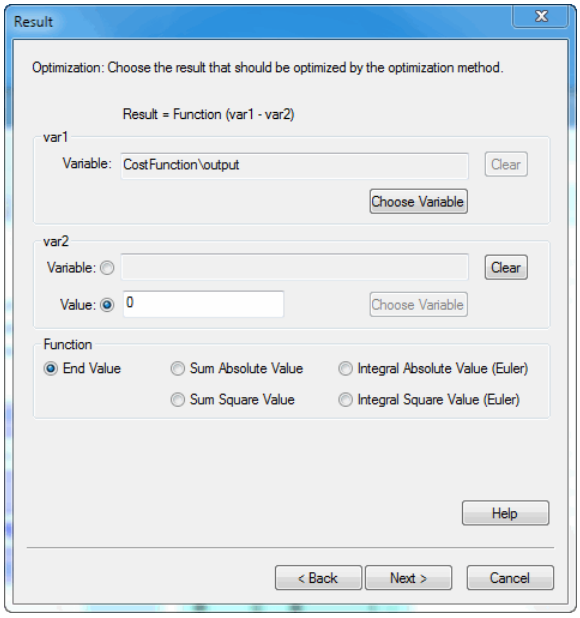
A window opens asking you which parameters / initial values should be varied during optimization:



4. Add the parameters / initial values of your choice:
5. Click the **Next** button.

Result

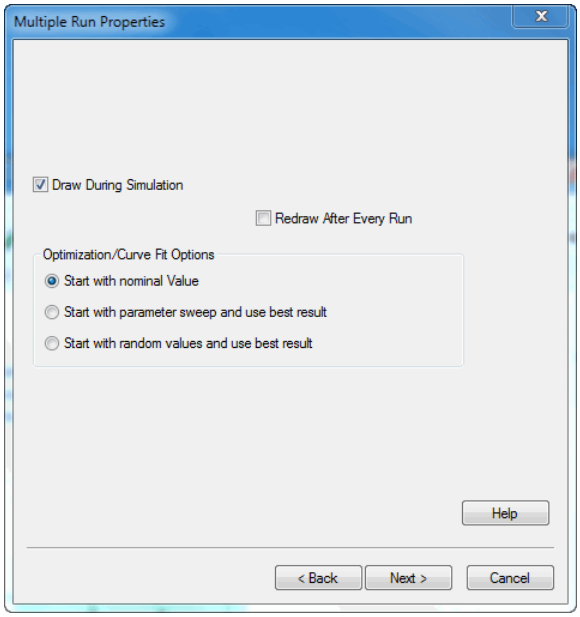
A window opens asking you which result should optimized:



6. Enter the result here using the various options of the window.
7. Click the **Next** button.

Run options

A window opens asking some simulation run options:



8. Select the desired run options and click the **Next** button.

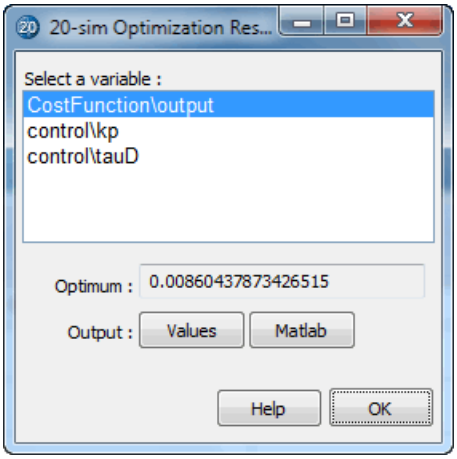
A window opens with a summary of the chosen options. If your are not satisfied you can use the **Back** button to go to a previous window and change settings.

9. Click the **Finish** button to close the **Multiple Run** Wizard.

10. From the **Simulation** menu click the **Multiple Run** command to perform the Optimization.

Optimization Results

After simulation has been done, an Optimization Results window will open, showing the optimum value of the result and corresponding parameters:



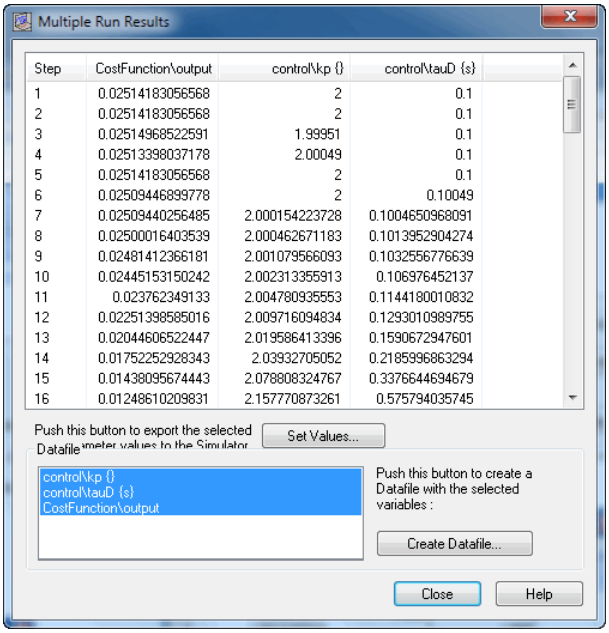
You can select the following items:

- *Values:* Choose this button to open the list that shows the numerical output of the optimization runs.
- *Matlab:* Export the results to Matlab.
- *OK:* Close the Optimization Results Window.

You can always re-open the Multiple Run Results window, by selecting the **Multiple Run Results** command of the **View** Menu.

11. By choosing the **Values** button, a new window is opened, showing the numerical results of the parameter sweep.

In the **Results list** of this window the subsequent runs are shown and (if available) the value of the chosen result:



You can select the following items:

- *Set Values:* Choose the values of the selected run, as new parameter values of your model.
- *Create Datafile*: Use this button to store the values of the selected variables on file. Use the control-key and shift-key to make multiple selections.

8.7.4 Optimization Methods

In the Multiple Run Wizard you have to specify a result which will be optimized. The result is a function of model variables and the simulation run:

result = f(i,v1,v2,...)

with i the number of the simulation run. You also have to specify the parameters that should be varied to find the optimum result. We can group these parameters in a parameter vector:

parameter vector = p(i)

with again *i* the number of the simulation run. All methods in 20-sim for finding an optimum of the result use the same iterative process:

1. The initial parameter vector is determined, e.g. *p(1)* and the corresponding function value *f(1)*.
2. A search direction *r(1)* and a stepsize *s(1)* are determined.
3. Perform a new simulation run to find the parameter vector *p(2) = p(1) + s(1)*r(1)*.

4. Calculate *f(2)*.
5. When the *f(2)* is smaller than *f(1)* and the difference between the two is smaller than a given tolerance, stop the process. The optimum has been found.
6. When the *f(2)* is smaller than *f(1)* and the difference between the two is larger than a given tolerance, proceed the process at step 2
7. Otherwise a new stepsize and/or a new search direction are determined and the process is proceeded at step 3.

The choice of the stepsize is of importance for the speed an accuracy of the search process. A small stepsize will make the optimization process last very long. A large stepsize will make it less accurate. Most methods will therefore use a variable stepsize.

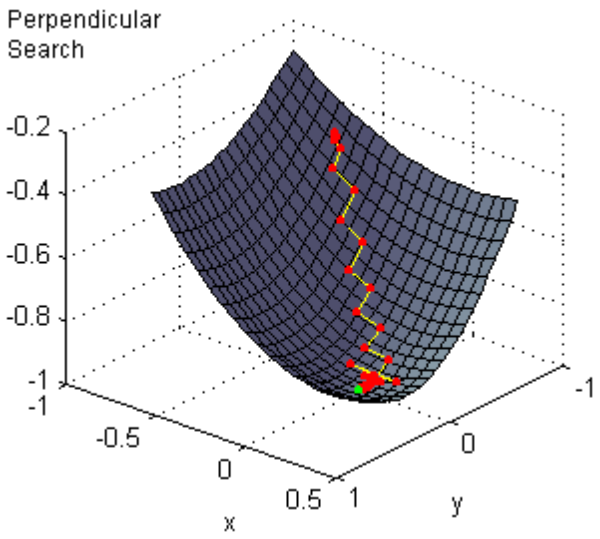
Of equal importance is the proper choice of the search direction. Methods for finding the search direction, can be divided in two groups: direct search methods and gradient search methods. The gradient of a function, is its slope at a certain point. Gradient search methods use this slope to find the optimal direction of search.

Methods

The optimization methods that are supported in 20-sim will now be explained. The pictures at the right visualize the methods with two varying parameters x (horizontal) and y (horizontal) and the corresponding result (vertical).

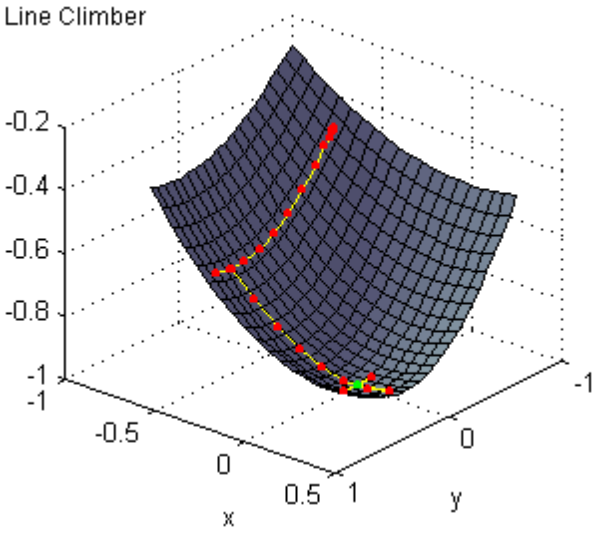
1. Perpendicular Search (direct search)

The perpendicular search method uses a search direction that is always perpendicular to the parameter axis. This means that only one parameter at a time is varied. All other parameters keep the same value. After one step, a next parameter is taken and the process continues.



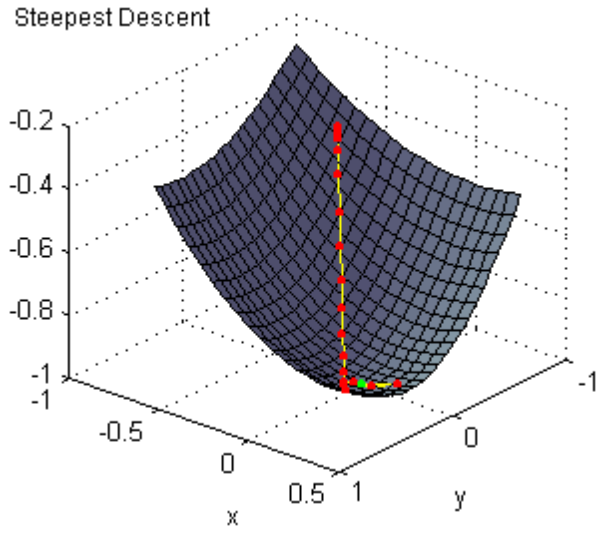
2. Line Climber (direct search)

The line climber method uses a search direction that is always perpendicular to the parameter axis. This means that only one parameter at a time is varied. All other parameters keep the same value. After a minimum has been found the next parameter is varied and the process continues.



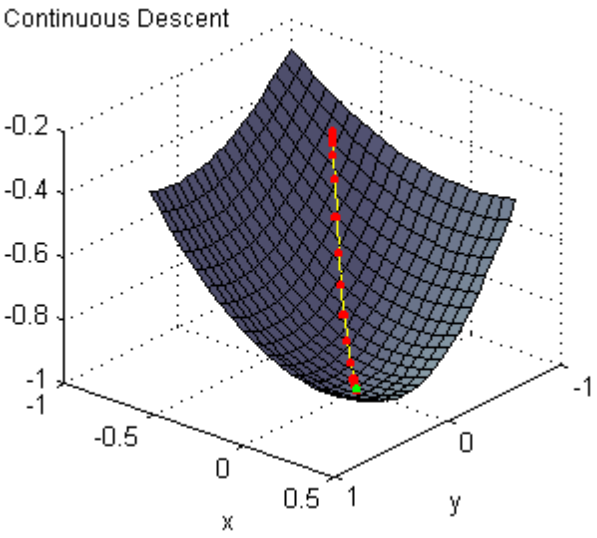
3. Steepest Descent (gradient search)

The steepest descent method starts its search in the direction of the steepest slope. This direction is kept for each new step until a minimum has been found. Then a new search direction is determined and the process continues.



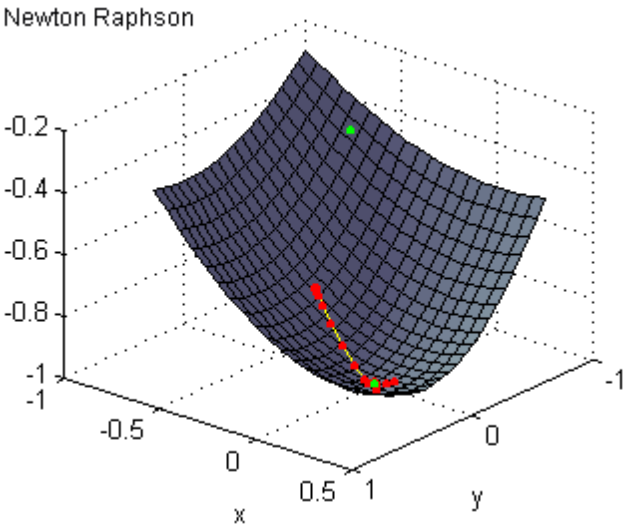
4. Continuous Descent (gradient search)

The continuous descent method starts its search in the direction of the steepest slope. After each new step a new search direction is determined and the process continues.



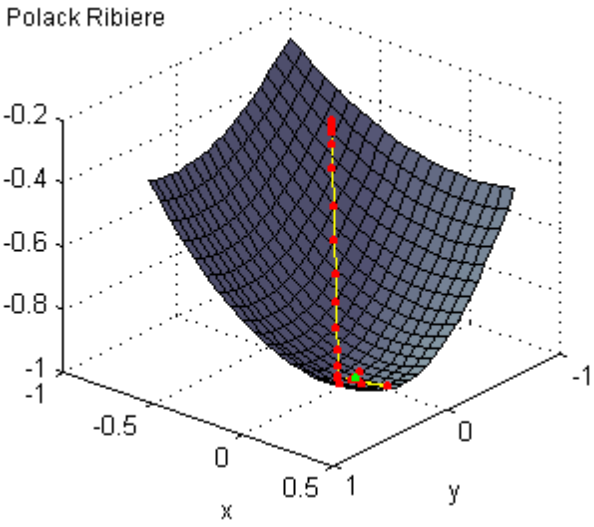
5. Newton Raphson (gradient search)

The Newton Raphson method not only uses the gradient of a function, but also the second order gradient to determine the search direction. This direction is kept for each new step until a minimum has been found. Then a new search direction is determined and the process continues. Note: The method only converges for a positive second order gradient, i.e. near the minimum. This is shown in the figure to the right. For x = -0.7 and y = -0.9 the method does not converge. For x = -0.5 and y = -0.3 the method does converge.



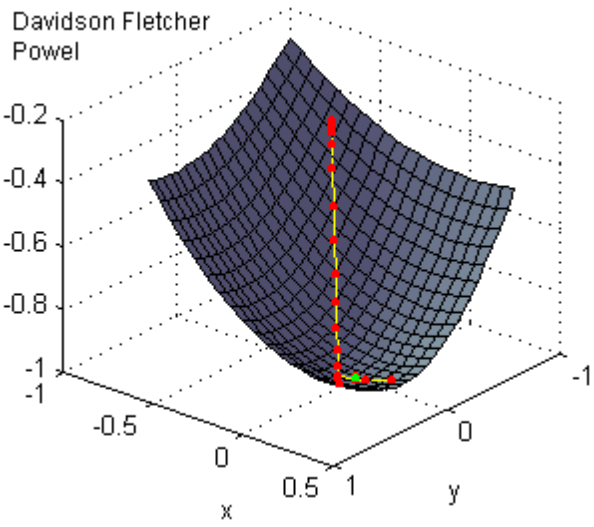
6. Polack Ribiere (gradient search)

The Polack Ribiere method not only uses the gradient of a function, but also the second order gradient to determine the search direction. The second order gradient is estimated based on previous search directions. The search direction is kept for each new step until a minimum has been found. Then a new search direction is determined and the process continues.



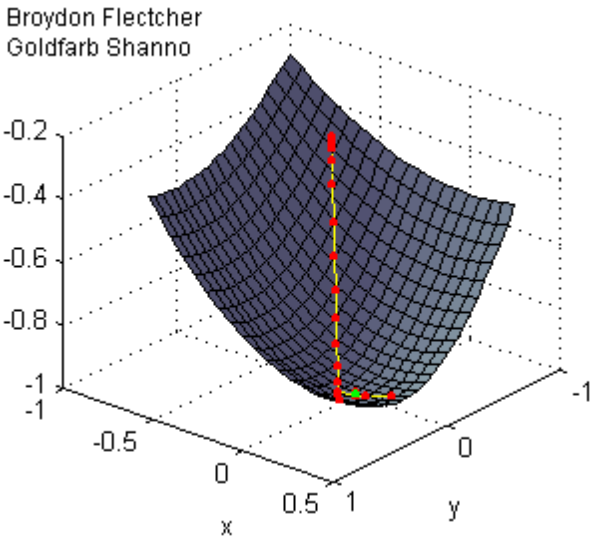
7. Davidson Fletcher Powel (gradient search)

The Davidson Fletcher Powel method not only uses the gradient of a function, but also the second order gradient to determine the search direction. The second order gradient is estimated based on previous search directions. The search direction is kept for each new step until a minimum has been found. Then a new search direction is determined and the process continues



8. Broydon Fletcher Goldfarb Shanno (gradient search)

The Broydon Fletcher Goldfarb Shanno method not only uses the gradient of a function, but also the second order gradient to determine the search direction. The second order gradient is estimated based on previous search directions. The search direction is kept for each new step until a minimum has been found. Then a new search direction is determined and the process continues.

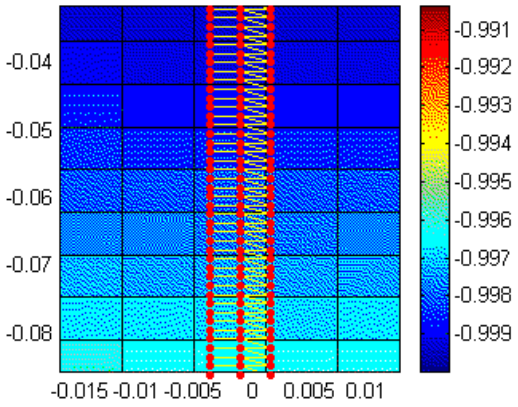


What method should be used?

There is no general answer to this question. Some remarks can however be made:

1. Gradient search methods (3 to 8), need more steps (note that each step means a simulation run) to determine the gradient. The Newton Raphson methods needs the most steps, because this method also needs additional steps for the determination of the second order gradient.

2. When the direction of the slope is not exactly the same as the search direction, direct search methods (1 and 2) may start to bounce, i.e. continuously change direction while making little progress. this is shown in the figure below:



Users are therefore advised to use methods 1 and 2 only for optimizations with one parameter, and use the methods 7 and 8 for optimizations with more parameters. Use the other methods for checking the results or educational purposes.

More information on these methods can be found in:

Bazaraa, M.S., Sherali, H.D., Shetty C.M. (1990), Nonlinear Programming, Theory and Algorithms, John Wiley & Sons Inc. New York, ISBN 0-471-59973-5.

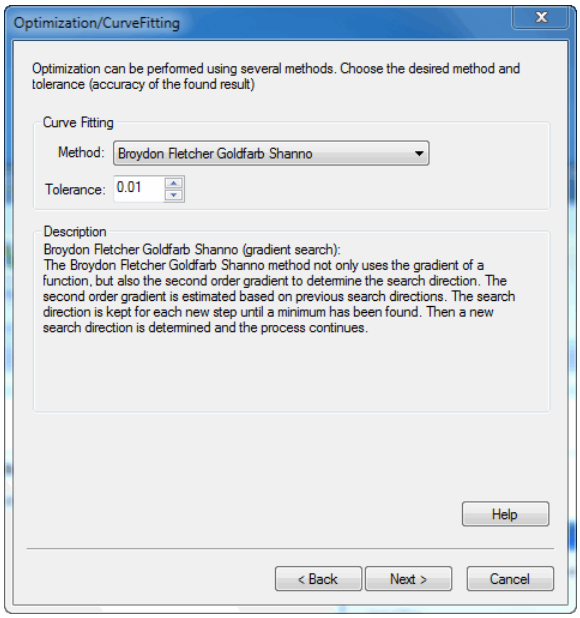
8.7.5 Curve Fitting

Using the Parameter Sweep option of the *Time Domain Toolbox*, you can fit your model to a given result using variation of parameters / initial values.

Optimization Method

1. From the **Tools** menu select the **Time Domain Toolbox** and then **Curve Fittting**.

A window opens asking you which optimization method should used for curve fitting:

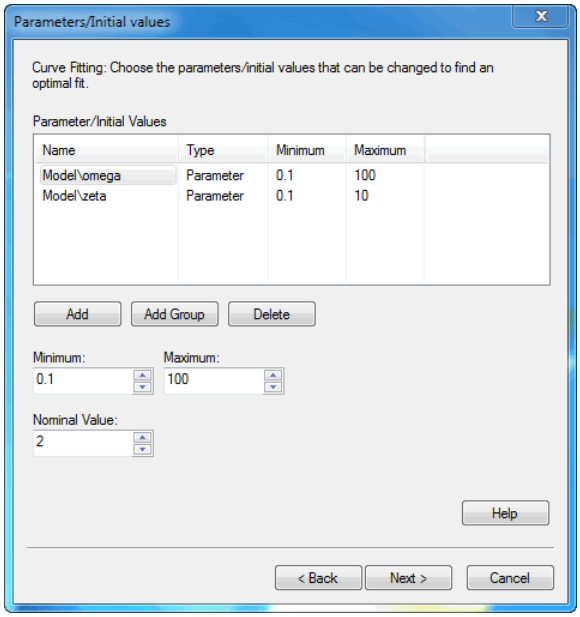


2. Select the Optimization method that should be used.

3. Select the Tolerance (accuracy of the found fit) that should be used and click **Next**.

Parameters / Initial Values

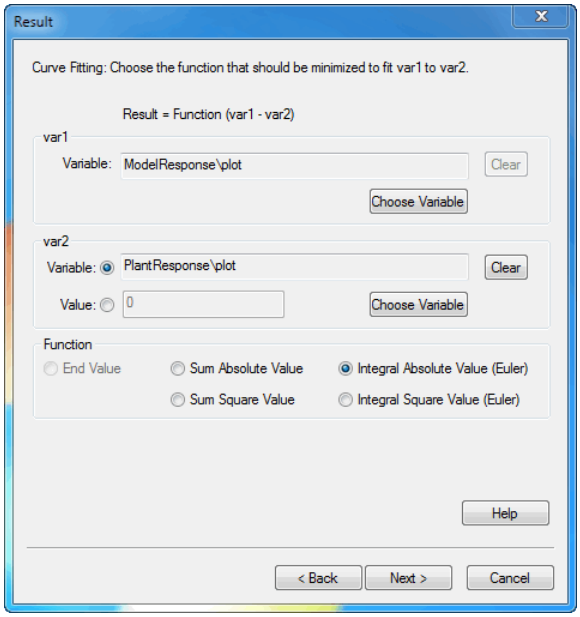
A window opens asking you which parameters / initial values should be varied during curve fitting:



4. Add the parameters / initial values of your choice:
5. Click the **Next** button.

Result

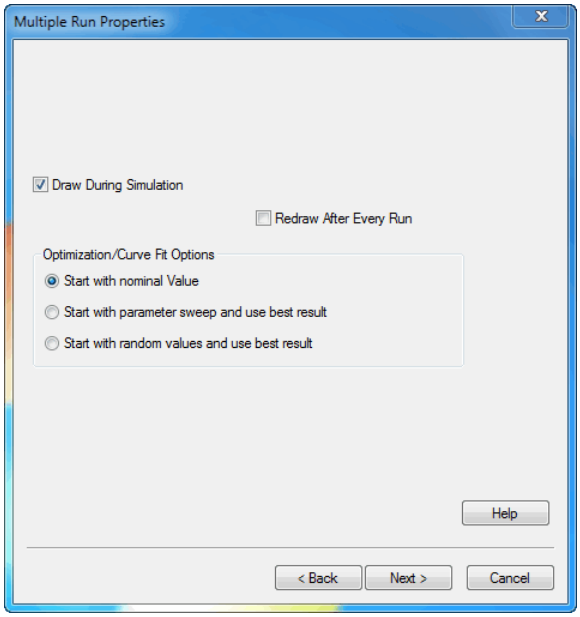
A window opens asking you which result should minimized to fit your model (variable 1) to a given result (variable 2):



6. Enter the result here using the various options of the window.
7. Click the **Next** button.

Run options

A window opens asking some simulation run options:



8. Select the desired run options and click the **Next** button.

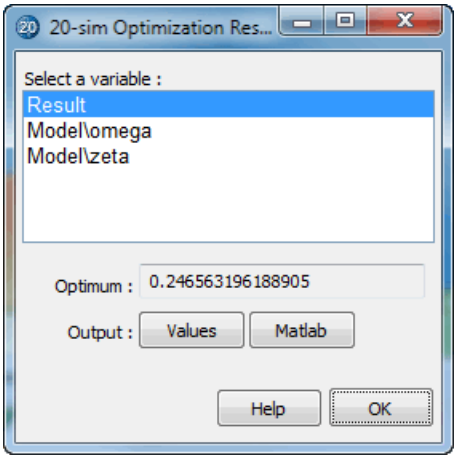
A window opens with a summary of the chosen options. If your are not satisfied you can use the Back button to go to a previous window and change settings.

9. Click the **Finish** button to close the **Multiple Run Wizard**.

10. From the **Simulation** menu click the **Multiple Run** command to perform the Curve Fitting.

Optimization Results

After simulation has been done, the Optimization Results window will open, showing the minimum value of the result and corresponding parameters:



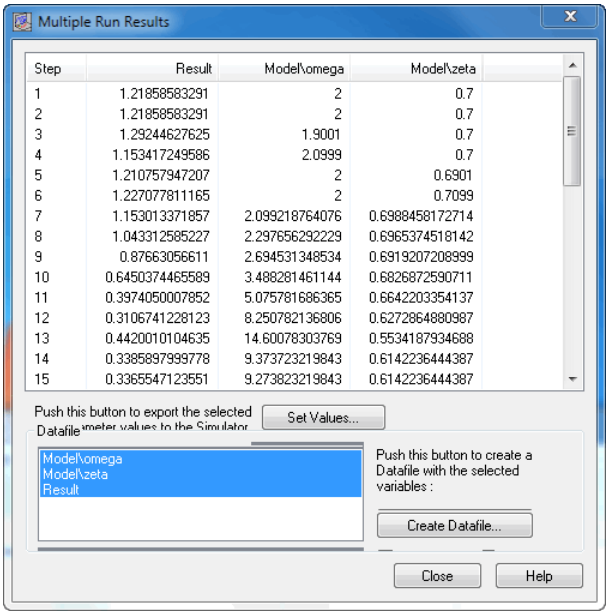
You can select the following items:

- *Values:* Choose this button to open the list that shows the numerical output of the curve fitting runs.
- *Matlab:* Export the results to Matlab.
- *OK*: Close the Optimization Results Window.

You can always re-open the **Multiple Run Results** window, by selecting the **Multiple Run Results** command of the **View** Menu.

11. By choosing the **Values** button, a new window is opened, showing the numerical results of the parameter sweep.

In the **Results list** of this window the subsequent runs are shown and the value of the chosen result:



You can select the following items:

- *Set Values*: Choose the values of the selected run, as new parameter values of your model.
- *Create Datafile*: Use this button to store the values of the selected variables on file. Use the control-key and shift-key to make multiple selections.

8.7.6 Sensitivity Analysis

Using the Tolerance Analysis option of the Multiple Run Wizard, you can analyze the effects of variation of parameters / initial values on the performance of your model. Three methods of Tolerance analysis are supported in the Multiple Run Wizard. One of these methods is Sensitivity analysis.

Sensitivity analysis starts with a simulation run with nominal parameter values. After the run a given result is monitored *(r)*. Then, one by one, each parameter *(pi)* is changed with a given percentage *(to pi + dpi)* and a simulation run is performed. After the simulation run the changed result is monitored *(r + dri)*. After all the runs, the results are displayed as sensitivities, where sensitivity is defined as the change in result divided by the change in parameter:

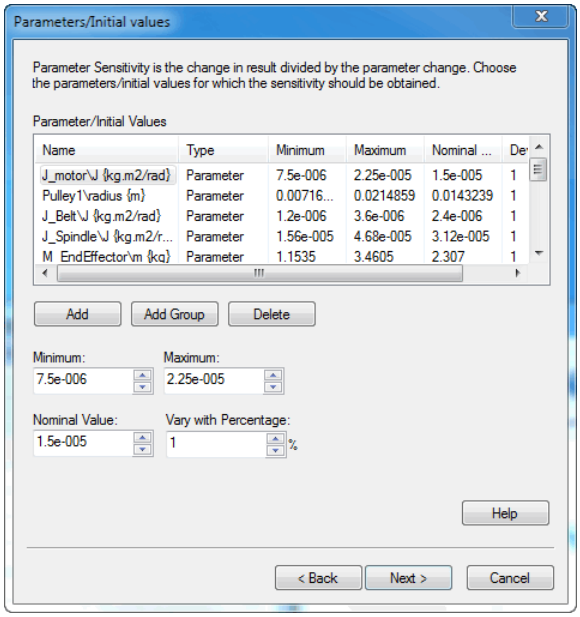
$$Si = dri / dpi$$

A large sensitivity means that the result is highly dependant of the parameter value. This can be used for optimization (change the parameter value) or design (change the design to make it less dependant of the parameter).

Parameters / Initial Values

1. From the **Tools** menu select the **Time Domain Toolbox** and then **Sensitivity Analysis**.

A window opens asking you which parameters / initial values should be changed for the Sensitivity analysis:

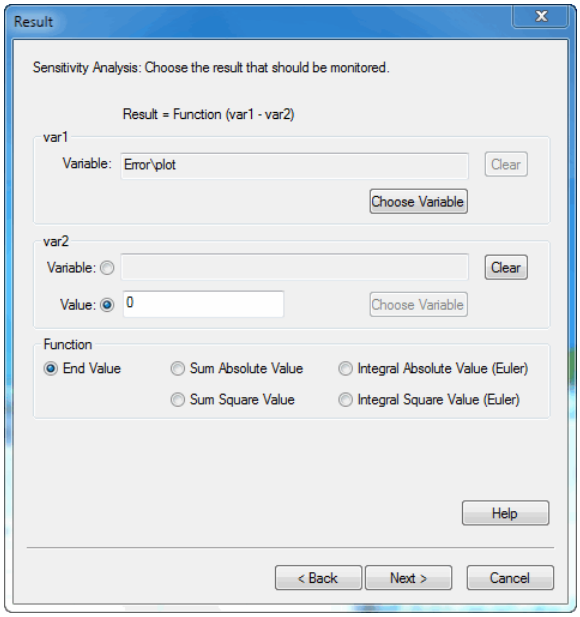


3. Add the parameters / initial values of your choice:

4. Click the **Next** button.

Result

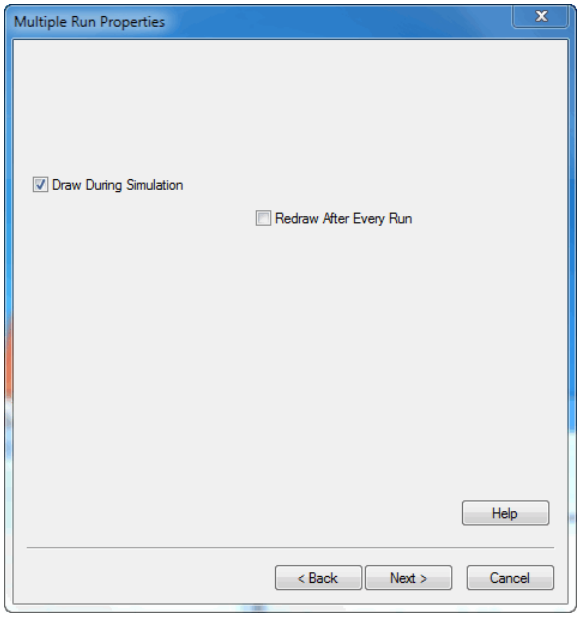
A window opens asking you which result should be monitored for the sensitivity:



5. Enter the result here using the various options of the window.
6. Click the **Next** button.

Run options

A window opens asking some simulation run options:



7. Select the desired run options and click the **Next** button.

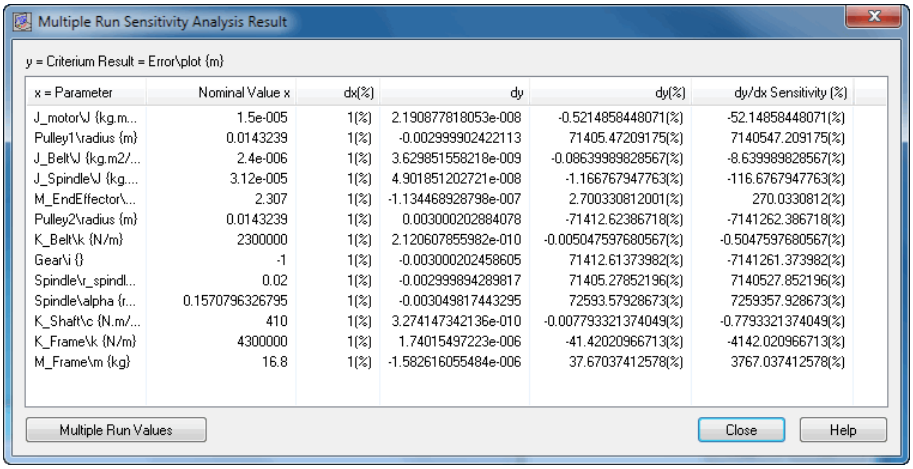
A window opens with a summary of the chosen options. If your are not satisfied you can use the **Back** button to go to a previous window and change settings.

8. Click the **Finish** button to close the Multiple Run Wizard.

9. From the **Simulation** menu click the **Multiple Run** command to perform the Sensitivity Analysis.

Sensitivity Analysis Results

After simulation has been done, a Sensitivity Analysis Results window will open, showing the sensitivities:



x = Parameter	Nominal Value x	dx(%)	dy	dy(%)	dy/dx Sensitivity (%)
J_motor\J {kg.m...	1.5e-005	1(%)	2.190877818053e-008	-0.5214858448071(%)	-52.14858448071(%)
Pulley1\radius {m}	0.0143239	1(%)	-0.002999902422113	71405.47209175(%)	7140547.209175(%)
J_Belt\J {kg.m2/...	2.4e-006	1(%)	3.629851558218e-009	-0.086399898628567(%)	-8.639989828567(%)
J_Spindle\J {kg....	3.12e-005	1(%)	4.901851202721e-008	-1.166767947763(%)	-116.6767947763(%)
M_EndEffector\...	2.307	1(%)	-1.134468928798e-007	2.700330812001(%)	270.0330812(%)
Pulley2\radius {m}	0.0143239	1(%)	0.003000202884078	-71412.62386718(%)	-7141262.386718(%)
K_Belt\k {N/m}	2300000	1(%)	2.120607855982e-010	-0.005047597680567(%)	-0.5047597680567(%)
Gear\i {}	-1	1(%)	-0.003000202458605	71412.61373982(%)	-7141261.373982(%)
Spindle\r_spindl...	0.02	1(%)	-0.002999894289817	71405.27852196(%)	7140527.852196(%)
Spindle\alpha {r...	0.1570796326795	1(%)	-0.003049817443295	72593.57928673(%)	7259357.928673(%)
K_Shaft\c {N.m/...	410	1(%)	3.274147342136e-010	-0.007793321374049(%)	-0.7793321374049(%)
K_Frame\k {N/m}	4300000	1(%)	1.74015497223e-006	-41.42020966713(%)	-4142.020966713(%)
M_Frame\m {kg}	16.8	1(%)	-1.582616055484e-006	37.67037412578(%)	3767.037412578(%)

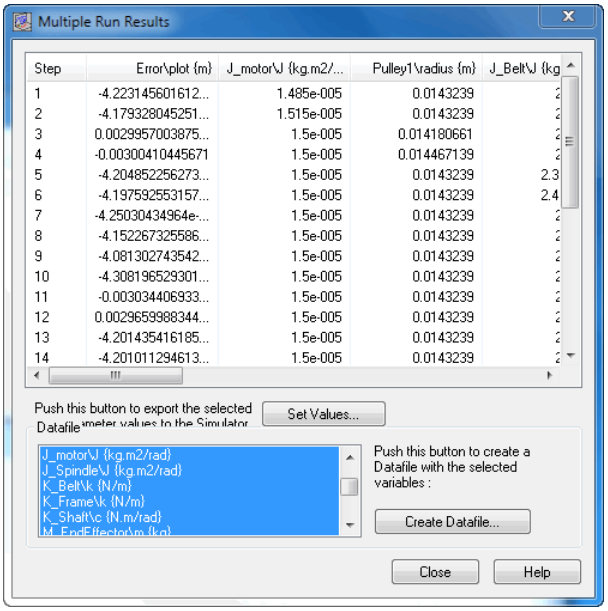
You can select the following items:

- *Multiple Run Values*: Choose this button to open the list that shows the output of the various runs performed during the sensitivity analysis.
- *Close*: Close the Sensitivity Analysis Results Window.

You can always re-open the *Multiple Run Results* window, by selecting the *Multiple Run Results* command of the View Menu.

10. By choosing the **Multiple Run Values** button, a new window is opened, showing output of the various runs performed during the sensitivity analysis:

In the *Results list* of this window the subsequent runs are shown and the corresponding sensitivities.



You can select the following items:

- *Set Values:* Choose the values of the selected run, as new parameter values of your model.
- *Create Datafile:* Use this button to store the values of the selected variables on file. Use the control-key and shift-key to make multiple selections.

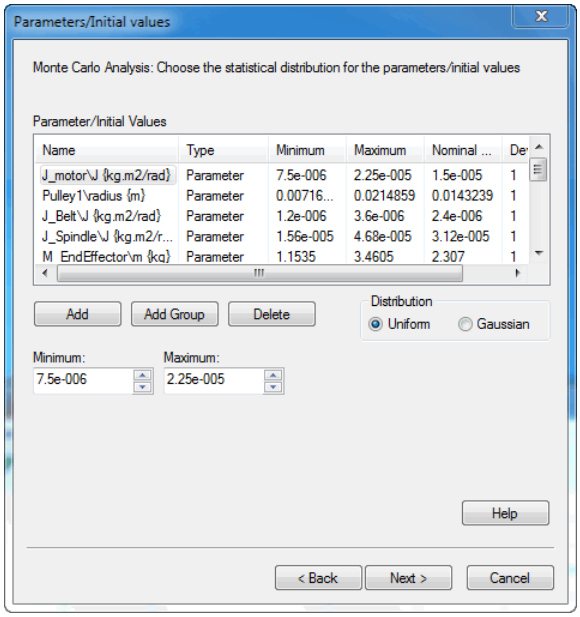
8.7.7 Monte Carlo Analysis

Using the Tolerance Analysis option of the Multiple Run Wizard, you can analyze the effects of variation of parameters / initial values on the performance of your model. Three methods of Tolerance analysis are supported in the Multiple Run Wizard. One of these methods is Monte Carlo Analysis. Using Monte Carlo analysis, you can perform a predefined number of simulation runs with variation of parameter values according to a predefined distribution function.

Parameters / Initial Values

1. From the **Tools** menu select the **Time Domain Toolbox** and then **Monte Carlo Analysis**.

A window opens asking you which parameters / initial values should be varied for the Monte Carlo analysis:

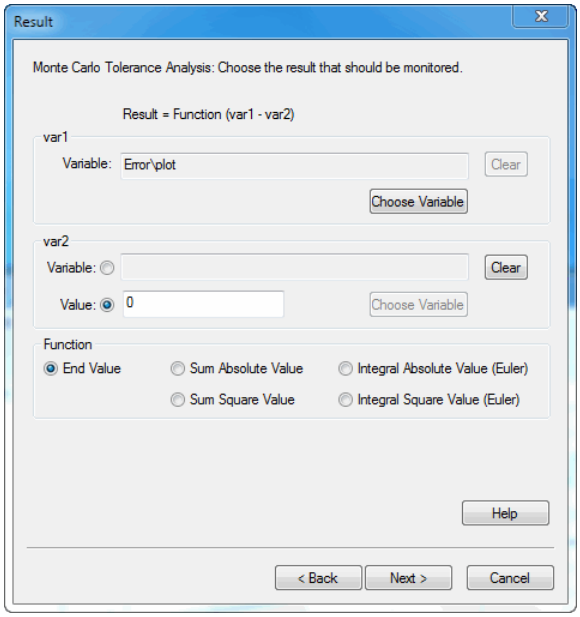


2. Add the parameters / initial values of your choice:

3. Click the **Next** button.

Result

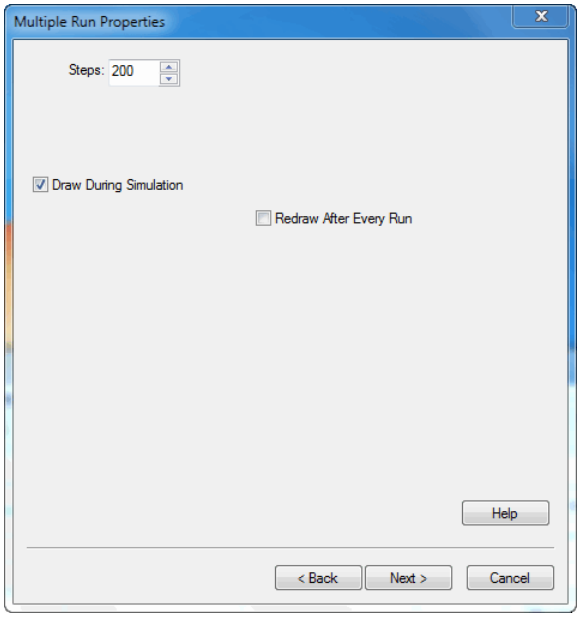
A window opens asking you which result should be monitored during the Monte Carlo Analysis:



4. Enter the result here using the various options of the window.
5. Click the **Next** button.

Run options

A window opens asking some simulation run options:



6. Select the desired run options and click the **Next** button.

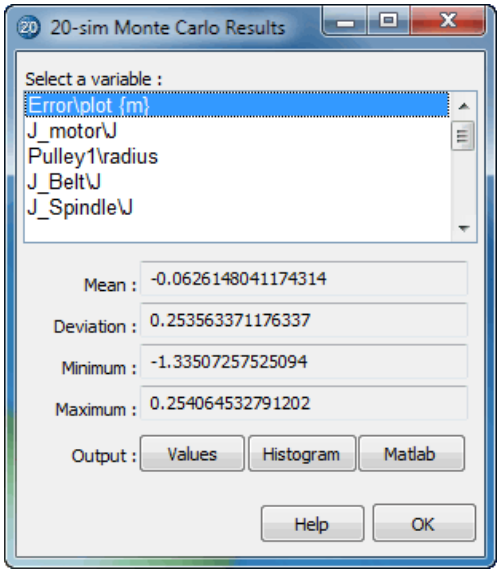
A window opens with a summary of the chosen options. If your are not satisfied you can use the **Back** button to go to a previous window and change settings.

7. Click the **Finish** button to close the Multiple Run Wizard.

8. From the **Simulation menu** click the **Multiple Run** command to perform the Monte Carlo Analysis.

Monte Carlo Analysis Results

After simulation has been done, a Monte Carlo Analysis Results window will open, showing the results:

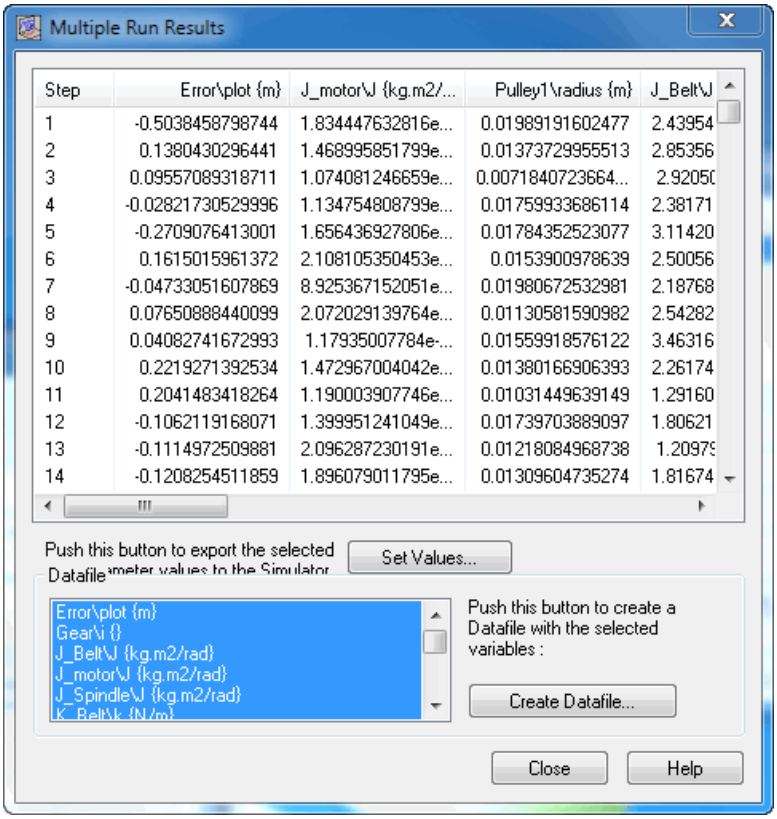


You can select the following items:

- *Values:* Choose this button to open the list that shows the output of the various runs performed during the Monte Carlo analysis.
- *Histogram:* Choose this button to open a histogram of the selected item.
- *Matlab*: Export the results to Matlab.
- *OK:* Close the Multiple Run Results Window.

You can always re-open the **Multiple Run Results** window, by selecting the **Multiple Run Results** command of the **View** Menu.

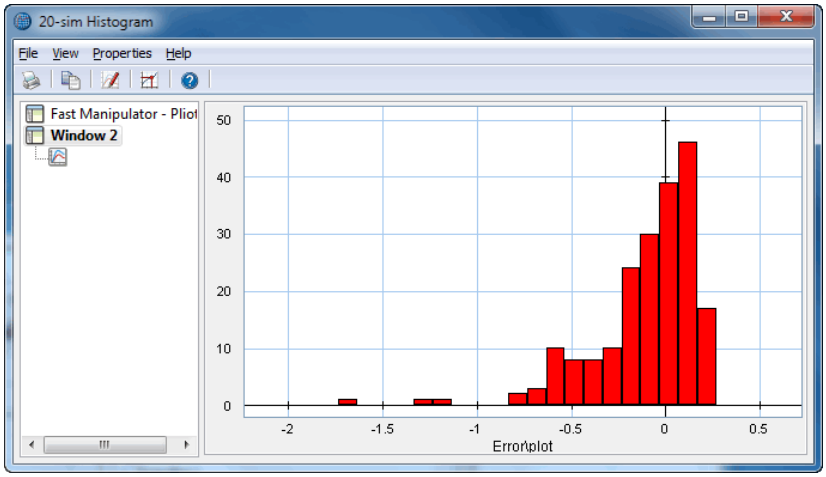
9. By choosing the **Multiple Run Values** button, a new window is opened, showing output of the various runs performed during the Monte Carlo analysis.



You can select the following items:

- *Set Values*: Choose the values of the selected run, as new parameter values of your model.
- *Create Datafile*: Use this button to store the values of the selected variables on file. Use the control-key and shift-key to make multiple selections.

By choosing the **Histogram** button, a new window is opened, showing distribution of the result variable:



8.7.8 Variation Analysis

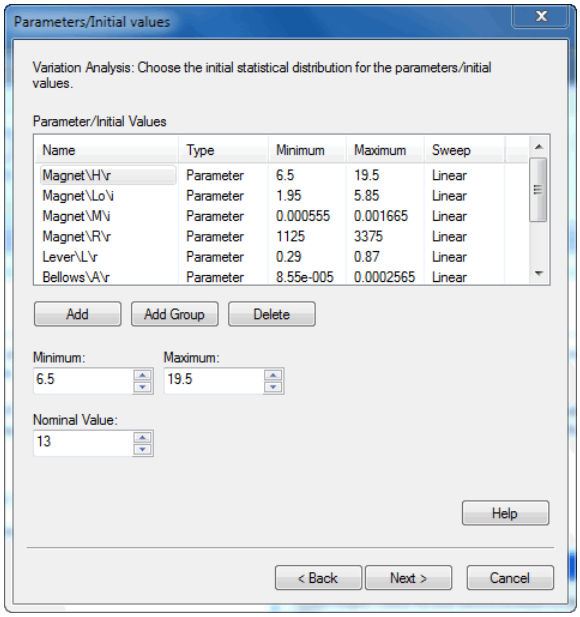
Using the Tolerance Analysis option of the Multiple Run Wizard, you can analyze the effects of variation of parameters / initial values on the performance of your model. Three methods of Tolerance analysis are supported in the Multiple Run Wizard. One of these methods is Variation Analysis.

Using Variation analysis, you can perform a predefined number of simulation runs with variation of parameter values according to a predefined distribution function. During the runs a given result is monitored. After the runs the statistical range of the result is presented. You can restrict the parameter distributions to get the result between desired bounds.

Parameters / Initial Values

1. From the **Tools** menu select the **Time Domain Toolbox** and then **Variation Analysis**.

A window opens asking you which parameters / initial values should be varied for the Variation analysis:

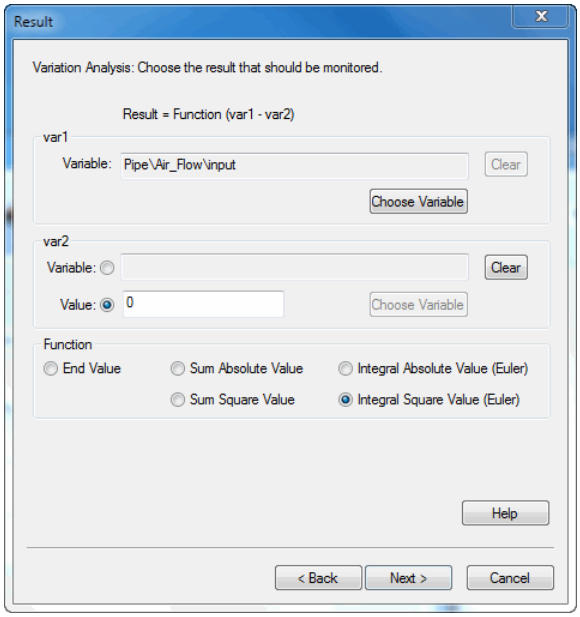


2. Add the parameters / initial values of your choice:

3. Click the **Next** button.

Result

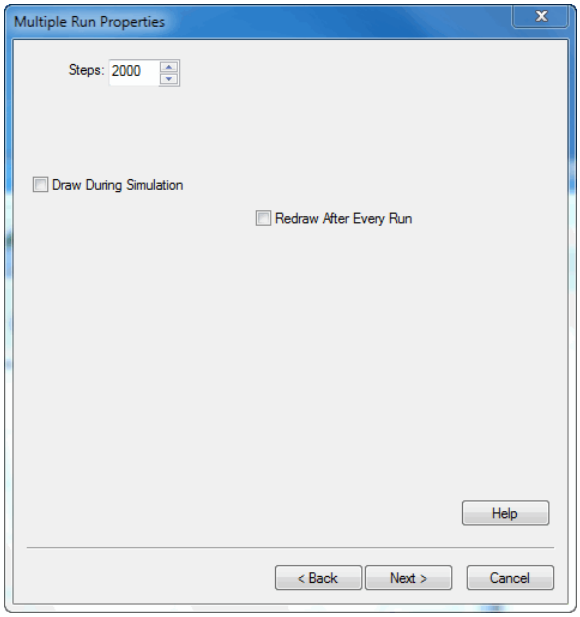
A window opens asking you which result should be monitored during the Variation Analysis:



4. Enter the result here using the various options of the window.
5. Click the **Next** button.

Run options

A window opens asking some simulation run options:



6. Select the desired run options and click the **Next** button.

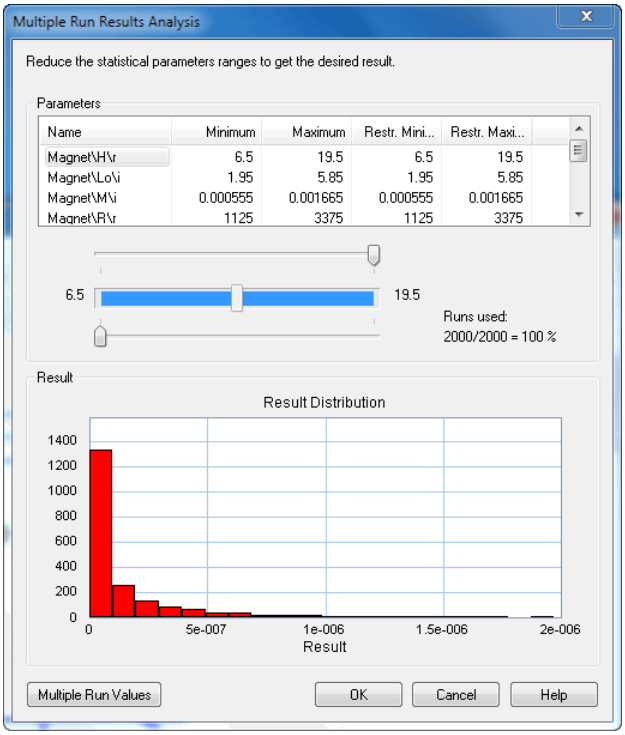
A window opens with a summary of the chosen options. If your are not satisfied you can use the **Back** button to go to a previous window and change settings.

7. Click the **Finish** button to close the **Multiple Run Wizard**.

8. From the **Simulation menu** click the **Multiple Run command** to perform the Variation Analysis.

Variation Analysis Results

After simulation has been done, a Variation Analysis Results window will open, showing the results. A histogram shows the distribution of the result. Using the slider bar you can see how restraining the distribution of the parameters effects the distribution of the result.

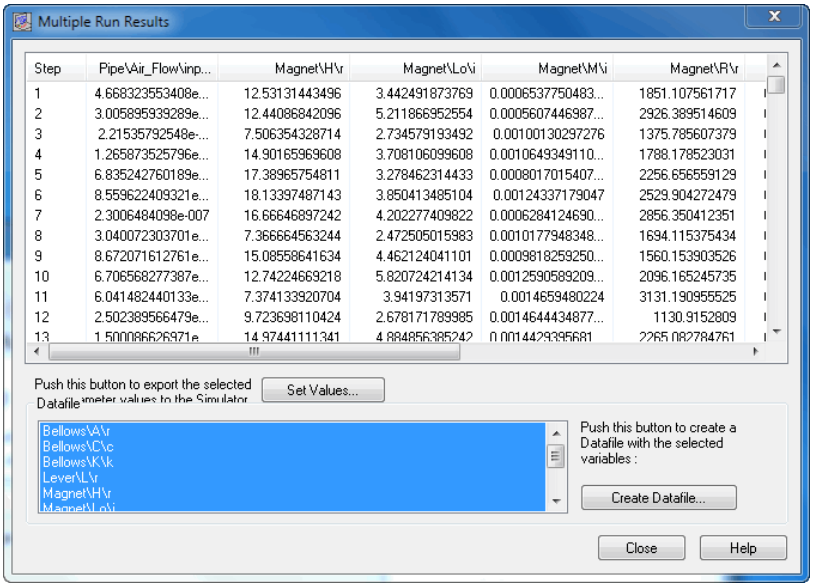


You can select the following items:

- *Slider Bar*: Select a parameter and restrict its distribution with the slider bar. You will see the histogram showing the result change.
- *Multiple Run Values*: Choose this button to open the list that shows the output of the various runs performed during the Variation Analysis.
- *OK:* Close the Multiple Run Results Window.

You can always re-open the Multiple Run Results window, by selecting the **Multiple Run Results** command of the **View** Menu.

9. By choosing the **Multiple Run Values** button, a new window is opened, showing output of the various runs performed during the Variation analysis:



You can select the following items:

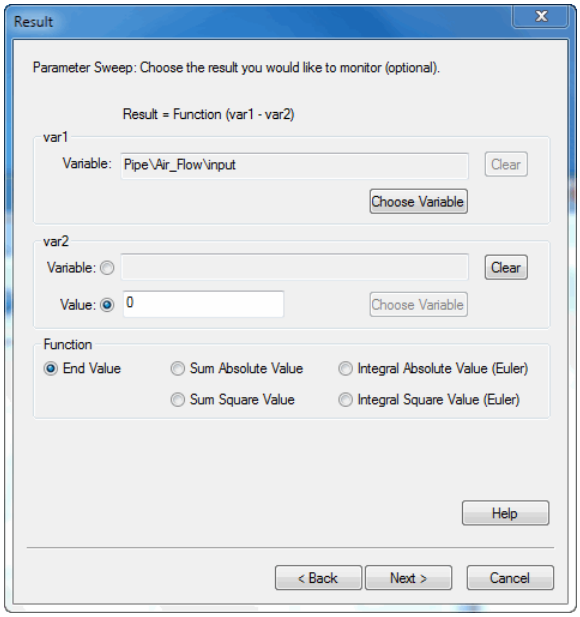
- *Set Values:* Choose the values of the selected run, as new parameter values of your model.
- *Create Datafile*: Use this button to store the values of the selected variables on file. Use the control-key and shift-key to make multiple selections.

8.7.9 External DLL

You can analyze the effects of variation of parameters / initial values using your own method with the use of an external DLL-function. Ask Controllab Products for details.

8.7.10 Cost Function

In the Multiple Run Wizard, the Result window allows you to define a result (the *Cost Function*) that will be used during simulations (Optimization, Curve Fitting) and displayed after the simulations runs (Parameter sweep, Optimization, Curve Fitting, Sensitivity, Monte Carlo, Variation Analysis, External DLL).



The result can be a function of one or two variables. Instead of the second variable an (offset) value can be entered.

Items

- *Choose Variable:* Select this button to select a model variable
- *Clear:* Use this button the remove a chosen model variable.
- Function:
  - *End value:* The value at the end of the simulation run.
  - *Sum Absolute Value*: The sum of all absolute values during a simulation run.
  - *Sum Square Value*: The sum of all values squared during a simulation run.
  - *Integral Absolute Value:* The integral of all absolute values during a simulation run (uses Euler integration).
  - *Integral Square Value:* The integral of all values squared during a simulation run (uses Euler integration).

User Defined Cost Function

You can easily use your own *Cost Function* for the result variable:

1. Create your own cost function in the model.
2. Use the **Choose** button to select as var1 the output of your function.

3. Set *var2* to a zero value.
4. Choose **End Value** to prevent further operations on your function.

8.8 Scripting Toolbox

8.8.1 Introduction

20-sim scripting allows you to run tasks in 20-sim automatically using specialized scripting functions. With these functions you can open models, run simulations, change parameters, store results and much more. Scripting support was introduced in 20-sim 4.4.

20-sim run automatically by a script in Octave or Matlab.

These scripting functions are not very useful stand-alone. But when they are used within a numerical computation environments, you can use these environments to combine scripts, run sequences of scripts, run numerical routines on the outcomes and much more! XML-RPC is used to communicate the scripting functions and data to and from 20-sim. This allows scripts to be run from every environment that supports XML-RPC. 20-sim scripting is currently available in the numerical computation environments Octave and Matlab.

In this chapter you will learn how to run basic scripts and make scripts on your own. We will assume that you will use Octave as the scripting environment, but you can also use Matlab. The workflow for both tools is equivalent.

The next sections explain:

- **Installation for scripting**:
  - *20-sim:* enabling the XML-RPC scripting interface in 20-sim
  - *Octave:* installing Octave as scripting environment
  - *Matlab:* installing Matlab as scripting environment
- **Prepare Scripting Folder:** extract the 20-sim scripting functions and documentation to your work directory.
- **Basic Script:** run your first script and see how a basic script is made.
- **Advanced Scripts:** see how you can expand the basic script to perform more advanced tasks.
- **Writing your own Scripts:** How to write your own scripts.

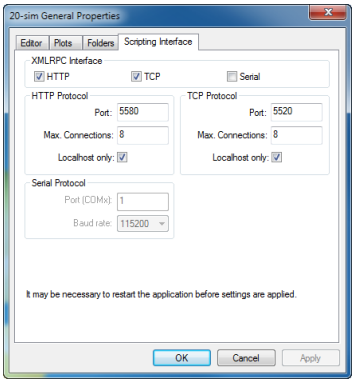
Note: Scripting is not supported in the 20-sim Viewer/Demonstration Version!

8.8.2 Installation for Scripting: 20-sim

20-sim uses XML-RPC as a protocol to communicate scripting functions with external packages. By default the XML-RPC interface is turned on only for your local computer.

To enable/disable and configure the 20-sim scripting support:

1. Open **20-sim**.
2. Go to **Tools/Options** and select the **Scripting Interface** tab.



Scripting Interface settings tab.

3. To enable the 20-sim scripting support, make sure that the **HTTP** and **TCP** checkboxes under *XMLRPC Interface* are enabled.

By default, 20-sim will only accept scripting connections from your local computer (**Localhost only** option is enabled).

Your firewall may generate a warning message and ask you to allow network communication for 20-sim.

4. Set the **firewall** to **allow communication**.

8.8.3 Installation for Scripting: Octave

What is Octave?

GNU Octave is a high-level language, primarily intended for numerical computations. The package is open source and can be freely distibuted. GNU Octave offers functionality similar to Matlab users. If you have experience with Matlab, running Octave will be very easy. Users with no experience with Octave nor Matlab are advised to read a proper introduction to GNU Octave first. You will find lot of pages and videos on the Internet.

Installation

The versions Octave-3.6.2, Octave-3.6.4 and Octave-3.8.2 have been tested with 20-sim scripting at the time of release. Work is in progress to provide support for the latest release Octave-4.0.0.

1. Go to: http://www.20sim.com/product/octave.html and follow the instructions.

If this webpage is unavailable, you can follow the instructions below:

First choose the Octave version you wish to install.

For version **3.6.x**:

1. Go to the Octave download site (http://sourceforge.net/projects/octave/.).
2. Click on the **Files** tab and click on **Octave Windows Binaries**.
3. Select the **Octave 3.6.4 for Windows MinGW installer**.
4. Now you can **download** the files **Octave3.6.4_gcc4.6.2_yyyyxxxx.7z** (Octave Installation) and **Octave3.6.4_gcc4.6.2_pkgs_yyyyxxxx.7z** (Octaveforge Packages).
5. Create an installation directory which doesn't have space chars (i.e. C:\Octave).
6. **Unzip** the file **Octave3.6.4_gcc4.6.2_yyyyxxxx.7z** and **copy** it to the installation directory.
7. Copy the **shortcut link** `C:\Octave\Octave3.6.4_gcc4.6.2.lnk` to your **desktop**. This is a shortcut to start Octave.exe.

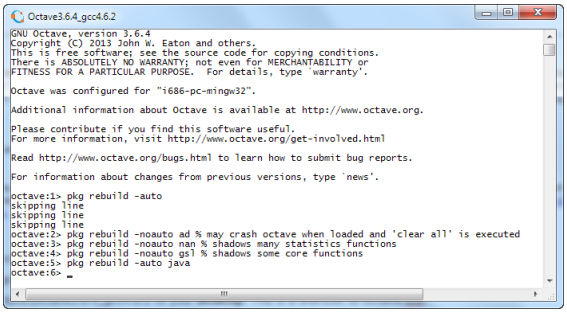
Note: Unzipping can be done with programs like 7-zip (http://www.7-zip.org/)
Note: There is a bug with Windows 8 and running Octave. In order to use Octave start Octave with `octave.exe -i --line-editing`. See the Octave wiki webpage for more information.

8. **Unzip** the file **Octave3.6.4_gcc4.6.2_pkgs_yyyyxxxx.7z** and **copy** it to the installation directory.
9. **Launch Octave** (e.g. the link to Octave.exe).

10. **Execute** the following five rebuild commands from the Octave console (e.g. re-type every line followed by ENTER):

pkg rebuild -auto

```
pkg rebuild -noauto ad
pkg rebuild -noauto nan % shadows many statistics functions
pkg rebuild -noauto gsl % shadows some core functions
pkg rebuild -auto java
```



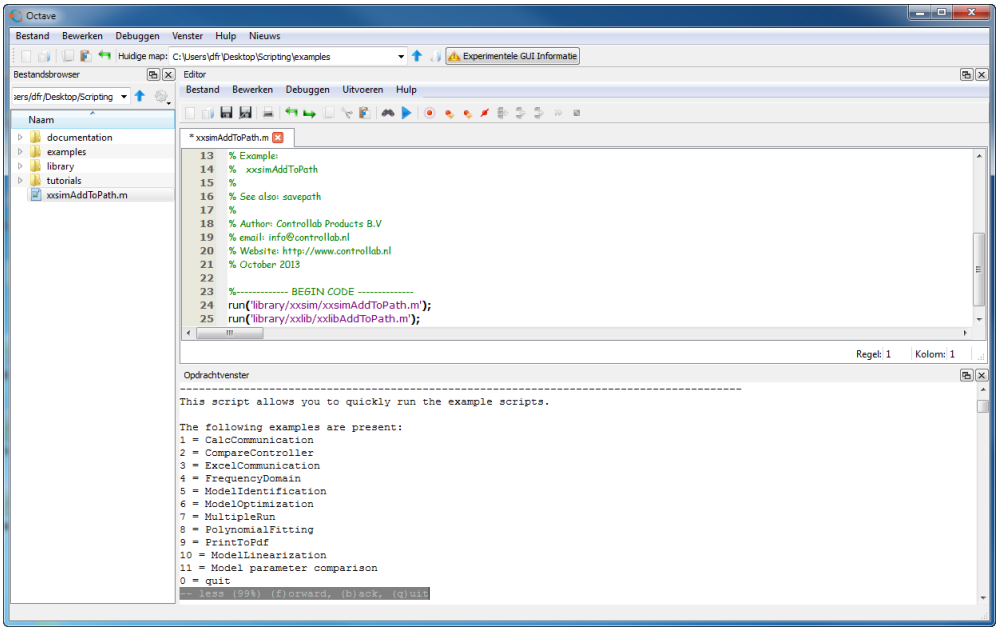
The Octave command window.

11. Close and **restart Octave**.

For version **3.8.x**:

1. Go to: http://www.20sim.com/downloads/files/ThirdParty/octave-3.8.2-2-installer.exe
2. **Run** the installer and follow the wizard.
3. **Launch Octave** using the script: C:\Octave\Octave-3.8.2\octave.bat
4. **Execute** the following commands to install packages io, control and signal:

```
pkg install -forge io
pkg install -forge control
pkg install -forge signal
```



The Octave GUI.

Your Octave installation is now ready to use.

8.8.4 Installation for Scripting: Matlab

What is Matlab?

Matlab is a high-level language, primarily intended for numerical computations. The package is commercially distributed by the Mathworks. If you don't have the resources to purchase Matlab, you can run use Octave, which offers similar functionality.

Versions

Matlab R2011, R2012 and R2013 have been tested with 20-sim scripting but older versions may also work fine.

Installation

See the Matlab documentation from the Mathworks for information on installing Matlab. No special (additional) installation is needed to use 20-sim scripting from Matlab.

Note: Scripting in Matlab is similar to Octave. You can type exactly the same commands as given for Octave in the next sections.

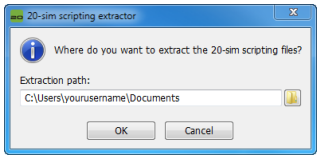
8.8.5 Prepare Scripting Folder

20-sim comes with a *Scripting Folder* that contains documentation of all scripting functions, the function library and example scripts. You have to install this folder to use scripting.

Installation

1. Open the **Install Scripting** program from the Windows *Start* menu (located under ***20-sim 4.5***)
   - or -
   Go to the folder where 20-sim is installed (e.g. ***C:\Program Files\20-sim 4.5\Scripting*** or ***C:\Program Files (x86)\20-sim 4.5\Scripting***) and open **20simScripting.exe**

2. This will open a dialog where you can choose where to extract the 20-sim scripting files. **Change** the path to a local working folder of your choice (for example: ***C:\Users\yourusername\Documents\20simscripting***)



20-sim Scripts extraction

Note: To write/modify scripts, the scripting folder should be accessible and writable by the user. Do not install the scripting folder on *C:\Program Files (x86)* or *C:\Program Files.*

For the remainder of this chapter, we use the name *scripting working folder* when we refer to the folder where you just extracted the 20-sim scripting files.

Contents

Your newly created *scripting working folder* contains a number of subfolders:

1. **documentation**: This folder contains the scripting API documentation: a list of supported functions and their syntax. You can browse the documentation by opening the *index.html* file in your web browser.

   Note: the API documentation is also accessible from the Windows *Start* menu under *20-sim 4.5\Scripting API documentation*

2. **library**: This folder contains the core scripting functions.

3. **tutorials**: This folder contains basis scripts with a step by step explanation. You can use these scripts as a base for your own scripts.

4. **examples**: This folder contains some more advanced scripts.

8.8.6 Basic Script

When the scripting files are properly installed in your *scripting working folder*, we can run some tutorial scripts. Tutorial scripts are a step by step demonstrations of usage of the scripting functionality in 20-sim. These scripts are found in the *tutorials* subfolder of the *scripting working folder*. We will start with a basic script that opens and runs a 20-sim model.

1. Open **20-sim**.
2. Open **Octave** (or **Matlab**).
3. In Octave/Matlab, change the local working directory to the tutorial folder inside your *scripting working folder* . E.g . type:

```
cd 'C:\Users\yourusername\Documents\20simscripting\tutorials'
```

Note: 20-sim should be open before running the script!

4. In Octave/Matlab, execute the following command (e.g. type the following **case sensitive** command followed by ENTER):

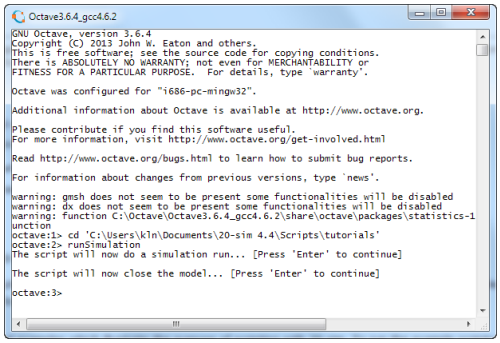
```
runSimulation
```

Note: Octave may give a cryptic "*undefined near line x column 1*" message, if you type the command as `runsimulation` instead of `runSimulation`!

5. Now Octave / Matlab will give a message and ask you to press **ENTER** to continue.

The model *ControlledSystem.emx* is loaded into 20-sim and simulated.

6. Again Octave/Matlab will give a message and ask you to press **ENTER** to continue.



Now the simulation and model will be unloaded.

Inspecting the script

To see how the script is made, you can inspect it with a text editor.

1. Open a file browser and go to the tutorials folder (e.g. C:\Users\yourusername \Documents\20simscripting\tutorials)

2. Open the file *runSimulation.m* with a text editor like **Notepad**.

Core Functions

The core functions of the *runSimulation* script are:

- **addpath**: The script starts with the command addpath('../library/xxsim'); This will enable Octave / Matlab to use the 20-sim scripting functions that are stored in the library subfolder of your scripting working folder.
- **xxsimConnect**: This command opens a connection to 20-sim.
- **xxSimOpenModel**: This command opens a model in 20-sim by giving the filename including the full path.
- **xxsimProcessModel**: This command will process the model.
- **xxsimRun**: This command will run a simulation.
- **xxsimCloseModel**: This command will remove the simulation model from 20-sim.

These functions are the basis of scripting in 20-sim and will be present in this order in most scripts. Therefore you can use the script *runSimulation.m* as a template for any new script that you create.

8.8.7 Advanced Scripts

Now that we have seen the core functions of a script we will run and check some more advanced scripts.

1. Open **20-sim**.
2. Open **Octave** (or **Matlab**).
3. In Octave/Matlab, change the local working directory to the tutorial folder inside your scripting working folder. E.g . type:

```
cd 'C:\Users\yourusername\Documents\20simscripting\tutorials'
```

Set Parameter Values

4. In Octave/Matlab, execute the following script (e.g. type the command followed by ENTER):

```
SetParameterAndRun
```

This script will open the model *ControlledSystem.emx* and run a simulation. Then a model parameter is changed and a second simulation run is performed. As explained in the previous topic, you can inspect the script in a text editor.

Compared to the basic script you will find a new function:

- **xxsimSetParameters**: This function is used to set the parameter in the model with the new value.

Multiple Runs

5. In Octave/Matlab, execute the following script (e.g. type the command followed by ENTER):

```
multipleRun
```

This script will open the model *ControlledSystem.emx* and run a simulation multiple times while changing a parameter. Then a model parameter is changed and a second simulation run is performed.

Read Parameter Values

6. In Octave/Matlab, execute the following script (e.g. type the command followed by ENTER):

```
readAndSetParameters
```

This script will open the model *ControlledSystem.emx* and run a simulation. Then a model parameter is read from file and changed accordingly in the model, followed by a second simulation run.

You will find these new functions:

- **addpath**: An additional path is given (../library/xxlib) to allow addtional (user defined) functions.
- **xxlibReadCsv**: This function is used to read a parameter name and value from a spreadsheet file.

Store Simulation Results

7. In Octave/Matlab, execute the following script (e.g. type the command followed by ENTER):

```
modelVerification
```

This script will open the model *ControlledSystem.emx* and run a simulation. After the run the simulation results are stored and plotted in Octave/Matlab. You will find these new functions:

- **xxsimSetLogVariables**: Define which variables are going to be logged during the simulation run.
- **xxsimGetLogVariables**: Export the logged variables after the simulation run to Octave/Matlab.

Examples

8. In Octave/Matlab, change the local working directory to the tutorial folder. E.g. type:

```
cd 'C:\Users\yourusername\Documents\20simscripting\examples'
```

here you can find more example scripts.

8.8.8 Writing your own Scripts

Writing your own 20-sim scripts is easy if you follow the following guidelines:

Location

Create your own subfolder inside your scripting working folder. This allows you to update the 20-sim scripting files when new versions of 20-sim are released.

Template

Use the basic script *runSimulation* as a template. It contains the basic functions that should be present in any script. Change the paths and filenames to ensure that the script runs your model.

Add Functions

Add functions to your script file until the examples and help.

Help

You can find help on scripting functions in two ways:

1. Using the help file: in the scripting folder open the file *documentation\index.html*
2. Using Octave/Matlab: In Octave/Matlab you can type *help <function name>*. Then information on this function will be displayed. E.g:

```
help xxsimList
```

9 Library

9.1 Bond Graph

9.1.1 Bond Graph Models

Bond graphs are a network-like description of physical systems in terms of ideal physical processes. With the bond graph method we split up the system characteristics into an (imaginary) set of separate elements. Each element describes an idealized physical process. To facilitate drawing of bond graphs, the common elements are denoted by special symbols. This library contains all kind of bond graph elements. If you want to know more about modeling with bond graphs, please have a look at the modeling tutorial.

The standard library contains single dimensional bond graph elements. The 2d Library contains elements with multi-bonds of size 2 and the 3d library contains elements with multi-bonds of size 3.

9.1.2 C

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model represents a power continuous storage element. The element has a preferred effort out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred flow out causality. The constitutive equations then contain a derivation.

effort out causality (preferred):

```
state = int(p.f) + state(0);
p.e = state/c;
output = state;
```

flow out causality:

```
state = c*p.e;
p.f = d state / dt;
output = state;
```

Interface

Ports	Description
p	Input port of the storage element.

Causality

preferred effort out	A flow out causality results in a derivative constitutive equation.

Outputs

output	The output signal is equal to the state.

Parameters

c	The storage element constant.

Initial Values

state(0)	The initial value of the storage element.

9.1.3 CC

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model represents a two port C-element. Normally the equation of a two port C-element is written in vector notation as:

*p.e = (1/C)*p.f*

When the C matrix is singular simulation will not be possible. Therefore the element is described in a safer notation as:

*p.e = A*p.f , with A = (1/C);*

Both ports have a preferred effort out causality:

```
state1 = int (p1.f);
state2 = int (p2.f);
p1.e = a11*state1 + a12*state2;
p2.e = a21*state1 + a22*state2;
```

Interface

Ports	**Description**
p1,p2	Input ports of the storage element.

Causality

preferred effort out p1 preferred effort out p2	A flow out causality results in a derivative constitutive equation.

Outputs

state1
state2

Parameters

a11, a12, a21, a22	The storage element constants.

Initial Values

state1_initial state2_initial	The initial values of the storage element.

9.1.4 EffortSensor

Library

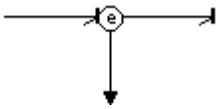
System , Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind**: Bond Graphs.

Description

This model can be inserted in any bond to yield the effort of that bond as an output signal.



To ensure success, use drag and drop to place the sensor exactly on top in the middle of the bond. As a result the model will be automatically connected.

Interface

Ports

p1, p2	Input and output port of the effort sensor.

Outputs

effort	Effort of the bond.

9.1.5 EnergySensor

Library

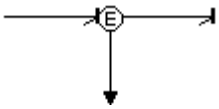
System , Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind:** Bond Graphs.

Description

This model can be inserted in any bond to yield the energy that is carried through that bond as an output signal.



To ensure success, use drag and drop to place the sensor exactly on top in the middle of the bond. As a result the model will be automatically connected.

Interface

Ports

p1, p2	Input and output port of the energy sensor.

Outputs

E	Energy carried through the bond.

9.1.6 FlowSensor

Library

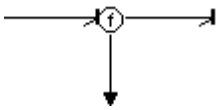
System , Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind**: Bond Graphs.

Description

This model can be inserted in any bond to yield the flow of that bond as an output signal.



To ensure success, use drag and drop to place the sensor exactly on top in the middle of the bond. As a result the model will be automatically connected.

Interface

Ports

p1, p2	Input and output port of the flow sensor.

Outputs

flow	Flow of the bond.

9.1.7 GY

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal gyrator, a power continuous relation between the effort of one port and the flow of the other port and vice-versa. The model can have both ports with an effort out causality or both ports with a flow out causality:

effort out causality:

```
p1.e = r * p2.f;
p2.e = r * p1.f;
```

flow out causality:

```
p1.f = 1/r * p2.e;
p2.f = 1/r * p1.e;
```

Interface

Ports	**Description**
p1, p2	Input and output port of the gyrator.
Causality	
p1 equal p2	The causality of both ports must be equal.
Parameters	
r	Gyration ratio.

9.1.8 I

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model represents a power continuous storage element. The element has a preferred flow out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred effort out causality. The constitutive equations then contain a derivation.

flow out causality (preferred):

```
state = int(p.e) + state(0);
p.f = state/i;
output = state;
```

effort out causality:

```
state = i*p.f;
p.e = d state / dt;
output = state;
```

Interface

Ports	**Description**
p	Input port of the storage element.
Causality	
preferred flow out	An effort out causality results in a derivative constitutive equation.
Outputs	
output	The output signal is equal to the state.
Parameters	
i	The storage element constant.
Initial Values	
state(0)	The initial value of the storage element.

9.1.9 IC

Library

Bond Graph

Use

Domains: Continuous. **Size:** 1-D. **Kind:** Block Diagrams, Bond Graphs.

Description

This model is a combination of an I storage element and a C storage element. Consequently the constitutive equation must be written as a matrix-vector multiplication. The constitutive equation of this element is given below:

```
state1 = int(p1.f);
state2 = int(p2.e);
p1.e = state1*a11 + state2*a21;
p2.f = state1*a21 + state2*a22;
output2 = state1;
output2 = state2;
```

The Maxwell reciprocity demands a12 = a21. Therefore only 3 matrix parameters are necessary.

Interface

Ports	**Description**
p1,p2	Input ports of the storage element.
Causality	
fixed effort out p1	
fixed flow out p2	
Outputs	
output1, output2	The output signals are equal to the states.
Parameters	
a11,a21,a22	The storage element constants.
Initial Values	
state1(0), state2(0)	The initial values of the storage element.

9.1.10 MGY

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal modulated gyrator, a power continuous relation between the effort of one port and the flow of the other port and vice-versa. The gyration ratio can be set to a certain (fluctuating) value, given by an input signal. The model can have both ports with an effort causality or both ports with a flow causality:

effort out causality:

```
p1.e = input * p2.f;
p2.e = input * p1.f;
```

flow out causality:

```
p1.f = 1/input * p2.e;
p2.f = 1/input * p1.e;
```

Interface

Ports	Description
p1, p2	Input and output port of gyrator.
Causality	
p1 equal p2	The causality of both ports must be equal.
Inputs	
input	Modulated gyration ratio.

9.1.11 MR

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents a linear friction/resistor equation. It can have an effort as well as a flow causality. In that case the constitutive equation, as shown below, is simply inverted. The friction/resistor parameter can be set to a (fluctuating) value, given by an input signal.

effort out causality:

*p.e = input*p.f*

flow out causality:

p.f = p.e/input

Interface

Ports	Description
p	Input port of the R-element.

Causality

indifferent

Inputs

input	The (modulated) friction/resistor parameter.

9.1.12 MSe

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal modulated effort source. The effort can be set to a (fluctuating) value given by an input signal. The flow is indifferent.

p.e = input;

Interface

Ports	Description
p	Output port of the effort source.

Causality

fixed effort out

Inputs

input	Modulation signal.

9.1.13 MSf

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal modulated flow source. The flow can be set to a (fluctuating) value given by an input signal. The effort is indifferent.

```
p.f = input;
```

Interface

Ports	Description
p	Output port of the flow source.

Causality

fixed flow out

Inputs

input	Modulation signal.

9.1.14 MTF

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal modulated transformer. The model represents a power continuous relation between the efforts and flows of both its ports. The transform ratio can be set to a certain (fluctuating) value, given by an input signal. The causality is always mixed: one port has an effort out causality while the other has a

flow out causality:

```
p1.e = input * p2.e;
p2.f = input * p1.f;
```

or:

*p2.e = 1/input * p1.e;*
*p1.f = 1/input * p2.f;*

Interface

Ports	**Description**
p1, p2	Input and output port of the transformer.

Causality

p1 notequal p2

Inputs

input	Modulated transform ratio.

9.1.15 OneJunction

Library

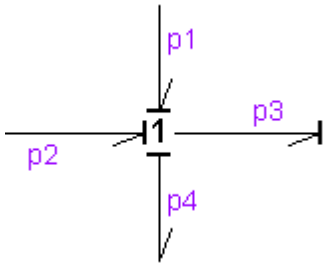
Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind**: Bond Graphs.

Description

This junction represents a power continuous (no energy storage, dissipation or generation) connection of elements. The sum of the efforts on all ports is zero and the flows on all ports are equal. The constitutive equations are for example:



p1.e + p2.e - p3.e - p4.e = 0;

p1.f = p2.f = p3.f = p4.f;

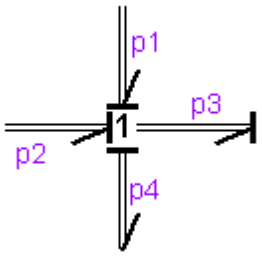
flow = p4.f;

A one junction has only one initial port p defined. Because any number of bonds may be connected, successive connected bonds are named p1, p2, p3 etc.

The plus or minus signs of the effort equation depend on the direction of the bonds. A bond pointing towards the one junction results in a plus sign and vice-versa. Only one port of a one junction may determine the flow. In the example port p4 determines the flow.

Multi Bonds

Bonds with a size larger than one (multi bonds) can also be connected to a one junction. Al connected bonds, however, must have the same size.



p1.e + p2.e - p3.e + p4.e = 0;

p1.f = p2.f = p3.f = p4.f;

flow = p4.f;

Naming conventions are equal to those of the single bond (size 1) junctions. Successive connected bonds are named p1, p2, p3 etc. To denote single elements of a multibond, matrix notation is used. E.g. the effort of bond number 3 of a multibond p2 can be denoted as p2.e[3] (columnvector notation) or p2.e[3,1] (matrix notation).

Interface

Ports	**Description**
p[any]	Any number of ports may be connected.
Causality	
1_effort p	Only one port may have an effort out causality (as seen from the element).
Outputs	
flow	The output signal is equal to the flow of the ports.

Limitations

All connected bonds must have the same size.

9.1.16 OneJunction-Activity

Library

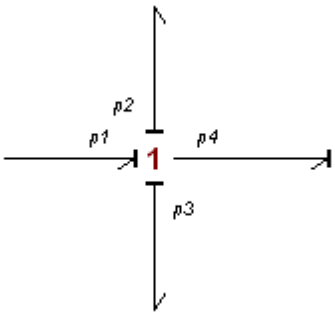
Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This junction is equal to a normal one junction but monitors the powerflows through the connected bonds. If you use this junction and connect some bonds, it could look like the figure below. The bond numbers are important for identification. You can see them by clicking **Port Names** from the **View** menu.



The constitutive equations for this example are:

p1.e - p2.e - p3.e - p4.e = 0;
p1.f = p2.f = p3.f = p4.f;
flow = p4.f;

The powerflows through the junction are monitored:

power = p.e . p.f;*

Note that power is a vector with a size equal to the number of bonds that are attached to the junction. The first element of the vector is the power of the first bond *(p1* in the example figure), the second element is the power of the second bond (*p2* in the example figure) and so on. To see which bonds are active and which bonds are not, the activity is calculated:

activity = int (abs (power));

To get an easy comparison, the relative activity is calculated:

maximum_activity = max (activity);
relative_activity = activity ./ maximum_activity

Interface

Ports	**Description**
p[any]	Any number of ports may be connected.
Causality	
1_effort p	Only one port may have a effort out causality (as seen from the element).
Outputs	
flow	The output signal is equal to the effort of the ports.
Variables	
power	Vector of which the elements are the powers of the connected bonds.

activity	Vector of which the elements are the integrated absolute powers of the connected bonds.
relative_activity	Vector of which the elements are the relative activities of the connected bonds.

Limitations

All connected bonds must have size 1.

9.1.17 PowerSensor

Library

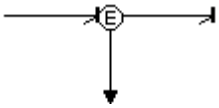
System , Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind**: Bond Graphs.

Description

This model can be inserted in any bond to yield the power that is carried through that bond as an output signal.



To ensure success, use drag and drop to place the sensor exactly on top in the middle of the bond. As a result the model will be automatically connected.

Interface

Ports

p1, p2	Input and output port of the power sensor.

Outputs

E	Power carried through the bond.

9.1.18 Power Splitter

Library

System , Bond Graph

Use

Domains: Continuous. **Size**: n-D. **Kind**: Bond Graphs.

Description

This model can be used to split a multi-bond into single bonds or vice versa. Default 2 bonds can be connected. Use the right mouse menu and choose *Edit Implementation* to change the number of bonds.

Interface

Inputs/Outputs

P(any)	Multi-bond (dimension [n,m]).
p	single bonds (a total of n*m)

Note

The multi-bond and corresponding single bonds must have opposite orientations. I.e. when the multi-bond is pointing **towards** the the power-splitter, all single bonds must point **from** the power splitter and when the multi-bond is pointing **from** the the power-splitter, all single bonds must point towards the signal splitter.

9.1.19 PSensor

Library

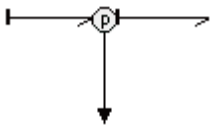
System , Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind**: Bond Graphs.

Description

This model can be inserted in any bond to yield the integral of the effort of that bond as an output signal.



To ensure success, use drag and drop to place the sensor exactly on top in the middle of the bond. As a result the model will be automatically connected.

Interface

Ports

p1, p2	Input and output port of the flow sensor.

Outputs

p	Integral of the effort of the bond.

9.1.20 Qsensor

Library

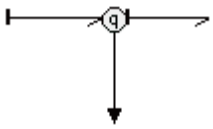
System , Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind**: Bond Graphs.

Description

This model can be inserted in any bond to yield the integral of the flow of that bond as an output signal.



To ensure success, use drag and drop to place the sensor exactly on top in the middle of the bond. As a result the model will be automatically connected.

Interface

Ports

p1, p2	Input and output port of the flow sensor.

Outputs

q	Integral of the flow of the bond.

9.1.21 R

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents a linear friction/resistor equation. It can have an effort out as well as a flow out causality. In the last case the constitutive equation, as shown below, is simply inverted.

effort out causality:

```
p.e = r*p.f;
```

flow out causality:

```
p.f = p.e/r;
```

Interface

Ports	Description
p	Input port of the R-element.

Causality

indifferent

Parameters

r	The friction/resistor parameter.

9.1.22 Se

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal effort source. The effort can be set to a certain constant value, the flow is indifferent.

```
p.e = s;
```

Ports	Description
p	Output port of the effort source.

Causality

fixed effort out

Parameters

s	The constant value of the generated effort

9.1.23 Sf

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal flow source. The flow can be set to a certain constant value, the effort is indifferent.

p.f = s;

Interface

Ports	Description
p	Output port of the flow source.

Causality

fixed flow out

Parameters

s	The constant value of the generated flow.

9.1.24 SGY

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal gyrator with gyration ratio 1. The model represents a one to one power continuous relation between the effort of one port and the flow of the other port and vice-versa. The model can be used to transform a C element into a I element etc. The model can have both ports with an effort causality or both ports with a flow causality:

effort out causality:

p1.e = p2.f;
p2.e = p1.f;

flow out causality:

p1.f = p2.e;
p2.f = p1.e;

Interface

Ports	Description
p1, p2	Input and output port of the gyrator.

Causality

p1 equal p2

9.1.25 STF

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal transformer with transform ratio 1. The model has no real effect but allows for domain changes (one port having another domain as the other). The model represents a power continuous relation between the efforts and flows of both its ports. The causality is always mixed: one port has an effort causality while the other has a flow causality:

p1.e = p2.e
p2.f = p1.f

or:

p2.e = p1.e
p1.f = p2.f

Interface

Ports	Description
p1, p2	Input and output port of the simple transformer.

Causality

p1 notequal p2	The causality of both ports must be different.

9.1.26 SwitchingOneJunction

Library

Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind**: Bond Graphs.

Description

This junction represents a switching 1 junction. Depending on the *condition* value it can connect / disconnect additional elements to / from the bond graph.

The *p12* junction port is connected when *condition <> 0.* The sum of the efforts on all three ports is zero and the flows on all ports are equal.
When disconnected (*condition == 0*), the effort and flow of port *p12* are both zero. Ports *p1 and p2* are still connected (flow = zero, efforts are equal).

Interface

Ports

p1, p2	Input and output port of the 1 junction.
p12	Conditional port (only connected when *condition* evaluates to a non-zero value)

Inputs

condition	Switching condition (non-zero value = connected)

Limitations

All connected bonds must have the same size.

9.1.27 SwitchingZeroJunction

Library

Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind**: Bond Graphs.

Description

This junction represents a switching 0 junction. Depending on the *condition* value it can connect / disconnect additional elements to / from the bond graph.

The *p12* junction port is connected when *condition <> 0.* The sum of the flows on all three ports is zero and the efforts on all ports are equal.
When disconnected (*condition == 0*), the effort and flow of port *p12* are both zero. Ports *p1 and p2* are still connected (effort = zero, flows are equal).

Interface

Ports

p1, p2	Input and output port of the 1 junction.
p12	Conditional port (only connected when *condition* evaluates to a non-zero value)

Inputs

condition	Switching condition (non-zero value = connected)

Limitations

All connected bonds must have the same size.

9.1.28 TF

Library

Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This model represents an ideal transformer. The model represents a power continuous relation between the efforts and flows of both its ports. The causality is always mixed: one port has an effort causality while the other has a flow causality:

*p1.e = r * p2.e*
*p2.f = r * p1.f*

or:

*p2.e = 1/r * p1.e*
*p1.f = 1/r * p2.f*

Interface

Ports	**Description**
p1, p2	Input and output port of the transformer.
Causality	
p1 notequal p2	The causality of both ports must be different.
Parameters	
r	Transform ratio

9.1.29 ZeroJunction

Library

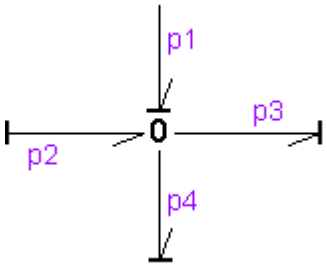
Bond Graph

Use

Domains: Continuous. **Size**: [n,m]. **Kind**: Bond Graphs.

Description

This junction represents a power continuous (no energy storage, dissipation or generation) connection of elements. The sum of the flows on all ports is zero and the efforts on all ports are equal. The constitutive equations are for example:



p1.f + p2.f - p3.f - p4.f = 0;

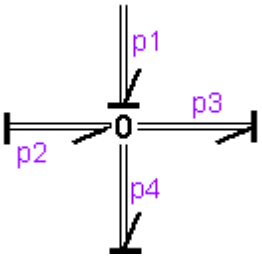
p1.e = p2.e = p3.e = p4.e;

effort = p1.e;

A zero junction has only one initial port p defined. Because any number of bonds may be connected, successive connected bonds are named p1, p2, p3 etc.
The plus or minus signs of the flow equation depend on the direction of the bonds. A bond pointing towards the zero junction results in a plus sign and vice-versa. Only one port of a zero junction may determine the effort. In the example port p1 determines the effort.

Multi Bonds

Bonds with a size larger than one (multi bonds) can also be connected to a zero junction. Al connected bonds, however, must have the same size.



p1.f + p2.f - p3.f + p4.f = 0;

p1.e = p2.e = p3.e = p4.e;

effort = p1.e;

Naming conventions are equal to those of the single bond (size 1) junctions. Successive connected bonds are named p1, p2, p3 etc. To denote single elements of a multibond, matrix notation is used. E.g the effort of bond number 3 of a multibond p2 can be denoted as p2.e[3] (columnvector notation) or p2.e[3,1] (matrix notation).

Interface

Ports	**Description**
p[any]	Any number of ports may be connected.
Causality	
1_flow p	Only one port may have a flow out causality (as seen from the element).
Outputs	
effort	The output signal is equal to the effort of the ports.

Limitations

All connected bonds must have the same size.

9.1.30 ZeroJunction-Activity

Library

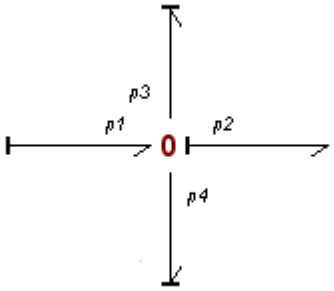
Bond Graph

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Bond Graphs.

Description

This junction is equal to a normal zero junction but monitors the powerflows through the connected bonds. If you use this junction and connect some bonds, it could look like the figure below. The bond numbers are important for identification. You can see them by clicking **Port Names** from the **View** menu.



The constitutive equations for this example are:

```
p1.f - p2.f - p3.f - p4.f = 0;
p1.e = p2.e = p3.e = p4.e;
output = p2.e;
```

The powerflows through the junction are monitored:

```
power = p.e .* p.f;
```

Note that power is a vector with a size equal to the number of bonds that are attached to the junction. The first element of the vector is the power of the first bond (p1 in the example figure), the second element is the power of the second bond (p2 in the example figure) and so on. To see which bonds are active and which bonds are not, the activity is calculated:

```
activity = int (abs (power));
```

To get an easy comparison, the relative activity is calculated:

```
maximum_activity = max (activity);
relative_activity = activity ./ maximum_activity
```

Interface

Ports	**Description**
p[any]	Any number of ports may be connected.
Causality	
1_flow p	Only one port may have a flow out causality (as seen from the element).
Outputs	
effort	The output signal is equal to the effort of the ports.
Variables	
power	Vector of which the elements are the powers of the connected bonds.

activity	Vector of which the elements are the integrated absolute powers of the connected bonds.
relative_activity	Vector of which the elements are the relative activities of the connected bonds.

Limitations

All connected bonds must have size 1.

9.1.31 3d

C-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model is the multiport equivalent of the single C storage element. Consequently the constitutive equation must be written as a matrix-vector multiplication. The element has a preferred effort out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred flow out causality. The constitutive equations then contain a derivation, which can only be simulated when the Backward Differentiation Formula integration algorithm is available:

effort out causality (preferred):

```
state = int(p.f) + state(0);
p.e = inverse(C)*state;
outp = state;
```

flow out causality:

```
state = C*p.e;
p.f = d state / dt;
output = state;
```

Interface

Ports	Description
p[3]	Input port of the storage element (columnvector with size 3).

Causality

preferred effort out	A flow out causality results in a derivative constitutive equation.

Outputs

output[3]	The output signal is equal to the state (columnvector with size 3).

Parameters

C[3,3]	The storage element constants (matrix of size [3,3]).

Initial Values

state(0)[3]	The initial values of the storage element (columnvector with size 3).

Limitations

The preferred equation contains an inverted C matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

GY-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single gyrator. Consequently the constitutive equation must be written as a matrix-vector multiplication. The model can have both ports with an effort out causality or both ports with a flow out causality:

effort out causality:

```
p1.e = transpose(R) * p2.f;
p2.e = R * p1.f;
```

flow out causality:

```
p1.f = inverse(transpose(R)) * p2.e;
p2.f = inverse(R) * p1.e;
```

Interface

Ports	Description
p1[3], p2[3]	Input and output port of the gyrator (columnvectors with size 3).
Causality	
p1 equal p2	The causality of both ports must be equal.
Parameters	
R[3]	Gyration ratio (matrix of size [3,3]).

Limitations

The flow out equations contain an inverted R matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

I-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model is the multiport equivalent of the single I storage element. Consequently the constitutive equation must be written as a matrix-vector multiplication. The element has a preferred flow out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred effort out causality. The constitutive equations then contain a derivation, which can only be simulated when the Backward Differentiation Formula integration algorithm is available:

flow out causality (preferred):

```
state = int(p.e) + state(0);
p.f = inverse(I)*state;
output = state;
```

effort out causality:

```
state = I*p.f;
p.e = d state / dt;
output = state;
```

Interface

Ports	Description
p[3]	Input port of the storage element (columnvector with size 3).
Causality	
preferred effort out	A flow out causality results in a derivative constitutive equation.
Outputs	
output[3]	The output signal is equal to the state (columnvector with size 3).
Parameters	
I[3,3]	The storage element constants (matrix of size [3,3]).
Initial Values	
state(0)[3]	The initial values of the storage element (columnvector with size 3).

Limitations

The preferred equation contains an inverted I matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

MGY-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single modulated gyrator. Consequently the constitutive equation must be written as a matrix-vector multiplication. The gyration ratio can be set to a certain (fluctuating) value, given by an input signal. The model can have both ports with an effort out causality or both ports with a flow out causality:

effort out causality:

*p1.e = transpose(r) * p2.f;*
*p2.e = r * p2.f;*

flow out causality:

*p1.f = inverse(transpose(r)) * p2.e;*
*p2.f = inverse(r) * p1.e;*

Interface

Ports	**Description**
p1[3], p2[3]	Input and output port of the gyrator (columnvectors with size 3).
Causality	
p1 equal p2	The causality of both ports must be equal.
Inputs	
r[3,3]	Modulated gyration ratio (size [3,3]).

Limitations

The flow out equations contain a matrix inversion of the modulation signal. The elements of this input signal should always have non-singular values.

MR-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single dissipative element. Consequently the constitutive equation must be written as a matrix-vector multiplication. This model can have an effort out as well as a flow out causality. In the last case the constitutive equation, as shown below, is simply inverted. The friction/resistor parameter can be set to a (fluctuating) value, given by an input signal.

effort out causality:

```
p.e = r*p.f;
```

flow out causality:

```
p.f = inverse(r)*p.e;
```

Interface

Ports	**Description**
p[3]	Input port of the R-element (columnvector with size 3).
Causality	
indifferent	
Input	
r[3,3]	The (modulated) friction/resistor parameter (size [3,3]).
Parameters	
R	The friction parameters (matrix of size [3,3]).

Limitations

The flow out equation contains an inverted R matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

MSe-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single modulated effort source. The effort can be set to a (fluctuating) value given by an input signal. The flow is indifferent.

p.e = input;

Ports	Description
p[3]	Output port of the effort source (columnvector with size 3).

Causality

fixed effort out

Inputs

input[3]	Modulation signal (columnvector with size 3).

MSf-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single modulated flow source. The flow can be set to a (fluctuating) value given by an input signal. The flow is indifferent.

p.f = input;

Ports	Description
p[3]	Output port of the flow source (columnvector with size 3).

Causality

fixed flow out

Inputs

input[3]	Modulation signal (columnvector with size 3).

MTF-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single modulated transformer. Consequently the constitutive equation must be written as a matrix-vector multiplication. The transform ratio can be set to a certain (fluctuating) value, given by an input signal. The causality is always mixed: one port has an effort out causality while the other has a flow out causality:

*p1.e = transpose(r) * p2.e;*
*p2.f = r * p1.f;*

or:

*p2.e = inverse(transpose(r)) * p1.e;*
*p1.f = inverse(r) * p2.f;*

Interface

Ports	Description
p1[3], p2[3]	Input and output port of the gyrator (columnvectors with size 3).

Causality

p1 notequal p2

Inputs

r[3,3]	Modulated transform ratio (size [3,3]).

Limitations

The second set of equations contain a matrix inversion of the modulation signal. The elements of this input signal should always have non-singular values.

R-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single dissipative element. Consequently the constitutive equation must be written as a matrix-vector multiplication. This model can have an effort out as well as a flow out causality. In the last case the constitutive equation, as shown below, is simply inverted.

effort out causality:

```
p.e = R*p.f;
```

flow out causality:

```
p.f = inverse(R)*p.e;
```

Interface

Ports	Description
p[3]	Input port of the R-element (columnvector with size 3).

Causality

indifferent

Parameters

R	The friction parameters (matrix of size [3,3]).

Limitations

The flow out equation contains an inverted R matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

Se-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model is the multiport equivalent of the single effort source. The effort can be set to a certain constant value, the flow is indifferent.

p.e = S;

Ports	Description
p[3]	Output port of the effort source (columnvector with size 3).

Causality

fixed effort out

Parameters

S[3]	The constant value of the generated effort (columnvector with size 3).

Sf-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model is the multiport equivalent of the single flow source. The effort can be set to a certain constant value, the flow is indifferent.

p.f = S;

Ports	Description
p[3]	Output port of the flow source (columnvector with size 3).

Causality

fixed flow out

Parameters

s[3]	The constant value of the generated flow (columnvector with size 3).

SGY-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single symplectic gyrator. Consequently the constitutive equation must be written as a matrix-vector multiplication. This model represents an ideal gyrator with gyration ratio equal to the identity matrix. The model represents a one to one power continuous relation between the effort of one port and the flow of the other port and vice-versa. The model can be used to transform a C-3 element into a I-3 element etc. The model can have both ports with an effort causality or both ports with a flow causality:

effort out causality:

p1.e = p2.f;
p2.e = p1.f;

flow out causality:

p1.f = p2.e;
p2.f = p1.e;

Interface

Ports	**Description**
p1[3], p2[3]	Input and output port of the gyrator (columnvectors with size 3).

Causality

p1 equal p2	The causality of both ports must be equal.

STF-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single symplectic transformer. This model represents an ideal transformer with a transform ratio equal to the identity matrix. The model represents a one to one power continuous relation between the effort of one port and the flow of the other port and vice-versa. The causality is always mixed: one port has an effort causality while the other has a flow causality:

p1.e = p2.e;
p2.f = p1.f;

or:

p2.e = p1.e
p1.f = p2.f

Interface

Ports	Description
p1[3], p2[3]	Input and output port of the transformer (columnvectors with size 3).
Causality	
p1 notequal p2	The causality of both ports must be different.

TF-3

Library

Bond Graph

Use

Domains: Continuous. **Size**: 3-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single transformer. Consequently the constitutive equation must be written as a matrix-vector multiplication. The causality is always mixed: one port has an effort causality while the other has a flow causality:

*p1.e = transpose(R) * p2.e;*
*p2.f = R * p1.f;*

or:

*p2.e = inverse(transpose(R)) * p1.e*
*p1.f = inverse(R) * p2.f*

Interface

Ports	**Description**
p1[3], p2[3]	Input and output port of the transformer (columnvectors with size 3).
Causality	
p1 notequal p2	The causality of both ports must be different.
Parameters	
R[3,3]	Transform ratio (matrix of size [3,3]).

Limitations

The lower set of equations contain an inverted R matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

9.1.32 2d

C-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model is the multiport equivalent of the single C storage element. Consequently the constitutive equation must be written as a matrix-vector multiplication. The element has a preferred effort out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred flow out causality. The constitutive equations then contain a derivation, which can only be simulated when the Backward Differentiation Formula integration algorithm is available:

effort out causality (preferred):

state = int(p.f) + state(0);
*p.e = inverse(C)*state;*
outp = state;

flow out causality:

*state = C*p.e;*
p.f = d state / dt;
output = state;

Interface

Ports	**Description**
p[2]	Input port of the storage element (columnvector with size 2).
Causality	
preferred effort out	A flow out causality results in a derivative constitutive equation.
Outputs	
output[2]	The output signal is equal to the state (columnvector with size 2).
Parameters	
C[2,2]	The storage element constants (matrix of size [2,2]).
Initial Values	
state(0)[2]	The initial values of the storage element (columnvector with size 2).

Limitations

The preferred equation contains an inverted C matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

GY-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single gyrator. Consequently the constitutive equation must be written as a matrix-vector multiplication. The model can have both ports with an effort out causality or both ports with a flow out causality:

effort out causality:

```
p1.e = transpose(R) * p2.f;
p2.e = R * p1.f;
```

flow out causality:

```
p1.f = inverse(transpose(R)) * p2.e;
p2.f = inverse(R) * p1.e;
```

Interface

Ports	**Description**
p1[2], p2[2]	Input and output port of the gyrator (columnvectors with size 2).
Causality	
p1 equal p2	The causality of both ports must be equal.
Parameters	
R[2]	Gyration ratio (matrix of size [2,2]).

Limitations

The flow out equations contain an inverted R matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

I-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model is the multiport equivalent of the single I storage element. Consequently the constitutive equation must be written as a matrix-vector multiplication. The element has a preferred flow out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred effort out causality. The constitutive equations then contain a derivation, which can only be simulated when the Backward Differentiation Formula integration algorithm is available:

flow out causality (preferred):

```
state = int(p.e) + state(0);
p.f = inverse(I)*state;
output = state;
```

effort out causality:

```
state = I*p.f;
p.e = d state / dt;
output = state;
```

Interface

Ports	**Description**
p[2]	Input port of the storage element (columnvector with size 2).
Causality	
preferred effort out	A flow out causality results in a derivative constitutive equation.
Outputs	
output[2]	The output signal is equal to the state (columnvector with size 2).
Parameters	
I[2,2]	The storage element constants (matrix of size [2,2]).
Initial Values	

state(0)[2]	The initial values of the storage element (columnvector with size 2).

Limitations

The preferred equation contains an inverted I matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

MGY-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single modulated gyrator. Consequently the constitutive equation must be written as a matrix-vector multiplication. The gyration ratio can be set to a certain (fluctuating) value, given by an input signal. The model can have both ports with an effort out causality or both ports with a flow out causality:

effort out causality:

*p1.e = transpose(r) * p2.f;*
*p2.e = r * p2.f;*

flow out causality:

*p1.f = inverse(transpose(r)) * p2.e;*
*p2.f = inverse(r) * p1.e;*

Interface

Ports	**Description**
p1[2], p2[2]	Input and output port of the gyrator (columnvectors with size 2).
Causality	
p1 equal p2	The causality of both ports must be equal.
Inputs	
r[2,2]	Modulated gyration ratio (size [2,2]).

Limitations

The flow out equations contain a matrix inversion of the modulation signal. The elements of this input signal should always have non-singular values.

MR-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single dissipative element. Consequently the constitutive equation must be written as a matrix-vector multiplication. This model can have an effort out as well as a flow out causality. In the last case the constitutive equation, as shown below, is simply inverted. The friction/resistor parameter can be set to a (fluctuating) value, given by an input signal.

effort out causality:

```
p.e = r*p.f;
```

flow out causality:

```
p.f = inverse(r)*p.e;
```

Interface

Ports	Description
p[2]	Input port of the R-element (columnvector with size 2).

Causality

indifferent

Input

r[2,2]	The (modulated) friction/resistor parameter (size [2,2]).

Parameters

R	The friction parameters (matrix of size [2,2]).

Limitations

The flow out equation contains an inverted R matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

MSe-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single modulated effort source. The effort can be set to a (fluctuating) value given by an input signal. The flow is indifferent.

p.e = input;

Ports	**Description**
p[2]	Output port of the effort source (columnvector with size 2).

Causality

fixed effort out

Inputs

input[2]	Modulation signal (columnvector with size 2).

MSf-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single modulated flow source. The flow can be set to a (fluctuating) value given by an input signal. The flow is indifferent.

p.f = input;

Ports	Description
p[2]	Output port of the flow source (columnvector with size 2).

Causality

fixed flow out

Inputs

input[2]	Modulation signal (columnvector with size 2).

MTF-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single modulated transformer. Consequently the constitutive equation must be written as a matrix-vector multiplication. The transform ratio can be set to a certain (fluctuating) value, given by an input signal. The causality is always mixed: one port has an effort out causality while the other has a flow out causality:

*p1.e = transpose(r) * p2.e;*
*p2.f = r * p1.f;*

or:

*p2.e = inverse(transpose(r)) * p1.e;*

*p1.f = inverse(r) * p2.f;*

Interface

Ports	Description
p1[2], p2[2]	Input and output port of the gyrator (columnvectors with size 2).

Causality

p1 notequal p2

Inputs

r[2,2]	Modulated transform ratio (size [2,2]).

Limitations

The second set of equations contain a matrix inversion of the modulation signal. The elements of this input signal should always have non-singular values.

R-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single dissipative element. Consequently the constitutive equation must be written as a matrix-vector multiplication. This model can have an effort out as well as a flow out causality. In the last case the constitutive equation, as shown below, is simply inverted.

effort out causality:

*p.e = R*p.f;*

flow out causality:

*p.f = inverse(R)*p.e;*

Interface

Ports	Description
p[2]	Input port of the R-element (columnvector with size 2).

Causality

indifferent

Parameters

R	The friction parameters (matrix of size [2,2]).

Limitations

The flow out equation contains an inverted R matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

Se-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model is the multiport equivalent of the single effort source. The effort can be set to a certain constant value, the flow is indifferent.

$$p.e = S;$$

Ports	Description
p[2]	Output port of the effort source (columnvector with size 2).

Causality

fixed effort out

Parameters

S[2]	The constant value of the generated effort (columnvector with size 2).

Sf-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Block Diagrams, Bond Graphs.

Description

This model is the multiport equivalent of the single flow source. The effort can be set to a certain constant value, the flow is indifferent.

p.f = S;

Ports	Description
p[2]	Output port of the flow source (columnvector with size 2).

Causality

fixed flow out

Parameters

s[2]	The constant value of the generated flow (columnvector with size 2).

SGY-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single symplectic gyrator. Consequently the constitutive equation must be written as a matrix-vector multiplication. This model represents an ideal gyrator with gyration ratio equal to the identity matrix. The model represents a one to one power continuous relation between the effort of one port and the flow of the other port and vice-versa. The model can be used to transform a C-2 element into a I-2 element etc. The model can have both ports with an effort causality or both ports with a flow causality:

effort out causality:

p1.e = p2.f;
p2.e = p1.f;

flow out causality:

p1.f = p2.e;
p2.f = p1.e;

Interface

Ports	**Description**
p1[2], p2[2]	Input and output port of the gyrator (columnvectors with size 2).

Causality

p1 equal p2	The causality of both ports must be equal.

STF-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single simple transformer. This model represents an ideal transformer with a transform ratio equal to the identity matrix. The model has no real effect but allows for domain changes (one port having another domain as the other). The model represents a power continuous relation between the efforts and flows of both its ports. The causality is always mixed: one port has an effort causality while the other has a flow causality:

p1.e = p2.e;
p2.f = p1.f;

or:

p2.e = p1.e
p1.f = p2.f

Interface

Ports	**Description**

p1[2], p2[2]	Input and output port of the transformer (columnvectors with size 2).

Causality

p1 notequal p2	The causality of both ports must be different.

TF-2

Library

Bond Graph

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Bond Graphs.

Description

This model is the multiport equivalent of the single transformer. Consequently the constitutive equation must be written as a matrix-vector multiplication. The causality is always mixed: one port has an effort causality while the other has a flow causality:

*p1.e = transpose(R) * p2.e;*
*p2.f = R * p1.f;*

or:

*p2.e = inverse(transpose(R)) * p1.e*
*p1.f = inverse(R) * p2.f*

Interface

Ports	**Description**
p1[2], p2[2]	Input and output port of the transformer (columnvectors with size 2).

Causality

p1 notequal p2	The causality of both ports must be different.

Parameters

R[2,2]	Transform ratio (matrix of size [2,2]).

Limitations

The lower set of equations contain an inverted R matrix. The elements of this matrix should be chosen with care to prevent this matrix from becoming singular.

9.2 Iconic Diagrams

9.2.1 Iconic Diagrams

The Iconic Diagram library contains components which are very useful for modeling physical systems. The following libraries are available.

- Electric
- Hydraulics
- Mechanical
- Thermal

9.2.2 Electric

Electric

The Electric library contains components which are very useful for modeling electrical systems. The library contains the following sections:

- Actuators
- Components
- Sources

Actuators

CMABender

Library

Iconic Diagrams\Electric\Actuators
Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents a piezo actuator. The actuator translates a voltage difference at port *p1* to a mechanical position difference between the base at port *p2* and the end effector at port *p3*.

Although the underlying equations of this model are equal to the equations of the `CMAStrecher.emx model`, the default parameter values are typical for bending.

Interface

Ports	Description
p	Translation port.

Causality

fixed force out

Input

C	Capacitance [F]
KF	Voltage to force conversion factor [N/V]
m	Equivalent mass [kg]
k	Stiffness [N/m]
B	Relative damping ratio []
KB	Force to voltage conversion factor [V/N]

CMAStretcher

Library

Iconic Diagrams\Electric\Actuators
Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents a piezo actuator. The actuator translates a voltage difference at port *p1* to a mechanical position difference between the base at port *p2* and the end effector at port *p3*.

Although the underlying equations of this model are equal to the equations of the CMABender.emx model, the default parameter values are typical for stretching.

Interface

Ports	**Description**
p	Translation port.

Causality

fixed force out

Input

C	Capacitance [F]
KF	Voltage to force conversion factor [N/V]
m	Equivalent mass [kg]
k	Stiffness [N/m]
B	Relative damping ratio []
KB	Force to voltage conversion factor [V/N]

DCMotor

Library

Iconic Diagrams\Electric\Actuators
Iconic Diagrams\Mechanical\Electric

Implementations

Default
IR

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric, Electric).

Description - Default

This models represents an ideal DC-motor with no energy loss. The electric port has separate high and low terminals. The equations are

```
p1.i = p1_high.i = p1_low.i;
p1.u = p1_high.u - p1_low.u;
```

The model can have mixed forms of causality

```
p1.u = k * p2.omega;
p2.T = k * p1.i;
```

or:

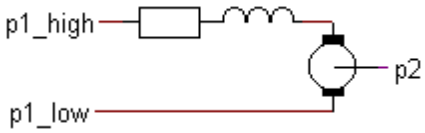
```
p2.omega = p1.u / k;
p1.i = p2.T / k;
```

Interface - Default

Ports	**Description**
p1_high, p1_low	Both terminals of the Electric port p1.
p2	Rotation port.
Causality	
mixed	See equations above.
Parameters	
k	motor constant [Nm/A]

Description - IR

This models represents an ideal DC-motor with inductance and resistance.



The electric port has separate high and low terminals. The equations are

p1.i = p1_high.i = p1_low.i;
p1.u = p1_high.u - p1_low.u;

Interface - IR

Ports	**Description**
p1_high, p1_low	Both terminals of the Electric port p1.
p2	Rotation port.
Causality	
mixed	See equations above.
Parameters	
k	motor constant [Nm/A]
L	motor inductance [H]
R	motor resistance [Ohm]

Components

Capacitor

Library

Iconic Diagrams\Electric\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This model represents an ideal capacitor. The element has a preferred voltage out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred current out causality. The constitutive equations then contain a derivation. The port p of the inductor model has separate high and low terminals. The equations are:

p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;

voltage out causality (preferred):

*p.u = (1/C)*int(p.i);*

current out causality:

*p.i = C * ddt(p.u);*

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Electric port p.
Causality	
preferred voltage out	A current out causality results in a derivative constitutive equation.
Parameters	
C	capacitance [F]
Initial Values	
p.u_initial	The initial charge of the capacitor [V].

CurrentSensor

Library

Iconic Diagrams\Electric\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric), Block Diagrams.

Description

This model translates a current to an output signal. It has a voltage out causality. The port p of the model has separate high and low terminals. The equations are:

```
p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;
p.u = 0;
i = p.i;
```

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Electric port p.
Causality	
fixed voltage out	

Output

i	Current [A].

Diode

Library

Iconic Diagrams\Electric\Components

Implementations

Default
Exponential

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description - Default

This is an ideal electrical diode. The model is a switch which is open when the voltage drop v < 0 and closed when the voltage drop v > 0. The heart of the model consist of a resistance that is changed by the input voltage from almost zero (on) to a very large value (off). By proper selection of the on and off resistances, they can be effectively zero and infinity in comparison to other circuit elements. The port p of the diode model has separate high and low terminals. The equations are:

p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;

voltage out causality:

R = if p.i > 0 then Ron else Roff end;
*p.u = R * p.i;*

current out causality:

R = if p.u > 0 then Ron else Roff end;
p.i = p.u / R;

Interface - Default

Ports	**Description**
p_high, p_low	Both terminals of the Electric port p.
Causality	
indifferent p	
Parameters	
Ron	Resistance when diode is turned on [Ohm]
Roff	Resistance when diode is turned off [Ohm]

Description - Exponential

This is an electrical diode described by an exponential expression. The port p of the diode model has separate high and low terminals. The equations are:

p.i = p_high.i = p_low.i
p.u = p_high.u - p_low.u

*uT = (k * T) / e;*
*p.i = Is * (exp (p.u / uT) - 1);*

with k the Boltzmann constant (k = 1.380658e-23 {J/K}

Interface - Exponential

Ports	**Description**
p_high, p_low	Both terminals of the Electric port p.
Causality	
indifferent p	
Parameters	
T	operating temperature [K]
Is	reverse saturation current [A]

DoubleSwitch

Library

Iconic Diagrams\Electric\Components

Implementations

Level
Amplitude

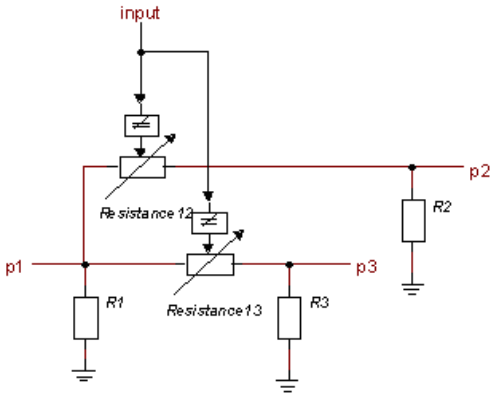
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description - Level

This model represents an almost ideal switch. The switch connects port *p1* with *p2* when the input signal is larger than the threshold value *vt* and connects port *p1* with *p3* when the input signal is smaller or equal to the threshold value *vt*.

The heart of the model consist of two resistances that can be changed by an input signal from almost zero (on) to a very large value (off). By proper selection of the on and off resistances, they can be effectively zero and infinity in comparison to other circuit elements.



Layout of the switch model.

The equations of the resistances are:

R = if input > vt then Ron else Roff end;
*p.u = R * p.i;*

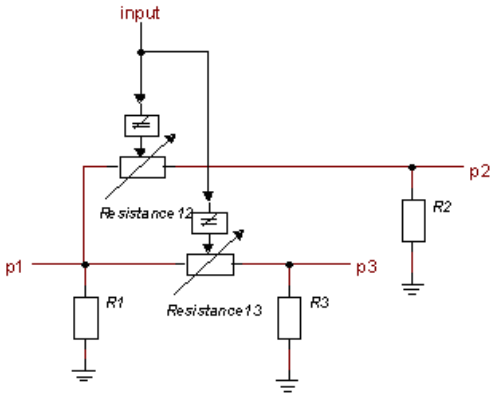
Interface - Level

Ports	**Description**
p1, p2, p3	The electric ports (Electric).
input	The switching signal.
Causality	
indifferent p1, p2, p3	
Parameters	
R1\R	Ground resistance at port 1 [Ohm]
R2\R	Ground resistance at port 2 [Ohm]
R3\R	Ground resistance at port 3 [Ohm]
Resistance12\Ron	Resistance between p1 and p2 when switch is turned on [Ohm]
Resistance12\Roff	Resistance between p1 and p2 when switch is turned off [Ohm]
Resistance12\vt	Threshold value between p1 and p2, switch is turned on when input > Resistance12\vt
Resistance13\Ron	Resistance between p1 and p3 when switch is turned on [Ohm]
Resistance13\Roff	Resistance between p1 and p3 when switch is turned off [Ohm]
Resistance13\vt	Threshold value between p1 and p3, switch is turned on when input > Resistance13\vt

Description - Amplitude

This model represents an almost ideal switch. The switch connects port *p1* with *p2* when the **absolute value** of the input signal is larger than the threshold value *Resistance12 \vt* and connects port *p1* with *p3* when the **absolute value** of the input signal is smaller or equal to the threshold value *Resistance13\vt*.

The heart of the model consist of two resistances that can be changed by an input signal from almost zero (on) to a very large value (off). By proper selection of the on and off resistances, they can be effectively zero and infinity in comparison to other circuit elements.



Layout of the switch model.

The equations of the resistances are:

```
R = if abs(input) > vt then Ron else Roff end;
p.u = R * p.i;
```

Interface - Amplitude

Ports	**Description**
p1, p2, p3	The electric ports (Electric).
input	The switching signal.
Causality	
indifferent p1, p2, p3	
Parameters	
R1\R	Ground resistance at port 1 [Ohm]
R2\R	Ground resistance at port 2 [Ohm]
R3\R	Ground resistance at port 3 [Ohm]
Resistance12\Ron	Resistance between p1 and p2 when switch is turned on [Ohm]
Resistance12\Roff	Resistance between p1 and p2 when switch is turned off [Ohm]
Resistance12\vt	

	Threshold value between p1 and p2, switch is turned on when abs(input) > Resistance12\vt
Resistance13\Ron	Resistance between p1 and p3 when switch is turned on [Ohm]
Resistance13\Roff	Resistance between p1 and p3 when switch is turned off [Ohm]
Resistance13\vt	Threshold value between p1 and p3, switch is turned on when abs(input) > Resistance13\vt

Note

To get a simple switching behavior, pay attention to chose the parameters *Resistance12 \vt* and *Resistance13\vt* equal. When these parameters are not chosen equal a more complex switching behavior can be obtained.

Ground

Library

Iconic Diagrams\Electric\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This model represents the ground (voltage = 0). The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. which gives the constitutive equations:

p1.u = p2.u = .. = pn.u = 0;
p1.i = free; p2.i = free; ..; pn.i = free;

Interface

Ports	**Description**
p [any]	Any number of connections can be made (Electric).
Causality	
Fixed voltage out	All ports have a fixed voltage out causality.

Inductor

Library

Iconic Diagrams\Electric\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This model represents an ideal electrical inductance. The element has a preferred current out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred voltage out causality. The constitutive equations then contain a derivation. The port p of the inductor model has separate high and low terminals. The equations are:

p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;

current out causality (preferred):

p.i = (1/L)int(p.u);

voltage out causality:

*p.u = L * ddt(p.i);*

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Electric port p.
Causality	
preferred current out	An voltage out causality results in a derivative constitutive equation.
Parameters	
L	Inductance [H]
Initial Values	
p.i_initial	The initial flux in the inductor.

Node

Library

Iconic Diagrams\Electric\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This node model represents a structural connection between two or more branches of an electrical circuit. The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc.

Interface

Ports	Description
p [any]	Any number of connections can be made.

OpAmp

Library

Iconic Diagrams\Electric\Components

Implementations

Default
Universal

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description - Default

This model represents an ideal operational amplifier. The input resistance is infinite and the output resistance is zero. The port *p_in* of the OpAmp model has separate high and low terminals:

```
p_in.u = p_in_high.u - p_in_low.u;
p_in_high.i = p_in_low.i = p_in.i = 0;
p_out.u = A*p_in.u;
```

Interface - Default

Ports	Description
p_in_high,	Both terminals of the input port p_in (Electric).
P_in_low	Output port (Electric).
p_out	

Causality

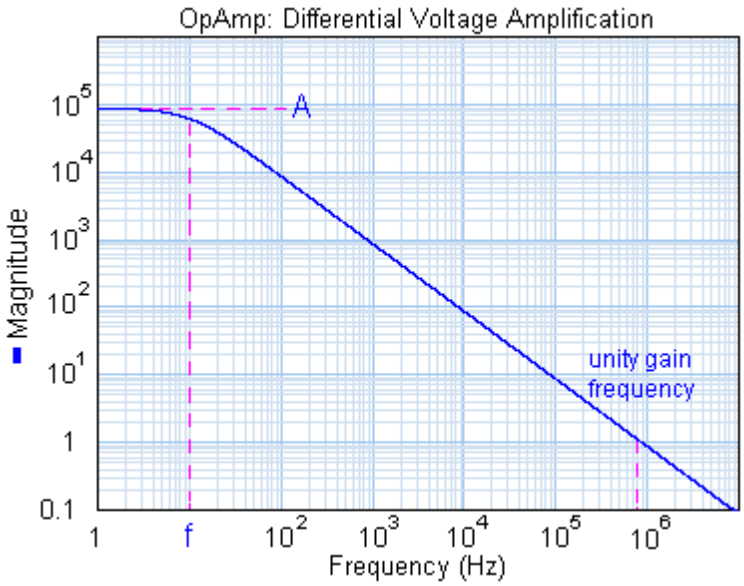
fixed current out
p_in
fixed voltage out
p_out

Parameters

A Amplification []

Description - Universal

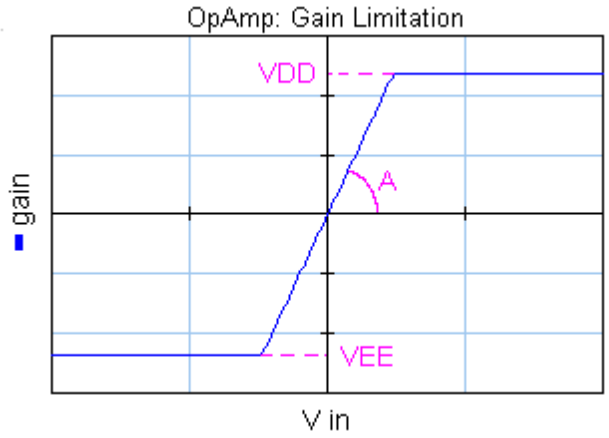
This model represents an operational amplifier with a first order decreasing voltage amplification.



The amplification is parameterized by a DC-gain (A) and a cut-off frequency (f). This yields a unity gain frequency of:

unity gain frequency = A*f;

The gain is limited to yield an output that is between the positive (VDD) and negative (VEE) supply voltage.



The input resistance is infinite and the output resistance is equal to the parameter R. The port p_in of the OpAmp model has separate high and low terminals:

p_in.u = p_in_high.u - p_in_low.u
p_in_high.i = p_in_low.i = p_in.i = 0;
*p_out.u = Adiff*p_in.u - R*p_out.u;*

where Adiff is a dynamic gain according to the bode and gain plot.

Interface - Universal

Ports	**Description**
p_in_high,	Both terminals of the input port p_in (Electric).
P_in_low	Output port (Electric).
p_out	
Causality	
fixed current out p_in	
preferred voltage out p_out	
Parameters	
A	DC gain or DC Amplification
f	Cut-off frequency [Hz]
VDD	Positive supply voltage [V], choose VDD > VEE!
VEE	Negative supply voltage [V], choose VDD > VEE!
R	Output impedance [Ohm]

Potentiometer

Library

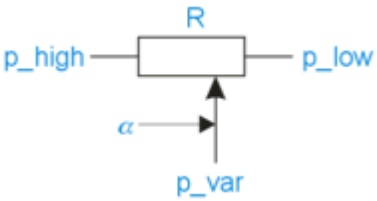
Iconic Diagrams\Electric\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This model represents an ideal potentiometer.



It can have various forms of causality. The equations are:

```
R1 = (1 - alpha) * R;
R2 = alpha * R;
p_high.u - p_var.u = p_high.i * R1;
p_var.u - p_low.u = p_low.i * R2;
p_high.i + p_var.i = p_low.i;
```

where alpha denotes the position of the indicator and can vary from zero to one.

Interface

Ports	**Description**
p_high, p_low	Both terminals of the resistor.
p_var	Indicator terminal.
Causality	
indifferent	
Parameters	
R	Resistance [Ohm].
alpha	Indicator position (0 <= alpha <= 1).

Rectifier

Library

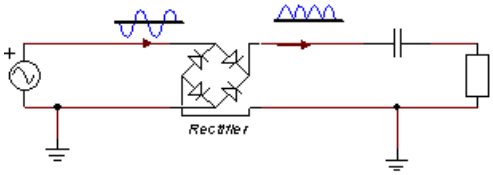
Iconic Diagrams\Electric\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This model represents an diode bridge rectifier consisting of a set four diodes. Given an alternating input current, the bridge produces an output current which is always positively oriented and has the same amplitude as the input current. The bridge behavior is ideal. No diode characteristics are implemented and no power is dissipated.



A circuit with a diode bridge rectifier.

Interface

Ports	Description
p1_high, p1_low	Both terminals of the Electric input port p1.
p2_high, p2_low	Both terminals of the Electric output port p2.

Causality

p1 not equal p2

Resistor

Library

Iconic Diagrams\Electric\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This model represents an ideal electrical resistor. It can have an voltage out as well as a current out causality. In the last case the constitutive equation, as shown below, is simply inverted. The port p of the damper model has separate high and low terminals. The equations are:

p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;

Voltage out causality:

*p.u = R * p.i;*

Current out causality:

p.i = p.u / R;

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Electric port p.
Causality	
indifferent	
Parameters	
R	Resistance [Ohm].

Switch

Library

Iconic Diagrams\Electric\Components

Implementations

Level
Amplitude

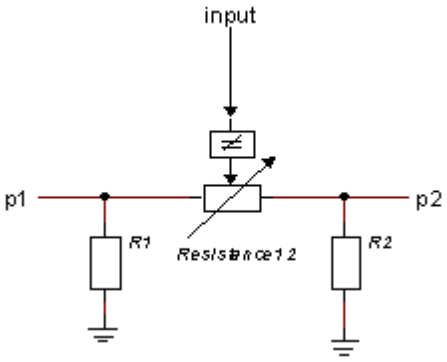
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description - Level

This model represents an almost ideal switch. The switch is turned on when the input signal is larger than the threshold value *vt* and it is turned off when the input signal is smaller or equal to the threshold value *vt*.

The heart of the model consist of a resistance that can be changed by an input signal from almost zero (on) to a very large value (off). By proper selection of the on and off resistances, they can be effectively zero and infinity in comparison to other circuit elements.



Layout of the switch model.

The equations of the resistance are:

R = if input > vt then Ron else Roff end;
*p.u = R * p.i;*

Interface - Level

Ports	**Description**
p1, p2	Both electric ports (Electric).
input	The switching signal.

Causality

indifferent p1, p2

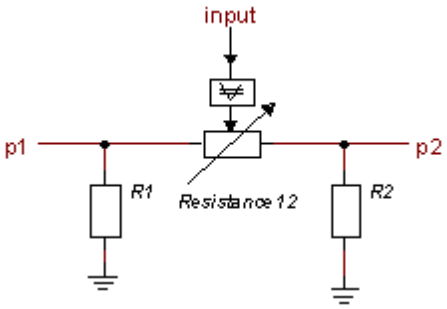
Parameters

R1\R	Ground resistance at port 1 [Ohm]
R2\R	Ground resistance at port 2 [Ohm]
Resistance12\Ron	Resistance when switch is turned on [Ohm]
Resistance12\Roff	Resistance when switch is turned off [Ohm]
Resistance12\vt	Threshold value, switch is turned on when input > vt

Description - Amplitude

This model represents an almost ideal switch. The switch is turned on when the amplitude of the input signal is larger than the threshold value *vt* and it is turned off when the amplitude of the input signal is smaller or equal to the threshold value *vt.*

The heart of the model consist of a resistance that can be changed by an input signal from almost zero (on) to a very large value (off). By proper selection of the on and off resistances, they can be effectively zero and infinity in comparison to other circuit elements.



Layout of the switch model.

The equations of the resistance are:

```
R = if abs(input) > vt then Ron else Roff end;
p.u = R * p.i;
```

Interface - Amplitude

Ports	**Description**
p1, p2	Both electric ports (Electric).
input	The switching signal.
Causality	
indifferent p1,p2	
Parameters	
R1\R	Ground resistance at port 1 [Ohm]
R2\R	Ground resistance at port 2 [Ohm]
Resistance12\Ron	Resistance when switch is turned on [Ohm]
Resistance12\Roff	Resistance when switch is turned off [Ohm]
Resistance12\vt	Threshold value, switch is turned on when abs(input) > vt

Transformer

Library

Iconic Diagrams\Electric\Components

Implementations

Default
Induction

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description - Default

This models represents an ideal transformer. The transformer does not have internal resistors, capacitance or inductance. The causality of this model is always mixed: one port has a voltage causality while the other has a current out causality:

```
p_1.u = n * p_2.u;
p_2.i = n * p_1.i;
```

or:

```
p_2.u = 1/n * p_1.u;
p_1.i = 1/n * p_2.i;
```

Interface - Default

Ports	**Description**
p_1,p_2	Electrical ports
Causality	
p_small notequal	
p_large	
Parameters	
n	turns ratio []

Description - Induction

This models represents a transformer with inductance and mutual inductance. The equations are:

```
p_1.u = L1 * ddt(p_1.i) + M * ddt(p_2.i);
p_2.u = M * ddt(p_1.i) + L2 * ddt(p_2.i);
```

Interface - Induction

Ports	**Description**
p_1,p_2	Electrical ports
Causality	
indifferent	
Parameters	
L1	Primary inductance [H]
L2	Secondary Inductance [H]
M	Coupling Inductance [H]

VoltageSensor

Library

Iconic Diagrams\Electric\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric), Block Diagrams.

Description

This model translates a voltage difference to an output signal. It has a current out causality. The port p of the model has separate high and low terminals. The equations are:

```
p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;
p.i = 0;
u = p.u;
```

Interface

Ports	Description
p_high, p_low	Both terminals of the Electric port p.
Causality	
fixed current out	
Output	
u	Voltage [V].

Sources

ControlledCurrentSource

Library

Iconic Diagrams\Electric\Sources

Implementations

Voltage
Current

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description - Voltage

This model represents an ideal voltage controlled current source. Both ports of the model have separate high and low terminals:

p_in.u = p_in_high.u - p_in_low.u;
p_in_high.i = p_in_low.i = p_in.i;
p_out.u = p_out_high.u - p_out_low.u = 0
p_out_high.i = p_out_low.i = p_in.i;

The equations of this model are:

p_in.i = 0;
p_out.i = p_in.u;
p_out.u = indifferent;

Interface - Voltage

Ports	Description
p_in_high,	Both terminals of the input port p_in (Electric).
P_in_low	Both terminals of the output port p_out (Electric).
p_out_high,	
P_out_low	

Causality

fixed current out
p_in
fixed current out
p_out

Description - Current

This model represents an ideal current controlled current source. Both ports of the model have separate high and low terminals:

p_in.u = p_in_high.u - p_in_low.u;
p_in_high.i = p_in_low.i = p_in.i;
p_out.u = p_out_high.u - p_out_low.u = 0
p_out_high.i = p_out_low.i = p_in.i;

The equations of this model are:

p_in.u = 0;
p_out.i = p_in.i;
p_out.u = indifferent;

Interface - Current

Ports	Description
p_in_high,	Both terminals of the input port p_in (Electric).
P_in_low	Both terminals of the output port p_out (Electric).
p_out_high,	
P_out_low	

Causality

fixed voltage out
p_in
fixed current out
p_out

ControlledVoltageSource

Library

Iconic Diagrams\Electric\Sources

Implementations

Voltage
Current
PI
KP

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description - Voltage

This model represents an ideal voltage controlled voltage source. Both ports of the model have separate high and low terminals:

p_in.u = p_in_high.u - p_in_low.u;
p_in_high.i = p_in_low.i = p_in.i;
p_out.u = p_out_high.u - p_out_low.u = 0
p_out_high.i = p_out_low.i = p_in.i;

The equations of this model are:

p_in.i = 0;
p_out.u = p_in.u;
p_out.i = indifferent;

Interface - Voltage

Ports	Description
p_in_high,	Both terminals of the input port p_in (Electric).
P_in_low	Both terminals of the output port p_out (Electric).
p_out_high,	
P_out_low	

Causality

fixed current out
p_in

fixed voltage out
p_out

Description - Current

This model represents an ideal current controlled voltage source. Both ports of the model have separate high and low terminals:

p_in.u = p_in_high.u - p_in_low.u;
p_in_high.i = p_in_low.i = p_in.i;
p_out.u = p_out_high.u - p_out_low.u = 0
p_out_high.i = p_out_low.i = p_in.i;

The equations of this model are:

p_in.u = 0;
p_out.u = p_in.i;
p_out.i = indifferent;

Interface - Current

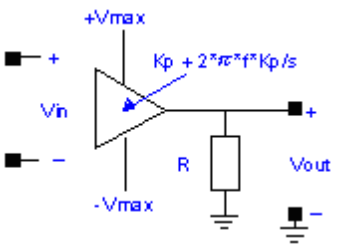
Ports	**Description**
p_in_high,	Both terminals of the input port p_in (Electric).
P_in_low	Both terminals of the output port p_out (Electric).
p_out_high,	
P_out_low	

Causality

fixed voltage out
p_in
fixed voltage out
p_out

Description - PI

This model represents a voltage controlled voltage source. The output voltage is connected to the input voltage through a PI-controller and limited to a maximum and minimum value of ±*Vmax*. The output voltage is subjected to an output resistance *R*.



Both ports of the model have separate high and low terminals:

p_in.u = p_in_high.u - p_in_low.u;
p_in_high.i = p_in_low.i = p_in.i;
p_out.u = p_out_high.u - p_out_low.u = 0
p_out_high.i = p_out_low.i = p_in.i;

Interface - PI

Ports	Description
p_in_high, p_in_low	Both terminals of the input port p_in (Electric).
p_out_high, p_out_low	Both terminals of the output port p_out (Electric).

Causality

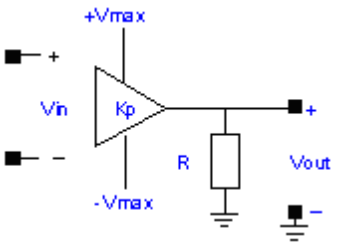
fixed current out p_in
fixed voltage out p_out

Parameters

Kp	Proportional gain []
f	Integration frequency [Hz]
Vmax	Maximum output voltage [V]
R	Output resistance [Ohm]

Description - KP

This model represents a voltage controlled voltage source. The output voltage is proportional to the input voltage and is limited to a maximum and minimum value of ±*Vmax*. The output voltage is subjected to an output resistance *R*.



Both ports of the model have separate high and low terminals:

p_in.u = p_in_high.u - p_in_low.u;
p_in_high.i = p_in_low.i = p_in.i;
p_out.u = p_out_high.u - p_out_low.u = 0
p_out_high.i = p_out_low.i = p_in.i;

Interface - KP

Ports	Description
p_in_high, p_in_low	Both terminals of the input port p_in (Electric).
p_out_high, p_out_low	Both terminals of the output port p_out (Electric).
Causality	
fixed current out p_in	
fixed voltage out p_out	
Parameters	
Kp	Proportional gain []
Vmax	Maximum output voltage [V]
R	Output resistance [Ohm]

CurrentSource

Library

Iconic Diagrams\Electric\Sources

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This model represents an ideal current source with a constant current. The current can be set to a certain constant value, the voltage is indifferent. The port *p* of the model has separate high and low terminals. The equations are:

```
p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;

p.u = indifferent;
p.i = i;
```

Interface

Ports	Description
p_low, p_high	Both terminals of the Electric port p.
Causality	
fixed current out	
Parameters	
i	Current [A].

ModulatedCurrentSource

Library

Iconic Diagrams\Electric\Sources

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric), Block Diagrams.

Description

This model represents an ideal current source with a variable current. The current can be set to a (fluctuating) value given by the input signal *i*, the voltage is indifferent.

p.i = i;

Interface

Ports	Description
p	Electric port.
Causality	
fixed current out	
Input	
i	Current [A].

ModulatedVoltageSource

Library

Iconic Diagrams\Electric\Sources

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric), Block Diagrams.

Description

This model represents an ideal voltage source with a variable voltage. The voltage can be set to a (fluctuating) value given by the input signal *u*, the current is indifferent.

p.u = u;

Interface

Ports	Description
p	Electric port.
Causality	

fixed voltage out

Input

u	Voltage [V].

VoltageSource

Library

Iconic Diagrams\Electric\Sources

Implementations

DC
AC

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description - DC

This model represents an ideal voltage source with a constant voltage. The voltage can be set to a certain constant value, the current is indifferent. The port *p* of the model has separate high and low terminals. The equations are:

```
p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;

p.i = indifferent;
p.u = u;
```

Interface - DC

Ports	Description
p	Both terminals of the Electric port p.

Causality

fixed voltage out

Parameters

u	Voltage [V].

Description - AC

This model represents an ideal voltage source with a sinusoidal voltage. The voltage can be set to a certain constant value, the current is indifferent. The port *p* of the model has separate high and low terminals. The equations are:

```
p.i = p_high.i = p_low.i;
p.u = p_high.u - p_low.u;

p.i = indifferent;
p.u = U*sin(2*pi*f*time);
```

Interface - AC

Ports	**Description**
p	Both terminals of the Electric port p.
Causality	
fixed voltage out	
Parameters	
U	Voltage amplitude [V].
f	Voltage frequency [Hz]

9.2.3 Hydraulics

Hydraulics

The 20-sim Hydraulics library contains components to model the dynamic behavior of hydraulic circuits. Most components have lumped volumes to ensure a seamless connection with other components. Therefore almost all components can be connected arbitrarily. Care has to be taken that physically relevant circuits are created.

The elementary components (without lumped volumes) are stored in the basic libraries. These models should only be used by experienced modelers who want to create their own library components.

Hydraulic components can easily be coupled to other library models. The pumps and motors have a rotation port that can be coupled to models of the Rotation library. The cylinder models have a translation port that can be coupled to models of the Translation library. Some models have a variable input that can be coupled to models of the Signal library.

To get a good impression of the use the components of this library, have a look at the example circuits.

Ports

Every library component has one or more hydraulic connectors, which are called ports in 20-sim. Ports always have two variables: pressure [Pa] and volume flow [m3/s]. The following ports a and b are usually provided:

a

a.p	Pressure in [Pa] with respect to atmospheric pressure (p stands for pressure).
a.phi	Volume flow [m3/s], positive if oil is entering the component at port a (phi stands for flow).

b

b.p	Pressure in [Pa] with respect to atmospheric pressure (p stands for pressure).
b.phi	Volume flow [m3/s], negative if oil is entering the component at port b (phi stands for flow).

Pressure

The 20-sim hydraulics library uses the common definition of pressure in Pa:

1 Pa = 1 N/m2 = 1e-5 bar

The air pressure at sea level is taken as the zero value:

0 Pa = air pressure

Consequently the pressure for liquid oil that starts to vaporize is negative. In 20-sim the default value of the vapour pressure is:

p_vapour = -0.999e5 Pa

The vapour pressure is the minimum pressure. Some models have a safeguard on the pressure to prevent it to become smaller. You can change the value of the vapour pressure but take care that always a value is chosen that is smaller than zero. Otherwise some of the models will not work correctly.

Bulk modulus

No liquid is fully incompressible. The compliance characteristics of the oil in a hydraulic system is a vital parameters affecting the response. The effect of compressibility is incorporated by entering the bulk modulus of the fluid:

*dp = -B*dV/V*

Here dp is the pressure , V the volume and B the bulk modulus. For small pressure variations the compressibility effect my be rewritten as:

*p = B/V*int(flow)*

The bulk modulus is a very uncertain parameter. It depends on the percentage of air dissolved in the fluid, the pressure and the temperature. In most textbooks a values of 1.5 to 1.75 [GPa] is used. 20-sim uses a default value of 1.6 [GPa]. If fluid should be simulated at elevated temperatures or with a high percentage or air dissolved, you should use a smaller value for the bulk modulus.

Units

20-sim uses SI-units for the hydraulic components. If you want to use other units select them in the Parameters Editor or the Variables Chooser.

Laminar flow

Many hydraulic models have a laminar (leakage) flow:

*flow = G * dp*

with *G* the flow conductance and *dp* the pressure difference. The table below shows the flow rates for various conductance values and pressure differences of 10 [bar] and 400 [bar].

G $m^3/s.Pa$	P Pa	P bar	flow m^3/s	flow l/min
1.0e-14	1.0e6	10	2.89e-8	0.002
1.0e-14	4.0e7	400	1.82e-7	0.01
1.0e-12	1.0e6	10	1.44e-6	0.09
1.0e-12	4.0e7	400	9.12e-6	0.6
1.0e-10	1.0e6	10	1.44e-4	9
1.0e-10	4.0e7	400	9.12e-4	55
1.0e-8	1.0e6	10	2.89e-3	173
1.0e-8	4.0e7	400	1.82e-2	1095

If the flow rate is very small (leakage flow), choose *G* below 1.0e-14 [m3/s.Pa]. If the flow rate is very large (open connection to a tank), choose *G* 1.0e-9 or higher if desired.

Turbulent flow

Some hydraulic models have a turbulent (orifice) flow described by:

*flow = Cd * A * sqrt(2/ rho * dp)*

with *Cd* the discharge coefficient, *A* the orifice area, *rho* the fluid density and *dp* the pressure difference. Except for small pressure around zero, the discharge coefficient is constant. In most textbooks a value of 0.6 is recommended. The table below shows the flow rates for various orifice areas and pressure differences of 10 [bar] and 400 [bar].

Cd $m^3/s.Pa$	A m^2	rho kg/m^3	dp Pa	dp bar	flow m^3/s	flow l/min
0.6	1.0e-9	865	1.0e6	10	1.0e-8	0.0006
0.6	1.0e-9	865	4.0e7	400	4.0e-7	0.024

0.6	5.0e-8	865	1.0e6	10	1.0e-6	0.06
0.6	5.0e-8	865	4.0e7	400	4.0e-5	2.4
0.6	5.0e-6	865	1.0e6	10	1.0e-4	6
0.6	5.0e-6	865	4.0e7	400	4.0e-3	240
0.6	1.0e-4	865	1.0e6	10	1.0e-2	600
0.6	1.0e-4	865	4.0e7	400	4.0e-1	24000

If the flow rate is very small (leakage flow) choose A smaller than 1.0e-9 [m2]. If the flow rate should be very large (open connection) choose A equal to 1.0e-4 or higher if desired. For very small pressure differences the flow will become laminar and the discharge coefficient will drop. For such cases, specialized models should be applied. If they are not available, use a laminar flow model with a small conductance value.

Simulation

Hydraulic circuits can be simulated with all the available integration methods. In practice the default integration method (BDF) will give the most accurate and quickest response. Take care with hydraulic circuits that will give high pressure peaks and small flow rates. These circuits are sometimes hard to simulate. In practice high pressure peaks should be reduced with pressure relief valves to avoid damage. Fortunately adding pressure relief valves will in most cases improve the simulation response.

Disclaimer

All models have been tested with standard configurations. This will however not ensure that valid results will be found at all times. For example temperature effects are not included and certain parameter values will lead to unstable simulations. Therefore any application of the library without validation of the user is at its own risk!

Literature

The 20-sim hydraulic library has been based on the following literature:

P. Dransfield, Hydraulic Control Systems - Design and Analysisi of Their Dynamics, Springer 1981,ISBN 3-540-10890-4.
P. Beater, Entwurf Hydraulischer Maschinen, Modelbildung, Stabilitätsanalyse und Simulation hydrostatischer Antriebe und Steurungen, Springer 1999, ISBN 3-540-65444-5.

The models have been designed as close as possible to the Modelica hydraulic library (www.modelica.org). To compare both libraries the example model Closed Circuit Drive Train.emx has been added.

Components

HydraulicInertia

Library

Iconic Diagrams\Hydraulics\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

When a fluid flow changes speed, a force is needed to accelerate the fluid. This force is equal to the acceleration multiplied by the mass of the fluid.

$$F = m * dv/dt$$

The fluid mass is equal to the pipe area multiplied by the pipe length and the fluid density:

$$m = A * l * rho$$

The force will result in a pressure differential:

$$F = A * dp = A * (pa.p - pb.p)$$

We can rewrite the formulas as

$$dp = rho * l * dv/dt$$

and by introducing the fluid flow *phi*:

$$v = phi/A$$

we get the formula for an hydraulic inertia:

$$dp = (rho * l / A) * d\ phi/dt$$

Interface

Ports	Description
pa, pb	Both terminals of the hydraulic inertia.

Causality

fixed pressure out
pa

fixed pressure out

pb

Parameters

rho	Mass density of the fluid [kg/m3].
d	pipe diameter [m]
l	pipe length [m]

Node

Library

Iconic Diagrams\Hydraulics\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This node model represents a connection between two or more lines of an hydraulic circuit. The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc.

Interface

Ports	Description
p [any]	Any number of connections can be made.

Cylinders

Basic Cylinders

Cylinder

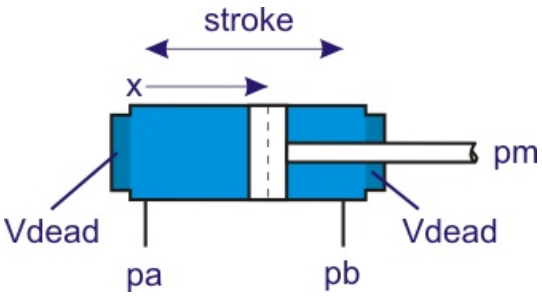
Library

Iconic Diagrams\Hydraulics\Cylinders\Basic Cylinders

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics/Translational).

Description

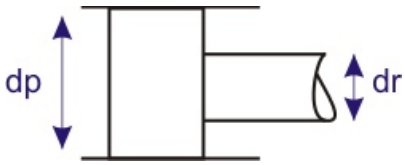


This is an ideal model of cylinder with no mass and no friction. The volume of the chambers is given by:

Va = Vdead + Aa(x + x_initial)*
Vb = Vdead + Ab(stroke - x - x_initial)*

with *Aa* the piston area and *Ab* the effective area at the rod side. *x* is the piston position and *Vdead* the initial volume when piston position is zero. The piston areas *Aa* and *Ab* are related to the piston diameter *dp* and rod diameter *dr* by:

*Aa = pi * dp^2 / 4*
*Ab = pi * dp^2 / 4 - pi * dr^2 / 4*



There is no restriction on the travel of the piston. Only a warning is given when the maximum stroke is exceeded or the piston position gets negative.

Interface

Ports	Description

pa, pb	hydraulic port
pm	translation port

Causality

preferred pressure out pa
preferred force out pm

Parameters

Vdead	rest volume of oil in cylinder chamber when closed [N.s/m]
B	effective bulk modulus [Pa]
stroke	stroke length [m]
x_initial	starting position of piston [m]
dp	piston diameter [m]
dr	rod diameter [m]

CylinderSpringReturn

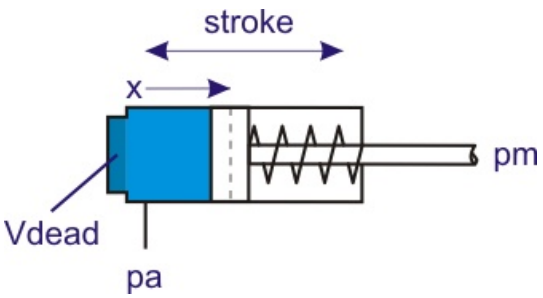
Library

Iconic Diagrams\Hydraulics\Cylinders\Basic Cylinders

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics/Translational).

Description

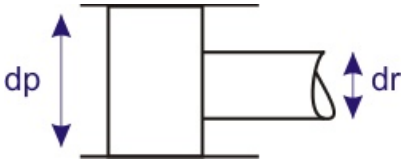


This is an ideal model of a single acting cylinder with no mass and no friction and a return spring to drive the cylinder back when there is no pressure in the chamber. The volume of the chamber is given by:

$$V = Vdead + Aa*(x + x\_initial)$$

with *Aa* the piston area, *x* the piston position and *Vdead* the initial volume when piston position is zero. The piston area *Aa* is related to the piston diameter *dp* by:

*Aa = pi * dp^2 / 4*



The driving force of the return spring is determined by two parameters: the force for piston position zero *(Fspr_min)* and the force when the piston is at the other side *(Fspr_max).* The spring constant *(kspr)* and initial spring position *(x0)* are calculated out of these parameters by:

kspr = (Fspr_max - Fspr_min)/stroke
x0 = -Fspr_min/kspr

There is no restriction on the travel of the piston. Only a warning is given when the maximum stroke is exceeded or the piston position gets negative.

Interface

Ports	**Description**
pa	hydraulic port
pm	translation port
Causality	
preferred pressure out pa	
preferred force out pm	
Parameters	
Vdead	rest volume of oil in cylinder chamber when closed [N.s/m]
B	effective bulk modulus [Pa]
stroke	stroke length [m]
x_initial	starting position of piston [m]
dp	piston diameter [m]
Fspr_min	minimum return spring force (at x = 0) [N]
Fspr_max	maximum return spring force (at x = stroke) [N]

CylinderChamberA

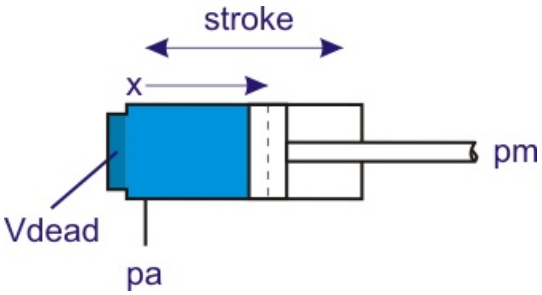
Library

Iconic Diagrams\Hydraulics\Cylinders\Basic Cylinders

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics/Translational).

Description

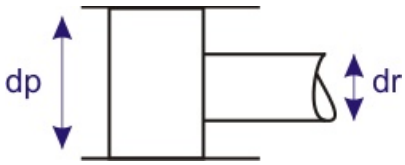


This is an ideal model of a single acting cylinder with no mass and no friction. The volume of the chamber is given by:

$$V = Vdead + Aa*(x + x\_initial)$$

with *Aa* the piston area, *x* the piston position and *Vdead* the initial volume when piston position is zero. The piston area *Aa* is related to the piston diameter *dp* by:

$$Aa = pi * dp^2 / 4$$



There is no restriction on the travel of the piston. Only a warning is given when the maximum stroke is exceeded or the piston position gets negative.

Interface

Ports	Description
pa	hydraulic port
pm	translation port

Causality

preferred pressure out pa

preferred force out pm

Parameters

Vdead	rest volume of oil in cylinder chamber when closed [N.s/
B	m]
stroke	effective bulk modulus [Pa]
x_initial	stroke length [m]
dp	starting position of piston [m]
	piston diameter [m]

CylinderChamberB

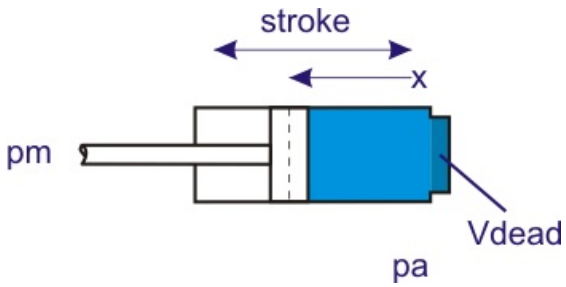
Library

Iconic Diagrams\Hydraulics\Cylinders\Basic Cylinders

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics/Translational).

Description

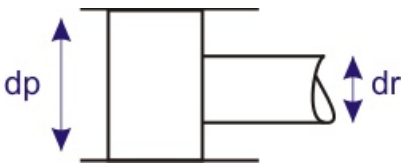


This is an ideal model of a single acting cylinder with no mass and no friction. The volume of the chamber is given by:

V = Vdead + Aa(x + x_initial)*

with *Aa* the piston area, *x* the piston position and *Vdead* the initial volume when piston position is zero. The piston area *Aa* is related to the piston diameter *dp* by:

*Aa = pi * dp^2 / 4*



There is no restriction on the travel of the piston. Only a warning is given when the maximum stroke is exceeded or the piston position gets negative.

Interface

Ports	**Description**
pa	hydraulic port
pm	translation port
Causality	
preferred pressure out pa	
preferred force out pm	
Parameters	
Vdead	rest volume of oil in cylinder chamber when closed [N.s/m]
B	effective bulk modulus [Pa]
stroke	stroke length [m]
x_initial	starting position of piston [m]
dp	piston diameter [m]

CylinderDouble

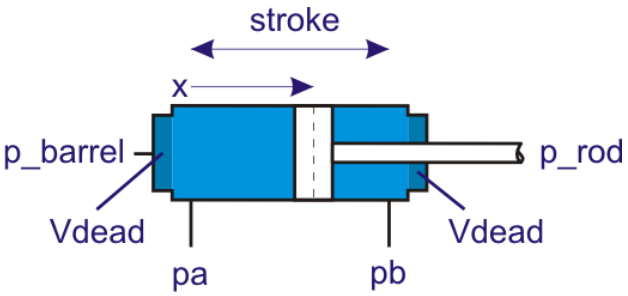
Library

Iconic Diagrams\Hydraulics\Cylinders

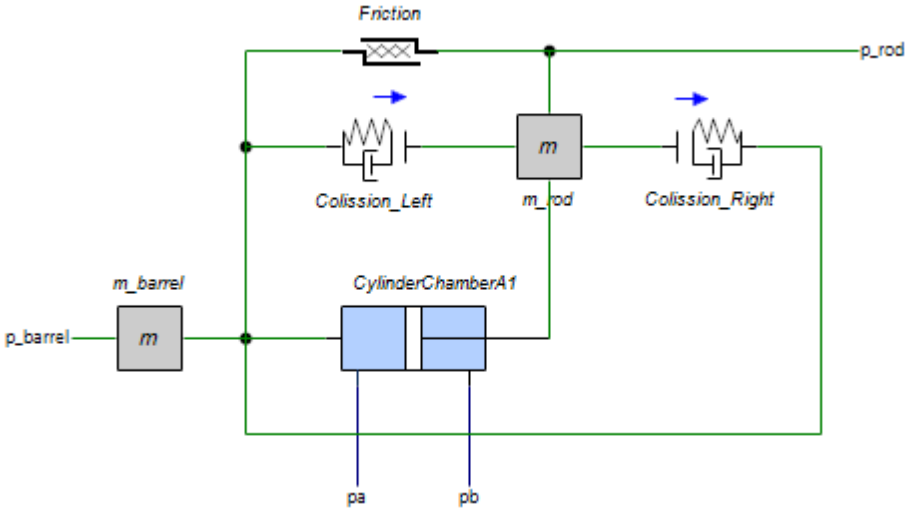
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics/Translational).

Description



This is the extended model of a double acting cylinder with mass, friction and end stops.

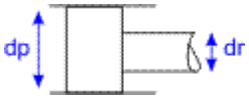


The volume of the chambers is given by:

Va = Vdead + Aa(x + x_initial)*
Vb = Vdead + Ab(stroke - x - x_initial)*

with *Aa* the piston area and *Ab* the effective area at the rod side. *x* is the piston position and *Vdead* the initial volume when piston position is zero. The piston areas *Aa* and *Ab* are related to the piston diameter *dp* and rod diameter *dr* by:

*Aa = pi * dp^2 / 4*
*Ab = pi * dp^2 / 4 - pi * dr^2 / 4*



The travel of the piston is restricted. At the cylinder heads two collision models prevent the piston from traveling any further. The friction between the piston and the cylinder walls is modeled by static and viscous friction.

Interface

Ports	**Description**
pa, pb	hydraulic ports
p_barrel, p_rod	translation ports
Causality	
preferred pressure out pa	
preferred force out pm	
Parameters	
Vdead	rest volume of oil in cylinder chamber when closed, [m3]

B	effective bulk modulus [Pa]
stroke	stroke length [m]
x_initial	starting position of piston [m]
dp	piston diameter [m]
dr	rod diameter [m]
m_barrel	barrel mas [kg]
m_rod	rod and piston mass [kg]
kc	stiffness during collision with cylinder heads [N/m]
dc	damping during collision with cylinder heads [N.s/m]
Fc	static friction [N]
dv	viscous friction coefficient [N.s/m]
slope	steepness of the static friction curve []

Note

When a the cylinder piston collides with the cylinder heads, simulation may get very slow or even become unstable. In these cases you are advised to use the BDF-method with default settings. Try to change the absolute integration error until a stable simulation is obtained!

CylinderSingleSpringReturn

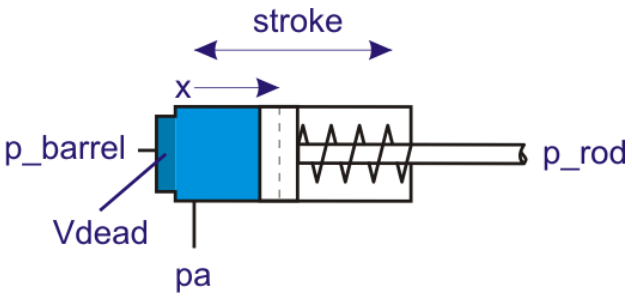
Library

Iconic Diagrams\Hydraulics\Cylinders

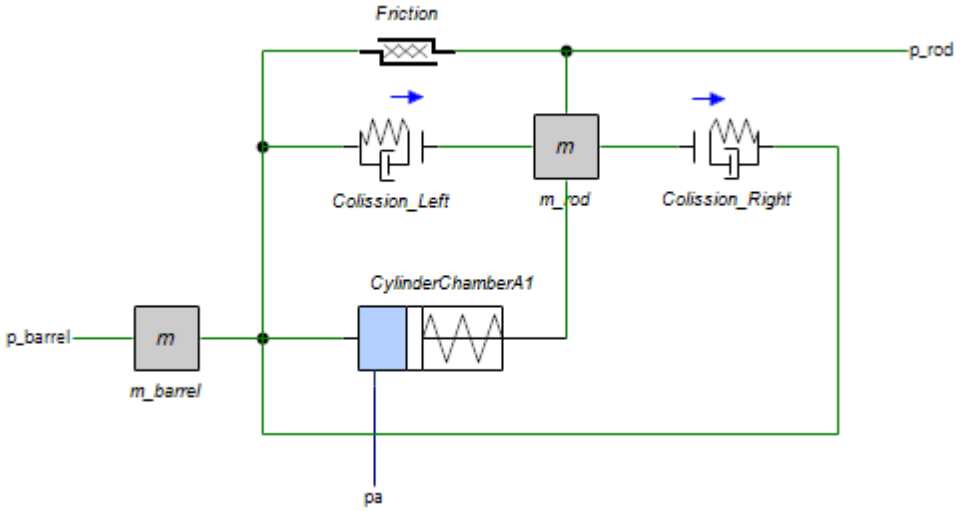
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics/Translational).

Description



This is the extended model of a single acting cylinder with mass, friction, end stops and a return spring to drive the cylinder back when there is no pressure in the chamber.

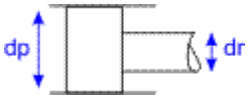


The volume of the chamber is given by:

V = Vdead + Aa(x + x_initial)*

with *Aa* the piston area, *x* the piston position and *Vdead* the initial volume when piston position is zero. The piston area *Aa* is related to the piston diameter *dp* by:

*Aa = pi * dp^2 / 4*



The driving force of the return spring is determined by two parameters: the force for piston position zero *(Fspr_min)* and the force when the piston is at the other side *(Fspr_max)*. The spring constant *(kspr)* and initial spring position *(x0)* are calculated out of these parameters by:

kspr = (Fspr_max - Fspr_min)/stroke
x0 = -Fspr_min/kspr

The travel of the piston is restricted. At the cylinder heads two collision models prevent the piston from traveling any further. The friction between the piston and the cylinder walls is modeled by static and viscous friction.

Interface

Ports	**Description**
pa	hydraulic port
p_barrel, p_rod	translation ports

Causality

preferred pressure out pa

preferred force out pm

Parameters

Vdead	rest volume of oil in cylinder chamber when closed [m3]
B	effective bulk modulus [Pa]
stroke	stroke length [m]
x_initial	starting position of piston [m]
dp	piston diameter [m]
m_barrel	barrel mass [kg]
m_rod	rod and piston mass [kg]
kc	stiffness during collision with cylinder heads [N/m]
dc	damping during collision with cylinder heads [N.s/m]
Fc	static friction [N]
dv	viscous friction coefficient [N.s/m]
slope	steepness of the static friction curve []

Note

When a the cylinder piston collides with the cylinder heads, simulation may get very slow or even become unstable. In these cases you are advised to use the BDF-method with default settings. Try to change the absolute integration error until a stable simulation is obtained!

CylinderSingle

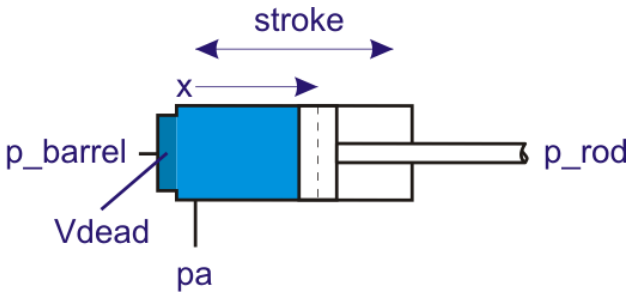
Library

Iconic Diagrams\Hydraulics\Cylinders

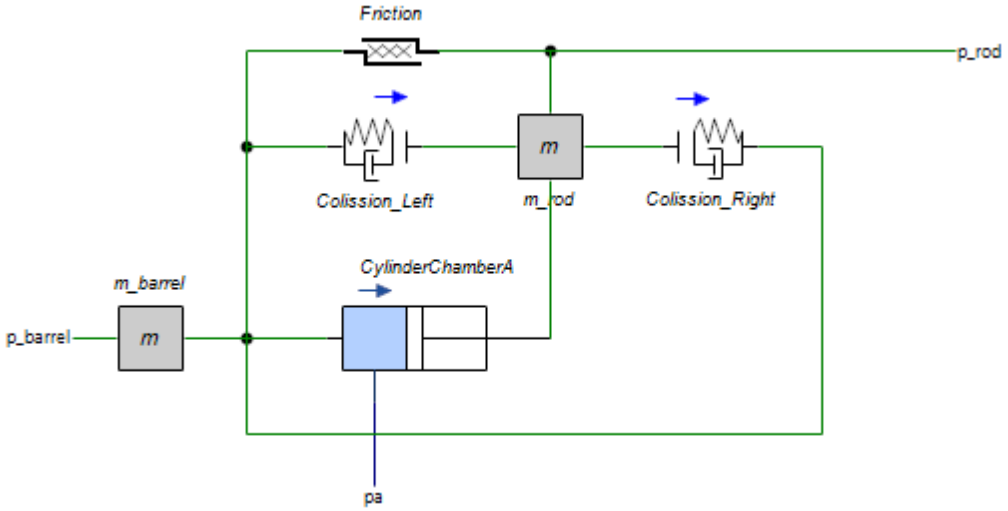
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics/Translational).

Description



This is the extended model of a single acting cylinder with mass, friction and end stops.

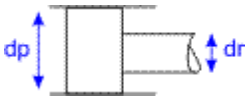


The volume of the chamber is given by:

$$V = Vdead + Aa*(x + x\_initial)$$

with *Aa* the piston area, *x* the piston position and *Vdead* the initial volume when piston position is zero. The piston area *Aa* is related to the piston diameter *dp* by:

$$Aa = pi * dp^2 / 4$$



The travel of the piston is restricted. At the cylinder heads two collision models prevent the piston from traveling any further. The friction between the piston and the cylinder walls is modeled by static and viscous friction.

Interface

Ports	**Description**
pa	hydraulic port
p_barrel, p_rod	translation ports
Causality	
preferred pressure out pa	
preferred force out pm	
Parameters	
Vdead	rest volume of oil in cylinder chamber when closed [m3]
B	effective bulk modulus [Pa]
stroke	stroke length [m]

x_initial	starting position of piston [m]
dp	piston diameter [m]
m_barrel	barrel mass [kg]
m_rod	rod and piston mass [kg]
kc	stiffness during collision with cylinder heads [N/m]
dc	damping during collision with cylinder heads [N.s/m]
Fc	static friction [N]
dv	viscous friction coefficient [N.s/m]
slope	steepness of the static friction curve []

Note

When a the cylinder piston collides with the cylinder heads, simulation may get very slow or even become unstable. In these cases you are advised to use the BDF-method with default settings. Try to change the absolute integration error until a stable simulation is obtained!

Pumps

Basic Pumps

DisplacementMotor

Library

Iconic Diagrams\Hydraulics\Pumps\Basic Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes an ideal motor with an axial speed that is proportional to the input flow rate:

```
pa.phi = pb.phi = i * p_rot.omega;
i = D / (2*pi);
```

The torque is equal to:

```
p.T = i*(pa.p - pb.p);
```

If the port pressure is smaller than the vapour pressure (p < p_vapour), the flow is zero. If the port pressure becomes larger than the vapour pressure the flow gradually grows to its normal value, until the atmospheric pressure (p = 0) is reached.

Interface

Ports	Description
pa	inlet port (hydraulic)

pb	outlet port (hydraulic)
p_rot	axis (rotation)

Causality

fixed volume flow out pa

fixed volume flow out pb

fixed torque out p_rot

Parameters

D	displacement per revolution [m3]

DisplacementPump

Library

Iconic Diagrams\Hydraulics\Pumps\Basic Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes an ideal pump with a displacement that is proportional to the speed of the input axis:

```
pa.phi = pb.phi = i * p_rot.omega;
i = D / (2*pi);
```

The torque is equal to:

```
p.T = i*(pa.p - pb.p);
```

If the inlet pressure is smaller than the vapour pressure(p < p_vapour), the flow is zero. If the inlet pressure is larger than the vapour pressure the flow gradually grows to its normal value, until the atmospheric pressure (p = 0) is reached.

Interface

Ports	Description
pa	inlet port (hydraulic)
pb	outlet port (hydraulic)
p_rot	axis (rotation)

Causality

fixed volume flow out pa

fixed volume flow
out pb
preferred angular
velocity out p_rot

Parameters

D	displacement per revolution [m3]

FlowSource

Library

Iconic Diagrams\Hydraulics\Pumps\Basic Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model represents an ideal flow source with a fixed flow given by the parameter phi:

p.phi = phi;

The model has no leakage.

Interface

Ports	**Description**
p	

Causality

fixed volume flow
out p

Parameters

phi	Volume flow [m3/s]

ModulatedFlowSource

Library

Iconic Diagrams\Hydraulics\Pumps\Basic Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model represents an ideal flow source with a variable flow that is given by input signal *phi:*

p.phi = phi;

The model has no leakage.

Interface

Ports	**Description**
p	
Causality	
fixed volume flow	
out p	
Input	
phi	Volume flow [m3/s]

ModulatedPressureSource

Library

Iconic Diagrams\Hydraulics\Pumps\Basic Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model represents an ideal pressure source with a fixed pressure given by the parameter pressure:

p.p = pressure;

The model has no leakage.

Interface

Ports	**Description**
p	
Causality	
fixed pressure out	
Input	
pressure	Pressure [Pa]

PressureSource

Library

Iconic Diagrams\Hydraulics\Pumps\Basic Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model represents an ideal pressure source with a fixed pressure given by the parameter pressure:

```
p.p = pressure;
```

The model has no leakage.

Interface

Ports	**Description**
p	
Causality	
fixed pressure out	
Parameters	
p	Pressure [Pa]

VariableDisplacementMotor

Library

Iconic Diagrams\Hydraulics\Pumps\Basic Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes an ideal motor with an axial speed that is proportional to the input flow rate:

```
pa.phi = pb.phi = i * p_rot.omega;
i = D * c / (2*pi);
```

The torque is equal to:

```
p.T = i*(pa.p - pb.p);
```

The displacement is controllable by the input signal *c*. For a positive rotation of the driving axis the flow is:

c >= 1 *maximum flow from port 1 to port 2*

c = 0	*zero flow*
c <= -1	*maximum flow from port 2 to port 1*

If the port pressure is smaller than the vapour pressure (p < p_vapour), the flow is zero. If the port pressure becomes larger than the vapour pressure the flow gradually grows to its normal value, until the atmospheric pressure (p = 0) is reached.

Interface

Ports	**Description**
pa	inlet port (hydraulic)
pb	outlet port (hydraulic)
p_rot	axis (rotation)

Causality

fixed volume flow out pa

fixed volume flow out pb

fixed torque out p_rot

Input

c	relative displacement

Parameters

D	displacement per revolution [m3]

VariableDisplacementPump

Library

Iconic Diagrams\Hydraulics\Pumps\Basic Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes an ideal pump with a displacement that is proportional to the speed of the input axis:

```
pa.phi = pb.phi = i * p_rot.omega;
i = D * c / (2*pi);
```

The torque is equal to:

```
p.T = i*(pb.p - pa.p);
```

The displacement is controllable by the input signal *c*. For a positive rotation of the driving axis the flow is:

c >= 1	*maximum flow from port 1 to port 2*
c = 0	*zero flow*
c <= -1	*maximum flow from port 2 to port 1*

If the port pressure is smaller than the vapour pressure (p < p_vapour), the flow is zero. If the port pressure is larger than the vapour pressure the flow gradually grows to its normal value, until the atmospheric pressure (p = 0) is reached.

Interface

Ports	**Description**
pa	inlet port (hydraulic)
pb	outlet port (hydraulic)
p_rot	axis (rotation)
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
fixed torque out p_rot	
Input	
c	relative displacement
Parameters	
D	displacement per revolution [m3]

CentrifugalPump

Library

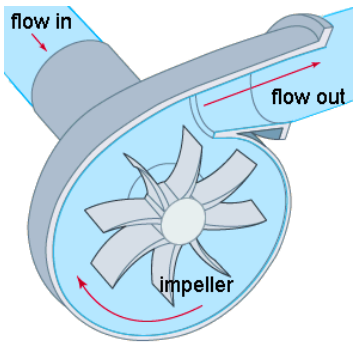
Iconic Diagrams\Hydraulics\Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

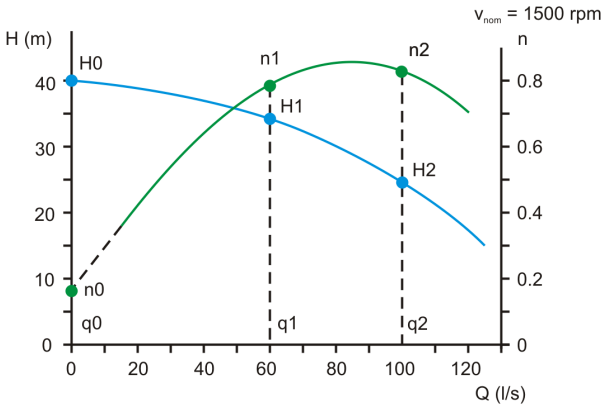
Description

A centrifugal pump converts the input power of a rotating shaft into kinetic energy of a the liquid by accelerating the liquid through an impeller.



The pump is characterized by two curves:

1. The head (*H*) as function of the volume flow at a preset speed.
2. The efficiency (*n*) of the pump as a function of the fluid flow at a present speed. The efficiency is defined as the flow output power divided by rotational input power.



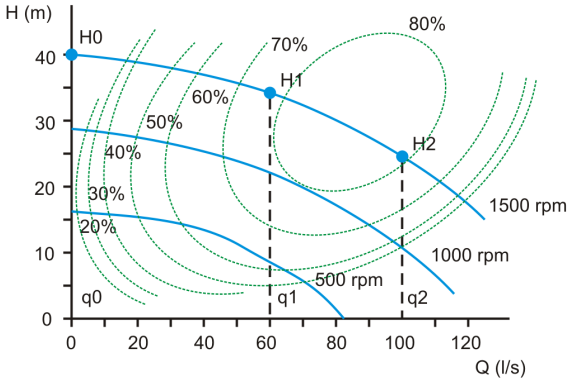
The pump curves are approximated by two second order polynomials:

$$H = ha*q^2 + hb*q + hc$$
$$n = na*q^2 + nb*q + nc$$

To fit the polynomial curves, the heads and efficiencies have to be specified at three different flows. The first flow (*q0*) is zero. The other two flows (*q1* and *q2*) can be chosen arbitrary.

Note: some pump flows extrapolate the efficiency curves to zero at zero fluid flow. This would mean that at a zero fluid flow no power can be transmitted and the pump can

never start! Therefore in this model always use an efficiency that is non-zero at zero flow.



Some pump characteristics give multiple input speeds. This model automatically takes into account for varying input speeds of the input axis by transforming the pump curve using the affinity rules.

$$H_a = H_b*(v_b/v_a)^2$$
$$Q_a = Q_b*(v_b/v_a)$$

Some pump characteristics give the efficiencies as regions. You have then have to estimate the efficiencies at the chosen flows *q0*, *q1* and *q2*.

Interface

Ports	**Description**
p_t	inlet port, tank (hydraulic)
p_a	outlet port (hydraulic)
p_rot	axis (rotation)
Causality	
velocity out p_rot	
Input	
phi	Volume flow [m3/s]
Parameters	
q1	flow 1 (choose an arbitrary point between zero and nominal)
q2	flow 2 (nominal flow)
H0	head at zero flow

H1	head at flow 1
H2	head at flow 2 (nominal flow)
n0	efficiency (0 < n < 1) at zero flow
n1	efficiency (0 < n < 1) at flow 1
n2	efficiency (0 < n < 1) at flow 2 (nominal flow)
v_nom	nominal speed of the pump
rho	fluid density

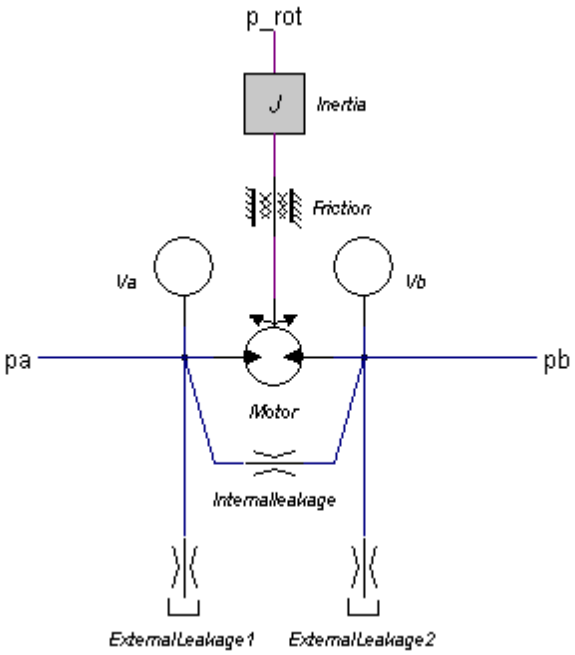
DisplacementMotor-Leakage

Library

Iconic Diagrams\Hydraulics\Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).



Description

This model describes an motor with internal and external leakage and an axial speed that is proportional to the input flow rate:

```
pa.phi = pb.phi = i * p_rot.omega;
i = D / (2*pi);
```

The actual flows at the inlet and outlet port may be slightly different because of the flow into the lumped volumes and the leakage flows. The leakage flows are modeled by laminar resistances. The torque is equal to:

```
p.T = i*(pa.p - pb.p);
```

If the port pressure is smaller than the vapour pressure (p < p_vapour), the flow is zero. If the port pressure is larger than the vapour pressure the flow gradually grows to its normal value, until the atmospheric pressure (p = 0) is reached.

Interface

Ports	**Description**
pa	inlet port (hydraulic)
pb	outlet port (hydraulic)
p_rot	axis (rotation)
Causality	
preferred pressure out pa	
preferred pressure out pa	
preferred angular velocity out p_rot	
Parameters	
ExternalLeakage1\p_preload	Tank pressure [Pa]
ExternalLeakage1\G	Conductance of laminar resistance [m3/s.Pa]
ExternalLeakage2\p_preload	Tank pressure [Pa]
ExternalLeakage2\G	Conductance of laminar resistance [m3/s.Pa]
InternalLeakage\G	Conductance of laminar resistance [m3/s.Pa]
Va\V	Volume of oil under pressure [m3]
Va\B	Effective bulk modulus [Pa]
Va\p_initial	The starting pressure of the volume [Pa]
Vb\V	Volume of oil under pressure [m3]
Vb\B	Effective bulk modulus [Pa]
Vb\p_initial	The starting pressure of the volume [Pa]
Motor\D	Displacement per revolution [m3]
Friction\d	Damping [N.m.s/rad]
Inertia\J	Moment of inertia [kg.m2]

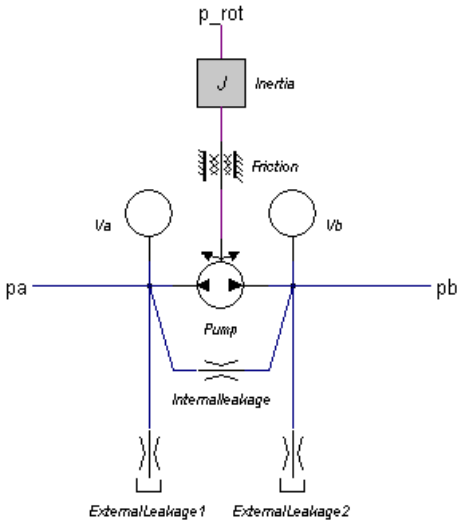
DisplacementPump-Leakage

Library

Iconic Diagrams\Hydraulics\Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).



Description

This model describes a pump with internal and external leakage and a displacement that is proportional to the speed of the input axis:

```
i = D * c / (2*pi);
pa.phi = pb.phi = i * p_rot.omega;
```

The actual flows at the inlet and outlet port may be slightly different because of the flow into the lumped volumes and the leakage flows. The leakage flows are modeled by laminar resistances. The torque of the pump is equal to:

```
p.T = i*(pa.p - pb.p);
```

If the inlet pressure is smaller than the vapour pressure(p < p_vapour), the flow is zero. If the inlet pressure becomes larger than the vapour pressure the flow gradually grows to its normal value, until the atmospheric pressure (p = 0) is reached.

Interface

Ports	Description
pa	inlet port (hydraulic)

pb	outlet port (hydraulic)
p_rot	axis (rotation)

Causality

preferred pressure out pa
preferred pressure out pa
preferred angular velocity out
p_rot

Parameters

ExternalLeakage1\p_preload	Tank pressure [Pa]
ExternalLeakage1\G	Conductance of laminar resistance [m3/s.Pa]
ExternalLeakage2\p_preload	Tank pressure [Pa]
ExternalLeakage2\G	Conductance of laminar resistance [m3/s.Pa]
InternalLeakage\G	Conductance of laminar resistance [m3/s.Pa]
Va\V	Volume of oil under pressure [m3]
Va\B	Effective bulk modulus [Pa]
Va\p_initial	The starting pressure of the volume [Pa]
Vb\V	Volume of oil under pressure [m3]
Vb\B	Effective bulk modulus [Pa]
Vb\p_initial	The starting pressure of the volume [Pa]
Pump\D	Displacement per revolution [m3]
Friction\d	Damping [N.m.s/rad]
Inertia\J	Moment of inertia [kg.m2]

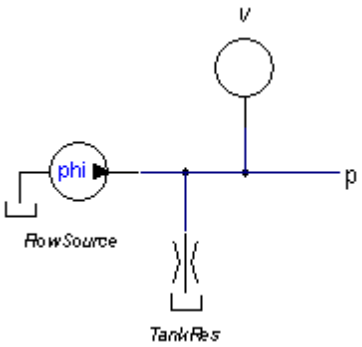
FlowSource-Leakage

Library

Iconic Diagrams\Hydraulics\Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).



Description

This model represents a flow source with a fixed flow and leakage. The flow is given by the parameter *phi:*

*p.phi = phi - G*p.p;*

The actual flow at the outlet port may be slightly different because of the flow into the lumped volume at the outlet port and the leakage flow.

Interface

Ports	**Description**
p	
Causality	
fixed volume flow out	
p	
Input	
phi	Volume flow [m3/s]
Parameters	
FlowSource\phi	Volume flow [m3/s]
TankRes\p_preload	Tank pressure [Pa]
TankRes\G	Conductance of laminar resistance [m3/s.Pa]
V\V	Volume of oil under pressure [m3]
V\B	Effective bulk modulus [Pa]

V\p_initial	The starting pressure of the volume [Pa]

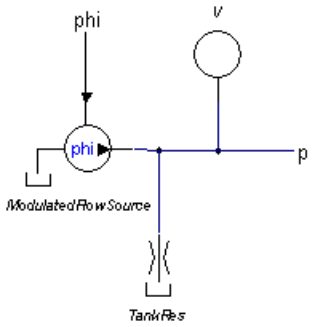
ModulatedFlowSource-Leakage

Library

Iconic Diagrams\Hydraulics\Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).



Description

This model represents a flow source with a variable flow and leakage. The variabele flow is given by an input signal:

*p.phi = phi - G*p.p;*

The actual flow at the outlet port may be slightly different because of the flow into the lumped volume at the outlet port and the leakage flow.

Interface

Ports	**Description**
p	
Causality	
fixed volume flow out	
p	
Input	
phi	Volume flow [m3/s]
Parameters	
TankRes\p_preload	tank pressure [Pa]
TankRes\G	Conductance of laminar resistance [m3/s.Pa]
V\V	Volume of oil under pressure [m3]
V\B	Effective bulk modulus [Pa]

V\p_initial The starting pressure of the volume [Pa]

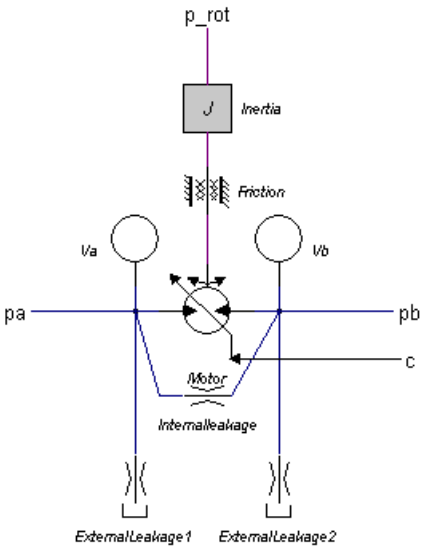
VariableDisplacementMotor-Leakage

Library

Iconic Diagrams\Hydraulics\Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).



Description

This model describes an motor with internal and external leakage and an axial speed that is proportional to input flow rate:

```
pa.phi = pb.phi = i * p_rot.omega;
i = D / (2*pi);
```

The actual flows at the inlet and outlet port may be slightly different because of the flow into the lumped volumes and the leakage flows. The leakage flows are modeled by laminar resistances. The torque is equal to:

```
p.T = i*(pa.p - pb.p);
```

The displacement is controllable by the input signal *c*. For a positive rotation of the driving axis the flow is:

c >= 1 *maximum flow from port 1 to port 2*

c = 0	*zero flow*
c <= -1	*maximum flow from port 2 to port 1*

If the port pressure is smaller than the vapour pressure (p < p_vapour), the flow is zero. If the port pressure is larger than the vapour pressure the flow gradually grows to its normal value, until the atmospheric pressure (p = 0) is reached.

Interface

Ports	**Description**
pa	inlet port (hydraulic)
pb	outlet port (hydraulic)
p_rot	axis (rotation)
Causality	
preferred pressure out pa	
preferred pressure out pa	
preferred angular velocity out p_rot	
Input	
c	relative displacement
Parameters	
ExternalLeakage1\p_preload	Tank pressure [Pa]
ExternalLeakage1\G	Conductance of laminar resistance [m3/s.Pa]
ExternalLeakage2\p_preload	Tank pressure [Pa]
ExternalLeakage2\G	Conductance of laminar resistance [m3/s.Pa]
InternalLeakage\G	Conductance of laminar resistance [m3/s.Pa]
Va\V	Volume of oil under pressure [m3]
Va\B	Effective bulk modulus [Pa]
Va\p_initial	The starting pressure of the volume [Pa]
Vb\V	Volume of oil under pressure [m3]
Vb\B	Effective bulk modulus [Pa]
Vb\p_initial	The starting pressure of the volume [Pa]
Motor\D	Displacement per revolution [m3]
Friction\d	Damping [N.m.s/rad]
Inertia\J	Moment of inertia [kg.m2]

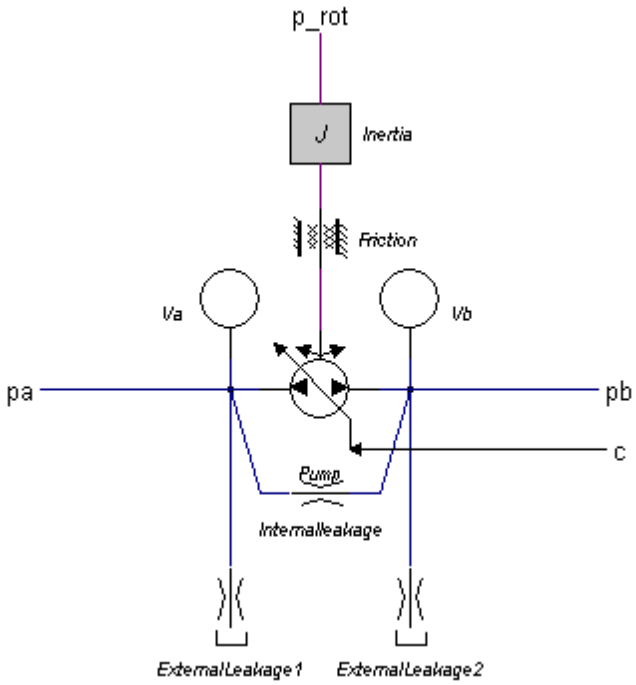
VariableDisplacementPump-Leakage

Library

Iconic Diagrams\Hydraulics\Pumps

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).



Description

This model describes a pump with internal and external leakage and a displacement that is proportional to the speed of the input axis:

```
i = D * c / (2*pi);
pa.phi = pb.phi = i * p_rot.omega;
```

The actual flows at the inlet and outlet port may be slightly different because of the flow into the lumped volumes and the leakage flows. The leakage flows are modeled by laminar resistances. The torque of the pump is equal to:

```
p.T = i*(pa.p - pb.p);
```

The displacement is controllable by the input signal c. For a positive rotation of the driving axis the flow is:

c >= 1	*maximum flow from port 1 to port 2*

c = 0	*zero flow*
c <= -1	*maximum flow from port 2 to port 1*

If the port pressure is smaller than the vapour pressure (p < p_vapour), the flow is zero. If the port pressure becomes larger than the vapour pressure the flow gradually grows to its normal value, until the atmospheric pressure (p = 0) is reached.

Interface

Ports	**Description**
pa	inlet port (hydraulic)
pb	outlet port (hydraulic)
p_rot	axis (rotation)
Causality	
preferred pressure out pa	
preferred pressure out pa	
preferred angular velocity out p_rot	
Input	
c	relative displacement
Parameters	
ExternalLeakage1\p_preload	Tank pressure [Pa]
ExternalLeakage1\G	Conductance of laminar resistance [m3/s.Pa]
ExternalLeakage2\p_preload	Tank pressure [Pa]
ExternalLeakage2\G	Conductance of laminar resistance [m3/s.Pa]
InternalLeakage\G	Conductance of laminar resistance [m3/s.Pa]
Va\V	Volume of oil under pressure [m3]
Va\B	Effective bulk modulus [Pa]
Va\p_initial	The starting pressure of the volume [Pa]
Vb\V	Volume of oil under pressure [m3]
Vb\B	Effective bulk modulus [Pa]
Vb\p_initial	The starting pressure of the volume [Pa]
Pump\D	Displacement per revolution [m3]
Friction\d	Damping [N.m.s/rad]
Inertia\J	Moment of inertia [kg.m2]

Restrictions

LaminarResistance

Library

Iconic Diagrams\Hydraulics\Restrictions

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes the laminar flow through a component:

```
pa.phi = pb.phi = G*(pa.p - pb.p);
```

The pressure at both ports has a lower limit which is equal to the vapour pressure. Therefore the actual equations used in this component are:

```
p1_lim = if pa.p < p_vapour then p_vapour else pa.p end;
p2_lim = if pb.p < p_vapour then p_vapour else pb.p end;
dp = p1_lim - p2_lim;
pa.phi = G*dp;
pb.phi = pa.phi;
```

There is no check on the validity of laminar flow in this component!

Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
Parameters	
G	Conductance of laminar resistance [m3/s.Pa], G >= 0!.

Orifice

Library

Iconic Diagrams\Hydraulics\Restrictions

Use

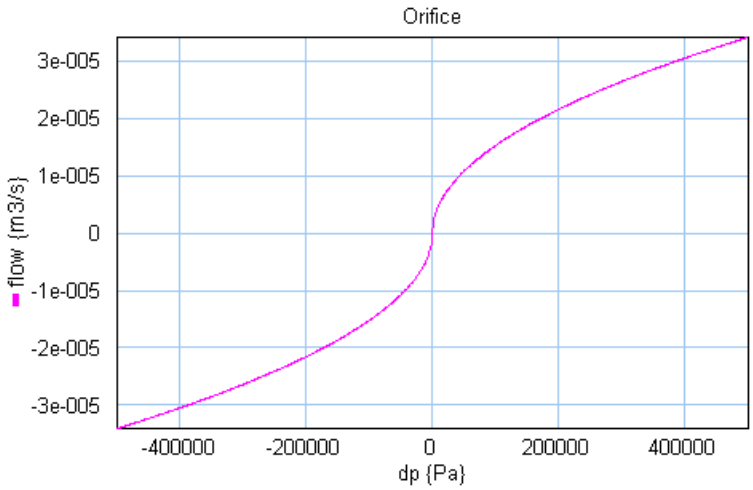
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes the laminar/turbulent flow through an orifice if no cavitation occurs. The flow depends on the pressure difference:

```
dp = pa.p - pb.p
phi = sign(dp) * Cd * A * sqrt( (2/rho) * abs(dp) ) + GLeak * dp;
```

Here *Cd* is the discharge coefficient which normally has a value between 0.55 and 0.7 and *A* is the area of the orifice opening. *Gleak* is the conductance of laminar leakage flow when the valve is closed.



The pressure at both ports has a lower limit that is equal to the vapour pressure. Therefore the actual equations used in this component are:

```
p1_lim = if pa.p < p_vapour then p_vapour else pa.p end;
p2_lim = if pb.p < p_vapour then p_vapour else pb.p end;
dp = p1_lim - p2_lim;
```

Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
Parameters	
rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0!

A	Orifice Area [m2], A >= 0!
Cd	Discharge coefficient [].

VariableLaminarResistance

Library

Iconic Diagrams\Hydraulics\Restrictions

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes the laminar flow through a component:

```
pa.phi = pb.phi = G*(pa.p - pb.p);
```

The variable conductance G is given by an input signal. The pressure at both ports has a lower limit that is equal to the vapour pressure. Therefore the actual equations used in this component are:

```
p1_lim = if pa.p < p_vapour then p_vapour else pa.p end;
p2_lim = if pb.p < p_vapour then p_vapour else pb.p end;
dp = p1_lim - p2_lim;
pa.phi = G*dp;
pb.phi = pa.phi;
```

There is no check on the validity of laminar flow in this component!

Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
Input	
G	Conductance of laminar resistance [m3/s.Pa], G >= 0.

VariableOrifice

Library

Iconic Diagrams\Hydraulics\Restrictions

Use

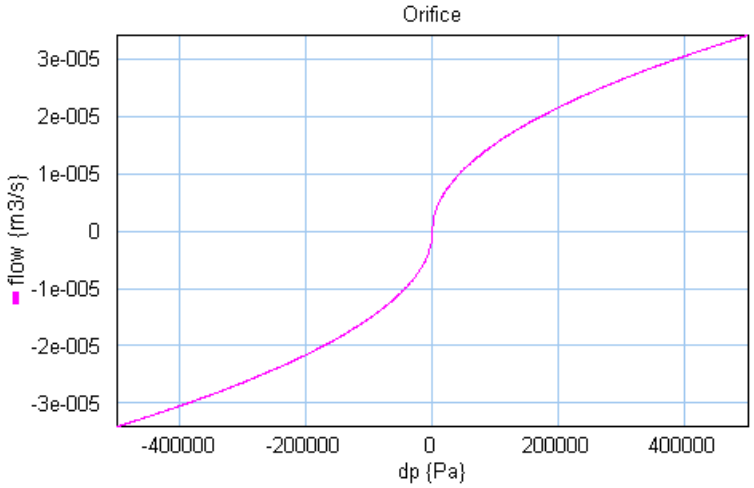
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes the laminar/turbulent flow through an orifice if no cavitation occurs. The flow depends on the pressure difference:

```
dp = pa.p - pb.p;
phi = sign(dp) * Cd * A * sqrt( (2/rho) * abs(dp) ) + GLeak * dp;
```

Here *Cd* is the discharge coefficient that normally has a value between 0.55 and 0.7 and *A* is the area of the orifice opening. The area *A* is given as an input signal.



The pressure at both ports has a lower limit that is equal to the vapour pressure. Therefore the actual equations used in this component are:

```
p1_lim = if pa.p < p_vapour then p_vapour else pa.p end;
p2_lim = if pb.p < p_vapour then p_vapour else pb.p end;
dp = p1_lim - p2_lim;
```

Interface

Ports	Description
pa, pb	Both terminals of the hydraulic component.

Causality

fixed volume flow

out pa

fixed volume flow	
out pb	
Input	
A	Orifice Area [m2], A >= 0.
Parameters	
rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >=0!
Cd	Discharge coefficient [], Cd >=0!

Sensors

FlowSensor

Library

Iconic Diagrams\Hydraulics\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model gives the measured flow as an output signal. The port p of the model has separate high and low terminals. The equations are:

```
p.phi = p_high.phi = p_low.phi;
p.p = p_high.p - p_low.p;
p.p = 0;
phi = p.phi;
```

The model has no leakage.

Interface

Ports	**Description**
p	
Causality	
fixed volume flow	
out	
Output	
phi	Volume flow [m3/s]

HeadSensor

Library

Iconic Diagrams\Hydraulics\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This sensor gives the pressure in head:

*H = p / (rho * g)*

with *rho* the density of the fluid and *g* the specific gravity. The head is the rise of fluid in meters if an open tube would be connected. The figure below shows an example for water at 1 bar and 20 degrees (rho = 998 kg/m3) giving a head of 10.2 m.

Interface

Ports	**Description**
p	
Causality	
indifferent	
Parameters	
rho	Mass density of the fluid [kg/m3].

PowerSensor

Library

Iconic Diagrams\Hydraulics\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model gives the power that is tranferred over a hydraulic line:

```
p_high.p = p_low.p;
p_high.phi = p_low.phi;
P = p_high.phi*p_high.p;
```

The model has no leakage.

Interface

Ports	Description
p	
Causality	
indifferent	
Output	
P	Power [W].

PressureSensor

Library

Iconic Diagrams\Hydraulics\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model gives the measured pressure as an output signal:

```
pressure = p.p;
p.phi = 0;
```

The model has no leakage.

Interface

Ports	Description
p	
Causality	
fixed volume flow	
out	
Output	
pressure	Absolute velocity [m/s].

Valves

Basic Valves

CheckValve

Library

Iconic Diagrams\Hydraulics\Valves\Basic Valves

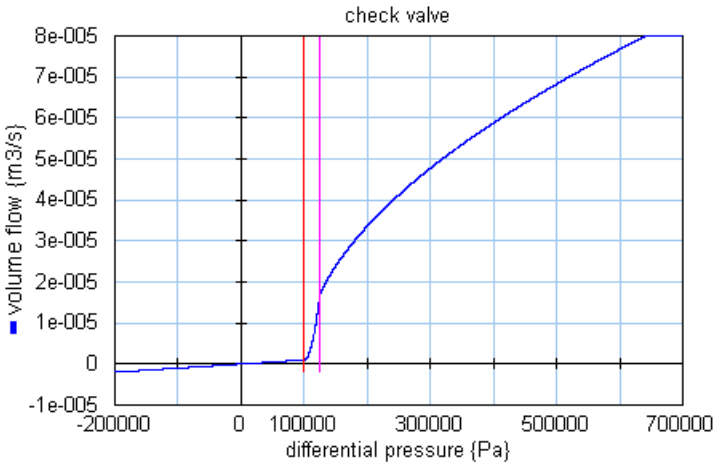
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes a spring-loaded check valve. The resistance depends on the pressure difference:

*dp < pclosed => valve closed, only leakage: pa.phi = pb.phi = dp * GLeak*
pclosed < dp < popen => working range, i. e. valve partially opened
*popen < dp => valve wide open: pa.phi = pb.phi = Cd * A * sqrt(dp - pclosed);*



Interface

Ports	Description
pa, pb	Both terminals of the hydraulic component.

Causality

fixed volume flow out pa
fixed volume flow out pb

Parameters

pclosed	Valve is closed under this pressure [Pa].
popen	Valve is fully open above this pressure [Pa].
GLeak	Conductance of closed valve [m3/s.Pa].
A	Maximum valve area when the valve is open [m2], A >= 0!
Cd	Discharge coefficient [].

FlowControlValve

Library

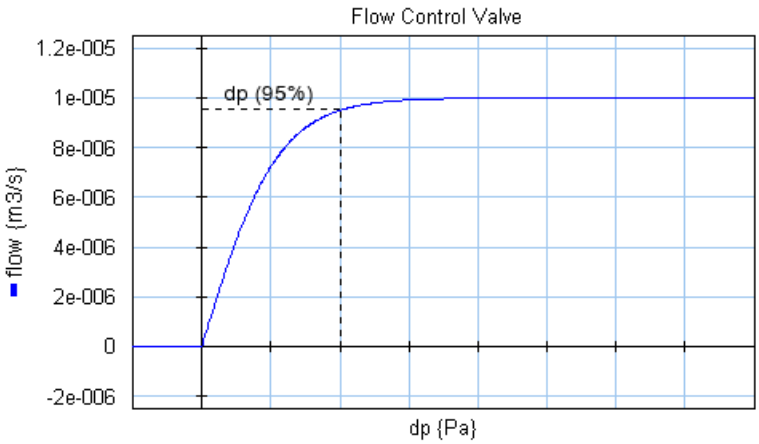
Iconic Diagrams\Hydraulics\Valves\Basic Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

Flow control valves have the purpose to provide a constant flow, independent of the downstream pressure. The flow is obtained by means of a pressure difference controller using the pressure drop over an orifice. Therefore flow control valves always need a certain pressure drop before the desired flow can be achieved.



In this model the flow is modeled by an tanh function. The pressure drop is defined as the pressure where 95% of the desired flow rate is achieved.

```
dp = pa.p - pb.p
phi = if dp > 0 then
  phi*arctanh( (arctan(0.95)/p_drop) * dp) + GLeak * dp;
else
  GLeak * dp
end;
```

Here *phi* is the desired flow and *p_drop* is the 95% pressure drop. *Gleak* is the conductance of laminar leakage flow when the valve is closed. The pressure at both ports has a lower limit that is equal to the vapour pressure. Therefore the actual equations used in this model are:

```
p1_lim = if pa.p < p_vapour then p_vapour else pa.p end;
p2_lim = if pb.p < p_vapour then p_vapour else pb.p end;
dp = p1_lim - p2_lim;
```

Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
Parameters	
phi	Desired flow [m3/s]
p_drop	pressure drop at 95% flow [Pa]
GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0!

FourThreeWayDirectionalValve

Library

Iconic Diagrams\Hydraulics\Valves\Basic Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model is equal to the four three way proportional valve model, except for the spool position. The spool position is rounded to 0, 1 or -1:

```
-0.5 < spoolpos < 0.5 => pos = 0
spoolpos >= 0.5 => pos = 1
spoolpos <= -0.5 => pos = -1
```

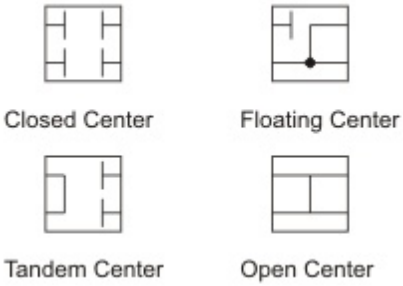
The spool dynamics is modeled by a second order transfer function:

```
sp = SO(f,d,discrete(pos))
```

which is characterized by the bandwidth *(f)* and damping *(d)*.

Implementation

The 4/3-way directional control valve is implemented with various spool centre configurations. The configurations are shown in the picture below:



When you drag and drop a model in the editor, you will be asked which implementation you want to choose. During modeling you can easily change the spool center configuration:

1. Select the valve model.
2. Click the right mouse button to open the right mouse menu.
3. Click *Edit Implementation* and choose another implementation.

Interface

Ports	**Description**
pp, pt, pa, pb,	All terminals of the valve.
Causality	
fixed volume flow out pp	
fixed volume flow out pt	
fixed volume flow out pa	
fixed volume flow out pb	
Inputs	
spoolpos	position of the spool valve: 0 =closed, 1 = open 0 <= spoolpos <= 1
Parameters	
rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flows [m3/s.Pa], GLeak >= 0.
Amax	Maximum spool valve areas when the valves are open [m2], Amax >
Cd	0.

overlap	Discharge coefficient [], Cd > 0.
f	The valve overlaps in closed position [], -1 < overlap < 1.
d	Bandwidth of the spool dynamics [Hz], f > 0.
	Damping of the spool dynamics [], d > 0

FourThreeWayProportionalValve

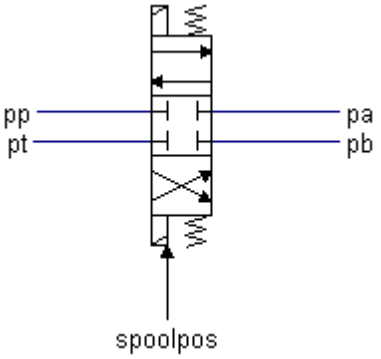
Library

Iconic Diagrams\Hydraulics\Valves\Basic Valves

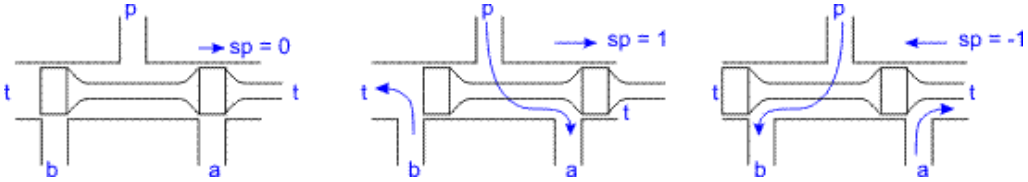
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description



This model describes a 4/3-way proportional control valve with second order spool dynamics.



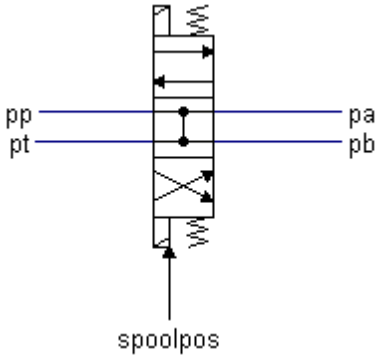
The spool position is indicated by the variable *sp:*

sp = -1	*Flow from p to b and a to t*
-1 < sp < 0	*Partial flow from p to b and a to t*
sp = 0	*No flow*
0 < sp < 1	*Partial flow from p to a and b to t*

sp = 1 *Flow from p to a and b to t*

The flow through the valves is described as laminar/turbulent flow through an orifice. A detailed description of the flow equations can be found in the description of the model TwoTwoWayDirectionalValve.

In the neutral spool position (sp = 0) all valves are closed. A positive overlap indicates that the spool must travel a certain distance before the valves open. A negative overlap indicates that the valves are already open in the neutral position. A 4/3-way directional control valve with negative overlap could therefore also be indicated by the figure below.



The overlap is indicated by the parameter *overlap,* which is given as a fraction of the spool position. Note that overlap is only active in the neutral position. For the spool position equal to 1 or -1 the valves are completely open or completely closed.

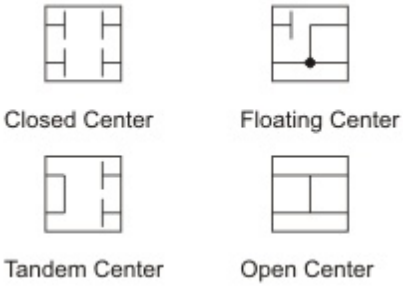
The spool position *sp* is a function of the input signal *spoolpos*:

sp = SO(f,d,discrete(spoolpos))

where *SO* is a second order transfer function to model the spool dynamics. The function is characterized by the bandwidth (*f*) and damping (*d*). The valve input position *spoolpos* should be limited to the range between -1 and 1.

Implementation

The 4/3-way proportional control valve is implemented with various spool centre configurations. The configurations are shown in the picture below:



When you drag and drop a model in the editor, you will be asked which implementation you want to choose. During modeling you can easily change the spool center configuration:

1. Select the valve model.
2. Click the right mouse button to open the right mouse menu.
3. Click *Edit Implementation* and choose another implementation

Interface

Ports	**Description**
pp, pt, pa, pb,	All terminals of the valve.
Causality	
fixed volume flow out pp	
fixed volume flow out pt	
fixed volume flow out pa	
fixed volume flow out pb	
Inputs	
spoolpos	position of the spool valve: 0 =closed, 1 = open 0 <= spoolpos <= 1
Parameters	
rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flows [m3/s.Pa], GLeak >= 0.
Amax	Maximum spool valve areas when the valves are open [m2], Amax >
Cd	0.
overlap	Discharge coefficient [], Cd > 0.
f	The valve overlaps in closed position [], -1 < overlap < 1.
d	Bandwidth of the spool dynamics [Hz], f > 0.

Damping of the spool dynamics [], d > 0

LoopFlushingValve

Library

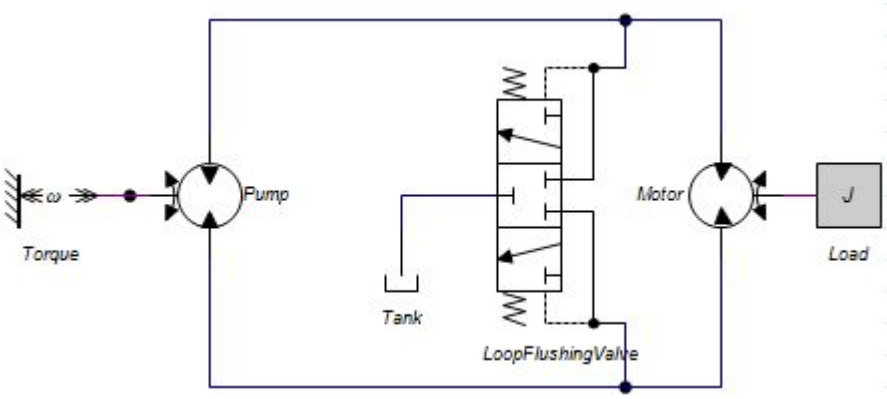
Iconic Diagrams\Hydraulics\Valves\Basic Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

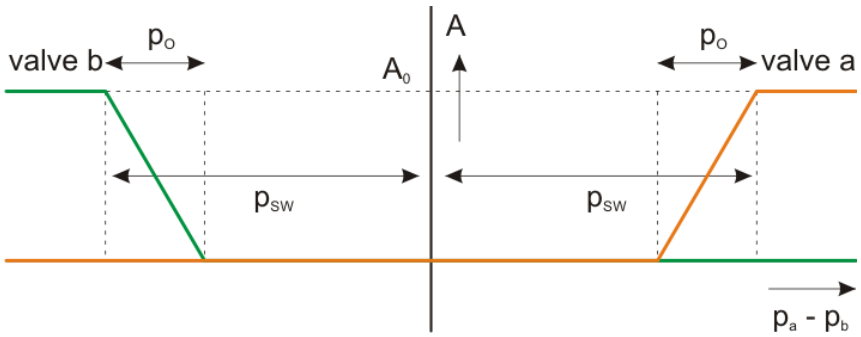
Description

Loop flushing valves are used to maintain a high quality working fluid in closed hydraulic circuits. In closed hydraulic circuits the oil is continuously flowing from a pump to an actuator. A loop flushing valve allows the oil to leave the circuit for cooling and filtering. If one side (*pa*) of the valve has a higher pressure, oil will flow from the other side (*pb*) out of the closed circuit and vice versa.



The circuit is always equipped (not shown above) with a charge pump to keep the suction side of the pump on a pre-pressure to avoid cavitation and so damage to the pump, and to exchange the fluid that is flushed.

Loop flushing valves are spring operated. I.e. a certain pressure difference between port *a* and *b* is required to open the valve. This pressure can be adjusted and is indicated by the parameter *p_sw*.



A small pressure difference is required to turn a valve from completely closed to completely opened. This overlap pressure is indicated by the parameter *p_o*.

Interface

Ports	**Description**
pa, pb	Input terminals of the valve.
p_out	Output terminal.
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
fixed volume flow out p_out	
Inputs	
spoolpos	position of the spool valve: 0 =closed, 1 = open 0 <= spoolpos <= 1
Parameters	
rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flows [m3/s.Pa], GLeak >= 0.
Amax	Maximum spool valve areas when the valves are open [m2], Amax > 0.
Cd	Discharge coefficient [], Cd > 0.
f	Bandwidth of the spool dynamics [Hz], f > 0.
d	Damping of the spool dynamics [], d > 0
p_o	Overlap pressure [Pa].
p_sw	Switching pressure [Pa].

PilotOperatedCheckValve

Library

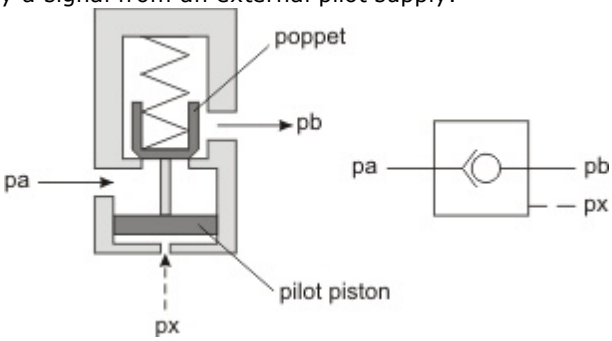
Iconic Diagrams\Hydraulics\Valves\Basic Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

A pilot operated check valve is a check valve that is normally closed but can be opened for reverse flow by a signal from an external pilot supply.



The figure above shows the standard design using a pilot piston with a stem to unseat the check valve poppet for reverse flow. The pilot piston has an area three to four times that of the poppet seat. This produces enough force to open the poppet against backpressure. Some pilot-operated check valves have area ratios up to 100:1, allowing a very low pilot pressure to open the valve against high backpressure.

The flow through a pilot operated check depends on the differential pressure *dp* between the inlet port *pa* and the outlet port *pb* and is equal to:

*flow = Cd * A * valve * sqrt(dp);*

The variable *valve* indicates the valve opening and varies between zero (closed) and 1 (open). The opening depends on a force balance which depends on the pressures of the inlet and outlet port, the spring force and the pressure of the pilot port *px:*

*valve = limit((pa.p - pb.p - pclosed + px.p*kp)/(popen - pclosed) , 0 , 1)*

The pilot ratio *kp* is ratio of the pilot piston area and the check valve area. For a large pilot ratio a relatively small pilot pressure is sufficient to open the valve against a large outlet pressure.

Interface

Ports	Description
pa	Inlet port
pb	Outlet port
px	Pilot port

Causality

fixed volume flow out pa
fixed volume flow out pb
fixed volume flow out pt

Parameters

pclosed	Valve is closed under this pressure [Pa] and zero pilot pressure.
popen	Valve is fully open above this pressure [Pa] and zero pilot pressure.
GLeak	Conductance of closed valve [m3/s.Pa].
A	Maximum valve area when the valve is open [m2], A >= 0!
Cd	Discharge coefficient [].
kp	Pilot ratio [].

PressureCompensator

Library

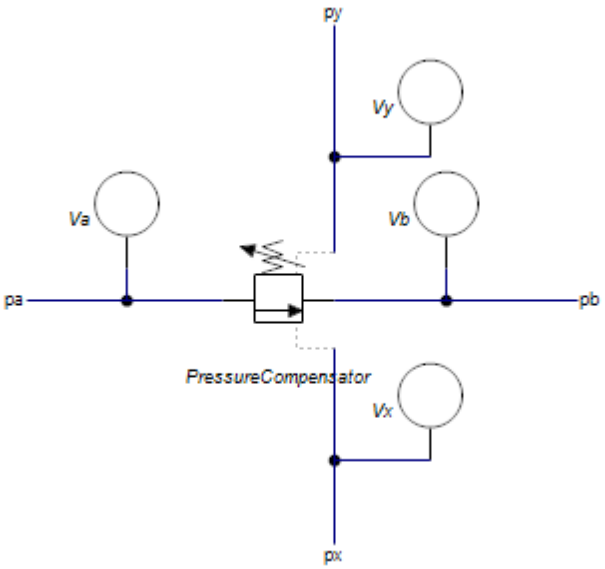
Iconic Diagrams\Hydraulics\Valves\Basic Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description - Default

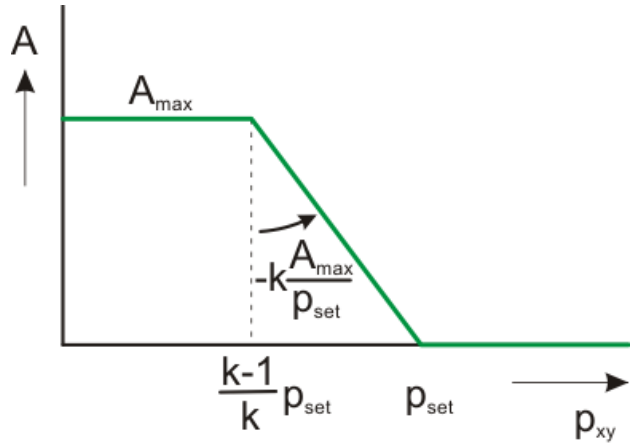
A pressure compensator is an component that maintains a constant differential pressure across it sensing ports *px* and *py* by regulating the flow through the component from port *pa* to port *pb.* A pressure compensator can be used to maintain a constant flow over a valve by keeping the pressure drop over the valve constant. The sensing ports *py* and *px* are then connected with the input and output of the valve.



The flow through the element is laminar or turbulent depending on the pressure drop *p_ab* over *pa* and *pb*:

```
p_ab = pa.p - pb.p;
phi = sign(p_ab) * Cd * A * sqrt( (2/rho) * abs(dp) ) + GLeak * p_ab;
```

The opening *A* depends on the pressure drop *p_xy* over *pa* and *pb:*



The maximum opening *A_max* is determined by the maximum flow *phi_max* at the maximum pressure drop *p_in_max* over the component:

*A_max = phi_max / (Cd * sqrt((2/rho) * p_in_max));*

The opening is at its maximum value when the differential pressure across it sensing ports *px* and *py* is below the pressure ((*k-1)/k)*p_set*. If the differential pressure across it sensing ports *px* and *py* is too high, the opening is closed. In between it varies between fully open and fully closed. The speed of response of the components is determined by the proportional gain *k* (typical choose k = 2 to 5). The proportional gain determines the slope of the valve opening. If *k* is large the regulator will respond quickly but also induce oscillations in the circuit.

The valve is implemented with second order spool dynamics. I.e. the spool opening is characterized by the bandwidth (*f*) and damping (*d*).

Interface

Ports	**Description**
pa, pb	Input and output terminals of the component.
px, py	Sensing terminals of the component. These are used to measure the pressure and have almost zero flow.

Causality

fixed volume flow
out pa
fixed volume flow
out pb
fixed volume flow
out px
fixed volume flow
out py

Parameters

rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flows [m3/s.Pa], GLeak >= 0.
Cd	Discharge coefficient [], Cd > 0.
p_set	Desired differential pressure over xy (px.p - py.p), [Pa]
p_in_max	maximum inlet pressure, [Pa]
phi_max	maximum flow at maximum inlet pressure (pa.p) and zero outlet pressure (pb.p), [m3/s]
f	natural frequency of the second order spool dynamics [Hz]
d	relative damping of the second order spool dynamics [Hz]
k	proportional gain [-]

PressureReliefValve

Library

Iconic Diagrams\Hydraulics\Valves\Basic Valves

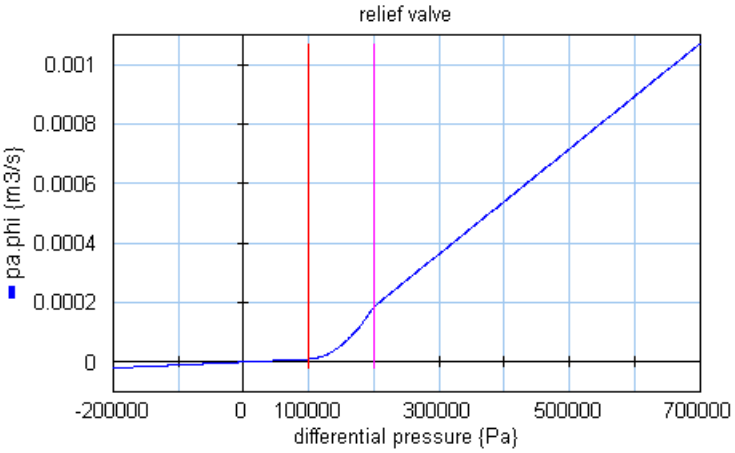
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes a pressure relief valve. The resistance depends on the pressure difference:

dp = pa.p - pb.p
*dp < pclosed => valve closed, only leakage: pa.phi = pb.phi = dp * GLeak*
pclosed < dp < popen => working range, i. e. valve partially opened
*popen < dp => valve wide open: pa.phi = pb.phi = dp * GOpen*



Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
Parameters	
pclosed	Valve is closed under this pressure [Pa].
popen	Valve is fully open above this pressure [Pa].
GLeak	Conductance of closed valve [m3/s.Pa].
GOpen	Conductance of open valve [m3/s.Pa].

PressureReducingValve

Library

Iconic Diagrams\Hydraulics\Valves\Basic Valves

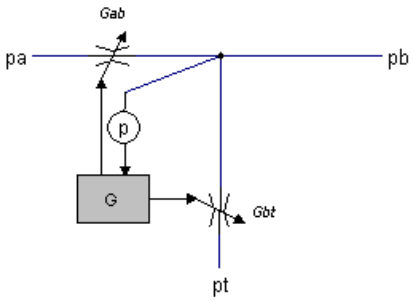
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

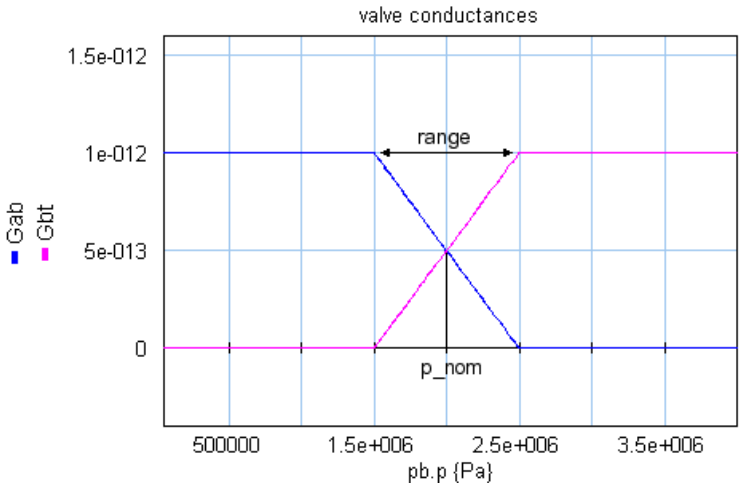
Description

Pressure reducing valves have the purpose to provide a constant pressure (p_nom), independent of the upstream pressure. They are used to limit the pressure of a primary circuit to a desired pressure for a secondary circuit.

The desired pressure (*p_nom*) is obtained by means of a flow over a valve between an input port (*pa*) and an output port (*pb*). The valve is modeled by a variable conductance). Depending on the pressure downstream (*pb.p*) the valve is opened (large conductance) or closed (zero conductance). Therefore pressure reducing valves always need a small flow to establish the correct downstream pressure.



When the downstream pressure (*pb.p*) increases over the desired pressure a second valve opens to allow flow from the downstream port (*pb*) to a third port (*pt*). The third port is mostly connected to the tank.



The valves have a pressure region for opening an closing. This region is indicated by the parameter *range*.

Interface

Ports	**Description**
pa	Upstream port
pb	Downstram port
pt	Relief port

Causality

fixed volume flow out pa
fixed volume flow out pb
fixed volume flow out pt

Parameters

p_nom	Desired downstream pressure [Pa].
range	Range for opening and closing the valves [Pa].
Gab	The maximum conductance from a to b [m3/s.Pa].
Gbt	The maximum conductance from b to t [m3/s.Pa].

ShuttleValve

Library

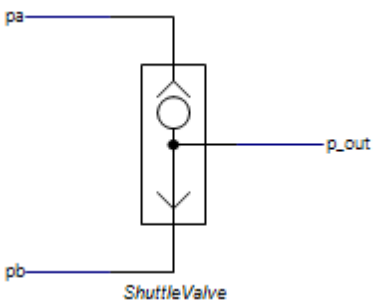
Iconic Diagrams\Hydraulics\Valves\Basic Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description - Default

A shuttle valve is a type of valve which allows fluid to flow through it from one of two sources. Shuttle valves accept flow from two different sources (*pa*) and (*pb*) and divert the highest pressure to a single outlet port (*p_out*). Shuttle valves are commonly used in Load Sensing circuits and at a cylinder to measure the working pressure, as well as Brake circuits. Normal shuttle valve are mostly ball and poppet types valves.



The default implementation of the shuttle valve assumes ideal behaviour:

```
p.out.p = maximum(pa.p, pb.p)
pa.phi = pb.phi = 0;
```

This model sets the flows of the input ports to zero and assumes the output flow is zero as well. However the output flow is determined by the component that is connected with the output port. A warning is given when the output flow exceeds 1.0e-5 m3. If flows are important in your model, use the Spool Dynamics implementation of this model (see below).

Interface - Default

Ports	Description
pa, pb	Input terminals of the valve.
p_out	Output terminal.

Causality

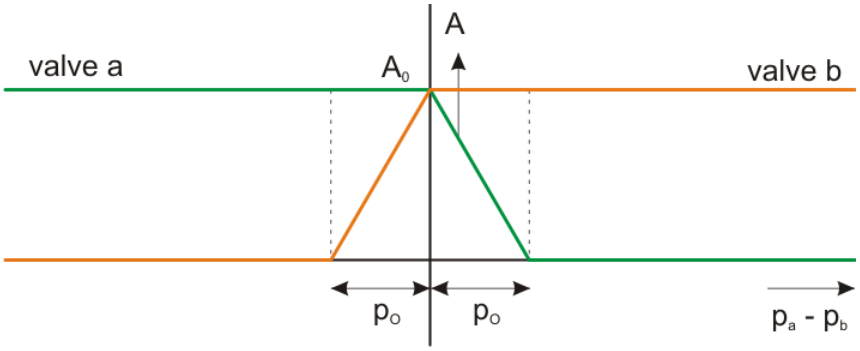
fixed volume flow out pa

fixed volume flow out pb

fixed pressure out
p_out

Description - BallDynamics

This implementation is equal to the default implementation but with modeled dynamics. A small pressure difference between the inlet ports *pa* and *pb* is enough to make the ball switch from one side to the other. This pressure is called the overlap pressure *po.*



Interface - BallDynamics

Ports	**Description**
pa, pb	Input terminals of the valve.
p_out	Output terminal.
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
fixed volume flow out p_out	
Parameters	
rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flows [m3/s.Pa], GLeak >= 0.
Amax	Maximum valve areas when the valves are open [m2], Amax > 0.
Cd	Discharge coefficient [], Cd > 0.
f	Bandwidth of the valve dynamics [Hz], f > 0.
d	Damping of the valve dynamics [], d > 0
p_o	Overlap pressure [Pa].

TwoTwoWayDirectionalValve

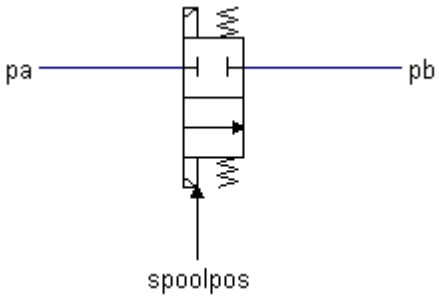
Library

Iconic Diagrams\Hydraulics\Valves\Basic Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

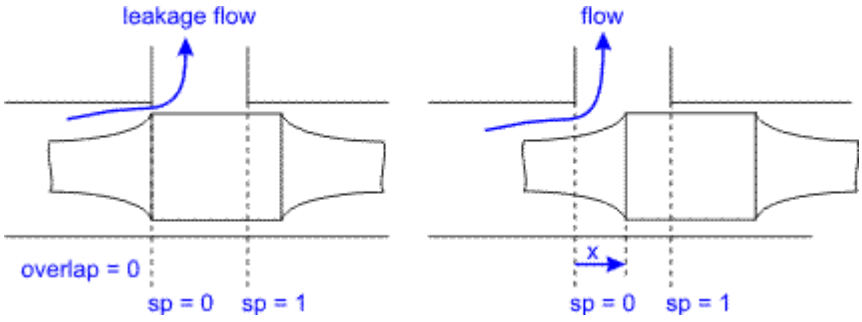
Description



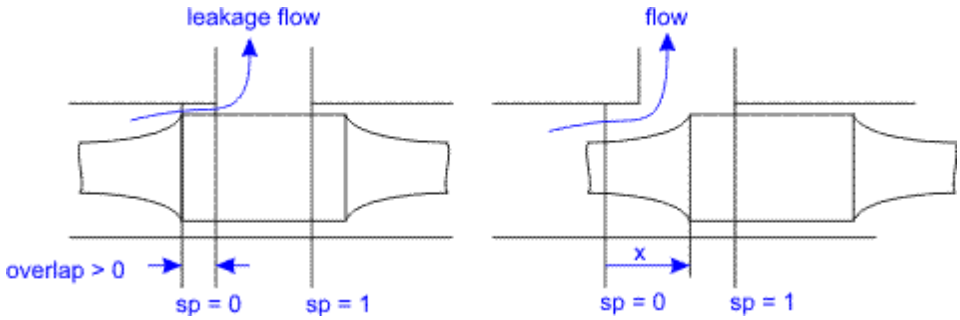
This model describes a 2/2-way directional control valve with second order spool dynamics. The flow through the valve is described as laminar/turbulent flow through an orifice:

dp = pa.p - pb.p
*phi = sign(dp) * Cd * A(sp) * sqrt((2/rho) * abs(dp)) + GLeak * dp;*

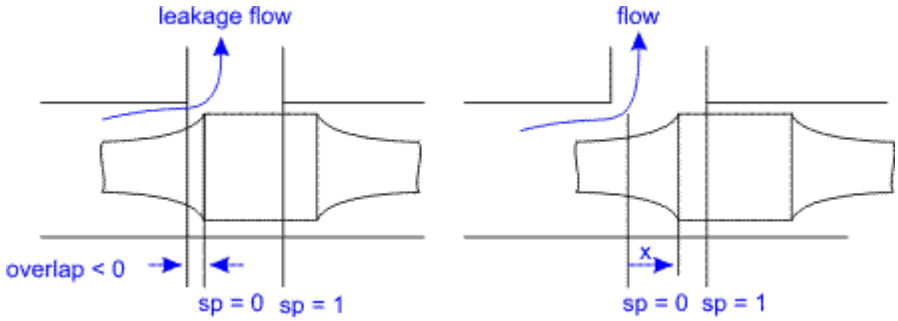
Here *Cd* is the discharge coefficient which normally has a value between 0.55 and 0.7. A(*sp*) is the area of the orifice opening. A(*sp*) depends linearly on the spool position *sp* and varies between 0 (sp = 0) and the maximum area *Amax* (*sp* = 1). *Gleak* is the conductance of laminar leakage flow when the valve is closed. The relative opening of the spool valve is indicated by *sp*. For a closed valve the spool position (*sp*) is equal to zero and for an open valve the spool position (*sp*) is equal to 1.



In the neutral spool position (*sp = 0*) the valve is just closed. A positive overlap indicates that the spool must travel a certain distance before the valve opens.



A negative overlap indicates that the valve is already open in the neutral position.



The overlap is indicated by the parameter overlap, which is given as a fraction of the spool position.

The spool position sp is a function of the input signal spoolpos:

sp = SO(f,d,discrete(spoolpos))

where *SO* is a second order transfer function to model the spool dynamics. The function is characterized by the bandwidth (*f*) and damping (*d*). The model acts as a directional valve (a valve which is either open or closed) because the input signal *spoolpos* is rounded to 0 or 1:

spoolpos < 0.5 => 0
spoolpos >= 0.5 => 1

The pressure at both ports has a lower limit which is equal to the vapour pressure. Therefore the actual equations used in this component are:

pa_lim = if pa.p < p_vapour then p_vapour else pa.p end;
pb_lim = if pb.p < p_vapour then p_vapour else pb.p end;

Interface

Ports	Description
pa, pb	Both terminals of the valve.
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
Inputs	
spoolpos	position of the spool valve spoolpos < 0.5 => valve is closed spoolpos >= 0.5 => valve is open
Parameters	
rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0.
Amax	Maximum spool valve area when the valve is open [m2], Amax > 0.
Cd	Discharge coefficient [], Cd > 0.
overlap	The valve overlap in closed position [], -1 < overlap < 1.
f	Bandwidth of the spool dynamics [Hz], f > 0.
d	Damping of the spool dynamics [], d > 0.

TwoTwoWayProportionalValve

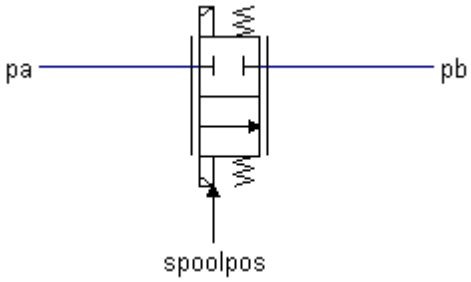
Library

Iconic Diagrams\Hydraulics\Valves\Basic Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

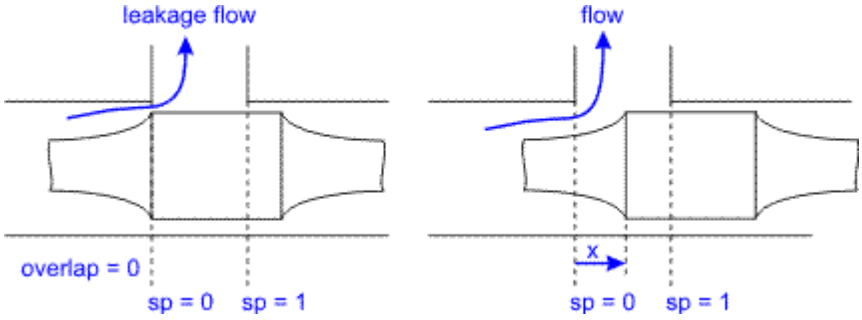
Description



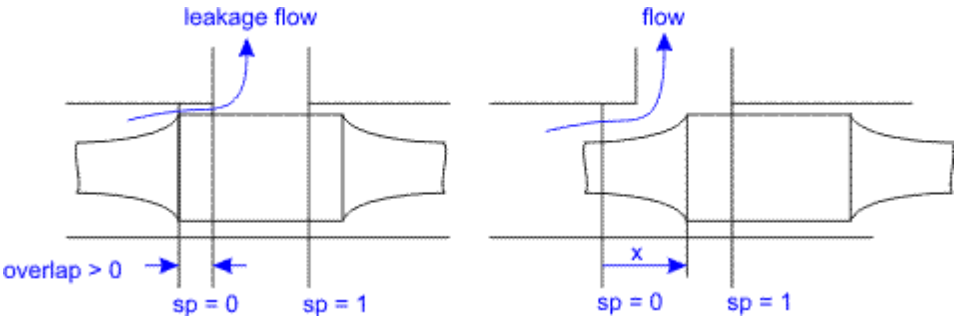
This model describes a 2/2-way proportional control valve with second order spool dynamics. The flow through the valve is described as laminar/turbulent flow through an orifice:

```
dp = pa.p - pb.p
phi = sign(dp) * Cd * A(sp) * sqrt( (2/rho) * abs(dp) ) + GLeak * dp;
```

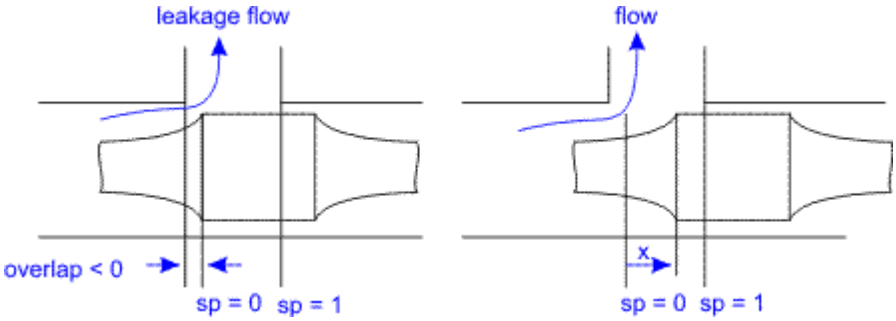
Here *Cd* is the discharge coefficient which normally has a value between 0.55 and 0.7. A(*sp*) is the area of the orifice opening. A(*sp*) depends linearly on the spool position *sp* and varies between 0 (*sp = 0*) and the maximum area *Amax* (*sp = 1*). *Gleak* is the conductance of laminar leakage flow when the valve is closed. The relative opening of the spool valve is indicated by sp. For a closed valve the spool position (*sp*) is equal to zero and for an open valve the spool position (*sp*) is equal to 1.



In the neutral spool position (*sp = 0*) the valve is just closed. A positive overlap indicates that the spool must travel a certain distance before the valve opens.



A negative overlap indicates that the valve is already open in the neutral position.



The overlap is indicated by the parameter overlap, which is given as a fraction of the spool position.

The spool position *sp* is a function of the input signal *spoolpos*:

sp = SO(f,d,spoolpos)

where *SO* is a second order transfer function to model the spool dynamics. The function is characterized by the bandwidth *(f)* and damping *(d)*. The valve input position *spoolpos* should be limited to the range between 0 and 1.

The pressure at both ports has a lower limit which is equal to the vapour pressure. Therefore the actual equations used in this component are:

pa_lim = if pa.p < p_vapour then p_vapour else pa.p end;
pb_lim = if pb.p < p_vapour then p_vapour else pb.p end;

Interface

Ports	Description
pa, pb	Both terminals of the valve.

Causality

fixed volume flow
out pa
fixed volume flow
out pb

Inputs

spoolpos	position of the spool valve: 0 =closed, 1 = open 0 <= spoolpos <= 1

Parameters

rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0.
Amax	Maximum spool valve area when the valve is open [m2], Amax > 0.
Cd	Discharge coefficient [], Cd > 0.
overlap	The valve overlap in closed position [], -1 < overlap < 1.
f	Bandwidth of the spool dynamics [Hz], f > 0.
d	Damping of the spool dynamics [], d > 0.

CheckValve-States

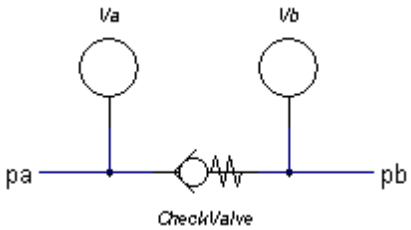
Library

Iconic Diagrams\Hydraulics\Valves

Use

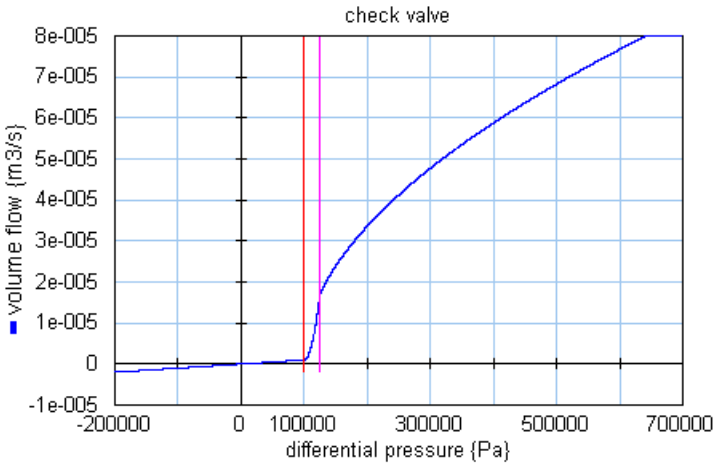
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description



This model describes a spring-loaded check valve with lumped volumes. The resistance depends on the pressure difference:

*dp < pclosed => valve closed, only leakage: pa.phi = pb.phi = dp * GLeak*
pclosed < dp < popen => working range, i. e. valve partially opened
*popen < dp => valve wide open: pa.phi = pb.phi = K * sqrt(dp - pclosed);*



Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
Causality	
preferred pressure out pa	
preferred pressure out pb	
Parameters	
CheckValve\pclosed	Valve is closed under this pressure [Pa].
CheckValve\popen	Valve is fully open above this pressure [Pa].
CheckValve\GLeak	Conductance of closed valve [m3/s.Pa].
CheckValve\A	Maximum valve area when the valve is open [m2], A >=
CheckValve\Cd	0!
Va\V	Discharge coefficient [].
Va\B	Volume of oil under pressure [m3]
Va\p_initial	Effective bulk modulus [Pa]
Vb\V	The starting pressure of the volume [Pa]
Vb\B	Volume of oil under pressure [m3]
Vb\p_initial	Effective bulk modulus [Pa]
	The starting pressure of the volume [Pa]

CounterbalanceValve

Library

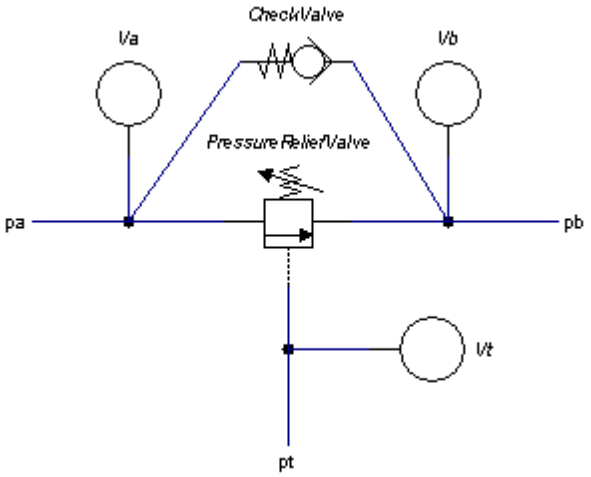
Iconic Diagrams\Hydraulics\Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model an externally piloted counter balance valve. The model consists of an externally piloted pressure relief valve with a check valve in parallel. The ports of the model are connected to lumped volumes.



The Counterbalance valve is applied as a *brake valve* to get a positive control on a hydraulic cylinder or motor with a negative load. The check valve is used to get a free running actuator in one direction and the pressure relief valve is used to control the actuator in the other direction. The relief valve is controlled by the external pilot pressure.

Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
pt.	

Causality

fixed volume flow out pa
fixed volume flow out pb
fixed volume flow out pt

Parameters

CheckValve\pclosed	Valve is closed under this pressure [Pa].
CheckValve\popen	Valve is fully open above this pressure [Pa].
CheckValve\GLeak	Conductance of closed valve [m3/s.Pa].
CheckValve\A	Maximum valve area when the valve is open [m2], A >=
CheckValve\Cd	0!
PressureReliefValve\pclosed	Discharge coefficient [].
PressureReliefValve\popen	Valve is closed under this pressure [Pa].
PressureReliefValve\GLeak	Valve is fully open above this pressure [Pa].
PressureReliefValve\GOpen	Conductance of closed valve [m3/s.Pa].
Va\V	Conductance of open valve [m3/s.Pa].
Va\B	Volume of oil under pressure [m3]
Va\p_initial	Effective bulk modulus [Pa]
Vb\V	The starting pressure of the volume [Pa]
Vb\B	Volume of oil under pressure [m3]
Vb\p_initial	Effective bulk modulus [Pa]
Vt\V	The starting pressure of the volume [Pa]
Vt\B	Volume of oil under pressure [m3]
Vt\p_initial	Effective bulk modulus [Pa]
	The starting pressure of the volume [Pa]

FlowControlValve-States

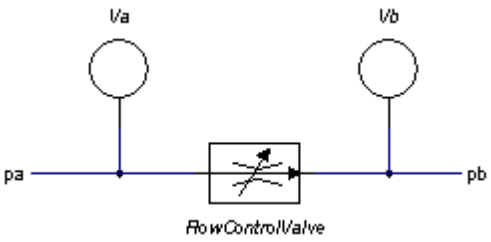
Library

Iconic Diagrams\Hydraulics\Valves\Basic Valves

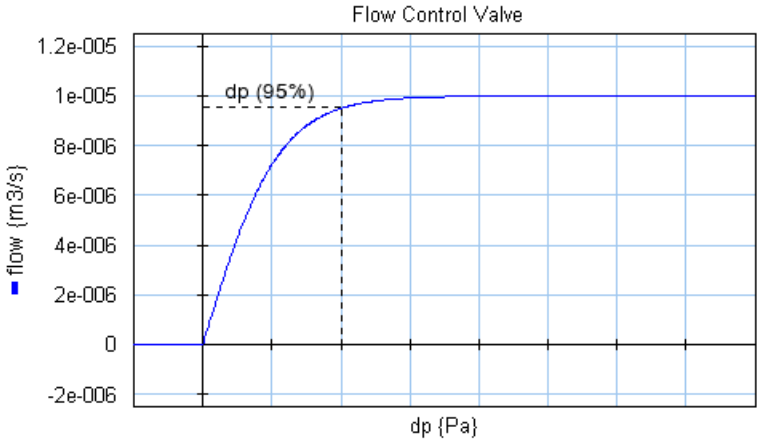
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description



This model describes a flow control valve with with lumped volumes.



In this model the flow is modeled by an *tanh* function. The pressure drop is defined as the pressure where 95% of the desired flow rate is achieved.

```
dp = pa.p - pb.p
phi = if dp > 0 then
  phi*arctanh( (arctan(0.95)/p_drop) * dp) + GLeak * dp;
else
  GLeak * dp
end;
```

Here *phi* is the desired flow and *p_drop* is the 95% pressure drop. *Gleak* is the conductance of laminar leakage flow when the valve is closed. The pressure at both ports has a lower limit that is equal to the vapour pressure. Therefore the actual equations used in this model are:

```
p1_lim = if pa.p < p_vapour then p_vapour else pa.p end;
p2_lim = if pb.p < p_vapour then p_vapour else pb.p end;
dp = p1_lim - p2_lim;
```

Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
Parameters	
FlowControlValve\phi	Desired flow [m3/s]
FlowControlValve\p_drop	pressure drop at 95% flow [Pa]
FlowControlValve\GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0!
Va\V	Volume of oil under pressure [m3]
Va\B	

Va\p_initial	Effective bulk modulus [Pa]
Vb\V	The starting pressure of the volume [Pa]
Vb\B	Volume of oil under pressure [m3]
Vb\p_initial	Effective bulk modulus [Pa]
	The starting pressure of the volume [Pa]

FourThreeWayDirectionalValve-States

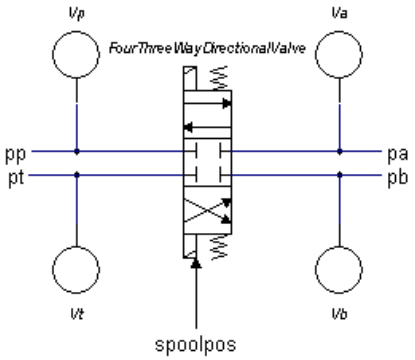
Library

Iconic Diagrams\Hydraulics\Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description



This model describes a 4/3-way directional control valve with second order spool dynamics and lumped volumes. The flow through the valves is described as laminar/ turbulent flow through an orifice. A detailed description of the valve can be found in FourThreeWayDirectionalValve.htm.

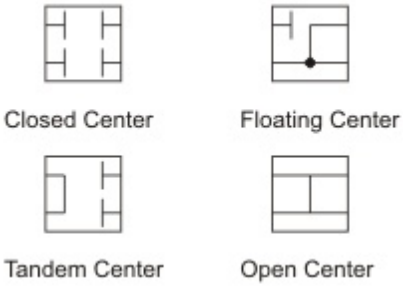
The spool position is indicated by the input spoolpos:

sp = -1	*Flow from p to b and a to t*
sp = 0	*No flow*
sp = 1	*Flow from p to a and b to t*

Values between -1 and 0 or between 0 and 1 are rounded to the nearest value.

Implementation

The 4/3-way proportional control valve is implemented with various spool center configurations. The configurations are shown in the picture below:



When you drag and drop a model in the editor, you will be asked which implementation you want to choose. During modeling you can easily change the spool center configuration:

1. Select the valve model.
2. Click the right mouse button to open the right mouse menu.
3. Click *Edit Implementation* and choose another implementation.

Interface

Ports	**Description**
pp, pt, pa, pb	All terminals of the valve.
Causality	
preferred effort out pp	
preferred effort out pt	
preferred effort out pa	
preferred effort out pb	
Inputs	
spoolpos	position of the spool valve spoolpos < 0.5 => valve is closed spoolpos >= 0.5 => valve is open
Parameters	
FourThreeWayDirectionalValve\rho	Mass density of the fluid [kg/m3].
FourThreeWayDirectionalValve\GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0.
FourThreeWayDirectionalValve\Amax	Maximum spool valve area when the valve is open [m2], Amax > 0.
FourThreeWayDirectionalValve\Cd	Discharge coefficient [], Cd > 0.
FourThreeWayDirectionalValve\overlap	The valve overlap in closed position [], -1 < overlap < 1.
	Bandwidth of the spool dynamics [Hz], f > 0.
	Damping of the spool dynamics [], d > 0.
	Volume of oil under pressure [m3]

FourThreeWayDirectionalValv	Effective bulk modulus [Pa]
e\f	The starting pressure of the volume [Pa]
FourThreeWayDirectionalValv	Volume of oil under pressure [m3]
e\d	Effective bulk modulus [Pa]
Va\V	The starting pressure of the volume [Pa]
Va\B	
Va\p_initial	
Vb\V	
Vb\B	
Vb\p_initial	

FourThreeWayProportionalValve-States

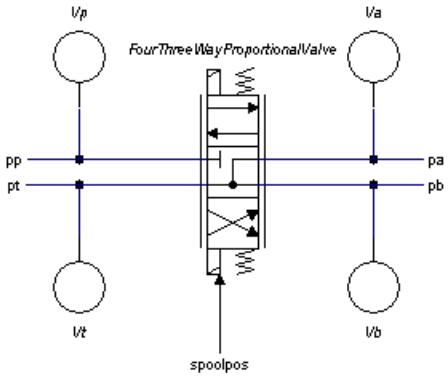
Library

Iconic Diagrams\Hydraulics\Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description



This model describes a 4/3-way proportional control valve with second order spool dynamics and lumped volumes. The flow through the valves is described as laminar/ turbulent flow through an orifice. A detailed description of the valve can be found in FourThreeWayProportionalValve.htm.

The spool position is indicated by the input *spoolpos*:

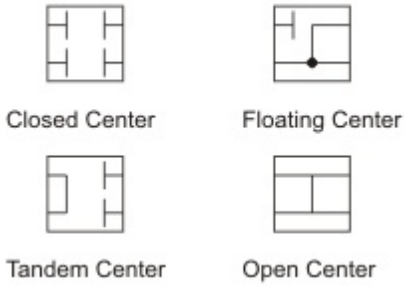
sp = -1 *Flow from p to b and a to t*

-1 < sp < 0 *Partial flow from p to b and a to t*

sp = 0	*No flow*
0 < sp < 1	*Partial flow from p to a and b to t*
sp = 1	*Flow from p to a and b to t*

Implementation

The 4/3-way proportional control valve is implemented with various spool centre configurations. The configurations are shown in the picture below:



When you drag and drop a model in the editor, you will be asked which implementation you want to choose. During modeling you can easily change the spool center configuration:

1. Select the valve model.
2. Click the right mouse button to open the right mouse menu.
3. Click *Edit Implementation* and choose another implementation.

Interface

Ports	**Description**
pp, pt, pa, pb	All terminals of the valve.
Causality	
preferred effort out pp	
preferred effort out pt	
preferred effort out pa	
preferred effort out pb	
Inputs	
spoolpos	value of the spool position
Parameters	
FourThreeWayProportionalVal ve\rho	Mass density of the fluid [kg/m3].
FourThreeWayProportionalVal ve\GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0.

FourThreeWayProportionalValve\Amax	Maximum spool valve area when the valve is open [m2], Amax > 0.
FourThreeWayProportionalValve\Cd	Discharge coefficient [], Cd > 0.
FourThreeWayProportionalValve\overlap	The valve overlap in closed position [], -1 < overlap < 1.
FourThreeWayProportionalValve\f	Bandwidth of the spool dynamics [Hz], f > 0.
FourThreeWayProportionalValve\d	Damping of the spool dynamics [], d > 0.
Va\V	Volume of oil under pressure [m3]
Va\B	Effective bulk modulus [Pa]
Va\p_initial	The starting pressure of the volume [Pa]
Vb\V	Volume of oil under pressure [m3]
Vb\B	Effective bulk modulus [Pa]
Vb\p_initial	The starting pressure of the volume [Pa]

LoopFlushingValve-States

Library

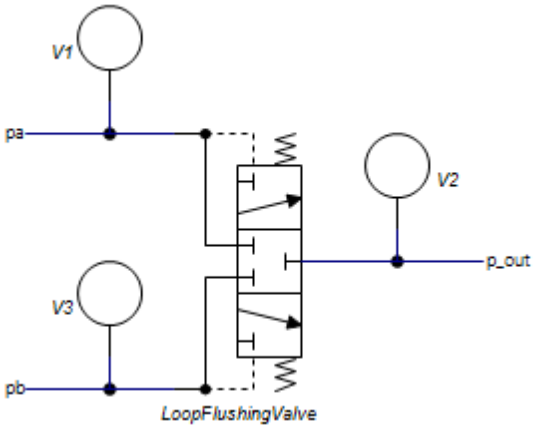
Iconic Diagrams\Hydraulics\Valves

Use

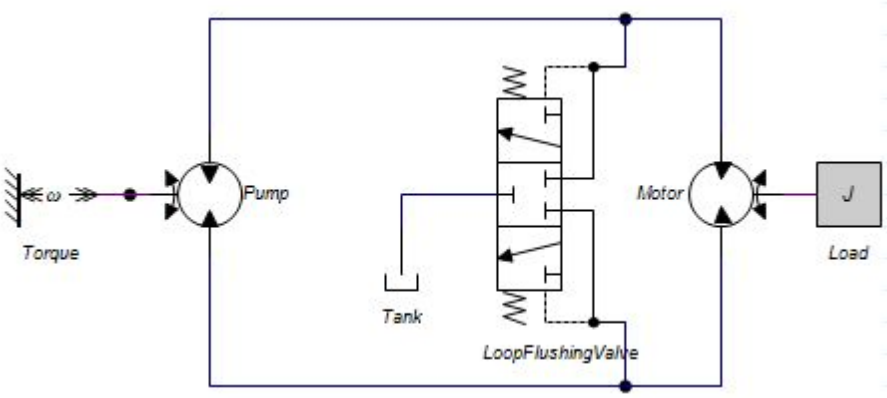
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model describes a loop flushing valve with lumped volumes.

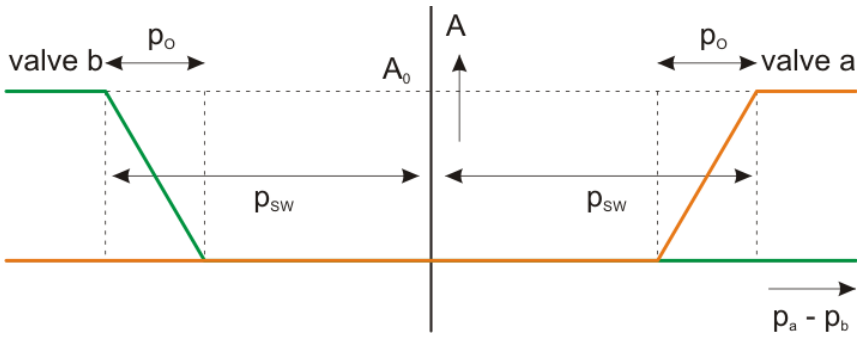


Loop flushing valves are used to maintain a high quality working fluid in closed hydraulic circuits. In closed hydraulic circuits the oil is continuously flowing from a pump to an actuator. A loop flushing valve allows the oil to leave the circuit for cooling and filtering. If one side (*pa*) of the valve has a higher pressure, oil will flow from the other side (*pb*) out of the closed circuit and vice versa.



The circuit is always equipped (not shown above) with a charge pump to keep the suction side of the pump on a pre-pressure to avoid cavitation and so damage to the pump, and to exchange the fluid that is flushed.

Loop flushing valves are spring operated. I.e. a certain pressure difference between port *a* and *b* is required to open the valve. This pressure can be adjusted and is indicated by the parameter *p_sw*.



A small pressure difference is required to turn a valve from completely closed to completely opened. This overlap pressure is indicated by the parameter *p_o*.

Interface

Ports	**Description**
pa, pb	Input terminals of the valve.
p_out	Output terminal.
Causality	
preferred pressure out pa	
preferred pressure out pb	
preferred pressure out p_out	
Inputs	
spoolpos	position of the spool valve: 0 =closed, 1 = open 0 <= spoolpos <= 1
Parameters	
rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flows [m3/s.Pa], GLeak >= 0.
Amax	Maximum spool valve areas when the valves are open [m2], Amax > 0.
Cd	Discharge coefficient [], Cd > 0.
f	Bandwidth of the spool dynamics [Hz], f > 0.
d	Damping of the spool dynamics [], d > 0
p_o	Overlap pressure [Pa].
p_sw	Switching pressure [Pa].

PilotOperatedCheckValve-States

Library

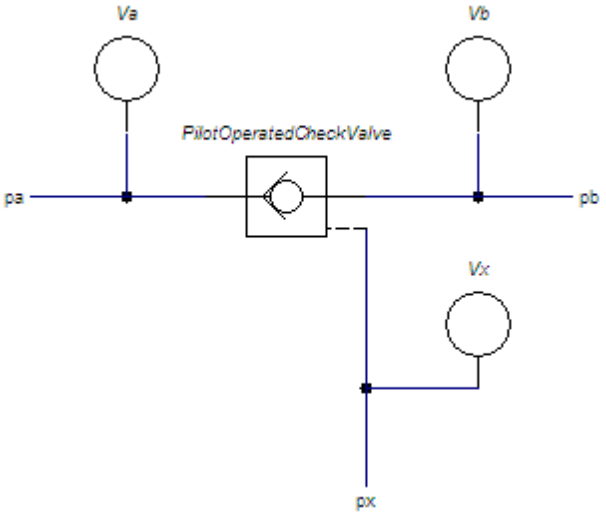
Iconic Diagrams\Hydraulics\Valves\Basic Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model is a pilot operated check valve with lumped volumes.



The flow through a pilot operated check depends on the differential pressure *dp* between the inlet port *pa* and the outlet port *pb* and is equal to:

*flow = Cd * A * valve * sqrt(dp);*

The variable *valve* indicates the valve opening and varies between zero (closed) and 1 (open). The opening depends on a force balance which depends on the pressures of the inlet and outlet port, the spring force and the pressure of the pilot port *px:*

*valve = limit((pa.p - pb.p - pclosed + px.p*kp)/(popen - pclosed) , 0 , 1)*

The pilot ratio *kp* is ratio of the pilot piston area and the check valve area. For a large pilot ratio a relatively small pilot pressure is sufficient to open the valve against a large outlet pressure.

Interface

Ports	Description
pa	Inlet port
pb	Outlet port
px	Pilot port

Causality

fixed volume flow out pa
fixed volume flow out pb
fixed volume flow out pt

Parameters

pclosed	Valve is closed under this pressure [Pa] and zero pilot pressure.
popen	Valve is fully open above this pressure [Pa] and zero pilot pressure.
GLeak	Conductance of closed valve [m3/s.Pa].
A	Maximum valve area when the valve is open [m2], A >= 0!
Cd	Discharge coefficient [].
kp	Pilot ratio [].

PressureCompensator-States

Library

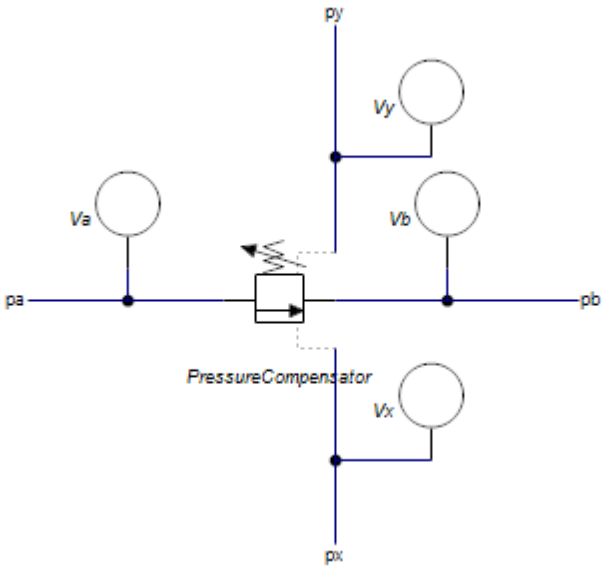
Iconic Diagrams\Hydraulics\Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description - Default

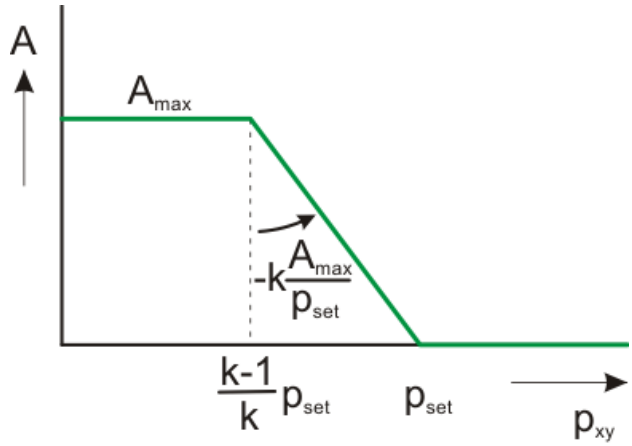
This model is a pressure compensator with lumped volumes. A pressure compensator is an component that maintains a constant differential pressure across it sensing ports *px* and *py* by regulating the flow through the component from port *pa* to port *pb.* A pressure compensator can be used to maintain a constant flow over a valve by keeping the pressure drop over the valve constant. The sensing ports *py* and *px* are then connected with the input and output of the valve.



The flow through the element is laminar or turbulent depending on the pressure drop *p_ab* over *pa* and *pb*:

```
p_ab = pa.p - pb.p;
phi = sign(p_ab) * Cd * A * sqrt( (2/rho) * abs(dp) ) + GLeak * p_ab;
```

The opening *A* depends on the pressure drop *p_xy* over *pa* and *pb:*



The maximum opening *A_max* is determined by the maximum flow *phi_max* at the maximum pressure drop *p_in_max* over the component:

```
A_max = phi_max / ( Cd * sqrt( (2/rho) * p_in_max ) );
```

The opening is at its maximum value when the differential pressure across it sensing ports *px* and *py* is below the pressure ((*k-1)/k)*p_set*. If the differential pressure across it sensing ports *px* and *py* is too high, the opening is closed. In between it varies between fully open and fully closed. The speed of response of the components is determined by the proportional gain *k* (typical choose k = 2 to 5). The proportional gain determines the slope of the valve opening. If *k* is large the regulator will respond quickly but also induce oscillations in the circuit.

The valve is implemented with second order spool dynamics. I.e. the spool opening is characterized by the bandwidth (*f*) and damping (*d*).

Interface

Ports	**Description**
pa, pb	Input and output terminals of the component.
px, py	Sensing terminals of the component. These are used to measure the pressure and have almost zero flow.

Causality

preferred
pressure out pa
preferred
pressure out pb
preferred
pressure out px
preferred
pressure out py

Parameters

rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flows [m3/s.Pa], GLeak >= 0.
Cd	Discharge coefficient [], Cd > 0.
p_set	Desired differential pressure over xy (px.p - py.p), [Pa]
p_in_max	maximum inlet pressure, [Pa]
phi_max	maximum flow at maximum inlet pressure (pa.p) and zero outlet pressure (pb.p), [m3/s]
f	natural frequency of the second order spool dynamics [Hz]
d	relative damping of the second order spool dynamics [Hz]
k	proportional gain [-]

PressureReducingValve-States

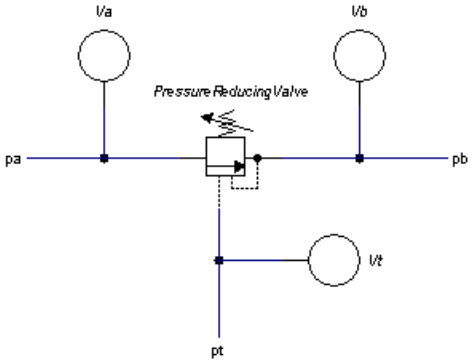
Library

Iconic Diagrams\Hydraulics\Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description



This model describes a pressure reducing valve with lumped volumes. Pressure reducing valves have the purpose to provide a constant pressure (*p_nom*), independent of the upstreamstream pressure. They are used to limit the pressure of a primary circuit to a desired pressure for a secondary circuit.

Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
pt	Pilot
Causality	
fixed volume flow out pa	
fixed volume flow out pb	
Parameters	
FlowControlValve\phi	Desired flow [m3/s]
FlowControlValve\p_drop	pressure drop at 95% flow [Pa]
FlowControlValve\GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0!
Va\V	Volume of oil under pressure [m3]
Va\B	Effective bulk modulus [Pa]
Va\p_initial	The starting pressure of the volume [Pa]
Vb\V	Volume of oil under pressure [m3]
Vb\B	Effective bulk modulus [Pa]
Vb\p_initial	Effective bulk modulus [Pa]

The starting pressure of the volume [Pa]

PressureReliefValve-States

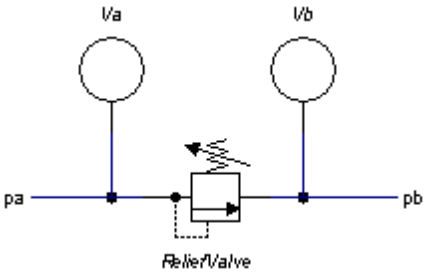
Library

Iconic Diagrams\Hydraulics\Valves

Use

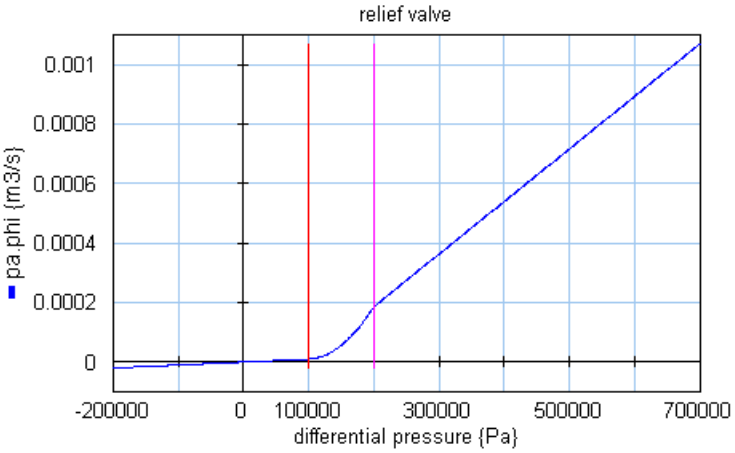
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description



This model describes a pressure relief valve with lumped volumes. The resistance depends on the pressure difference:

dp = pa.p - pb.p
*dp < pclosed => valve closed, only leakage: pa.phi = pb.phi = dp * GLeak*
pclosed < dp < popen => working range, i. e. valve partially opened
*popen < dp => valve wide open: pa.phi = pb.phi = dp * GOpen*



Interface

Ports	**Description**
pa, pb	Both terminals of the hydraulic component.
Causality	
preferred pressure out pa	
preferred pressure out pb	
Parameters	
PressureReliefValve\pclosed	Valve is closed under this pressure [Pa].
PressureReliefValve\popen	Valve is fully open above this pressure [Pa].
PressureReliefValve\GLeak	Conductance of closed valve [m3/s.Pa].
PressureReliefValve\GOpen	Conductance of open valve [m3/s.Pa].
Va1\V	Volume of oil under pressure [m3]
Va\B	Effective bulk modulus [Pa]
Va\p_initial	The starting pressure of the volume [Pa]
Vb\V	Volume of oil under pressure [m3]
Vb\B	Effective bulk modulus [Pa]
Vb\p_initial	The starting pressure of the volume [Pa]

ShuttleValve-States

Library

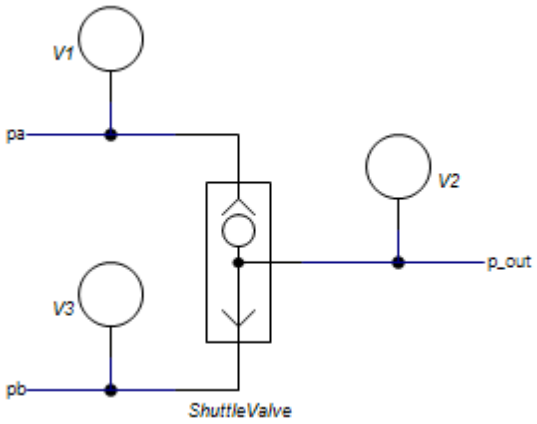
Iconic Diagrams\Hydraulics\Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description - Default

This model describes a loop shuttle valve with lumped volumes. A shuttle valve is a type of valve which allows fluid to flow through it from one of two sources. Shuttle valves accept flow from two different sources (*pa*) and (*pb*) and divert the highest pressure to a single outlet port (*p_out*). Shuttle valves are commonly used in Load Sensing circuits and at a cylinder to measure the working pressure, as well as Brake circuits. Normal shuttle valve are mostly ball and poppet types valves.



The default implementation of the shuttle valve assumes ideal behaviour:

```
p.out.p = maximum(pa.p, pb.p)
pa.phi = pb.phi = 0;
```

This model sets the flows of the input ports to zero and assumes the output flow is zero as well. However the output flow is determined by the component that is connected with the output port. A warning is given when the output flow exceeds 1.0e-5 m3. If flows are important in your model, use the Spool Dynamics implementation of this model (see below).

Interface - Default

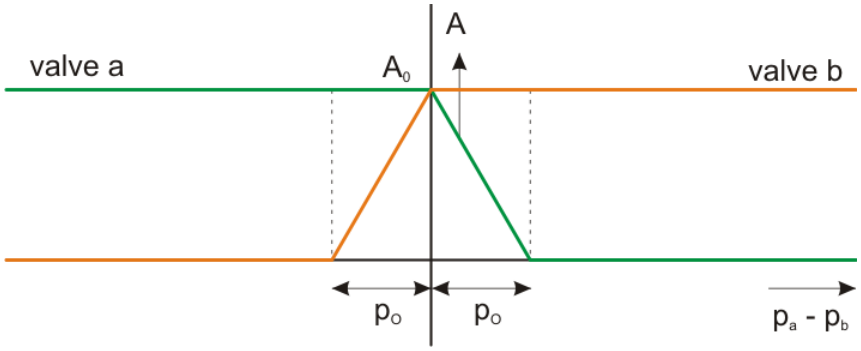
Ports	Description
pa, pb	Input terminals of the valve.
p_out	Output terminal.

Causality

preferred
pressure out pa
preferred
pressure out pb
preferred
pressure out
p_out

Description - BallDynamics

This model describes a loop shuttle valve with lumped volumes. This implementation is equal to the default implementation but with modeled dynamics. A small pressure difference between the inlet ports *pa* and *pb* is enough to make the ball switch from one side to the other. This pressure is called the overlap pressure *po.*



Interface - BallDynamics

Ports	**Description**
pa, pb	Input terminals of the valve.
p_out	Output terminal.

Causality

preferred
pressure out pa
preferred
pressure out pb
preferred
pressure out
p_out

Parameters

rho	Mass density of the fluid [kg/m3].
GLeak	Conductance of the laminar leakage flows [m3/s.Pa], GLeak >= 0.

Amax	Maximum valve areas when the valves are open [m2], Amax > 0.
Cd	Discharge coefficient [], Cd > 0.
f	Bandwidth of the valve dynamics [Hz], f > 0.
d	Damping of the valve dynamics [], d > 0
p_o	Overlap pressure [Pa].

TwoTwoWayValve-States

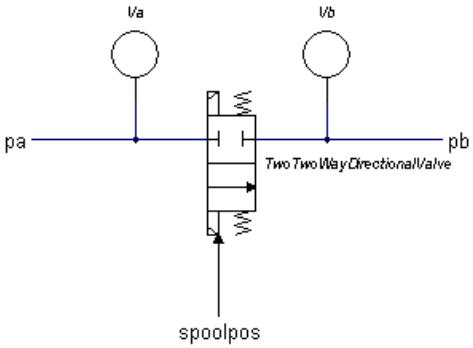
Library

Iconic Diagrams\Hydraulics\Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description



This model describes a 2/2-way directional control valve with second order spool dynamics and lumped volumes. The flow through the valve is described as laminar/ turbulent flow through an orifice. A detailed description of the valve can be found in TwoTwoWayDirectionalValve.htm.

Interface

Ports	**Description**
pa, pb	Both terminals of the valve.
Causality	
preferred effort out pa	
preferred effort out pb	
Inputs	
spoolpos	position of the spool valve spoolpos < 0.5 => valve is closed spoolpos >= 0.5 => valve is open

Parameters

TwoTwoWayDirectionalValve \rho	Mass density of the fluid [kg/m3].
TwoTwoWayDirectionalValve \GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0.
TwoTwoWayDirectionalValve \Amax	Maximum spool valve area when the valve is open [m2], Amax > 0.
TwoTwoWayDirectionalValve \Cd	Discharge coefficient [], Cd > 0.
TwoTwoWayDirectionalValve \overlap	The valve overlap in closed position [], -1 < overlap < 1.
TwoTwoWayDirectionalValve \f	Bandwidth of the spool dynamics [Hz], f > 0.
TwoTwoWayDirectionalValve \d	Damping of the spool dynamics [], d > 0.
Va\V	Volume of oil under pressure [m3]
Va\B	Effective bulk modulus [Pa]
Va\p_initial	The starting pressure of the volume [Pa]
Vb\V	Volume of oil under pressure [m3]
Vb\B	Effective bulk modulus [Pa]
Vb\p_initial	The starting pressure of the volume [Pa]

TwoTwoWayProportionalValve-States

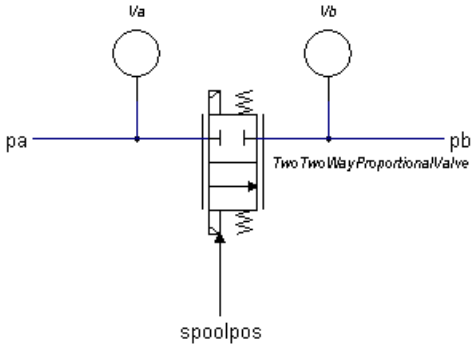
Library

Iconic Diagrams\Hydraulics\Valves

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description



This model describes a 2/2-way proportional control valve with second order spool dynamics and lumped volumes. The flow through the valve is described as laminar/ turbulent flow through an orifice. A detailed description of the valve can be found in TwoTwoWayProportionalValve.htm.

Interface

Ports	**Description**
pa, pb	Both terminals of the valve.
Causality	
preferred effort out pa	
preferred effort out pb	
Inputs	
spoolpos	position of the spool valve: 0 =closed, 1 = open 0 <= spoolpos <= 1
Parameters	
TwoTwoWayDirectionalValve \rho	Mass density of the fluid [kg/m3].
TwoTwoWayDirectionalValve \GLeak	Conductance of the laminar leakage flow [m3/s.Pa], GLeak >= 0.
TwoTwoWayDirectionalValve \Amax	Maximum spool valve area when the valve is open [m2], Amax > 0.
	Discharge coefficient [], Cd > 0.
	The valve overlap in closed position [], -1 < overlap < 1.

TwoTwoWayDirectionalValve	Bandwidth of the spool dynamics [Hz], f > 0.
\Cd	Damping of the spool dynamics [], d > 0.
TwoTwoWayDirectionalValve	Volume of oil under pressure [m3]
\overlap	Effective bulk modulus [Pa]
TwoTwoWayDirectionalValve	The starting pressure of the volume [Pa]
\f	Volume of oil under pressure [m3]
TwoTwoWayDirectionalValve	Effective bulk modulus [Pa]
\d	The starting pressure of the volume [Pa]
Va\V	
Va\B	
Va\p_initial	
Vb\V	
Vb\B	
Vb\p_initial	

Volumes

Accumulator

Library

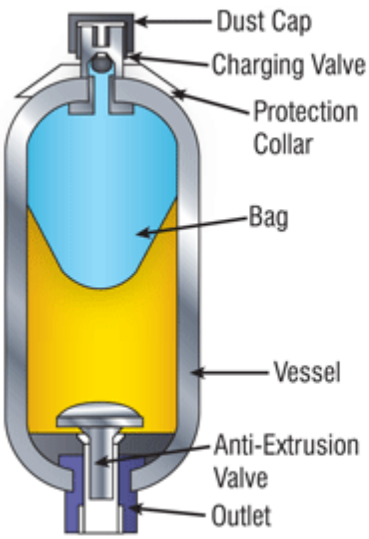
Iconic Diagrams\Hydraulic\Volumes

Use

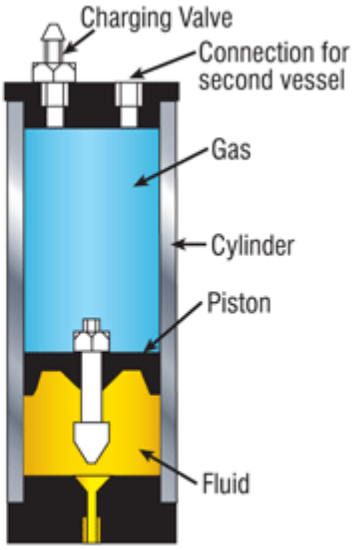
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

Accumulators consist of a gas filled chamber, pressurized by an oil filled chamber. The chambers are separated by a bladder or piston. If oil enters the chamber the gas chamber will reduce in size and the pressure will rise. If the oil pressures sinks, the gas chamber will expand and drive oil out of the oil chamber. The accumulator thus acts a a storage of hydraulic energy. Most accumulators are filled with nitrogen gas which is also used in this model. You can change to other gasses by changing the gas parameters.



Bladder Accumulator



Piston Accumulator

Use

When preparing an accumulator for use with the oil at ambient pressure, the gas is pre-charged to a certain pressure *p_pr*. The gas chamber will expand its maximum size which is called the accumulator volume *V*. During operation, oil will flow into the chamber and the gas chamber will change size. The oil and gas pressure will be in balance and indicated with the gas pressure *p_gas*. The resulting gas volume is name *V_gas*. We assume that during pre-charging the gas temperature is constant and equal to the ambient temperature *T_amb*. During operation, the gas temperature may change but the ambient temperature is assumed constant.

Thermal Model

Due to compression, the gas temperature will increase and the accumulator will heat up. This heat may be lost to the environment. In this model the heat loss is modeled by a first order transfer function using a thermal time constant *tau*. If the accumulator is heated up and the gas keeps its volume, an exponential temperature decrease will be found. The thermal time constant is the time where the temperature is decreased by 63%. However, the gas will be compressed if the temperature sinks. As a rule of thumb take for the thermal time constant the time it takes for the temperature to sink by 50%. By making tau infinite, the accumulator does not convert heat to the environment.The change is gas volume is then called *adiabatic* expansion or compression. By making tau very small, the accumulator instantly convert all heat to the environment.The change is gas volume is then called *isothermal* expansion or compression.

Gas Laws

The accumulator model is based on the Van der Waals equation for a real gas. This model takes into account that gas gas particles have finite size and attractive forces. The Van der Waals model uses the critical gas temperature *T_cr* and the critical gas pressure *p_cr* as parameters. By taking the parameter *T_cr* = 0 the Van der Waals model changes into the ideal gas model.

By changing the parameters, the accumulator model can represent various gas models:

1) Ideal Gas: set *T_cr* = 0
2) Ideal Gas, Adiabatic Expansion: set *T_cr* = 0 and *tau* very large (i.e 1.0e9)
3) Ideal Gas, Isothermal Expansion: set T_cr = 0 and tau very small (i.e 1.0e-9)
4) Real Gas: set the critical temperature of the gas (i.e. nitrogen: T_cr = 126.2 {K}) and set the thermal time constant to a specific value

Interface

Ports	**Description**
p	hydraulic port
Causality	
fixed volume flow	
out	
Parameters	
p_vapour	vapour pressure [Pa].
G	conductance of the accumulator input [m3/s.Pa].
V	max gas volume of the accumulator [m3]
p_pr	pre-charge pressure [Pa]
p_oil	oil pressure at the start [Pa]
T_amb	ambient temperature [K]
tau	thermal time constant [s]
R_s	specific gas constant [J/kg.K]
c_v	specific heat of the gas at constant volume [J/kg.K]
T_cr	critical gas temperature [K]
p_cr	critical gas pressure [Pa]

TankNoRes.em

Library

Iconic Diagrams\Hydraulic\Volumes

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model represents a tank with a preload pressure against atmosphere. The default preload pressure of 0 [Pa] means that the tank has atmospheric pressure. A preload pressure of 1e5 [Pa] means that the tank has an absolute pressure of 2e5 [Pa] etc.

Interface

Ports	Description
p	

Causality

fixed pressure out

Parameters

p_preload	tank pressure [Pa]

TankRes

Library

Iconic Diagrams\Hydraulic\Volumes

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model represents a tank with a preload pressure against atmosphere and a resistance of the tank pipe. The default preload pressure of 0 [Pa] means that the tank has atmospheric pressure. A preload pressure of 1e5 [Pa] means that the tank has an absolute pressure of 2e5 [Pa] etc.

There is no check on the validity of laminar flow in this component!

Interface

Ports	Description
p	

Causality

fixed volume flow

out p

Parameters

p_preload	tank pressure [Pa]
G	Conductance of laminar resistance [m3/s.Pa]

Volume

Library

Iconic Diagrams\Hydraulic\Volumes

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model represents a lumped volume with a constant bulk modulus. If the calculated pressure falls below the vapour pressure the pressure is NOT limited to the vapour pressure.

Interface

Ports	**Description**
p	
Causality	
preferred	
pressure out	
Parameters	
V	Volume of oil under pressure [m3]
B	Effective bulk modulus [Pa]
p_initial	The starting pressure of the volume [Pa]

ParasiticVolume

Library

Iconic Diagrams\Hydraulic\Volumes

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Hydraulics).

Description

This model represents a tiny volume (default: 1 ml) if the causality of the port is pressure out, and a zero flow, if the causality is flow out. A parasitic volume is a tiny volume that can be added to elements to make them more easy to simulate. Some elements require a pressure as input variable. Coupling two such elements leads to a mathematical problem: both yield a volume flow as function of the input pressure. Adding a parasitic volume in between will solve the mathematical problem (the parasitic volume calculates the pressure) without changing the dynamics much (the volume is small).

The parasitic element has a likes causality. If two parasitic volumes are coupled, one of them will get a pressure out causality (representing a tiny volume) and the other will get a flow out casuality (representing a zero flow). I.e only one of them is a volume and the other becomes ineffective.

Interface

Ports	**Description**
p	
Causality	
likes pressure out	
Parameters	**(hidden)**
V	Volume of oil under pressure [m3]
B	Effective bulk modulus [Pa]
p_initial	The starting pressure of the volume [Pa]

9.2.4 Mechanical

Mechanical

The Mechanical library contains components which are very useful for modeling mechanical systems. The library contains the following sections:

- Mechanics
  - Rotation
    - Actuators
    - Components
    - Friction
    - Gears
    - Sensors
  - Translation
    - 2D Small Rotation
    - 3D Small Rotation
    - Actuators
    - Components
    - Friction
    - Sensors
    - Transmission

Rotation

Actuators

AccelerationActuator-Relative

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model represents an ideal actuator. An angular acceleration input signal is integrated to an angular velocity difference between its two terminals:

p_high.omega = p_low.omega + int(alpha,omega_initial);
p_low.T = p_high.T = indifferent;

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Rotation port p.
Causality	
fixed angular velocity out	
Input	
alpha	Angular acceleration [rad/s^2].
Parameters	
omega_initial	Initial angular velocity output of the integration [rad/s].

AccelerationActuator

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model represents an ideal actuator. An angular acceleration input signal is integrated to an angular velocity at the rotation port. The actuator is mounted to the fixed world:

```
p.omega = int(alpha,omega_initial);
p.T = indifferent;
```

Interface

Ports	**Description**
p	Rotation port.
Causality	
fixed angular velocity out	
Input	
alpha	Angular acceleration [rad/s^2].
Parameters	
omega_initial	Initial angular velocity at the port [rad/s].

ACMotor-TorqueLoop.em

Note

This model was named ServoMotor.emx in previous versions of 20-sim!

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

Industrial servo motors in the medium to high power range generally consist of an AC-electric and drive (the digital controlled current supply of the motor). The drive takes care of the correct supply of current to let the motor follow every desired path. Unfortunately the exact implementation and performance of drives and motors is one of the trade secrets of the commercial motor suppliers. Fortunately modern servo-motors are designed well enough to be described by general parameters that can be found in every data sheet.

The servo motor model describes a digitally controlled AC-motor that is described by general parameters. The model is suited for modeling machine dynamics, i.e. models where the machine behavior is the topic of interest and not the motor itself. For studying the exact motor behavior (temperature, wear, vibration etc.) other motor models should be used. A complete description of the motor is given in the second part of this topic.

Interface

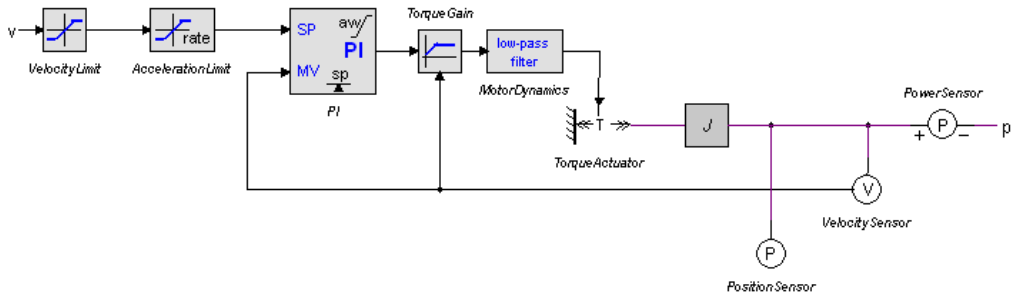
Ports	**Description**
p	Rotation port.
Causality	
preferred angular velocity out p	
Inputs	
v	velocity setpoint [rad/s]
Parameters	
VelocityLimit	Maximum motor speed (>0) [rad/s]
\v_max	Maximum allowed acceleration (>0) [rad/s^2]
AccelerationLimit	Bandwidth drive (half of the sample frequency, >0) [Hz]
\a_max	Proportional gain []
AccelerationLimit	Integral time constant (>0) [s]
\f_e	Tracking time constant (>0) [s]
PI\K	Maximum motor torque (>0) [Nm]
PI\Ti	Motor speed where torque gain starts to drop (>0) [rad/s]
PI\Ta	
PI\T_max	
TorqueGain\vd	
TorqueGain\g0	Torque gain at zero motor speed (>0, choose equal to 1 to turn off torque gain drop []
MotorDynamics\fc	Bandwidth drive and motor electronics [Hz]
Inertia\J	Motor inertia [kg.m^2/rad]

Variables	There are some variables that can be of interest when inspecting the servo performance:
PI\error	The error between setpoint and motor velocity [rad/s].
PositionSensor	Motor Angle [rad]
\phi	Motor Speed [rad/s]
VelocitySensor	Motor Acceleration [rad/s2]
\omega	Output Torque [Nm].
Inertia\alpha	Output Power [W].
PowerSensor\T	

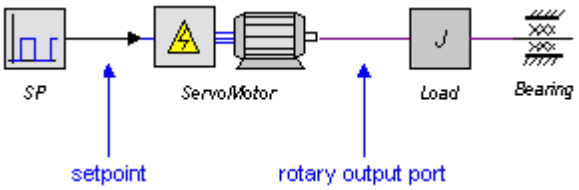
PowerSensor\P

Model

The complete servo motor model is shown below. Every block that is used to form this model will be explained in the next sections. The model has a velocity input signal v and a rotary output port p.



The servo motor model can be used in a machine model as shown in the example below.



Blocks

Velocity Limit

Most servo motors have a velocity limit to prevent the motor from damage. This limit can be set in the velocity VelocityLimit block. If no limit is known choose twice the nominal velocity.

Acceleration Limit

To prevent the motor from damage, most servos have a maximum acceleration limit. This limit is very important in a model because it can seriously degrade the systems performance.

To find the acceleration, the derivative of the input must be calculated. This is done by means of a first order differentiation with bandwidth f_c. The bandwidth is equal to one half the of the sample frequency of the velocity control loop. Older drives may have loops that run on sample frequencies ranging from 100 Hz to 500 Hz. Modern drives can run the velocity control loop on sample frequencies of 2 kHz or more.

To switch the maximum acceleration limit off, choose the maximum acceleration parameter high enough and choose the bandwidth sufficiently high, preferably 10 times as large as the bandwidth f_e of the motor electronics (specified in the motor dynamics block).

PI-Controller

Most servos will use velocity as the setpoint signal. To calculate the desired torque out of the velocity setpoint, PI-controllers are widely used. For many servos, the controller parameters can be changed by the user, but this is not a very easy task. Therefore these servos often offer automatic tuning facilities.

In 20-sim you can use the Optimization toolbox for automatic tuning. A good starting point is to take the error variable of the PI-controller and minimize it by changing the controller parameters Kp , Ti and Ta. It is important to realize that automatic tuning will tune the controller parameters for a certain task. If you run another task, i.e. perform a simulation with another setpoint, the results may degrade. For varying tasks it is better to use parameters the give a good response over a wide area of loads. To help the user find good starting values, the table below shows some motors, loads and corresponding starting value for the controller parameters.

Motor				Load	controller		
Nominal Power [kW]	Nominal Torque [Nm]	Nominal Speed [rpm]	Peak Torque [Nm]	Load + Motor Inertia [kg.m2]	Kp	Ti	Ta
0.38	1.2	3000	4.6	0.005	250	0.1	0.1
1.13	3.6	3000	14	0.002	1000	0.4	0.4
7.54	24	3000	134	0.1	5000	2	2
11	36	3000	90	0.4	15000	6	6
55	177	3000	531	2	75000	30	30
560	1792	3000	6092.8	25	750000	300	300

The user has to find the load that should be driven in terms of the rotational inertia. If there is a gear that changes the rotary motion to a linear motion this means the load has to be divided by the square of the gear ratio to get the corresponding load in the rotational domain. Do not forget to add the inertia of the motor itself! For many gear ratios the load in terms of rotational inertia will be of the same magnitude as the motor inertia.

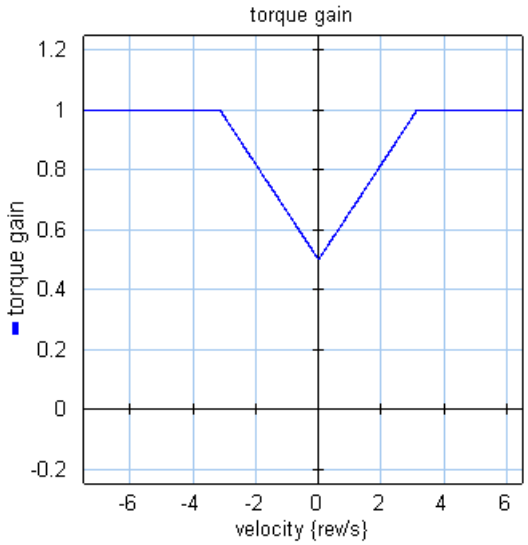
In this model PI-controller with an anti-windup facility is used. This controller is the heart of the servo motor model. It is therefore important to know something about PI-control. More information can be found in the PID controllers section of the library.

AC-motors have a limited output torque which is a complex function of the controller settings, the applied speed, duty cycle, temperature and more. In the servo motor model simply the torque limit of the PI-controller T_max is used which limits the generated torque to a maximum. Because this is a simplification of the real behavior, the maximum torque limit should only be used to inspect what happens to the system when the motor torque is approaching its maximum value. For a proper evaluation of its system, the user should also look at the unconstrained torque. Run a simulation and note the amount of time that the motor runs at a maximum torque. This is called the duty cycle. Every motor manual will have tables showing duty cycles and the corresponding chances of overheating.

Torque Gain

Standard AC-motors, normally designed to run at base speeds between 850 to 3500 rpm, are not particularly well suited for low-speed operation, as their efficiency drops with the reduction in speed. They may also be unable to deliver sufficient smooth torque at low speeds.

An ideal motor will always have a torque gain of 1, i.e. an input signal of 1 Nm will always result in an output torque of 1 Nm. Most industrial motors have a torque gain that is only equal to 1 for higher speeds. At low speeds the torque gain will drop due to electronic limitations. This behavior is represented by the graph below. At speeds below a drop-off speed vd the torque gain drops linearly to a user defined zero speed torque gain g0. If no torque gain drop is desired, choose g0 equal to 1.



Motor Dynamics

The electrical circuits of the drive and wiring of the motor have a limited bandwidth. This is modeled with a first order low-pass filter with a bandwidth f_e. Typical motor bandwidths vary from 3 kHz tot 20 kHz.

Torque source

In the torque source block the torque signal is converted to an iconic diagram port p.

Inertia

This block describes the rotor inertia. The equations of motion of the inertia are used to get the angular acceleration without needing derivatives.

Sensors

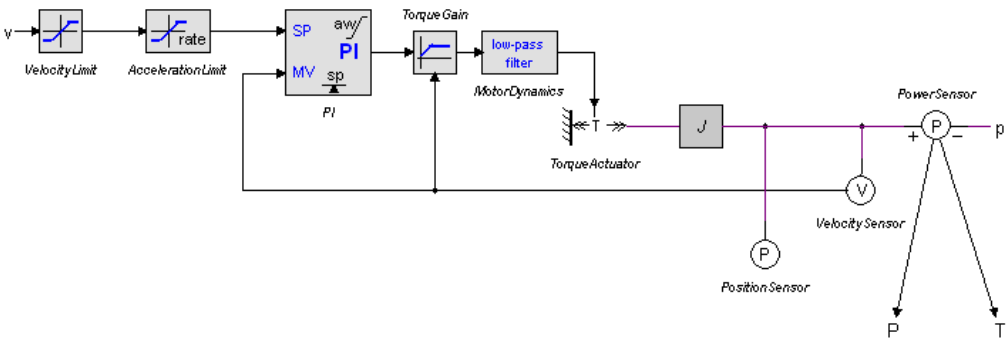
Finally four sensors are available to plot the position, velocity, acceleration and output power.

Variables

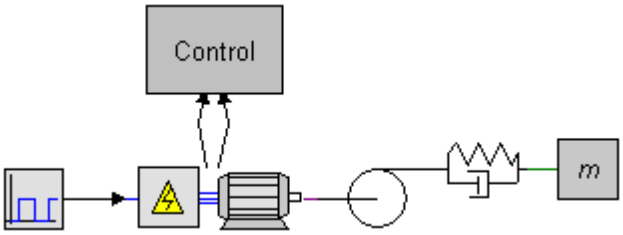
There are some variables in the model that may be of special interest

PI\error	The error between setpoint and motor velocity [rad/s].
PositionSensor	Motor Angle [rad]
\phi	Motor Speed [rad/s]
VelocitySensor	Motor Acceleration [rad/s2]
\omega	Output Torque [Nm].
Inertia\alpha	Output Power [W].
PowerSensor\T	
PowerSensor\P	

All these variables are output signals of their respective blocks. By inserting extra outputs in the servo motor model you can use as output signals for usage in other parts of your model. In the pictures below is shown how to define signals for the torque and power.



In the servo model define a torque and a power output and connect them.



One level higher you can use these signals for modeling, for example in a high level control unit.

ACMotor

Library

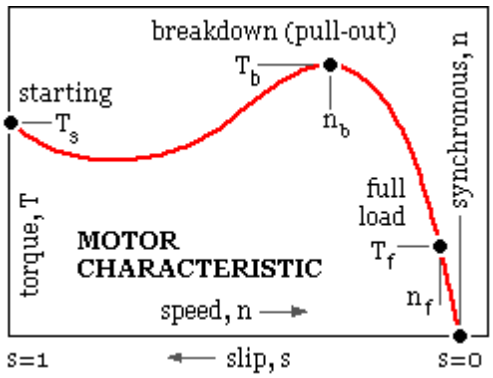
Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This is a model of an industrial AC electric motor with squirrel cage rotor. The motor is not controlled by a power electric convertor. Instead its performance is directly defined by a torque speed curve.



The torque speed curve is fully defined by the parameters shown in the picture. The synchronous speed is defined as:

*n = 2 * voltage frequency / no. of poles*

Interface

Ports	**Description**
p	Output axis (Rotation)
Parameters	
Ts	start torque (zero speed) [Nm]
n	synchronous speed (zero torque) [rad/s]
Tm	maximum torque [Nm]
nm	speed at maximum torque [rad/s]
Tf	full load torque [Nm]
nf	speed at full load torque [rad/s]
J	motor inertia [kgm^2/rad]

DCMotor

Library

Iconic Diagrams\Electric\Actuators
Iconic Diagrams\Mechanical\Electric

Implementations

Default
IR

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric, Electric).

Description - Default

This models represents an ideal DC-motor with no energy loss. The electric port has separate high and low terminals. The equations are

```
p1.i = p1_high.i = p1_low.i;
p1.u = p1_high.u - p1_low.u;
```

The model can have mixed forms of causality

```
p1.u = k * p2.omega;
p2.T = k * p1.i;
```

or:

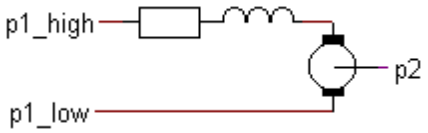
```
p2.omega = p1.u / k;
p1.i = p2.T / k;
```

Interface - Default

Ports	**Description**
p1_high, p1_low	Both terminals of the Electric port p1.
p2	Rotation port.
Causality	
mixed	See equations above.
Parameters	
k	motor constant [Nm/A]

Description - IR

This models represents an ideal DC-motor with inductance and resistance.



The electric port has separate high and low terminals. The equations are

p1.i = p1_high.i = p1_low.i;
p1.u = p1_high.u - p1_low.u;

Interface - IR

Ports	**Description**
p1_high, p1_low	Both terminals of the Electric port p1.
p2	Rotation port.
Causality	
mixed	See equations above.
Parameters	
k	motor constant [Nm/A]
L	motor inductance [H]
R	motor resistance [Ohm]

PositionActuator-Relative

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

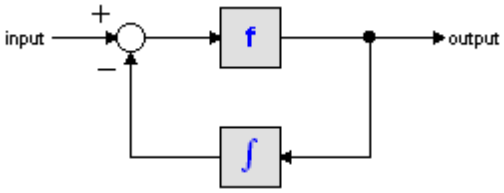
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model represents an ideal actuator. An angle input signal is differentiated by a state variable filter to an angular velocity difference between its two terminals:

p_high.omega = p_low.omega + dphi/dt;
p_low.T = p_high.T = indifferent;

Differentiation is performed by a state variable filter:



The S-domain function of this filter is equal to:

$$output = \frac{s}{\frac{s}{f}+1} \cdot input$$

where f is the cut-off frequency. For very high values of f, the output becomes the pure derivative of the input. High values of f, however, increase simulations times. A good trade-off is a starting value of 1e5.

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Rotation port p.
Causality	
fixed angular velocity out	
Input	
phi	Angle [rad].
Parameters	
f	Cut-off frequency of the differentiation [Hz].
omega_initial	Initial angular velocity output of the differentiation [rad/s].

ServoMotor

Library

Iconic Diagrams\Mechanical\Rotation\Actuators
Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This is a masked model which opens the Servo Motor Editor when edited. The servo Motor Editor is a tool that can shows the torque speed plots numerous permanent magnet motors and can generate a dynamic model from any motor that you select. The following motor types are supported:

1. Brush DC
2. Brushless DC (trapezoidal EMC and square wave currents)
3. AC synchronous (sinusoidal EMC and sinusoidal currents)
4. AC synchronous linear (sinusoidal EMC and sinusoidal currents)

Interface

Depending on the type of motor that you have selected, the interface can vary:

DC Brush

Ports	**Description**
p	Rotation port.
Causality	
fixed rotational velocity out	
Input	
i	The input current [A]

DC Brushless

Ports	**Description**
p	Rotation port.
Causality	
fixed rotational velocity out	
Input	
i	The maximum input current [A]

AC Synchronous

Ports	Description
p	Rotation port.
Causality	
fixed rotational velocity out	
Input	
i_rms	The rms phase current [A]

AC Synchronous Linear

Ports	Description
p	Translation port.
Causality	
fixed velocity out	
Input	
i_rms	The rms phase current [A]

For more information on the parameters an variables of this model is referred to the Mechatronic Toolbox.

PositionActuator

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

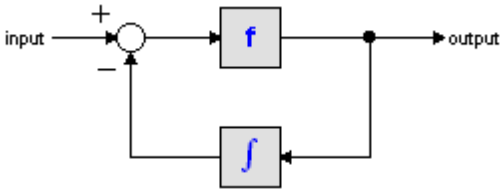
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model represents an ideal actuator. An angle input signal is differentiated by a state variable filter to an angular velocity at the rotation port. The actuator is mounted to the fixed world:

```
p.omega = dphi/dt;
p.T = indifferent;
```

Differentiation is performed by a state variable filter:



The S-domain function of this filter is equal to:

$$output = \frac{s}{\frac{s}{f}+1} \cdot input$$

where *f* is the cut-off frequency. For very high values of *f*, the output becomes the pure derivative of the input. High values of *f,* however, increase simulations times. A good trade-off is a starting value of 1e5.

Interface

Ports	**Description**
p	Rotation port.
Causality	
fixed angular velocity out	
Input	
phi	Angle [rad].
Parameters	
f	Cut-off frequency of the differentiation [Hz].
omega_initial	Initial angular velocity at the port [m/s].

Steppermotor

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

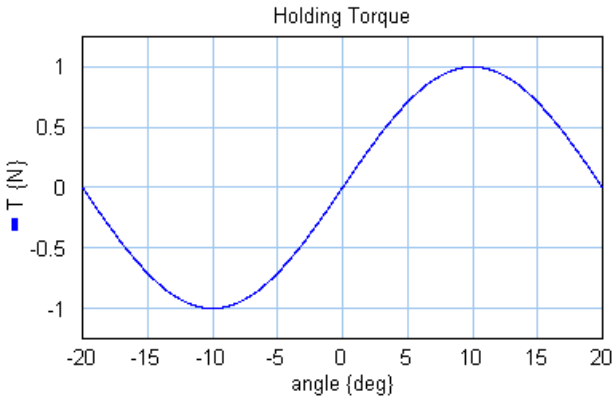
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

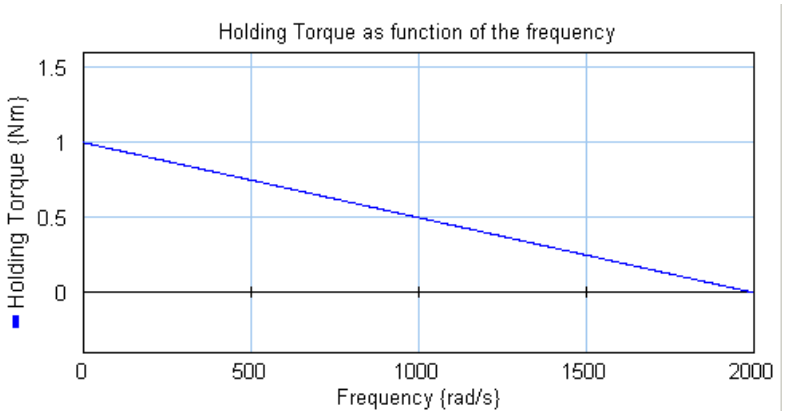
A stepper motor is a device used to convert electrical pulses into discrete mechanical rotational movements. The minimum movement, invoked by one pulse, is called the *step angle.* The electrical pulses are mostly generated by a pulse generator that converts a setpoint change in a corresponding amount of pulses. This model describes a combination of a pulse generator and stepper motor.

At standstill the torque required to deflect the motor a full step is called the *holding torque*. The holding torque normally is much higher than required to drive the load, and thus acts as a strong brake to hold the load. The holding torque as function of the output angle is shown in the graph below. Around zero angle the curves shows the behavior of a spring. By changing the input by one step, the force will increase to its maximum and move the the load.



Motor torque a s function of the angle (holding toque = 1 [N], step angle = [10°]).

If the subsequent steps are generated fast enough the holding torque will start to decrease. The maximum holding torque as function of the rotation speed is shown in the figure below. This force curve is commonly describe as the *pull out* curve.



Pull out curve: maximum generated torque as a function of the rotational speed.

Ports	**Description**
input	Desired output angle.
p_out	Rotation port.
Causality	
fixed current out p_in	
preferred angular velocity out p_out	
Inputs	
input	Stepper motor input
Parameters	
step_angle	Minimum change of the output axis [deg].
tau	Time constant coil [s].
Th	Holding torque [Nm].
J	Inertia rotor [Nm^2]
B	Relative damping at the stepper motor resonance [].
fmax	Maximum frequency motor (frequency at which the torque gets zero) [rad/s].

Torque-Relative

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents an ideal actuator. The actuator applies a torque between its two terminals. This torque can be set to a certain constant value, the angular velocity is indifferent.

p_high.T = p_low.T = T

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Rotational port p.
Causality	
fixed torque out	
Input	
T	Torque [Nm].

Torque

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents an ideal actuator. The actuator is mounted to the fixed world and applies a torque. The torque can be set to a certain constant value, the angular velocity is indifferent.

p.T = T;

Interface

Ports	**Description**
p	Rotation port.
Causality	

fixed torque out

Parameters

T	Torque [Nm].

TorqueActuator-Relative

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model represents an ideal actuator. The actuator applies a torque between its two terminals. This torque can be set to a (fluctuating) value given by the input signal *T*, the angular velocity is indifferent.

$$p\_high.T = p\_low.T = T$$

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Rotation port p.

Causality

fixed torque out

Input

T	Torque [Nm].

TorqueActuator

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model represents an ideal actuator. The actuator is mounted to the fixed world and applies a torque. The torque can be set to a (fluctuating) value given by the input signal *T*, the angular velocity is indifferent.

p.T = T;

Interface

Ports	**Description**
p	Rotation port.
Causality	
fixed torque out	
Input	
T	Torque [Nm].

Velocity-Relative

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents an ideal actuator. The actuator applies an angular velocity difference between its two terminals. This angular velocity can be set to a constant value *omega*, the force is indifferent.

p_low.omega = p_high.omega + omega

Interface

Ports	**Description**

p_high, p_low	Both terminals of the Rotation port p.

Causality

fixed angular velocity out	

Parameter

v	Angular velocity [rad/s].

Velocity

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents an ideal actuator. The actuator is mounted to the fixed world and applies an angular velocity. This angular velocity can be set to a certain constant value, the torque is indifferent.

```
p.omega = omega;
```

Interface

Ports	Description
p	Rotation port.

Causality

fixed angular velocity out	

Parameters

omega	Angular velocity [rad/s].

VelocityActuator-Relative

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model represents an ideal actuator. The actuator applies an angular velocity difference between its two terminals. This angular velocity can be set to a (fluctuating) value given by the input signal *v*, the force is indifferent.

p_low.omega = p_high.omega + omega

Interface

Ports	Description
p_high, p_low	Both terminals of the Rotation port p.
Causality	
fixed angular velocity out	
Input	
omega	Angular velocity [rad/s].

VelocityActuator

Library

Iconic Diagrams\Mechanical\Rotation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model represents an ideal actuator. The actuator is mounted to the fixed world and applies an angular velocity. This angular velocity can be set to a (fluctuating) value given by the input signal *omega,* the torque is indifferent.

p.omega = omega;

Interface

Ports	Description
p	Rotation port.
Causality	
fixed angular velocity out	
Input	

omega	Angular velocity [rad/s].

Components

Backlash

Library

Iconic Diagrams\Mechanical\Rotation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents backlash by a spring damper system equivalent to the translational backlash model. The port p of this model has separate high and low terminals. The equations are:

p.T = p_high.T = p_low.T
p.omega = p_high.omega - p_low.omega

Interface

Ports	**Description**
p_high, p_low	Both terminals of port p (Rotation).
Causality	
fixed torque out	
Parameters	
s	Interval of the play [rad]
k1	Stiffness in the play [Nm/rad]
k2	Stiffness outside the play [Nm/rad]
d1	Damping inside the play [Nms/rad]
d2	Damping outside the play [Nms/rad]
ep	Relative round off (1e-6 -> sharp edges, 1e-2 -> smoother)
Parameters	
x_initial	Initial position in the play [rad], -s/2 < x_initial < s/2

Bearing

Library

Iconic Diagrams\Mechanical\Rotation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

In this model bearing friction is represented as linear viscous friction. The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. 20-sim will automatically create equations such that the resulting torque *p.T* is equal to the sum of the torques of all connected ports p1 .. pn. The angular velocities of all connected ports are equal to *p.omega*. The model can have an torque out as well as an angular velocity out causality. In the last case the constitutive equation, as shown below, is simply inverted:

```
p.T = sum(p1.T, p2.T, ....)
p.omega = p1.omega = p2.omega = ....
```

Torque out causality:

```
p.T = d * p.omega;
```

Angular velocity out causality:

```
p.omega = p.T / d;
```

Interface

Ports	**Description**
p[any]	Any number of connections can be made (Rotation).
Causality	
indifferent	
Parameters	
d	Damping [Nms/rad]

Brake

Library

Iconic Diagrams\Mechanical\Rotation\Components

Implementations

C

V
CV
SCVS
LuGre

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Introduction

The brake models represent a disk brake or other type of brake where a rotation is stopped by applying a friction force. The amount of friction depends on the normal force that is applied and the friction function that is used. The normal force is given by the input signal *Fn.* The brake is mounted to the fixed world.

The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. 20-sim will automatically create equations such that the resulting torque *p.T* is equal to the sum of the torques of all connected ports p1 .. pn. The angular velocities of all connected ports are equal to *p.omega*.

p.T = sum(p1.T, p2.T, ...)
p.omega = p1.omega = p2.omega = ...

Due to the use of normal force, the brake models all have a fixed torque out causality. The constitutive equations are therefore described as:

```
p.T = Fn * f(p.omega);
```

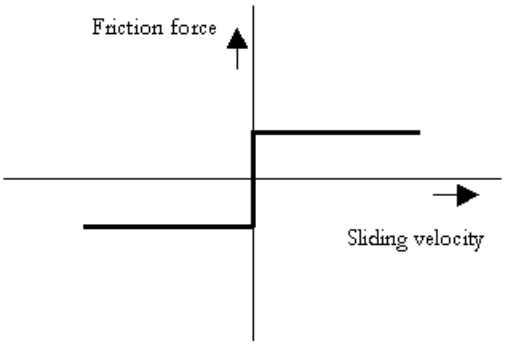
with *f* the friction function.

Description - C

This model represents a brake with braking force described as coulomb friction. The brake is mounted to the fixed world. The amount of friction depends on the normal force that is applied:

```
p.T = Fn*mu_c*tanh(slope*p.omega);
```

Fn: normal force (given by the input signal Fn)
mu_c: the coulomb friction coefficient
slope: the steepness of the coulomb friction curve.



Interface - C

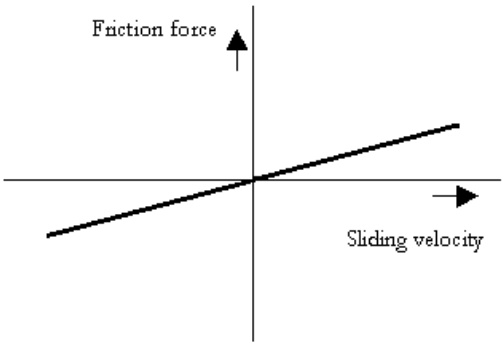
Ports	**Description**
p[any]	Any number of connections can be made (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
mu_c	Coulomb friction coefficient [m]
slope	Steepness of Coulomb friction curve [s/rad]

Description - V

This model represents a brake with braking force described as viscous friction. The brake is mounted to the fixed world. The amount of friction depends on the normal force that is applied:

```
p.T = Fn*mu_v*p.omega;
```

Fn: normal force (given by the input signal *Fn*)
mu_v: the viscous friction coefficient



Interface - V

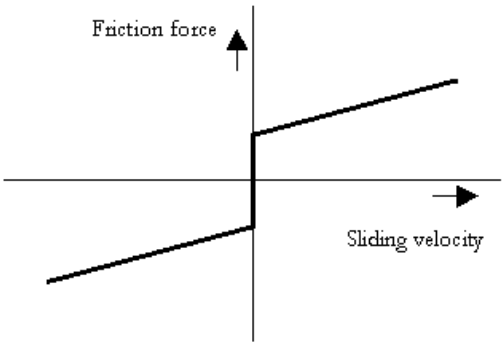
Ports	Description
p[any]	Any number of connections can be made (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
mu_v	Viscous friction coefficient [m.s/rad]

Description - CV

This model represents a brake with braking force described as coulomb plus viscous friction. The brake is mounted to the fixed world. The amount of friction depends on the normal force that is applied:

```
p.T = Fn*(mu_c*tanh(slope*p.omega) + mu_v*p.omega);
```

Fn: normal force (given by the input signal Fn)
mu_v: the viscous friction coefficient
mu_c: the coulomb friction coefficient
slope: the steepness of the coulomb friction curve.



Interface - CV

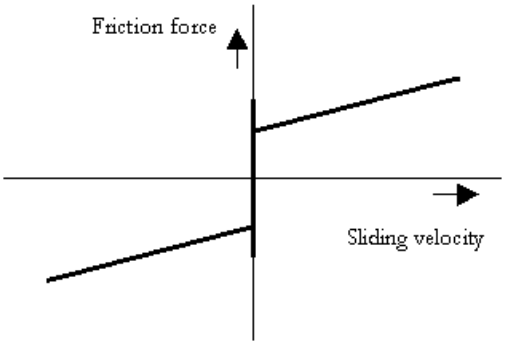
Ports	Description
p[any]	Any number of connections can be made (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
mu_v	Viscous friction coefficient [ms/rad]
mu_cv	Coulomb friction coefficient [m]
slope	Steepness of Coulomb friction curve [s/rad]

Description - SCVS

This model represents a brake with braking force described as static plus coulomb plus viscous plus Stribeck friction. The brake is mounted to the fixed world. The amount of friction depends on the normal force that is applied:

```
p.T = Fn *
(( mu_c + (mu_st*abs(tanh( slope*p.omega )) - mu_c)
* exp( -((p.omega / v_st)^2 )) ) * sign(p.omega)
+ mu_v * p.omega);
```

Fn: normal force (given by the input signal Fn)
mu_s: the static friction coefficient
mu_v: the viscous friction coefficient
mu_c: the coulomb friction coefficient
slope: the steepness of the coulomb and static friction curve.
v_st: the characteristic Stribeck velocity.



Interface - SCVS

Ports	Description
p[any]	Any number of connections can be made (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
mu_s	Static friction coefficient [m]
mu_v	Viscous friction coefficient [m.s/rad]
mu_cv	Coulomb friction coefficient [m]
slope	Steepness of Coulomb friction curve [s/rad]
v_st	Characteristic Stribeck velocity [rad/s]

Description - LuGre

This model represents a brake with braking force described by the LuGre friction model. The brake is mounted to the fixed world. The amount of friction depends on the normal force that is applied:

```
p.T = FN*f_lg(p.omega);
```

Fn: normal force (given by the input signal Fn)
f_lg: the LuGre friction model

Interface - LuGre

Ports	Description
p[any]	Any number of connections can be made (Rotation).
Causality	
Fixed Torque out	

Input

Fn	Normal force [N]

Parameters

mu_c	Coulomb friction coefficient
mu_s	Static friction coefficient [m]
mu_v	Viscous friction coefficient [ms/rad]
v_st	Characteristic Stribeck velocity [rad/s]
mu_k	rotational stiffness coefficient at zero speed [m/rad]

Clutch

Iconic Diagrams\Mechanical\Rotation\Components

Implementations

C
V
CV
SCVS
LuGre

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Introduction

The clutch models model represent friction relative to other objects. The amount of friction depends on the normal force that is applied and the friction function that is used. The normal force is given by the input signal *FN.*

The port *p* of the clutch model has separate high and low terminals. The equations are:

p.T = p_high.T = p_low.T
p.omega = p_high.omega - p_low.omega

Due to the use of normal force, the clutch models all have a fixed torque out causality. The constitutive equations are therefore described as:

*p.T = Fn * f(p.omega);*

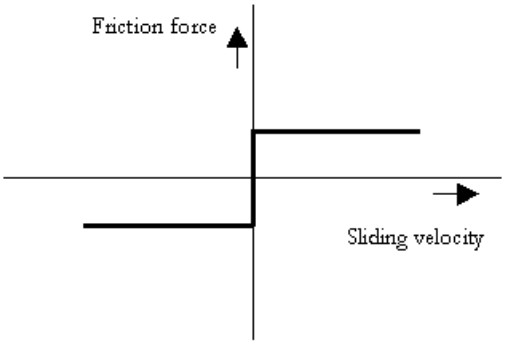
with *f* the friction function.

Description - C

This model represents bearing with friction force described as coulomb friction. The amount of friction depends on the normal force that is applied and the friction function that is used:

*p.T = Tc*tanh(slope*p.omega);*

Tc: coulomb friction
slope: the steepness of the coulomb friction curve.



Interface - C

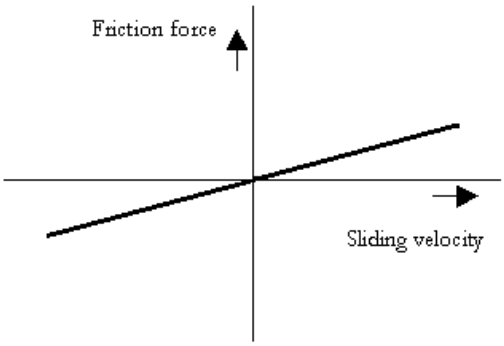
Ports	**Description**
p	Both terminals of port p (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
Tc	Coulomb friction [N.m]
slope	Steepness of Coulomb friction curve [s/rad]

Description - V

This model represents a clutch with friction force described as viscous friction. The amount of friction depends on the normal force that is applied and the friction function that is used:

*p.T = Fn*mu_v*p.omega;*

Fn: normal force (given by the input signal Fn)
mu_v: the viscous friction coefficient



Interface - V

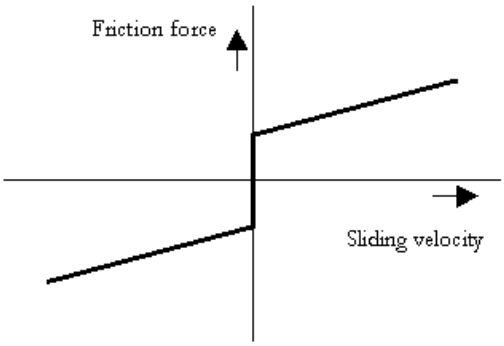
Ports	**Description**
p_high, p_low	Both terminals of port p (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
mu_v	Viscous friction coefficient [m.s/rad]

Description - CV

This model represents a clutch with friction force described as coulomb plus viscous friction. The amount of friction depends on the normal force that is applied and the friction function that is used:

p.T = Fn(Tc*tanh(slope*p.omega) + mu_v*p.omega);*

Fn: normal force (given by the input signal Fn)
mu_v: the viscous friction coefficient
Tc: the coulomb friction coefficient
slope: the steepness of the coulomb friction curve.



Interface - CV

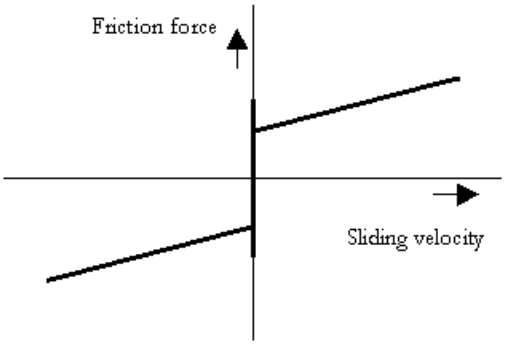
Ports	**Description**
p_high, p_low	Both terminals of port p (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
mu_v	Viscous friction coefficient [m.s/rad]
Tc	Coulomb friction [N.m]
slope	Steepness of Coulomb friction curve [s/rad]

Description - SCVS

This model represents a clutch with friction force described as static plus coulomb plus viscous plus Stribeck friction. The amount of friction depends on the normal force that is applied and the friction function that is used:

```
p.T = Fn *
(( Tc + (mu_st*abs(tanh( slope*p.omega )) - Tc)
* exp( -((p.omega / v_st)^2 )) ) * sign(p.omega)
+ mu_v * p.omega);
```

Fn: normal force (given by the input signal Fn)
mu_s: the static friction coefficient
mu_v: the viscous friction coefficient
Tc: the coulomb friction coefficient
slope: the steepness of the coulomb and static friction curve.
v_st: the characteristic Stribeck velocity.



Interface - SCVS

Ports	**Description**
p_high, p_low	Both terminals of port p (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
mu_s	Static friction coefficient [m]
mu_v	Viscous friction coefficient [m.s/rad]
Tc	Coulomb friction [N.m]
slope	Steepness of Coulomb friction curve [s/rad]
v_st	Characteristic Stribeck velocity [rad/s]

Description - LuGre

This model represents a clutch with friction force described by the LuGre friction model. The amount of friction depends on the normal force that is applied and the friction function that is used:

```
p.T = FN*f_lg(p.omega);
```

Fn: normal force (given by the input signal Fn)
f_lg: the LuGre friction model

Interface- LuGre

Ports	**Description**
p_high, p_low	Both terminals of port p (Rotation).
Causality	
Fixed torque out	

Input

Fn	Normal force [N]

Parameters

Tc	Coulomb friction coefficient
mu_s	Static friction coefficient [m]
mu_v	Viscous friction coefficient [m.s/rad]
v_st	Characteristic Stribeck velocity [rad/s]
mu_k	rotational stiffness coefficient at zero speed [m/rad]

Damper

Library

Iconic Diagrams\Mechanical\Rotation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents a linear damper. It can have an torque out as well as an angular velocity out causality. In the last case the constitutive equation, as shown below, is simply inverted. The port *p* of the damper model has separate high and low terminals. The equations are:

p.T = p_high.T = p_low.T
p.omega = p_high.omega - p_low.omega

Torque out causality:

*p.T = d * p.omega;*

Angular velocity out causality:

p.omega = p.T / d;

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Rotation port p.

Causality

indifferent

Parameters

d	damping [Nms/rad]

FixedWorld

Library

Iconic Diagrams\Mechanical\Rotation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents the fixed world (angular velocity = 0). The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. which gives the constitutive equations:

p1.omega = p2.omega = .. = pn.omega = 0;
p1.T = free; p2.T = free; ..; pn.T = free;

Interface

Ports	Description
p [any]	Any number of connections can be made.

Causality

Fixed angular velocity out	All ports have a fixed angular velocity out causality.

Friction

Iconic Diagrams\Mechanical\Rotation\Components

Implementations

C
V
CV
SCVS
LuGre

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Introduction

The friction model represent bearing friction with various friction models. The model have only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. 20-sim will automatically create equations such that the resulting force *p.F* is equal to the sum of the forces of all connected ports p1 .. pn. The velocities of all connected ports are equal to *p.v*.

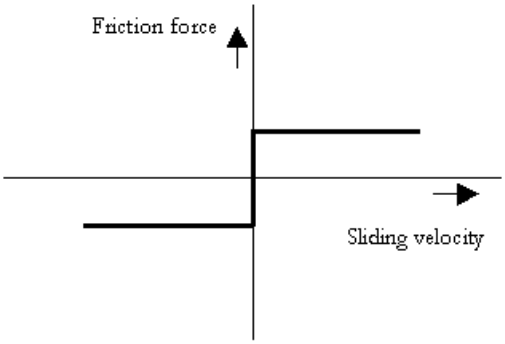
p.F = sum(p1.F, p2.F, ...)
p.v = p1.v = p2.v = ...

Description - C

This model represents a bearing with friction torque described as coulomb friction:

*p.T = Tc*tanh(slope*p.omega);*

Tc: the coulomb friction
slope: the steepness of the coulomb friction curve.



Interface - C

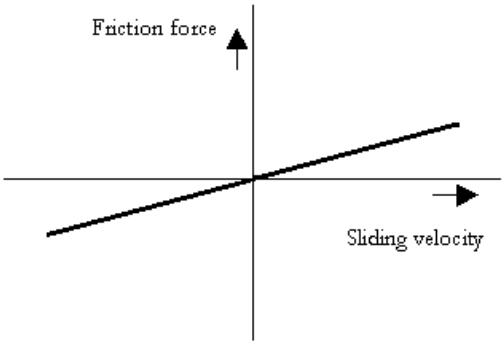
Ports	**Description**
p[any]	Any number of connections can be made (Rotation).
Causality	
Fixed torque out	
Parameters	
Tc	Coulomb friction [N.m]
slope	Steepness of Coulomb friction curve [s/rad]

Description - V

This model represents a bearing with friction force described as viscous friction:

*p.T = d*p.omega;*

d: the viscous damping



Interface - V

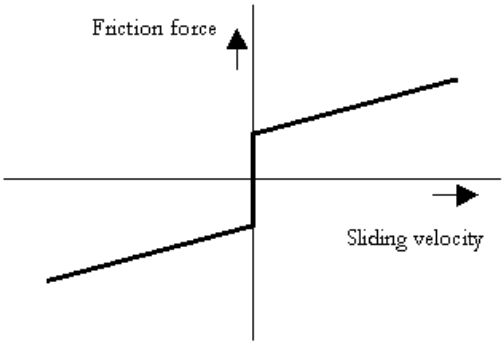
Ports	**Description**
p[any]	Any number of connections can be made (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
d	Viscous friction torque or damping [N.m.s/rad]

Description - CV

This model represents a bearing with friction force described as coulomb plus viscous friction:

*p.T = Tc*tanh(slope*p.omega) + d*p.omega;*

d: the viscous damping
Tc: the coulomb friction
slope: the steepness of the coulomb friction curve.



Interface - CV

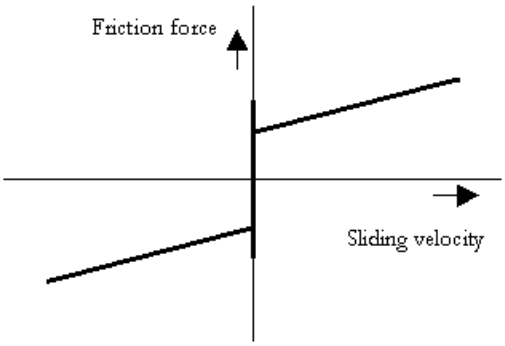
Ports	Description
p[any]	Any number of connections can be made (Rotation).
Causality	
Fixed torque out	
Input	
Fn	Normal force [N]
Parameters	
mu_v	Viscous damping [N.m.s/rad]
Tc	Coulomb friction [N.m]
slope	Steepness of Coulomb friction curve [s/rad]

Description - SCVS

This model represents a bearing with friction force described as static plus coulomb plus viscous plus Stribeck friction:

```
p.T = (( Tc + (Tst*abs(tanh( slope*p.omega )) - Tc)
* exp( -((p.omega / v_st)^2 )) ) * sign(p.omega)
+ d * p.omega);
```

Tst: the static friction
d: the viscous damping
Tc: the coulomb friction
slope: the steepness of the coulomb and static friction curve.
v_st: the characteristic Stribeck velocity.



Interface - SCVS

Ports	Description
p[any]	Any number of connections can be made (Rotation).
Causality	

Fixed torque out

Input

Fn	Normal force [N]

Parameters

Tst	Static friction torque [N.m]
d	Viscous friction torque or damping [N.m.s/rad]
Tc	Coulomb friction [N.m]
slope	Steepness of Coulomb friction curve [s/rad]
v_st	Characteristic Stribeck velocity [rad/s]

Description - LuGre

This model represents a bearing with friction force described by the LuGre friction model:

```
p.T = f_lg(p.omega);
```

f_lg: the LuGre friction model

Interface- LuGre

Ports	**Description**
p[any]	Any number of connections can be made (Rotation).

Causality

Fixed torque out

Input

Fn	Normal force [N]

Parameters

Tc	Coulomb friction [N.m]
Tst	Static friction torque [m]
d	Viscous damping [N.m.s/rad]
v_st	Characteristic Stribeck velocity [rad/s]
c	Stiffness at zero speed [N.m/rad]

Inertia

Library

Iconic Diagrams\Mechanical\Rotation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents an ideal rotational inertia. The element has a preferred angular velocity out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred torque out causality. The constitutive equations then contain a derivation. The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. 20-sim will automatically create equations such that the resulting torque *p.T* is equal to the sum of the torques of all connected ports p1 .. pn and that the angular velocities of all connected prots is equal to *p.omega*.

```
p.T = sum(p1.T, p2.T, ....)
p.omega = p1.omega = p2.omega = ....
```

angular velocity out causality (preferred):

```
alpha = p.T/J;
p.omega = int(alpha);
phi = int(p.omega);
```

torque out causality:

```
alpha = ddt(p.omega);
p.T = J*alpha;
phi = int(p.omega);
```

Interface

Ports	**Description**
p[any]	Any number of connections can be made (Rotation).
Causality	
preferred angular velocity out	An torque out causality results in a derivative constitutive equation.
Variables	
phi	angle [rad]
alpha	angular acceleration [rad/s^2]
Parameters	
J	moment of inertia [kgm^2]

Initial Values

p.omega_initial	The initial velocity of the inertia [rad/s].
phi_initial	The initial angle of the inertia [rad].

Node

Library

Iconic Diagrams\Mechanical\Rotation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This node model represents a structural connection between two (or more) shafts, where the velocity of all connected shafts is equal. The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc.

Interface

Ports	Description
p [any]	Any number of connections can be made (Rotation).

Spring

Library

Iconic Diagrams\Mechanical\Rotation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents an ideal rotational spring. The element has a preferred torque out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred angular velocity out causality. The constitutive equations then contain a derivation. The port *p* of the spring model has separate high and low terminals. The equations are:

p.T = p_high.T = p_low.T
p.omega = p_high.omega - p_low.omega

torque out causality (preferred):

phi = int(p.omega);
*p.T = c * phi;*

angular velocity out causality:

p.omega = ddt(phi);
phi = p.T/k;

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Rotational port p.
Causality	
preferred torque out	An angular velocity out causality results in a derivative constitutive equation.
Variables	
phi	torsion of the spring [rad]
Parameters	
c	Stiffness [Nm/rad]
Initial Values	
phi_initial	The initial torsion of the spring [rad].

SpringDamper

Library

Iconic Diagrams\Mechanical\Rotation\Components

Implementations

Default
Stiffness
Frequency

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description - Default

This model represents an ideal rotational spring with damper. The element has a preferred torque out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred angular velocity out causality. The constitutive equations then contain a derivation. The port *p* of the spring model has separate high and low terminals. The equations are:

p.T = p_high.T = p_low.T
p.omega = p_high.omega - p_low.omega

Torque out causality (preferred):

phi = int(p.omega);
*p.T = c * phi + d*p.omega;*

Angular velocity out causality:

p.omega = ddt(phi);
*phi = (p.T - d*p.omega)/c;*

Interface - Default

Ports	**Description**
p_high	Two ports of the spring (Rotation).
p_low	
Causality	
preferred torque out	An angular velocity out causality results in a derivative constitutive equation.
Variables	
phi	torsion of the spring [rad]
Parameters	
c	Rotational stiffness [Nm /rad]
d	Damping [Nms/rad]
Initial Values	
phi_initial	The initial torsion of the spring [rad].

Description - Stiffness

This model represents an ideal rotational spring with damper. The damping value (*d*) is calculated on the basis of a known stiffness (*c*), relative damping (*b*) and reference inertia (*J*). The inertia is only used to compute the damping (no actual mass is used in this component).

The element has a preferred torque out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred angular velocity out causality. The constitutive equations then contain a derivation. The port p of the spring model has separate high and low terminals. The equations are:

p.T = p_high.T = p_low.T
p.omega = p_high.omega - p_low.omega

Torque out causality (preferred):

phi = int(p.omega);
*p.F = c * phi + d*p.omega;*
*d = 2*b*sqrt(c*J);*

Angular velocity out causality:

p.omega = ddt(phi);
*phi = (p.T - d*p.omega)/c;*
*d = 2*b*sqrt(c*J);*

Interface - Stiffness

Ports	Description
p_high	Two ports of the spring (Rotation).
p_low	

Causality

preferred torque out	An angular velocity out causality results in a derivative constitutive equation.

Variables

phi	torsion of the spring [rad]
d	damping [Nms/rad]

Parameters

c	Rotational stiffness [Nm /rad]
b	Relative damping []
J	Moment of inertia [kgm^2]

Initial Values

phi_initial	The initial torsion of the spring [rad].

Description - Frequency

This model represents an ideal rotational spring with damper. The stiffness *(c)* is calculated on basis of a known resonance frequency *(f)*. The damping value *(d)* is calculated on the basis of the stiffness, relative damping *(b)* and reference inertia *(J)*. The inertia is only used to compute the damping (no actual mass is used in this component).

The element has a preferred torque out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred angular velocity out causality. The constitutive equations then contain a derivation. The port *p* of the spring model has separate high and low terminals. The equations are:

p.T = p_high.T = p_low.T
p.omega = p_high.omega - p_low.omega

Torque out causality (preferred):

phi = int(p.omega);
*p.T = c * phi + d*p.omega;*
c = J(2*pi*f)^2;*
*d = 2*b*sqrt(c*J);*

Angular velocity out causality:

```
p.omega = ddt(phi);
phi = (p.T - d*p.omega)/c;
c = J*(2*pi*f)^2;
d = 2*b*sqrt(c*J);
```

Interface - Frequency

Ports	Description
p_high p_low	Two ports of the spring (Rotation).
Causality	
preferred torque out	An angular velocity out causality results in a derivative constitutive equation.
Variables	
phi	torsion of the spring [rad]
c	rotational stiffness [Nm /rad]
d	damping [Nms/rad]
Parameters	
f	Resonance frequency [Hz]
b	Relative damping []
J	Moment of inertia [kgm^2]
Initial Values	
phi_initial	The initial torsion of the spring [rad].

Unbalance

Library

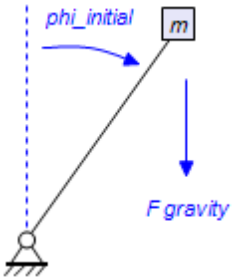
Iconic Diagrams\Mechanical\Rotation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model represents an unbalanced mass with an offset *r* to the axis of rotation and a starting angle *phi_initial*.



The mass will act as an inertia *J* with a gravity induced disturbance torque:

```
J = r^2 * m;
alpha = (p.T + m*r*g_n*sin(phi))/ J;
p.omega = int (alpha);
phi = int (p.omega, phi_initial);
```

The element has a preferred angular velocity out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred torque out causality. The constitutive equations then contain a derivation. The model has only one rotation port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. 20-sim will automatically create equations such that the resulting torque *p.T* is equal to the sum of the torques of all connected ports p1 .. pn and that the angular velocities of all connected ports is equal to *p.omega*.

```
p.T = sum(p1.T, p2.T, ....)
p.omega = p1.omega = p2.omega = ....
```

Interface

Ports	**Description**
p[any]	Any number of connections can be made (Rotation).
Causality	
preferred angular velocity out	An torque out causality results in a derivative constitutive equation.
Variables	
J	moment of inertia [kgm^2]
phi	angle [rad]
alpha	angular acceleration [rad/s^2]

Parameters

m	mass [kg]
r	distance of mass from center line of rotation [m]

Initial Values

p.omega_initial	The initial velocity of the inertia [rad/s].
phi_initial	The starting angle of the mass [rad].

ZeroTorque

Library

Iconic Diagrams\Mechanical\Rotation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

This model can be used to connect any open end of another model that is not connected to the fixed world. It generates a fixed torque of 0 N while the angular velocity is free:

```
p.omega = indifferent;
p.T = 0;
```

Interface

Ports	Description
p	rotation port

Causality

Fixed torque out

Gears

BeltPulley

Library

Iconic Diagrams\Mechanical\Rotation\Gears

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation/Translation).

Description

This models represents a belt and pulley. The connection to the pulley is through the rotation port *p_rot*. The connection to the belt is through the translation port *p_trans*. The model is ideal, i.e. there are no compliances or inertias. The causality of this model is always mixed: torque out & velocity out or angular velocity out & force out:

*p_rot.T = radius * p_trans.F*
*p_trans.v = radius * p_rot.omega*

or:

*p_trans.F = 1/radius * p_rot.T*
*p_rot.omega = 1/radius * p_trans.v*

Interface

Ports	**Description**
p_rot	Rotation port.
p_trans	Translation port.
Causality	
p_rot notequal	
p_trans	
Parameters	
radius	pulley radius [m]

Cam-Wizard

Library

Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation/Translation).

Description

This is a masked model which opens the Cam Wizard when edited. Depending on the selections entered, various cam motion profiles can be generated.

Interface

Ports	Description
p_in	Driving axis (Rotation)
p_out	Output port with resulting motion (Rotation or Translation)

Parameters

stroke	amplitude of resulting motion
start_angle	start angle motion
stop_angle	angle when the maximum is reached
return_angle	start angle of the return motion
end_angle	finish angle of the return motion

CamRod

Library

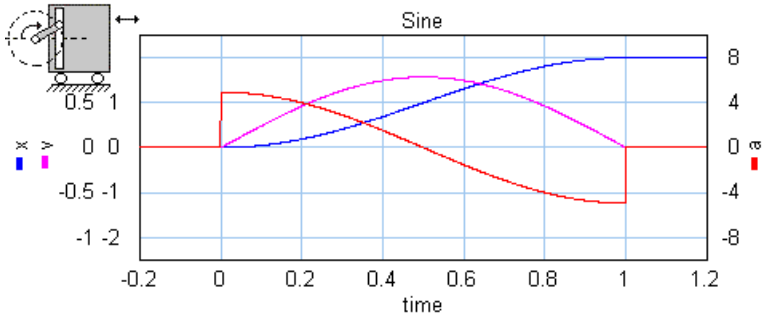
Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This models represents a cam and rod mechanism. If the input shaft is rotating with a constant speed, the output motion is a pure sinusoidal.



The mechanism is ideal, i.e., it does not have inertia, friction or geometrical limitations. It has one rotation port *(p_in)* and one translation port *(p_out)*. The causality of this model is always mixed: one port has a force out causality while the other has a velocity out causality:

*p_in.T = i * p_out.F*
*p_out.v = i * p_in.omega*

The transmission ratio (i) is the ratio of the velocities of both ports (in fact a sinusoidal function of the shaft angle).

Interface

Ports	Description
p_in	Driving axis (Rotation)
p_out	Output port with resulting motion (Translation)

Causality

fixed torque out
p_in
fixed velocity out
p_out

Parameters

stroke	Stroke of the translation port (is equal to half the length of the rod).

CrankRod

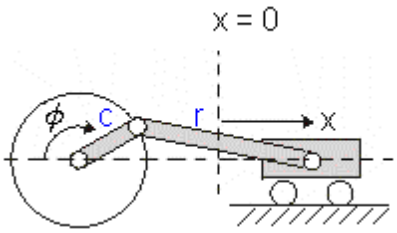
Library

Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description



This models represents a crank and rod mechanism. The mechanism is ideal, i.e., it does not have inertia, friction or geometrical limitations. It has one rotation port (*p_in*) and one translation port (*p_out*). The causality of this model is always mixed: one port has a force out causality while the other has a velocity out causality:

*p_in.T = i * p_out.F*
*p_out.v = i * p_in.omega*

The transmission ratio (i) is the ratio of the velocities of both ports. It is a function of the shaft angle, the crank length and the rod length.

Interface

Ports	Description
p_in	Driving axis (Rotation)
p_out	Output port with resulting motion (Translation)

Causality

fixed torque out
p_in
fixed velocity out
p_out

Parameters

crank_length	Crank length [m]
rod_length	Rod length [m]

DifferentialGear

Library

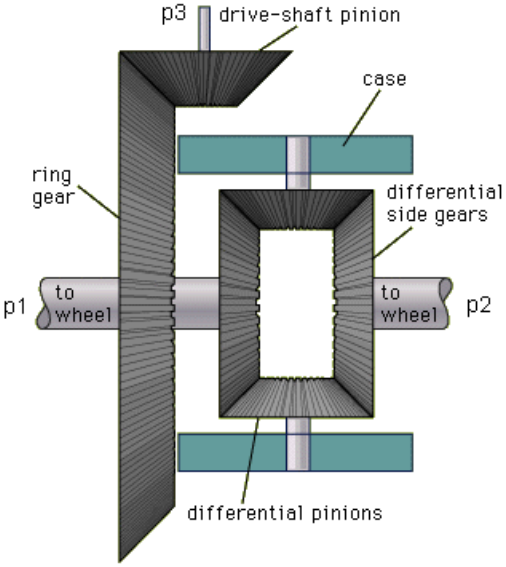
Iconic Diagrams\Mechanical\Rotation\Gears

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

A differential gear is well known for its use in automotive mechanics. It transfers power from the engine to the wheels, dividing the force equally between them but permitting them to follow paths of different lengths when turning a corner.



When p3 is the drive-shaft and p1 and p3 the shafts connected to the wheels, the constitutive equations for this model are:

```
p3.omega*2*i = p1.omega + p2.omega;
p1.T = p2.T = p3.T/(2*i);
```

with i the gear ratio. The gear ratio is equal to the diameter of the drive shaft pinion divided by the diameter of the ring gear, which makes it equivalent to the angular velocity of the drive shaft divided by the average angular velocity of the wheel shafts:

```
i = p3.omega / ((p1.omega + p2.omega)/2);
```

Interface

Ports	**Description**
p1	drive shaft port (Rotation)
p2,p3	driven shafts ports (Rotation)
Parameters	
i	gear ratio []

Differential

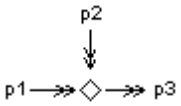
Library

Iconic Diagrams\Mechanical\Rotation\Gears

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

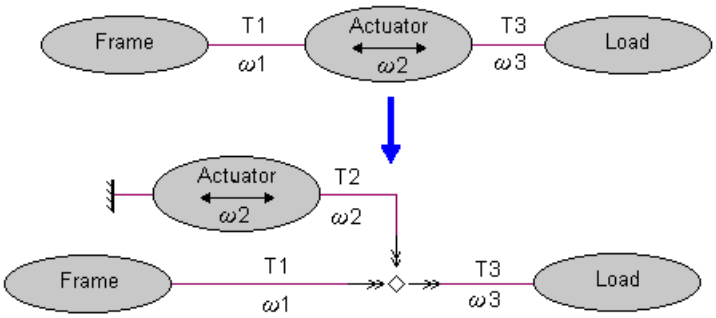
Description



This model is equivalent to the fork model of the Translation library. It represents a special type of node where the torques are equal and the angular velocities are added:

p3.omega = p1.omega + p2.omega;
p1.T = p2.T = p3.T;

This model can for example be used for actuators that generate a torque difference. With the differential model an equivalent model can be found with the actuator attached to the fixed world:



Interface

Ports	Description
p1,p2,p3	Rotation ports.

Gear

Library

Iconic Diagrams\Mechanical\Rotation\Gears

Implementations

Ideal

Lossy

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description - Ideal

This models represents any type of gearbox with two counter rotating shafts. The gear is ideal, i.e., it does not have inertia or friction. The gear has one fast moving shaft and one slow moving shaft. The gearbox has a reduction of i : 1 and thus a transmission ratio of 1/i.

The causality of this model is always mixed: one port has a torque out causality while the other has an angular velocity out causality:

*p_in.T = -1/i * p_out.T*
*p_out.omega = -1/i * p_in.omega*

or:

*p_out.T = -i * p_in.T*
*p_in.omega = -i * p_out.omega*

Interface - Ideal

Ports	**Description**
p_in	Input port
p_out	Output port

Causality

p_in not equal
p_out

Parameters

i	gearbox reduction, i > 1

Description - Efficiency

This models represents any type of gearbox with two counter rotating shafts. The gearbox has a reduction of i : 1 and thus a transmission ratio of 1/i.

*p_in.omega = -i * p_out.omega*

Unlike the model of an ideal gearbox, this model includes power loss. The power loss is represented by the efficiency, where the efficiency is defined as the output power divided by the input power:

eff = Pout / Pin

The efficiency is a value between zero and one and given by the gearbox manufacturer. If you do no know the value, a good guess is a power loss of 3% per stage (eff = 0.97). If your gearbox for example contains three stages, you can set the efficiency as:

*eff = 0.97*0.97*0.97 = 0.91*

Interface - Efficiency

Ports	Description
p_in	Input port
p_out	Output port

Causality

p_in not equal
p_out

Parameters

i	gearbox reduction [-], i > 1
eff	gearbox efficiency [-]

Description - Lossy

This models represents any type of gearbox with two counter rotating shafts. The gearbox has a reduction of i : 1 and thus a transmission ratio of 1/i.

*p_in.omega = -i * p_out.omega*

Unlike the model of an ideal gearbox, this model includes rotational inertia and power loss. The inertia is defined at the input axis. If the gearbox manufacturer gives the inertia at the output axis you can calculate the inertia at input axis as:

Jinput = Joutput/i^2

The power loss is represented by the efficiency, where the efficiency is defined as the output power divided by the input power:

eff = Pout / Pin

The efficiency is a value between zero and one and given by the gearbox manufacturer. If you do no know the value, a good guess is a power loss of 3% per stage (eff = 0.97). If your gearbox for example contains three stages, you can set the efficiency as:

*eff = 0.97*0.97*0.97 = 0.91*

Interface - Lossy

Ports	Description
p_in	Input port
p_out	Output port

Causality

preferred angular
velocity out p_in
preferred angular
velocity out p_out

Parameters

i	gearbox reduction [-], i > 1
J	moment of inertia [kgm^2]
eff	gearbox efficiency [-]

PlanetaryGear

Library

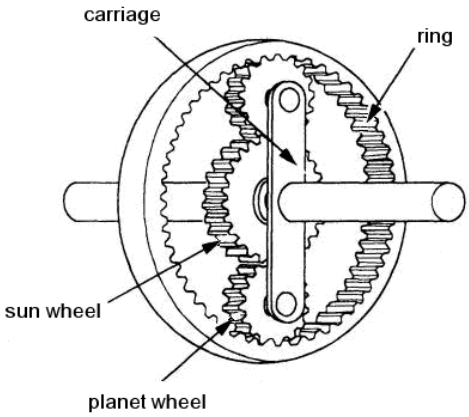
Iconic Diagrams\Mechanical\Rotation\Gears

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description

A planetary gear is well known for its use in automotive mechanics. It transfers power from a sun wheel to the planet wheels and the ring. In standard planetary gears, the planet wheels are connected by a carriage. I.e the gear has three rotation ports: the sun, the carriage and the ring.



The model that is used here is ideal. I.e. there is only power flow between the three axes. No internal dynamics or friction are incorporated. If the inertia cannot be neglected, you can easily add inertia models to the ports of the planetary gear model.

The number of teeth of the planet wheels are uniquely defined by the number of teeth of the sun wheel and the ring by:

zp = (zr - zs)/2;

Therefore the ratio between the sun teeth and the ring teeth:

z = zr / zs;

determine the transmission ratio of the gearbox and therefore the dynamic equations of the gearbox:

```
carriage.omega = sun.omega * (1/(1 + z)) + ring.omega * (z/(1 + z));
ring.T = carriage.T * (z/(1 + z));
sun.T = carriage.T * (1/(1 + z));
```

Interface

Ports	**Description**
sun	shaft attached to the sun wheel (Rotation)
carriage	shaft attached to the carriage (Rotation)
ring	shaft attached to the ring (Rotation)
Parameters	
z	number of ring teeth divided by the number of sun teeth []

RackPinionGear

Library

Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Implementations

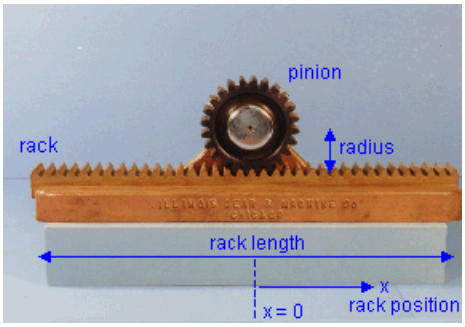
FixedPinion
FixedRack

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation/Translation).

Description - FixedPionion

This models represents a rack and pinion gear. The connection to the pinion gear is through the rotation port *p_rot*. The connection to the rack is through the translation port *p_trans*. The model is ideal, i.e. there is no compliance nor inertia nor backlash.



In this model the pinion bearing is connected to the fixed world and the rack is free to move. This in contrary to the model FixedRackPinionGear where the pinion bearing is free to move and the rack is connected to the fixed world.

The causality of this model is always mixed: torque out & velocity out or angular velocity out & force out:

*p_rot.T = radius * p_trans.F*
*p_trans.v = radius * p_rot.omega*

or:

*p_trans.F = 1/radius * p_rot.T*
*p_rot.omega = 1/radius * p_trans.v*

The rack position is determined by the internal variable x. For x = 0, the pinion gear is at the middle of the rack. When the pinion crosses the end of the rack, i.e.

abs(x) > rack_length/2

a warning is given, *"WARNING: rack length has been exceeded at the rack and pinion gear!"*, and the simulation is stopped.

Interface - FixedPionion

Ports	**Description**
p_rot	Rotation port.
p_trans	Translation port.
Causality	
p_rot notequal	
p_trans	
Parameters	
radius	pinion gear pitch radius [m]
rack_length	rack length [m]
Variables	

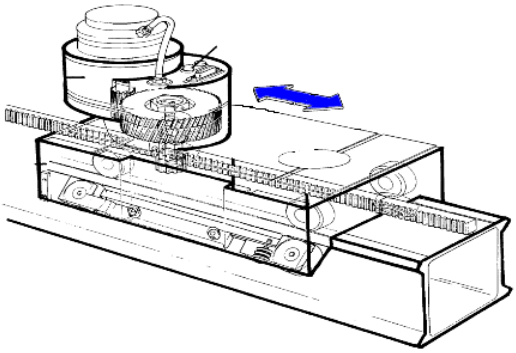
x	Internal variable which denotes the rack position, abs(x) < rack_length/2 else simulation halted.

Initial values

x_initial	Initial rack position, abs(x_initial) < rack_length/2

Description - FixedRack

This models represents a fixed rack and pinion gear. The connection to the pinion gear is through the rotation port *p_rot*. The connection to the rack is through the translation port *p_trans*. The model is ideal, i.e. there is no compliance nor inertia nor backlash.



In this model the pinion bearing is free to move and the rack is connected to the fixed world. This in contrary to the model RackPinionGear where the pinion bearing is connected to the fixed world and the rack is free to move.

The causality of this model is always mixed: torque out & velocity out or angular velocity out & force out:

*p_rot.T = radius * p_trans.F*
*p_trans.v = radius * p_rot.omega*

or:

*p_trans.F = 1/radius * p_rot.T*
*p_rot.omega = 1/radius * p_trans.v*

The rack position is determined by the internal variable x. For x = 0 the pinion gear is at the middle of the rack. When the pinion crosses the end of the rack, i.e.

abs(x) > rack_length/2

a warning is given, *"WARNING: rack length has been exceeded at the rack and pinion gear!"*, and the simulation is stopped.

Interface - FixedRack

Ports	Description
p_rot	Rotation port.
p_trans	Translation port.
Causality	
p_rot notequal p_trans	
Parameters	
radius	pinion gear pitch radius [m]
rack_length	rack length [m]
Variables	
x	Internal variable which denotes the rack position, abs(x) < rack_length/2 else simulation halted.
Initial values	
x_initial	Initial rack position, abs(x_initial) < rack_length/2

Spindle

Library

Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Introduction

This models represents a spindle and nut. It transfers an angular motion of the spindle into a translational motion of the nut. The model is ideal, i.e., it does not have inertia or friction. The causality of this model is always mixed: one port has a torque out causality while the other has an angular velocity out causality:

*p_spindle.T = i * p_nut.F*
*p_nut.v = i * p_spindle.omega*

or:

*p_nut.F = 1/i * p_spindle.T*
*p_spindle.omega = 1/i * p_nut.v*

The model has two implementations which calculate the transform ratio *i* out of different parameters.

Description - Pitch

In this implementation the transform ratio is calculated using the *pitch* (the advance of the nut during one revolution of the spindle):

*i = pitch / (2 * pi);*

Interface - Pitch

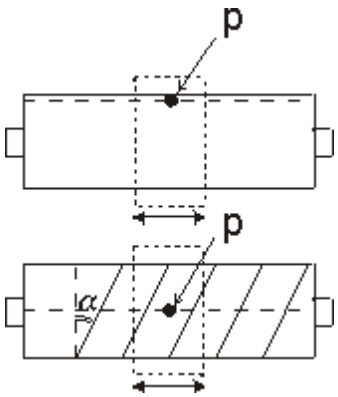
Ports	**Description**
p_spindle	Port at the spindle shaft (Rotation).
p_nut	Port at the wheel (Translation).
Causality	
p_spindle	
notequal p_nut	
Parameters	
pitch	translation of the nut during one revolution of the spindle [m]

Description - LeadAngle

This implementation calculates the transform ratio out of the lead angle *alpha* and the radius *r_spindle* of the spindle:

*i = tan(alpha)*r_spindle;*

The pitch angle is shown in the figure below. *r_spindle* is the effective radius of the spindle, i.e. the radius from the center of the spindle to the pitch point *p*.



Interface -LeadAngle

Ports	**Description**
p_spindle	Port at the spindle shaft (Rotation).
p_nut	Port at the wheel (Translation).

Causality

p_spindle

notequal p_nut

Parameters

r_spindle	effective radius of the spindle [m]
alpha	lead angle of the spindle [rad]

TimingBelt

Library

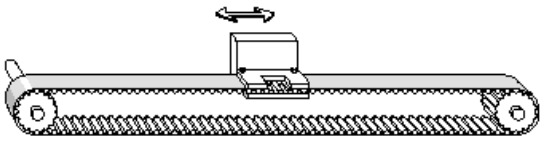
Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

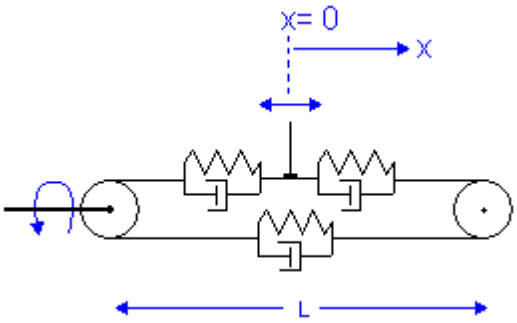
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation/Translation).

Introduction

This models represents a timing belt, used for linear positioning.



It has a rotating pulley which drives the the belt and clamp. The timing belt is modeled by a series of spring damper elements that convey the rotation of the pulley to a clamp translation. Because the output position is moving, stiffness and damping values are not constant.



The stiffness for a piece of belt can be expressed as:

$$k = E*A / l$$

with

E = Modulus of elasticity {N/m2}
A = Belt area {m2}
l = belt length {m}

If the belt is sufficiently pre-tensioned, the stiffness experienced at the clamp can be expressed as the combination of three individual belt parts:

$$k = E*A/(0.5*l+x) + 1/(\ 1 / E*A/(0.5*l - x)\ + 1 / E*A / l\ )$$

which can be rewritten to:

$$k = E*A*(\ 1/(0.5*l+x) + 1/(1.5*l - x)\ )$$

The stiffness approaches infinity as the clamp moves to the driven pulley (x = -l/2) and has a minimum value when the clamp moves to the other pulley (x = 0.5*l). The minimum stiffness is equal to:

$$k = 2*E*A / l$$

The belt position is determined by the internal variable *x*. For *x* = 0 the clamp is in the middle. When the position crosses the driven pulley, i.e.

$$x < - belt\_length/2$$

the simulation is stopped: *"Error: clamp position larger than belt end!".* When the position crosses the other pulley, i.e.

$$x > belt\_length/2$$

the simulation is also stopped, *"Error: clamp position smaller than belt start!".*

Description - Default

In this model the minimum stiffness is used, based on an output position at a length L of the driven pulley.

$$k = 2*E*A / l$$

Description - VariableStiffness

In this model a variable stiffness is used equal to:

$$k = E*A*(\ 1/(0.5*l+x) + 1/(1.5*l - x)\ )$$

Take care not to let the clamp get too close to the driven pulley, because the stiffness will then grow to infinity!

Interface

Ports	Description
p_rot	Rotation port.
p_trans	Translation port.

Causality

p_rot	notequal
p_trans	

Parameters

radius	pinion gear pitch radius [m]
d	damping N.s/m]
E	Modulus of elasticity [N/m2]
A	*A* = Belt area [m2]
l	belt length [m]

Variables

x	clamp position, abs(x) < belt length/2

Initial values

x_initial	Initial clamp position, abs(x_initial) < belt length/2

Transmission

Library

Iconic Diagrams\Mechanical\Rotation\Gears

Implementations

Ideal
Lossy

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description - Ideal

This models represents any type of gearbox with two shafts rotating in the same direction. The gear is ideal, i.e., it does not have inertia or friction. The gear has one fast moving shaft and one slow moving shaft. The gearbox has a reduction of i : 1 and thus a transmission ratio of 1/i.

The causality of this model is always mixed: one port has a torque out causality while the other has an angular velocity out causality:

*p_in.T = 1/i * p_out.T*
*p_out.omega = 1/i * p_in.omega*

or:

*p_out.T = i * p_in.T*
*p_in.omega = i * p_out.omega*

Interface - Ideal

Ports	Description
p_in	Input port
p_out	Output port

Causality

p_in not equal
p_out

Parameters

i	gearbox reduction, i > 1

Description - Efficiency

This models represents any type of gearbox with two shafts rotating in the same direction. The gearbox has a reduction of i : 1 and thus a transmission ratio of 1/i.

*p_in.omega = i * p_out.omega*

Unlike the model of an ideal gearbox, this model includes power loss. The power loss is represented by the efficiency, where the efficiency is defined as the output power divided by the input power:

eff = Pout / Pin

The efficiency is a value between zero and one and given by the gearbox manufacturer. If you do no know the value, a good guess is a power loss of 3% per stage (eff = 0.97). If your gearbox for example contains three stages, you can set the efficiency as:

*eff = 0.97*0.97*0.97 = 0.91*

Interface - Efficiency

Ports	Description
p_in	Input port
p_out	Output port

Causality

p_in not equal
p_out

Parameters

i	gearbox reduction [-], i > 1
eff	gearbox efficiency [-]

Description - Lossy

This models represents any type of gearbox with two shafts rotating in the same direction. The gearbox has a reduction of i : 1 and thus a transmission ratio of 1/i.

*p_in.omega = i * p_out.omega*

Unlike the model of an ideal gearbox, this model includes rotational inertia and power loss. The inertia is defined at the input axis. If the gearbox manufacturer gives the inertia at the output axis you can calculate the inertia at input axis as:

Jinput = Joutput/i^2

The power loss is represented by the efficiency, where the efficiency is defined as the output power divided by the input power:

eff = Pout / Pin

The efficiency is a value between zero and one and given by the gearbox manufacturer. If you do no know the value, a good guess is a power loss of 3% per stage (eff = 0.98). If your gearbox for example contains three stages, you can set the efficiency as:

*eff = 0.97*0.97*0.97 = 0.91*

Interface - Lossy

Ports	**Description**
p_in	Input port
p_out	Output port

Causality

preferred angular velocity out p_in

preferred angular velocity out p_out

Parameters

i	gearbox reduction [-], i > 1
J	moment of inertia [kgm^2]
eff	gearbox efficiency [-]

UniversalCoupling

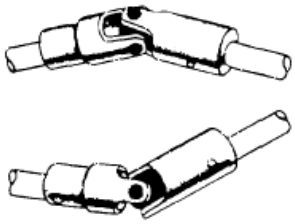
Library

Iconic Diagrams\Mechanical\Rotation\Gears

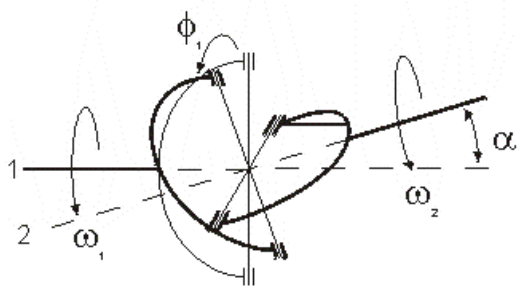
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description



Universal Joint couplings are used to couple axes which are not aligned. The can be used in single or double set-up. Consider the single Universal Joint coupling, shown in the picture below.



Due to the nature of the coupling, the output angular velocity w1 will show a sinusoidal ripple, compared to the input angular velocity w2:

$$\omega_2 = \frac{\cos(\alpha)}{1-\cos^2(\varphi_1)\sin^2(\alpha)} \cdot \omega_1$$

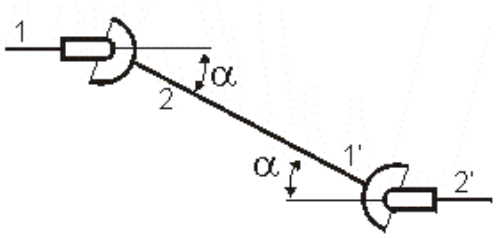
with α the angle between the two axes and j1 the initial rotation of the input axis. The initial rotation j1 is defined with respect to a plane that is spanned by the axes (1 and 2).

When two Universal Joint couplings are combined, the angular velocity of the output axis will be equal to the angular velocity of the input axis (e.g. no ripple) when two conditions are met. When the output axis of the first coupling is connected to the input axis of the second coupling, these conditions are:

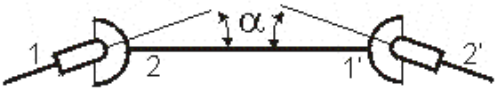
1. For both couplings the absolute value of the angle α should be equal.

2. The difference in initial rotation of both couplings (j1 - j2) should be equal to +p/2 [rad] or -p/2 [rad].

Two alignments are shown in the figure below.



α = 0.35, α' = -0.35 , j1 = 0, j1' = p/2 [rad]



α = 0.35, α' = 0.35 , j1 = 0, j1' = p/2 [rad]

Interface

Ports	**Description**
p1, p2	The input and output axes (Rotation)

Causality

Input

p1 not equal p2

Parameters

alpha	The angle between the two axes (α).

Initial Values

phi1_initial	The initial rotation of the input axis (j1).

Wormgear

Library

Iconic Diagrams\Mechanical\Rotation\Gears

Implementations

Ideal
Backlash

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Description - Ideal

This models represents a worm gear with spindle (also referred to as worm) and wheel (also referred to as gear). The gear is ideal, i.e., it does not have inertia or friction. The causality of this model is always mixed: one port has a torque out causality while the other has an angular velocity out causality:

*p_spindle.T = i * p_wheel.T*
*p_wheel.omega = i * p_spindle.omega*

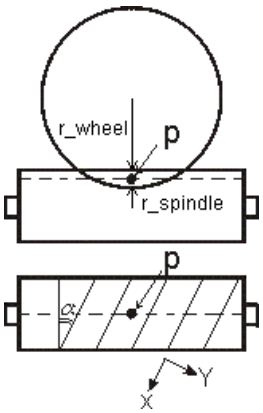
or:

*p_wheel.T = 1/i * p_spindle.T*
*p_spindle.omega = 1/i * p_wheel.omega*

with

*i = tan(alpha)*r_spindle/r_wheel;*

Alpha is the pitch angle as shown in the figure below. *r_wheel* and *r_spindle* are the effective radii of the wheel and spindle, i.e. the radii from the center of the wheel and spindle to the pitch point *p*.



Interface - Ideal

Ports	**Description**
p_spindle	Port at the spindle shaft (Rotation).
p_wheel	Port at the wheel (Rotation).

Causality

p_wheel notequal
p_spindle

Parameters

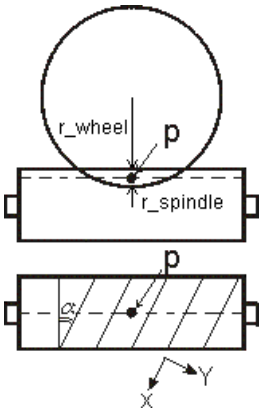
r_spindle	pitch radius spindle [m]
r_wheel	pitch radius wheel [m]
alpha	lead angle of the spindle [rad]

Description - Backlash

This models represents a worm gear with spindle (also referred to as worm) and wheel (also referred to as gear). Between spindle and wheel, backlash and friction is modelled. The causality of this model is always fixed: both ports have a torque out causality.

Analysis

The spindle and wheel have one ore more pair of teeth contacting each other. The effective point of contact is called the pitch point ***P***. At the pitch point a frame is defined, with an x- and y-direction. During rotation of the wheel and spindle, the teeth will experience a normal force *Fn* in the x-direction and a friction force *Ff* in the y-direction.



Between each pair of teeth, a clearance exists (y-direction), causing backlash. This effect is time-dependent, because of the tooth pairs coming in and out of contact. During normal operation this is a high frequency effect, which is filtered out by the damping of the construction. We therefore only use an **average backlash** at the pitch point. Using the same reasoning as backlash, an **average friction** is used at the pitch point.

To define both backlash and friction **a frame is used at the pitch point** (see picture above) to indicate velocities. The velocity in x-direction is the sliding velocity between the teeth and used to calculate the friction force. The position in y-direction is the position in the clearance and used to calculate the backlash force.

Backlash

The backlash is described by the standard formula that is also used in the Backlash.emx model. The normal force is modelled by a spring-damper system. Inside the clearance a low damping and stiffness is used (*k1 and d1*), while a high stiffness and damping (*k2 and d2*) is used at both ends of the clearance. This yields a normal force *Fn* of:

$$p.F = 0.5 \cdot k_2 \cdot \left(x - \sqrt{\left(-x - \frac{s}{2}\right) + (ep \cdot s)^2} + x + \sqrt{\left(x - \frac{s}{2}\right) + (ep \cdot s)^2} \right) + k_1 \cdot x + d_{1or\,2} \cdot p.v$$

Friction

Friction is described as static plus coulomb plus viscous plus Stribeck friction:

```
p.F = Fn *
(( mu_c + (mu_st*abs(tanh( slope*p.v )) - mu_c)
* exp( -((p.v / v_st)^2 )) ) * sign(p.v)
+ mu_v * p.v);
```

with:

Fn: the normal force (given by the backlash formula)
mu_s: the static friction coefficient
mu_v: the viscous friction coefficient
mu_c: the coulomb friction coefficient
slope: the steepness of the coulomb and static friction curve.
v_st: the characteristic Stribeck velocity.

Self-Locking

A worm gear is said to be self-locking, or irreversible when the wheel cannot drive the spindle. This condition is obtained, if the lead angle of the worm is small and the friction force between the teeth is high enough. Then the friction force becomes larger than the driving force on the teeth. Suppose we only consider static and Coulomb friction and both friction coefficients are equal. Then self locking is obtained when the following condition holds:

mu_s > tan(alpha)
mu_c > tan(alpha)

Interface - Backlash

Ports	**Description**
p_spindle	Port at the spindle shaft (Rotation).
p_wheel	Port at the wheel (Rotation).
Causality	
fixed torque out p_spindle	
Parameters	
r_spindle	pitch radius spindle [m]
r_wheel	pitch radius wheel [m]
alpha	lead angle of the spindle [rad]
s	Interval of the play [m]
k1	Stiffness in the play [N/m]
k2	Stiffness outside the play [N/m]
d1	Damping inside the play [Ns/m]
d2	Damping outside the play [Ns/m]
ep	Relative round off (1e-6 -> sharp edges, 1e-2 -> smoother)
mu_s	Static friction coefficient []
mu_v	Viscous friction coefficient [s/m]
mu_cv	Coulomb friction coefficient []
slope	Steepness of Coulomb friction curve [s/m]
v_st	Characteristic Stribeck velocity [m/s]
Interesting Variables	

Fn	Average normal force between teeth [N]
Ff	Average friction force between teeth [N]
v_x	Velocity in x-direction at the pitch frame [m/s] (used in friction
v_y	formula)
x_x	Velocity in y-direction at the pitch frame [m/s] (used in backlash
x_y	formula)
	Position in x-direction at the pitch frame [m/s] (used in friction formula)
	Position in y-direction at the pitch frame [m/s] (used in backlash formula)

Sensors

AccelerationSensor-Absolute

Library

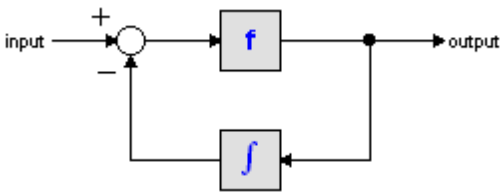
Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model describes an acceleration sensor which derives an angular acceleration output out of a port angular velocity by differentiation. Differentiation is performed by a state variable filter:



The S-domain function of this filter is equal to:

$$output = \frac{s}{\frac{s}{f}+1} \cdot input$$

where *f* is the cut-off frequency. For very high values of *f*, the output becomes the pure derivative of the input. High values of *f,* however, increase simulations times. A good trade-off is a starting value of 1e5.

The equations of this model are:

alpha = d(p.omega)/dt;
p.T = indifferent;

Interface

Ports	**Description**
p	Rotation port p.
Causality	
fixed force out	
Output	
alpha	Absolute angular acceleration [rad/s^2]
Parameters	
f	Cut-off frequency of the differentiation [Hz].
alpha_initial	Initial angular acceleration [m/s].

AccelerationSensor-Relative

Library

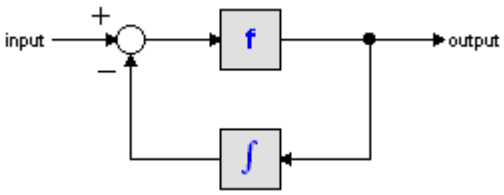
Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model describes an acceleration sensor which derives an angular acceleration output out of a angular velocity difference (between high and low terminals) by differentiation. Differentiation is performed by a state variable filter:



The S-domain function of this filter is equal to:

$$output = \frac{s}{\frac{s}{f}+1} \cdot input$$

where *f* is the cut-off frequency. For very high values of *f*, the output becomes the pure derivative of the input. High values of *f*, however, increase simulations times. A good trade-off is a starting value of 1e5.

The equations of this model are:

alpha = d(p_high.omega - p_low.omega)/dt;
p_low.T = p_high.T = indifferent;

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Mechanical port p.
Causality	
fixed force out	
Output	
alpha	Angular acceleration (measured as the difference between both terminals) [rad/s^2]
Parameters	
f	Cut-off frequency of the differentiation [Hz].
alpha_initial	Initial angular acceleration [m/s].

Encoder

Library

Iconic Diagrams\Mechanical\Rotation\Sensors

Implementations

Ideal
Absolute
Incremental

Use

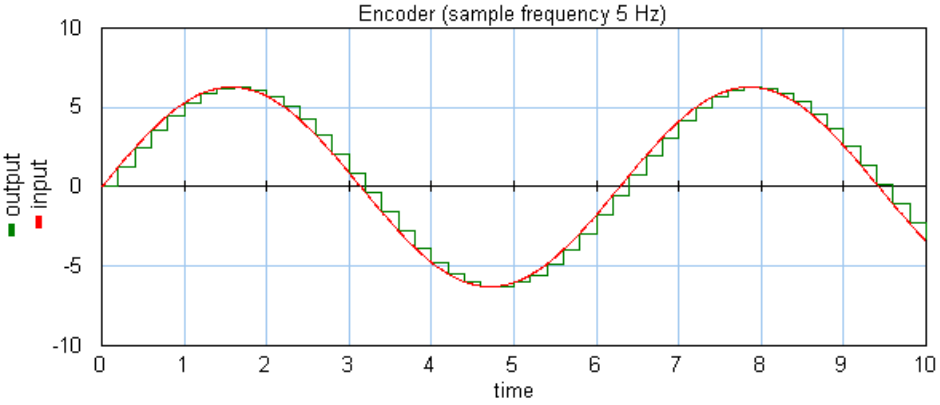
Domains: Continuous/Discrete. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation) / Block Diagrams.

Description - Ideal

This is a model of an optical encoder combined with sampling. The Encoder has no internal friction. The output signal is equal to the sampled shaft shaft angle (in radians).

Example

The shaft angle is equal to a sine with amplitude 2*pi (one revolution forward and one back), the sampling rate is 5 Hz.



Interface - Ideal

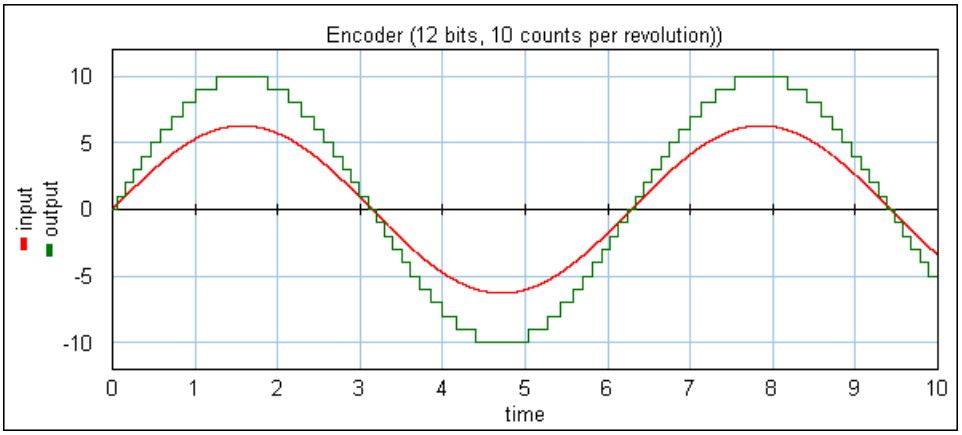
Ports	**Description**
p	Rotation port
Causality	
fixed torque out	(The output torque is zero)
Outputs	**Description**
output	sampled shaft angle (rad)

Description - Absolute

This is a model of an optical encoder combined with counting logic and analog to digital conversion. The encoder has no internal friction. Counting is absolute. The shaft angle is measured with an accuracy of a given number of counts per revolution.

Example

An encoder has an accuracy of 10 counts per revolution. This means an input signal with value 6.283185307 gives an output of 10. This is illustrated in the figure below. The input is a sine with amplitude 2*pi (one revolution forward and one back).



Overflow

This model has no overflow.

Interface - Absolute

Ports	**Description**
p	Rotation port
Causality	
fixed torque out	(The output torque is zero)
Outputs	**Description**
output	number of counts
Parameters	
counts	counts per revolution.

Description - Incremental

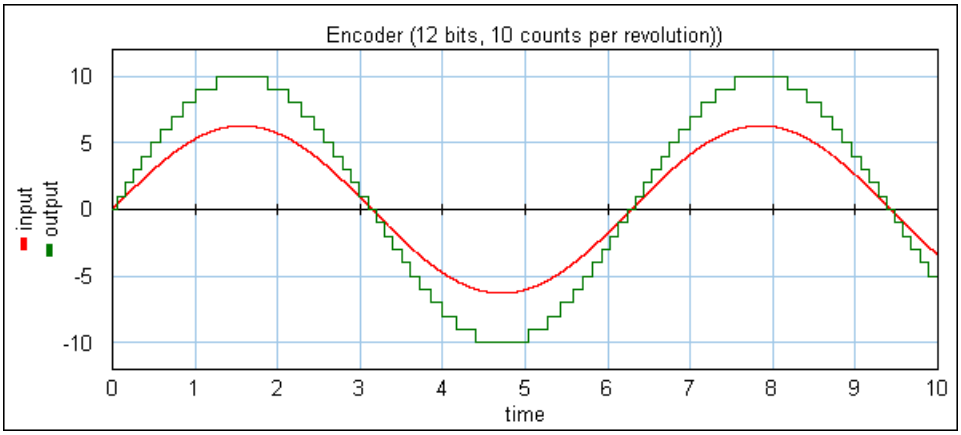
This is a model of an optical encoder combined with counting logic and analog to digital conversion. The encoder has no internal friction. Counting is incremental. The shaft angle is measured with an accuracy of a given number of counts per revolution and a window defined by the number of bits.

Example

An encoder has an accuracy of 2000 counts per revolution. This means an input signal with value 6.283185307 gives an output of 2000.

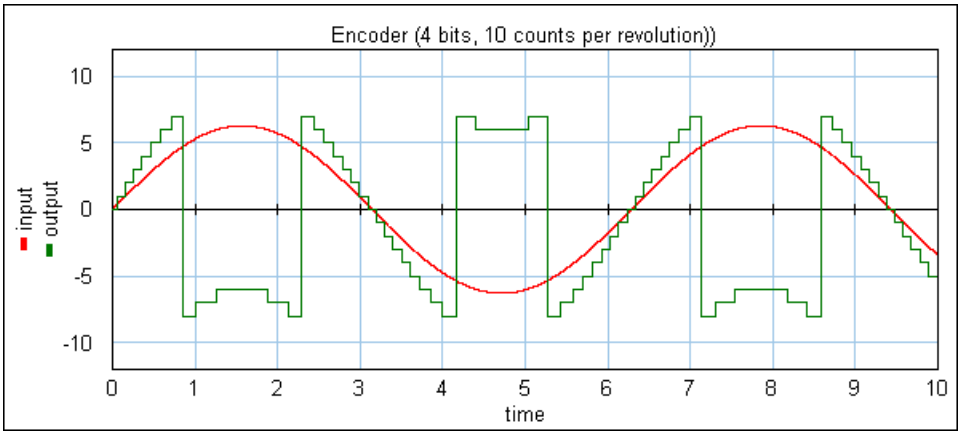
Example

An encoder has an accuracy of 10 counts per revolution and a 5 bits counter. This means an input signal with value 6.283185307 gives an output of 10 and the output is clipping between +15 and -16. This is illustrated in the figure below. The input is a sine with amplitude 2*pi (one revolution forward and one back).



Encoder output without overflow.

The next figure shows the same settings but only 4 bits are used. This means the output is clipping between +7 and -8.



Encoder output with overflow.

Overflow

The maximum and minimum number of counts are equal to 2(bits-1) -1 and -2(bits-1). If the number of counts passes 2(bits-1) -1, counting continues at -2(bits-1) and vice versa.

Interface - Incremental

Ports	Description
p	Rotation port

Causality	
fixed torque out	(The output torque is zero)
Outputs	**Description**
output	number of counts
Parameters	
counts	counts per revolution.
bits	maximum counting interval is between $2^{(bits-1)}$ -1 and $-2^{(bits-1)}$.

Limitations

The output of this model is a discrete signal. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

PositionSensor-Absolute

Library

Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (, Block Diagrams.), Block Diagrams.

Description

This model translates a angular position to an output signal. It has a force out causality. The equations are:

```
p.T = 0;
phi = int(p.omega);
```

Interface

Ports	**Description**
p_high, p_low	Rotation port p.
Causality	
fixed force out	
Output	
phi	Absolute angular position [rad]
Initial Values	
phi_initial	Initial angular position [rad]

PositionSensor-Relative

Library

Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model translates a angular position difference to an output signal. It has a force out causality. The port *p* of the model has separate high and low terminals. The equations are:

```
p.T = p_high.T = p_low.T;
p.omega = p_high.omega - p_low.omega;
p.T = 0;
phi = int(p.omega);
```

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Rotation port p.
Causality	
fixed force out	
Output	
phi	Relative angular position [rad]
Initial Values	
phi_initial	Initial angular position [rad]

Potentiometer

Library

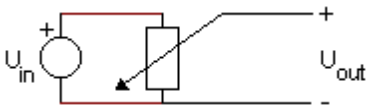
Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation / Electric).

Description

Potentiometers have a variable resistance according to a certain rotation. By establishing a voltage (see picture below) across the resistor it is possible to get a proportional relation between the angle and the output voltage.



with

*Uout = Umax * angle / (turns*2*pi)*

Here Umax is the maximum voltage and turns is the maximum number of turns.

Interface

Ports	**Description**
p1	Rotation port.
p2	Electric port.

Causality

fixed	torque	out	(The output torque is zero)
p1			The output is equal to Uout.
fixed	voltage	out	
p2			

Parameters

Umax	Maximum output voltage [V]
turns	maximum allowed number of turns []

PowerSensor

Library

Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This is an ideal sensor (no dissipation or other effects) that yields the power that flows through the model as output signal. The equations are:

```
p_high.T = p_low.T
p_high.omega = p_low.omega
P = p_high.T * p_high.omega;
```

Interface

Ports	**Description**

p_high, p_low	Both rotation ports.
Causality	
p_high not equal p_low	
Output	
P	Power [w].

Tachometer

Library

Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous/Discrete. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation) / Block Diagrams.

Description

A tachometer is a transducer for measuring the rotation speed. It is normally mounted at the shaft of a motor and gives a voltage signal proportional to the rotation speed:

$$V = Kv \cdot v$$

with Kv the tachometer gradient [V / rad/s] and v the shaft velocity [rad/s]. This equation represents the ideal case. Due to the slots and commutator segments of the tachometer, a noise ripple will exists:

$$Vn = \frac{Kn}{100} \cdot Kv \cdot v \cdot \cos\left(N \cdot v \cdot t\right)$$

with N the ripple frequency [cycles per turn] and Kn the peak to peak noise Ripple [in % of the output voltage]. To convert the Voltage Gradient parameter from RPM units [V / RPM] to radian units [V / rad/s], use the conversion formula:

$$Kv_{rad/s} = Kv_{RPM} \cdot \frac{2 \cdot \pi}{60}$$

Interface

Ports	**Description**
p	Rotation port.
Causality	
fixed torque out	(The output torque is zero)
Outputs	**Description**
output	Output voltage [V]

Parameters

Kv	Voltage Gradient [V / rad/s].
Kn	Noise Ripple , relative to output voltage [%].
N	Ripple frequency in cycles per turn [].

TorqueSensor

Library

Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model translates an applied torque to an output signal. It has an angular velocity out causality. The port *p* of the model has separate high and low terminals. The equations are:

```
p.T = p_high.T = p_low.T;
p.omega = p_high.omega - p_low.omega;
p.omega = 0;
T = p.T;
```

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Rotation port p.

Causality

fixed angular velocity out

Output

T	Applied torque [N/m]

VelocitySensor-Absolute

Library

Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model translates an angular velocity to an output signal. It has a force out causality. The equations are:

```
p.T = 0;
omega = p.omega;
```

Interface

Ports	Description
p	Mechanical port p.
Causality	
fixed force out	
Output	
omega	Absolute angular velocity [rad/s]

VelocitySensor-Relative

Library

Iconic Diagrams\Mechanical\Rotation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation), Block Diagrams.

Description

This model translates a velocity difference to an output signal. It has a force out causality. The port p of the model has separate high and low terminals. The equations are:

```
p.T = p_high.T = p_low.T;
p.omega = p_high.omega - p_low.omega;
p.T = 0;
omega = p.omega;
```

Interface

Ports	Description
p_high, p_low	Both terminals of the Rotation port p.
Causality	
fixed force out	
Output	
omega	Relative angular velocity [rad/s]

Translation

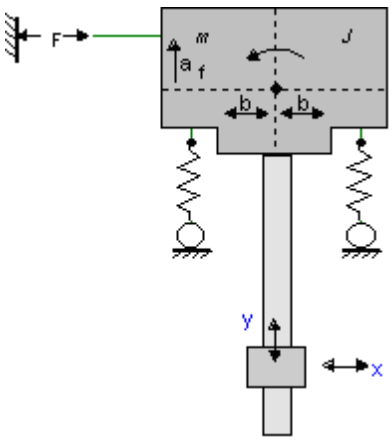
2DSmallRotation

2D Library (Small Rotations)

Introduction

The 2DSmallRotation library contains models that describe 2-D planar motion and are valid for small rotations. Due to these small rotations, the dynamic equations that describe the various models are simple and can be linearized symbolically. The library is therefore suited for modeling mechanical systems that experience small rotations and must be thoroughly analyzed in the frequency domain.

Consider the guidance system below. Due to the limited stiffness of the guidance wheels, the central body will experience a small rotation that results in a movement in the x-direction at the end-effector. To properly model the end-effector movement as a function of the driven force, this rotation has to be incorporated. To inspect the resulting resonance frequencies in a bode plot, the model must be linearized. If linearization is performed symbolically, the effect of parameter changes (*af, b, J*) can be inspected directly in the bode plot, which is very useful when the system is still in the design phase.



To perform symbolic linearization, the example system should be modeled using the models of the 2DSmallAngles library.

2D

When we speak of **2D motion** we often mean all planar motion. However, to describe this motion, **three degrees of freedom** (displacement x and y and rotation q) are used. In 20-sim we keep to the common naming standard of 2D for planar motion (3 degrees of freedom) and 3D for spatial motion (6 degrees of freedom). In 20-sim the three degrees of freedom for planar motion are combined in a vector notation:

d.o.f.	identity	forces	velocities
x	position	P.F[1] [N]	P.v[1] [m/s]

y	position	P.F[2] [N]	P.v[2] [m/s]
q	angle	P.F[3] [Nm]	P.v[3] [rad/s]

Note

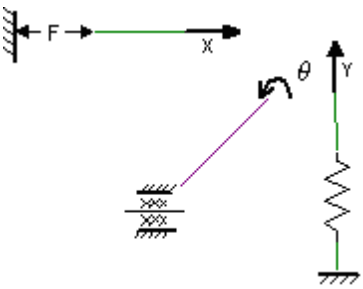
It is not possible in 20-sim to use vector elements with mixed units. Therefore element number 3 will be displayed with units [m/s] and [N] although it really is [rad/s] and [Nm]!

Ports

1D

The models in the 2DSmallRotation library have two types of connection ports: 1D and 2D. The 1D ports can represent movement and force (x- or y-direction) or an angular displacement and torque.

The models of the standard translation and rotation libraries can be used connect to these 1D ports as is shown in the picture below.



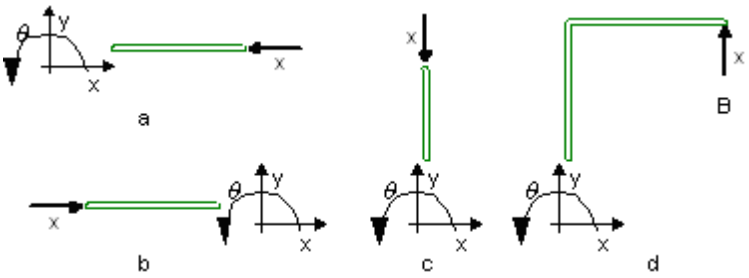
2D

The 2D connection port is a vector that represents three degrees of freedom. The vector notation that is used in 20-sim is:

d.o.f.	identity	forces	velocities
x	position	P.F[1] [N]	P.v[1] [m/s]
y	position	P.F[2] [N]	P.v[2] [m/s]
q	angle	P.F[3] [Nm]	P.v[3] [rad/s]

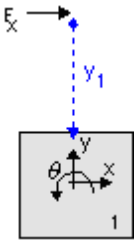
In 20-sim it is not possible to use vector ports with mixed units. This means that element number 3 will be always be displayed with the units [m/s] and [N] although it should be interpreted as [rad/s] and [Nm].

To connect 2D ports, only other models with 2D ports can be used. A double line indicates this vector connection.



Small Rotations

Consider the body below:



Given a force *Fx* in *x*-direction acting upon the body at an offset *y1* from the center, the resulting force at the center of the body is:

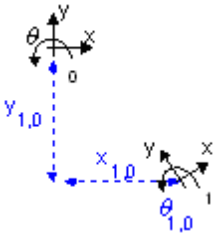
*F1x = Fx * cos(q)*
*F1y = Fx * sin(q)*
*F1q = Fx * -y1*

For small angles this can be simplified to:

F1x = Fx
F1y = 0
*F1q = Fx * -y1*

A force acting upon a body at a certain offset is rewritten to a force at the center of the body. This is common in multibody dynamics and called a transformation. A force acting at a certain point can be transformed to see its effect at another point.

Consider two frames which have an arbitrary offset as shown in the picture below. We want to know how a force in frame 0 is transformed to a force in frame 1. We assume that frame 1 has an offset of x1 and y1 with respect to frame 0 and a rotation q1 with respect to frame 0. In order to make clear that the offsets are given with respect to frame 0 we will write the with extra indices (*x1,0* , *y1,0* and q1,*0*).



Given an arbitrary force *F1,0* at frame 0 the resulting force *F1,1* at frame 1 is equal to:

$$F_{1,1} = R_0^1 \cdot F_{1,0} \quad \text{with} \quad R_0^1 = \begin{bmatrix} \cos(\theta_{1,0}) & \sin(\theta_{1,0}) & 0 \\ -\sin(\theta_{1,0}) & \cos(\theta_{1,0}) & 0 \\ -y_{1,0} & x_{1,0} & 1 \end{bmatrix}$$

For small angles this can be simplified to:

$$F_{1,1} = R_{01} \cdot F_{1,0} \quad \text{with} \quad R_{01} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -y_{1,0} & x_{1,0} & 1 \end{bmatrix}$$

The accuracy of this simplification depends on the angle q. For an angle of 0.5 degrees (0.009 rad) this results in an almost 1% error in the calculation of *F1,1*. Considering the cumulative effects of this error when multiple models are used, an angle of maximum 0.05 degrees (<1 mrad) is an upper bound for the models of the 2DSmallRotation library.

The simplification of the equations has two advantages. First because of the small rotations there is no frame rotation any more. Therefore all frames have the same direction (there is only an offset in x-direction and y-direction). Interpreting results is therefore much easier. But the biggest advantage is the simplification of equations which allows us to linearize models symbolically and thus perform tasks in the frequency like parameter sweeps!

Connections

Degrees of Freedom

As described in the previous topic, the models in 2DSmallRotation library have two types of connection ports: 1D and 2D. The 1D port can represent movement in the x-direction, the y-direction or an angular displacement q.

The 2D port is a vector that contains all three degrees of freedom. The vector notation that is used in 20-sim is:

d.o.f.	identity	forces	velocities
x	position	P.F[1] [N]	P.v[1] [m/s]
y	position	P.F[2] [N]	P.v[2] [m/s]
q	angle	P.F[3] [Nm]	P.v[3] [rad/s]

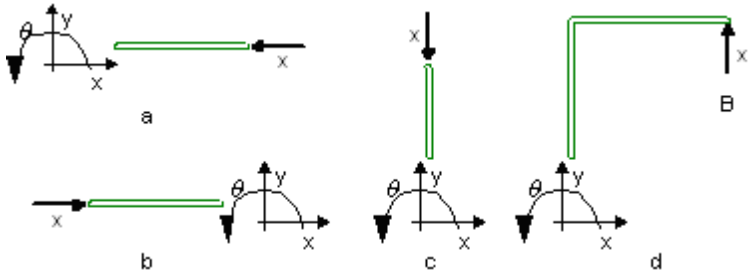
The vector element number (1, 2 or 3) denotes the identity of the degree of freedom (x, y and q). To create meaningful models, we have to define the positive direction of each degree of freedom:

d.o.f.	positive direction
x	from left to right
y	from bottom to top
q	counter clockwise

This definition is arbitrary. We could as well define it any other way. In the 2DSmallRotation library we have chosen the definition above. It will help us define new models and interpret simulation results.

Orientation Independent

The connections between models only indicate how variables (forces and velocities) are transferred from one model to the other and vice versa. The orientation of the models in the Graph Editor (how they are drawn) is therefore not important. **Flipping**, **rotating** or **moving** a model **does not change the model**! All four representations in the figure below are the same although their individual models are oriented differently.



The figure above shows a point model in x-direction connected to a 2D body. The point indicates a force in *x*-direction that is applied to the body. Since we defined that the positive *x*-direction is from the left to the right, intuitively figure b leads to the correct interpretation.

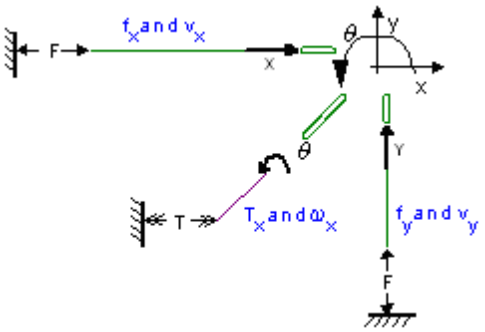
This indicates a problem. Although all models describe the same behavior, we would intuitively interpret them differently. It is therefore useful to impose some rules to prevent misinterpretation of a model by other users.

Drawing Rules

1. Preferably leave the orientation of the models of the 2DSmallRotation library as they pop up on the screen.
2. If necessary, only flip or rotate models in a direction that does not lead to misinterpretations.
3. If possible, draw 1D connections in the following direction:

d.o.f.	**positive direction**
x	from left to right
y	from bottom to top
q	from left bottom to right top

An example is shown in the figure below:



TwoDBody

TwoDBody

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Iconic Diagrams (2D Planar).

Description

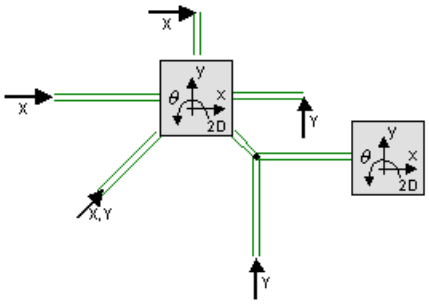
The central model in the 2DSmallRotation library is the TwoDBody model. It describes a body with three degrees of freedom: x, y and q. The TwoDBody is a point mass with rotational inertia. The constitutive equations are therefore very simple:

P.v = I . int(P.F);*

With

I = [1/M; 1/M; 1/J];

Any number of connections can be made to the TwoDBody model. All forces acting upon the body are added.



Because any number of connections can be made, successive ports are named *P1*, *P2*, *P3* etc. 20-sim will automatically create equations such that the resulting force *P.F* is equal to the sum of the forces of all connected ports *P1* . *Pn*. The velocities of all connected ports are equal to *P.v.* The model has a preferred velocity out causality. The corresponding constitutive equations then contain an integration. The model can also have the non-preferred force out causality. The constitutive equations then contain a differentiation.

P.F = sum(P1.F, P2.F, ...)
P.v = P1.v = P2.v = ...

velocity out causality (preferred):

I = [1/M;1/M;1/J]
*P.v = I*int(P.F);*

force out causality:

I = [1/M;1/M;1/J]
*P.F = I-1*ddt(P.v);*

Interface

Ports	Description

P[3]	Port with three degrees of freedom (x, y, q). Any number of connections can be made.

Causality

preferred velocity

out

Parameters

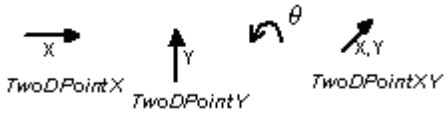
M	Mass [kg].
J	Moment of inertia [kgm^2/rad]

Note

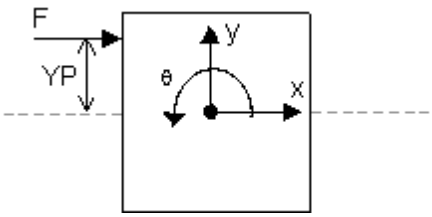
- A TwoDBody model can be connected to any other element of the 2D-library except other 2D-bodies. Two bodies can always be simplified to one TwoDBody model with the combined mass and inertia of both single bodies.
- TwoDBody models can be connected to 1D models using the TwoDPoint models as intermediate. If a TwoDPoint is connected to a body, it constrains the movement of the body in one direct

TwoDPoint Models

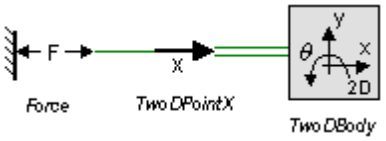
TwoDPoint models are used to connect 1D models to TwoDBodies. They indicate a 1D force or torque that is applied in a certain direction, onto a body.



A point model forms the connection between the single degree of freedom part of a system and the center of mass of a body. Consider the example below, where force F in x-direction (single degree of freedom) is pushing upon a body (three degrees of freedom) at an offset *YP* from the body center of mass.



To create such a construct, the TwoDPoint-X model is used:

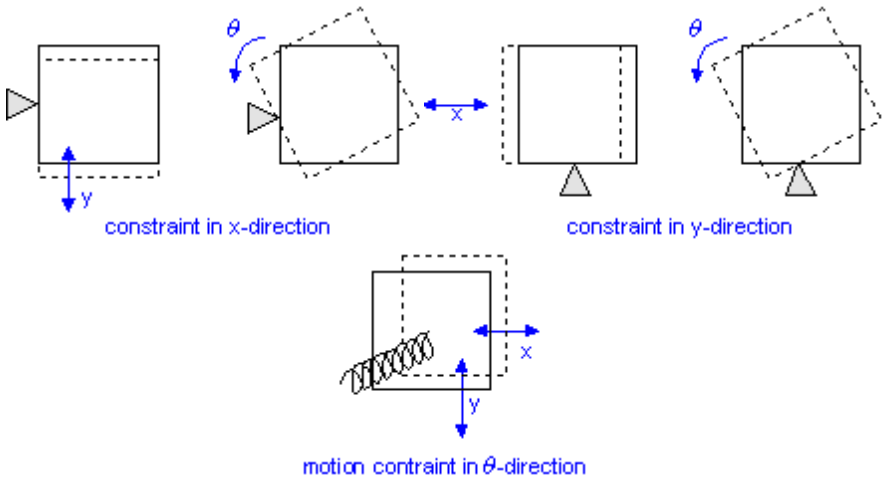


The TwoDPoint-X model describes the translation of a 1D force (x-direction) to a 2D-force (x-, y- and q-direction) and vice versa the translation of a 2D-velocity (x-, y- and q-direction) to a 1D velocity (x-direction).

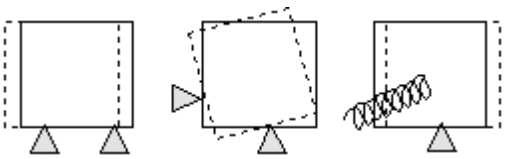
There are four point models. Three point models transform from x-direction, y-direction and q-direction to 2D. One point models transforms from direction with arbitrary orientation to 2D.

Fixation of bodies

If a TwoDBody model is used in combination with TwoDPoint models, at least three non-coinciding, unequal (i.e. not having the same offsets) TwoDPoints must be connected to prevent the body from free motion. This is obvious when we interpret the point models physically. Point models can be interpreted as constraints on motion in one direction. The motion in the other two directions is free.



To show that two points are not sufficient to constrain all motions look at the figure below.



TwoDPoint

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Implementations

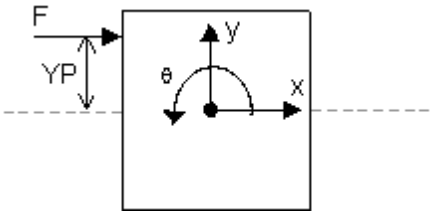
X
Y
theta
XY

Use

Domains: Continuous. **Size**: 1-D/2-D. **Kind**: Iconic Diagrams (1D Translation, 2D Planar).

Description - X

TwoDPoint models are used to connect 1D models to TwoDBodies. They indicate a 1D force or torque that is applied in a certain direction, onto a body. The TwoDPoint-X model forms the connection between the single degree of freedom in x-direction and the center of mass of a body. Consider the example below, where force F in x-direction (single degree of freedom) is pushing upon a body (three degrees of freedom) at an offset *YP* from the body center of mass.



To create such a construct the TwoDPoint-X model is used:

The TwoDPoint-X model describes the transformation of force in x-direction to a force in three degrees of freedom. The single degree of freedom port *p_in* describes the connection in x-direction and the three three degree of freedom port *P_out* describes connection with the 2D-body (x, y and q).

```
P_out.F[1] = p_in.F;                    // x-direction
P_out.F[2] = 0;                         // y-direction
P_out.F[3] = -p_in.F * YP;              // rotation

p_in.v = P_out.v[1] - YP *              // x-direction
P_out.v[3];
```

Interface - X

Ports	Description
p_in	1D Translation port (x).
P_out[3]	2D Port with three degrees of freedom (x, y, q).

Causality

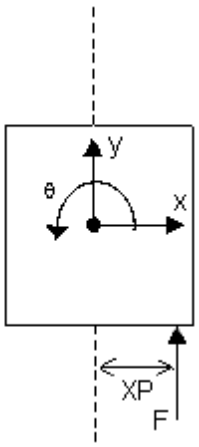
fixed force out
P_out
fixed velocity out
p_in

Parameters

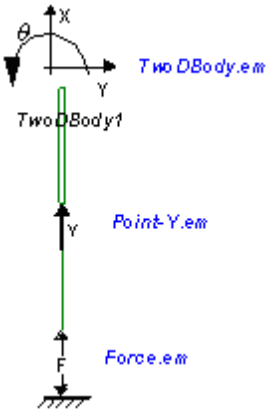
YP	Distance (y-direction) between connection and center of mass [m].

Description - Y

TwoDPoint models are used to connect 1D models to TwoDBodies. They indicate a 1D force or torque that is applied in a certain direction, onto a body. The TwoDPoint-Y model forms the connection between the single degree of freedom in y-direction and the center of mass of a body. Consider the example below, where force F in y-direction (single degree of freedom) is pushing upon a body (three degrees of freedom) at an offset *XP* from the body center of mass.



To create such a construct the 2D-point-Y model is used:



The 2D-point-Y model describes the transformation of force in y-direction to a force in three degrees of freedom. The single degree of freedom port *p_in* describes the connection in y-direction and the three three degree of freedom port *P_out* describes connection with the 2D-body (x, y and q).

```
P_out.F[1] = 0;                    // x-direction
P_out.F[2] = p_in.F;               // y-direction
P_out.F[3] = p_in.F * XP;          // rotation

p_in.v = P_out.v[2] + XP *         // y-direction
P_out.v[3];
```

Interface - Y

Ports	**Description**
p_in	1D Translation port (y).
P_out[3]	2D Port with three degrees of freedom (x, y, q).

Causality

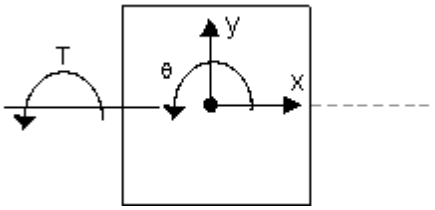
fixed force out
P_out
fixed velocity out
p_in

Parameters

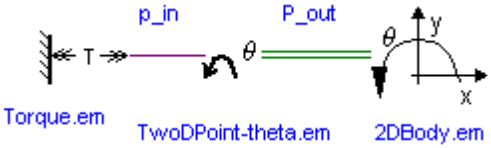
XP	Distance (x-direction) between connection and center of mass [m].

Description - theta

TwoDPoint models are used to connect 1D models to TwoDBodies. They indicate a 1D force or torque that is applied in a certain direction, onto a body. The TwoDPoint-Theta model forms the connection between the single degree of freedom rotation and the center of mass of a body. Consider the example below, where torque (single degree of freedom) is acting on a body (three degrees of freedom).



To create such a construct the TwoDPoint-Theta model is used:



The TwoDPoint-Theta model describes the transformation of a single rotation to a rotation of the 2D-body. The single degree of freedom port *p_in* describes the connection to the single degree rotation the three three degree of freedom port *P_out* describes connection with the 2D-body (x, y and q).

```
P_out.F[1] = 0;                 // x-direction
P_out.F[2] = 0;                 // y-direction
P_out.F[3] = p_in.T;            // rotation

p_in.omega = P_out.v[3];        // rotation
```

Interface - theta

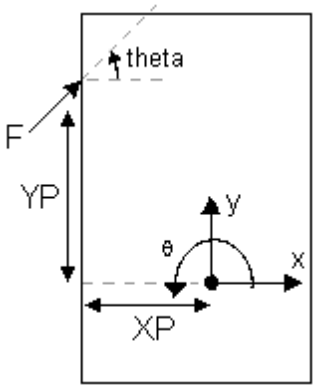
Ports	Description
p_in	1D Rotation port (q).
P_out[3]	2D Port with three degrees of freedom (x, y, q).

Causality

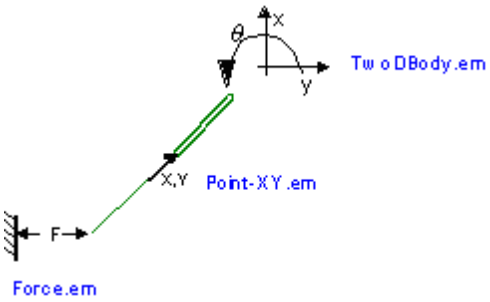
fixed force out
P_out
fixed velocity out
p_in

Description - XY

TwoDPoint models are used to connect 1D models to TwoDBodies. They indicate a 1D force or torque that is applied in a certain direction, onto a body. The TwoDPoint-XY model forms the connection between a single degree of freedom in arbitrary direction and the center of mass of a body. Consider the example below, where force *F* (single degree of freedom) is pushing upon a body (three degrees of freedom) with an arbitrary angle *theta* at an offset *XP* and *YP* from the body center of mass.



To create such a construct the TwoDPoint-XY model is used:



The TwoDPoint-XY model describes the transformation of a one degree of freedom force in an arbitrary direction to a force in three degrees of freedom. The single degree of freedom port *p_in* describes the connection in the arbitrary direction and the three three degree of freedom port *P_out* describes connection with the 2D-body (x, y and q).

```
Fx = cos(theta)*p_in.F;          // arbitrary-direction
Fy = sin(theta)*p_in.F;          // arbitrary-direction

P_out.F[1] = Fx;                 // x-direction
P_out.F[2] = Fy;                 // y-direction
P_out.F[3] = Fy * XP - Fx * YP;  // rotation
```

```
p_in.v = cos(theta)*P_out.v[1] +          // arbitrary-direction
sin(theta)*P_out.v[2] +
XP * sin(theta)* P_out.v[3] -
YP * cos(theta) * P_out.v[3];
```

Interface - XY

Ports	Description
p_in	1D Translation port in arbitrary direction.
P_out[3]	2D Port with three degrees of freedom (x, y, q).

Causality

fixed force out
P_out
fixed velocity out
p_in

Parameters

YP	Distance (y-direction) between connection and center of mass [m].
XP	Distance (x-direction) between connection and center of mass [m].
theta	Angle of impact of the single degree of freedom port [rad].

Note

- A TwoDBody has to be connected to at least three unequal TwoDPoint models to prevent it from free motion (rotation & translation).
- Flipping or rotating the model does not change the direction of applied forces or measured directions. Preferably leave the orientation as it pops up on the screen.

TwoDFixedWorld

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Iconic Diagrams (2D Planar).

Description

The TwoDFixedWorld represents the fixed world in 3 degrees of freedom. It can be connected to other 2D models to freeze their motion. Models that can be connected to the fixed world are:

1. TwoDSprings
2. TwoDLinearSlides

3. TwoDLinearActuators

Although other models like the TwoDBody can also be connected to the TwoDFixedWorld model, it makes no sense because it completely fixates all equations of motion.

Interface

Ports	Description
P	Port with three degrees of freedom (x, y, q).

Causality

fixed velocity out
P

TwoDZeroForce

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Iconic Diagrams (2D Planar).

Description

This model can be used to connect any open end of another 2D model that is not connected to the fixed world. It generates a three degree of freedom zero force while the velocity is free.

Interface

Ports	Description
P	Port with three degrees of freedom (x, y, q).

Causality

fixed force out P

Sensors

To inspect the model behavior you can plot variables. Sometimes not all variables are available. Then sensor model can be used.

1D

To use the 1D sensor models from the standard library, connect them to the 2D models using the TwoDPoint models.

2D

To measure variables directly from the 2D models, three absolute sensors are available showing the absolute position, velocity and acceleration. Also two relative sensors are available showing the force and powerflow between two elements.

TwoDPositionSensor

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Use

Domains: Continuous. **Size**: 1-D/2-D. **Kind**: Iconic Diagrams (1D Translation,1D Rotation, 2D Planar).

Description

This model is used to find the absolute position of a TwoDBody model. It has a port *P* which can be connected to the body and it has three output signals that denotes the *x*- and *y*-position and the angle of the body. The model has a force out causality. The equations are:

```
P.F = 0;
x = int (P.v[1]);
y = int (P.v[2]);
theta = int (P.v[3]);
```

Interface

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).
Causality	
fixed force out P	
Output	
x	Absolute position [m].
y	Absolute position [m].
theta	Absolute angle [rad].
Initial Values	
x_initial	Initial position [m]
y_initial	Initial position [m]
theta_initial	Initial angle [m]

TwoDVelocitySensor

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Use

Domains: Continuous. **Size**: 1-D/2-D. **Kind**: Iconic Diagrams (1D Translation,1D Rotation, 2D Planar).

Description

This model is used to find the absolute velocity of a TwoDBody model. It has a port *P* which can be connected to the body and it has three output signals that denotes the *x*- and *y*-velocity and the angular velocity of the body. The model has a force out causality. The equations are:

```
P.F = 0;
vx = P.v[1];
vy = P.v[2];
omega = P.v[3];
```

Interface

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).
Causality	
fixed force out P	
Output	
vx	Absolute velocity [m].
vy	Absolute velocity [m].
omega	Absolute angular velocity [rad].

TwoDAccelerationSensor

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

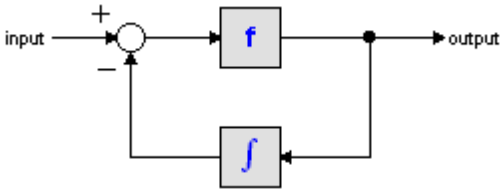
Use

Domains: Continuous. **Size**: 1-D/2-D. **Kind**: Iconic Diagrams (1D Translation, 1D Rotation, 2D Planar).

Description

This model is used to find the absolute acceleration of a TwoDBody model. It has a port *P* which can be connected to the body and it has three output signals that denotes the *x*- and *y*-acceleration and the angular acceleration of the body.

The acceleration output is calculated out of the velocity by differentiation. Differentiation is performed by a state variable filter:



The S-domain function of this filter is equal to:

$$output = \frac{s}{\frac{s}{f}+1} \cdot input$$

where *f* is the cut-off frequency. For very high values of *f*, the output becomes the pure derivative of the input. High values of *f*, however, increase simulations times. A good trade-off is a starting value of 1e5.

The equations of this model are:

```
P.F = 0;
ax = 2*pi*f*( P.v[1] - int(ax, 0));
ay = 2*pi*f*( P.v[2] - int(ay, 0));
alpha = 2*pi*f*( P.v[3] - int(alpha, 0));
```

Interface

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).
Causality	
fixed force out P	
Outputs	
ax	Absolute acceleration [m].
ay	Absolute acceleration [m].
alpha	Absolute acceleration [rad].
Initial Values	
ax_initial	Initial acceleration [m]
ay_initial	Initial acceleration [m]
alpha_initial	Initial angular acceleration [m]

TwoDForceSensor

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Use

Domains: Continuous. **Size**: 1-D/2-D. **Kind**: Iconic Diagrams (1D Translation, 1D Rotation, 2D Planar).

Description

This model can be used to measure the forces and torque between two components. It has a high and a low terminal to connect to both components. The equations are:

```
P.F = P_high.F = P_low.F;
P.v = P_high.v - P_low.v;
P.v = 0;
Fx = P.F[1];
Fy = P.F[2];
T = P.F[3];
```

Interface

Ports	Description
P[3]	Port with three degrees of freedom (x, y, q).

Causality

fixed velocity out

P

Outputs

Fx	Force [N].
Fy	Force [N].
T	Torque [Nm].

TwoDPowerSensor

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Use

Domains: Continuous. **Size**: 1-D/2-D. **Kind**: Iconic Diagrams (1D Translation, 1D Rotation, 2D Planar).

Description

This model can be used to measure the powerflow between two models. It has two ports to to both models. The equations are:

```
P.F = P_high.F = P_low.F;
P_high.F = P_low.F;
P_high.v = P_low.v;
P = P_high.v .* P_high.F;
Px = P[1];
Py = P[2];
Ptheta = P[3];
```

Interface

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).

Causality

fixed velocity out
P

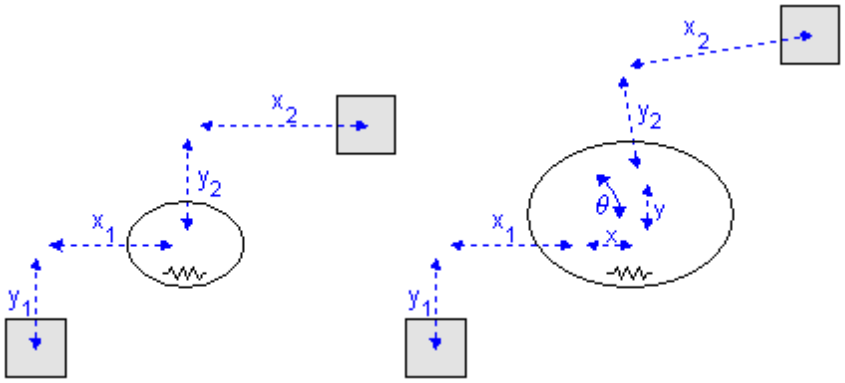
Outputs

Px	Power [W].
Py	Power [W].
Ptheta	Power [W].

Springs and other Flexible Components

2D Springs

A TwoDSpring model represents a flexible spring between two bodies. The center of stiffness is located relative to the two bodies. The offsets from the bodies to the center of stiffness are indicated by *x1*, *y1*, *x2* and *y2* as shown in the picture below (left).



A 2-D spring in rest (left) and stretched (right).

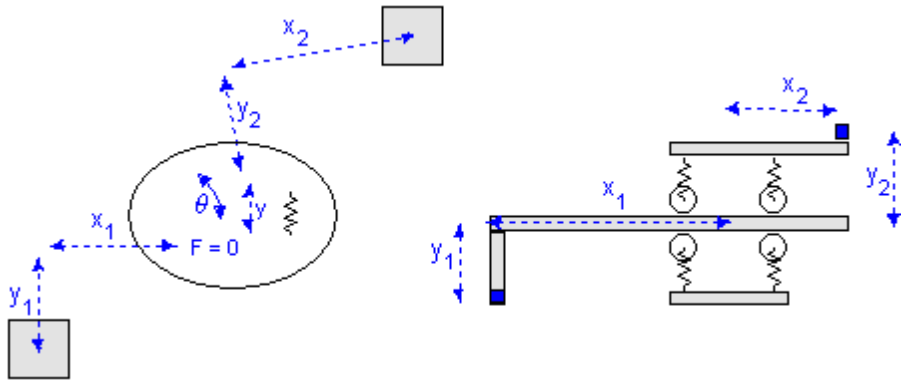
Note that *x1*, *y1*, *x2* and *y2* are defined with respect to the spring coordinate frame which is at the center of the spring. This means in the picture above *x1* and *y1* have a negative value and *x2* and *y2* have a positive value.

When the spring is stretched the offsets will change by the spring elongations *x*, *y* and q as indicated in the picture (right). The center of stiffness is always located precisely in the middle of the spring elongations. Because all spring equations will be calculated from the center of stiffness this means that the offsets for a stretched spring are (*x1* - *x*/2), (*y1* - *y*/2), (*x2* + *x*/2) and (*y2* + *y*/2).

For a realistic representation of vibration behavior, every 2D spring has a damper in parallel. If you want to turn off the effects of this damper, choose the damping parameters equal to zero. If you want to damp out unwanted vibrations, choose the damping parameters equal to 0.1% to 10% of the spring parameters.

Linear Slides

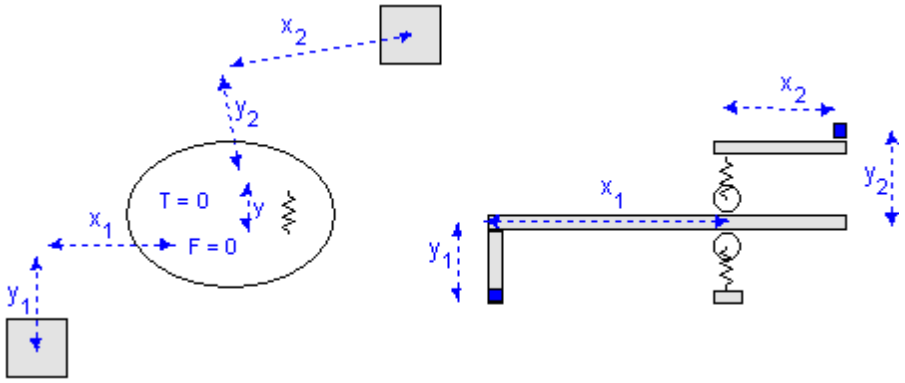
Take a look at the TwoDSpring model in the picture below. If we replace the spring in x-direction by a zero force we have created a linear slide.



Linear slide: x = free, y = stiff, q = stiff.

Consider a carriage that is clamped with four wheels to a slide. In the *x*-direction the carriage is free to move, which is equal to a zero force. The limited stiffness of the wheels is represented by springs. If there is a force with equal sign on both wheels, the carriage will slightly move in the *y*-direction. If there is a force with opposite sign on both wheels, the carriage will only rotate. This can be represented by two independent springs: one in the *y*-direction and q-direction.

If we repeat the same experiment and replace the spring in *x*-direction by a zero force and the spring in q-direction by a zero torque, we have created a linear slide with rotational freedom. In practice this can be found with carriages that are clamped with a limited set of wheels as shown below.



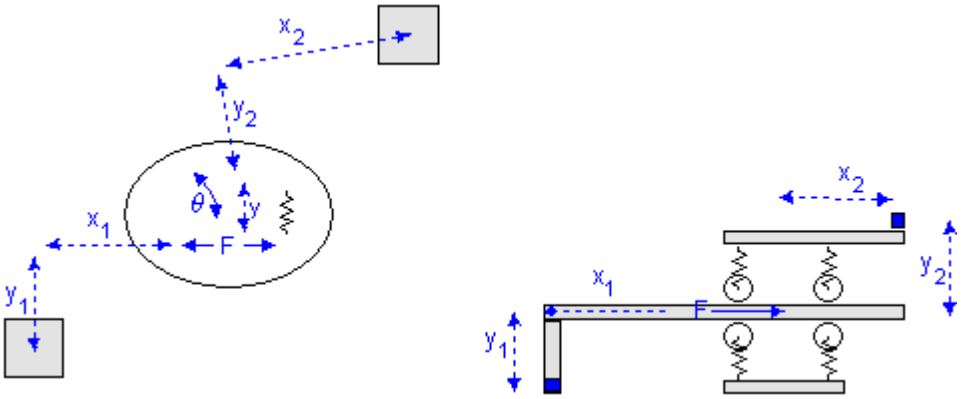
Linear slide: x = free, y = stiff, q = free.

The TwoDLibrary contains four Linear Slide models:

1	TwoDLinearSlide-X	*x* = free, *y* = stiff, *q* = stiff
2	TwoDLinearSlide-Y	*x* = stiff, *y* = free, *q* = stiff
3	TwoDLinearSlide-XTheta	*x* = free, *y* = stiff, *q* = free
4	TwoDLinearSlide-YTheta	*x* = free, *y* = stiff, *q* = free

Linear Actuators

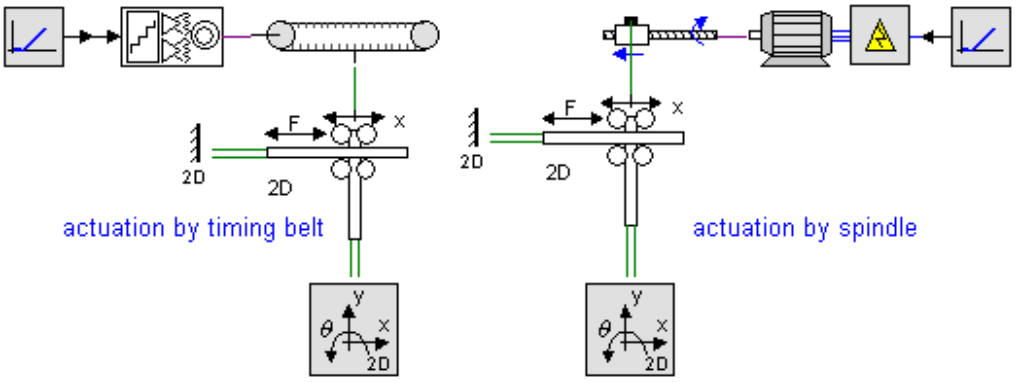
Instead of replacing the spring in *x*-direction with a zero force as with the linear slides, we can replace it by a 1D power port to allow all kind of connections with the models from the standard library. In this way we have created a linear actuator, that allows every kind of actuation.



Like the linear slides, for the linear actuators we can also replace the spring in q-direction by a zero torque, which makes the total number of linear actuator models equal to four:

1	TwoDLinearActuator-X	*Fx* and *vx* connected to a port
2	TwoDLinearActuator-Y	*Fy* and *vy* connected to a port
3	TwoDLinearActuator-XTheta	*Fx* and *vx* connected to a port, *T* = 0, w = 0
4	TwoDLinearActuator-YTheta	*Fy* and *vy* connected to a port, *T* = 0, w = 0

With the port connection ever possible form of actuation can be created. Two examples are shown below.



TwoDSpring

Library

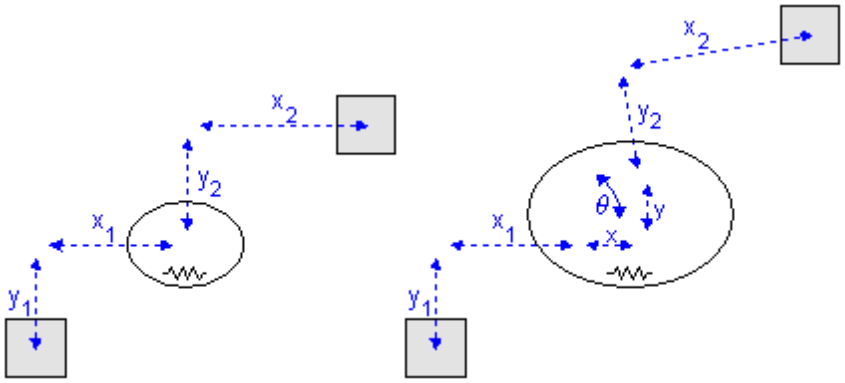
Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Use

Domains: Continuous. **Size**: 2-D. **Kind**: Iconic Diagrams (2D Planar).

Description

A TwoDSpring model represents a flexible connection between two bodies. The bodies are located relative to the center of stiffness. The offsets from the bodies to the center of stiffness are indicated by *x1*, *y1*, *x2* and *y2* as shown in the picture below (left).



A 2-D spring in rest (left) and stretched (right).

Note that *x1*, *y1*, *x2* and *y2* are defined with respect to the spring coordinate frame which is at the center of the spring. This means in the picture above *x1* and *y1* have a negative value and *x2* and *y2* have a positive value.

When the spring is stretched the offsets will change by the spring elongations *x*, *y* and *q* as indicated in the picture (right). The center of stiffness is always located precisely in the middle of the spring elongations. Because all spring equations will be calculated from the center of stiffness this means that the offsets for a stretched spring are (*x1* - *x*/2), (*y1* - *y*/2), (*x2* + *x*/2) and (*y2* + *y*/2).

For a realistic representation of vibration behavior, every 2D spring has a damper in parallel. If you want to turn off the effects of this damper, choose the damping parameters equal to zero. If you want to damp out unwanted vibrations, choose the damping parameters equal to 0.1% to 10% of the spring parameters.

Interface

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).
Causality	
fixed force out P	
Parameters	
x1	distance in x-direction from center of stiffness to port 1 [m]
y1	distance in y-direction from center of stiffness to port 1 [m]
x2	distance in x-direction from center of stiffness to port 2 [m]
y2	distance in y-direction from center of stiffness to port 2 [m]
kx	stiffness in x-direction [N/m]
ky	stiffness in y-direction [N/m]
kth	rotational stiffness [N.m/rad]
dx	damping in x-direction [N.s/m]

dy	damping in y-direction [N.s/m]
dth	rotational damping [N.m.s/rad]
Initial Values	
x_initial	The initial extension of the spring [m].
y_initial	The initial extension of the spring [m].
theta_initial	The initial rotation of the spring [rad].

TwoDLinearSlide

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Implementations

X
Y
XTheta
YTheta

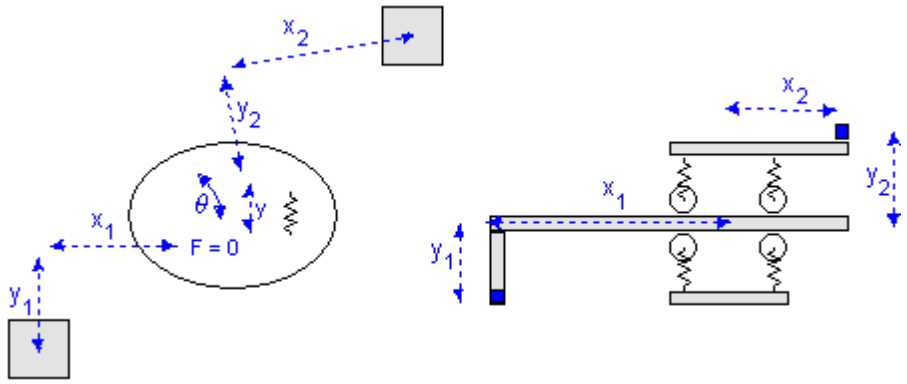
Use

Domains: Continuous. **Size**: 2-D. **Kind**: Iconic Diagrams (2D Planar).

Description - X

The TwoDLinearSlide-X model represents a flexible connection between two bodies in *y*- and q-direction. The bodies are located relative to the center of stiffness. The offsets from the bodies to the center of stiffness are indicated by the parameters *x1*, *y1*, *x2* and *y2* as shown in the picture below.

The TwoDLinearSLide-X model is based on the TwoDSpring model by replacing the spring and damper in x-direction by a zero force. Consider a carriage that is clamped with four wheels to a slide. In the x-direction the carriage is free to move, which is equal to a zero force. The limited stiffness of the wheels is represented by springs. If there is a force with equal sign on both wheels, the carriage will slightly move in the y-direction. If there is a force with opposite sign on both wheels, the carriage will only rotate. This can be represented by two independent springs: one in the y-direction and q-direction.



Linear slide: x = free, y = stiff, q = stiff.

For a realistic representation of vibration behavior, every spring has a damper in parallel. If you want to turn off the effects of this damper, choose the damping parameters equal to zero. If you want to damp out unwanted vibrations, choose the damping parameters equal to 0.1% to 10% of the spring parameters.

Interface - X

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).
Causality	
fixed force out P	
Parameters	
x1	distance in x-direction from center of stiffness to port 1 [m]
y1	distance in y-direction from center of stiffness to port 1 [m]
x2	distance in x-direction from center of stiffness to port 2 [m]
y2	distance in y-direction from center of stiffness to port 2 [m]
ky	stiffness in y-direction [N/m]
kth	rotational stiffness [N.m/rad]
dy	damping in y-direction [N.s/m]
dth	rotational damping [N.m.s/rad]
Initial Values	
y_initial	The initial extension of the spring [m].
theta_initial	The initial rotation of the spring [rad].

Description - Y

The TwoDLinearSlide-Y model is equal to the TwoDLinearSlide-X model except that here the movement is free in the y-direction.

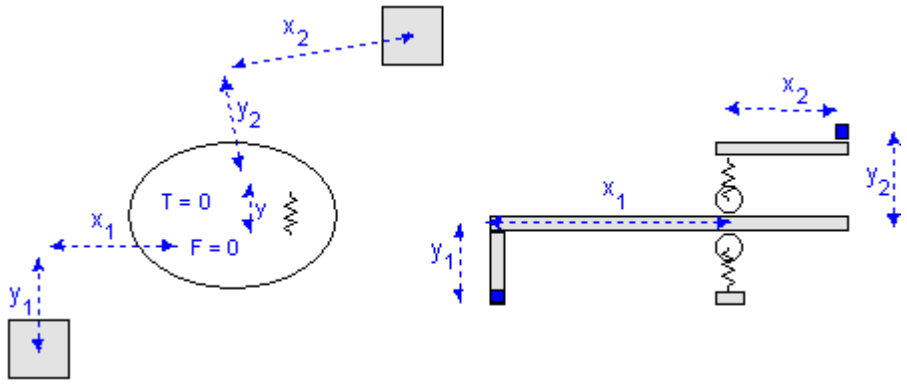
Interface - Y

Ports	Description
P[3]	Port with three degrees of freedom (x, y, q).
Causality	
fixed force out P	
Parameters	
x1	distance in x-direction from center of stiffness to port 1 [m]
y1	distance in y-direction from center of stiffness to port 1 [m]
x2	distance in x-direction from center of stiffness to port 2 [m]
y2	distance in y-direction from center of stiffness to port 2 [m]
kx	stiffness in x-direction [N/m]
kth	rotational stiffness [N.m/rad]
dx	damping in x-direction [N.s/m]
dth	rotational damping [N.m.s/rad]
Initial Values	
x_initial	The initial extension of the spring [m].
theta_initial	The initial rotation of the spring [rad].

Description - XTheta

The TwoDLinearSlide-XTheta model represents a flexible connection between two bodies in y-direction. The bodies are located relative to the center of stiffness. The offsets from the bodies to the center of stiffness are indicated by the parameters *x1*, *y1*, *x2* and *y2* as shown in the picture below.

The TwoDLinearSLide-X model is based on the TwoDSpring model by replacing the spring and damper in x- and q-direction by a zero force. Consider a carriage that is clamped with two wheels to a slide. In the x-direction the carriage is free to move, which is equal to a zero force. The carriage is also free to rotate, which is equal to a zero torque. The limited stiffness of the wheels is represented by springs. If there is a force on the wheels, the carriage will slightly move in the y-direction.



Linear slide: x = free, y = stiff, q = stiff.

For a realistic representation of vibration behavior, every spring has a damper in parallel. If you want to turn off the effects of this damper, choose the damping parameters equal to zero. If you want to damp out unwanted vibrations, choose the damping parameters equal to 0.1% to 10% of the spring parameters.

Interface - XTheta

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).
Causality	
fixed force out P	
Parameters	
x1	distance in x-direction from center of stiffness to port 1 [m]
y1	distance in y-direction from center of stiffness to port 1 [m]
x2	distance in x-direction from center of stiffness to port 2 [m]
y2	distance in y-direction from center of stiffness to port 2 [m]
ky	stiffness in y-direction [N/m]
dy	damping in y-direction [N.s/m]
Initial Values	
y_initial	The initial extension of the spring [m].

Description - YTheta

The TwoDLinearSlide-YTheta model is equal to the TwoDLinearSlide-XTheta model except that here the movement is free in the y and q-direction.

Interface - YTheta

Ports	**Description**

P[3]	Port with three degrees of freedom (x, y, q).

Causality

fixed force out P

Parameters

x1	distance in x-direction from center of stiffness to port 1 [m]
y1	distance in y-direction from center of stiffness to port 1 [m]
x2	distance in x-direction from center of stiffness to port 2 [m]
y2	distance in y-direction from center of stiffness to port 2 [m]
kx	stiffness in x-direction [N/m]
dx	damping in x-direction [N.s/m]

Initial Values

x_initial	The initial extension of the spring [m].

TwoDLinearActuator

Library

Iconic Diagrams\Mechanical\Translation\2DSmallAngles

Implementations

X
Y
XTheta
YTheta

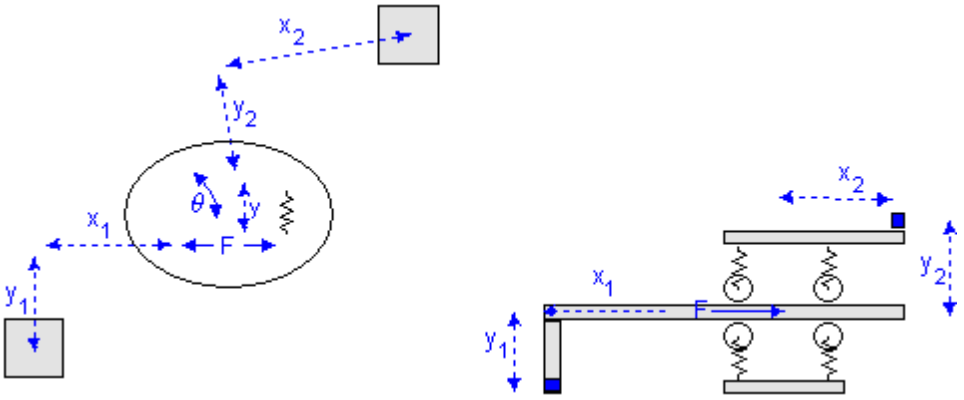
Use

Domains: Continuous. **Size**: 1-D / 2-D. **Kind**: Iconic Diagrams (1D Translation, 2D Planar).

Description - X

The TwoDLinearActuator-X model represents a flexible connection between two bodies in *y*- and q-direction. The TwoDLinearActuator-X model is based on the TwoDLinearSlide-X model. Instead of using a zero force in the x-direction, there is a coupling with a 1D translation port for the actuator force.

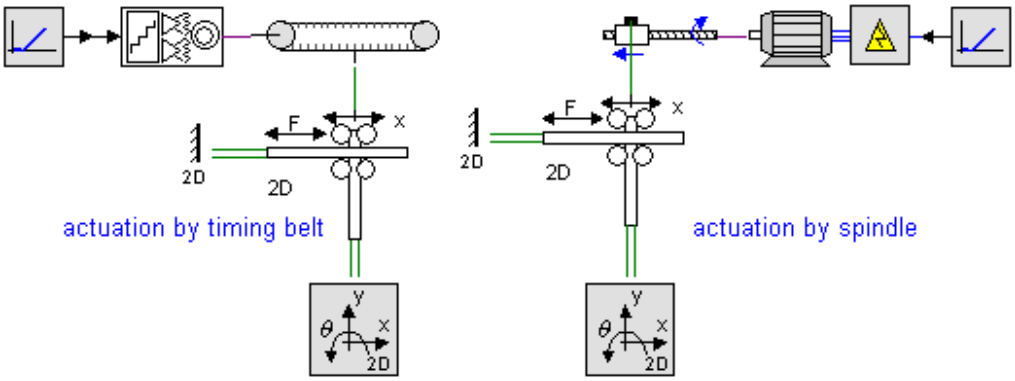
Consider a carriage that is clamped with four wheels to a slide. In the x-direction the carriage can be moved by a certain force. The limited stiffness of the wheels is represented by springs. If there is a force with equal sign on both wheels, the carriage will slightly move in the y-direction. If there is a force with opposite sign on both wheels, the carriage will only rotate. This can be represented by two independent springs: one in the y-direction and q-direction.



Linear slide: x = free, y = stiff, q = stiff.

For a realistic representation of vibration behavior, every spring has a damper in parallel. If you want to turn off the effects of this damper, choose the damping parameters equal to zero. If you want to damp out unwanted vibrations, choose the damping parameters equal to 0.1% to 10% of the spring parameters.

With the 1D port connection every possible form of actuation can be created using the models from the standard library. Two examples are shown below.



Interface - X

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).

p	1D translation port
Causality	
fixed force out P	
Parameters	
x1	distance in x-direction from center of stiffness to port 1 [m]
y1	distance in y-direction from center of stiffness to port 1 [m]
x2	distance in x-direction from center of stiffness to port 2 [m]
y2	distance in y-direction from center of stiffness to port 2 [m]
ky	stiffness in y-direction [N/m]
kth	rotational stiffness [N.m/rad]
dy	damping in y-direction [N.s/m]
dth	rotational damping [N.m.s/rad]
Initial Values	
y_initial	The initial extension of the spring [m].
theta_initial	The initial rotation of the spring [rad].

Description - Y

The TwoDLinearActuator-Y model is equal to the TwoDLinearActuator-X model except that here the force acts in the y-direction.

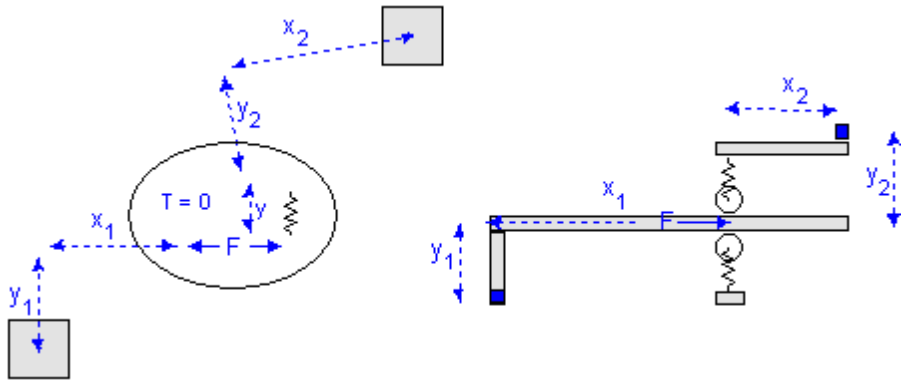
Interface - Y

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).
p	1D translation port
Causality	
fixed force out P	
Parameters	
x1	distance in x-direction from center of stiffness to port 1 [m]
y1	distance in y-direction from center of stiffness to port 1 [m]
x2	distance in x-direction from center of stiffness to port 2 [m]
y2	distance in y-direction from center of stiffness to port 2 [m]
kx	stiffness in x-direction [N/m]
kth	rotational stiffness [N.m/rad]
dx	damping in x-direction [N.s/m]
dth	rotational damping [N.m.s/rad]
Initial Values	
x_initial	The initial extension of the spring [m].
theta_initial	The initial rotation of the spring [rad].

Description - XTheta

The TwoDLinearActuator-XTheta model represents a flexible connection between two bodies in *y*-direction. The TwoDLinearActuator-XTheta model is based on the TwoDLinearSlide-XTheta model. Instead of using a zero force in the x-direction, there is a coupling with a 1D translation port for the actuator force.

Consider a carriage that is clamped with two wheels to a slide. In the x-direction the carriage can be moved by a certain force. The carriage is free to rotate, which is equal to a zero torque. The limited stiffness of the wheels is represented by springs. If there is a force acting on both wheels, the carriage will slightly move in the y-direction. If there is a force acting on the wheels, the carriage will slightly move in the y-direction.



Linear slide: x = free, y = stiff, q = stiff.

For a realistic representation of vibration behavior, the spring has a damper in parallel. If you want to turn off the effects of this damper, choose the damping parameters equal to zero. If you want to damp out unwanted vibrations, choose the damping parameter equal to 0.1% to 10% of the spring parameter.

With the 1D port connection every possible form of actuation can be created using the models from the standard library. For example see the TwoDLinearActuator-X model.

Interface - XTheta

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).
p	1D translation port
Causality	
fixed force out P	
Parameters	
x1	distance in x-direction from center of stiffness to port 1 [m]
y1	distance in y-direction from center of stiffness to port 1 [m]
x2	distance in x-direction from center of stiffness to port 2 [m]
y2	distance in y-direction from center of stiffness to port 2 [m]

ky	stiffness in y-direction [N/m]
dy	damping in y-direction [N.s/m]
Initial Values	
y_initial	The initial extension of the spring [m].

Description - YTheta

The TwoDLinearActuator-YTheta model is equal to the TwoDLinearActuator-XTheta model except that here the force acts in the y-direction.

Interface - YTheta

Ports	**Description**
P[3]	Port with three degrees of freedom (x, y, q).
p	1D translation port
Causality	
fixed force out P	
Parameters	
x1	distance in x-direction from center of stiffness to port 1 [m]
y1	distance in y-direction from center of stiffness to port 1 [m]
x2	distance in x-direction from center of stiffness to port 2 [m]
y2	distance in y-direction from center of stiffness to port 2 [m]
kx	stiffness in x-direction [N/m]
kth	rotational stiffness [N.m/rad]
dx	damping in x-direction [N.s/m]
dth	rotational damping [N.m.s/rad]
Initial Values	
x_initial	The initial extension of the spring [m].
theta_initial	The initial rotation of the spring [rad].

Tips

When you are working with the 2D-Library for small angles, it is good to keep the following tips in mind:

- The 2DSmallRotations library is very well suited for frequency domain analysis through linearization.
- The 2DSmallRotations library is not suited for large rotations. Consider a rotation of 0.05 degrees or 1 mrad as an upper bound for every model part.
- It is not possible in 20-sim to use vector elements with mixed units. Therefore element number 3 will be displayed with units [m/s] and [N] although it really is [rad/s] and [Nm]!
- The vector element number (1, 2 or 3) denotes the degree of freedom. Flipping or rotating of a model does not change this degree (e.g. a rotation does not change from x into y). Preferably leave the orientation of the models as they pop up on the screen.
- To create intuitive models, the positive direction of each degree of freedom is defined as: x from left to right, y from bottom to top, ? counter clockwise.
- To prevent misinterpretation of a model by other users, stick to the drawing rules that are given.
- A TwoDBody model can be connected to any other element of the 2D-library except other 2D-bodies.
- A TwoDBody model can be connected to 1D models using the TwoDPoint models as intermediate.
- TwoDPoint models can only be connected to TwoDBody models.
- If a TwoDBody model is connected to TwoDPoint models only, it has to be connected to at least three non-coinciding, unequal (i.e. not having the same offsets) TwoDPoints to prevent it from free motion.

3DSmallRotation

ThreeDLibrary

Library

The 3*DSmallRotation* library contains 3D models that are only suited for small rotations. The library is comparable to the 2DSmallRotation library and suited for modeling systems that, due to limited stiffness, experience rotation that should be accounted for.

Degrees of Freedom

All models in 3*DSmallRotation* library contain one or more ports with 6 degrees of freedom. The first three degrees of freedom denote the x-, y- and z-position and the last three degrees of freedom denote the rotation around the x-, y- and z-axes. A vector notation is used to denote the forces and velocities for these degrees of freedom. For a model with a port *P* the notation is:

degree	identity	forces	velocities
x	position	P.F[1] [N]	P.v[1] [m/s]

y	position	P.F[2] [N]	P.v[2] [m/s]
z	position	P.F[3] [N]	P.v[3] [m/s]
q_x	angle	P.F[4] [Nm]	P.v[4] [rad/s]
q_y	angle	P.F[5] [Nm]	P.v[5] [rad/s]
q_z	angle	P.F[6] [Nm]	P.v[6] [rad/s]

Note

- It is not possible in 20-sim to use vector elements with mixed units. Therefore element number 4 to 6 will be displayed with units [m/s] and [N] although it really is [rad/s] and [Nm]!
- The vector element number (1 to 6) denotes the identity of the degree of freedom. Flipping or rotating of a model does not change the this identity (e.g. a rotation does not change x into y). Preferably leave the orientation of the models as they pop up on the screen.

3D-body

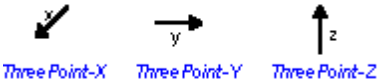
TwoDBody

The central model in the *3DSmallRotation* library is the 3D-body model which describes a body with 6 degrees of freedom.

3D-mass

The 3D-mass model is a body with only mass and zero rotational inertia.

Points



Three Point-X *Three Point-Y* *Three Point-Z*

The 3D-body model is nothing more than a center of mass. To connect a 3D-body with the outside (single degree of freedom) world, 3D-points are used. A 3D-point describes the offset between the center of mass and the connection point.

- A 3D-body has to be connected to 3D-points with at least three orthogonal nonzero offsets to prevent it from free rotation.
- A 3D-body has to be connected to at least three orthogonal 3D-point models to prevent it from free motion.

Sensors

The velocities of bodies can be inspected directly in the body models. To find positions sensor models are available.

ThreeDBody

Library

Iconic Diagrams\Mechanical\Translation\3DSmallAngles

Use

Domains: Continuous. **Size**: 6D. **Kind**: Iconic Diagrams (Translation,Rotation).

Description

This is a model of a 6 degree of freedom body. It is the equivalent of the 2D-body model. The first three degrees of freedom in this model are represented by a single mass and the last three degrees of freedom by a rotational inertia. The 3D-body has to be connected to at least three orthogonal 3D-points, which define the reaction forces.

The model has a preferred velocity out causality. The corresponding constitutive equations then contain an integration. The model can also have the non-preferred force out causality. The constitutive equations then contain a differentiation. Because any number of connections can be made, successive ports are named *P1*, *P2*, *P3* etc. 20-sim will automatically create equations such that the resulting force *P.F* is equal to the sum of the forces of all connected ports *P1* .. *Pn*. The velocities of all connected ports are equal to *P.v*

```
P.F = sum(P1.F, P2.F, ....)
P.v = P1.v = P2.v = ....
```

velocity out causality (preferred):

```
I = [1/M;1/M;1/M;1/J[1];1/J[2];1/J[3]];
P.v = I.*int(P.F);
```

force out causality:

```
I = [1/M;1/M;1/M;1/J[1];1/J[2];1/J[3]];
P.F = I-1.*ddt(P.v);
```

Interface

Ports	**Description**
P[6]	Port with 6 degrees of freedom. Any number of connections can be made.

Causality

preferred velocity out

Parameters

M	Mass [kg].
J[3]	Moment of inertia for three axes [kgm^2]

Note

- A body has to be connected to 3D-points with at least three orthogonal nonzero offsets to prevent it from free rotation.
- A 3D-body has to be connected to at least three orthogonal 3D-point models to prevent it from free motion.
- Flipping or rotating the model does not change the direction of applied forces or measured directions. Preferably leave the orientation as it pops up on the screen.
- It is not possible in 20-sim to use vector elements with mixed units. Therefore element number 4 to 6 will be displayed with units [m/s] and [N] although it really is [rad/s] and [Nm]!

ThreeDFixedWorld

Library

Iconic Diagrams\Mechanical\Translation\3DSmallAngles

Use

Domains: Continuous. **Size**: 6-D. **Kind**: Iconic Diagrams (Translation,Rotation).

Description

This model represents the fixed world in 6 degrees of freedom. It can be connected to a 3D-body model to freeze its motion.

Interface

Ports	**Description**
P	Port with 6 degrees of freedom.

Causality

fixed velocity out

P

ThreeDForceActuator-Relative

Library

Iconic Diagrams\Mechanical\Translation\3DSmallAngles

Use

Domains: Continuous. **Size**: 1-D/6-D. **Kind**: Iconic Diagrams (Translation,Rotation).

Description

This model represents an ideal actuator. The actuator applies a force between its two terminals. This force can be set to a (fluctuating) value given by the input signal *F*, the velocity is indifferent.

This model is the equivalent of the 1 degree of fredom model `ForceActuator-Relative.emx`. It model represents an ideal actuator. The actuator applies a force between its two terminals. The force can be set to a (fluctuating) value given by a vector K multiplied by a signal *F*, the velocity is indifferent. By setting the elements of the vector K equal to zero or one, specific elements of the 6 degrees of freedom output force can set equal to the input signal *F.*

*P_high.F[1:6] = P_low.F[1:6] = K*F;*

Interface

Ports	**Description**
P_high[6], P_low[6]	Both terminals of the port with 6 degrees of freedom.
Causality	
fixed force out P	
Parameters	
K[6]	Vector with 6 elements to set the 6 force elements on or off.

ThreeDForceActuator

Library

Iconic Diagrams\Mechanical\Translation\3DSmallAngles

Use

Domains: Continuous. **Size**: 1-D/6-D. **Kind**: Iconic Diagrams (Translation,Rotation).

Description

This model is the equivalent of the 1 degree of fredom model *ForceActuator.emx.* It model represents an ideal actuator. The actuator is mounted to the fixed world and applies a force. The force can be set to a (fluctuating) value given by a vector K multiplied by a signal *F*, the velocity is indifferent. By setting the elements of the vector K equal to zero or one, specific elements of the 6 degrees of freedom output force can set equal to the input signal *F.*

*P.F[1:6] = K*F;*

Interface

Ports	**Description**
P	Port with 6 degrees of freedom.
Causality	
fixed force out P	
Parameters	
K[6]	Vector with 6 elements to set the 6 force elements on or off.

ThreeDMass

Library

Iconic Diagrams\Mechanical\Translation\3DSmallAngles

Use

Domains: Continuous. **Size**: 6D. **Kind**: Iconic Diagrams (Translation,Rotation).

Description

This is a model of a 6 degree of freedom body without rotational inertia. The 3D-mass has to be connected to at least three orthogonal 3D-points, which define the reaction forces.

The model has a fised velocity out causality. Because any number of connections can be made, successive ports are named *P1*, *P2*, *P3* etc. 20-sim will automatically create equations such that the resulting force *P.F* is equal to the sum of the forces of all connected ports *P1* .. *Pn*. The velocities of all connected ports are equal to *P.v*

```
P.F = sum(P1.F, P2.F, ....)
P.v = P1.v = P2.v = ....
I = [1/M;1/M;1/M];
P.v[1..3] = I.*int(P.F);
P.v[4..6] = 0;
```

Interface

Ports	Description
P[6]	Port with 6 degrees of freedom. Any number of connections can be made.

Causality

fixed velocity out

Parameters

M	Mass [kg].

Note

- A 3D-body has to be connected to 3D-points with at least three orthogonal nonzero offsets to prevent it from free rotation.
- A 3D-body has to be connected to at least three orthogonal 3D-point models to prevent it from free motion.
- Flipping or rotating the model does not change the direction of applied forces or measured directions. Preferably leave the orientation as it pops up on the screen.
- It is not possible in 20-sim to use vector elements with mixed units. Therefore element number 4 to 6 will be displayed with units [m/s] and [N] although it really is [rad/s] and [Nm]!

ThreeDPoint

Library

Iconic Diagrams\Mechanical\Translation\3DSmallAngles

Implementations

X
Y
Z
XYZ

Use

Domains: Continuous. **Size**: 1-D/6-D. **Kind**: Iconic Diagrams (Translation,Rotation).

Description - X

A point model forms the connection between the single degree of freedom part of a system and the center of mass of a 3D-body.
The 3D-point-X model is the equivalent of the 2D-point-X model. It describes the translation of force in x-direction to a force in 6 degrees of freedom. Because there are three rotational degrees of freedom, the 3D-point-X model has two offsets, *YP* and *ZP*.

The single degree of freedom port *p_in* describes the connection in x-direction and the 6 degree of freedom port *P_out* describes connection with the 3D-body.

```
P_out.F[1] = p_in.F;                    // x-direction
P_out.F[2] = 0;                         // y-direction
P_out.F[3] = 0;                         // z-direction
P_out.F[4] = 0;                         // x-rotation
P_out.F[5] = ZP*p_in.F;                 // y-rotation
P_out.F[6] = -YP*p_in.F;                // z-rotation

p_in.v = P_out.v[1] - YP*P_out.v[6]     // x-direction
+ ZP *P_out.v[5];
```

As can bee seen from the equations, a nonzero offset *YP* or *ZP* (distance from center of mass) in a 3D-point will result in a momentum. Therefore a 3D-body has to be connected to 3D-points with at least three orthogonal nonzero offsets to prevent it from free rotation.

The equations also show that the y-direction and z-direction are not affected by the *3D-point-X* model. Therefore each 3D-body has to be connected to at least three orthogonal 3D-point models to prevent it from free motion.

Interface - X

Ports	Description

p_in	Translation port with one degree of freedom (x [m]).
P_out[3]	Port with 6 degrees of freedom.

Causality

fixed force out
P_out
fixed velocity out
p_in

Parameters

YP	Distance (y-direction) between connection and center of mass [m].
ZP	Distance (z-direction) between connection and center of mass [m].

Description - Y

A point model forms the connection between the single degree of freedom part of a system and the center of mass of a 3D-body.
The 3D-point-Y model is the equivalent of the 2D-point-Y model. It describes the translation of force in x-direction to a force in 6 degrees of freedom. Because there are three rotational degrees of freedom, the 3D-point-Y model has two offsets, X*P* and *ZP*.

The single degree of freedom port *p_in* describes the connection in x-direction and the 6 degree of freedom port *P_out* describes connection with the 3D-body.

```
P_out.F[1] = 0;                    // x-direction
P_out.F[2] = p_in.F;               // y-direction
P_out.F[3] = 0;                    // z-direction
P_out.F[4] = -ZP*p_in.F;           // x-rotation
P_out.F[5] = 0;                    // y-rotation
P_out.F[6] = XP*p_in.F;            // z-rotation

p_in.v = P_out.v[2] + XP*P_out.v[6]   // y-direction
- ZP *P_out.v[4];
```

As can bee seen from the equations, a nonzero offset X*P* or *ZP* (distance from center of mass) in a 3D-point will result in a momentum. Therefore a 3D-body has to be connected to 3D-points with at least three orthogonal nonzero offsets to prevent it from free rotation.

The equations also show that the x-direction and z-direction are not affected by the 3*D-point-Y* model. Therefore each 3D-body has to be connected to at least three orthogonal 3D-point models to prevent it from free motion.

Interface - Y

Ports	Description

p_in	Translation port with one degree of freedom (x [m]).
P_out[3]	Port with 6 degrees of freedom.

Causality

fixed force out
P_out
fixed velocity out
p_in

Parameters

XP	Distance (x-direction) between connection and center of mass [m].
ZP	Distance (z-direction) between connection and center of mass [m].

Description - Z

A point model forms the connection between the single degree of freedom part of a system and the center of mass of a 3D-body.
The 3D-point-Z model is the equivalent of the 2D-point-X model. It describes the translation of force in z-direction to a force in 6 degrees of freedom. Because there are three rotational degrees of freedom, the 3D-point-Y model has two offsets, X*P* and *YP*.

The single degree of freedom port *p_in* describes the connection in x-direction and the 6 degree of freedom port *P_out* describes connection with the 3D-body.

```
P_out.F[1] = 0;                    // x-direction
P_out.F[2] = 0;                    // y-direction
P_out.F[3] = p_in.F;               // z-direction
P_out.F[4] = YP*p_in.F;            // x-rotation
P_out.F[5] = -XP*p_in.F;           // y-rotation
P_out.F[6] = 0;                    // z-rotation

p_in.v = P_out.v[3] - XP*P_out.v[5]   // y-direction
+ YP *P_out.v[4];
```

As can bee seen from the equations, a nonzero offset X*P* or Y*P* (distance from center of mass) in a 3D-point will result in a momentum. Therefore a 3D-body has to be connected to 3D-points with at least three orthogonal nonzero offsets to prevent it from free rotation.

The equations also show that the x-direction and y-direction are not affected by the 3*D-point-Z* model. Therefore each 3D-body has to be connected to at least three orthogonal 3D-point models to prevent it from free motion.

Interface - Z

Ports	**Description**
p_in	Translation port with one degree of freedom (x [m]).

P_out[3]	Port with 6 degrees of freedom.

Causality

fixed force out
P_out
fixed velocity out
p_in

Parameters

XP	Distance (x-direction) between connection and center of mass [m].
YP	Distance (y-direction) between connection and center of mass [m].

Description - XYZ

A point model forms the connection between the single degree of freedom part of a system and the center of mass of a 3D-body.

The 3D-point-XYZ model is a combination of the 3D-point-X model, the 3D-point-Y model and the 3D-point-Z model. It describes the translation of force in x-direction, y-direction and z-ditrection to a force in 6 degrees of freedom. Because there are three rotational degrees of freedom, the model has three offsets, *XP*, *YP* and *ZP*.

The single degree of freedom ports *p_in*x, *p_iny* and *p_inz* describes the connection in x-direction, y-direction and z-direction and the 6 degree of freedom port *P_out* describes connection with the 3D-body.

```
P_out.F[1] = p_inx.F;                    // x-direction
P_out.F[2] = p_iny.F;                    // y-direction
P_out.F[3] = p_inz.F;                    // z-direction
P_out.F[4] = YP*p_inz.F - ZP*p_iny.F;    // x-rotation
P_out.F[5] = -XP*p_inz.F + ZP*p_inx.F;   // y-rotation
P_out.F[6] = XP*p_iny.F - YP*p_inx.F;    // z-rotation
```

```
p_inx.v = P_out.v[1] - YP*P_out.v[6] + ZP // x-direction
*P_out.v[5];                              // y-direction
p_iny.v = P_out.v[2] + XP*P_out.v[6] - ZP // z-direction
*P_out.v[4];
p_inz.v = P_out.v[3] - XP*P_out.v[5]
+ YP *P_out.v[4];
```

As can bee seen from the equations, a nonzero offset *XP, YP* or *ZP* (distance from center of mass) in a 3D-point will result in a momentum. **Therefore a 3D-body has to be connected to 3D-points with at least three orthogonal nonzero offsets to prevent it from free rotation**.

Interface - XYZ

Ports	Description
p_inx	Translation port with one degree of freedom [m].
p_iny	Translation port with one degree of freedom [m].
p_inz	Translation port with one degree of freedom [m].
P_out[3]	Port with 6 degrees of freedom.

Causality

fixed force out
P_out
fixed velocity out
p_in

Parameters

YP	Distance (y-direction) between connection and center of mass [m].
XP	Distance (x-direction) between connection and center of mass [m].
theta	Angle of impact of the single degree of freedom port [rad].

Note

- A body has to be connected to 3D-points with at least three orthogonal nonzero offsets to prevent it from free rotation.
- A 3D-body has to be connected to at least three orthogonal 3D-point models to prevent it from free motion.
- Flipping or rotating the model does not change the direction of applied forces or measured directions. Preferably leave the orientation as it pops up on the screen.
- It is not possible in 20-sim to use vector elements with mixed units. Therefore element number 4 to 6 will be displayed with units [m/s] and [N] although it really is [rad/s] and [Nm]!

ThreeDPositionSensor

Library

Iconic Diagrams\Mechanical\Translation\3DSmallAngles

Use

Domains: Continuous. **Size**: 1-D/6D. **Kind**: Iconic Diagrams (Translation,Rotation).

Description

This model is used to find the positions of a 3D-body model. It has a port *P* which can be connected to the body and an output signal pos that denotes the positions (x,y,z,qx,qy,qz) of the body.

Interface

Ports	Description
P[3]	Port with 6 degrees of freedom.
Causality	
fixed force out P	
Output	
Pos(6)	Absolute positions [m].
Initial Values	
Pos_initial[6]	Initial positions [m]

Note

- The sensor model yields an absolute position starting at 0. Set the initial values to yield a position starting at other values.
- It is not possible in 20-sim to use vector elements with mixed units. Therefore element number 4 to 6 will be displayed with units [m] although it really is [rad]!

ThreeDSpringDamper

Library

Iconic Diagrams\Mechanical\Translation\3DSmallAngles

Use

Domains: Continuous. **Size**: 1-D/6-D. **Kind**: Iconic Diagrams (Translation,Rotation).

Description

This model is the equivalent of the 1 degree of freedom model `SpringDamper.emx`. It model represents an ideal spring-damper. The model has a preferred force out causality. The corresponding constitutive equations then contain integrations. The element can also have the non-preferred velocity out causality. The constitutive equations then contain derivatives. The spring-damper model has separate high and low ports. The equations are

```
P.F = P_high.F = P_low.F
P.v = P_high.v - P_low.v
```

Force out causality (preferred):

```
X = int(P.v);
P.F = S*X + D*P.v;
```

Velocity out causality:

P.v = ddt(X);
X = S-1(P.F - D*P.v);*

Interface

Ports	**Description**
P_high[6], P_low[6]	Both terminals of a port with 6 degrees of freedom.
Causality	
preferred force out P	
Variables	
X	Vector with 6 spring extension values [m].
Parameters	
S[6]	Stiffness vector [N/m]
D[6]	Damping vector [N.s/m]
Initial Values	
X_initial[6]	Vector with 6 initial extension values of the spring [m].

Note

It is not possible in 20-sim to use vector elements with mixed units. Therefore element number 4 to 6 will be displayed with units [m/s] and [N] although it really is [rad/s] and [Nm] etc.!

ThreeDZeroForce

Library

Iconic Diagrams\Mechanical\Translation\3DSmallAngles

Use

Domains: Continuous. **Size**: 6-D. **Kind**: Iconic Diagrams (Translation,Rotation).

Description

This model can be used to connect any open end of another 3D model that is not connected to the fixed world. It generates a three degree of freedom zero force while the velocity is free.

Interface

Ports	**Description**
P	Port with 6 degrees of freedom.
Causality	
fixed force out P	

Note

It is not possible in 20-sim to use vector elements with mixed units. Therefore element number 4 to 6 will be displayed with units [m/s] and [N] although it really is [rad/s] and [Nm] etc.!

Actuators

AccelerationActuator-Relative

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents an ideal actuator. An acceleration input signal is integrated to a velocity difference between its two terminals:

```
p_high.v = p_low.v + int(a,v_initial);
p_low.F = p_high.F = indifferent;
```

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Translation port p.
Causality	
fixed velocity out	
Input	
a	Acceleration [m/s^2].
Parameters	
v_initial	Initial velocity output of the integration [m/s].

AccelerationActuator

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents an ideal actuator. An acceleration input signal is integrated to a velocity at the translation port. The actuator is mounted to the fixed world:

p.v = int(a,v_initial);
p.F = indifferent;

Interface

Ports	**Description**
p	Translation port.
Causality	
fixed velocity out	
Input	
a	Acceleration [m/s^2].
Parameters	
v_initial	Initial velocity at the port [m/s].

CMABender

Library

Iconic Diagrams\Electric\Actuators
Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents a piezo actuator. The actuator translates a voltage difference at port *p1* to a mechanical position difference between the base at port *p2* and the end effector at port *p3*.

Although the underlying equations of this model are equal to the equations of the `CMAStrecher.emx model`, the default parameter values are typical for bending.

Interface

Ports	**Description**
p	Translation port.
Causality	
fixed force out	
Input	
C	Capacitance [F]
KF	Voltage to force conversion factor [N/V]
m	Equivalent mass [kg]
k	Stiffness [N/m]
B	Relative damping ratio []

KB	Force to voltage conversion factor [V/N]

CMAStretcher

Library

Iconic Diagrams\Electric\Actuators
Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents a piezo actuator. The actuator translates a voltage difference at port *p1* to a mechanical position difference between the base at port *p2* and the end effector at port *p3*.

Although the underlying equations of this model are equal to the equations of the CMABender.emx model, the default parameter values are typical for stretching.

Interface

Ports	**Description**
p	Translation port.
Causality	
fixed force out	
Input	
C	Capacitance [F]
KF	Voltage to force conversion factor [N/V]
m	Equivalent mass [kg]
k	Stiffness [N/m]
B	Relative damping ratio []
KB	Force to voltage conversion factor [V/N]

Force-Relative

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents an ideal actuator. The actuator applies a force between its two terminals. This force can be set to a constant value *F*, the velocity is indifferent.

p_high.F = p_low.F = F

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Translation port p.
Causality	
fixed force out	
Parameters	
F	Force [N]

Force

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This model represents an ideal actuator. The actuator is mounted to the fixed world and applies a force. The force can be set to a certain constant value, the velocity is indifferent.

p.F = F;

Interface

Ports	**Description**
p	Translation port.
Causality	
fixed force out	
Parameters	

F	Force [N].

ForceActuator-Relative

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents an ideal actuator. The actuator applies a force between its two terminals. This force can be set to a (fluctuating) value given by the input signal *F*, the velocity is indifferent.

$$p\_high.F = p\_low.F = F$$

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Translation port p.
Causality	
fixed force out	
Input	
F	Force [N]

ForceActuator

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents an ideal actuator. The actuator is mounted to the fixed world and applies a force. The force can be set to a (fluctuating) value given by the input signal *F*, the velocity is indifferent.

p.F = F;

Interface

Ports	**Description**
p	Translation port.
Causality	
fixed force out	
Input	
F	Force [N].

PositionActuator-Relative

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

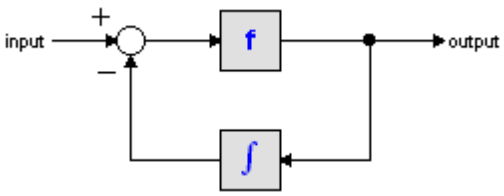
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents an ideal actuator. A position input signal is differentiated by a state variable filter to a velocity difference between its two terminals:

p_high.v = p_low.v + dx/dt;
p_low.F = p_high.F = indifferent;

Differentiation is performed by a state variable filter:



The S-domain function of this filter is equal to:

$$output = \frac{s}{\frac{s}{f} + 1} \cdot input$$

where *f* is the cut-off frequency. For very high values of *f*, the output becomes the pure derivative of the input. High values of *f*, however, increase simulations times. A good trade-off is a starting value of 1e5.

Interface

Ports	Description
p_high, p_low	Both terminals of the Translation port p.
Causality	
fixed velocity out	
Input	
x	Position [m].
Parameters	
f	Cut-off frequency of the differentiation [Hz].
v_initial	Initial velocity output of the differentiation [m/s].

PositionActuator

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

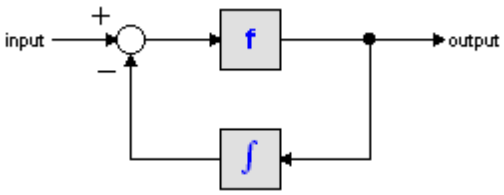
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents an ideal actuator. A position input signal is differentiated by a state variable filter to a velocity at the translation port. The actuator is mounted to the fixed world:

p.v = dx/dt;
p.F = indifferent;

Differentiation is performed by a state variable filter:



The S-domain function of this filter is equal to:

$$output = \frac{s}{\frac{s}{f}+1} \cdot input$$

where *f* is the cut-off frequency. For very high values of *f*, the output becomes the pure derivative of the input. High values of *f*, however, increase simulations times. A good trade-off is a starting value of 1e5.

Interface

Ports	**Description**
p	Translation port.
Causality	
fixed velocity out	
Input	
x	Position [m].
Parameters	
f	Cut-off frequency of the differentiation [Hz].
v_initial	Initial velocity at the port [m/s].

ServoMotor

Library

Iconic Diagrams\Mechanical\Rotation\Actuators
Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric).

Description

This is a masked model which opens the Servo Motor Editor when edited. The servo Motor Editor is a tool that can shows the torque speed plots numerous permanent magnet motors and can generate a dynamic model from any motor that you select. The following motor types are supported:

1. Brush DC
2. Brushless DC (trapezoidal EMC and square wave currents)
3. AC synchronous (sinusoidal EMC and sinusoidal currents)
4. AC synchronous linear (sinusoidal EMC and sinusoidal currents)

Interface

Depending on the type of motor that you have selected, the interface can vary:

DC Brush

Ports	Description
p	Rotation port.
Causality	
fixed rotational velocity out	
Input	
i	The input current [A]

DC Brushless

Ports	Description
p	Rotation port.
Causality	
fixed rotational velocity out	
Input	
i	The maximum input current [A]

AC Synchronous

Ports	Description
p	Rotation port.
Causality	
fixed rotational velocity out	
Input	
i_rms	The rms phase current [A]

AC Synchronous Linear

Ports	Description
p	Translation port.
Causality	
fixed velocity out	
Input	

i_rms	The rms phase current [A]

For more information on the parameters an variables of this model is referred to the Mechatronic Toolbox.

Velocity-Relative

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This model represents an ideal actuator. The actuator applies a velocity difference between its two terminals. This velocity can be set to a constant value v, the force is indifferent.

$$p\_low.v = p\_high.v + v$$

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Translation port p.
Causality	
fixed velocity out	
Parameter	
v	Velocity [m/s].

Velocity

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This model represents an ideal actuator. The actuator is mounted to the fixed world and applies a velocity. This velocity can be set to a certain constant value, the force is indifferent.

p.v = v;

Interface

Ports	**Description**
p	Translation port
Causality	
fixed velocity out	
Parameters	
v	velocity [m/s]

VelocityActuator-Relative

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents an ideal actuator. The actuator applies a velocity difference between its two terminals. This velocity can be set to a (fluctuating) value given by the input signal *v*, the force is indifferent.

p_high.v = p_low.v + v

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Translation port p.
Causality	
fixed velocity out	
Input	
v	Velocity [m/s].

VelocityActuator

Library

Iconic Diagrams\Mechanical\Translation\Actuators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model represents an ideal actuator. The actuator is mounted to the fixed world and applies a velocity. This velocity can be set to a (fluctuating) value given by the input signal *v*, the force is indifferent.

p.v = v;

Interface

Ports	**Description**
p	Translation port.
Causality	
fixed velocity out	
Input	
v	Velocity [m/s].

Components

Backlash

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

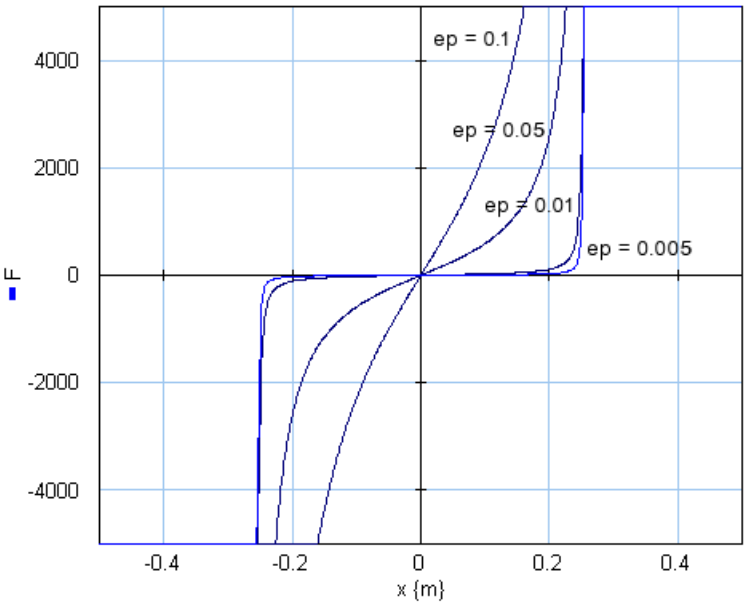
This model represents backlash by a spring damper system. Inside an outside the play, spring and damping can be set separately. Discontinuities are avoided by adding a round off. The model can has a force out causality. The port *p* of this model has separate high and low terminals. The equations are:

p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v

$$p.F = 0.5 \cdot k_2 \cdot \left(x - \sqrt{\left(-x - \frac{s}{2}\right) + (ep \cdot s)^2} + x + \sqrt{\left(x - \frac{s}{2}\right) + (ep \cdot s)^2} \right) + k_1 \cdot x + d_{1or\,2} \cdot p.v$$

with x the position within the play. Real backlash behavior is obtained by choosing low stiffness and damping values inside the play and choosing high stiffness and damping values outside the play.

The parameter *ep* determines the smoothness of the force curve that is obtained. A larger value (> 0.01) makes the force change gradually when the position reaches the play boundaries. A smaller value (< 0.001) makes the force change abruptly. This is shown in the figure below. A good starting value for *ep* is 1e-4.



Interface

Ports	**Description**
p_high, p_low	Both terminals of port p (Translation).
Causality	
fixed force out	
Parameters	
s	Interval of the play [m]
c1	Stiffness in the play [N/m]
c2	Stiffness outside the play [N/m]

d1	Damping inside the play [Ns/m]
d2	Damping outside the play [Ns/m]
ep	Relative round off (1e-6 -> sharp edges, 1e-2 -> smoother)
Initial Values	
x_initial	Initial position in the play [m], -s/2 < x_initial < s/2

Collision-Relative

Library

Iconic Diagrams\Mechanical\Translation\Components

Implementations

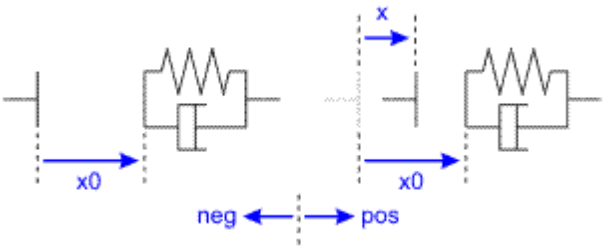
Right
Left

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description - Right

This model represents an collision of an object with another object. It can be used to indicate a possible collision to the right of an object.



The collision force is modeled by a spring and damper:

p.F = if x > x0 then k(x-x0) + d*limit(p.v,0,1e20) else 0 end;*

with a stiffness *k* and damping *d*. The *limit* function is used to prevent the damper force to become negative. The initial position of the spring is indicated by *x0* (see the figure above). Note that the positive direction is to the right. If *x* is larger that *x0*, the two sides have collided and are in contact. The collision model has separate high and low ports. The equations are

p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v

Interface - Right

Ports	Description
p_high	Two ports of the collision model.

p_low

Causality

fixed force out

Variables

x	extension of the spring [m]

Parameters

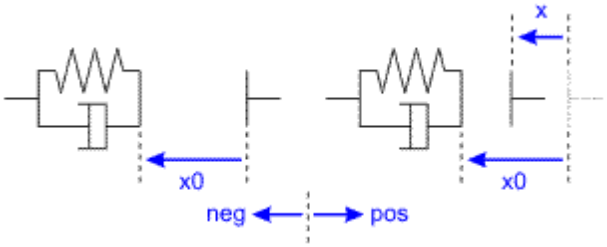
x0	unstretched spring position
k	stiffness [N/m]
d	damping [N.s/m]

Initial Values

x_initial	The initial extension of the spring [m].

Description - Left

This model represents an collision of an object with another object. It can be used to indicate a possible collision to the left of an object.



The collision force is modeled by a spring and damper. Note that the positive direction is to the right, so the initial position of the spring *x0* has a negative value in the figure above. If *x* is smaller that *x0*, the two sides have collided and are in contact.

Interface - Left

Ports	Description
p_high	Two ports of the collision model.
p_low	

Causality

fixed force out

Variables

x	extension of the spring [m]

Parameters

x0	unstretched spring position

k	stiffness [N/m]
d	damping [N.s/m]

Initial Values

x_initial	The initial extension of the spring [m].

Collision

Library

Iconic Diagrams\Mechanical\Translation\Components

Implementations

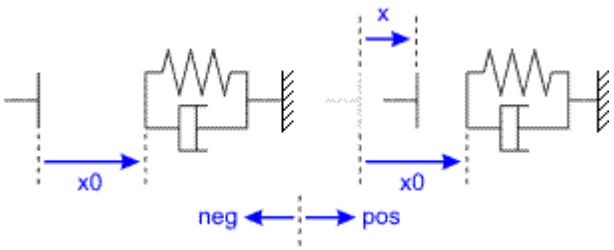
Right
Left

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description - Right

This model represents an collision of an object with the ground.



The collision force is modeled by a spring and damper:

p.F = if x > x0 then k(x-x0) + d*limit(p.v,0,1e20) else 0 end;*

with a stiffness *k* and damping *d*. The *limit* function is used to prevent the damper force to become negative. The initial position of the spring is indicated by *x0* (see the figure above). Note that the positive direction is to the right. If *x* is larger that *x0*, the two sides have collided and are in contact. The collision model has separate high and low ports. The equations are

p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v

Interface - Right

Ports	Description
p_high	Two ports of the collision model.
p_low	

Causality

fixed force out

Variables

x	extension of the spring [m]

Parameters

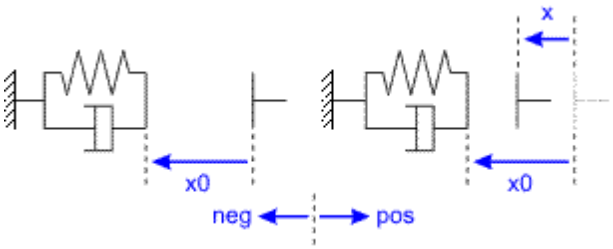
x0	unstretched spring position
k	stiffness [N/m]
d	damping [N.s/m]

Initial Values

x_initial	The initial extension of the spring [m].

Description - Left

This model represents an collision of an object with the ground.



The collision force is modeled by a spring and damper. Note that the positive direction is to the right, so the initial position of the spring *x0* has a negative value in the figure above. If *x* is smaller that *x0*, the two sides have collided and are in contact.

Interface - Left

Ports	**Description**
p_high	Two ports of the collision model.
p_low	

Causality

fixed force out

Variables

x	extension of the spring [m]

Parameters

x0	unstretched spring position
k	stiffness [N/m]
d	damping [N.s/m]

Initial Values

x_initial	The initial extension of the spring [m].

Damper

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This model represents a linear damper. It can have an force out as well as an velocity out causality. In the last case the constitutive equation, as shown below, is simply inverted. The port *p* of the damper model has separate high and low terminals. The equations are:

p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v

Force out causality:

*p.F = d * p.v;*

Velocity velocity out causality:

p.v = p.F / d;

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Translation port p.

Causality

indifferent

Parameters

d	damping [Ns/m]

FixedWorld

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This model represents the fixed world (velocity = 0). The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. which gives the constitutive equations:

p1.v = p2.v = .. = pn.v = 0;
p1.F = free; p2.F = free; ..; pn.F = free;

Interface

Ports	Description
p [any]	Any number of connections can be made (Translation).

Causality

Fixed velocity out	angular	All ports have a fixed velocity out causality.

Friction

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Implementations

C
V
CV
SCVS
LuGre

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Introduction

These models represents friction with the fixed world. The amount of friction depends on the normal force that is applied and the friction function that is used. The normal force is given by the input signal *Fn.*

The models have only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. 20-sim will automatically create equations such that the resulting force *p.F* is equal to the sum of the forces of all connected ports p1 .. pn. The velocities of all connected ports are equal to *p.v*.

p.F = sum(p1.F, p2.F, ...)
p.v = p1.v = p2.v = ...

Due to the use of normal force, the friction models all have a fixed force out causality. The constitutive equations are therefore described as:

*p.F = Fn * f(p.v);*

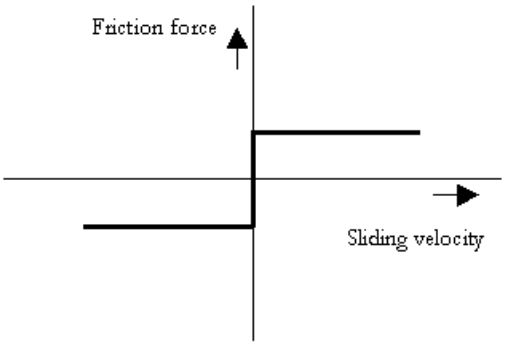
with *f* the friction function.

Description - C

This model represents friction with the fixed world described as coulomb friction:

*p.F = Fn*mu_c*tanh(slope*p.v);*

Fn: normal force (given by the input signal Fn)
mu_c: the coulomb friction coefficient
slope: the steepness of the coulomb friction curve.



Interface - C

Ports	**Description**
p[any]	Any number of connections can be made (Translation).
Causality	
Fixed force out	
Input	

Fn	Normal force [N]

Parameters

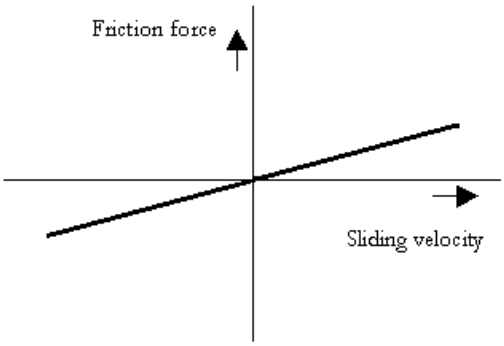
mu_c	Coulomb friction coefficient []
slope	Steepness of Coulomb friction curve [s/m]

Description - V

This model represents friction with the fixed world described as viscous friction:

*p.F = Fn*mu_v*p.v;*

Fn: normal force (given by the input signal Fn)
mu_v: the viscous friction coefficient



Interface - V

Ports	**Description**
p[any]	Any number of connections can be made (Translation).

Causality

Fixed force out

Input

Fn	Normal force [N]

Parameters

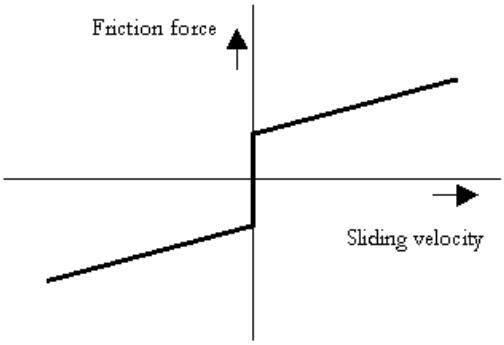
mu_v	Viscous friction coefficient [s/m]

Description - CV

This model represents friction with the fixed world described as coulomb plus viscous friction:

p.F = Fn(mu_c*tanh(slope*p.v) + mu_v*p.v);*

Fn: normal force (given by the input signal Fn)
mu_v: the viscous friction coefficient
mu_c: the coulomb friction coefficient
slope: the steepness of the coulomb friction curve.



Interface - CV

Ports	**Description**
p[any]	Any number of connections can be made (Translation)
Causality	
Fixed force out	
Input	
Fn	Normal force [N]
Parameters	
mu_v	Viscous friction coefficient [s/m]
mu_cv	Coulomb friction coefficient []
slope	Steepness of Coulomb friction curve [s/m]

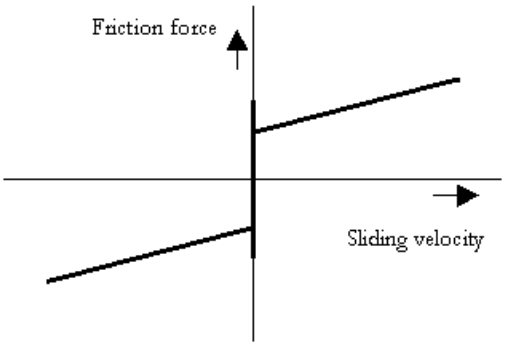
Description - SCVS

This model represents friction with the fixed world described as static plus coulomb plus viscous plus Stribeck friction:

```
p.F = Fn *
(( mu_c + (mu_st*abs(tanh( slope*p.v )) - mu_c)
* exp( -((p.v / v_st)^2 )) ) * sign(p.v)
+ mu_v * p.v);
```

Fn: normal force (given by the input signal Fn)
mu_s: the static friction coefficient
mu_v: the viscous friction coefficient
mu_c: the coulomb friction coefficient

slope: the steepness of the coulomb and static friction curve.
v_st: the characteristic Stribeck velocity.



Interface - SCVS

Ports	**Description**
p[any]	Any number of connections can be made (Translation).
Causality	
Fixed force out	
Input	
Fn	Normal force [N]
Parameters	
mu_s	Static friction coefficient []
mu_v	Viscous friction coefficient [s/m]
mu_cv	Coulomb friction coefficient []
slope	Steepness of Coulomb friction curve [s/m]
v_st	Characteristic Stribeck velocity [m/s]

Description - LuGre

This model represents friction with the fixed world described by the LuGre friction model:

```
p.F = FN*f_lg(p.v);
```

Fn: normal force (given by the input signal Fn)
f_lg: the LuGre friction model

Interface - LuGre

Ports	**Description**
p[any]	Any number of connections can be made (Translation).
Causality	

Fixed force out

Input

Fn	Normal force [N]

Parameters

mu_c	Coulomb friction coefficient []
mu_s	Static friction coefficient []
mu_v	Viscous friction coefficient [s/m]
v_st	Characteristic Stribeck velocity [m/s]
mu_k	Stiffness coefficient at zero speed []

Friction-Relative

Library

Iconic Diagrams\Mechanical\Translation\Components

Implementations

C
V
CV
SCVS
LuGre

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Introduction

These models represents friction relative to other objects. The amount of friction depends on the normal force that is applied and the friction function that is used. The normal force is given by the input signal *Fn.*

The port *p* of the friction models have separate high and low terminals. The equations are:

p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v

Due to the use of normal force, the friction models all have a fixed torque out causality. The constitutive equations are therefore described as:

*p.F = Fn * f(p.v);*

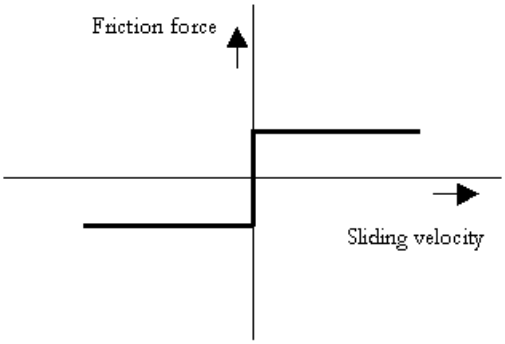
with f the friction function.

Description -C

This model represents friction between two terminals described as coulomb friction:

*p.F = Fn*mu_c*tanh(slope*p.v);*

Fn: normal force (given by the input signal Fn)
mu_c: the coulomb friction coefficient
slope: the steepness of the coulomb friction curve.



Interface - C

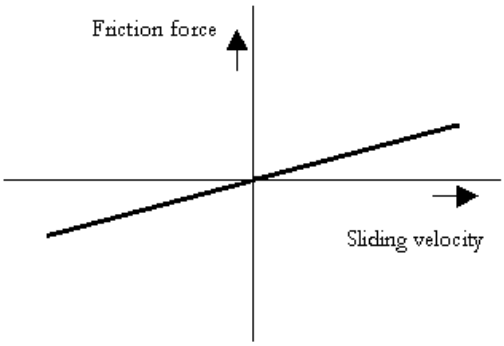
Ports	**Description**
p_high, p_low	Both terminals of port p (Translation).
Causality	
Fixed force out	
Input	
Fn	Normal force [N]
Parameters	
mu_c	Coulomb friction coefficient []
slope	Steepness of Coulomb friction curve [s/m]

Description - V

This model represents friction between two terminals described as viscous friction:

*p.F = Fn*mu_v*p.v;*

Fn: normal force (given by the input signal Fn)
mu_v: the viscous friction coefficient



Interface -V

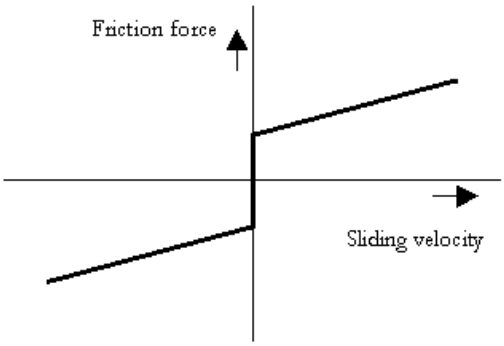
Ports	**Description**
p_high, p_low	Both terminals of port p (Translation).
Causality	
Fixed force out	
Input	
Fn	Normal force [N]
Parameters	
mu_v	Viscous friction coefficient [s/m]

Description - CV

This model represents friction between two terminals described as coulomb plus viscous friction:

p.F = Fn(mu_c*tanh(slope*p.v) + mu_v*p.v);*

Fn: normal force (given by the input signal Fn)
mu_v: the viscous friction coefficient
mu_c: the coulomb friction coefficient
slope: the steepness of the coulomb friction curve.



Interface - CV

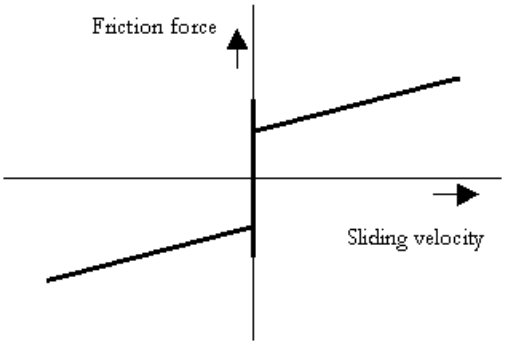
Ports	Description
p_high, p_low	Both terminals of port p (Translation).
Causality	
Fixed force out	
Input	
Fn	Normal force [N]
Parameters	
mu_v	Viscous friction coefficient [s/m]
mu_cv	Coulomb friction coefficient []
slope	Steepness of Coulomb friction curve [s/m]

Description - SCVS

This model represents friction between two terminals described as static plus coulomb plus viscous plus Stribeck friction:

```
p.F = Fn *
(( mu_c + (mu_st*abs(tanh( slope*p.v )) - mu_c)
* exp( -((p.v / v_st)^2 )) ) * sign(p.v)
+ mu_v * p.v);
```

Fn: normal force (given by the input signal Fn)
mu_s: the static friction coefficient
mu_v: the viscous friction coefficient
mu_c: the coulomb friction coefficient
slope: the steepness of the coulomb and static friction curve.
v_st: the characteristic Stribeck velocity.



Interface - SCVS

Ports	**Description**
p_high, p_low	Both terminals of port p (Translation).
Causality	
Fixed force out	
Input	
Fn	Normal force [N]
Parameters	
mu_s	Static friction coefficient []
mu_v	Viscous friction coefficient [s/m]
mu_cv	Coulomb friction coefficient []
slope	Steepness of Coulomb friction curve [s/m]
v_st	Characteristic Stribeck velocity [m/s]

Description - LuGre

This model represents friction between two terminals described by the LuGre friction model:

```
p.F = FN*f_lg(p.v);
```

Fn: normal force (given by the input signal Fn)
f_lg: the LuGre friction model

Interface - LuGre

Ports	**Description**
p_high, p_low	Both terminals of port p (Translation).
Causality	

Fixed force out

Input

Fn	Normal force [N]

Parameters

mu_c	Coulomb friction coefficient []
mu_s	Static friction coefficient []
mu_v	Viscous friction coefficient [s/m]
v_st	Characteristic Stribeck velocity [m/s]
mu_k	Stiffness coefficient at zero speed []

FrictionSimple

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Implementations

C
V
CV
SCVS
LuGre

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Introduction

These models represents friction with the fixed world. The amount of friction does not depend on the normal force but determined by parameters directly. The models have only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. 20-sim will automatically create equations such that the resulting force *p.F* is equal to the sum of the forces of all connected ports p1 .. pn. The velocities of all connected ports are equal to *p.v*.

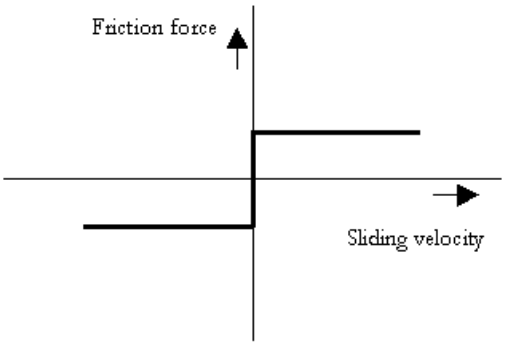
```
p.F = sum(p1.F, p2.F, ....)
p.v = p1.v = p2.v = ....
```

Description - C

This model represents friction with the fixed world described as coulomb friction:

```
p.F = Fc*tanh(slope*p.v);
```

Fc: the coulomb friction
slope: the steepness of the coulomb and static friction curve.



Interface - C

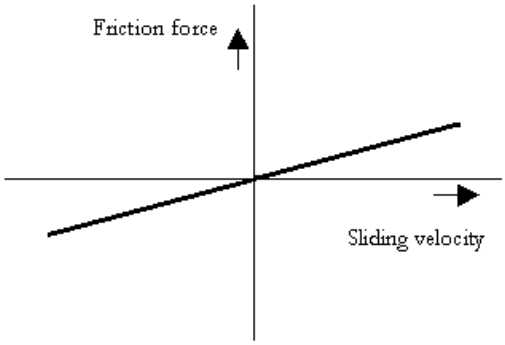
Ports	**Description**
p[any]	Any number of connections can be made (Translation).
Causality	
Fixed force out	
Parameters	
Fc	Coulomb friction [N]
slope	Steepness of Coulomb friction curve [s/m]

Description - V

This model represents friction with the fixed world described as viscous friction:

*p.F = d*p.v;*

d: the viscous friction or damping



Interface - V

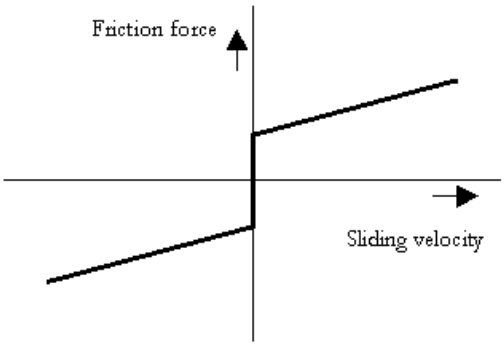
Ports	Description
p[any]	Any number of connections can be made (Translation).
Causality	
Fixed force out	
Parameters	
d	Viscous friction or damping [N.s/m]

Description - CV

This model represents friction with the fixed world described as coulomb plus viscous friction:

*p.F = Fc*tanh(slope*p.v) + d*p.v;*

d: the viscous friction or damping
Fc: the coulomb friction
slope: the steepness of the coulomb and static friction curve.



Interface - CV

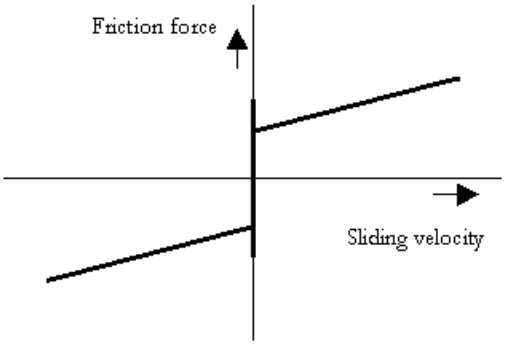
Ports	Description
p[any]	Any number of connections can be made (Translation)
Causality	
Fixed force out	
Parameters	
Fc	Coulomb friction [N]
d	Viscous friction or damping [N.s/m]
slope	Steepness of Coulomb friction curve [s/m]

Description - SCVS

This model represents friction with the fixed world described as static plus coulomb plus viscous plus Stribeck friction:

```
p.F = (( Fc + (Fst*abs(tanh( slope*p.v )) - Fc) * exp( -((p.v / v_st)^2 )) ) *
sign(p.v) + d * p.v);
```

Fst: the static friction
d: the viscous friction or damping
Fc: the coulomb friction
slope: the steepness of the coulomb and static friction curve.
v_st: the characteristic Stribeck velocity.



Interface - SCVS

Ports	**Description**
p[any]	Any number of connections can be made (Translation).
Causality	
Fixed force out	
Parameters	
Fc	Coulomb friction [N]
Fst	Static friction [N]
d	Viscous friction or damping [N.s/m]
v_st	Characteristic Stribeck velocity [m/s]
slope	Steepness of Coulomb friction curve [s/m]

Description - LuGre

This model represents friction with the fixed world described by the LuGre friction model:

```
p.F = f_lg(p.v);
```

f_lg: the LuGre friction model

Interface - LuGre

Ports	Description
p[any]	Any number of connections can be made (Translation).
Causality	
Fixed force out	
Parameters	
Fc	Coulomb friction [N]
Fst	Static friction [N]
d	Viscous friction or damping [N.s/m]
v_st	Characteristic Stribeck velocity [m/s]
k	Stiffness at zero speed [N/m]

FrictionSimple-Relative

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Implementations

C
V
CV
SCVS
LuGre

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Introduction

These models represents friction with the fixed world. The amount of friction does not depend on the normal force but determined by parameters directly. The models have only one initial port p defined. The port *p* of the friction models have separate high and low terminals. The equations are:

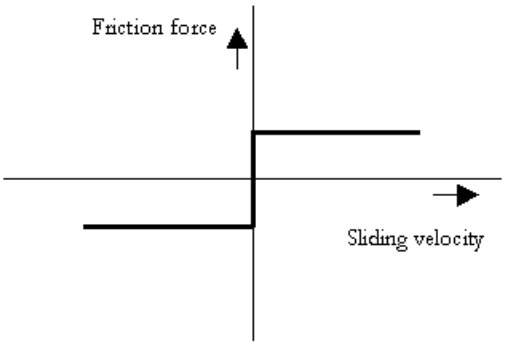
p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v

Description - C

This model represents friction with the fixed world described as coulomb friction:

*p.F = Fc*tanh(slope*p.v);*

Fc: the coulomb friction
slope: the steepness of the coulomb and static friction curve.



Interface - C

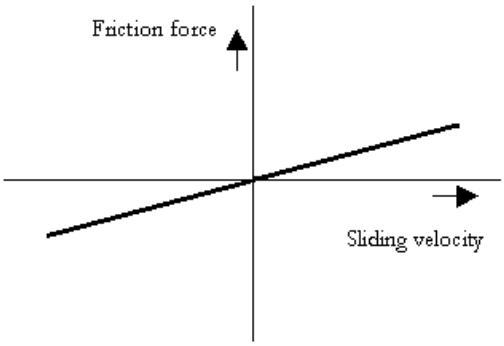
Ports	**Description**
p[any]	Any number of connections can be made (Translation).
Causality	
Fixed force out	
Parameters	
Fc	Coulomb friction [N]
slope	Steepness of Coulomb friction curve [s/m]

Description - V

This model represents friction with the fixed world described as viscous friction:

*p.F = d*p.v;*

d: the viscous friction or damping



Interface - V

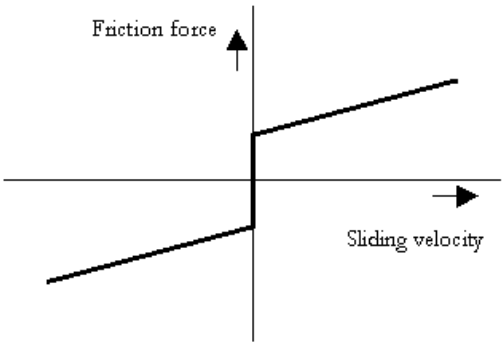
Ports	Description
p[any]	Any number of connections can be made (Translation).
Causality	
Fixed force out	
Parameters	
d	Viscous friction or damping [N.s/m]

Description - CV

This model represents friction with the fixed world described as coulomb plus viscous friction:

*p.F = Fc*tanh(slope*p.v) + d*p.v;*

d: the viscous friction or damping
Fc: the coulomb friction
slope: the steepness of the coulomb and static friction curve.



Interface - CV

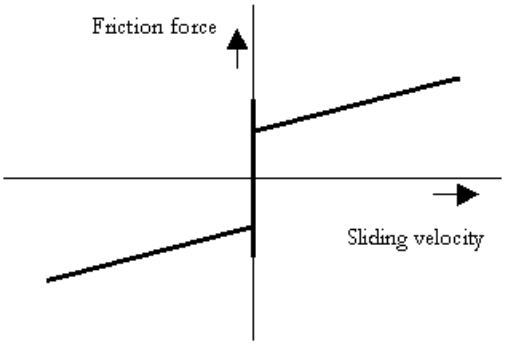
Ports	Description
p[any]	Any number of connections can be made (Translation)
Causality	
Fixed force out	
Parameters	
Fc	Coulomb friction [N]
d	Viscous friction or damping [N.s/m]
slope	Steepness of Coulomb friction curve [s/m]

Description - SCVS

This model represents friction with the fixed world described as static plus coulomb plus viscous plus Stribeck friction:

```
p.F = (( Fc + (Fst*abs(tanh( slope*p.v )) - Fc) * exp( -((p.v / v_st)^2 )) ) *
sign(p.v) + d * p.v);
```

Fst: the static friction
d: the viscous friction or damping
Fc: the coulomb friction
slope: the steepness of the coulomb and static friction curve.
v_st: the characteristic Stribeck velocity.



Interface - SCVS

Ports	**Description**
p[any]	Any number of connections can be made (Translation).
Causality	
Fixed force out	
Parameters	
Fc	Coulomb friction [N]
Fst	Static friction [N]
d	Viscous friction or damping [N.s/m]
v_st	Characteristic Stribeck velocity [m/s]
slope	Steepness of Coulomb friction curve [s/m]

Description - LuGre

This model represents friction with the fixed world described by the LuGre friction model:

```
p.F = f_lg(p.v);
```

f_lg: the LuGre friction model

Interface - LuGre

Ports	Description
p[any]	Any number of connections can be made (Translation).

Causality

Fixed force out

Parameters

Fc	Coulomb friction [N]
Fst	Static friction [N]
d	Viscous friction or damping [N.s/m]
v_st	Characteristic Stribeck velocity [m/s]
k	Stiffness at zero speed [N/m]

Mass

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description-Default

This model represents an ideal mass (no gravity). The element has a preferred velocity out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred force out causality. The constitutive equations then contain a derivation. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. 20-sim will automatically create equations such that the resulting force p.F is equal to the sum of the forces of all connected ports p1 .. pn. The velocities of all connected ports are equal to *p.v*.

p.F = sum(p1.F, p2.F, ...)
p.v = p1.v = p2.v = ...

velocity out causality (preferred):

a = p.F/m;
p.v = int(a);
x = int(p.omega);

force out causality:

```
a = ddt(p.v);
p.T = m*a;
x = int(p.v);
```

Description-Gravity

This model represents an ideal mass with gravity. It is equal to the default model with a gravitational acceleration *g* added:

```
a = p.F/m - g;
p.v = int(a);
x = int(p.omega);
```

The gravitational acceleration *g* acts in the negative direction. E.g. a free fall will give the model a negative acceleration, negative velocity and negative position.

Interface

Ports	**Description**
p[any]	Any number of connections can be made (Translation).
Causality	
preferred velocity out	An torque out causality results in a derivative constitutive equation.
Variables	
x	position [m]
a	acceleration [m/s^2]
p	impulse [Ns]
Parameters	
m	mass [kg]
g	acceleration of gravity [m/s^2]
Initial Values	
p.v_initial	The initial velocity of the mass.
x_initial	The initial position of the mass.

Modal

Library

Iconic Diagrams\Mechanical\Translation\Components

Implementations

Stiffness
Frequency

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description - Stiffness

This model represents a spring-damper-mass system that can be used for modal analysis. The springdamper is equivalent tot the `SpringDamper-Stiffness.emx` model. The mass is equivalent to the `Mass.emx` model. The model has one output that can be connected to the modal-summer.emx model.

Interface - Stiffness

Ports	Description
p	Translation port.

Causality

preferred velocity out

Parameters

k	Stiffness [N/m]
b	Relative damping []
m	Equivalent mass [kg]

Description - Frequency

This model represents a spring-damper-mass system that can be used for modal analysis. The springdamper is equivalent tot the *SpringDamper-Frequency* model. The mass is equivalent to the *Mass* model. The model has one output that can be connected to the *Modal-Summer* model.

Interface - Frequency

Ports	Description
p	Translation port.

Causality

preferred velocity out

Parameters

b	Relative damping []
f	Resonance frequency [Hz]

m	Equivalent mass [kg]

ModalSummer

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This model combines the results from all modal.emx models to one resulting output. The model has 8 input ports and one output port that shows the combined result. If more or less input ports are needed, you have to adapt the model (Choose *Go Down* and alter the internal description).

Interface

Ports	**Description**
p1 .. p8	Input ports for *modal.emx* models (Translation).
p_out	Port with the modal result (Translation)

Causality

preferred force out p1..p8

preferred velocity out p_out

Parameters

A1 .. A8	Gains for each input port.

Node

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This node model represents a structural connection between two or more parts, where the velocity of all connected parts is equal. The model has only one initial port p defined. Because any number of connections can be made, successive ports are named p1, p2, p3 etc.

Interface

Ports	Description
p [any]	Any number of connections can be made (Translation).

ShockAbsorber

Library

Iconic Diagrams\Mechanical\Translation\Components

Implementations

Right
Left

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Introduction

This model represents a shock absorber with an open end. This type of shock absorbers is used to protect equipment from large impacts when reaching the end of a line.

The damper has an internal piston which can travel a certain length (i.e. the stroke *l*) until it collides with the end cap. The shock absorber is modeled by a damper for the active damping part and a spring damper to model the impact with the end cap. The position of the damper is indicated by the parameter *x0.* Two implementations are available: one with the impact at the right and one with the impact at the left.

The damper is modeled by viscous damping, where the damping force is proportional with the velocity:

*p.F = d * v;*

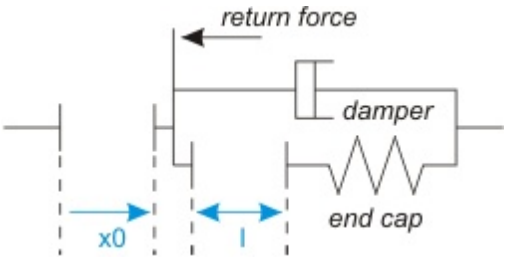
After a stroke of length *l*, the end caps are reached which a are modeled by a spring damper:

*p.F = k * x;*

To get the damper back in its original position a return force is added.

Description - Right

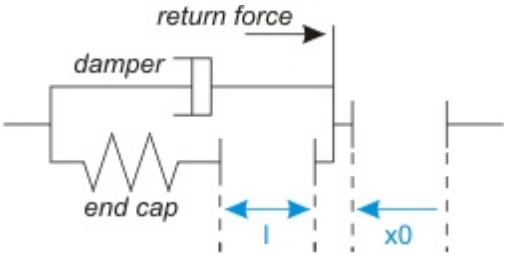
The *ShockAbsorber-Right* models a shock absorber with impact at the right of the system.



Shock absorber model with impact at the right.

Description - Right

The *ShockAbsorber-Left* models a shock absorber with impact at the left of the system.



Shock absorber model with impact at the left.

Interface

Ports	**Description**
p_high	Two ports of the shock absorber model.
p_low	

Causality

fixed force out

Parameters

k	stiffness at the end cap [N/m]
d	damping [N.s/m]
l	shock absorber stroke [m]
x0	position of the damper [m]

Fr	return force [N]

SkyHookDamper

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

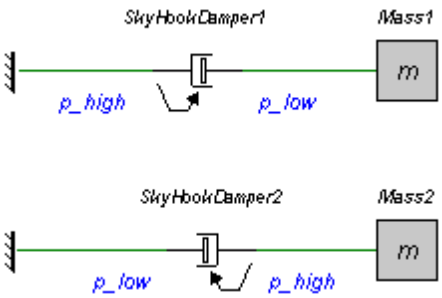
Description

This model represents a skyhook damper. It can have an force out as well as an velocity out causality. In the last case the constitutive equation, as shown below, is simply inverted. The model has a fixed force out causality. The equations are:

Force out causality:

```
p_high.F = d * p_high.v;
p_low.F = p_high.F
```

As the equations point out, the force of the skyhook damper only depends on the velocity at the port *p_high*. The position of *p_high* is indicated by an arrow, as shown in the figure below.



In this figure two models are shown. In the top model, the damping force of the skyhook damper is zero because the velocity at p_high is zero. Consequently the mass is free to move.

In the lower model, the damping force is proportional to the velocity of the mass. Here the skyhook damper acts as a normal damper.

Interface

Ports	Description
p_high, p_low	Both Translation ports of the skyhook damper.

Causality

fixed force out

Parameters

d	damping [Ns/m]

Spring

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This model represents an ideal translational spring. The element has a preferred force out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred velocity out causality. The constitutive equations then contain a derivation. The spring model has separate high and low ports. The equations are

```
p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v
```

Force out causality (preferred):

```
x = int(p.v);
p.F = k * x;
```

Velocity out causality:

```
p.v = ddt(x);
x = p.F/k;
```

Interface

Ports	**Description**
p_high	Two ports of the spring (Translation).
p_low	
Causality	
preferred force out	An velocity out causality results in a derivative constitutive equation.
Variables	
x	extension of the spring [m]
Parameters	

k	Stiffness [N/m]

Initial Values

x_initial	The initial extension of the spring [m].

SpringDamper

Library

Iconic Diagrams\Mechanical\Translation\Components

Implementations

Default
Stiffness
Frequency

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description - Default

This model represents an ideal translational spring with damper. The element has a preferred force out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred velocity out causality. The constitutive equations then contain a derivation. The spring-damper model has separate high and low ports. The equations are

p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v

Force out causality (preferred):

```
x = int(p.v);
p.F = k*x + d*p.v;
```

Velocity out causality:

```
p.v = ddt(x);
x = (p.F - d*p.v)/k;
```

Interface - Default

Ports	Description
p_high	Two ports of the spring (Translation).
p_low	

Causality

preferred force out

Variables

x	extension of the spring [m]

Parameters

k	Stiffness [N/m]
d	damping [N.s/m]

Initial Values

x_initial	The initial extension of the spring [m].

Description - Stiffness

This model represents another implementation of the ideal translational spring with damper. The damping value (*d*) is calculated on the basis of a known stiffness (*k*), relative damping (*b*) and mass reference (*m*). The mass is only used to compute the damping (no actual mass is used in this component).

The element has a preferred force out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred velocity out causality. The constitutive equations then contain a derivation. The spring-damper model has separate high and low ports. The equations are

```
p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v
```

Force out causality (preferred):

```
x = int(p.v);
p.F = k * x + d*p.v;
d = 2*b*sqrt(k*m);
```

Velocity out causality:

```
p.v = ddt(x);
x = (p.F - d*p.v)/k;
d = 2*b*sqrt(k*m);
```

Interface - Stiffness

Ports	**Description**
p_high	Two ports of the spring (Translation).
p_low	

Causality

preferred force out

Variables

x	extension of the spring [m]
d	damping [N.s/m]

Parameters

k	Stiffness [N/m]
b	Relative damping []
m	Reference mass [kg]

Initial Values

x_initial	The initial extension of the spring [m].

Description - Frequency

This model represents another implementation of the ideal translational spring with damper. The stiffness (*k*) is calculated on basis of a known resonance frequency. The damping value (*d*) is calculated on the basis of the calculated stiffness (*k*), relative damping (*b*) and mass reference (*m*). The mass is only used to compute the damping (no actual mass is used in this component).

The element has a preferred force out causality. The corresponding constitutive equations then contain an integration. The element can also have the non-preferred velocity out causality. The constitutive equations then contain a derivation. The spring-damper model has separate high and low ports. The equations are

```
p.F = p_high.F = p_low.F
p.v = p_high.v - p_low.v
```

Force out causality (preferred):

```
x = int(p.v);
p.F = k * x + d*p.v;
k = m*(2*pi*F)^2;
d = 2*b*sqrt(k*m);
```

Velocity out causality:

```
p.v = ddt(x);
x = (p.F - d*p.v)/k;
k = m*(2*pi*f)^2;
d = 2*b*sqrt(k*m);
```

Interface - Frequency

Ports	**Description**
p_high	Two ports of the spring (Translation).
p_low	

Causality

preferred force out

Variables

x	extension of the spring [m]

k	stiffness [N/m]
d	damping [N.s/m]

Parameters

b	Relative damping []
f	Resonance frequency [Hz]
m	Reference mass [kg]

Initial Values

x_initial	The initial extension of the spring [m].

ZeroForce

Library

Iconic Diagrams\Mechanical\Translation\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This model can be used to connect any open end of another model that is not connected to the fixed world. It generates a fixed force of 0 N while the velocity is free:

```
p.v = indifferent;
p.F = 0;
```

Interface

Ports	Description
p	Translation port

Causality

Fixed force out

Sensors

AccelerationSensor-Absolute

Library

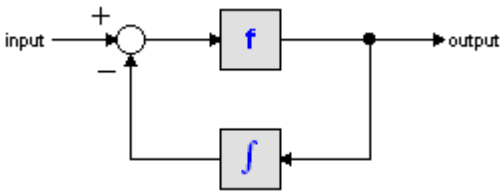
Iconic Diagrams\Mechanical\Translation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model describes an acceleration sensor which derives an acceleration output out of a port velocity by differentiation. Differentiation is performed by a state variable filter:



The S-domain function of this filter is equal to:

$$output = \frac{s}{\frac{s}{f} + 1} \cdot input$$

where *f* is the cut-off frequency. For very high values of *f*, the output becomes the pure derivative of the input. High values of *f*, however, increase simulations times. A good trade-off is a starting value of 1e5.

The equations of this model are:

a = d(p.v)/dt;
p.F = indifferent;

Interface

Ports	**Description**
p	Translation port p.
Causality	
fixed force out	
Output	
alpha	Acceleration [m/s^2]
Parameters	
f	Cut-off frequency of the differentiation [Hz].
a_initial	Initial acceleration [m/s].

AccelerationSensor-Relative

Library

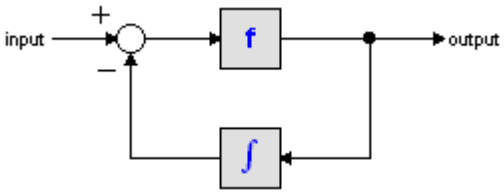
Iconic Diagrams\Mechanical\Translation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model describes an acceleration sensor which derives an acceleration output out of a velocity difference (between high and low terminals) by differentiation. Differentiation is performed by a state variable filter:



The S-domain function of this filter is equal to:

$$output = \frac{s}{\frac{s}{f}+1} \cdot input$$

where *f* is the cut-off frequency. For very high values of *f*, the output becomes the pure derivative of the input. High values of *f*, however, increase simulations times. A good trade-off is a starting value of 1e5.

The equations of this model are:

a = d(p_high.v - p_low.v)/dt;
p_low.F = p_high.F = indifferent;

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Translational port p.
Causality	
fixed force out	
Output	
a	Acceleration (measured as the difference between both terminals) [m/s^2]

Parameters

f	Cut-off frequency of the differentiation [Hz].
a_initial	Initial acceleration [m/s].

ForceSensor

Library

Iconic Diagrams\Mechanical\Translation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model translates an applied force to an output signal. It has a velocity out causality. The port *p* of the model has separate high and low terminals. The equations are:

p.F = p_high.F = p_low.F;
p.v = p_high.v - p_low.v;
F = p.F;
p.v = 0;

Interface

Ports	**Description**
p_high, p_low	Both terminals of the Translation port p.
Causality	
fixed velocity out	
Output	
F	Applied force [N].

PositionSensor-Absolute

Library

Iconic Diagrams\Mechanical\Translation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model translates a position to an output signal. It has a force out causality. The equations are:

p.F = 0;
x = int(p.v);

Interface

Ports	Description
p_high, p_low	Translation port p.
Causality	
fixed force out	
Output	
x	Absolute position [m].
Initial Values	
x_initial	Initial position [m]

PositionSensor-Relative

Library

Iconic Diagrams\Mechanical\Translation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model translates a position difference to an output signal. It has a force out causality. The port *p* of the model has separate high and low terminals. The equations are:

p.F = p_high.F = p_low.F;
p.v = p_high.v - p_low.v;
p.F = 0;
x = int(p.v);

Interface

Ports	Description
p_high, p_low	Both terminals of the Translation port p.
Causality	
fixed force out	

Output

x	Relative position [m].

Initial Values

x_initial	Initial position [m].

PowerSensor

Library

Iconic Diagrams\Mechanical\Translation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This is an ideal sensor (no dissipation or other effects) that yields the power that flows through the model as output signal. The equations are:

```
p_high.F = p_low.F
p_high.v = p_low.v
P = p_high.F * p_high.v;
```

Interface

Ports	Description
p_high, p_low	Both translation ports.

Causality

p_high not equal p_low

Output

P	Power [w].

VelocitySensor-Absolute

Library

Iconic Diagrams\Mechanical\Translation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model translates a velocity to an output signal. It has a force out causality. The equations are:

p.F = 0;
v = p.v;

Interface

Ports	Description
p	Translation port p.
Causality	
fixed force out	
Output	
v	Absolute velocity [m/s].

VelocitySensor-Relative

Library

Iconic Diagrams\Mechanical\Translation\Sensors

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation), Block Diagrams.

Description

This model translates a velocity difference to an output signal. It has a force out causality. The port *p* of the model has separate high and low terminals. The equations are:

p.F = p_high.F = p_low.F;
p.v = p_high.v - p_low.v;
p.F = 0;
v = p.v;

Interface

Ports	Description
p_high, p_low	Both terminals of the Translation port p.
Causality	
fixed force out	
Output	
v	Relative velocity [m/s].

Transmission

BeltPulley

Library

Iconic Diagrams\Mechanical\Rotation\Gears

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation/Translation).

Description

This models represents a belt and pulley. The connection to the pulley is through the rotation port *p_rot*. The connection to the belt is through the translation port *p_trans*. The model is ideal, i.e. there are no compliances or inertias. The causality of this model is always mixed: torque out & velocity out or angular velocity out & force out:

*p_rot.T = radius * p_trans.F*
*p_trans.v = radius * p_rot.omega*

or:

*p_trans.F = 1/radius * p_rot.T*
*p_rot.omega = 1/radius * p_trans.v*

Interface

Ports	**Description**
p_rot	Rotation port.
p_trans	Translation port.
Causality	
p_rot notequal	
p_trans	
Parameters	
radius	pulley radius [m]

Cam-Wizard

Library

Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation/Translation).

Description

This is a masked model which opens the Cam Wizard when edited. Depending on the selections entered, various cam motion profiles can be generated.

Interface

Ports	**Description**
p_in	Driving axis (Rotation)
p_out	Output port with resulting motion (Rotation or Translation)
Parameters	
stroke	amplitude of resulting motion
start_angle	start angle motion
stop_angle	angle when the maximum is reached
return_angle	start angle of the return motion
end_angle	finish angle of the return motion

CamRod

Library

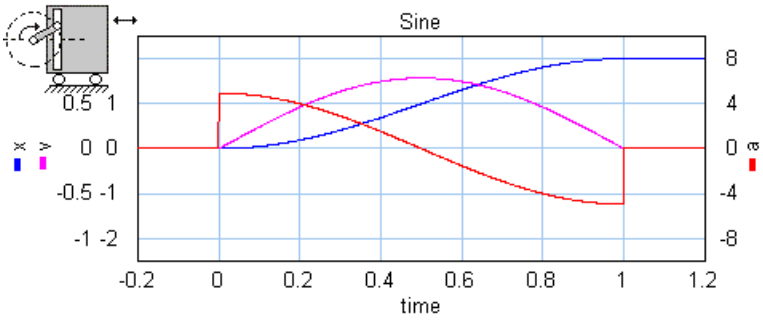
Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This models represents a cam and rod mechanism. If the input shaft is rotating with a constant speed, the output motion is a pure sinusoidal.



The mechanism is ideal, i.e., it does not have inertia, friction or geometrical limitations. It has one rotation port *(p_in)* and one translation port *(p_out)*. The causality of this model is always mixed: one port has a force out causality while the other has a velocity out causality:

*p_in.T = i * p_out.F*
*p_out.v = i * p_in.omega*

The transmission ratio (i) is the ratio of the velocities of both ports (in fact a sinusoidal function of the shaft angle).

Interface

Ports	**Description**
p_in	Driving axis (Rotation)
p_out	Output port with resulting motion (Translation)
Causality	
fixed torque out p_in	
fixed velocity out p_out	
Parameters	
stroke	Stroke of the translation port (is equal to half the length of the rod).

CrankRod

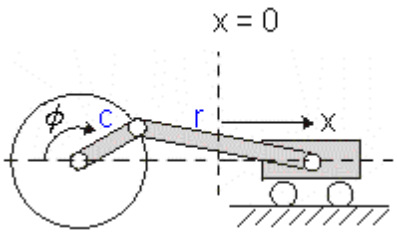
Library

Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description



This models represents a crank and rod mechanism. The mechanism is ideal, i.e., it does not have inertia, friction or geometrical limitations. It has one rotation port (*p_in*) and one translation port (*p_out*). The causality of this model is always mixed: one port has a force out causality while the other has a velocity out causality:

*p_in.T = i * p_out.F*
*p_out.v = i * p_in.omega*

The transmission ratio (i) is the ratio of the velocities of both ports. It is a function of the shaft angle, the crank length and the rod length.

Interface

Ports	**Description**
p_in	Driving axis (Rotation)
p_out	Output port with resulting motion (Translation)

Causality

fixed torque out
p_in
fixed velocity out
p_out

Parameters

crank_length	Crank length [m]
rod_length	Rod length [m]

Fork

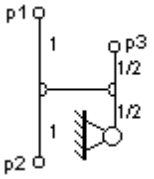
Library

Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

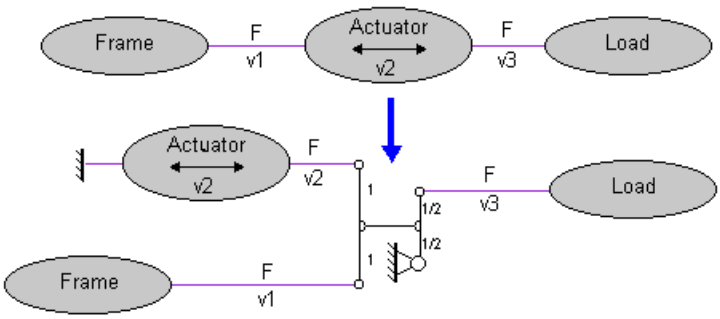
Description



This model is equivalent to the differential model of the Rotation library. It represents a special type of node where the forces are equal and the velocities are added:

p3.v = p1.v + p2.v;
p1.F = p2.F = p3.F;

This model can for example be used for actuators that generate a force difference. With the fork model an equivalent model can be found with the actuator attached to the fixed world:



Interface

Ports	Description
p1,p2,p3	Translation ports.

Lever

Library

Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This models represents any type of lever with two counter-moving ports. The lever is ideal, i.e., it does not have inertia, friction or geometrical limitations. The lever has one fast moving port and one slow moving port. The lever ratio is the (absolute) ratio of the velocities of both ports. The causality of this model is always mixed: one port has a force out causality while the other has an velocity out causality:

*p_fast.F = -i * p_slow.F*
*p_slow.v = -i * p_fast.v*

or:

*p_slow.F = -1/i * p_fast.F*
*p_fast.v = -1/i * p_slow.v*

Interface

Ports	**Description**
p_fast	Fast moving translation port.
p_slow	Slow moving translation port.

Causality

p_slow notequal
p_fast

Parameters

i	lever ratio p_slow.v / p_fast.v [], 0 < i < 1

Note

Keep the lever ratio *i* between 0 and 1. Confusion might otherwise exist:

- *i* > 1: the fast moving port is slower than the slow moving port (you better interchange the connections of the lever model).
- *i* < 0: the ports are not counter-moving (use the transmission model instead).

RackPinionGear

Library

Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Implementations

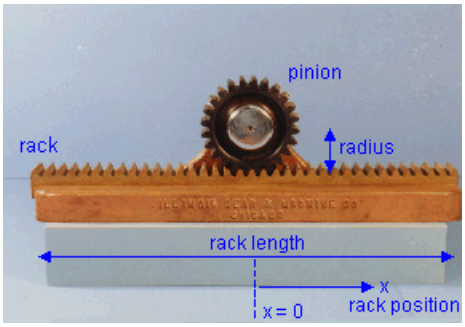
FixedPinion
FixedRack

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation/Translation).

Description - FixedPionion

This models represents a rack and pinion gear. The connection to the pinion gear is through the rotation port *p_rot*. The connection to the rack is through the translation port *p_trans*. The model is ideal, i.e. there is no compliance nor inertia nor backlash.



In this model the pinion bearing is connected to the fixed world and the rack is free to move. This in contrary to the model FixedRackPinionGear where the pinion bearing is free to move and the rack is connected to the fixed world.

The causality of this model is always mixed: torque out & velocity out or angular velocity out & force out:

*p_rot.T = radius * p_trans.F*
*p_trans.v = radius * p_rot.omega*

or:

*p_trans.F = 1/radius * p_rot.T*
*p_rot.omega = 1/radius * p_trans.v*

The rack position is determined by the internal variable x. For x = 0, the pinion gear is at the middle of the rack. When the pinion crosses the end of the rack, i.e.

abs(x) > rack_length/2

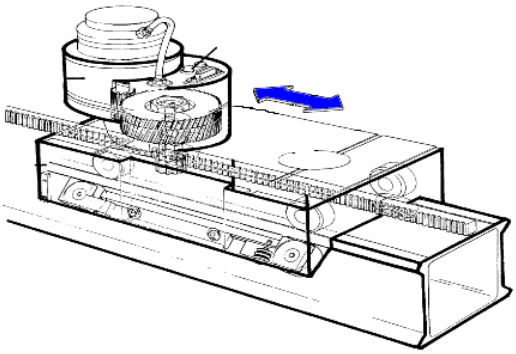
a warning is given, *"WARNING: rack length has been exceeded at the rack and pinion gear!"*, and the simulation is stopped.

Interface - FixedPionion

Ports	**Description**
p_rot	Rotation port.
p_trans	Translation port.
Causality	
p_rot notequal	
p_trans	
Parameters	
radius	pinion gear pitch radius [m]
rack_length	rack length [m]
Variables	
x	Internal variable which denotes the rack position, abs(x) < rack_length/2 else simulation halted.
Initial values	
x_initial	Initial rack position, abs(x_initial) < rack_length/2

Description - FixedRack

This models represents a fixed rack and pinion gear. The connection to the pinion gear is through the rotation port *p_rot*. The connection to the rack is through the translation port *p_trans*. The model is ideal, i.e. there is no compliance nor inertia nor backlash.



In this model the pinion bearing is free to move and the rack is connected to the fixed world. This in contrary to the model RackPinionGear where the pinion bearing is connected to the fixed world and the rack is free to move.

The causality of this model is always mixed: torque out & velocity out or angular velocity out & force out:

*p_rot.T = radius * p_trans.F*
*p_trans.v = radius * p_rot.omega*

or:

*p_trans.F = 1/radius * p_rot.T*
*p_rot.omega = 1/radius * p_trans.v*

The rack position is determined by the internal variable x. For x = 0 the pinion gear is at the middle of the rack. When the pinion crosses the end of the rack, i.e.

abs(x) > rack_length/2

a warning is given, *"WARNING: rack length has been exceeded at the rack and pinion gear!"*, and the simulation is stopped.

Interface - FixedRack

Ports	**Description**
p_rot	Rotation port.
p_trans	Translation port.
Causality	
p_rot notequal	
p_trans	
Parameters	
radius	pinion gear pitch radius [m]
rack_length	rack length [m]
Variables	
x	Internal variable which denotes the rack position, abs(x) < rack_length/2 else simulation halted.
Initial values	
x_initial	Initial rack position, abs(x_initial) < rack_length/2

Spindle

Library

Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation).

Introduction

This models represents a spindle and nut. It transfers an angular motion of the spindle into a translational motion of the nut. The model is ideal, i.e., it does not have inertia or friction. The causality of this model is always mixed: one port has a torque out causality while the other has an angular velocity out causality:

*p_spindle.T = i * p_nut.F*
*p_nut.v = i * p_spindle.omega*

or:

*p_nut.F = 1/i * p_spindle.T*
*p_spindle.omega = 1/i * p_nut.v*

The model has two implementations which calculate the transform ratio *i* out of different parameters.

Description - Pitch

In this implementation the transform ratio is calculated using the *pitch* (the advance of the nut during one revolution of the spindle):

*i = pitch / (2 * pi);*

Interface - Pitch

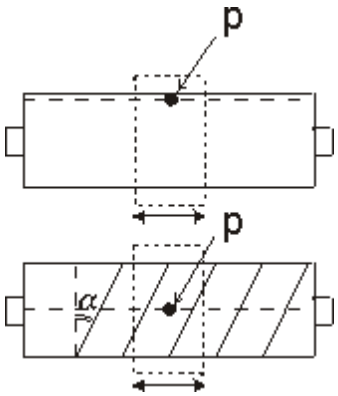
Ports	**Description**
p_spindle	Port at the spindle shaft (Rotation).
p_nut	Port at the wheel (Translation).
Causality	
p_spindle	
notequal p_nut	
Parameters	
pitch	translation of the nut during one revolution of the spindle [m]

Description - LeadAngle

This implementation calculates the transform ratio out of the lead angle *alpha* and the radius *r_spindle* of the spindle:

*i = tan(alpha)*r_spindle;*

The pitch angle is shown in the figure below. *r_spindle* is the effective radius of the spindle, i.e. the radius from the center of the spindle to the pitch point *p*.



Interface -LeadAngle

Ports	**Description**
p_spindle	Port at the spindle shaft (Rotation).
p_nut	Port at the wheel (Translation).
Causality	
p_spindle notequal p_nut	
Parameters	
r_spindle	effective radius of the spindle [m]
alpha	lead angle of the spindle [rad]

TimingBelt

Library

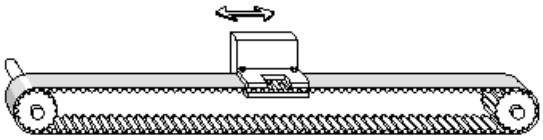
Iconic Diagrams\Mechanical\Rotation\Gears
Iconic Diagrams\Mechanical\Translation\Transmission

Use

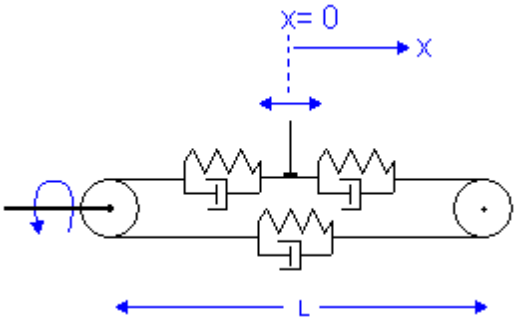
Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Rotation/Translation).

Introduction

This models represents a timing belt, used for linear positioning.



It has a rotating pulley which drives the the belt and clamp. The timing belt is modeled by a series of spring damper elements that convey the rotation of the pulley to a clamp translation. Because the output position is moving, stiffness and damping values are not constant.



The stiffness for a piece of belt can be expressed as:

$$k = E*A / l$$

with

E = Modulus of elasticity {N/m2}
A = Belt area {m2}
l = belt length {m}

If the belt is sufficiently pre-tensioned, the stiffness experienced at the clamp can be expressed as the combination of three individual belt parts:

$$k = E*A/(0.5*l+x) + 1/(\ 1 / E*A/(0.5*l - x)\ + 1 / E*A / l\ )$$

which can be rewritten to:

$$k = E*A*(\ 1/(0.5*l+x) + 1/(1.5*l - x)\ )$$

The stiffness approaches infinity as the clamp moves to the driven pulley (x = -l/2) and has a minimum value when the clamp moves to the other pulley (x = 0.5*l). The minimum stiffness is equal to:

$$k = 2*E*A / l$$

The belt position is determined by the internal variable *x*. For *x* = 0 the clamp is in the middle. When the position crosses the driven pulley, i.e.

x < - belt_length/2

the simulation is stopped: *"Error: clamp position larger than belt end!".* When the position crosses the other pulley, i.e.

x > belt_length/2

the simulation is also stopped, *"Error: clamp position smaller than belt start!".*

Description - Default

In this model the minimum stiffness is used, based on an output position at a length L of the driven pulley.

*k = 2*E*A / l*

Description - VariableStiffness

In this model a variable stiffness is used equal to:

*k = E*A*(1/(0.5*l+x) + 1/(1.5*l - x))*

Take care not to let the clamp get too close to the driven pulley, because the stiffness will then grow to infinity!

Interface

Ports	**Description**
p_rot	Rotation port.
p_trans	Translation port.
Causality	
p_rot notequal	
p_trans	
Parameters	
radius	pinion gear pitch radius [m]
d	damping N.s/m]
E	Modulus of elasticity [N/m2]
A	*A* = Belt area [m2]
l	belt length [m]
Variables	
x	clamp position, abs(x) < belt length/2
Initial values	
x_initial	Initial clamp position, abs(x_initial) < belt length/2

Transmission

Library

Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

This models represents any type of lever with ports moving in the same direction. The lever is ideal, i.e., it does not have inertia, friction or geometrical limitations. The lever has one fast moving port and one slow moving port. The transmission ratio is the ratio of the velocities of both ports. The causality of this model is always mixed: one port has a force out causality while the other has an velocity out causality:

*p_fast.F = i * p_slow.F*
*p_slow.v = i * p_fast.v*

or:

*p_slow.F = 1/i * p_fast.F*
*p_fast.v = 1/i * p_slow.v*

Interface

Ports	**Description**
p_fast	Fast moving rotation port.
p_slow	Slow moving rotation port.

Causality

p_slow notequal
p_fast

Parameters

i	transmission ratio p_slow.v / p_fast.v [], 0 < i < 1

Note

Keep the transmission ratio i between 0 and 1. Confusion might otherwise exist:

- *i* > 1: the fast moving port is slower than the slow moving port (you better interchange the connections of the transmission model).
- *i* < 0: the ports are counter-moving (use the lever model instead).

UniversalLever

Library

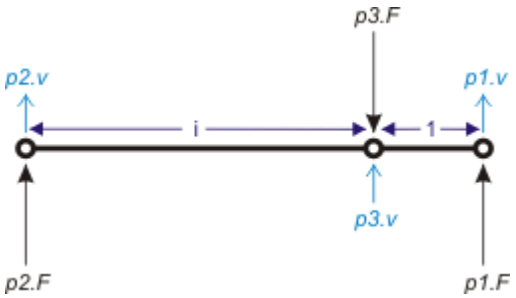
Iconic Diagrams\Mechanical\Translation\Transmission

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description

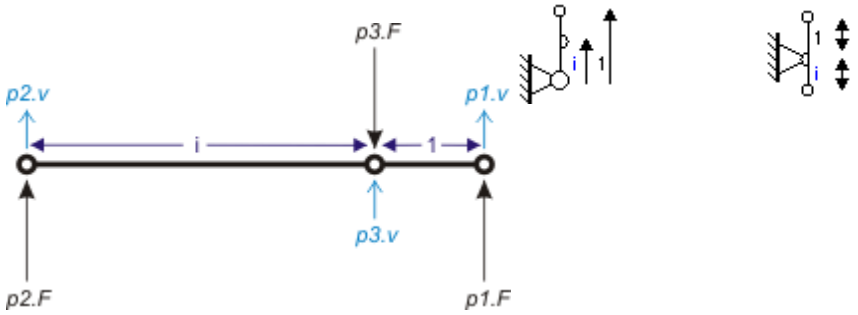
This model represents an ideal transmission with a free pivot point. The relations are:



*p3.v = (i/(1+i))*p1.v + (1/(1+i))*p2.v;*
*p2.F = (1/(1+i)*p3.F;*
*p1.F = (i/(1+i)*p3.F;*

Ground

Although each of the three ports may be connected to the ground, it is more efficient to use other models. When p1 or p2 is connected to the ground, this model is equal to the Transmission.emx model. When p3 is connected to the ground, this model is equal to the Lever.emx model.



Transmission-Universal Transmission Lever

Interface

Ports	Description
p1,p2,p3	Translation ports.

Parameters

i	Transmission ratio []

9.2.5 Thermal

Thermal

The Thermal library contains components which are very useful for modeling thermal systems. The following libraries are available.

- Components
- Generators
- Sensors

Note

- For domains that have port variables that do no multiply to power the name *pseudo* is used. This model has pseudothermal ports. The port variables of a pseudothermal port are the heat flow, p.dQ [W], and the temperature, p.T [K], which do not multiply to power.
- The base unit of temperature is degrees Kelvin. You can select degrees Celsius in the Parameters Editor or Variables Chooser desired. 20-sim will take care that internally always SI units (Kelvin) are used for calculations.

Components

Convection

Library

Iconic Diagrams\Mechanical\Thermal\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This model describes linear heat convection between a material surface and gas or fluid flow. The basic equation is:

```
p.dQ = (p1.T - p2.T)*Gc;
```

where *p1.T* is the temperature of at the surface and *p2.T* the gas or fluid temperature. *Gr* is the thermal conductance which is function of the surface area *A* and the convection coefficient *h*:

```
Gr = A*h;
```

In most cases *h* is not contant but a non-analytical function of the gas or fluid speed and other poperties. In most cases *Gr* is experimentally defined. Therefore *G* is given as an input signal.

Interface

Ports	**Description**
p1	Material port
p2	Fluid port
Causality	
indifferent	
Input	
G	thermal conductance [W/K]

Ground

Library

Iconic Diagrams\Mechanical\Thermal\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This is a general model for the heat storage that is so large that is can be considered constant (e.g. the environment). It is comparable to the ground in the electrical domain.

```
T0 = 25 {degC};
p.T = T0;
```

Interface

Ports	**Description**
p[any]	Any number of connections can be made (pseudothermal).
Causality	
fixed temperature out	An torque out causality results in a derivative constitutive equation.
Parameters	
T0	temperature [K]

HeatCapacity

Library

Iconic Diagrams\Mechanical\Thermal\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This is a general model for the heat storage in a specific material. A constant temperature distribution in the material is assumed and a constant heat capacity:

```
E = int(p.dQ) + C*T0;
p.T = E/C;
```

T0 is the initial temperature of the material, *E* the internal energy {J} and *C* the thermal capacity {J/K}. Because any number of connections can be made, successive ports are named p1, p2, p3 etc. 20-sim will automatically create equations such that the resulting heat flow p.dQ is equal to the sum of the heatflows of all connected ports p1 .. pn. The temperatures of all connected ports are equal to element temperature *p.T*.

```
p.dQ = sum(p1.dQ, p2.dQ, ....)
p.T = p1.T = p2.T = ....
```

The thermal capacity can be calculated with the specific heat capacity *cp* and material mass *m*:

```
C = cp*m;
```

Typical values for *cp* are:

water	4186
granite	790
glass	840
aluminium	900
concrete	840?
copper	387
silver	235
iron / steel	452
wood	1674
air (50 °C)	1046

Interface

Ports	Description

p[any]	Any number of connections can be made (pseudothermal).

Causality

preferred temperature out	An torque out causality results in a derivative constitutive equation.

Variables

E	internal energy [J]

Parameters

C	thermal capacity [J/K]
T0	initial temperature [K]

Radiation

Library

Iconic Diagrams\Mechanical\Thermal\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This model describes the heat transfer between to bodies through radiation*:*

*p.dQ =Gr*sigma*(p1.T^4 - p2.T^4);*

where *p1.T* and *p2.T* are the temperatures of the body surfaces and sigma is the Stefan-Boltzmann constant. For simple cases, Gr may be analytically computed. The analytical equations use epsilon, the surface emissitivy of a body which is in the range 0..1. Epsilon=1, if the body absorbs all radiation (= black body). Epsilon=0, if the body reflects all radiation and does not absorb any.

Typical values for epsilon are:

aluminium, polished	0.04
copper, polished	0.04
gold, polished	0.02
paper	0.09
rubber	0.95
wood	0.85..0.9

Analytical Equations for Gr

Small convex object in large enclosure (e.g., a hot machine in a room):

*Gr = e*A;*

where
e: Emission value of object (0..1)
A: Surface area of object where radiation heat transfer takes place
Two parallel plates:

Gr = A/(1/e1 + 1/e2 - 1);

where
e1: Emission value of plate1 (0..1)
e2: Emission value of plate2 (0..1)
A : Area of plate1 (= area of plate2)

Interface

Ports	**Description**
p1	Material port
p2	Fluid port

Causality

indifferent

Input

G	thermal conductance [W/K]

ThermalConductance

Library

Iconic Diagrams\Mechanical\Thermal\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

When a temperature difference exists in a material, heat will flow from the high temperature part to the low temperature part. This model describes the transport of heat through a block of material with a temperature difference on both sides of the block. The heat conducting capacity of the material is indicated by the parameter *G* (thermal conductance):

*p.dQ = p.T*G;*

This model is the dual form of the thermal resistor where the insulation capacity of the block is indicated by the parameter *R* (thermal resistance)*:*

p.dQ = p.T/R;

The heat flow through the material is ideal. I.e. there is no heat storage modeled. The thermal resistance model has separate high and low ports. The equations are

p.dQ = p_high.dQ = p_low.dQ
p.T = p_high.T - p_low.T

Block

The thermal conductance of a block can be calculated with the specific thermal conductivity *k*, the length *L* of the block, and the area *A* of both sides of the block:

*G = k*A/L*

Typical values of the thermal conductivity [W.m-1.K-1] are:

water	0.6
granite	2,1
glass	1.0
aluminium	210
concrete	1.28
copper	390
silver	430
iron / steel	40-70
wood	0.13
air	0.026

Interface

Ports	**Description**
p_high, p_low	Both terminals of the pseudothermal port p

Causality

indifferent

Parameters

G	thermal conductance [W/K]

ThermalResistor

Library

Iconic Diagrams\Mechanical\Thermal\Components

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

When a temperature difference exists in a material, heat will flow from the high temperature part to the low temperature part. This model describes the transport of heat through a block of material with a temperature difference on both sides of the block. The insulation capacity of the block is indicated by the parameter *R* (thermal resistance)*:*

p.dQ = p.T/R;

This model is the dual form of the thermal conductor where the heat conducting capacity of the material is indicated by the parameter *G* (thermal conductance):

*p.dQ = p.T*G;*

The heat flow through the material is ideal. I.e. there is no heat storage modeled. The thermal resistance model has separate high and low ports. The equations are

p.dQ = p_high.dQ = p_low.dQ
p.T = p_high.T - p_low.T

Block

The thermal resistance of a block can be calculated with the specific thermal conductivity *k*, the length *L* of the block, and the area *A* of both sides of the block:

*R = L/(k*A)*

Typical values of the thermal conductivity [W.m-1.K-1] are:

water	0.6
granite	2,1
glass	1.0
aluminium	210
concrete	1.28
copper	390
silver	430
iron / steel	40-70
wood	0.13
air	0.026

Interface

Ports	**Description**
p_high, p_low	Both terminals of the pseudothermal port p

Causality

indifferent

Parameters

R	thermal resistance [K/W]

Generators

HeatFlow

Library

Iconic Diagrams\Mechanical\Thermal\Generators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This model yields a heat flow that is set to a constant value. The temperature is indifferent. The model can represent any heat generating object with a constant heat flow.

Interface

Ports	Description
p	

Causality

fixed heat flow out

Parameters

dQ	Heat flow [J/s]

ModulatedHeatFlow

Library

Iconic Diagrams\Mechanical\Thermal\Generators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This model yields a heat flow that is equal to a (fluctuating) value given by the input signal *dQ*. The temperature is indifferent. The model can represent any heat generating object with a fluctuating heat flow.

Interface

Ports	Description

p

Causality

fixed heat flow out

Parameters

dQ	Heat flow [J/s]

ModulatedTemperatureSource

Library

Iconic Diagrams\Mechanical\Thermal\Generators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This model has a temperature that is equal to a (fluctuating) value given by the input signal *T*. The heat flow is indifferent. This model can represent an object that has a heat capacity which is so much larger than the rest of the system, that its temperature can be seen as an independant variable.

Interface

Ports	**Description**
p	
Causality	
fixed temperature out	
Input	
T	Variable temperature signal [K]

Resistor

Library

Iconic Diagrams\Thermal\Generators\Generators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal, Electric).

Description

This is a model of a resistor with a temperature coefficient. The resistance and voltage equation can be written as:

```
R = R_ref*(1 + alpha*(p_th.T - T_ref));
p_el.u = p_el.i * R;
```

where *R_ref* is the reference resistance at reference temperature *T_ref* and alpha the temperature coefficient of resistance. The actual temperature of the resistor is given by the pseudothermal port of the model , *p_el.T*. The heat generated by the resistor is equal to

```
p_th.dQ = p_el.u * p_el.i;
```

The electrical port *p* of the resistor model has separate high and low terminals. The equations are:

```
p_el.i = p_high_el.i = p_low_el.i;
p_el.u = p_el_high.u - p_el_low.u;
```

Interface

Ports	**Description**
p_el_high	Both terminals of the Electric port.
p_el_low	
p_th	The pseudothermal port
Causality	
indifferent p_el	
fixed heat flow	
out p_th	
Parameters	
R_ref	Reference resistance [Ohm].
T_ref	Reference temperature [K].
alpha	Temperature coefficient of the resistance []

TemperatureSource

Library

Iconic Diagrams\Mechanical\Thermal\Generators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This model has a temperature that is set to a certain constant value. The heat flow is indifferent. The model can represent a piece of material which is so large that its temperature variation is neglectable (e.g. the environment) and thus can be seen as an ideal temperature source

Interface

Ports	Description
p	
Causality	
fixed temperature out	
Parameters	
T	Fixed temperature [K]

Thermistor

Library

Iconic Diagrams\Thermal\Generators

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal, Electric).

Description

This is a model of a linear thermistor. Thermistors are thermally sensitive resistors and have, according to type, a negative (NTC), or positive (PTC) resistance/temperature coefficient. For linear thermistors the resistance and voltage equation can be written as:

```
R = R_ref*(1 + alpha*(p_th.T - T_ref));
p_el.u = p_el.i * R;
```

where *R_ref* is the reference resistance at reference temperature *T_ref* and alpha the temperature coefficient of resistance. The actual temperature of the themistor is given by the pseudothermal port of the model , *p_el.T*. The heat generated by the thermistor is equal to

```
p_th.dQ = p_el.u * p_el.i;
```

The electrical port *p* of the thermistor model has separate high and low terminals. The equations are:

```
p_el.i = p_high_el.i = p_low_el.i;
p_el.u = p_el_high.u - p_el_low.u;
```

Interface

Ports	Description
p_el_high	Both terminals of the Electric port.
p_el_low	
p_th	The pseudothermal port
Causality	

indifferent p_el
fixed heat flow
out p_th

Parameters

R_ref	Reference resistance [Ohm].
T_ref	Reference temperature [K].
alpha	Temperature coefficient of the resistance []

Sensors

HeatFlowSensor

Library

Iconic Diagrams\Mechanical\Thermal

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This is a model of an ideal heat flow sensor with an output signal *dQ* that is equal to the measured heat flow. The sensor is ideal which means that is does not store, loses or adds heat.

Interface

Ports	**Description**
p_high, p_low	Both ports of the sensor

Causality

p_high not equal p_low

Parameters

dQ	Measured heat flow [J/s]

TemperatureSensor

Library

Iconic Diagrams\Mechanical\Thermal

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Pseudothermal).

Description

This is a model of an ideal temperature sensor with an output signal *T* that is equal to the measured temperature. The heat flow into the model is zero to assure zero power loss.

Interface

Ports	**Description**
p	
Causality	
fixed heat flow out	
Parameters	
T	Measured temperature [K]

9.3 Signal

9.3.1 Block Diagram

Attenuate

Library

Signal\Block Diagram

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model divides an input signal by a constant value.

output = input/K

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
K	Division parameter

Delay-Pade

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model delays the input signal by "T" seconds using a 4th order Pade approximation. The model is linear which means that it can be used in models that should be linearized.

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
T	Delay time.

Delay-Step

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model delays its input signal one simulation step.

output = initial; (time = 0)
output = inp(time - hk); (time > 0)

where hk = simulation step size.

Note

To correctly use models with constant delays, simulation runs must be performed using integration algorithms with a constant time-step. Otherwise the model Delay-Time.emx should be used!

Interface

Inputs	**Description**
input	
Outputs	

output

Parameters

initial	The initial output value.

Note

- To correctly use models with constant delays, simulation runs must be performed using integration algorithms with a constant time-step. Otherwise the model Delay-Time.emx should be used!
- The delay-step.emx submodel is non-linear: i.e. models which include the *delay-step.emx* submodel cannot be linearized. Use the linear time delay Delay-Pade.emx if you want to perform linearization.

Delay-Time

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model stores the input signal and corresponding time in a buffer. After "delay" seconds the stored values are retrieved. When the simulation time step does not coincide with the delay time "delay", the output is calculated by first order interpolation.

output = initial; (time < delay)
output = inp(time - delay); (time >= delay)

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
initial	The initial output value.
delay	Delay time.

Delay-VariableTime

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size:** 1-D. **Kind**: Block Diagrams.

Description

This model stores the input signal and corresponding time in a buffer. After "delay" seconds the stored values are retrieved. This delay is a variable input signal. When the simulation time step does not coincide with the delay time "delay", the output is calculated by first order interpolation.

output = initial; (time < delay)
output = input(time - delay); (time >= delay)
output = input(time); (delay < 0)

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
initial	The initial output value.
delay	Delay time.

Note

- The maximum storage capacity of the buffer is 58254 input values. When many input values are stored the simulation speed may slow down considerably!
- The delay-variabletime.emx submodel is non-linear: i.e. models which include the delay-variabletime.emx submodel cannot be linearized. Use the linear time delay Delay-Pade.emx if you want to perform linearization.

Demux

Library

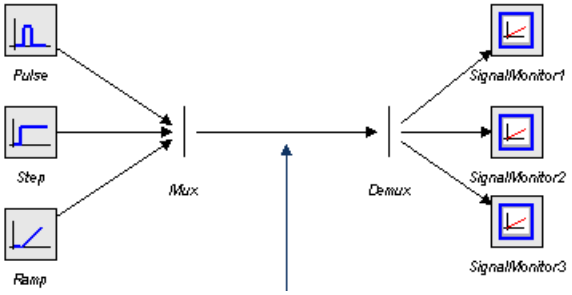
Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

You can use the Mux model to combine multiple input signals into one multi-dimensional signal.
This model can be used to spit the multi-dimensional signal again into single signals.



Interface

Inputs	**Description**
inputs	Default 2 signals can be connected. Use the right mouse menu and choose *Edit Implementation* to change the number of signals.
Outputs	
output[any]	The output signal size is equal to the number of inputs.

Differentiate-Calculus

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model approximates the differentiation of an input signal by the equation:

$$y = \frac{2}{3} \cdot \frac{u(t) - u(t-h)}{h} + \frac{1}{3} \cdot y(t-h)$$

where hk = simulation step size. The initial value of the output is equal to the parameter "initial". When the simulation step size is zero (time event, state event) the output value will be equal to the previous output value (the simulation step size part is not calculated to prevent a division by zero).

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
initial	The initial value of the output.

Differentiate-FO

Library

Signal\Block Diagram

Use

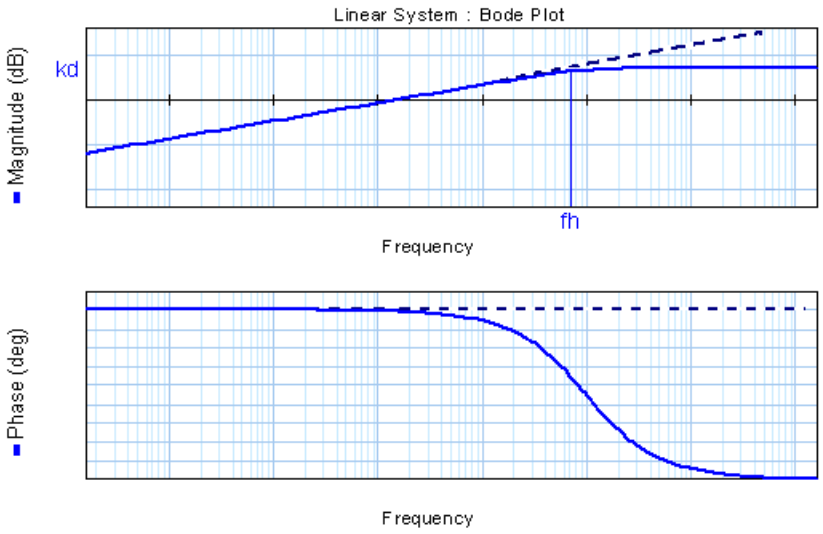
Domains: Continuous. **Size:** 1-D. **Kind**: Block Diagrams.

Description

This model is a first order approximation of a differentiation. The transfer function is:

$$Y(s) = kd \cdot \frac{2 \cdot \pi \cdot fh \cdot s}{s + 2 \cdot \pi \cdot fh} \cdot U(s)$$

As shown in the figure below, the model behaves as a differentiator for frequencies below *fl.*



Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
kd	Differentiation gain []
fl	Differentiation limit [Hz]
Initial Values	
output_initial	The initial value of the integral.

Differentiate-SVF

Library

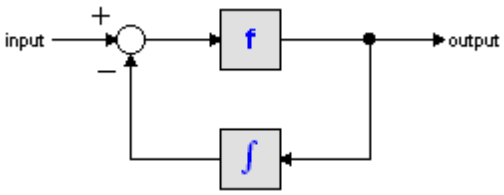
Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind:** Block Diagrams.

Description

This model approximates the differentiation of an input with a state variable filter:



As can be seen, no differentiation is used in this model and consequently all integration algorithms can be used. In the S-domain the output of this model is equal to:

$$output = \frac{s}{\frac{s}{f}+1} \cdot input$$

For very high values of N, the output becomes the pure derivative of the input. High values of N, however, increase simulations times. A good trade-off is a starting value of 10. If more accuracy is needed, this value can be increased.

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
N	Derivative Gain Limitation.
initial	The initial value of the output.

Differentiate

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model differentiates an input signal. The initial value of the output is equal to the parameter "initial".

output = ddt(input,initial);

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
initial	The initial value of the output.

Filter

Library

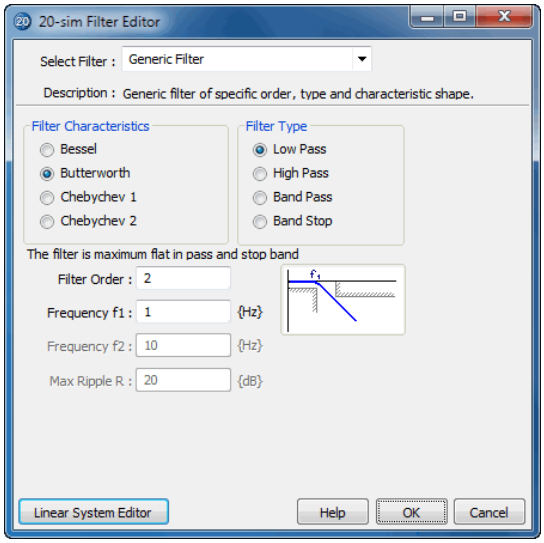
Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

When you select this model and click Go Down a special editor opens (Filter Editor), allowing you to choose a filter:



Interface

Inputs	Description
input	
Outputs	
output	
Initial Values	
	The model has internal states that are not accessible.
Parameters	
	Parameters are entered by the Filter Editor.

Gain

Library

Signal\Block Diagram

Use

Domains: Continuous, Discrete. Size: 1-D. Allowed in: Block Diagrams.

Description

This model multiplies an input signal with a constant value.

*output = K * input*

Interface

Inputs	Description

input	
Outputs	
output	
Parameters	
K	Gain

Integrate-ExpWindow

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model integrates an input signal over an exponential window with time constant T0 [s].

$$output(t) = \int_0^t input(\tau) \cdot e^{(t-\tau)/To} d\tau$$

The model behaves as a signal averager, with an effective window of T0 [s].

Interface

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
state(0)	The initial value of the integral.
Parameters	
T0	Window constant [s].

Integrate-FO

Library

Signal\Block Diagram

Use

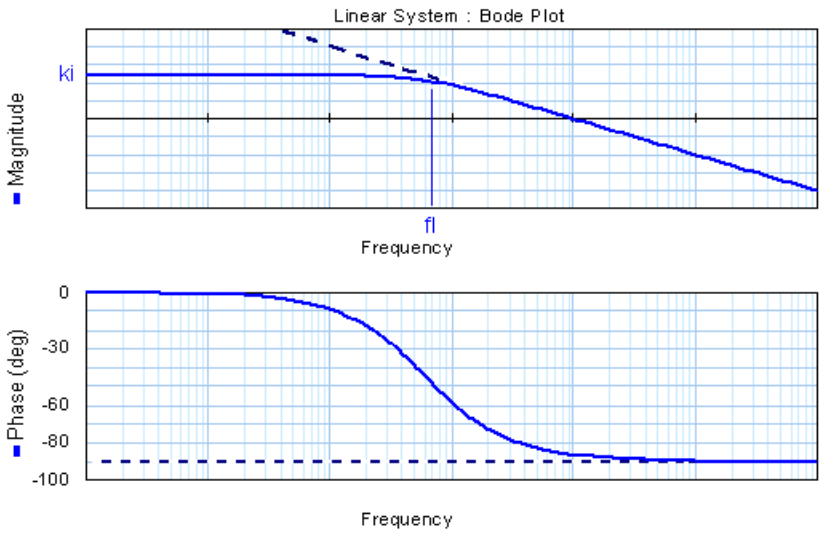
Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model is a first order approximation of an integration. The transfer function is:

$$Y(s) = \frac{ki}{2 \cdot \pi \cdot fl + s} \cdot U(s)$$

As shown in the figure below, the model behaves as an integrator for frequencies above the *fl*



Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
ki	Integration gain []
fl	Integration limit [Hz]
output_initial	Initial value.

Integrate-FOLimited

Library

Signal\Block Diagram

Use

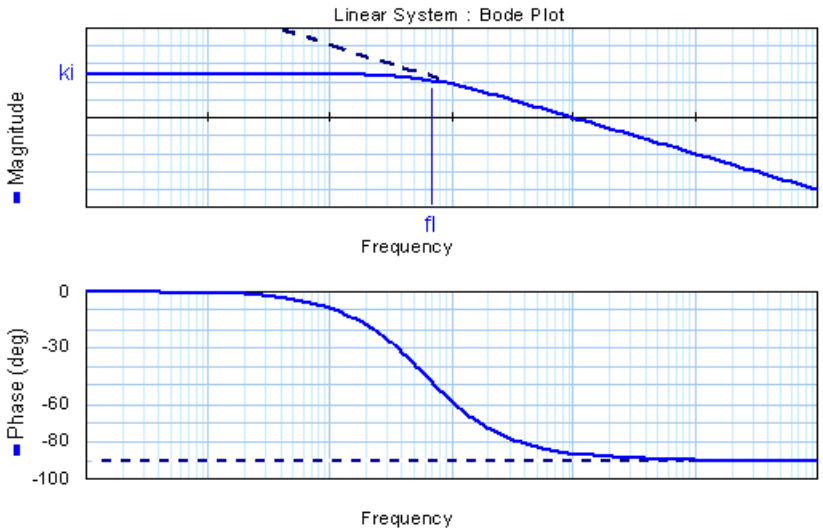
Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model is a first order approximation of an integration with an output value limited to a minimum and maximum value. The model checks the output value to see if the maximum of minimum values have been crossed. If so the input value is set to zero to prevent the integral from winding-up. Between the bounds the transfer function is:

$$Y(s) = \frac{ki}{2 \cdot \pi \cdot fl + s} \cdot U(s)$$

As shown in the figure below, the model behaves as an integrator for frequencies above the *fl*



Interface

Inputs	Description
input	

Outputs	
output	
Parameters	
ki	Integration gain []
fl	Integration limit [Hz]
output_initial	Initial value.
maximum	Maximum output value
minimum	Minimum output value

Integrate-Limited

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model integrates an input signal. The output of this integral is limited between a maximum and minimum bound. If the integral is in saturated condition and the input changes sign, the output wanders away from the bounds immediately.

output = int(inp) + state(0); (minimum <= output <= maximum)

Interface

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
state(0)	The initial value of the integral.
Parameters	
minimum	Minimum output value.
maximum	Maximum output value.

Integrate-RectWindow

Library

Signal\Block Diagram

Use

Domains: Continuous. Size: 1-D. Kind: Block Diagrams.

Description

This model integrates an input signal over a limited time interval which is T0 [s] wide.

$$output = \int_{t-T_0}^{t} input \cdot dt$$

The model behaves as a signal averager, with a rectangular window of T0 [s].

Interface

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
state(0)	The initial value of the integral.
Parameters	
T0	Integration interval.

Integrate-Reset

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description-Default

This model integrates an input signal. The output of this integral may be reset to any desired value each time the reset signal is unequal to zero.

output = int(inp) + state(0); (reset = 0)
output = reset to newoutput; (reset <> 0)

This implementation uses a normal integrator. That is why the model will also work when c-code is generated.

Interface-Default

Inputs	**Description**
input	Input signal
reset	When reset <> 0, the output changes to newoutput. New output.
newoutput	
Outputs	
output	
Initial Values	
state(0)	The initial value of the integral.

Description-Pure

This model integrates an input signal. The output of this integral may be reset to any desired value each time the reset signal is unequal to zero.

output = int(inp) + state(0); (reset = 0)
output = reset to newoutput; (reset <> 0)

This implementation uses the function *resint*. This function forces the integration method with a state event to search for the exact time where the integral is reset. The result is more accurate than the *Default* implementation, but this implementation will not work when c-code is generated.

Interface-Pure

Inputs	**Description**
input	Input signal
reset	When reset <> 0, the output changes to newoutput. New output.
newoutput	
Outputs	
output	
Initial Values	
state(0)	The initial value of the integral.

Integrate

Library

Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model integrates an input signal:

output = int(input) + state(0)

Interface

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
state(0)	The initial value of the integral.

Inverse

Library

Signal\Block Diagram

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the inverse of the input signal.

output = 1 / input

Interface

Inputs	**Description**
input	
Outputs	
output	

Linear System

Library

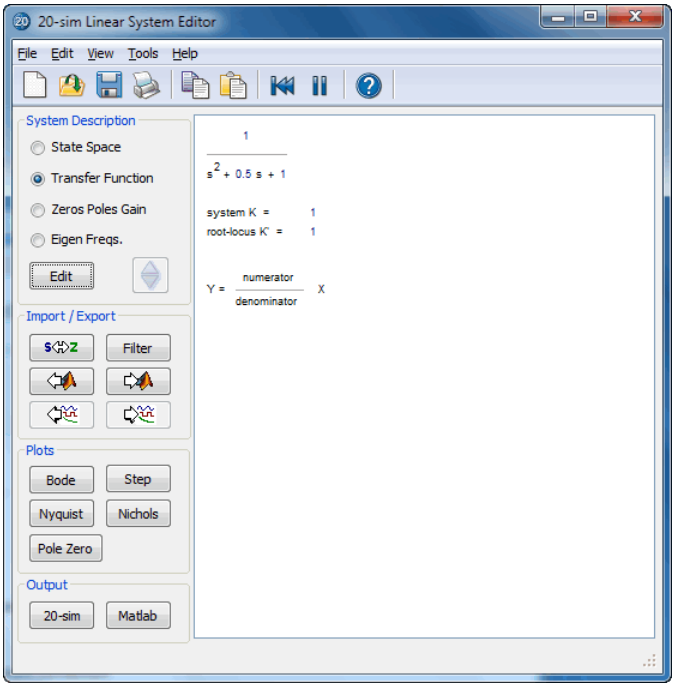
Signal\Block Diagram

Use

Domains: Continuous**. Size**: 1-D. **Kind**: Block Diagrams.

Description

When you select this model and click Go Down a special editor opens (Linear System Editor), allowing you to enter a linear system in State Space form, as a Transfer Function or by adding poles and zeros:



Interface

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
	The model has internal states that are not accessible.
Parameters	
	Parameters are entered by the Linear System Editor.

MultiplyDivide

Library

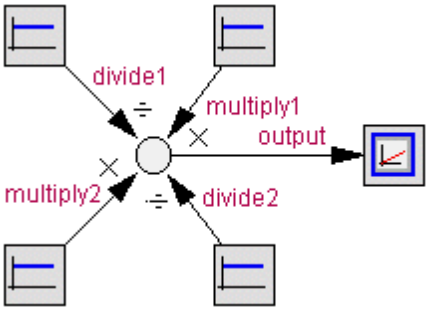
Signal\Block Diagram , System

Use

Domains: Continuous, Discrete. **Size**: [n,m]. **Kind**: Block Diagrams.

Description

This model yields a multiplication and/or division of one or more input signals. While connecting input signals, 20-sim will ask whether the signals should be multiplied or divided. The signals may have either size, but the size of all connected signals should be equal.



For example the MultiplyDivide model can have 2 signals connected that are multiplied and 2 signals that must be divided. 20-sim treats them as an array multiply and an array divide with:

multiply = [multiply1;multiply2];

and

divide = [divide1;divide2];

The output will then be equal to:

*output = (multiply1 * multiply2) / (divide1 * divide2);*

Interface

Inputs	Description
multiply[any] divide[any]	Any number of input signals (each of the same size) may be connected to be multiplied (multiply) or divided (divide).

Outputs

output

Limitations

The signals connected to this model should all have the same size.

Mux

Library

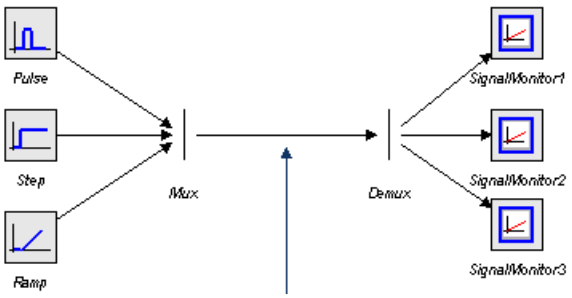
Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

You can use this model to combine multiple input signals into one multi-dimensional signal.
The Demux model can be used to spit the multi-dimensional signal again into single signals.



Interface

Inputs	Description
inputs	Default 2 signals can be connected. Use the right mouse menu and choose *Edit Implementation* to change the number of signals.
Outputs	
output[any]	The output signal size is equal to the number of inputs.

Negate

Library

Signal\Block Diagram

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the negative of the input signal.

output = -input

Interface

Inputs	**Description**
input	
Outputs	
output	

PlusMinus

Library

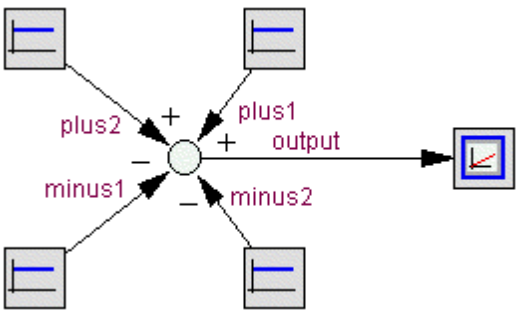
Signal\Block Diagram , System

Use

Domains: Continuous, Discrete. **Size**: [n,m]. **Kind**: Block Diagrams.

Description

This model yields a subtraction and/or summation of one or more input signals. While connecting input signals, 20-sim will ask whether the signals should be added or subtracted. The signals may have either size, but the size of all connected signals should be equal.



For example the PlusMinus model can have 2 signals connected that are added and 2 signals that must be subtracted. 20-sim treats them as an array plus and an array minus with:

plus = [plus1;plus2];

and

minus = [minus1;minus2];

The output will then be equal to:

output = plus1 + plus2 - minus1 - minus2;

Interface

Inputs	Description
plus[any] minus[any]	Any number of input signals (each of the same size) may be connected to be added (plus) or subtracted (minus).

Outputs

output

Limitations

The signals connected to this model should all have the same size.

SignalMonitor

Library

Signal\Block Diagram , Signal\Sources

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model shows the value of its input in the Simulator. When you open the Simulator (select Properties and Plot) the input of this model is automatically selected as plotvariable. As label for this variable the local name of the SignalMonitor model is chosen. It is therefore advised to give each SignalMonitor model a useful name (select the model, click the right mouse button and select Attributes from the right mouse menu).

Interface

Inputs	Description
input	The value of the input is shown in the Simulator using the local name of the model as label.

Splitter

Library

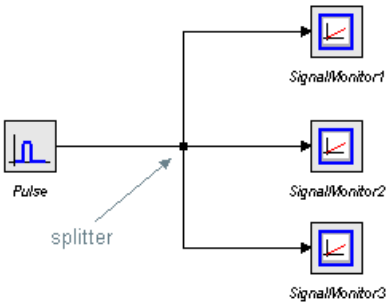
Signal\Block Diagram , System

Use

Domains: Continuous, Discrete. **Size**: [n,m]. **Kind**: Block Diagrams.

Description

This model can split one output signal into two or more output signals. The signals may have either size, but the size of all connected signals should be equal.



Interface

Inputs	**Description**
input	One input signal must be connected
Outputs	
output[any]	Any number of input signals (each of the same size) may be connected.

Limitations

The signals connected to this model should all have the same size.

9.3.2 Block Diagram Non-Linear

DeadZone

Library

Signal\Block Diagram Non-Linear

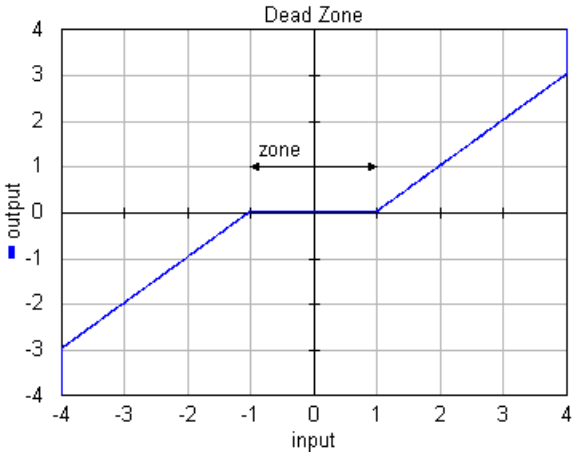
Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates zero output when the input is within a specified region, called the dead zone. Otherwise the output equals the input plus or minus half the zone.

output = input + zone/2; (input < -zone/2)
output = 0; (-zone/2 <= input <= zone/2)
output = input - zone/2; (input > zone/2)



Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
zone	Magnitude of the dead zone.

Limitations

Only single signals (i.e. signals of size 1) can be connected to this model.

Function-2DTable

Library

Signal\Block Diagram Non-Linear

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

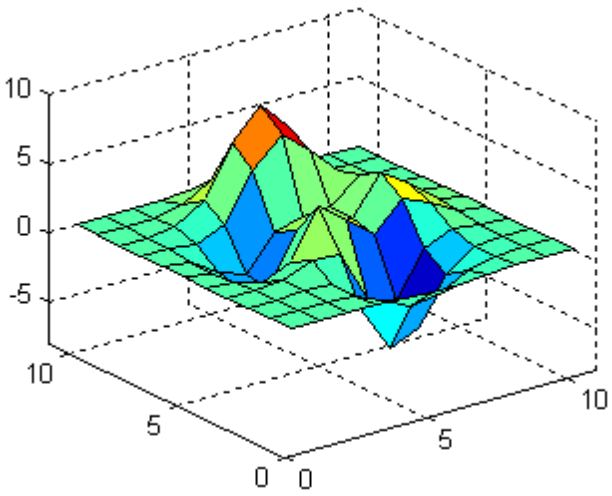
This model uses a two-dimensional table with data points to calculate the output *z = f(x,y)* as a function of the input *x* and *y*. The output *z* is calculated using linear interpolation between the table data points.

- The first row denotes the *x*-values and the first column denotes the *y*-values.
- The input data of the first column and first row needs to be monotonically increasing.
- Discontinuities are allowed, by providing the same input point twice in first row or first column of the table.
- Values outside of the table range, are computed by linear extrapolation of the last two points.

A table must be stored as an ASCII (text) file and should consist rows and columns of data. The first row consists of the *x*-values first column consist of the *y*-values. The row-column pairs are the corresponding *z*-values. Values must be separated by a space or a tab. No comment or other text may be part of the table-file. The filename of the input file can be specified using the complete path (e.g. c:\data\data.tbl). When no path is given, the file is assumed to be in the experiment directory.

Example

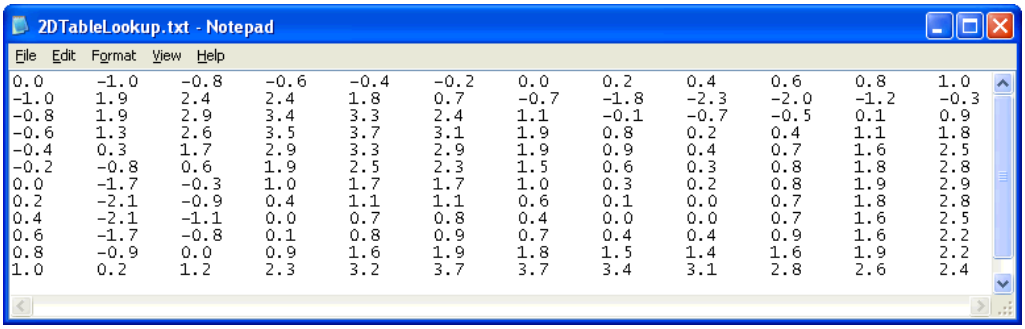
Suppose we want to use the well known Matlab *peaks* function:



This function can be put in a matrix form as:

0.0	-1.0	-0.8	-0.6	-0.4	-0.2	0.0	0.2	0.4	0.6	0.8	1.0
-1.0	1.9	2.4	2.4	1.8	0.7	-0.7	-1.8	-2.3	-2.0	-1.2	-0.3
-0.8	1.9	2.9	3.4	3.3	2.4	1.1	-0.1	-0.7	-0.5	0.1	0.9
-0.6	1.3	2.6	3.5	3.7	3.1	1.9	0.8	0.2	0.4	1.1	1.8
-0.4	0.3	1.7	2.9	3.3	2.9	1.9	0.9	0.4	0.7	1.6	2.5
-0.2	-0.8	0.6	1.9	2.5	2.3	1.5	0.6	0.3	0.8	1.8	2.8
0.0	-1.7	-0.3	1.0	1.7	1.7	1.0	0.3	0.2	0.8	1.9	2.9
0.2	-2.1	-0.9	0.4	1.1	1.1	0.6	0.1	0.0	0.7	1.8	2.8
0.4	-2.1	-1.1	0.0	0.7	0.8	0.4	0.0	0.0	0.7	1.6	2.5
0.6	-1.7	-0.8	0.1	0.8	0.9	0.7	0.4	0.4	0.9	1.6	2.2
0.8	-0.9	0.0	0.9	1.6	1.9	1.8	1.5	1.4	1.6	1.9	2.2
1.0	0.2	1.2	2.3	3.2	3.7	3.7	3.4	3.1	2.8	2.6	2.4

The first row and column denote the *x*-values and *y*-values and the other row-column pairs the corresponding peaks function values. This matrix can be entered in a text file:



The model 2DTableLookup.emx (*Examples**Signal Processing*) shows how to use this text file as a 2-D table.

Interface

Inputs	**Description**
input1	x-value
input2	y-value
Outputs	
output	z-value
Parameters	
filename	The filename of the table file.

Function-Absolute

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the absolute value of an input signal.

outp = abs(inp);

Interface

Inputs	**Description**
input	

Outputs
output

Function-Cosine

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the cosine of an input signal.

output = cos(input);

Interface

Inputs	**Description**
input	

Outputs
output

Function-DB

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model returns an output signal in decibel (dB):

*output = 20*log10(input);*

Interface

Inputs	**Description**
input	
Outputs	
output	

Function-Log

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model returns the natural logarithm of the input.

output = log(input);

Interface

Inputs	**Description**
input	
Outputs	
output	

Function-Power

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model raises the values of an input signal to the power p.

output = input^p;

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
p	

Function-Sign

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model returns the sign of the input. A zero input results in a zero output.

output = -1; (input < 0)
output = 0; (input = 0)
output = 1; (input > 0)

Interface

Inputs	**Description**
input	
Outputs	
output	

Function-Sine

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the sine of an input signal.

output = sin(input);

Interface

Inputs	**Description**
input	

Outputs

output

Function-Square

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in:** Block Diagrams.

Description

This model returns the square of the input signal.

output = input^2;

Interface

Inputs	**Description**
input	

Outputs

output

Function-SquareRoot

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the square root of an input signal.

outp = sqrt(inp);

Interface

Inputs	Description
input	

Outputs

output

Function-SquareRootSign

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the square root of an input signal with sign and multiplied by a parameter *p*.

*output = sign(input)*sqrt(inp)/p;*

Interface

Inputs	Description
input	

Outputs

output

Parameters

p

Function-SquareSign

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the square of an input signal with sign and multiplied by a parameter *p*.

```
output = p*sign(input)*input*input;
```

Interface

Inputs	Description
input	

Outputs
output

Parameters
p

Function-Table

Library

Signal\Block Diagram Non-Linear

Use

Domains: Continuous. Size: 1-D. Kind: Block Diagrams.

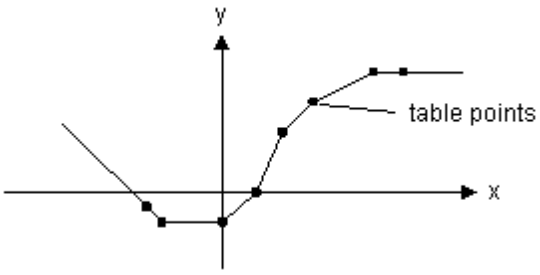
Description

This model uses a one-dimensional table with data points to calculate the output $y = f(x)$ as a function of the input x. The output y is calculated using linear interpolation between the table data points.

- The input data of the first column needs to be monotonically increasing.
- Discontinuities are allowed, by providing the same input point twice in the table.
- Values outside of the table range, are computed by linear extrapolation of the last two points.

table

-2.5	-0.5
-2	-1
0	-1
1	0
2	2
3	3.0
5	4
6	4.0



A table must be stored either as an ASCII (text) file or as a matrix inside the equation model. In both cases the table should consist two columns of data. The first column consists of the x-values and the second column of the corresponding y-values.

Reading data from a file

If the table is read from a file, each line of the file may only contain one x- and one corresponding y-value. The two values must be separated by a space or a tab. Each new set of x- and y-values must start on a new line. No comment or other text may be part of the table-file. The filename of the input file can be specified using the complete path (e.g. C:\data\data.tbl). When no path is given, the file is assumed to be in the experiment directory.

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
filename	The filename of the table file.

Reading data from a matrix

If the table is read from a matrix in an equation model, each row may only contain one x- and one corresponding y-value. The two values must be separated by a coma and each column must be separated by a semi-colon.

Note

Table interpolation can also be used directly in an equation model by using the function table.

Function-Tan

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the tangent of an input signal.

output = tan(input);

Note that the tangent goes infinity for input signals that are a multiple of pi.

Interface

Inputs	**Description**
input	
Outputs	
output	

Function-Truncation

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model rounds the input towards zero (i.e. it removes the fraction of the input).

output = trunc(input);

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	

p

SignalLimiter-Backlash

Library

Signal\Block Diagram Non-Linear

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model represents mechanical backlash between an input position and an output position.

output = 0; (input is within the dead zone)
output = input - zone/2; (input is increasing and out of the dead zone)
output = input + zone/2; (input is decreasing and out of the dead zone)

$$p.F = 0.5 \cdot k_2 \cdot \left(x - \sqrt{\left(-x - \frac{s}{2}\right) + \left(ep \cdot s\right)^2} + x + \sqrt{\left(x - \frac{s}{2}\right) + \left(ep \cdot s\right)^2} \right) + k_1 \cdot x + d_{1or\,2} \cdot p.v$$

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
zone	Size of the dead zone. *zone >= 0*
initial	Value of the output at time = 0. initial >= input(time = 0) - zone/2 *initial <= input(time = 0) + zone/2*

SignalLimiter-Hysteresis

Library

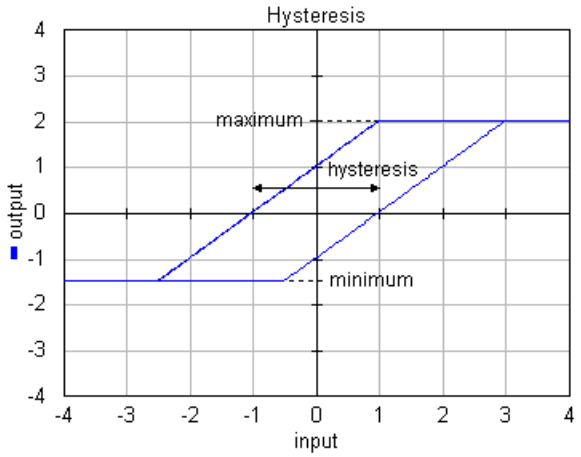
Signal\Block Diagram Non-Linear

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

The output signal of this submodel switches between *maximum* and *min*imum, depending on the sign of the input signal, with hysteresis applied. When traversing between *minimum* and *max*imum (dead zone), the rate of change is defined by the parameter *k*.



Interface

Inputs	Description
input	
Outputs	
output	
Parameters	
minimum	Minimum value of the output.
maximum	Maximum value of the output.
hysteresis	Magnitude of the hysteresis.
k	Rate of change in the dead zone.

SignalLimiter-JumpRateLimit

Library

Signal\Block Diagram Non-Linear

Use

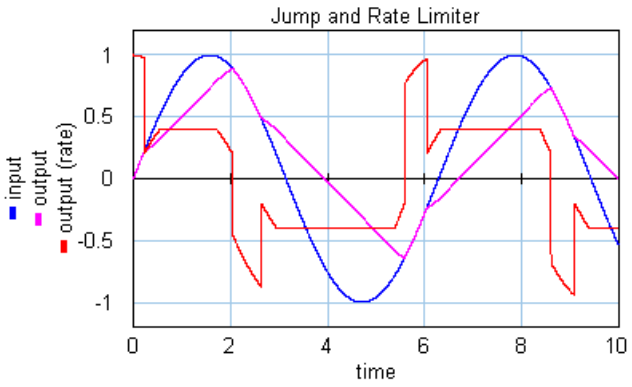
Domains: Discrete, Continuous. **Size:** 1-D. **Allowed in**: Block Diagrams.

Description

This model limits the input signal to a minimum and maximum jump and a minimum and maximum rate.

for minimum < input < maximum:
output = input
else
doutput/dt = dinput/dt; (minimumrate < dinput/dt < maximumrate)
doutput/dt = minimum; (dinput/dt < minimumrate)
output = maximum (dinput/dt > maximumrate)

En example is shown below where a sinusoidal input signal is limited to a minmum jump opf -0.2, a maximum jump of 0.2, a maximum rate of 0.4 and a minimum rate of -0.4.



Interface

Inputs	Description
input	

Outputs	
output	

Parameters	
minimum	Minimum jump of the output.
maximum	Maximum jump of the output.
minimumrate	Minimum rate of the output.
maximumrate	Maximum rate of the output.

SignalLimiter-Limit

Library

Signal\Block Diagram Non-Linear

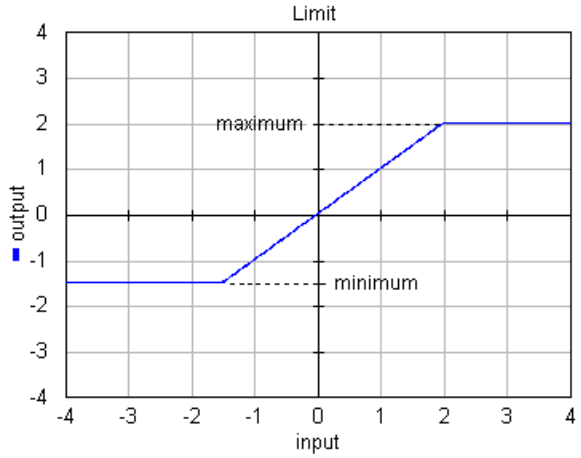
Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model limits the input signal to minimum and maximum bounds.

output = minimum; (input < minimum)
output = input; (minimum <= input <= maximum)
output = maximum (input > maximum)



Interface

Inputs	Description
input	
Outputs	
output	
Parameters	
minimum	Minimum value of the output.
maximum	Maximum value of the output.

SignalLimiter-PWM

Library

Signal\Block Diagram Non-Linear

Use

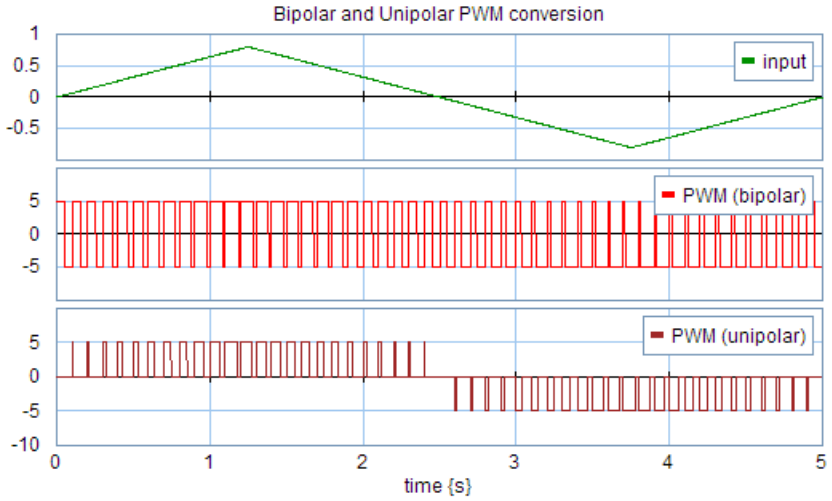
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

Pulse width modulation (PWM) is a powerful technique for controlling analog circuits with a processor's digital outputs. PWM is employed in a wide variety of applications, ranging from measurement and communications to power control and conversion. The output of a PWM block is a pulse train. The duty cycle, (the time that a pulse is *on* divided by the time the pulse is *off*) is proportional to the value of the input signal if the input is within the range < *-input_amp , +input_amp* >. For Bipolar PWM the output signal witches between a positive and a negative value. For Unipolar PWM the output signal witches between zero and positive or negative value.

input	bipolar PWM	Unipolar PWM
> input_amp	100% output_amp 0% -output_amp	100% output_amp 0% 0
input_amp	100% output_amp 0% -output_amp	100% output_amp 0% 0
0.5*input_amp	75% output_amp 25% -output_amp	50% output_amp 50% 0
0	50% output_amp 50% -output_amp	0
-0.5*input_amp	25% output_amp 75% -output_amp	50% 0 50% -output_amp
-input_amp	0% output_amp 100% -output_amp	0% 0 100% -output_amp
<-input_amp	0% output_amp 100% -output_amp	0% 0 100% -output_amp

An example is shown in the graph below. The parameter *input_amp* is equal to 1 and the parameter *max output* is equal to 5. The frequency of the PWM signal is 10 Hz.



Bipolar and Unipolar PWM conversion.

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
f	The modulating frequency [Hz].
input_amp	Maximum value of the input signal.
output_amp	The output value will switch between output_amp, 0 and -output_amp.

SignalLimiter-RateLimit

Library

Signal\Block Diagram Non-Linear

Use

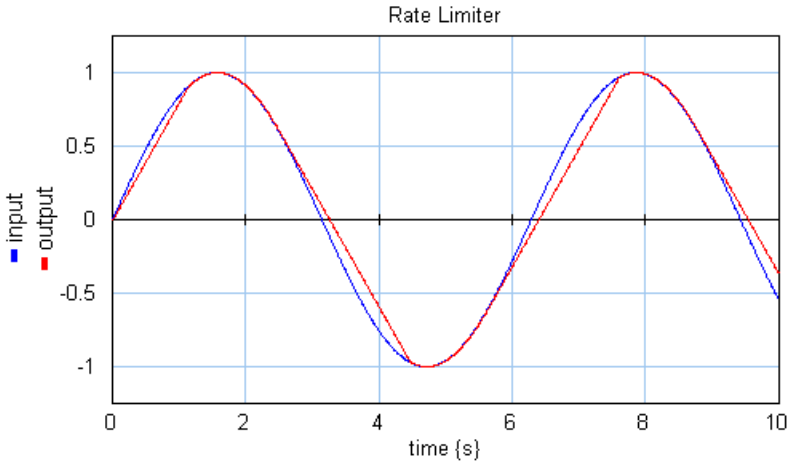
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model limits the input signal to a minimum and maximum rate.

doutput/dt = dinput/dt; (minimum < dinput/dt < maximum)
doutput/dt = minimum; (dinput/dt < minimum)
output = maximum (dinput/dt > maximum)

En example is shown below where a sinusoidal input signal is limited to a maximum rate of 0.4 and a minimum rate of -0.8.



The rate limiter uses a first order filter to approximate the derivative of the input. Below the cut-off frequency *fl* the approximation is accurate and the model behaves as a rate limiter. Above the cut-off frequency *fl* the model output weakens to zero. For high values of the cut-off frequency *fl*, the model will slow down simulation because it forces the integration method to take very small time-steps. A good trade-off for most models is a starting value of 100 [Hz].

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
minimum	Minimum rate of the output.
maximum	Maximum rate of the output.
fl	Cut-off frequency [Hz]

SignalLimiter-Relay

Library

Signal\Block Diagram Non-Linear

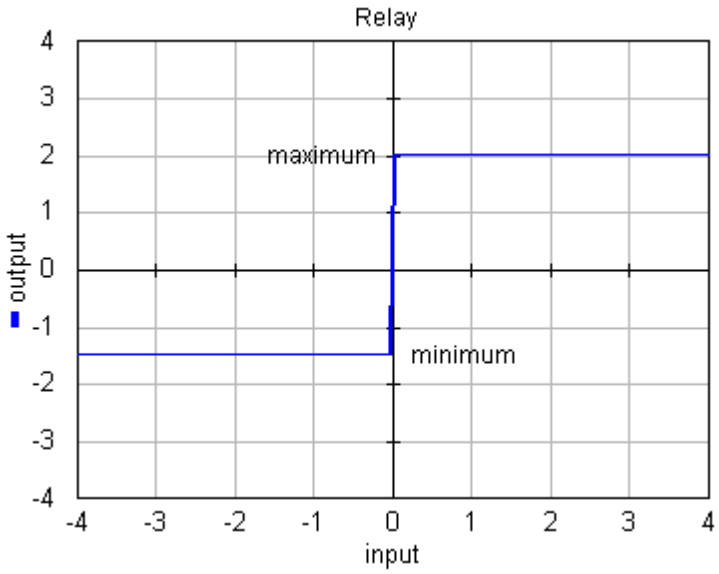
Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

The output signal of this submodel switches between a max and a min parameter using the sign of the input signal.

output = minimum; (input < 0)
output = maximum; (input >= 0)



Interface

Inputs	Description
input	
Outputs	
output	
Parameters	
minimum	Minimum value of the output.
maximum	Maximum value of the output.

SignalLimiter-RelayHysteresis

Library

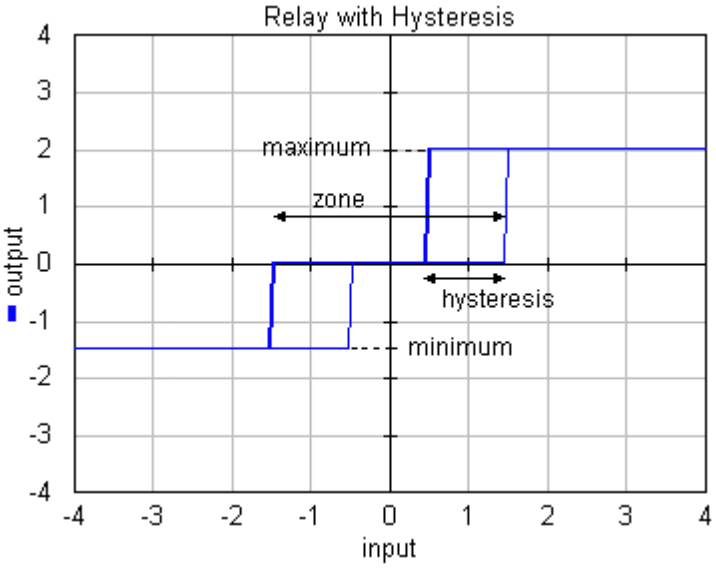
Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

The output signal of this submodel switches between max and min parameters, depending on the sign of the input signal, with hysteresis and dead zone applied.



Interface

Inputs	Description
input	
Outputs	
output	
Parameters	
minimum	Minimum value of the output.
maximum	Maximum value of the output.
zone	Magnitude of the dead zone.
hysteresis	Magnitude of the hysteresis.

switch-Break

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model switches between zero and the input signal, depending on the value of a condition input signal.

output = 0; (condition > 0)
output = input; (condition <= 0)

Interface

Inputs	Description
input_high	
input_low	
condition	Switching condition

Outputs

output

Switch-Default

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model switches between two input signals, depending on the value of a third input signal.

output = input_high; (condition >= 0)
output = inp_low; (condition < 0)

Interface

Inputs	Description
input_high	
input_low	

condition	Switching condition

Outputs

output

Switch-Make

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model switches between the input signal and zero, depending on the value of a condition input signal.

output = input; (condition > 0)
output = 0; (condition <= 0)

Interface

Inputs	**Description**
input_high	
input_low	
condition	Switching condition

Outputs

output

Switch-Maximum

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the maximum of two input signals.

output = input1; (input1 >= input2)
output = input2; (input1 < input2)

Interface

Inputs	Description
input1	
input2	

Outputs

output

Switch-Minimum

Library

Signal\Block Diagram Non-Linear

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model yields the minimum of two input signals.

output = input1; (input1 <= input2)
output = input2; (input1 > input2)

Interface

Inputs	Description
input1	
input2	

Outputs

output

9.3.3 Control

Controlled Linear Systems

ControlledLinearSystem

Library

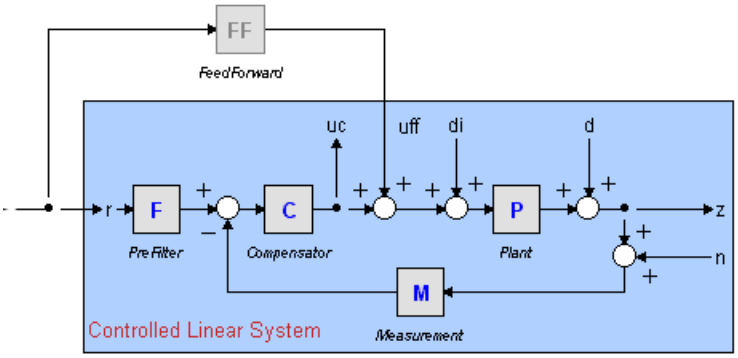
Signal\Control\Controlled Linear Systems

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a masked model which opens the Controller Design Editor when edited. Using this editor the settings for a controlled linear system can be entered.



Depending on the choices made, various inputs and outputs are available. In the picture above, the most complex model is shown. Read more about it in the Controller Design Editor topic.

Interface

Inputs	**Description**
r	Controller setpoint
d	Output Disturbance
n	Measurement Disturbance
di	Input Disturbance
uff	Feedforward Input
Outputs	**Description**
z	Plant output
uc	Feedforward Error Signal

Parameters

hidden	The parameters and initial values of this model are hidden for the user. Use the Controller Design Editor to edit this model (select the model and choose **Go Down**).

ControlledSystem

Library

Signal\Control\Controlled Linear Systems

Implementations

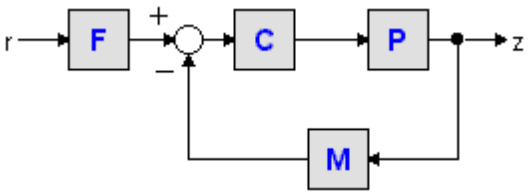
rz
rzff
rzffn
rzffnd
rzffnddi
rzn
rznd
rznddi

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Translation).

Description - rz

This model represents a controlled linear system described by graphically connected Linear systems:



These Linear Systems (F,C,P and M) can be edited using the Linear Systems Editor or replaced by drag and drop (any desired model) giving maximum flexibility.

Interface - rz

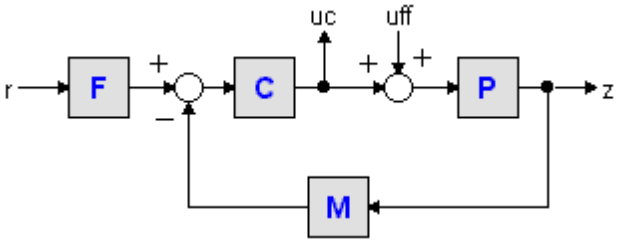
Inputs	**Description**
r	Controller setpoint
Outputs	**Description**
z	Plant output

Parameters

Depends on the implementation of the submodels.

Description - rzff

This model represents a controlled linear system described by graphically connected Linear systems:



These Linear Systems (F,C,P and M) can be edited using the Linear Systems Editor or replaced by drag and drop (any desired model) giving maximum flexibility.

Interface - rzff

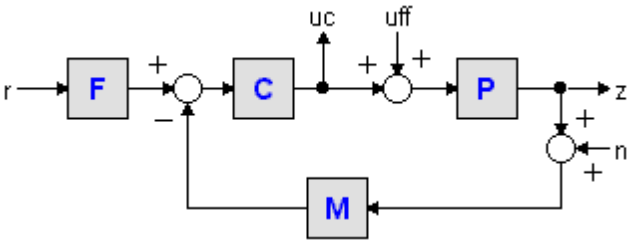
Inputs	**Description**
r	Controller setpoint
uff	Feedforward Input
Outputs	**Description**
z	Plant output
uc	Feedforward Error Signal

Parameters

Depends on the implementation of the submodels.

Description - rzffn

This model represents a controlled linear system described by graphically connected Linear systems:



These Linear Systems (F,C,P and M) can be edited using the Linear Systems Editor or replaced by drag and drop (any desired model) giving maximum flexibility.

Interface - rzffn

Inputs	**Description**

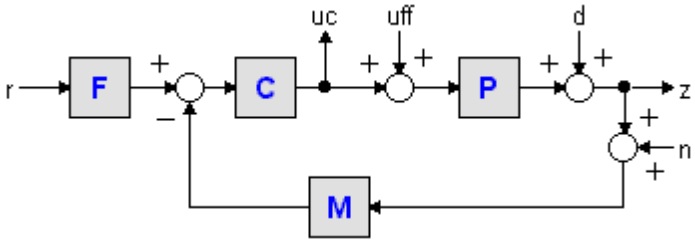
r	Controller setpoint
n	Measurement Disturbance
uff	Feedforward Input
Outputs	**Description**
z	Plant output
uc	Feedforward Error Signal

Parameters

Depends on the implementation of the submodels.

Description - rzffnd

This model represents a controlled linear system described by graphically connected Linear systems:



These Linear Systems (F,C,P and M) can be edited using the Linear Systems Editor or replaced by drag and drop (any desired model) giving maximum flexibility.

Interface - rzffnd

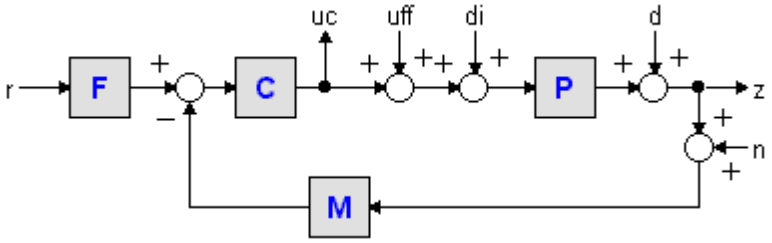
Inputs	**Description**
r	Controller setpoint
d	Output Disturbance
n	Measurement Disturbance
uff	Feedforward Input
Outputs	**Description**
z	Plant output
uc	Feedforward Error Signal

Parameters

Depends on the implementation of the submodels.

Description-rzffnddi

This model represents a controlled linear system described by graphically connected Linear systems:



These Linear Systems (F,C,P and M) can be edited using the Linear Systems Editor or replaced by drag and drop (any desired model) giving maximum flexibility.

Interface-rzffnddi

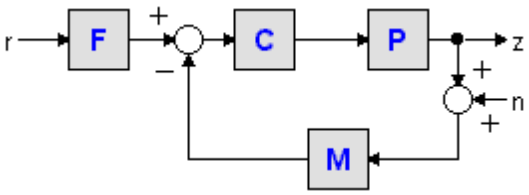
Inputs	**Description**
r	Controller setpoint
d	Output Disturbance
n	Measurement Disturbance
di	Input Disturbance
uff	Feedforward Input
Outputs	**Description**
z	Plant output
uc	Feedforward Error Signal

Parameters

Depends on the implementation of the submodels.

Description - rzn

This model represents a controlled linear system described by graphically connected Linear systems:



These Linear Systems (F,C,P and M) can be edited using the Linear Systems Editor or replaced by drag and drop (any desired model) giving maximum flexibility.

Interface - rzn

Inputs	**Description**
r	Controller setpoint

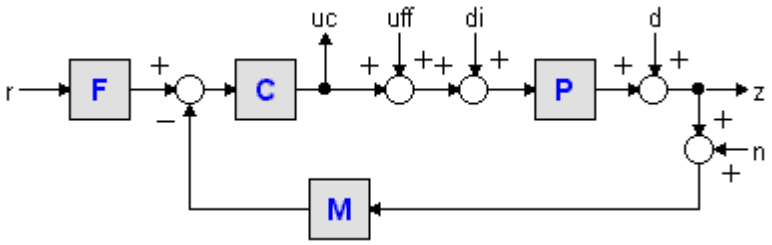
n	Measurement Disturbance
Outputs	**Description**
z	Plant output

Parameters

Depends on the implementation of the submodels.

Description - rznd

This model represents a controlled linear system described by graphically connected Linear systems:



These Linear Systems (F,C,P and M) can be edited using the Linear Systems Editor or replaced by drag and drop (any desired model) giving maximum flexibility.

Interface - rznd

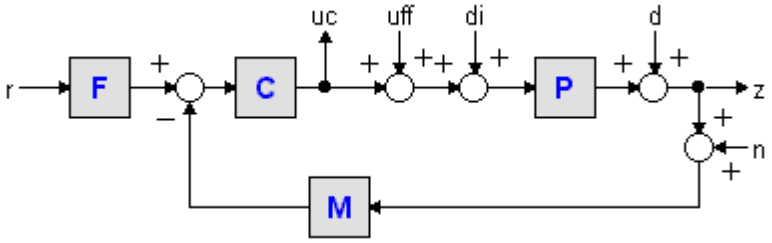
Inputs	**Description**
r	Controller setpoint
d	Output Disturbance
n	Measurement Disturbance
Outputs	**Description**
z	Plant output

Parameters

Depends on the implementation of the submodels.

Description - rznddi

This model represents a controlled linear system described by graphically connected Linear systems:



These Linear Systems (F,C,P and M) can be edited using the Linear Systems Editor or replaced by drag and drop (any desired model) giving maximum flexibility.

Interface - rznddi

Inputs	**Description**
r	Controller setpoint
d	Output Disturbance
n	Measurement Disturbance
di	Input Disturbance
Outputs	**Description**
z	Plant output
Parameters	
	Depends on the implementation of the submodels.

Neural Networks

BSplineNetwork

Library

Signal\Control\Neural Networks

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a masked model which opens the BSpline Editor when edited. Using this editor the settings for a B-Spline Network can be entered.

Interface

Inputs	**Description**
any	The user can defined any input.

error	The error signal: the difference between the desired output and the network output.
Outputs	**Description**
output	The network output
Parameters	
learning_rate	The learning rate
network_order	The order of the spline-interpolation.
online_learning	A boolean: Learn at each sample (True) or Learn after Leaving Spline (False).
regularization	Not yet supported.
regularization_width	Not yet supported.
load_weights	A boolean: Load weights before simulation.
save_weights	A boolean: Save weights after simulation.

MLPNetwork

Library

Signal\Control\Neural Networks

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a masked model which opens the MLP Editor when edited. Using this editor the settings for a MLP Network can be entered.

Interface

Inputs	**Description**
any	The user can defined any network input.
error	The error signal: the difference between the desired output and the network output.
Outputs	**Description**
any	The user can defined any network output.
Parameters	
nr_hidden	Number of hidden neurons.
learning_rate	The learning rate.
readWeights	A boolean: Load weights before simulation.
writeWeights	A boolean: Save weights after simulation.

PID Control

PIDControl

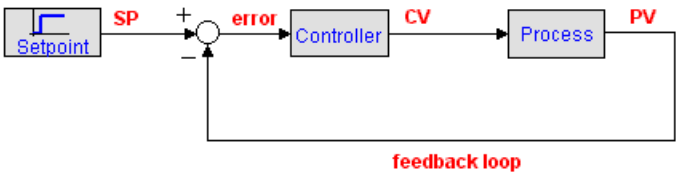
PID control stands for **P**roportional, **I**ntegral and **D**erivative control. PID controllers have been around since about 1940. Modern controllers perform the same functions as those, perhaps with a few embelishments and certainly more accurately, but the same functions nonetheless.

PID controllers are the best general purpose controllers to do the job. More sophisticated control algorithms will produce better performance when fitted to a specific process, but poorer performance results if the process changes. This sensitivity to process changes is called robustness, with more robust being less sensitive. The PID algorithm is an excellent trade-off between robustness and performance.

Although theory on PID-controllers is widespread and more or less uniform nowadays, in practice there are many algorithms and parameter settings. The 20-sim Control library, supports the most common algorithms.

Setpoint and other Variables

A standard control loop is shown in the figure below. A controller tries to steer a process in a way that minimizes the difference between a given setpoint and the output of that process. In other words, the controller tries to get the process output as close as possible to the given setpoint.



Setpoint

The **setpoint** (**SP**) is where the process output should match. It can be a static value (e.g. the desired temperature of a room) or a varying value (e.g. the desired position of a robot-tip).

The setpoint is where you would like the measurement to be. Error is defined as the difference between set-point and measurement.

Process

The **process** is the system that should be controlled. It is sometimes referred to as **plant** or **system**. The process should have has at least one impute to control its behavior and at least one output that is a measure of its behavior. Consider for example a gas heated house as process. The heater feed could be the process input and the measured room temperature the process output.

The process input is connected to the **controller output** or **controller variable** (**CV**). It is mostly the input for an actuator that can drive the process to a desired setpoint.

The **process output** is commonly known as **process variable** (**PV**), or measured variable (MV). Other names of the process output are system output and measurement.

Error

The **error** is defined as the setpoint minus the process output. It is a measure for how much the process deviates from the desired setpoint, and thus how much the controller should respond to get the process back to the desired setpoint.

Setpoint Weighting

For many controllers the controller input is equal to the error signal. Some controllers, however use specialized inputs for their proportional, integral and derivative parts (setpoint weighting). They have two inputs, one for the setpoint and one for the measured variable and are therefore also described as 2-DOF controllers.

Proportional Control

Gain

When we talk about the proportional action of a controller, we generally refer to the **proportional gain**. The action means that the controller moves in proportion to the error between setpoint (SP) and process output (PV):

*controller output = K*error = K*(SP - PV)*

where the gain is denoted by the parameter K. Many terms have been used by different manufacturers to designate this action. It has been called proportional gain, **gain**, throttling band, sensitivity and **proportional band**.

Proportional Band

In practice, the controller output is limited, either by its own limitations or by the limitations of the corresponding actuator. Let umax and umin denote the minimum and maximum output of the controller. The proportional band of the controller is then defined as:

$$PB = \frac{u_{max} - u_{min}}{K} \cdot 100\%$$

In the ideal case, a controller can have an unlimited output. The **proportional band** (**PB**) is then defined as:

$$PB = \frac{1}{K} \cdot 100\%$$

This definition of proportional band is often used instead of the controller gain. The value is expressed in percent (%).

Direct Acting / Reverse Acting

Suppose a process with a controller output CV and a process output PV is in steady state and kept to a certain setpoint SP. For proportional control the following equations can be found:

PV = P(CO)
*CV = K*error = K*(SP - PV)*

where P is the process function that yields a measured variable as a result of the controller output. If the controller output decreases as the measured variable increases the controller is said to be **direct acting**. If the controller output increases as the measured variable increases, then it is called **reverse acting**. In other words, the controller is direct acting if the **gain K is positive** and reverse acting if the **gain K is negative**.

This is not only valid for proportional control, but for all PID controllers in the 20-sim library. If you need a reverse acting controller, simply use a negative gain.

Offset

If you look at the equation for the proportional gain:

*CV = K*error = K*(SP - PV)*

you will notice that there has to be an error to produce a controller output. This means that with proportional control only, there will always be a small offset between the setpoint and the measured variable. To remove this offset, integral control has to be used.

Integral Control

With **integral** action, the controller output is proportional to the amount of time the error is present. Integral action eliminates offset that remains when proportional control is used.

*controller output = (1/Ti)*int(error)*

where the parameter Ti is called the **integral time**. Integral action is also know as reset and the parameter Ti as reset time.

Integral action gives the controller a large gain at low frequencies that results in eliminating offset. Integrals give information concerning the past. That is why integrals are always late. Integrals provide stability but have a tendency to get stuck in the past. In most controllers the proportional and integral action are combined. The output of the combined proportional and integral action (in s-domain) is then:

$$U = K \cdot \left(1 + \frac{1}{s \cdot Ti}\right) \cdot E$$

with E equal to SP - PV.

Derivative Control

With *derivative* action, the controller output is proportional to the rate of change of the measurement or error. Some manufacturers use the term rate or pre-act instead of derivative. Derivative, rate and pre-act are the same thing. The controller output is calculated by the rate of change of the error with time.

*controller output = Td*d(error)/dt = Td*d(SP - PV)/dt*

where the parameter Td is called ***derivative time***. Derivative control is mathematically the opposite of integral action, but while we might have an integral-only controller, we would never have a derivative-only controller. The reason for this is that derivative control only knows the error is changing. It does not know what the setpoint actually is.

Derivative action has the potential to improve performance when sudden changes in measured variable occur, but is should be used with care. It is mostly a matter of using enough, not too much.

Derivative Gain Limitation

In most commercial processes sudden changes in process output may appear. In most cases a sudden change in the slope of such a process output cannot be avoided at all times. Using such a process output in controllers with pure derivative action, would lead to unwanted steps in the controller output. Moreover, high frequency noise in the measured signals may lead to unwanted large outputs of the controller.

To prevent this unwanted effect, the derivative action can be filtered by a first-order system with time constant *Td/N*.

$$s \cdot K \cdot Td \rightarrow \frac{s \cdot K \cdot Td}{1 + s \cdot \frac{Td}{N}}$$

This approximation acts as a derivative for low-frequency signal components. The gain, however, is limited to *K*N.* This means that high-frequency measurement noise is amplified at most by a factor *KN*. This is why the parameter *N* is called the *derivative gain limitation*. Typical values of N are 8 to 20. Sometimes the reciprocal value of N is used, mostly with the name beta (beta = 1/N).

PID Controller Types

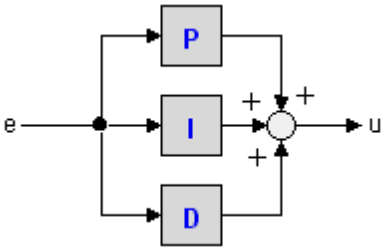
In literature various PID controller laws and types have been described. In industry two types prevail: the parallel form and the series form.

Parallel Form

A PID controller in parallel form (also known as standard form, ISA form or non-interacting form), has the control equation:

$$U = K \cdot \left(1 + \frac{1}{s \cdot Ti} + s \cdot Td\right) \cdot E$$

The controller actions (P, I and D) act independently as can be seen in the corresponding block diagram representation.

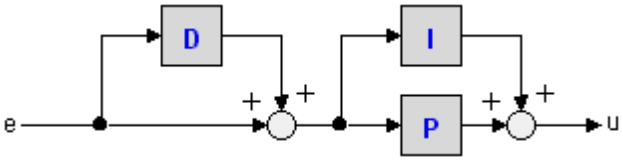


Series Form

A PID controller in series form (also known as interacting form), has the control equation:

$$U = K \cdot \left(1 + \frac{1}{s \cdot Ti}\right)\left(1 + s \cdot Td\right) \cdot E$$

The controller actions (P, I and D) act dependently as can be seen in the corresponding block diagram representation.



Note
P and PI controllers are the same in series and parallel form.

Setpoint Weighting

Standard PID controller equations operate on the error signal (Error Based):

$$P - action : E_p = SP - PV$$
$$I - action : E_I = SP - PV$$
$$D - action : E_D = SP - PV$$

A more flexible structure is obtained by treating the setpoint (SP) and the process output (PV) separately (Setpoint Weighting):

$$P-action: E_P = b \cdot SP - PV$$
$$I-action: E_I = SP - PV$$
$$D-action: E_D = c \cdot SP - PV$$

The I-action always operates on the error to insure that the error between setpoint and process output will be minimized.

In most commercial controllers, the parameters b an c are chosen 1 or 0:

b = 1 -> proportional control using the error: SP - PV (error feedback).
b = 0 -> only proportional control on the process output (PV).
c = 1 -> derivative control using the error: SP - PV (error feedback).
c = 0 -> only derivative control on the process output (PV).

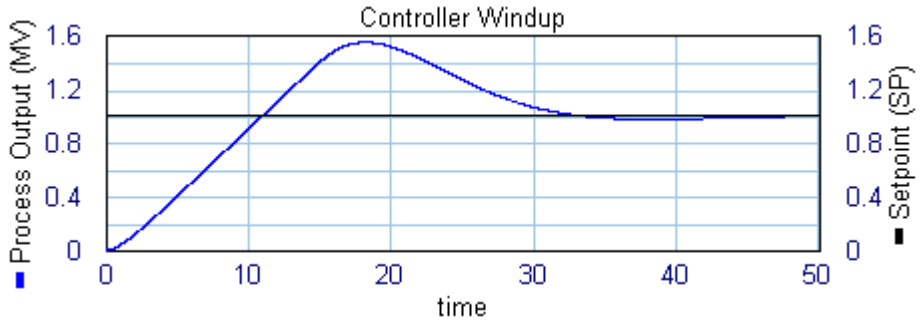
Anti-Windup

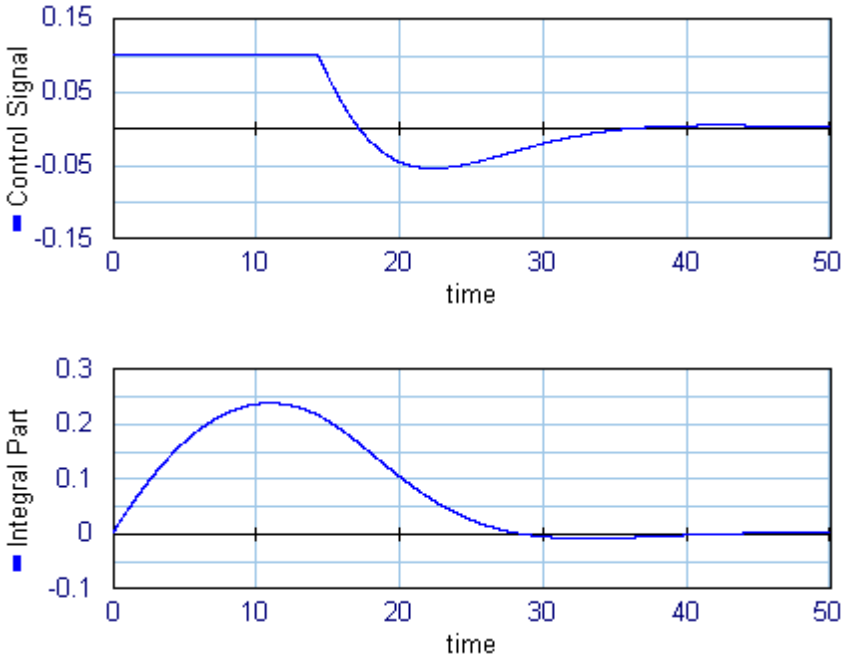
All actuators have physical limitations, a control valve cannot be more than fully open or fully closed, a motor has limited velocity, etc. This has severe consequences for control. Integral action in a PID controller is an unstable mode. This does not cause any difficulties when the loop is closed. The feedback loop will, however, be broken when the actuator saturates because the output of the saturating element is then not influenced by its input. The unstable mode in the controller may then drift to very large values. When the actuator desaturates it may then take a long time for the system to recover. It may also happen that the actuator bounces several time between high and low values before the system recovers.

Integrator windup

Integrator windup is illustrated in the figure below, which shows simulation of a system where the process dynamics is a saturation at a level of ±0.1 followed by a linear system with the transfer function:

$$G(s) = \frac{1}{s(s+1)} \cdot \mathrm{x}$$





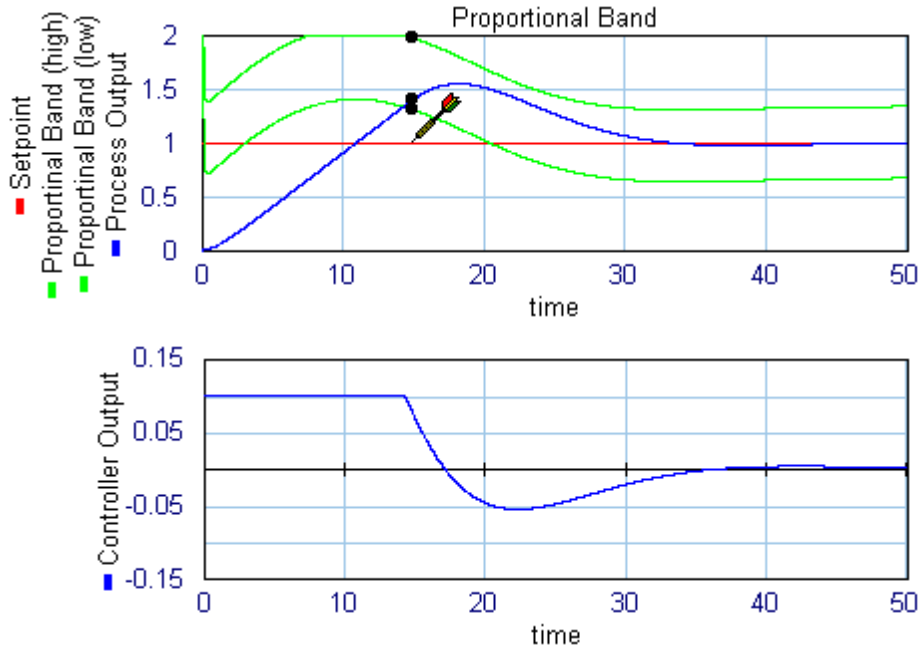
Because of the saturation in the actuator, the control signal saturates immediately when the step is applied. The control signal then remains in saturation level and the feedback is broken. The integral part continues to increase because the error (SP - PV) is positive. The integral part starts to decrease when the process output (PV) has become larger than the setpoint (SP), but the process output remains saturated because of the large integral part. Slowly the process output decreases towards the setpoint.

The net effect is that there is a large overshoot. This phenomenon is called "integrator windup". A good insight in windup is found when looking at the proportional band.

Proportional Band and Windup

The values of the process output that correspond to the minimum and maximum output are denoted as ymax and ymin. The controller operates linearly only if the process output is in the range (ymax , ymin). The controller output saturates when the process output is outside this band. A good insight into the windup problem is obtained by investigating the range (ymax , ymin). All 20-sim controller models in parallel form with anti-windup scheme, have the extra variables PB_high and PB_low which are equal to ymax and ymin.

An illustration of the proportional band is given below. The same linear system is used with the same controller. As can be seen, the actuator is saturated from t = 0 until t = 14. At t = 14 the process output enters the range (PB_high , PB_low) and controller feedback is regained.

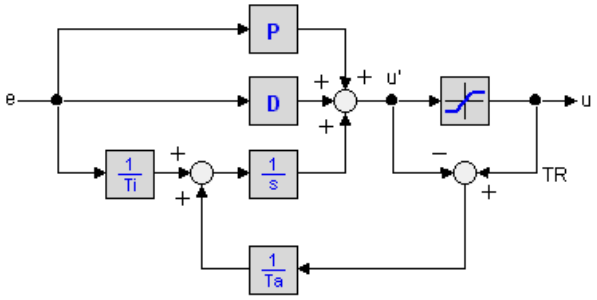


Anti-Windup

Integrator windup can be avoided, by making sure that the integral is kept to a proper value when the actuator saturates, so that the controller is ready to resume action, as soon as the control error changes. This anti-windup scheme is known as tracking or back calculation.

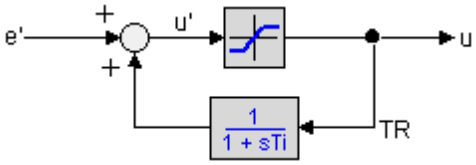
parallel form

As well known form of tracking is linear feedback anti windup. It is shown in the figure below (parallel form). The actuator is represented by a signal limiter. The difference between actuator input and output (TR) is fed back to the integrator through the gain 1/ Ta. As soon as the limiter saturates, this signal becomes non-zero and prevents the integrator from winding up. The tracking time constant Ta can be used to tune then amount of anti windup.



series form

The same scheme can also be used for controllers with a series form. A diagram is shown below.

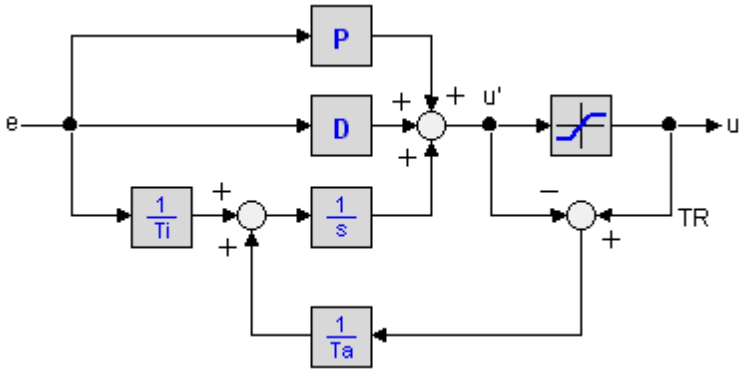


Tracking Time constant

To prevent the integrator from saturating, the tracking time constant must be chosen small. Too small values, however decrease the controller performance. As a rule of thumb Åström suggested to choose the tracking time constant Td <= Ta <= Ti. Some authors prefer a good controller performance and suggest to choose Ta = Ti.

External Tracking

As long as the actuator output is equal to the controller output, anti-windup scheme will not be activated and the controller is in normal operation (*control mode*). When the actuator saturates, the anti-windup scheme will be activated and prevent the controller output from wandering away. In effect the anti-windup scheme matches the controller output and actuator output. This is why the actuator output is also known as the tracking signal (TR).

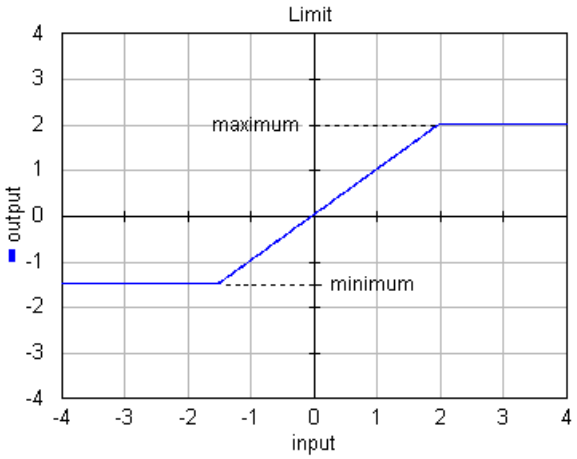


When an external actuator signal is used (external tracking signal) is is important to compensate for the actuator gain. Otherwise the tracking signal is not equal to the controller output, during normal operation and the anti-windup scheme is activated.

Actuator Model

Anti-windup schemes are based on the difference between actuator input (controller output) and actuator output. These signals are not always available. Therefore an actuator model can be used inside the controller to yield this difference. In the library models, a signal limiter is used the actuator model:

output = minimum; (input < minimum)
output = input; (minimum <= input <= maximum)
output = maximum (input > maximum)



Initial Output

The output of PID controllers at time 0 depends on the given setpoint (SP), process output (PV) and control law. In general the process will not be at rest (PV <> 0) and the setpoint will not be equal to the process output (SP <> 0). This will result in an undesired step in the controller signal at startup. Even if the setpoint is equal to the process output (SP = PV) at startup, the use of setpoint weighting may result in a step in the control signal:

$$P-\text{action}: E_P = b \cdot SP - PV$$
$$I-\text{action}: E_I = SP - PV$$
$$D-\text{action}: E_D = c \cdot SP - PV$$

Let PV be a non zero value and b or c equal to zero. The P or D -action will then lead to a step in the control signal.

To avoid such a step, the *initial output* of some PID-controllers can be set to any value. The slope of the the control output of these controllers at startup is equal to zero.

Initial output for with delays

Controllers with manual output at start-up, use the process output at t = 0, PV(0), to calculate an internal offset that compensates for the initial controller output. Care should be taken when delays are available between the process output and the controller input. Due to these, the measured process output at t = 0 in the controller (PV(0)') may not be the same as the real process output at t = 0 (PV(0)). As a result, a wrong offset will be calculated and the initial output will not have the desired value. To avoid this, all initial values of delay elements between the process and controller should be chosen properly (e.g. equal to the initial process output) !

Commercial Controllers

According to b Åström the table below summarizes the properties of some commercial controllers.

Controller	Structure	Setpoint Weighting b	Setpoint Weighting c	Derivative Gain Limitation N	Sampling Period s
1. Allen Bradley PLC 5	P	1.0	1.0	None	Load dependent
2. Baliley Net 90	P	0.0 or 1.0	0.0 or	10	0.25
3. Fisher Controls Provox	S	1.0	1.0	8	0.1, 0.25 or 1.0
4. Fisher Controls DPR	S	0.0	0.0	8	0.2
900,910	S	1.0	0.0	none	0.1
5. Fisher Porter Micro DCI	S	1.0	0.0 or	10	0.25
6. Foxboro Model 761	S	1.0	1.0	8	0.33, 0.5 or 1.0
7. Honeywell TDC	S	1.0	0.0	1-30	0.1
8. Moore Products Type	S	0.0	1.0	8	0.2
352	S	0.0 or 1.0	0.0	17 or 20	0.25
9. Alfa Laval ECA40, ECA	S	1.0	0.0	3.3 – 10	0.2
400	S	1.0	0.0	none	0.036 – 1.56

10. Taylor Mod 30 11. Toshiba TOSDIC 200 12. Turnbull TCS 6000 13. Yokogawa SLPC	P	0.0 or 1.0	1.0 1.0 0.0 or 1.0	10	0.1

Literature

1. K.J. Astrom, T. Hagglund, Pid Controllers, Instrument Society of America; ISBN: 1556175167, (1995), (avalaible at www.amazon.com)
2. B.G. Liptak, Instrument Engineers' Handbook, Volume 1: Process Measurement, CRC Pr; ISBN: 0801981972, (2000), (avalaible at www.amazon.com).
3. B.G. Liptak, Instrument Engineers' Handbook, Volume 2: Process Control, CRC Pr; ISBN: 0801982421, (2000), (avalaible at www.amazon.com).
4. G.K. McMillan, D.M. Considine, Process/Industrial Instruments and Controls Handbook, McGraw Hill Text, ISBN: 0070125821, (1999), (avalaible at www.amazon.com).
5. B. Wittenmark, K.J. Astrom, Computer-Controlled Systems: Theory and Design, Prentice Hall; ISBN: 0133148998, (1996), (avalaible at www.amazon.com).

Expert information can also be found in the sci.eng.control newsgroup. An exelent frequently asked questions list can be found on the internet pages of Ron Graham (http://www.tcnj.edu/~rgraham/PID-tuning.html).

Continuous

Naming Conventions

Two types of PID-controllers are available in this library. The first type of controllers use an **error** input and the second type of controllers use separate Setpoint (**SP**) and Measured Variable (**MV**) input.

Error Input

PID-Controllers with the error as a direct input signal, are all in series form. The name simply denotes the type of controller: P, PI, PD or PID. Two versions of each controller exists: continuous time and discrete time. They can be found in the subdirectories `Continuous` and `Discrete`.

Ports

All controllers have one input ports and one output port:

1. *error*: The error between setpoint and measured variable.
2. *output*: The controller output.

Parameters

Depending on the type of controller, the following parameters can be used:

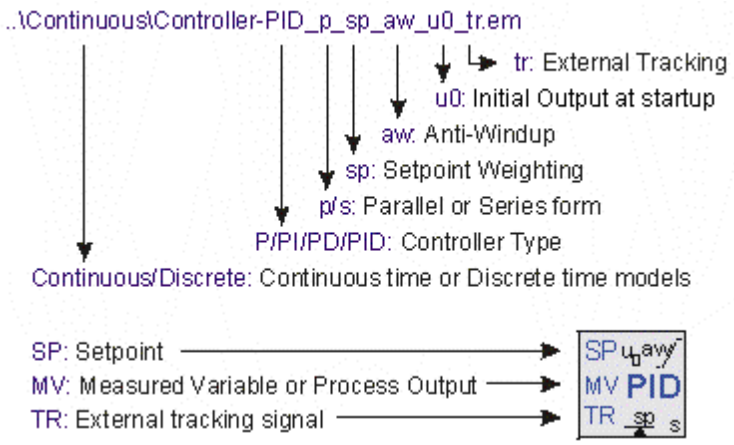
1. *kd*: The proportional gain of the controller.
2. *tauD*: The derivative time constant.
3. beta: The reciprocal derivative gain limitation.
4. *tau*I: The integral time constant.

Controller Wizard

A special form of controllers with error input is generated by the ControllerWizard.emx submodel. When you select this model and click **Go Down** the Filter Editor is opened, allowing you to specify controllers with frequency oriented parameters.

Separate Input

Controllers with separate inputs come in many forms. The naming denotes the type of controller and the options that are available. The type and options are also available in the model icons.



1. Two versions of each controller exists: continuous time and discrete time. They can be found in the subdirectories *Continuous* and *Discrete*. Continuous-time controllers have icons with blue text and discrete-time controllers have icons with green text.
2. The first characters denote the type of controller: *P*, *PD*, *PI* or *PID.*
3. The character p or s denote the form: parallel (*p*) or series (*p*).
4. If setpoint weighting is used, the term *sp* is added.
5. If an anti-windup scheme is incorporated, the term *aw* is added.
6. If the output at startup can be set manually, the term *u0* is added.
7. If an external tracking signal is used, the term *tr* is added.

Ports

All controllers have two inputs ports and one output port:

1. *SP*: The setpoint
2. *MV*: The measured variable (also called process variable).
3. *output*: The controller output.

Parameters

Depending on the type of controllers, the following parameters can be used:

1. *K*: The proportional gain of the controller.
2. *Td*: The derivative time constant.
3. *N*: The derivative gain limitation.
4. *Ti*: The integral time constant.
5. *b*: The setpoint weighting constant for the proportional part of the controller.
6. *c*: The setpoint weighting constant for the derivative part of the controller.
7. *Ta*: The tracking time constant of the anti-windup scheme.
8. *minimum*: The minimum controller output representing actuator saturation.
9. *maximum*: The maximum controller output representing actuator saturation.
10. *output_initial*: The controller output at start-up.

Initial Values

Although most controllers have one or more internal states, the initial values of these states do not need to be set manually. You use the default values (0) at the start of a simulation. To get a desired output of the controller at startup , the parameter output_initial should be used.

Discrete Controllers

The discrete controllers that are available in the library are directly derived from their continuous counterparts, using approximation by backward differences. These discrete models have identical behavior, as long as the sample time is not chosen too low. Discrete loops in 20-sim are automatically detected and assigned a default sampletime. You can change the sample time in the Run Properties Editor (in the **Simulator** from the **Properties** menu select the **Simulation** command).

Variables

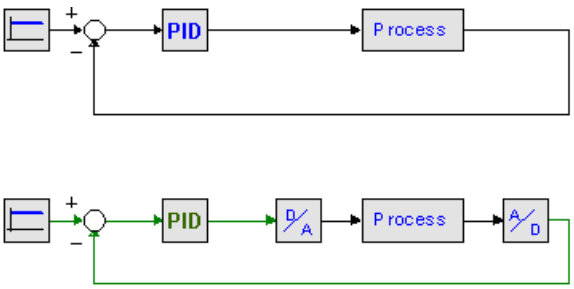
Some PID controllers (parallel form with anti-windup) have two internal variables that are of interest. PB_high and PB_low. These parameters are the upper and lower bound of the proportional band.

Controller Use

All controllers must can be used as shown in the figure below. The upper part shows a continuous time controller and the bottom part shows a discrete time controller.

Error Input

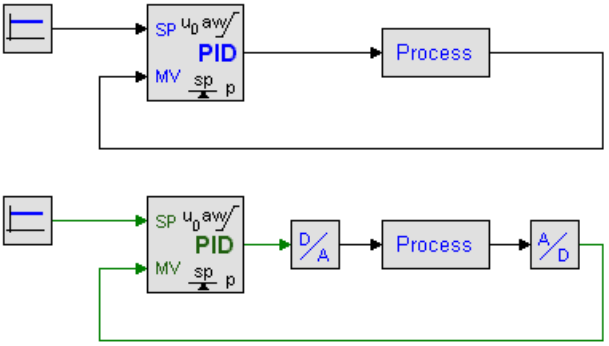
1. An error signal is generated with a plus minus element. Its serves as an input of the controller.
2. The controller output is connected with the process input.



As shown in the figure, analog to digital convertors and digital to analog convertors should be used, when connecting a discrete-time controller to a continuous-time process.

Separate Input

1. A setpoint signal is connected with the **SP** port of the controller.
2. The process output is connected with the **PV** port of the controller.
3. The controller output is connected with the process input.



As shown in the figure, analog to digital convertors and digital to analog convertors should be used, when connecting a discrete-time controller to a continuous-time process.

ControllerWizard

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete **Size**: 1-D. **Kind**: Block Diagrams.

Description

A special form of controllers with error input is generated by the `ControllerWizard.emx` submodel. When you select this model and click **Go Down** the Filter Editor is opened, allowing you to specify controllers with frequency oriented parameters.

Interface

Inputs	**Description**
u	The error input signal: plant-output minus set point.
Outputs	
y	Output signal: input for the plant.
Parameters	
	Look in the Filter Editor helpfile to find a detailed description of the possible controller types and their parameters.

Controller-P

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a P controller. The output of this controller (in s-domain) is:

$$U = K \cdot E$$

with E equal to SP - PV.

Interface

Inputs	**Description**
SP	Setpoint.
PV	Process output.

Outputs

CV	Controller output: input for the process.

Parameters

K	Proportional gain.

Controller-PD_p

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PD controller in parallel form. The output of this controller (in s-domain) is:

$$U = K \cdot \left(1 + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Initial Values	
yes	Should be left default!

Controller-PD_s

Library
Signal\Control\PID Control\Continuous

Use
Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description
This is a PD controller in series form. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Initial Values	
yes	Should be left default!

Controller-PI

Library
Signal\Control\PID Control\Continuous

Use
Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description
This is a PI controller. The output of this controller (in s-domain) is:

$$U = K \cdot \left(1 + \frac{1}{s \cdot Ti}\right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Ti	Integral time constant (Ti > 0).
Initial Values	
yes	Should be left default!

Controller-PI_sp

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV)\right)$$

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.

Parameters

K	Proportional gain.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.

Initial Values

yes	Should be left default!

Controller-PI_sp_aw

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) \right)$$

An anti-windup scheme has been added to this controller.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.

Outputs

output	Controller output: input for the process.

Parameters

K	Proportional gain.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.

Initial Values

yes	Should be left default!

Interesting Variables

PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PI_sp_aw_u0

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.

Outputs

output	Controller output: input for the process.

Parameters

K	Proportional gain.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
output_initial	The controller output at start-up.

Initial Values

yes	Should be left default!

Interesting Variables

PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PI_sp_aw_u0_tr

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up. An external tracking signal must be supplied to activate the anti-windup scheme.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
TR	Tracking signal.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
Ta	Tracking time constant.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_p

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form. The output of this controller (in s-domain) is:

$$U = K \cdot \left(1 + \frac{1}{s \cdot Ti} + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
Initial Values	
yes	Should be left default!

Controller-PID_p_sp.

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot (c \cdot SP - MV) \right)$$

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_p_sp_aw

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot (c \cdot SP - MV) \right)$$

An anti-windup scheme has been added to this controller.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_p_sp_aw_u0

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot (c \cdot SP - MV) \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_p_sp_aw_u0_tr

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot (c \cdot SP - MV) \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up. An external tracking signal must be supplied to activate the anti-windup scheme.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
TR	Tracking signal.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_s

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form. The output of this controller (in s-domain) is:

$$U = K \cdot \left(1 + \frac{1}{s \cdot Ti}\right)\left(\frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}}\right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
Initial Values	
yes	Should be left default!

Controller-PID_s_sp

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\left(b + \frac{1}{s \cdot Ti} \right) \cdot \frac{1 + s \cdot c \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot SP - \left(1 + \frac{1}{s \cdot Ti} \right) \frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot MV \right)$$

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Initial Values	
yes	Should be left default!

Controller-PID_s_sp_aw

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\left(b + \frac{1}{s \cdot Ti} \right) \cdot \frac{1 + s \cdot c \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot SP - \left(1 + \frac{1}{s \cdot Ti} \right) \frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot MV \right)$$

An anti-windup scheme has been added to this controller.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
Initial Values	
yes	Should be left default!

Controller-PID_s_sp_aw_u0

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\left(b + \frac{1}{s \cdot Ti} \right) \cdot \frac{1 + s \cdot c \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot SP - \left(1 + \frac{1}{s \cdot Ti} \right) \frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot MV \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!

Controller-PID_s_sp_aw_u0_tr

Library

Signal\Control\PID Control\Continuous

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\left(b + \frac{1}{s \cdot Ti} \right) \cdot \frac{1 + s \cdot c \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot SP - \left(1 + \frac{1}{s \cdot Ti} \right) \frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot MV \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up. An external tracking signal must be supplied to activate the anti-windup scheme.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
TR	Tracking signal.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!

P

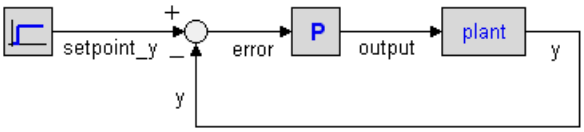
Library
Signal\Control\PID Control\Continuous

Use
Domains: Continuous, Discrete **Size**: 1-D. **Kind**: Block Diagrams.

Description
This is a P controller. The output of this controller is:

$$ouput = kp \cdot error$$

with the input of the controller equal to the error and the output of the controller used as the plant input:



Interface

Inputs	Description
error	Input signal: plant-output minus set point.
Outputs	
output	Output signal: input for the plant.
Parameters	
kp	Proportional gain.

PD

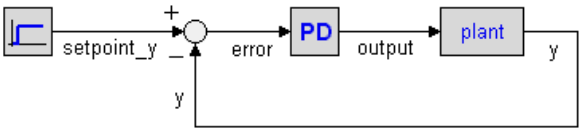
Library
Signal\Control\PID Control\Continuous

Use
Domains: Continuous, Discrete **Size**: 1-D. **Kind**: Block Diagrams.

Description
This is a PD controller in series form. The transfer function of an ideal PD-controller is:

$$G(s) = \frac{kp\,(1 + s \cdot tauD)}{1 + s \cdot beta \cdot tauD}$$

with the input of the controller equal to the error and the output of the controller used as the plant input:



Interface

Inputs	Description
error	Input signal: plant-output minus set point.
Outputs	
output	Output signal: input for the plant.
Parameters	
kp	Proportional gain.
tauD	Derivative gain (tauD <> 0).
beta	Tameness constant (0 < beta << 1).
Initial Values	
state_initial	output(0) = state_initial - kp*error(0)*/beta

PI

Library

Signal\Control\PID Control\Continuous

Use

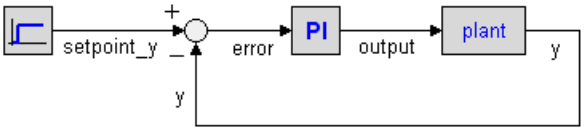
Domains: Continuous, Discrete **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller. The transfer function of the controller is:

$$G(s) = kp + \frac{kp}{s \cdot tauI}$$

with the input of the controller equal to the error and the output of the controller used as the plant input:



Interface

Inputs	Description
error	Input signal: plant-output minus set point.
Outputs	
output	Output signal: input for the plant.
Parameters	
kp	Proportional gain.
tauI	Integral gain (tauI <> 0).
Initial Values	
state(0)	output(0) = state(0) - kp*error(0)

PID

Library

Signal\Control\PID Control\Continuous

Use

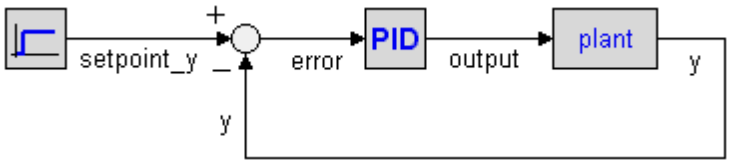
Domains: Continuous, Discrete **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form. The transfer function is:

$$G(s) = kp \cdot \left(1 + \frac{1}{s \cdot tauI}\right)\left(\frac{1 + s \cdot tauD}{1 + s \cdot beta \cdot tauD}\right)$$

with the input of the controller equal to the error and the output of the controller used as the plant input:



Interface

Inputs	**Description**
error	Input signal: plant-output minus set point.
Outputs	
output	Output signal: input for the plant.
Parameters	
kp	Proportional gain.
tauD	Derivative gain (tauD <> 0).
tauI	Integral gain (tauD <> 0).
beta	Tameness constant (0 < beta << 1).
Initial Values	
pdstate(0)	pdout(0) = pdstate(0) - kp*error(0)*/beta
pistate(0)	output(0) = pistate(0) - pdout(0)

Discrete

Naming Conventions

Two types of PID-controllers are available in this library. The first type used an error input and the second type of controller use separate setpoint and Measured Variable input. Both types use the same internal description, so the use is just a question of flavor.

Error Input

PID-Controllers with the error as a direct input signal, are all in series form. The name simply denotes the type of controller: P, PI, PD or PID. Two versions of each controller exists: continuous time and discrete time. They can be found in the subdirectories `Continuous` and `Discrete`.

Ports

All controllers have one input ports and one output port:

1. *error*: The error between setpoint and measured variable.
2. *output*: The controller output.

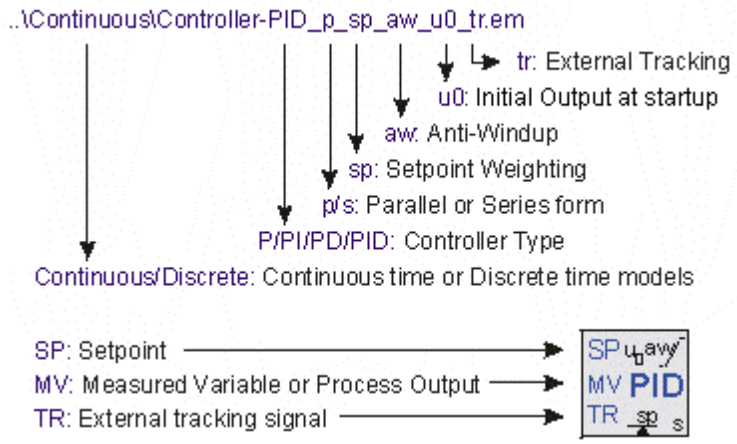
Parameters

Depending on the type of controller, the following parameters can be used:

1. *kd*: The proportional gain of the controller.
2. *tauD*: The derivative time constant.
3. beta: The reciprocal derivative gain limitation.
4. *tau*I: The integral time constant.

Separate Input

Controllers with separate inputs come in many forms. The naming denotes the type of controller and the options that are available. The type and options are also available in the model icons.



1. Two versions of each controller exists: continuous time and discrete time. They can be found in the subdirectories *Continuous* and *Discrete*. Continuous-time controllers have icons with blue text and discrete-time controllers have icons with green text.
2. The first characters denote the type of controller: *P*, *PD*, *PI* or *PID.*
3. The character p or s denote the form: parallel (*p*) or series (*p*).
4. If setpoint weighting is used, the term *sp* is added.
5. If an anti-windup scheme is incorporated, the term *aw* is added.
6. If the output at startup can be set manually, the term *u0* is added.
7. If an external tracking signal is used, the term *tr* is added.

Ports

All controllers have two inputs ports and one output port:

1. *SP*: The setpoint
2. *MV*: The measured variable (also called process variable).
3. *output*: The controller output.

Parameters

Depending on the type of controllers, the following parameters can be used:

1. *K*: The proportional gain of the controller.
2. *Td*: The derivative time constant.
3. *N*: The derivative gain limitation.
4. *Ti*: The integral time constant.
5. *b*: The setpoint weighting constant for the proportional part of the controller.
6. *c*: The setpoint weighting constant for the derivative part of the controller.
7. *Ta*: The tracking time constant of the anti-windup scheme.
8. *minimum*: The minimum controller output representing actuator saturation.
9. *maximum*: The maximum controller output representing actuator saturation.
10. *output_initial*: The controller output at start-up.

Initial Values

Although most controllers have one or more internal states, the initial values of these states do not need to be set manually. You use the default values (0) at the start of a simulation. To get a desired output of the controller at startup , the parameter output_initial should be used.

Discrete Controllers

The discrete controllers that are available in the library are directly derived from their continuous counterparts, using approximation by backward differences. These discrete models have identical behavior, as long as the sample time is not chosen too low. Discrete loops in 20-sim are automatically detected and assigned a default sampletime. You can change the sample time in the Run Properties Editor (in the **Simulator** from the **Properties** menu select the **Simulation** command).

Variables

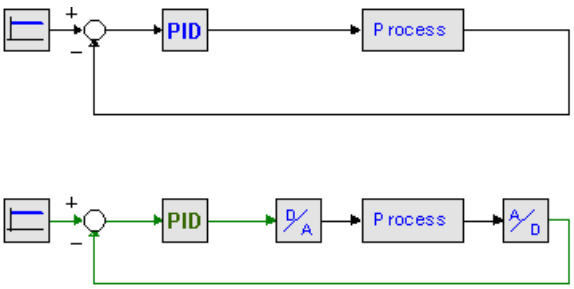
Some PID controllers (parallel form with anti-windup) have two internal variables that are of interest. PB_high and PB_low. These parameters are the upper and lower bound of the proportional band.

Controller Use

All controllers must can be used as shown in the figure below. The upper part shows a continuous time controller and the bottom part shows a discrete time controller.

Error Input

1. An error signal is generated with a plus minus element. Its serves as an input of the controller.
2. The controller output is connected with the process input.

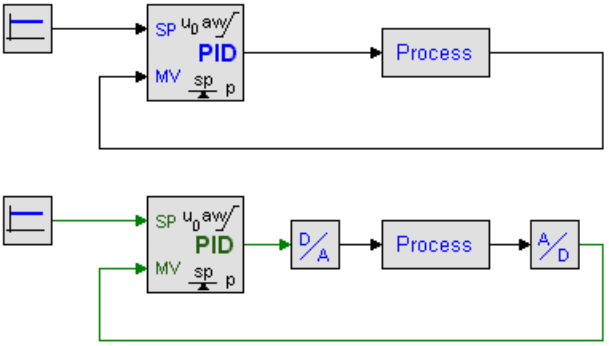


As shown in the figure, analog to digital convertors and digital to analog convertors should be used, when connecting a discrete-time controller to a continuous-time process.

Separate Input

1. A setpoint signal is connected with the **SP** port of the controller.
2. The process output is connected with the **PV** port of the controller.

The controller output is connected with the process input.



As shown in the figure, analog to digital convertors and digital to analog convertors should be used, when connecting a discrete-time controller to a continuous-time process.

Controller-P

Library
Signal\Control\PID Control\Discrete

Use
Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description
This is a P controller. The output of this controller (in s-domain) is:

$$U = K \cdot E$$

with E equal to SP - PV.

Interface

Inputs	**Description**
SP	Setpoint.
PV	Process output.
Outputs	
CV	Controller output: input for the process.
Parameters	
K	Proportional gain.

Controller-PD_p

Library
Signal\Control\PID Control\Discrete

Use
Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description
This is a PD controller in parallel form. The output of this controller (in s-domain) is:

$$U = K \cdot \left(1 + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	Description
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Initial Values	
yes	Should be left default!

Controller-PD_s

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PD controller in series form. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	Description
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	

K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.

Initial Values

yes	Should be left default!

Controller-PI

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller. The output of this controller (in s-domain) is:

$$U = K \cdot \left(1 + \frac{1}{s \cdot Ti}\right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.

Outputs

output	Controller output: input for the process.

Parameters

K	Proportional gain.
Ti	Integral time constant (Ti > 0).

Initial Values

yes	Should be left default!

Controller-PI_sp

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) \right)$$

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
Initial Values	
yes	Should be left default!

Controller-PI_sp_aw

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) \right)$$

An anti-windup scheme has been added to this controller.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PI_sp_aw_u0

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PI_sp_aw_u0_tr

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up. An external tracking signal must be supplied to activate the anti-windup scheme.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
TR	Tracking signal.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
Ta	Tracking time constant.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_p

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form. The output of this controller (in s-domain) is:

$$U = K \cdot \left(1 + \frac{1}{s \cdot Ti} + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
Initial Values	
yes	Should be left default!

Controller-PID_p_sp.

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot (c \cdot SP - MV) \right)$$

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_p_sp_aw

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot (c \cdot SP - MV) \right)$$

An anti-windup scheme has been added to this controller.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_p_sp_aw_u0

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot (c \cdot SP - MV) \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_p_sp_aw_u0_tr

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in parallel form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left((b \cdot SP - MV) + \frac{1}{s \cdot Ti} \cdot (SP - MV) + \frac{s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot (c \cdot SP - MV) \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up. An external tracking signal must be supplied to activate the anti-windup scheme.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
TR	Tracking signal.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!
Interesting Variables	
PB_high, PB_low	Upper and lower bound of the proportional band.

Controller-PID_s

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form. The output of this controller (in s-domain) is:

$$U = K \cdot \left(1 + \frac{1}{s \cdot Ti}\right)\left(\frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}}\right) \cdot E$$

with E equal to SP - MV.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
Initial Values	
yes	Should be left default!

Controller-PID_s_sp

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\left(b + \frac{1}{s \cdot Ti} \right) \cdot \frac{1 + s \cdot c \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot SP - \left(1 + \frac{1}{s \cdot Ti} \right) \frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot MV \right)$$

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Initial Values	
yes	Should be left default!

Controller-PID_s_sp_aw

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\left(b + \frac{1}{s \cdot Ti} \right) \cdot \frac{1 + s \cdot c \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot SP - \left(1 + \frac{1}{s \cdot Ti} \right) \frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot MV \right)$$

An anti-windup scheme has been added to this controller.

Interface

Inputs	**Description**
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
Initial Values	
yes	Should be left default!

Controller-PID_s_sp_aw_u0

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\left(b + \frac{1}{s \cdot Ti} \right) \cdot \frac{1 + s \cdot c \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot SP - \left(1 + \frac{1}{s \cdot Ti} \right) \frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot MV \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up.

Interface

Inputs	Description
SP	Setpoint.
MV	Measured variable or process output.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
minimum	Minimum controller output.
maximum	Maximum controller output.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!

Controller-PID_s_sp_aw_u0_tr

Library

Signal\Control\PID Control\Discrete

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form with setpoint weighting. The output of this controller (in s-domain) is:

$$U = K \cdot \left(\left(b + \frac{1}{s \cdot Ti} \right) \cdot \frac{1 + s \cdot c \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot SP - \left(1 + \frac{1}{s \cdot Ti} \right) \frac{1 + s \cdot Td}{1 + s \cdot \frac{Td}{N}} \cdot MV \right)$$

An anti-windup scheme has been added to this controller as well as an offset for manual output at start-up. An external tracking signal must be supplied to activate the anti-windup scheme.

Interface

Inputs	Description
SP	Setpoint.
MV	Measured variable or process output.
TR	Tracking signal.
Outputs	
output	Controller output: input for the process.
Parameters	
K	Proportional gain.
Td	Derivative time constant (Td > 0).
N	Derivative gain limitation.
Ti	Integral time constant (Ti > 0).
b	Proportional setpoint weighting parameter.
c	Derivative setpoint weighting parameter.
Ta	Tracking time constant.
output_initial	The controller output at start-up.
Initial Values	
yes	Should be left default!

P

Library

Signal\Control\PID Control\Discrete

Use

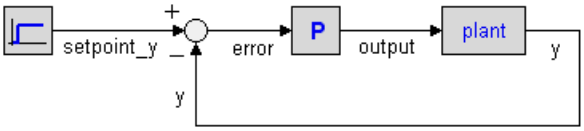
Domains: Continuous, Discrete **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a P controller. The output of this controller is:

$$\text{ouput} = \text{kp} \cdot \text{error}$$

with the input of the controller equal to the error and the output of the controller used as the plant input:



Interface

Inputs	Description
error	Input signal: plant-output minus set point.
Outputs	
output	Output signal: input for the plant.
Parameters	
kp	Proportional gain.

PD

Library

Signal\Control\PID Control\Discrete

Use

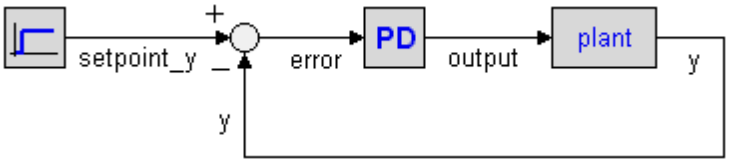
Domains: Continuous, Discrete **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PD controller in series form. The transfer function of an ideal PD-controller is:

$$G(s) = \frac{kp\,(1 + s \cdot tauD)}{1 + s \cdot beta \cdot tauD}$$

with the input of the controller equal to the error and the output of the controller used as the plant input:



Interface

Inputs	Description

error	Input signal: plant-output minus set point.

Outputs

output	Output signal: input for the plant.

Parameters

kp	Proportional gain.
tauD	Derivative gain (tauD <> 0).
beta	Tameness constant (0 < beta << 1).

Initial Values

state_initial	output(0) = state_initial - kp*error(0)*/beta

PI

Library

Signal\Control\PID Control\Discrete

Use

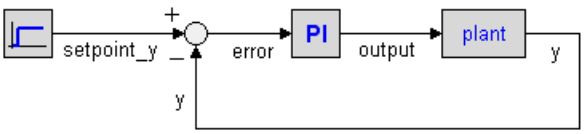
Domains: Continuous, Discrete **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PI controller. The transfer function of the controller is:

$$G(s) = kp + \frac{kp}{s \cdot taul}$$

with the input of the controller equal to the error and the output of the controller used as the plant input:



Interface

Inputs	**Description**
error	Input signal: plant-output minus set point.

Outputs

output	Output signal: input for the plant.

Parameters

kp	Proportional gain.
tauI	Integral gain (tauI <> 0).

Initial Values

state(0)	output(0) = state(0) - kp*error(0)

PID

Library

Signal\Control\PID Control\Discrete

Use

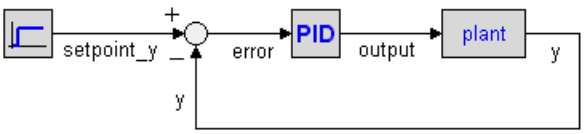
Domains: Continuous, Discrete **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a PID controller in series form. The transfer function is:

$$G(s) = kp \cdot \left(1 + \frac{1}{s \cdot tauI}\right)\left(\frac{1 + s \cdot tauD}{1 + s \cdot beta \cdot tauD}\right)$$

with the input of the controller equal to the error and the output of the controller used as the plant input:



Interface

Inputs	Description
error	Input signal: plant-output minus set point.

Outputs

output	Output signal: input for the plant.

Parameters

kp	Proportional gain.
tauD	Derivative gain (tauD <> 0).
tauI	Integral gain (tauD <> 0).
beta	Tameness constant (0 < beta << 1).

Initial Values

pdstate(0)	pdout(0) = pdstate(0) - kp*error(0)*/beta

pistate(0)	output(0) = pistate(0) - pdout(0)

9.3.4 Cost Functions

Continuous

CostFunction

Library

Signal\Cost Functions\Continuous

Implementations

IE
ISE
IAE
ISTE
ITAE
ITSE
IAEWAI
ISEWSI
ITAEWAI

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric, Electric).

Description - IE

This submodel yields the following cost function:

output = int(e);

Description - ISE

This submodel yields the following cost function:

output = int(e2);

Description

This submodel yields the following cost function:

output = int(abs(e));

Description - ISTE

This submodel yields the following cost function:

*output = int(time2*e2);*

Description - ITAE

This submodel yields the following cost function:

*output = int(time*abs(e));*

Description - ITSE

This submodel yields the following cost function:

*output = int(time*e2);*

I

Description - IAEWAI

This submodel yields the following cost function:

*output = int(abs(e) + lambda*abs(input));*

Description - ISEWSI

This submodel yields the following cost function:

*output = int(e2 + lambda*input2);*

Description

This submodel yields the following cost function:

*output = int(time*abs(e) + lambda*abs(input));*

Interface

Inputs	**Description**
e	Input signal
input	Input signal
Outputs	
output	
Parameters	
lambda	Weighting factor.
Initial Values	
output_initial	The initial value of the output.

Discrete

CostFunction

Library

Signal\Cost Functions\Discrete

Implementations

IE
ISE
IAE

ISTE
ITAE
ITSE
IAEWAI
ISEWSI
ITAEWAI

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Iconic Diagrams (Electric, Electric).

Description - IE

This submodel yields the following cost function:

output = sum(e);

Description - ISE

This submodel yields the following cost function:

output = sum(e2);

Description - IAE

This submodel yields the following cost function:

output = sum(abs(e));

Description - ISTE

This submodel yields the following cost function:

*output = sum(time2*e2);*

Description - ITAE

This submodel yields the following cost function:

*output = sum(time*abs(e));*

Description - ITSE

This submodel yields the following cost function:

*output = sum(time*e2);*

I

Description - IAEWAI

This submodel yields the following cost function:

*output = sum(abs(e) + lambda*abs(input));*

Description - ISEWSI

This submodel yields the following cost function:

*output = sum(e2 + lambda*input2);*

Description

This submodel yields the following cost function:

*output = sum(time*abs(e) + lambda*abs(input));*

Interface

Inputs	**Description**
e	Input signal
input	Input signal
Outputs	
output	
Parameters	
lambda	Weighting factor.
Initial Values	
output_initial	The initial value of the output.

9.3.5 Discrete

AD

Library

Signal\Discrete

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model incorporates the basic functions of an analog to digital convertor: sample and hold, output window (i.e. restriction of the output between a minimum and maximum value) and quantization. The quantization is specified in bits. For example 8 bits quantization means the output has 2^8 - 1 = 255 possible values between the given minimum and maximum.

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
minimum	Minimum output value.
maximum	Maximum output value
bits	Quantization levels (bits).
initial	Initial output

Limitations

- The input of this model is a continuous signal. The output of this model is a discrete signal. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).
- The initial output is bounded by the window given by the minimum and maximum and rounded by the number of bits used.

Clock-Discrete

Library

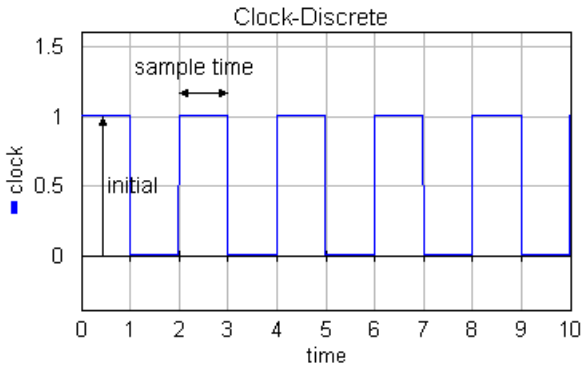
Signal\Discrete

Use

Domains: Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This models generates a logical discrete clock signal, i.e. a signal that changes from true (1) to false (0) and vice-versa, each sample time.



Interface

Outputs	**Description**
output	
Parameters	
initial	Initial value of the output.

Limitations

The output of this model is a discrete signal. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

DA-Delay

Library

Signal\Discrete

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model incorporates the basic functions of a digital to analog convertor: zero order hold, output window (i.e. restriction of the output between a minimum and maximum value) and quantization. The quantisation is specified in bits. For example 8 bits quantisation means the output has 2^8 - 1 = 255 possible values between the given minimum and maximum.

The standard models of the discrete library assume that there is no time needed to perform the calculations the a discrete loop. In general these calculations do take time. To take this into account, the output of this model is delayed in time. This time delay can be set by the user and should correspond to the time needed to perform all calculations of the discrete loop in a real system.

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
minimum	Minimum output value.
maximum	Maximum output value
bits	Quantisation levels (bits).
t_calc	Time needed to perform all calculations of the discrete loop.

Limitations

The input of this model is a discrete signal. The output of this model is a continuous signal. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

DA

Library

Signal\Discrete

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model incorporates the basic functions of a digital to analog convertor: zero order hold, output window (i.e. restriction of the output between a minimum and maximum value) and quantization. The quantization is specified in bits. For example 8 bits quantization means the output has 2^8 - 1 = 255 possible values between the given minimum and maximum.

Interface

Inputs	**Description**
input	

Outputs

output

Parameters

minimum	Minimum output value.
maximum	Maximum output value
bits	Quantization levels (bits).

Limitations

The input of this model is a discrete signal. The output of this model is a continuous signal. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

Delay-n

Library

Signal\Discrete

Use

Domains: Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is the the z-delay function with n samples delay.

*output(k) = input(k - n*T)*
*with k = n * T , n = 1,2,3,...*

The sampletime T can be set in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

Interface

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
output_initial	The initial value of the block.
Parameters	
n	The number of sample delays (n = 1,2,3,..).

Limitations

- The parameter n can not be changed during simulation. If it is increased, recoompiling of the model is necessary.
- The input and output of this model are discrete signals. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

Delay

Library

Signal\Discrete

Use

Domains: Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is the the z-delay function with one sample interval delay.

output(k) = input(k - T)
*with k = n * T , n = 0,1,2,3,...*

The sampletime T can be set in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

Interface

Inputs	Description
input	

Outputs

output

Initial Values

output_initial	The initial value of the block.

Limitations

The input and output of this model are discrete signals. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

DiscreteDifferential

Library

Signal\Discrete

Use

Domains: Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model represents the discrete equivalent of the continuous differentiation:

output(0) = initial; (k = 0)
output(1) = (u(1) - initial)/sampletime; (k = 1)
*output (k) = (u(k) - u(k-1))/sampletime; (k = n * T , n = 2,3,4,...*

The sampletime T can be set in the Simulator (choose **Properties**, **Simulation** and **Discrete System**). The discrete transfer function of this model is:

*H(z) = (z - 1) / (z * sampletime)*

which is the equivalent (using the backward difference transformation) of the continuous time transferential:

H(s) = s

Interface

Inputs	Description
input	

Outputs

output

Parameters

initial	Initial value of the output.

Limitations

The input and output of this model are discrete signals. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

DiscreteIntegral

Library

Signal\Discrete

Use

Domains: Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model represents the discrete equivalent of the continuous integration:

output(k) = initial; (k = 0)
*inp(k-T)*T + inp(k-2T)*T + inp(k-3T)*T*
*+ ... + inp(0)*T + initial; (k = n * T , n = 1,2,3,...)*

The sampletime T can be set in the Simulator (choose **Properties**, **Simulation** and **Discrete System**). The discrete transfer function of this model is:

H(z) = sampletime / (z - 1)

which is the equivalent (using the forward difference transformation) of the continuous time transfer function:

H(s) = 1/s

Interface

Inputs	**Description**
input	

Outputs

output

Parameters

initial	Initial value of the output.

Limitations

The input and output of this model are discrete signals. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

Hold

Library

Signal\Discrete

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model forms the interconnection between a discrete and continuous system: zero order hold.

Interface

Inputs	**Description**
input	discrete signal
Outputs	
output	continuous signal

Limitations

The input of this model is a discrete signal. The output of this model is a continuous signal. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

LinearSystem

Library

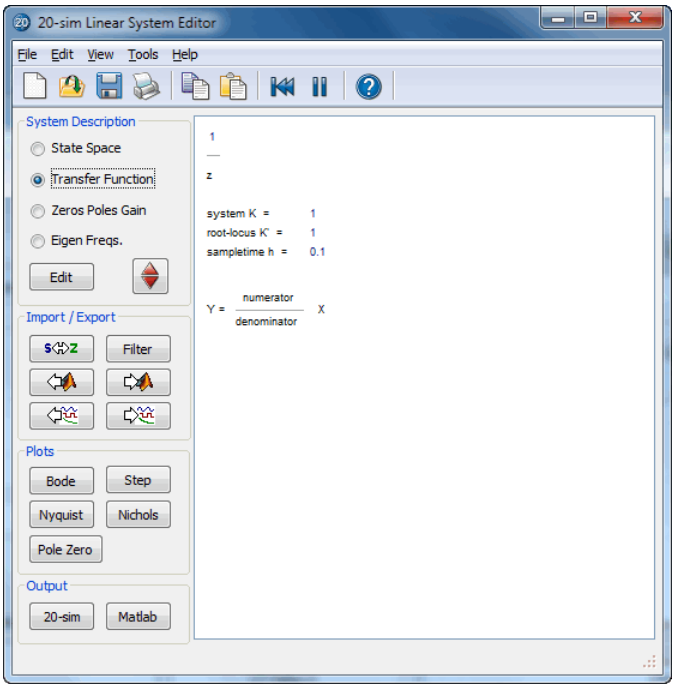
Signal\Discrete

Use

Domains: Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

When you select this model and click *Go Down* a special editor opens (Linear System Editor), allowing you to enter a linear system in State Space form, as a Transfer Function or by adding poles and zeros:



Note

This is a discrete linear system, using the transformation. The transformation is based on a sampletime (here 0.1 s). You can change the sampletime by going to a continuous-time linear system (click the S⇔Z button) and back (click again the S⇔Z button).

Interface

Inputs	Description
input	

Outputs

output

Initial Values

The model has internal states that are not accessible.

Parameters

Parameters are entered by the Linear System Editor.

Quantize-Round

Library

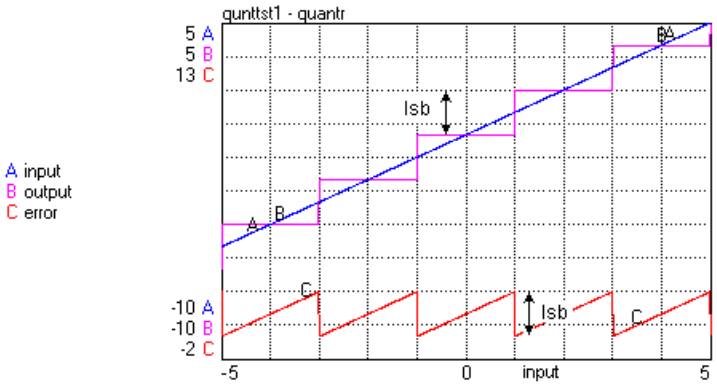
Signal\Discrete

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model can be used to represent quantization of an input signal. It rounds the input as shown in the graph below. The quantization interval is specified by the least significant bit (lsb). The maximum output error is half a least significant bit.



Note

Compare this model with the model Quantisize-Truncate which represents quantization by truncation. Truncation results in a maximum error of a least significant bit.

Interface

Inputs	**Description**
input	
Outputs	
output	
Parameters	
lsb	Least significant bit.

Quantize-Truncate

Library

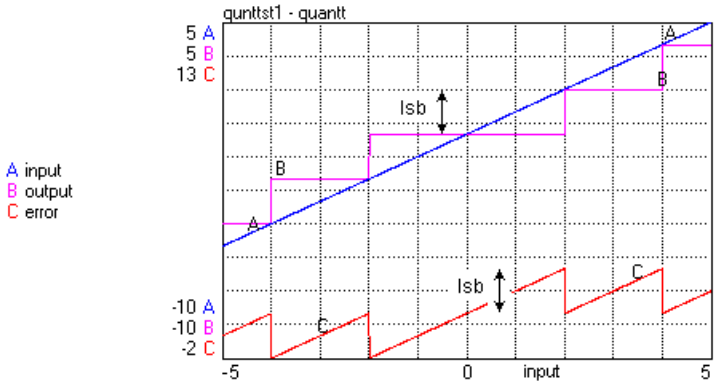
Signal\Discrete

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model can be used to represent quantization of an input signal. It truncates the input as shown in the graph below. The quantization interval is specified by the least significant bit (lsb). The maximum output error is a least significant bit.



Note

Compare this model with the model Quantisize-Round which represents quantization by rounding. Rounding results in a maximum error of half a least significant bit.

Interface

Inputs	Description
input	

Outputs

output

Parameters

lsb	Least significant bit.

Sample

Library

Signal\Discrete

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model samples and holds the input every T [s].

output = input; (t = k)
*where k = n*T , with n = 0,1,2,3, ...*

The sampletime T can be set in the Simulator (choose **Properties**, **Run** and **Discrete System**).

Interface

Inputs	Description
input	

Outputs

output

Limitations

The input of this model is a continuous signal. The output of this model is a discrete signal. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

Sigma

Library

Signal\Discrete

Use

Domains: Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model represents a discrete summation operator:

output(k) = initial;
with k = 0
outp(k) = inp(k-T) + inp(k-2T) + inp(k-3T) + ...+ inp(0) + initial
*with k = n * T , n = 1,2,3,...*

The sampletime T can be set in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

Interface

Inputs	**Description**
input	

Outputs

output

Parameters

initial	Initial value of the output.

Limitations

The input and output of this model are discrete signals. 20-sim will automatically detect the existence of discrete models. Each chain of discrete models will be assigned a specific sampletime. You can set this sample time to any desired value in the Simulator (choose **Properties**, **Simulation** and **Discrete System**).

9.3.6 Events

Event

Library

Signal\Events

Implementations

CrossingBoth
CrossingDown
CrossingUp

Use

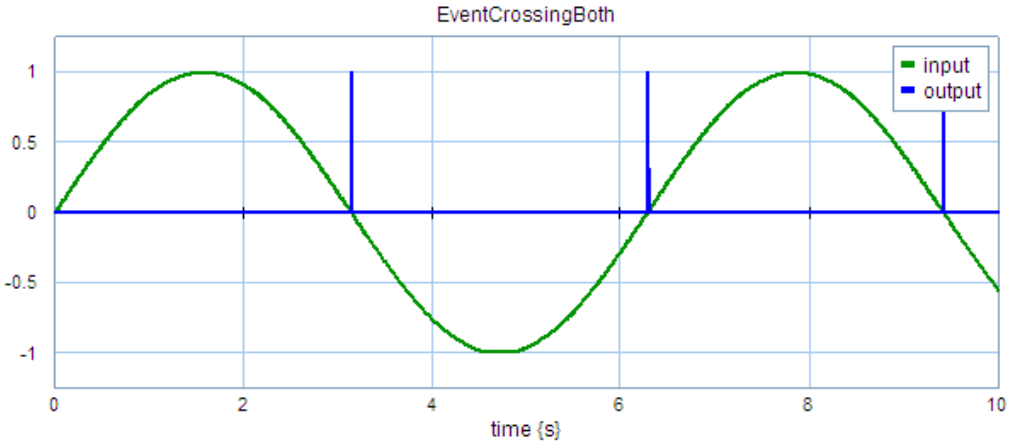
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description - CrossingBoth

This is the block implementation of the event function. The output signal is a boolean which goes from false (0) to true (1) when the input signal crosses zero (offset = 0):

output = event (input - offset);

If desired, you can change the parameter *offset* to trigger the event on a non-zero input value.



The output of the event block for a sinusoidal input (parameter offset = 0).

Interface - CrossingBoth

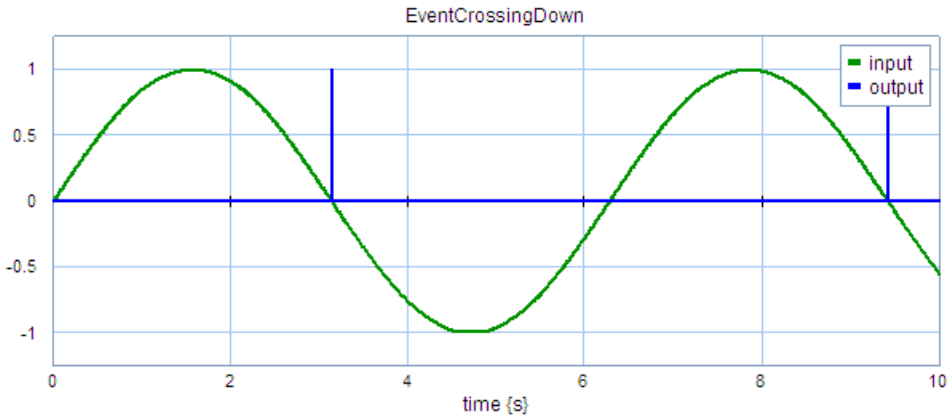
Inputs	**Description**
input	
Outputs	**Description**
output	event signal (boolean)
Parameters	
offset	output is true (event) when the input signal crosses this value (default = 0)

Description - CrossingDown

This is the block implementation of the eventdown function. The output signal is a boolean which goes from false (0) to true (1) when the input signal crosses zero with a negative slope (offset = 0):

output = eventdown (input - offset);

If desired, you can change the parameter *offset* to trigger the event on a non-zero input value.



The output of the event block for a sinusoidal input (parameter offset = 0).

Interface - CrossingDown

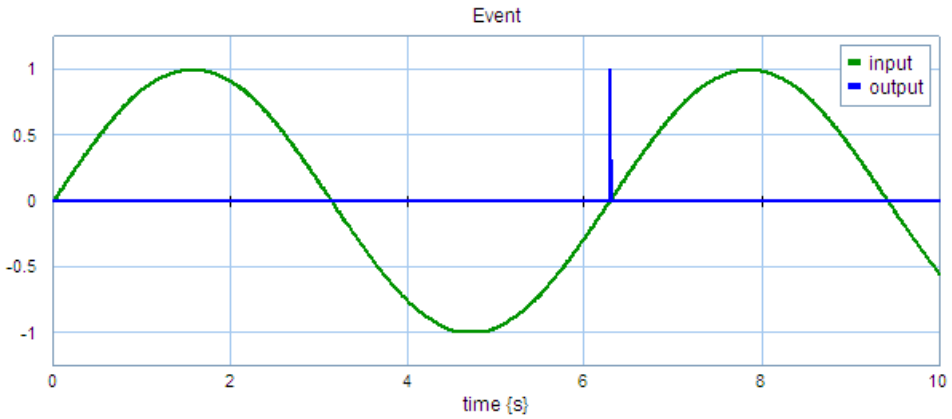
Inputs	**Description**
input	
Outputs	**Description**
output	event signal (boolean)
Parameters	
offset	output is true (event) when the input signal crosses this value (default = 0).

Description - CrossingUp

This is the block implementation of the eventup function. The output signal is a boolean which goes from false (0) to true (1) when the input signal crosses zero with a positive slope (offset = 0):

output = eventup (input - offset);

If desired, you can change the parameter *offset* to trigger the event on a non-zero input value.



The output of the event block for a sinusoidal input (parameter offset = 0).

Interface - CrossingUp

Inputs	**Description**
input	
Outputs	**Description**
output	event signal (boolean)
Parameters	
offset	output is true (event) when the input signal crosses this value (default = 0).

FrequencyEvent

Library

Signal\Events

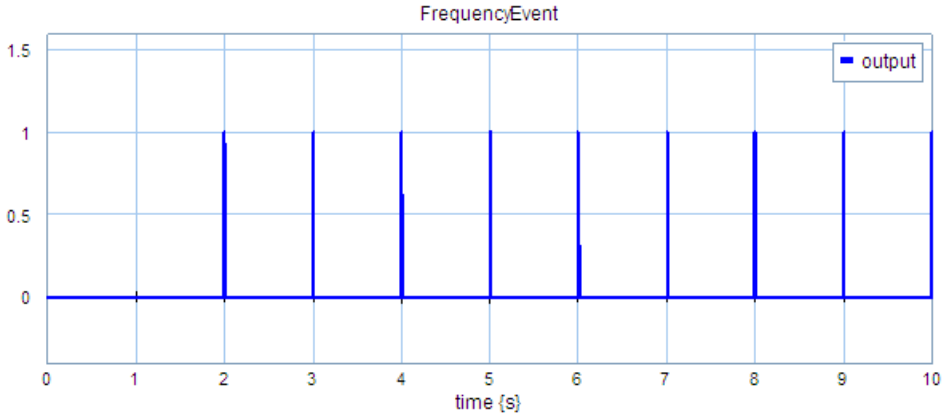
Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This is the block implementation of the frequencyevent function. The output signal is a boolean which is true (1) every time when *p* [s] have passed. The function starts after an offset of *o* [s]. The offset parameter is optional.

output = frequencyevent(p,o);



The output of the frequencyevent block with a period of 1 s and an offset of 2 s.

Interface

Outputs	**Description**
output	event signal (boolean)
Parameters	
period	period in [s]
offset	offset in [s]

TimeEvent

Library

Signal\Events

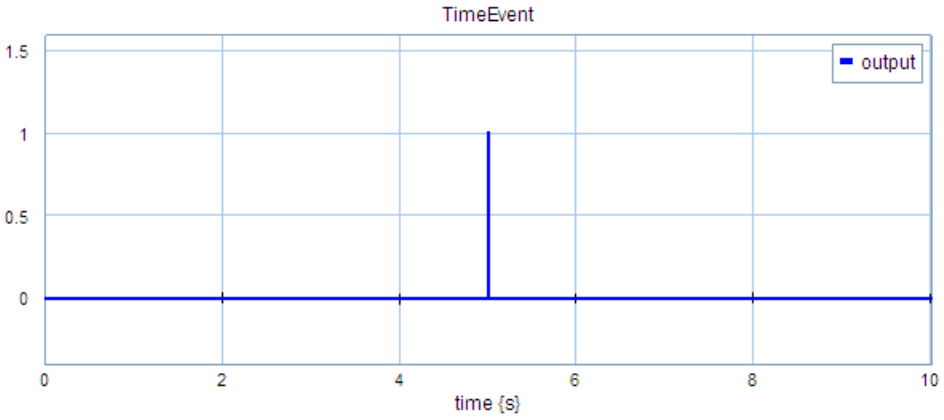
Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This is the block implementation of the timeevent function. The output signal is a boolean which is true (1) at a specific time given by the parameter *triggerTime*:

output = timeevent(triggerTime);



The output of the timeevent block with a triggerTime of 5 s.

Interface

Outputs	**Description**
output	event signal (boolean)
Parameters	
triggerTime	time in [s] when the output is true.

9.3.7 Filters

Filter

Library

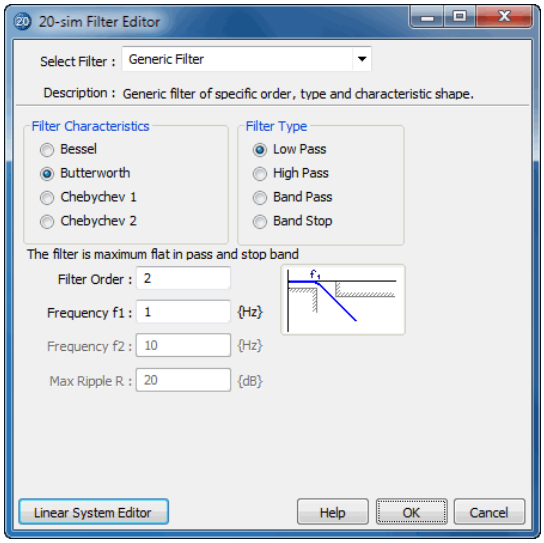
Signal\Block Diagram

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

When you select this model and click *Go Down* a special editor opens (Filter Editor), allowing you to choose a filter:



Interface

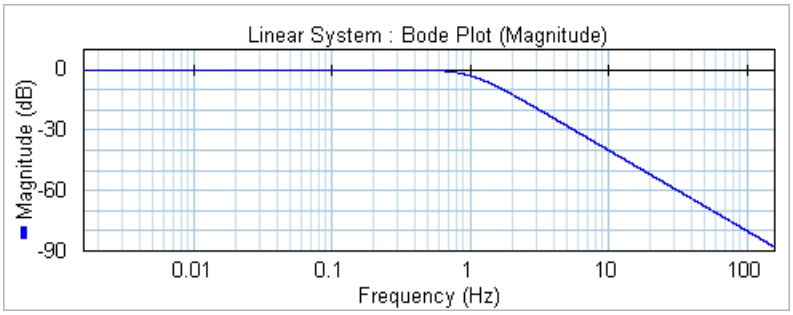
Inputs	Description
input	
Outputs	
output	
Initial Values	
	The model has internal states that are not accessible.
Parameters	
	Parameters are entered by the Filter Editor.

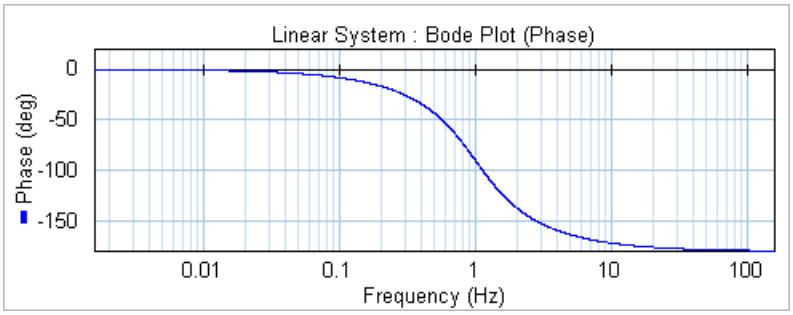
LowPassFilter-BW2

Library
Signal\Filters

Use
Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description
This is a second order Butterworth filter. The bandwidth is given in Hz. The phase and magnitude plot (bandwidth = 1 Hz) are shown in the figure below.





Interface

Outputs	**Description**
input,output	
Parameters	
BW	Bandwidth (Hz)
Initial Values	
y_initial	Output value of the filter at t = 0 s.

LowPassFilter-BW4

Library

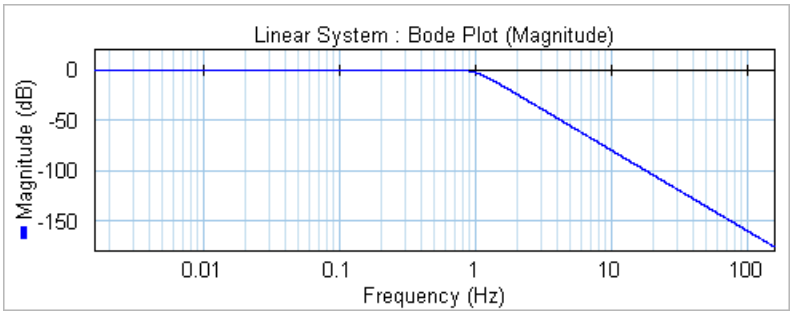
Signal\Filters

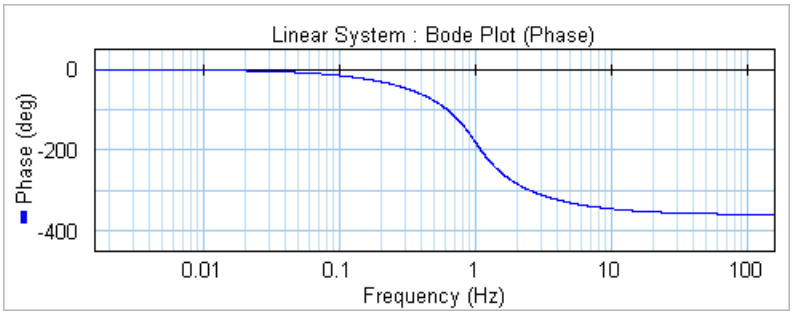
Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a fourth order Butterworth filter. The bandwidth is given in Hz. The phase and magnitude plot (bandwidth = 1 Hz) are shown in the figure below.





Interface

Outputs	**Description**
input,output	
Parameters	
BW	Bandwidth (Hz)
Initial Values	
y_initial	Output value of the filter at t = 0 s.

LowPassFilter-FO

Library

Signal\Filters

Use

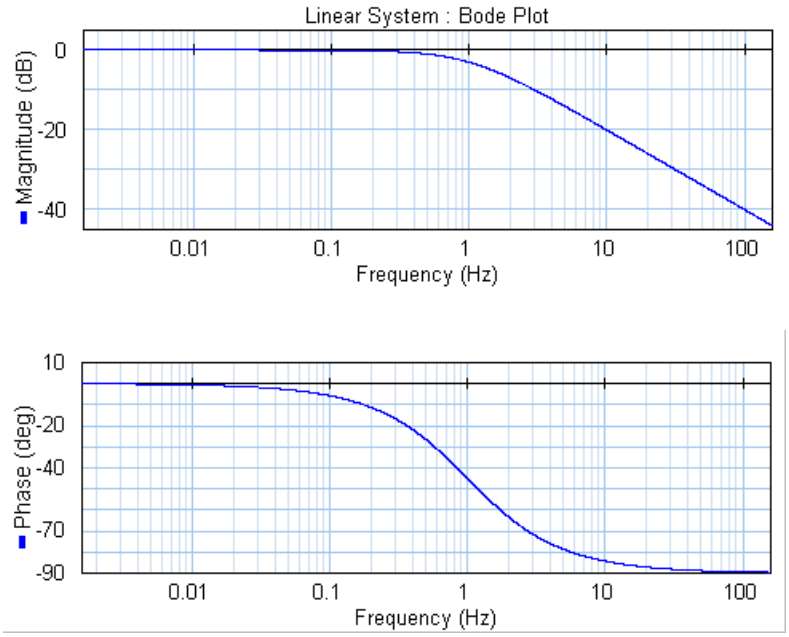
Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a first order filter. with transfer function:

$$Y(s) = \frac{1}{\frac{s}{2 \cdot \pi \cdot BW} + 1} \cdot U(s)$$

The bandwidth is given in Hz. The phase and magnitude plot (bandwidth = 1 Hz) are shown in the figure below.



Interface

Outputs	**Description**
input,output	
Parameters	
BW	Bandwidth (Hz)
Initial Values	
state_initial	Output value of the filter at t = 0 s.

LowPassFilter-SO

Library

Signal\Filters

Use

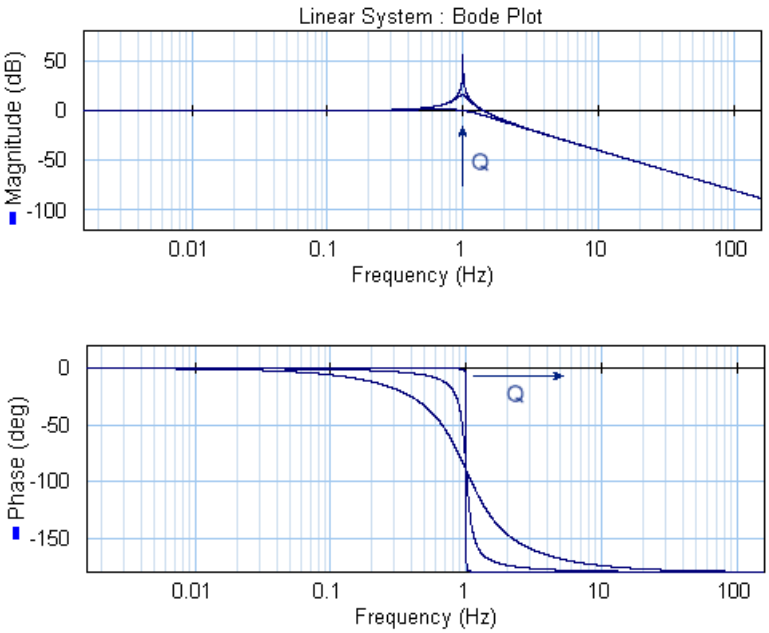
Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a second order filter. with transfer function:

$$Y(s) = \frac{(2 \cdot \pi \cdot BW)^2}{s^2 + 2 \cdot \pi \cdot BW \frac{s}{Q} + (2 \cdot \pi \cdot BW)^2} \cdot U(s)$$

The bandwidth is given in Hz. The phase and magnitude plot (bandwidth = 1 Hz) are shown in the figure below.



Interface

Outputs	Description
input,output	

Parameters

BW	Bandwidth (Hz)
Q	Quality ()

Initial Values

state1_initial	Output rate of the filter at t = 0 s.
state2_initial	Output value of the filter at t = 0 s.

9.3.8 Import Export

Matlab

DoFromMatlab

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This submodel sends a command to the Matlab workspace during simulation, and gets the output signal from the Matlab workspace.

Interface

Outputs	**Description**
output	The variable from the Matlab workspace.

Parameters

command	Command string in Matlab format

name	Name of the variable inside the Matlab workspace.

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

DoMatlab-Final

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This submodel sends a command to the Matlab workspace at the end of the simulation.

Interface

Parameters	**Description**
command	Command string in Matlab format

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

DoMatlab-Initial

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This submodel sends a command to the Matlab workspace at the start of the simulation.

Interface

Inputs	**Description**
Outputs	

Parameters

command	Command string in Matlab format

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

DoMatlab

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in:** Block Diagrams.

Description

This submodel sends a command to the Matlab workspace during simulation.

Interface

Parameters	Description
command	Command string in Matlab format

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

FromMatlab

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This submodel gets an output from the Matlab workspace during simulation.

Interface

Parameters	Description
name	Name of the variable inside the Matlab workspace

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

ToDoFromMatlab

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size:** 1-D. **Allowed in**: Block Diagrams.

Description

This submodel sends the input signal to the Matlab workspace, sends the command to the workspace, and gets the output variable back. This is done during the 20-sim simulation.

Interface

Inputs	**Description**
input	The variable for the Matlab workspace.
Outputs	
output	The variable from the Matlab workspace
Parameters	
command	Command string in Matlab format
nameInput	Name of the input variable in the Matlab workspace.
nameOutput	Name of the output variable in the Matlab workspace.

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

ToDoMatlab

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in:** Block Diagrams.

Description

This submodel sends the input signal and a command to the Matlab workspace during simulation.

Interface

Inputs	Description
input	The variable for the Matlab workspace.

Outputs

Parameters

command	Command string in Matlab format
name	Name of the variable inside the Matlab workspace.

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

ToMatlab-Plot

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This submodel sends the input signal to the Matlab workspace, stores the input inside the Matlab workspace into a matrix, and plots the result in a graph. Each row of the matrix corresponds with a major simulation step, so the input is limited to a scalar or vector. Note: the name is set during the processing phase only

Interface

Inputs	Description
input	The variable to send to the Matlab workspace.

Parameters

name	Name of the variable inside the Matlab workspace

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

ToMatlab-Store

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This submodel sends the input signal to the Matlab workspace and stores the input inside the Matlab workspace into a matrix. Each row of the matrix corresponds with a major simulation step, so the input is limited to a scalar or vector. Note: the name is set during the processing phase only.

Interface

Inputs	**Description**
input	The variable to send to the Matlab workspace.

Outputs

Parameters

name	Name of the variable inside the Matlab workspace

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

ToMatlab-Timed

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size:** 1-D. **Allowed in:** Block Diagrams.

Description

This submodel sends the time and input signal to the Matlab workspace. The input is limited to a scalar or vector. Note: the name is set during the processing phase only.
Interface

Inputs	**Description**
input	The variable to send to the Matlab workspace.
Parameters	
name	Name of the variable inside the Matlab workspace

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

ToMatlab-TimedPlot

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. **Size:** 1-D. **Allowed in:** Block Diagrams.

Description

This submodel sends the time and input signal to the Matlab workspace, stores the time and input inside the Matlab workspace into a matrix, and plots the result in a graph. Each row of the matrix corresponds with a major simulation step, so the input is limited to a scalar or column vector. Note: the name is set during the processing phase only

Interface

Inputs	**Description**
input	The variable to send to the Matlab workspace.
Parameters	
name	Name of the variable inside the Matlab workspace

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

ToMatlab-TimedStore

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. Size: 1-D. Allowed in: Block Diagrams.

Description

This submodel sends the time and input signal to the Matlab workspace and stores the time and input inside the Matlab workspace into a matrix. Each row of the matrix corresponds with a major simulation step, so the input is limited to a scalar or column vector. Note: the name is set during the processing phase only

Interface

Inputs	**Description**
input	The variable to send to the Matlab workspace.
Parameters	
name	Name of the variable inside the Matlab workspace

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

ToMatlab

Library

Signal\Import Export\Matlab

Use

Domains: Continuous, Discrete. Size: 1-D. Allowed in: Block Diagrams.

Description

This submodel sends a variable to the Matlab workspace during simulation.

Interface

Inputs	**Description**
input	The variable to send to the Matlab workspace.
Parameters	
name	Name of the variable inside the Matlab workspace

Tip

20-sim uses a single quote ' for strings. If you wish to include a single quote inside the command string for Matlab, use 2 single quotes '' in the command string instead.

9.3.9 Logical

Continuous

And

Library

Signal\Logical\Continuous

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

The output signal of this model is according the truth table below:

input1 \ input2	**false**	**true**
false	false	false
true	false	true

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

	Description
Inputs	
input1	
input2	
Outputs	
output	

Clock

Library

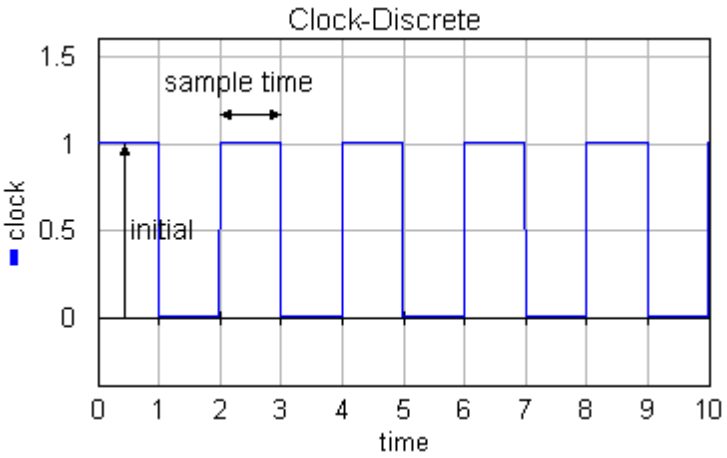
Signal\Logical\Continuous

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This models generates a logical continuous clock signal, i.e. a signal that changes from true (1) to false (0) and back, each period T.



Interface

	Description
Outputs	
output	
Parameters	
initial	Initial value of the output.
T	Clock-period.

DTypeFlipFlop

Library

Signal\Logical\Continuous

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

The initial output of this model is equal to the parameter *initial*. Otherwise the output signal is equal to the input signal each time the control input changes from false to true:

output = input; (oldcontrol = false and control = true)
output = oldoutput; (otherwise)

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

Inputs	**Description**
input	
control	
Outputs	
output	
Parameters	
initial	Initial value of the output.

Invertor

Library

Signal\Logical\Continuous

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

The output signal of this model is according the truth table below:

input	**output**
false	true

true	false	

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

Inputs	**Description**
input	
Outputs	
output	

Logical-Xor

Library

Signal\Logical\Continuous

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

The output signal of this model is according the truth table below:

input2 input1	**false**	**true**
false	false	true
true	true	false

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

	Description
Inputs	
input1	
input2	
Outputs	
output	

Nand

Library

Signal\Logical\Continuous

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

The output signal of this model is according the truth table below:

input1 \ input2	**false**	**true**
false	true	true
true	true	false

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

	Description
Inputs	
input1	
input2	
Outputs	
output	

Nor

Library
Signal\Logical\Continuous

Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description
The output signal of this model is according the truth table below:

input1 \ input2	**false**	**true**
false	true	false
true	false	false

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

	Description
Inputs	
input1	
input2	
Outputs	
output	

Or

Library
Signal\Logical\Continuous

Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description
The output signal of this model is according the truth table below:

input1 \ input2	**false**	**true**
false	false	true
true	true	true

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

	Description
Inputs	
input1	
input2	
Outputs	
output	

SetResetFlipFlop

Library

Signal\Logical\Continuous

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is the continuous version of `SetResetFlipFlop-Discrete.emx`. The initial output of this model is equal to the parameter *initial*. Otherwise the output signal is according the truth table below:

input1	input2	output	
false	false	previous output	
true	false	true	*set condition*
false	true	false	*reset condition*
true	true	previous output	

with

	false	true
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

Inputs	Description
input1	
input2	

Outputs

output

Parameters

initial	Initial value of the output.

TriggerTypeFlipFlop

Library

Signal\Logical\Continuous

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

Initially the output is equal to the parameter initial. Otherwise the output signal is changed each time the control input changes from false to true:

output = not oldoutput; (oldcontrol = false and control = true)
output = oldoutput; (otherwise)

with

	false	true
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

Inputs	Description
input	
control	

Outputs

output

Parameters

initial	Initial value of the output.

Discrete

And

Library

Signal\Logical\Discrete

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

The output signal of this model is according the truth table below:

input2 / input1	**false**	**true**
false	false	false
true	false	true

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

	Description
Inputs	
input1	
input2	
Outputs	
output	

Clock

Library

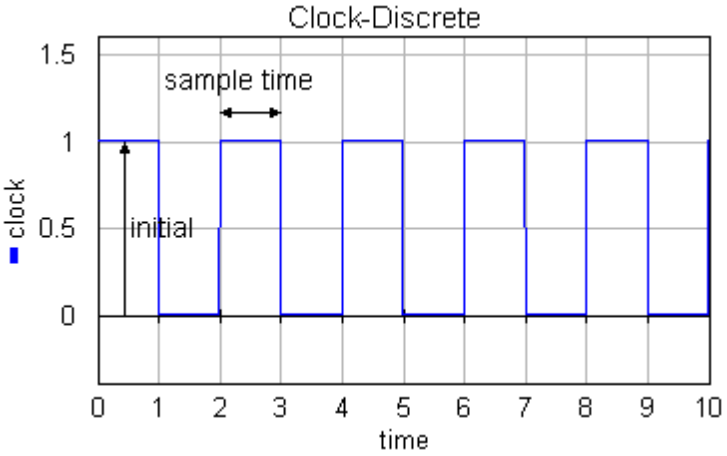
Signal\Logical\Discrete

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This models generates a logical discrete clock signal, i.e. a signal that changes from true (1) to false (0) and vice-versa, each sample time.



Interface

Outputs	**Description**
output	
Parameters	
initial	Initial value of the output.
T	Clock-period.

DTypeFlipFlop

Library

Signal\Logical\Discrete

Use

Domains: Discrete. **Size**: 1-D. **Kind**: Block Diagrams.
The initial output of this model is equal to the initial value *oldoutput_initial*. Otherwise the output signal is equal to the input signal each time the control input changes from false to true:

output = input; (oldcontrol = false and control = true)
output = oldoutput; (otherwise)

with

	false	true
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

Inputs	Description
input	
control	
Outputs	
output	
Initial Values	
oldcontrol_initial	Initial value of the control signal.
oldoutput_initial	Initial value of the output.

Invertor

Library

Signal\Logical\Discrete

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

The output signal of this model is according the truth table below:

input	output
false	true
true	false

with

	false	true
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

	Description
Inputs	
input	
Outputs	
output	

Nand

Library
Signal\Logical\Discrete
Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
The output signal of this model is according the truth table below:

input2 / input1	**false**	**true**
false	true	
true	true	

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

	Description
Inputs	
input1	
input2	
Outputs	
output	

Nor

Library
Signal\Logical\Discrete

Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description
The output signal of this model is according the truth table below:

input1 \ input2	**false**	**true**
false	true	
true	false	

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

	Description
Inputs	
input1	
input2	
Outputs	
output	

Or

Library
Signal\Logical\Discrete

Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description
The output signal of this model is according the truth table below:

input1 \ input2	**false**	**true**
false	false	
true	true	

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

Inputs	**Description**
input1	
input2	
Outputs	
output	

SetResetFlipFlop

Library

Signal\Logical\Discrete

Use

Domains: Discrete. **Size**: 1-D. **Kind**: Block Diagrams.

Description

The initial output of this model is equal to the initial_value *oldoutput_initial*. Otherwise the output signal is according the truth table below:

input1	input2	output	
false	false	previous output	
true	false	true	*set condition*
false	true	false	*reset condition*
true	true	previous output	

with

	false	true
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

Inputs	Description
input1	
input2	
Outputs	
output	
Initial Values	
oldoutput_initial	Initial value of the output.

TriggerTypeFlipFlop

Library

Signal\Logical\Discrete

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

Initially the output is equal to the initial value *oldoutput_intitial*. Otherwise the output signal is changed each time the control input changes from false to true:

output = not oldoutput; (oldcontrol = false and control = true)
output = oldoutput; (otherwise)

Interface

Inputs	Description
control	
Outputs	
output	
Initial Values	
oldcontrol_initial	Initial value of the control signal.
oldoutput_initial	Initial value of the output.

Xor

Library
Signal\Logical\Discrete
Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
The output signal of this model is according the truth table below:

input1 \ input2	**false**	**true**
false	false	
true	true	

with

	false	**true**
inputs	<= 0.5	> 0.5
output	0.0	1.0

Interface

Inputs	**Description**
input1	
input2	
Outputs	
output	

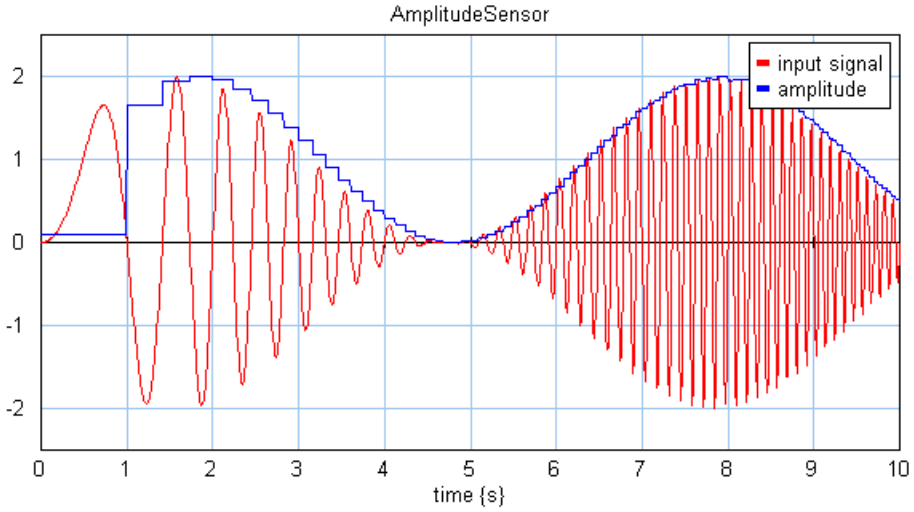
9.3.10 Signal Processing

AmplitudeSensor

Library
Signal\Signal Processing
Use
Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model measures the amplitude of harmonic signals, i.e. periodic signals, that go through zero twice every period. For half a period (between the passes through zero) the input signal is monitored. The largest value is stored and given as the absolute output signal, the next half period. That is why this sensor always has an output delay of half a period.



Interface

Inputs	Description
input	
Outputs	
output	
Parameters	
initial	Initial output value

Maximum

Library
Signal\Signal Processing
Use
Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
This submodel yields the maximum value of an input signal.
Interface

Inputs	**Description**
input	Input signal
Outputs	
output	Maximum value of the input signal

Mean

Library
Signal\Signal Processing
Use
Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
This submodel yields the mean of an input signal using the integral function:

output = int(input)/time

Interface

Inputs	**Description**
input	Input signal
Outputs	
output	Mean value of the input signal

Minimum

Library

Signal\Signal Processing

Use

Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This submodel yields the minimum value of an input signal.

Interface

Inputs	**Description**
input	Input signal
Outputs	
output	Minimum value of the input signal

MovingAverage

Library

Signal\Signal Processing

Use

Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This submodel yields the average of an input signal over a window T0 (s) using an integral function:

output = int(input,0)/T0

Interface

Inputs	**Description**
input	Input signal
Outputs	
output	
Parameters	
T0	Time window

RootMeanSquare

Library
Signal\Signal Processing
Use
Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
This submodel yields the root mean square of an input signal using an integral function:

output = sqrt(int(input2,0)/time)

Interface

Inputs	**Description**
input	Input signal

Outputs
output

StandardDeviation

Library
Signal\Signal Processing
Use
Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
This submodel yields the standard deviation of an input signal using an integral function:

output = sqrt(int((input - mean)2 / time))

where mean is the mean of the input
Interface

Inputs	**Description**
input	Input signal

Outputs

output	Standard deviation of the input signal

Variance

Library
Signal\Signal Processing
Use
Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
This submodel yields the variance of an input signal using an integral function:

output = int((input - mean)2 / time)

where mean is the mean of the input
Interface

Inputs	**Description**
input	Input signal
Outputs	
output	Variance of the input signal

9.3.11 Sources

Constant

Library
Signal\Sources
Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
The output of this model is equal to the parameter *C*.

output = C;
Interface

Outputs	**Description**
output	
Parameters	
C	The output is equal to C.

DataFromFile

Library

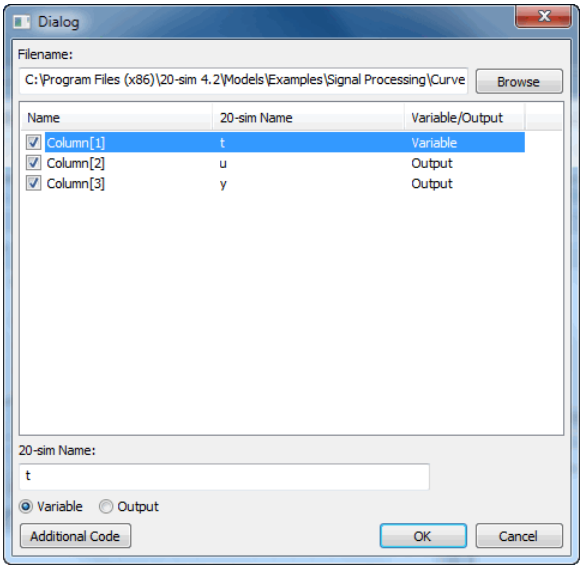
Signals\Sources

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a masked model which opens the Data Input Wizard when edited. You can use this wizard to open a data file.



Items

- *Filename*: Enter the filename of the datafile or use the *Browse* button.
- *Name*: The columns of data.
- *20-sim Name*: The 20-sim name that corresponds with the column data. You can enter any desired name.
- *Variable / Output*: The data may be a variable (useful for plotting in the simulator) or an output of the model (use as an input for other models).

Interface

Outputs	Description
outputs	user defined outputs

File format

20-sim supports data files in Comma Separated Values format (.csv). The first line is used as a header describing the column name. Without a proper header line, you have to define the 20-sim variable names for each column yourself. The first column should always contain the time values. The other columns should contain the corresponding data values (see also data).

Example CSV file format with header:

```
"time","x","y"
0.0, 1, 2
0.1, 5, 10
0.2, 6, 11
```

Joystick

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model uses the joystick.dll function to get a joystick input.

Tips

- Use attempting real-time simulation when using joystick input to prevent the simulation from being calculated to quickly.

Interface

Outputs	**Description**
outputs	Various outputs
Parameters	
Scale	Various scaling parameters

Keyboard

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model uses the keyboard.dll function to get a keyboard input. The keyboard input is an integer number and displayed by the output signal *keycode*. The default model has four additional output signals with a default value of zero and a value of 1 when the keyboard arrows are pressed.

Tips

- Inspect the contents of the model and change it to read other keyboard inputs.
- Open the Example model *Examples/2D mechanics/ScaraRobotKeyboard* to see a demonstration of the keyboard input.
- Use attempting real-time simulation when using keyboard inputs to prevent the simulation from being calculated to quickly.

Interface

Outputs	**Description**
left	left keyboard arrow pressed
right	right keyboard arrow pressed
up	up keyboard arrow pressed
down	down keyboard arrow pressed
keycode	number of the keyboard input
Parameters	
Scale	Various scaling parameters

MotionProfile

Library

Signals\Sources

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a masked model which opens the Motion Profile Wizard (Old Style) when edited. Depending on the selections entered, various motion profiles can be generated.

Interface

Outputs	Description
x	position
v	velocity
a	acceleration

Parameters

stroke	amplitude of the profile
start_time	start time of the profile
stop_time	time when the maximum is reached
return_time	start time of the return motion
end_time	finish time of the return motion
period	period of the profile

MotionProfile-Wizard

Library

Signals\Sources

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This is a masked model which opens the Motion Profile Wizard when edited. Depending on the selections entered, various motion profiles can be generated.

Interface

Outputs	Description
x	position
v	velocity
a	acceleration

Parameters

stroke	amplitude of the profile
start_time	start time of the profile
stop_time	time when the maximum is reached
return_time	start time of the return motion
end_time	finish time of the return motion
period	period of the profile

One

Library
Signal\Sources
Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
The output of this model is equal to one.

output = 1.0;

Interface

Outputs	Description
output	

Pi

Library
Signal\Sources
Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
The output of this model is equal to pi.

output = p;

Interface

Outputs	Description
output	

SignalGenerator-Cycloid

Library
Signal\Sources
Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
This model generates a cycloidal step signal:

*tDelta = 2*pi*(time - start_time)/(stop_time - start_time)*

output = 0; (tDelta < 0)
output = amplitude(tDelta - sin(tDelta))/2*pi; (0 <= tDelta <= 2*pi)*
*output = amplitude; (tDelta > 2*pi)*

Interface

Outputs	**Description**
output	
Parameters	
start_time	The start time of the step.
stop_time	The stop time of the step.
amplitude	The amplitude of the step.

SignalGenerator-FileInput

Library

Signal\Source

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

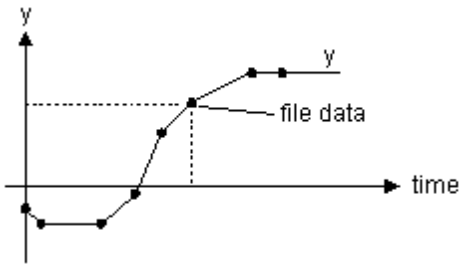
Description

This model generates an output by linear interpolation of data read from file. The data on file is stored in columns. The first column contains the time values (*t*) and the second column contains the corresponding output values (*y*).

- The time data of the first column needs to be monotonically increasing.
- Discontinuities are allowed, by providing the same time point twice in the table.
- Values outside of the range, are computed by linear extrapolation of the last two points.

file:

0	0.5
0.5	-1
2.5	-1
3.5	0
4.5	1
5.5	1.75
7	2.5



8 2.5

Example of an input file.

The input file must be an ASCII (text) file and should consist at least two columns of data. The first column (number 0) should always contain the time values. The other columns (number 1, 2, 3, etc.) should contain the corresponding data values. The parameter *column* is used to specify which column is used for as output data.

The various values must be separated by a comma, a space or a tab. Each new set of time and data values must start on a new line. No comment or other text may be part of the file. The filename of the input file can be specified using the complete path (e.g. c:\data\data.tbl). When no path is given, the file is assumed to be in the experiment directory.

Interface

Outputs	**Description**
output	
Parameters	
filename	Filename of the input file.
column	column number.

Example

Example data.tbl file with header and 1 column with data:

```
"time","y"
0.0, 0.5
0.5, -1
2.5, -1
3.6, 9
```

SignalGenerator-Gaussian Noise

Library

Signal\Sources

Implementations

frequency_limited_FOH
frequency_limited_ZOH
unlimited

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description - frequency_limited_FOH

This model generates a gaussian noise signal with gaussian distribution and variance 1 and random seed s. The noise is multiplied by the parameter amplitude.

*output = amplitude*gauss(1,seed);*

The noise signal is limited to a maximum frequency, which can be adjusted (parameter). Values in-between two noise samples are interpolated (first order hold).

Interface - frequency_limited_FOH

Outputs	**Description**
output	
Parameters	
seed	random seed
amplitude	-amplitude <= output <= +amplitude
frequency	frequency band of the noise

Description - frequency_limited_ZOH

This model generates a gaussian noise signal with gaussian distribution and variance 1 and random seed s. The noise is multiplied by the parameter amplitude.

*output = amplitude*gauss(1,seed);*

The noise signal is limited to a maximum frequency, which can be adjusted (parameter). Values in-between two noise samples are interpolated (first order hold).

Interface - frequency_limited_ZOH

Outputs	**Description**
output	
Parameters	
seed	random seed
amplitude	-amplitude <= output <= +amplitude
frequency	frequency band of the noise

Description - unlimited

This model generates a gaussian noise signal with gaussian distribution and variance 1 and random seed s. The noise is multiplied by the parameter amplitude.

*output = amplitude*gauss(1,seed);*

The noise signal has no frequency limitation.

Note

It is advised only to use this implementation with fixed step integration methods such as Euler and Runge Kutta 2/4. If you run this implementation with a variable step integration method, simulations may get very slow!

Interface - unlimited

Outputs	Description
output	

Parameters

seed	random seed
amplitude	-amplitude <= output <= +amplitude

Limitations

seed must be a number in the region <0,65000>.

Random Seed

20-sim generates a sequence of random numbers for each simulation differently depending upon the value of the random seed parameter. The random noise function and gaussian noise function are affected by this. The default value of the random seed is 0. The maximum value is 65000.

Default value (0)

When the random seed value is 0 (default value), 20-sim generates a new sequence of random numbers for each simulation and for each new random function. E.g. when two random functions with default random seed value (0) are used in one model, they will generate different sequences of random numbers during a simulation.

Other values (>0)

When the random seed value is chosen larger than zero, 20-sim generates the same sequence of random numbers for each simulation. Moreover 20-sim will generate the same sequence of random numbers for each random function that uses the same random seed parameter (>0). E.g. when two random functions with random seed value 50, are used in one model, they will generate the same sequence of random numbers during a simulation.

SignalGenerator-Pulse

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This models yields a pulse signal:

output = 0; (time < start_time)
output = amplitude; (start_time <= time <= stop_time)
output = 0; (time >= stop_time)

Interface

Outputs	**Description**
output	
Parameters	
start_time	The start time of the pulse.
stop_time	The stop time of the pulse.
amplitude	The amplitude of the step.

SignalGenerator-Ramp

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This models yields a ramp signal:

output = 0; (time < start_time)
*output = slope * ramp(start_time); (time >= start_time)*

Interface

Outputs	**Description**
output	
Parameters	
start_time	The start time of the ramp.
slope	The slope of the ramp.

SignalGenerator-Random

Library

Signal\Sources

Implementations

frequency_limited_FOH
frequency_limited_ZOH
unlimited

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description - frequency_limited_FOH

This model generates a random (noise) signal with values between -amplitude and +amplitude and random seed s. The

output = ran(ampl,seed);

The noise signal is limited to a maximum frequency, which can be adjusted (parameter). Values in-between two noise samples are interpolated (first order hold).

Interface - frequency_limited_FOH

Outputs	**Description**
output	
Parameters	
seed	random seed
amplitude	-amplitude <= output <= +amplitude
frequency	frequency band of the noise

Description - frequency_limited_ZOH

This model generates a random (noise) signal with values between -amplitude and +amplitude and random seed s.

output = ran(ampl,seed);

The noise signal is limited to a maximum frequency, which can be adjusted (parameter). Values in-between two noise samples are interpolated (first order hold).

Interface - frequency_limited_ZOH

Outputs	**Description**
output	
Parameters	
seed	random seed
amplitude	-amplitude <= output <= +amplitude
frequency	frequency band of the noise

Description - unlimited

This model generates a random (noise) signal with values between -amplitude and +amplitude and random seed s.

output = ran(ampl,seed);

The noise signal has no frequency limitation.

Note

It is advised only to use this implementation with fixed step integration methods such as Euler and Runge Kutta 2/4. If you run this implementation with a variable step integration method, simulations may get very slow!

Interface - unlimited

Outputs	**Description**
output	

Parameters

seed	random seed
amplitude	-amplitude <= output <= +amplitude

Limitations

seed must be a number in the region <0,65000>.

SignalGenerator-Step

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This models yields a step signal:

output = 0; (time < start time)
output = amplitude; (time >= start_time)

Interface

Outputs	**Description**
output	

Parameters

start_time	The start time of the step.
amplitude	The amplitude of the step.

SignalGenerator-StepTime

Library

Signal\Sources

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

This model generates an output signal equal to the simulator time step.

Interface

Outputs	Description
output	Simulated time.

Signalgenerator-Sweep

Library

Signal\Sources

Implementations

Default
Logarithmic
ZeroMean

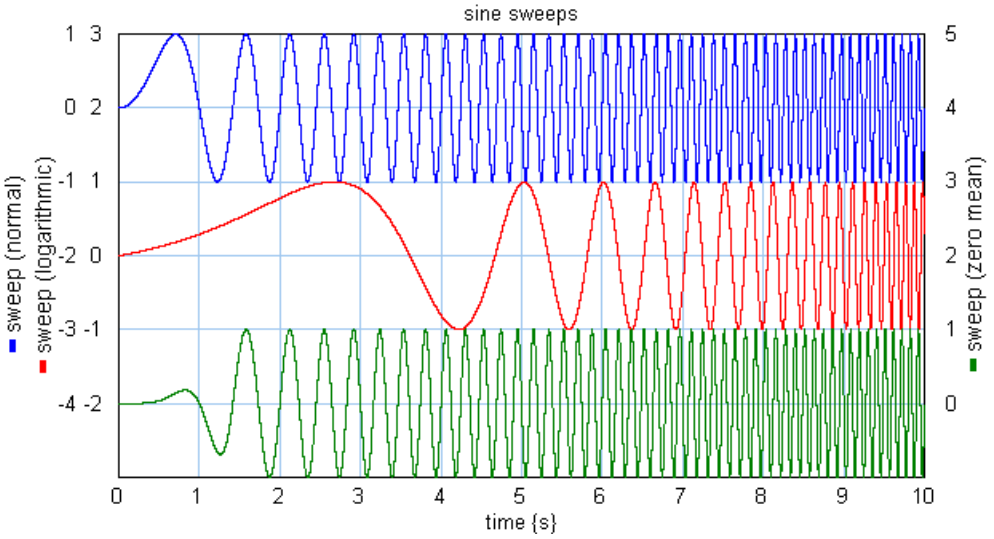
Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description - Default

The output of this model is a linear sine sweep. The signal starts with angular frequency "omega_start" and ends at an angular frequency "omega_stop".

In the picture below the three available sweep functions are shown. The curve on top shows the linear sine sweep. The middle curve shows the logarithmic sine sweep and the bottom curve shows the linear sine sweep with zero mean.



Interface - Default

Outputs	Description
output	

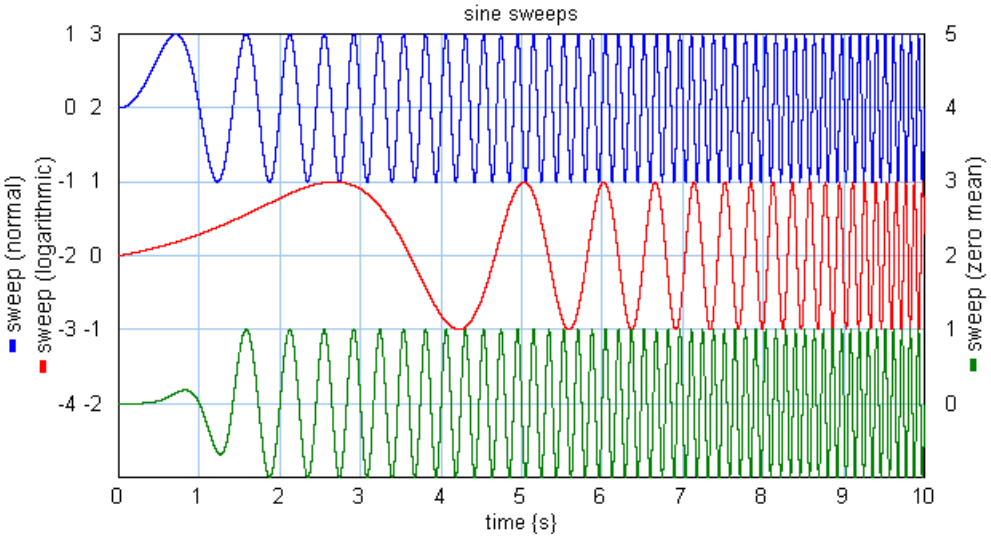
Parameters

start_time	The start time of the sweep.
stop_time	The stop time of the sweep.
amplitude	The amplitude of the sweep.
omega_start	The start frequency of the sweep.
omega_stop	The stop frequency of the sweep.

Description - Logarithmic

The output of this model is a logarithmic sine sweep. Compared to the linear sine sweep the lower frequencies are stretched out and the higher frequencies are squeezed. The signal starts with angular frequency "omega_start" and ends at an angular frequency "omega_stop".

In the picture below the three available sweep functions are shown. The curve on top shows the linear sine sweep. The middle curve shows the logarithmic sine sweep and the bottom curve shows the linear sine sweep with zero mean. The middle curve clearly uses more time for the lower frequencies and less time for the higher frequencies.



Interface - Logarithmic

Outputs	**Description**
output	

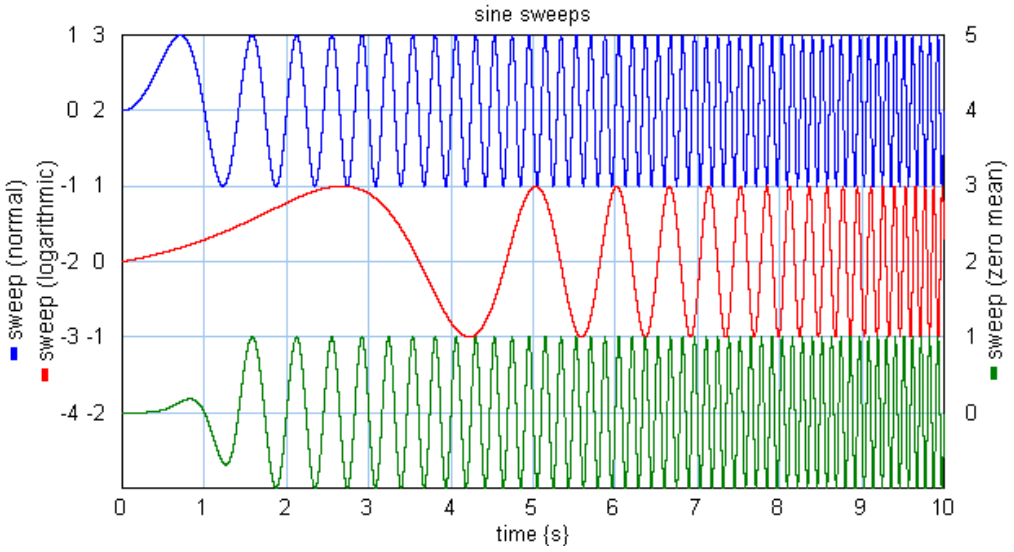
Parameters

start_time	The start time of the sweep.
stop_time	The stop time of the sweep.
amplitude	The amplitude of the sweep.
omega_start	The start frequency of the sweep.
omega_stop	The stop frequency of the sweep.

Description - ZeroMean

The output of this model is a linear sine sweep with compensation in the first period to get a zero mean output. A zero mean is important when the signal is integrated twice. To prevent drift the mean should be zero. Double integrals can occur in physical systems. For example when a force sweep is applied to a mechanical system and the position is measured, the mean force should be precisely zero to prevent the system from drifting away. The signal starts with angular frequency "omega_start" and ends at an angular frequency "omega_stop".

In the picture below the three available sweep functions are shown. The curve on top shows the linear sine sweep. The middle curve shows the logarithmic sine sweep and the bottom curve shows the linear sine sweep with zero mean. The bottom curve differs slightly from the top curve in the first period.



Interface - ZeroMean

Outputs	**Description**
output	
Parameters	
start_time	The start time of the sweep.
stop_time	The stop time of the sweep.
amplitude	The amplitude of the sweep.
omega_start	The start frequency of the sweep.
omega_stop	The stop frequency of the sweep.

SignalGenerator-Time

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates an output signal equal to the (simulated) time.

output = time;

Interface

Outputs	Description
output	Simulated time.

SignalMonitor

Library

Signal\Block Diagram , Signal\Sources

Use

Domains: Continuous, Discrete. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model shows the value of its input in the Simulator. When you open the Simulator (select Properties and Plot) the input of this model is automatically selected as plotvariable. As label for this variable the local name of the SignalMonitor model is chosen. It is therefore advised to give each SignalMonitor model a useful name (select the model, click the right mouse button and select Attributes from the right mouse menu).

Interface

Inputs	Description
input	The value of the input is shown in the Simulator using the local name of the model as label.

WaveGenerator-Cosine

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates a cosine wave.

*output = amplitude * cos(omega * time);*

Interface

Outputs	Description

output

Parameters

amplitude	The amplitude of the wave.
omega	The angular frequency of the wave.

WaveGenerator-PhasedSine

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates a sine wave with phase shift.

*output = amplitude * sin(omega * time + phase);*

Interface

Outputs	**Description**
output	

Parameters

amplitude	The amplitude of the wave.
omega	The angular frequency of the wave.
phase	The phase shift of the wave.

WaveGenerator-Saw

Library

Signal\Source

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates a saw wave.

output = upward ramp; (each cycle)

Interface

Outputs	**Description**
output	
Parameters	
amplitude	The amplitude of the wave.
omega	The angular frequency of the wave.

WaveGenerator-Sine

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates a sine wave.

*output = amplitude * sin(omega * time);*

Interface

Outputs	**Description**
output	
Parameters	
amplitude	The amplitude of the wave.
omega	The angular frequency of the wave.

WaveGenerator-Square

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates a block wave:

output = amplitude; (first wave halve)
output = 0; (second wave halve)

Interface

Outputs	**Description**
output	
Parameters	
amplitude	The amplitude of the wave.
omega	The angular frequency of the wave.

WaveGenerator-SquareExp

Library

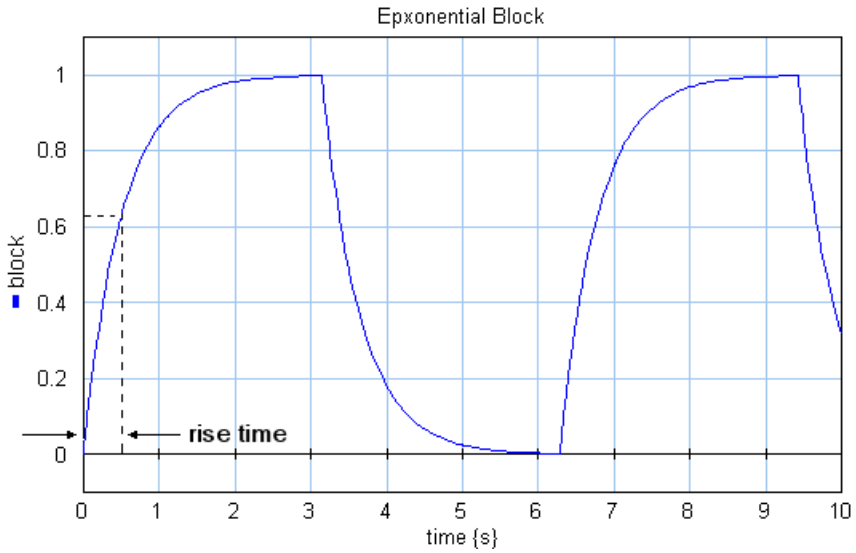
Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates a non-ideal block wave. The steepness of the edges are defined by the rise time. The rise time is defined as 63% of the amplitude being reached.



The output can be described as an exponential function:

output = amplitude(1-exp(time/rise_time)); (first wave halve)*
*output = amplitude*exp(time/rise_time); (second wave halve)*

Interface

Outputs	**Description**
output	
Parameters	
amplitude	The amplitude of the wave.
omega	The angular frequency of the wave.
rise_time	Rise time

WaveGenerator-Triangle

Library

Signal\Sources

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates a triangular wave.

output = upward ramp; (first wave halve)
output = downward ramp; (second wave halve)

Interface

Outputs	**Description**
output	
Parameters	
amplitude	The amplitude of the wave.
omega	The angular frequency of the wave.

Zero

Library
Signal\Sources
Use
Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
The output of this model is equal to zero.

output = 0.0;

Interface

Outputs	**Description**
output	

9.3.12 Stochastic

Mean

Library
Signal\Signal Processing
Use
Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
This submodel yields the mean of an input signal using the integral function:

output = int(input)/time

Interface

Inputs	**Description**
input	Input signal
Outputs	
output	Mean value of the input signal

SignalGenerator-Gaussian Noise

Library

Signal\Sources

Implementations

frequency_limited_FOH
frequency_limited_ZOH
unlimited

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description - frequency_limited_FOH

This model generates a gaussian noise signal with gaussian distribution and variance 1 and random seed s. The noise is multiplied by the parameter amplitude.

*output = amplitude*gauss(1,seed);*

The noise signal is limited to a maximum frequency, which can be adjusted (parameter). Values in-between two noise samples are interpolated (first order hold).

Interface - frequency_limited_FOH

Outputs	**Description**
output	
Parameters	
seed	random seed
amplitude	-amplitude <= output <= +amplitude
frequency	frequency band of the noise

Description - frequency_limited_ZOH

This model generates a gaussian noise signal with gaussian distribution and variance 1 and random seed s. The noise is multiplied by the parameter amplitude.

*output = amplitude*gauss(1,seed);*

The noise signal is limited to a maximum frequency, which can be adjusted (parameter). Values in-between two noise samples are interpolated (first order hold).

Interface - frequency_limited_ZOH

Outputs	**Description**
output	
Parameters	
seed	random seed

amplitude	-amplitude <= output <= +amplitude
frequency	frequency band of the noise

Description - unlimited

This model generates a gaussian noise signal with gaussian distribution and variance 1 and random seed s. The noise is multiplied by the parameter amplitude.

*output = amplitude*gauss(1,seed);*

The noise signal has no frequency limitation.

Note

It is advised only to use this implementation with fixed step integration methods such as Euler and Runge Kutta 2/4. If you run this implementation with a variable step integration method, simulations may get very slow!

Interface - unlimited

Outputs	**Description**
output	

Parameters

seed	random seed
amplitude	-amplitude <= output <= +amplitude

Limitations

seed must be a number in the region <0,65000>.

Signalgenerator-Random

Library

Signal\Sources

Implementations

frequency_limited_FOH
frequency_limited_ZOH
unlimited

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description - frequency_limited_FOH

This model generates a random (noise) signal with values between -amplitude and +amplitude and random seed s. The

output = ran(ampl,seed);

The noise signal is limited to a maximum frequency, which can be adjusted (parameter). Values in-between two noise samples are interpolated (first order hold).

Interface - frequency_limited_FOH

Outputs	**Description**
output	
Parameters	
seed	random seed
amplitude	-amplitude <= output <= +amplitude
frequency	frequency band of the noise

Description - frequency_limited_ZOH

This model generates a random (noise) signal with values between -amplitude and +amplitude and random seed s.

output = ran(ampl,seed);

The noise signal is limited to a maximum frequency, which can be adjusted (parameter). Values in-between two noise samples are interpolated (first order hold).

Interface - frequency_limited_ZOH

Outputs	**Description**
output	
Parameters	
seed	random seed
amplitude	-amplitude <= output <= +amplitude
frequency	frequency band of the noise

Description - unlimited

This model generates a random (noise) signal with values between -amplitude and +amplitude and random seed s.

output = ran(ampl,seed);

The noise signal has no frequency limitation.

Note

It is advised only to use this implementation with fixed step integration methods such as Euler and Runge Kutta 2/4. If you run this implementation with a variable step integration method, simulations may get very slow!

Interface - unlimited

Outputs	Description
output	

Parameters	
seed	random seed
amplitude	-amplitude <= output <= +amplitude

Limitations

seed must be a number in the region <0,65000>.

SignalGenerator-RandomInteger

Library

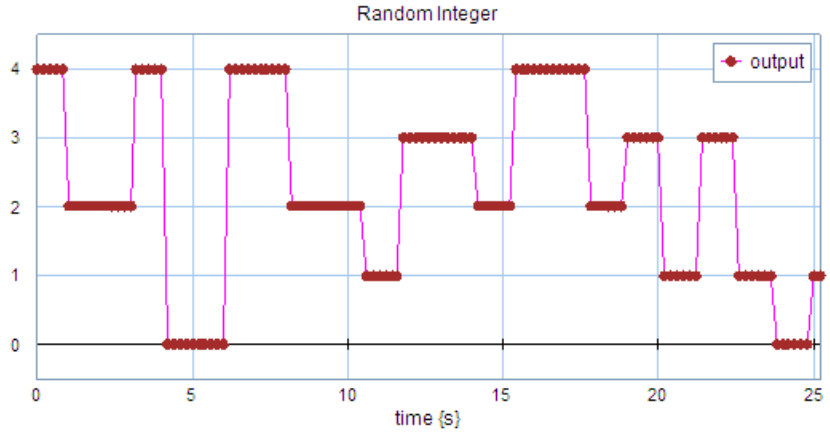
Signal\Stochastic

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates uniformly distributed random integers in the range [0, *m*-1], where *m* is an integer. All output random variables are independent and identically distributed. With the parameter random seed you can choose to make the distribution variable or fixed for each simulation run.



RandomInteger output with a frequency of 1 Hz and m = 5.

The output signal is limited to a maximum frequency, which can be adjusted (parameter). Values in-between two samples are interpolated (first order hold).

Interface

Outputs	Description
output	

Parameters	
seed	random seed
m	the output range [0, m-1]
frequency	frequency band of the output

Limitations

seed must be a number in the region <0,65000>.

SignalGenerator-VariableBlock

Library

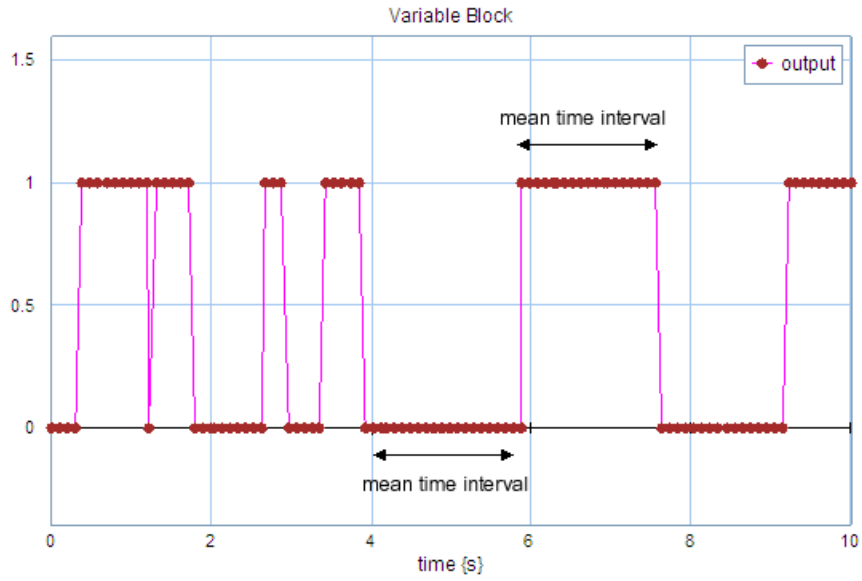
Signal\Stochastic

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates a block signal that alternates between 0 and 1. The mean time interval of a section is given by the parameter *meanTimeInterval*. The distribution of the time interval is uniform between [0, 2**meanTimeInterval*]. With the parameter random seed you can choose to make the distribution variable or fixed for each simulation run.



VariableBlock output with a mean time interval of 1 s.

Interface

Outputs	Description
output	
Parameters	
meanTimeInterval	mean time {s} before the output signal changes
seed	random seed

Limitations

seed must be a number in the region <0,65000>.

SignalGenerator-VariablePulse

Library

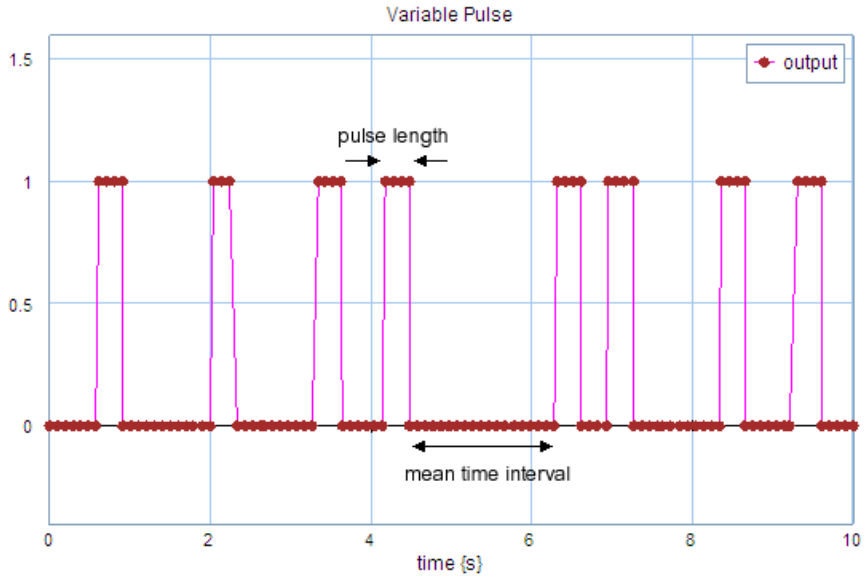
Signal\Stochastic

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description

This model generates a pulse train with a pulse value of 1. The pulse width is given by the parameter *pulseLength*. The mean time interval between two pulses is given by the parameter *meanTimeInterval*. The distribution of the time interval is uniform between [0, 2**meanTimeInterval*]. With the parameter random seed you can choose to make the distribution variable or fixed for each simulation run.



VariablePulse output with a pulse length of 0.3 s and a mean time interval of 1 s.

Interface

Outputs	**Description**
output	
Parameters	
meanTimeInterval	mean time {s} before the output signal changes
pulseLength	length of the pulse {s}
seed	random seed

Limitations

seed must be a number in the region <0,65000>.

StandardDeviation

Library
Signal\Signal Processing
Use
Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
This submodel yields the standard deviation of an input signal using an integral function:

output = sqrt(int((input - mean)2 / time))

where mean is the mean of the input
Interface

Inputs	**Description**
input	Input signal
Outputs	
output	Standard deviation of the input signal

Variance

Library
Signal\Signal Processing
Use
Domains: Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.
Description
This submodel yields the variance of an input signal using an integral function:

output = int((input - mean)2 / time)

where mean is the mean of the input
Interface

Inputs	**Description**
input	Input signal
Outputs	
output	Variance of the input signal

9.3.13 Transfer Functions

Using Transfer Functions

The Transfer Functions library contains a number of predefined transfer functions and a general transfer function model that opens the Linear System Editor.

Without Dead Time

name	formula	description
LinearSystem	-	Any Linear System without time delay
FO	$Y(s)=\frac{k}{\tau s+1}\cdot U(s)$	First Order
FOL	$Y(s)=\frac{k\left(s\tau_L+1\right)}{s\tau+1}\cdot U(s)$	First Order with Lead Time
FOLA	$Y(s)=\frac{k\left(s\tau+1\right)}{s\tau_L+1}\cdot U(s)$	First Order with Lag
SOI	$Y(s)=\frac{k}{s(\tau s+1)}\cdot U(s)$	Second Order with Integrator
SOO	$Y(s)=\frac{k}{(\tau_1 s+1)(\tau_2 s+1)}\cdot U(s)$	Second Order Overdamped
SOOL	$Y(s)=\frac{k(\tau_L s+1)}{(\tau_1 s+1)(\tau_2 s+1)}\cdot U(s)$	Second Order Overdamped with Lead Time
SOU	$Y(s)=\frac{k}{\tau^2 s^2+2\zeta\tau s+1}\cdot U(s)$	Second Order Underdamped
SOUO	$Y(s)=\frac{\omega^2 k}{s^2+2\zeta\omega s+\omega^2}\cdot U(s)$	Second Order Underdamped, Omega description

With Dead Time

name	formula	description
ZOD	$Y(s)=ke^{-\theta s}\cdot U(s)$	Zero Order plus Dead Time

FOD	$Y(s) = \frac{ke^{-\theta s}}{\tau s + 1} \cdot U(s)$	First Order plus Dead Time
SOID	$Y(s) = \frac{ke^{-\theta s}}{s(\tau s + 1)} \cdot U(s)$	Second Order with Integrator plus Dead Time
SOOD	$Y(s) = \frac{ke^{-\theta s}}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot U(s)$	Second Order Overdamped plus Dead Time
SOOLD	$Y(s) = \frac{k(\tau_I s + 1)e^{-\theta s}}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot U(s)$	Second Order Overdamped with Lead Time plus Dead Time
SOUD	$Y(s) = \frac{ke^{-\theta s}}{\tau^2 s^2 + 2\zeta\tau s + 1} \cdot U(s)$	Second Order Underdamped plus Dead Time
SOUDO	$Y(s) = \frac{\omega^2 ke^{-\theta s}}{s^2 + 2\varsigma\omega s + \omega^2} \cdot U(s)$	Second Order Underdamped plus Dead Time, Omega description

Linear System

Library

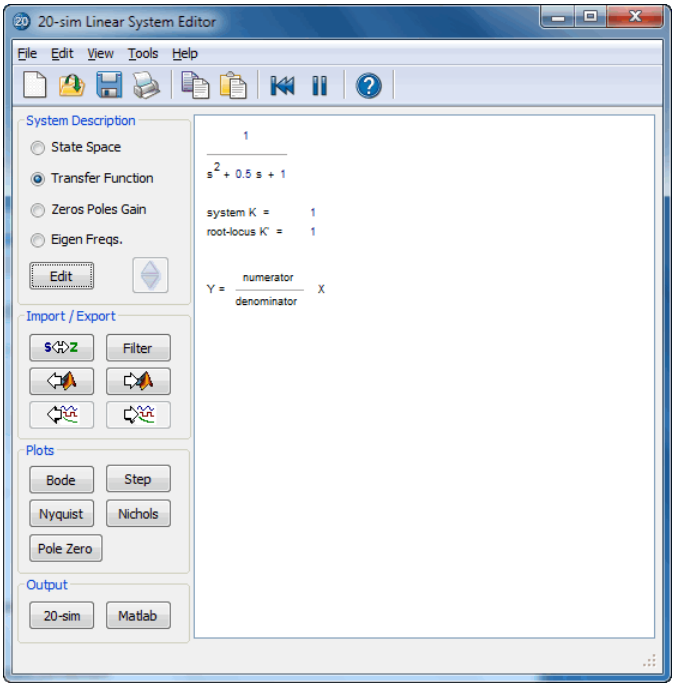
Signal\TransferFunctions

Use

Domains: Continuous. **Size**: 1-D. **Kind**: Block Diagrams.

Description

When you select this model and click *Go Down* a special editor opens (Linear System Editor), allowing you to enter a linear system in State Space form, as a Transfer Function or by adding poles and zeros:



Interface

Inputs	Description
input	
Outputs	
output	
Initial Values	
	The model has internal states that are not accessible.
Parameters	
	Parameters are entered by the Linear System Editor.

TransferFunction

Library

Signal\Transfer Functions

Implementations

FO
FOL
FOLA
SOI
SOO
SOU
SOUO
SOOL

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description - FO

This model describes a first order process determined by k and tau. The transfer function of this process is:

$$Y(s) = \frac{k}{\tau s + 1} \cdot U(s)$$

Interface - FO

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
state_initial	The initial value of the output.
Parameters	
k	Proportional gain [].
tau	Process time constant [s].

Description - FOL

This model describes a first order process with lead time determined by the parameters k, tau and tauL. The transfer function of this process is:

$$Y(s) = \frac{k(s\tau_L + 1)}{s\tau + 1} \cdot U(s)$$

Interface - FOL

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
state_initial	Internal state.
Parameters	
k	Proportional gain [].
tau	Process time constant [s].
tauL	Lead time constant [s].

Description - FOLA

This model describes a first order process with lag determined by the parameters k, tau and tauL. The transfer function of this process is:

$$Y(s) = \frac{k(s\tau + 1)}{s\tau_L + 1} \cdot U(s)$$

Interface - FOLA

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
state_initial	Internal state.
Parameters	
k	Proportional gain [].
tau	Process time constant [s].
tauL	Lag time constant [s].

Description - SOI

This model describes a second order process with integrator determined by the parameters k and tau. The transfer function of this process is:

$$Y(s) = \frac{k}{s(\tau s + 1)} \cdot U(s)$$

Interface - SOI

Inputs	Description
input	
Outputs	
output	
Initial Values	
state1_initial	Internal state.
state2_initial	The initial value of the output.
Parameters	
k	Proportional gain [].
tau	Process time constant [s].

Description - SOO

This model describes an overdamped second order process determined by the parameters k, tau1 and tau2. The transfer function of this process is:

$$Y(s) = \frac{k}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot U(s)$$

Interface - SOO

Inputs	Description
input	
Outputs	
output	
Initial Values	
state1_initial	Internal state.
state2_initial	The initial value of the output.
Parameters	

k	Proportional gain [].
tau1	First process time constant [s].
tau2	Second process time constant [s].

Description - SOU

This model describes an underdamped second order process, determined by the parameters k, tau and zeta. The transfer function of this process is:

$$Y(s) = \frac{k}{\tau^2 s^2 + 2\zeta\tau s + 1} \cdot U(s)$$

Interface - SOU

Inputs	**Description**
input	

Outputs

output

Initial Values

state1_initial	Internal state.
state2_initial	The initial value of the output after the dead time has passed.

Parameters

k	Proportional gain [].
tau	Natural period of oscillation [s].
zeta	Damping factor [].

Description - SOUO

This model describes an underdamped second order process, determined by the parameters k, omega and zeta. The transfer function of this process is:

$$Y(s) = \frac{\omega^2 k}{s^2 + 2\zeta\omega s + \omega^2} \cdot U(s)$$

Interface - SOUO

Inputs	**Description**
input	

Outputs

output

Initial Values

state1_initial	Internal state.
state2_initial	The initial value of the output after the dead time has passed.

Parameters

k	Proportional gain [].
omega	Natural frequency [rad/s].
zeta	Damping factor [].

Description - SOOL

This model describes an overdamped second order process with lead time determined by the parameters k, tau1, tau2 and tauL. The transfer function of this process is:

$$Y(s) = \frac{k(\tau_L s + 1)}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot U(s)$$

Interface - SOOL

Inputs	**Description**
input	

Outputs

output

Initial Values

state1_initial	Internal state.
state2_initial	The initial value of the output.

Parameters

k	Proportional gain [].
tau1	First process time constant [s].
tau2	Second process time constant [s].
tauL	Lead time constant [s].

TransferFunctionWithDeadTime

Library

Signal\Transfer Functions

Implementations

ZOD
FOD
SOID
SOOD
SOUD
SOOLD
SOUDO

Use

Domains: Discrete, Continuous. **Size**: 1-D. **Allowed in**: Block Diagrams.

Description - ZOD

This model describes a first zero process plus dead time determined by k and theta. The transfer function of this process is:

$$Y(s) = ke^{-\theta s} \cdot U(s)$$

Interface - ZOD

Inputs	Description
input	

Outputs

output

Parameters

k	Proportional gain [].
theta	Dead time [s].

Description - FOD

This model describes a first order process plus dead time determined by k tau and theta. The transfer function of this process is:

$$Y(s) = \frac{ke^{-\theta s}}{\tau s + 1} \cdot U(s)$$

Interface - FOD

Inputs	Description
input	
Outputs	
output	
Initial Values	
state_intitial	The initial value of the output after the dead time has passed.
Parameters	
k	Proportional gain [].
tau	Process time constant [s].
theta	Dead time [s].

Description - SOID

This model describes a second order process with integrator plus dead time determined by the parameters k and tau. The transfer function of this process is:

$$Y(s) = \frac{k}{s(\tau s + 1)} \cdot U(s)$$

Interface - SOID

Inputs	Description
input	
Outputs	
output	
Initial Values	
state1_initial	Internal state.
state2_initial	The initial value of the output after the dead time has passed.
Parameters	
k	Proportional gain [].
tau	Process time constant [s].
theta	Dead Time [s].

Description - SOOD

This model describes an overdamped second order process plus dead time determined by the parameters k, tau1 and tau2. The transfer function of this process is:

$$Y(s) = \frac{ke^{-\theta s}}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot U(s)$$

Interface - SOOD

Inputs	Description
input	
Outputs	
output	
Initial Values	
state1_initial	Internal state.
state2_initial	The initial value of the output after the dead time has passed.
Parameters	
k	Proportional gain [].
tau1	First process time constant [s].
tau2	Second process time constant [s].
theta	Dead Time [s].

Description - SOUD

This model describes an underdamped second order process plus Dead Time, determined by the parameters k, tau and zeta. The transfer function of this process is:

$$Y(s) = \frac{ke^{-\theta s}}{\tau^2 s^2 + 2\zeta\tau s + 1} \cdot U(s)$$

Interface - SOUD

Inputs	Description
input	
Outputs	
output	
Initial Values	

state1_initial	Internal state.
state2_initial	The initial value of the output after the dead time has passed.

Parameters

k	Proportional gain [].
tau	Natural period of oscillation [s].
zeta	Damping factor []
theta	Dead Time [s].

Description - SOOLD

This model describes an overdamped second order process with lead time plus dead time determined by the parameters k, tau1, tau2 and tauL. The transfer function of this process is:

$$Y(s) = \frac{k(\tau_L s + 1)e^{-\theta s}}{(\tau_1 s + 1)(\tau_2 s + 1)} \cdot U(s)$$

Interface - SOOLD

Inputs	**Description**
input	

Outputs

output

Initial Values

state1_initial	Internal state.
state2_initial	The initial value of the output after the dead time has passed.

Parameters

k	Proportional gain [].
tau1	First process time constant [s].
tau2	Second process time constant [s].
tauL	Lead time constant [s].
theta	Dead Time [s].

Description - SOUDO

This model describes an underdamped second order process plus Dead Time, determined by the parameters k, omega and zeta. The transfer function of this process is:

$$Y(s) = \frac{\omega^2 k e^{-\theta s}}{s^2 + 2\zeta\omega s + \omega^2} \cdot U(s)$$

Interface - SOUDO

Inputs	**Description**
input	
Outputs	
output	
Initial Values	
state1_initial	Internal state.
state2_initial	The initial value of the output after the dead time has passed.
Parameters	
k	Proportional gain [].
omega	Natural frequency [rad/s].
zeta	Damping factor []
theta	Dead Time [s].

10 Modeling Tutorial

10.1 Friction

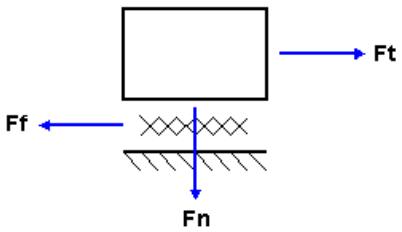
10.1.1 Introduction

In all machines incorporating parts with relative motion, friction is present. Friction is a natural resistance to relative motion between two contacting bodies. At particular locations in machines, friction is an undesirable property. Friction can lead to bad performance of for example a servo system. At other locations, friction plays an important role and is desirable, as is the case for example in actions as walking and grasping and in a brake system.
In most systems, friction has a negative influence on the performance of the system. Under certain conditions, friction in a system can result in effects like a steady state error, tracking errors, especially nearby velocity reversals (reversal bump effect: getting stuck when moving through zero velocity), limit cycles (hunting) and/or stick-slip.

To overcome these negative effects, insight in friction phenomena, and proper models of these phenomena are useful. This chapter will give attention to these aspects of friction and describe the friction models that are available in the 20-sim library.

10.1.2 Normal Force

Assume a mass is dragged over a surface and experiences friction.



A tension force *Ft* has to be applied to overcome the amount of friction force *Ff*. The friction force will depend on the applied normal force as:

*Ft = Fn * f(x,v,...);*

where *f* is the friction function which depends on the displacement *x*, velocity *v* etc.

In most cases a normal force is available (disk breaks, surface friction, clutches etc.). Therefore all models in the 20-sim friction library are described with a normal force input. For those models that do not experience a normal force (bearings), a unity normal force *Fn* = 1 [N] is applied.

10.1.3 Friction Phenomena

There are different phenomena of friction. The following friction phenomena are described in the friction models in 20-sim:

- *static friction*: the torque or force necessary to initiate motion from rest (the so called break-away force); it is often larger than the Coulomb friction,
- *Coulomb friction (kinetic friction, dynamic friction)*: the friction component that is only dependent of the direction of velocity, not of the magnitude of the velocity,
- *viscous friction*: the friction component that is proportional to velocity and goes to zero at zero velocity,
- *Stribeck friction (Stribeck effect)*: the friction phenomenon that arises from the use of fluid lubrication and gives rise to decreasing friction with increasing velocity at low velocity,
- *pre-sliding displacement (Dahl effect):* the spring-like behavior of friction that causes a displacement linear dependent on the applied force if this applied force is less than the break-away force,
- *varying break-away force (rising static friction)*: the dependence of the break-away force on the rate of increase of the applied force,
- *frictional lag*: the delay in the change of the friction force as a function of a change in the velocity.

Beside the friction phenomena mentioned above, some other effects with respect to friction are reported in the literature. These effects are not incorporated into the 20-sim library of friction models:

- *Time-dependent friction*: From experiments it is known that friction changes with time. These changes of friction with time are due to such things as loss of lubricant, deformation of the surface material, change in temperature due to generated heat and/or accumulation of wear debris.
- *Position-dependent friction*: A dependence of the friction on the position of a system is another effect that is experimentally observed and is well known by a lot of researchers. This position-dependency is caused by spatial inhomogeneities in the transmission of the system due to contact geometry and/or loading which varies as a function of position. As the load varies, the normal force between the sliding surfaces varies, causing a varying friction (friction is linear dependent on the normal force). By preloading the transmission elements and roller bearings this dependency of friction on the load can be decreased.
- *Direction-dependent friction*: A lot of researchers have found the friction to be dependent on the direction of the motion of a system. Different Coulomb and viscous friction levels in the left and right directions of a single, linear motion have been observed experimentally on many occasions. Theoretically, this may be due to anisotropies in material or geometry.

10.1.4 Wet and Dry Friction

Wet Friction

Most machines are lubricated in order to decrease the effects of friction. But in most cases, this is not sufficient and significant friction effects remain.

There are four regimes of lubrication in a system with grease or oil in which the phenomena act that are mentioned in the previous section: static friction, boundary lubrication, partial fluid lubrication and full lubrication. Each of these four regimes contribute to the dynamics of the system. In the figure below, these four regimes are given in the plot of the friction force as a function of the sliding velocity, sometimes referred to as the *Stribeck curve*.

A description of the four regimes is:

Regime 1: Static friction and pre-sliding displacement

Although in this regime of static friction there is no sliding, because of the fact that contacts between two surfaces are compliant, there are small motions when a force is applied. The contact between two surfaces can be seen as an elastic contact. The displacement is an approximately linear function of the applied force (a spring-like behavior), up to a critical force at which breakaway occurs and sliding begins. The transition from the elastic contact with pre-sliding displacement in static friction to sliding is not abrupt: first sliding starts at the boundary of a contact and then it propagates toward the center.

Regime 2: Boundary lubrication

In this regime the sliding velocity is very low and is not adequate to build a fluid film of the lubricant between the sliding surfaces. The boundary layer of the surfaces serves to provide lubrication. In most cases, the friction in this boundary lubrication regime is higher than in case of fluid lubrication, as is the situation in regimes three and four.

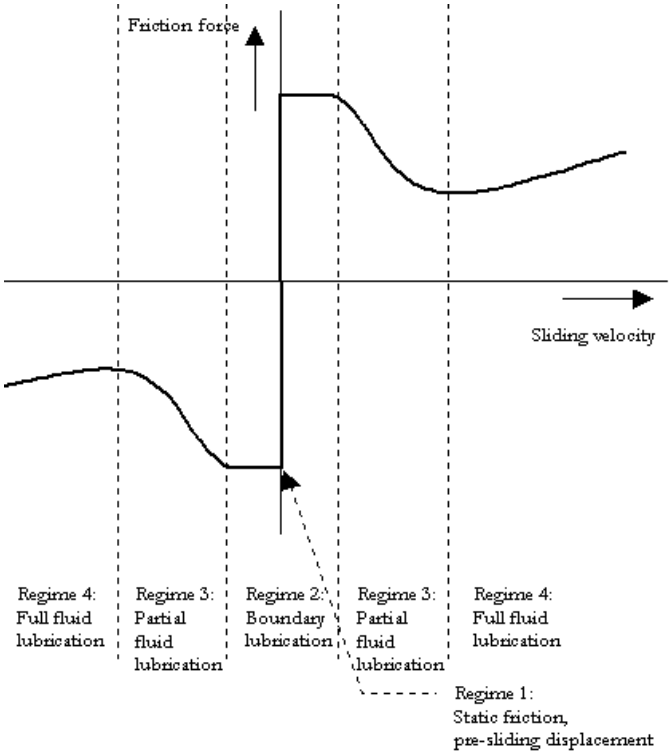
Regime 3: Partial fluid lubrication

Lubricant is brought into the load-bearing region between the sliding surfaces through motion. Some lubricant is expelled by pressure arising from the load, but as a result of the viscosity not all of the lubricant will escape and thus a thin film of lubricant is formed between the sliding surfaces. Not at all places of the contact of the surfaces this film will exist; at some places there is still solid-to-solid contact: there is partial fluid lubrication. As the velocity becomes larger, more lubricant will be brought between the surfaces and less solid-to-solid contacts will exist. This explains the descending friction force when the velocity increases, called the Stribeck effect.

There is a time lag between a change in the velocity or load conditions and the change in friction to a new steady state level: the phenomenon of frictional lag takes place. This effect is a consequence of the state in the frictional contact that does not come to its new equilibrium instantly: it takes time for the lubricant to be brought or removed between the load bearing region of the sliding surfaces.

Regime 4: Full fluid lubrication

In this regime the solid-to-solid contacts are all eliminated. The wear is reduced by orders of magnitude. The viscosity of the lubricant is determinative for the friction force and the friction phenomenon now at work is the well-behaved viscous friction, besides the Coulomb friction.



The four dynamic regimes of friction in the plot of friction force as a function of sliding velocity.

Dry Friction

Dry friction is friction between two bodies in absence of contaminations of the contact surfaces. This is an ideal situation which is not possible to achieve in real systems, due to the fact that chemical reactions will occur at the surfaces. Therefore dry friction will show some of the effects described above. Some special effects due to the use of lubricants, such as the Stribeck effect and sometimes viscous friction will not be significant in dry friction.

10.1.5 Static and Dynamic Phenomena

The friction phenomena can be grouped in static and dynamic phenomena:

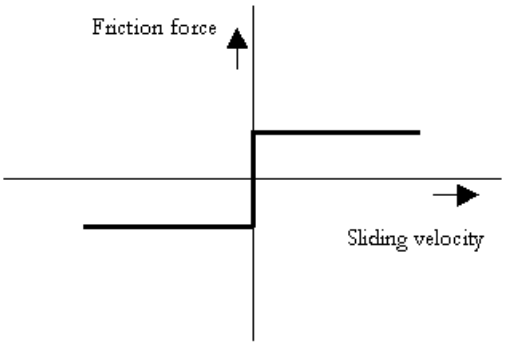
- static friction phenomena:
  - static friction
  - Coulomb friction (dynamic friction)
  - viscous friction
  - Stribeck friction (Stribeck effect)
- dynamic friction phenomena:
  - pre-sliding displacement (Dahl effect)
  - rising static friction (varying break-away force)
  - frictional lag

Static friction models only have a static dependency on velocity. Even when the applied force is very small, the result is a non-zero velocity. This means a significant displacement is found after some time. Therefore static friction models should only be used in systems where the applied forces are large (>> static friction), or change sign often.

Dynamic friction models incorporate a spring like behavior for small forces. I.e. no significant displacement is found when the applied force is smaller that the static friction.

Coulomb friction

Static friction phenomena only have a static dependency on velocity. The first static friction model was the classic model of friction of Leonardo Da Vinci: friction force is proportional to load, opposes the direction of motion and is independent of contact area. Coulomb (1785) further developed this model and the friction phenomena described by the model became known as *Coulomb friction*. The model is given in the figure below.



The Coulomb friction model.

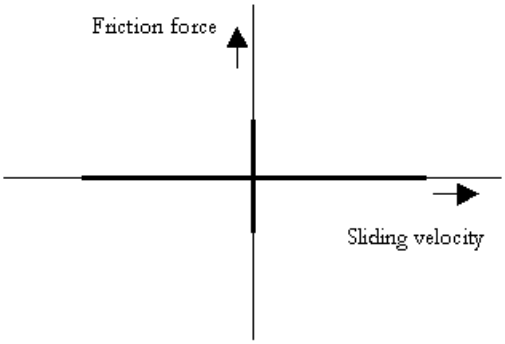
The friction force can be described as:

*Ff = Fn * mu_c * sign (v);*

with *mu_c* the *Coulomb friction coefficient*. The Coulomb friction model is, because of its simplicity, often used. In many textbooks is is also referred to as *dynamic friction* and mu_s described as the *dynamic friction coefficient*.

Static Friction

Morin (1833) introduced the idea of static friction: friction force opposes the direction of motion when the sliding velocity is zero.



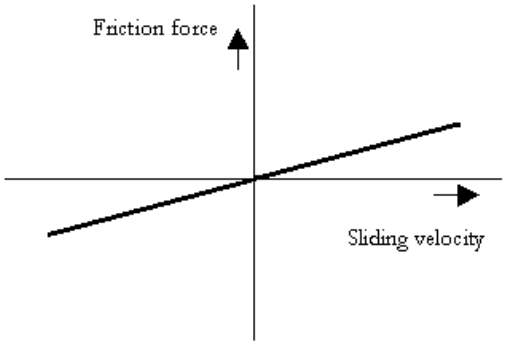
The static friction force is equal to the tensile forces until a maximum or minimum is reached:

*Ff_max = Fn * mu_s;*
*Ff_min = -Fn * mu_s;*

with *mu_s* the *static friction coefficient*.

Viscous friction

Reynolds (1866) developed expressions for the friction force caused by the viscosity of lubricants. The term viscous friction is used for this friction phenomenon.



The viscous friction model.

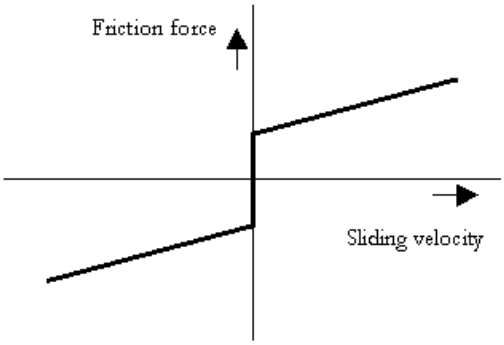
The friction force can be described as:

*Ff = Fn * mu_v * v;*

with *mu_v* the **viscous friction coefficient**.

Coulomb plus Viscous friction

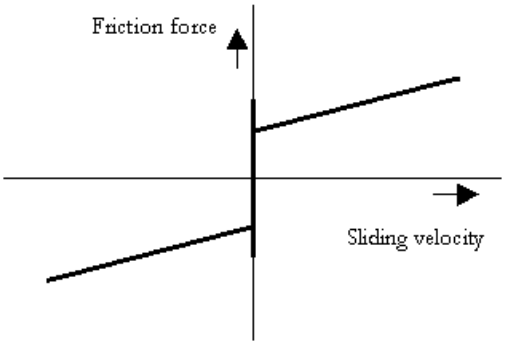
This viscous friction combined with the Coulomb friction gave the model given in the figure below.



The Coulomb plus viscous friction model.

Static plus Coulomb plus Viscous friction

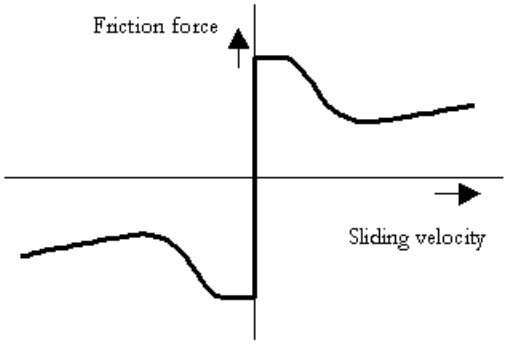
When static friction is added, a friction model appears that is commonly used in engineering: the *Static plus Coulomb plus Viscous friction model*. This model is depicted in the figure below.



The static, Coulomb plus viscous friction model.

Static plus Coulomb plus Viscous plus Stribeck friction

Stribeck (1902) observed that for low velocities, the friction force is decreasing continuously with increasing velocities and not in a discontinuous matter as described above. This phenomenon of a decreasing friction at low, increasing velocities is called the Stribeck friction or effect. The model including *Static, Coulomb, Viscous and Stribeck friction* is given below.



The static, Coulomb, viscous plus Stribeck friction model.

The Stribeck effect is described by the parameter *v_st* which is the characteristic Stribeck velocity. It denotes the sliding velocity where only a 37% of the static friction is active. In other words, small values of *v_st* give a fast decreasing Stribeck effect and large values of *v_st* give a slow decreasing Stribeck effect.

Continuous Functions

All these friction models above show a discontinuity at zero velocity. At zero velocity the friction force as a function of only velocity is not specified. What for example should the friction force be at zero velocity in the Coulomb friction model? The discontinuous behavior of this model near zero velocity is sometimes approximated with a continuous function, for example:

```
F = tanh(slope*v);
```

and the inverse

```
v = arctanh(F)/slope;
```

with *slope* a very large constant. The use of continuous functions results in straightforward models that are easy to simulate for large forces and velocities. For small forces and velocities, the the slope parameter has to be chosen sufficiently high, to avoid creep (the model starts to move, even when the applied force is smaller than the static or Coulomb friction force). This results in stiff models, which are hard to simulate.

LuGre friction model

The LuGre model is inspired by the bristle interpretation of friction (surfaces are very irregular at the microscopic level and two surfaces make contact at a number of asperities, which can be thought of as elastic bristles), in combination with lubricant effects.

Friction is modeled as the average deflection force of elastic springs. When a tangential force is applied the bristles will deflect. If the deflection is sufficiently large, the bristles start to slip. The average defection at steady state slip is determined by the velocity, so that for low velocities the steady state deflection, and therefore the friction force, decreases with increasing velocity. This corresponds to more lubricant being forced into the interface, and pushing the surfaces apart, as the velocity increases. This produces the Stribeck effect. The model also includes rate dependent friction phenomena such as varying break-away force and frictional lag. The model behaves like a well damped spring with a stiffness at zero speed for small motions.

10.1.6 Comparison of Friction Models

In order to get insight in the friction phenomena, a friction model should incorporate as much friction phenomena and behave as much as real friction. In the table below an overview of the different models of the 20-sim library with respect to the friction phenomena is given.

	Coulomb model	**Coulomb, viscous model**	**static, Coulomb, viscous model**	**static, Coulomb, viscous, Stribeck model**	**LuGre model**
static friction phenomena					
Coulomb	V	V	V	V	V
viscous		V	V	V	V
static			V	V	V
Stribeck				V	V
dynamic friction phenomena					
pre-sliding displacement					V
varying break-away force					V
frictional lag					V

From this overview it can be seen that all friction phenomena are incorporated in the LuGre model. From the literature, it is known that the behavior of the LuGre model is much alike real friction.

For most engineering applications, a static friction model, for example a static, Coulomb, viscous friction model, is sufficient. If all friction effects should be modeled, the LuGre friction model seems best suited.

10.1.7 Literature

1. Altpeter, F., Ghorbel, F., Longchamp, R., *A Singular Perturbation Analysis Of Two Friction Models Applied To A Vertical EDM-Axis*, Motion Control '98, France, September 1998, pp. 7-12

2. Altpeter, F., Ghorbel, F., Longchamp, R., *Relationship Between Two Friction Models: A Singular Perturbation Approach*, Proceedings of the 37th IEEE Conference on Decision & Control, Tampa, Florida USA, December 1998, pp. 1572-1574

3. Altpeter, F., Myszkorowski, M., Longchamp, R., *Identification For Control Of Drives With Friction*, IFAC Conference of Industrial Systems, Belfort, France, May 1997, pp. 673-677

4. Altpeter, F., Necsulescu, D., Longchamp, R., *Friction Modelling And Identification Issues For Electric Drives*, Electromotion '97, Cluj-Napoca, Romania, May 1997, pp. 149-154

5. Armstrong-Hélouvry, B., *Control of Machines with Friction*, Kluwer Academic Publishers, Boston, 1991

6. Armstrong-Hélouvry, B., Dupont, P., Wit, C.C. de, *A Survey of Models, Analysis Tools and Compensation Methods for the Control of Machines with Friction*, Automatica, Vol. 30, No. 7, 1994, pp. 1083-1138

7. Canudas de Wit, C., Olssen, H., Aström, K.J., Lischinsky, P., *A New Model for Control of Systems with Friction*, IEEE Transactions On Automatic Control, Vol. 40, No. 3, March 1995, pp. 419-425

8. Du, H., Nair, S.S., *Modelling and Compensation of Low-Velocity Friction With Bounds*, IEEE Transactions On Control Systems Technology, Vol. 7, No. 1, January 1999, pp. 110-121

9. Gäfvert, M., *Comparison of Two Dynamic Friction Models*, Department of Automatic Control, Lund Institute of Technology, 1996

10. Lischinsky, P., Wit, C.C. de, Morel, G., *Friction Compensation for an Industrial Hydraulic Robot*, IEEE Control Systems, February 1999, pp. 25-32

11. Menon, K., Krishnamurthy, K., *Control of Low velocity friction and gear backlash in a machine tool feed drive system*, Mechatronics, 9, 1999, pp. 33-52

10.2 Bond Graphs

10.2.1 Dynamic Systems

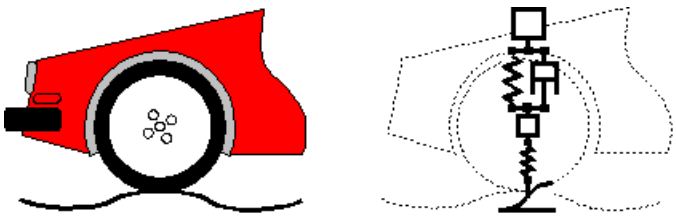
An important aspect in mechatronic systems is the dynamic behavior, i.e. the behavior of the system as function of time. Especially in systems that display swift changes or systems that should behave accurately, the dynamic behavior is important. It is therefore useful to predict the dynamic behavior of a system. Modeling and simulation is useful for making such predictions.

There are various methods of modeling and simulating dynamic systems. A well known one is the *Lumped Parameter Method*.

With the *Lumped Parameter Method*, the dynamic behavior of a system is concentrated in discrete points. The interaction of these points gives us insight in the behavior of the real system. The more discrete points are used the more accurate the model will be.

There are various ways to represent lumped parameter models. Well known representations are iconic diagrams, differential equations, block diagrams and bond graphs.

Let's consider the suspension of a car. A lumped parameter model starts with the identification of the various lumps (or parts or components) of this system. We start with the car body. It is supposed to be a rigid body and therefore this part is represented with the icon of a mass. The suspension of the car is represented by a spring damper combination. The wheel is considered to have a significant mass and to be elastic. This component is therefore represented by a mass and spring icon. Finally the road is modeled by path generator function. The resulting ideal physical model (IPM) is shown below.

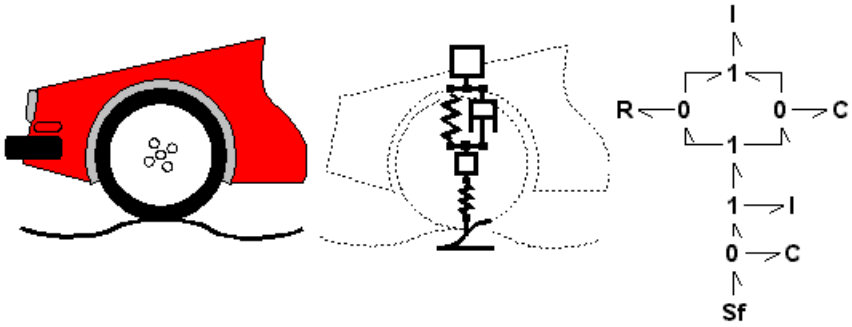


Iconic diagram of a car suspension.

10.2.2 Bond Graphs

Bond graphs are a network-like description of physical systems in terms of ideal physical processes. With the bond graph method we split up the system characteristics into an (imaginary) set of separate elements. Each element describes an idealized physical process. To facilitate drawing of bond graphs, the common elements are denoted by special symbols.

Look again at the car suspension example. In the picture below at the right a bond graph is shown that has been entered in 20-sim. All elements of the ideal physical model have corresponding elements in the bond graph. The connections between the elements in the bond graph, which are known as bonds, represent ideal energy transfer between the elements, i.e. no energy is stored, generated or dissipated. Bonds are drawn with harpoons (the half arrows).



The car suspension model (middle) and corresponding bond graph model (right).

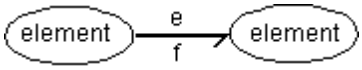
For the mechanical domain, ideal velocity sources are in bond graphs denoted by the symbol *Sf*. Dampers are denoted by an *R*, springs by a *C* and masses by a *I*. With a *1* a structural connection of elements is denoted and with a *0* a velocity difference is denoted.

A bond graph describes a physical system as a number of physical concepts (the elements) connected by energy flows (the bonds).

10.2.3 Effort and Flow

A bond between two elements transfers power from one element to the other. This flow of energy can be described in many ways. For bond graphs a uniform approach is chosen:

The flow of energy between two elements (and thus a bond) is always characterized by two variables, of which the product is power. According to the bond graph notation, these variables are called *effort* (e) and *flow* (f).



Effort and flow as variables of a bond.

The effort and flow variables make up a combination that is typical for a physical domain. The product of effort and flow is always power. We call such a pair of variables *power conjugated variables*. For example voltage and current are used for electrical networks and force and velocity are used for mechanical (translation) systems. The table below shows the variables for the domains that are currently supported in 20-sim.

Domain	**effort (e)**	**flow (f)**
power	effort *e*	flow *f*
mechanical (translation)	force *F* [N]	velocity *v* [m/s]
mechanical (rotation)	torque *T* [Nm]	angular velocity *omega* [rad/s]
pneumatic	pressure *p* [Pa]	volume flow *phi* [m^3/s]
thermal	temperature *T* [K]	entropy flow *dS* [J/Ks]
electric	voltage *u* [V]	current *i* [A]
hydraulic	pressure *p* [Pa]	volume flow *phi* [m^3/s]
magnetic	current *i* [A]	voltage *u* [V]
pseudothermal	temperature *T* [K]	heat flow *dQ* [W]

The effort and flow variables for several domains.

To most general domain is power. Bonds of this domain can connected to elements of all domains. For the thermal domain there is one pair of effort and flow, T and dS, that multiplies to power. The pair T and dQ however, is more often used but does not multiply to power. Therefore the domain with these variables is called pseudothermal.

Note

- There is a direct relation between the across and through variables of an iconic diagram an the effort and flow variables of a bond graph:

Domain	**Non-mechanical domains**	**Mechanical Domains**
across variable	effort	flow
through variable	flow	effort

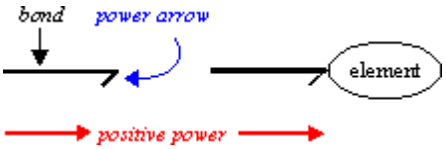
The relation between across and through variables and effort and flow variables.

- 20-sim propagates domains. If a bond of the general power domain is coupled with an element of the electrical domain, it automatically becomes an electrical bond. The other end of the bond can then only be connected to another element of the electrical domain.
- If you want to use a bond of a domain that is not supported or has variables other than across and through, use the general power domain. The across and through variables can then be used as an alias for your own variables.

10.2.4 Bonds

The power of a bond is positive when both effort and flow have a positive value or both effort and flow have a negative value (power = effort × flow).

To denote the orientation, i.e. the direction of positive power, we use the harpoon at the end of a bond. An element with an inward bond connected, consumes power when the product of effort and flow is positive.

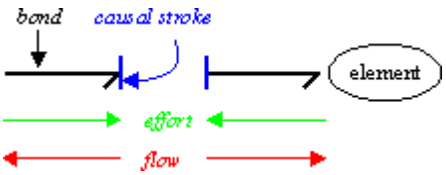


A bond as power connection.

When two elements are connected by a bond, one element will always determine the effort, while the opposite element will always determine the flow. The element that determines the effort, gets an enforced flow from the other element.

We can therefore also see a bond as a bilateral signal connection (effort-signal and flow-signal), of which the directions are opposite to each other.

With *direction* we mean the direction of the flow of information, just like in a block diagram. In a block diagram the effort and flow variables, which together form the flow of power, are not shown as a couple. This breaks up the symmetry between the physical system and the the structure of the model.



A bond as a bilateral signal flow.

The interpretation of a bond as a bilateral signal flow, does not fix the individual direction of the effort and the flow. It only means the direction of the effort and flow are opposite. For the derivation of a simulation model out of a bond graph however, the individual directions are of importance.

To indicate the individual direction of the effort and flow, we use a small stroke (causal stroke) perpendicular to the bond. This stroke indicates the direction of the effort. The direction of the flow is opposite.

The choice of the direction of signals, also known as causality, depends on the element that is connected to the bond.

10.2.5 Standard Elements

To facilitate drawing of bond graphs, the common elements are denoted by special mnemonic symbols. The elements can be divided in several classes:

- Junctions: (0 junction, 1 junction) elements that couple energy between various other elements.
- Buffers: (C,I) elements that store energy.
- Dissipators: (R) elements that dissipate energy.
- Sources: (Se,Sf) elements that generate energy.
- Modulated Sources: (MSe, MSf) elements that generate energy (driven by signals).
- Transformers and Gyrators: (TF, GY) elements that convert energy (ideally).
- Modulated Transformers and Gyrators: (MTF, MGY) elements that convert energy (driven by signals).
- Other elements: other elements and user defined elements.

10.2.6 Orientation

The orientation of a bond, i.e. the direction of the half arrow, denotes the direction of positive power. An element with an incoming bond consumes power when the product of effort and flow is positive. For R, C and I-elements an incoming bond is the standard orientation. For sources the standard orientation is outgoing, because a source supplies power to the rest of the system. For the TF and GY-element the standard orientation is one incoming bond and one outgoing bond, since this reflects the natural flow of power. For the other bonds in a model, as much as possible an orientation from source to load is applied.

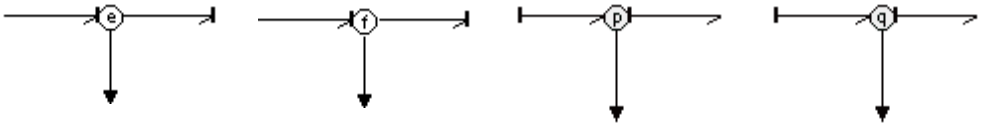
10.2.7 Bonds and Signals

Bonds Graph models are useful when describing systems with powerflow. In mechatronic systems these systems are usually coupled to systems that handle (powerless) signals. These systems are more conveniently described by block diagrams. Bond graphs can be combined with block diagrams. The coupling can be performed by special submodels.

Sensors

In general a sensor measures the flow, the effort, the integral of the flow or the integral of the effort. Sensors can be found in the 20-sim bond graph library.

EffortSensor FlowSensor Psensor Qsensor

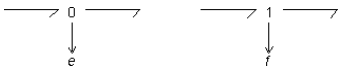


measures effort | measures flow | measures integral off effort | measures integral of flow

Junctions

In 20-sim 1 junctions have a signal output which is equal to the flow. From a 1 junction a signal can be drawn which can be used as input for a block diagram. In a similar way an effort signal can be drawn from a 0 junction.

0 junction | 1 junction

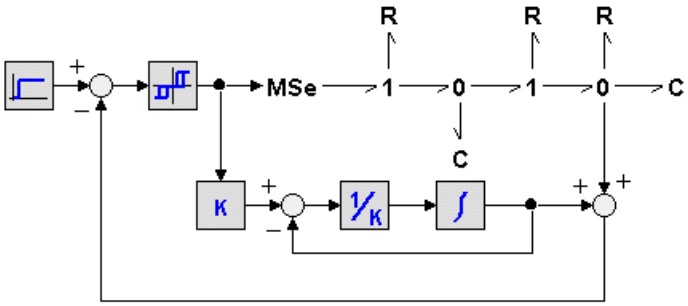


Modulated Elements

The result of a block diagram can be converted into power by means of a generator. In a bond graph model, this can be done by connecting a signal to modulated source elements. These elements convert input signals into efforts or flows. Other elements that use a signal input are modulated transformers and modulated gyrators.

Example

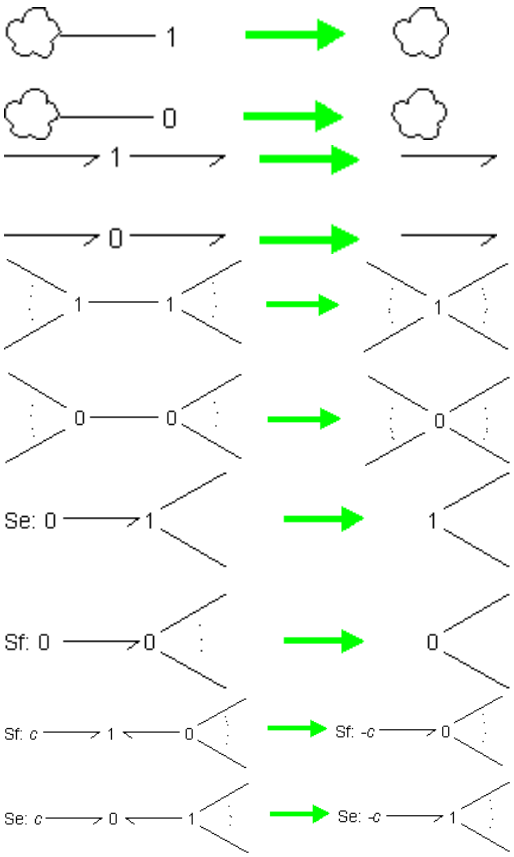
An example model where a bond graph and a block diagram are coupled is shown in the picture below. As can be seen an effort signal is measured at the 0 junction. In the block diagram part it is processed and fed back into the bond graph at the modulated effort source.



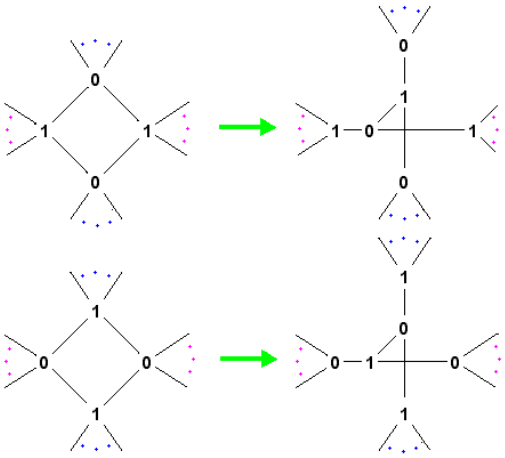
10.2.8 Simplification of Bond Graph Models

When a bond graph model has been created by converting all elements of the iconic diagram into bond graph elements and connecting the elements, simplifications can be performed. This can be easily done by applying the following set of rules (the bonds that have no half arrow, are allowed in both orientations).

1. Eliminate loose junctions.
2. Eliminate junctions.
3. Melt equal junctions.
4. Eliminate sources with a zero output.
5. Eliminate junction in combination with a sign change on the source element.



6. Eliminate a double difference (two 0-junctions coupled with two 1-junctions)



Note

Simplification rules 1, 2, 3 and 6 can be performed automatically in 20-sim using the Simplify Model command.

10.2.9 Causality

Causal analysis is the determination of the direction of the efforts and flows in a bond graph model. The result is a causal bond graph which can be considered as a compact block diagram. From causal bond graph we can directly derive an equivalent block diagram. In 20-sim causality is assigned automatically. To create a causal bond graph 20-sim will perform the following steps.

1. Apply causality for all elements with fixed causality (Se,Sf,Mse,MSf,user defined models), i.e. assign a causal stroke to all bonds connected to elements with fixed causality.

2. Apply,as much as possible, causality for elements that have a constraint causality (0 junction, 1 junction, TF,GY,MTF,MGY,user defined models).

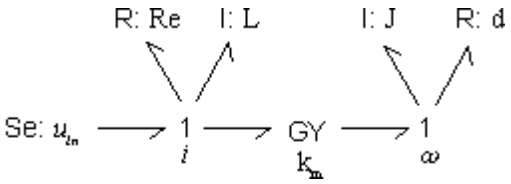
3. Apply causality for an arbitrary element with preferred causality (C,I,user defined models). If possible, let this be the preferred causality. Now iterate step 2 and 3 as far as possible.

4. Apply causality for an arbitrary element with indifferent causality (R,user defined models). If this causality is not fixed, choose a causality arbitrarily. Now iterate step 2, 3 and 4 as far as possible.

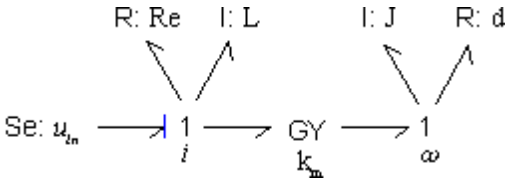
In many cases, the bond graph model is causal after step 2. However, sometimes as *causal conflict* occurs. This means the model is not correct arithmetically. Often this points out an ill-defined model. Redefinition of the ideal physical model will solve the problem. Sometimes the model is only causal after step 4. This means the model contains an *algebraic loop*. Models with algebraic loops are generally hard to simulate.

Example

Causal assignment will be illustrated by an example. We will manually create a causal bond graph out of the following model:

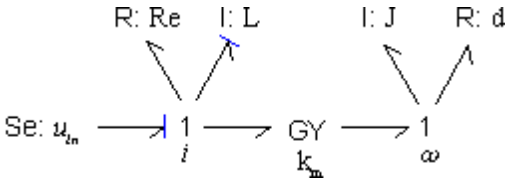


1. Apply fixed causality:

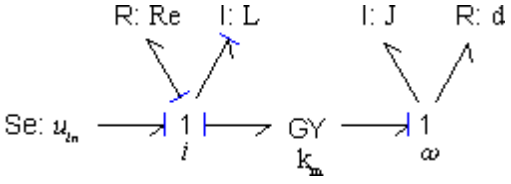


2. Apply constraint causality. This step is not yet applicable.

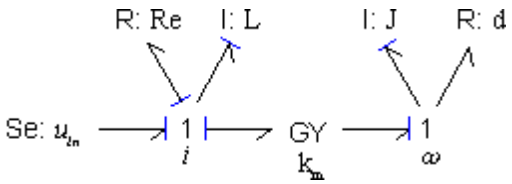
3. Apply preferred causality. We can choose between the two I-elements. Let's select the left one and assign causality in its preferred form:



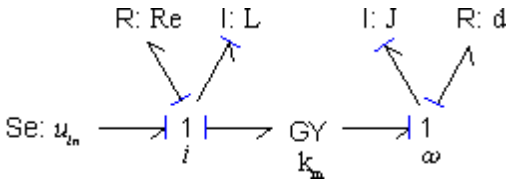
4. Apply constraint causality. The 1 junction on the left has one bond with effort-out causality (seen from the 1 junction). The other bonds connected must therefore have effort-in causality (seen from the 1 junction). In a same way we can now assign causality for the GY-element:



5. Apply preferred causality. Only the right I-element is left. We can assign causality in its preferred form:



6. Apply constraint causality. The 1 junction on the right has one bond with effort-out causality (seen from the 1 junction). The other bonds connected must therefore have effort-in causality. Therefore we can assign causality for the R-element:



10.2.1 Creating a Bond Graph model 0

There are various methods described in literature to convert a ideal physical model into a bond graph. The simplest method is a direct conversion of the parts of the ideal physical model into bond graph elements. It is based on the fact that for every physical domain, junctions have a special interpretation. This allows us to make tables of ideal physical model parts and their direct representation as bond graph elements. The drawback of the method is that it is not generally applicable: for physical domains that are not described here, the user is referred to other methods.

1. Create an ideal physical model.
2. Replace each part of the ideal physical model with corresponding bond graph elements. Standard replacements are shown in the tables of this tutorial.
3. Select the bond graph elements in the 20-sim library and drag and drop them to the 20-sim editor.
4. Connect the elements according to the ideal physical model.
5. Simplify the bond graph according to the given rules.
6. Compile the bond graph model and run a simulation.

10.2.1 From Iconic Diagram to Bond Graph 1

There are various methods described in literature to convert a iconic diagrams into bond graph models. The simplest method is a direct conversion of iconic diagrams into bond graph elements. It is based on the fact that for every physical domain, junctions have a special interpretation. This allows us to make tables of iconic diagrams and their direct representation as bond graph elements. The drawback of the method is that it is not generally applicable: for physical domains that are not described here, the user is referred to other methods.

- Mechanical Domain
- Electrical Domain

10.2.1 Iconic Diagrams to Bond Graphs (Electrical 2 Domain)

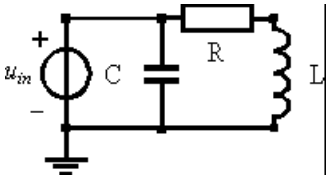
In the electrical domain, terminals and knots can be replaced by 0 junctions. With this knowledge and the description of various icons a table can be created that shows icons of electrical elements and their equivalent bond graph elements. The method of creating a bond graph model is:

1. Create an iconic diagram of the electrical system.
2. Reference points or grounds are split up as much as possible.
3. Replace every element of the diagram by its bond graph equivalent using the conversion table.
4. Replace every knot that has been left by a 0 junction.
5. Connect all elements and junctions with bonds according to the layout of the electrical system.
6. Simplify the resulting bond graph model, using the given set of simplification rules.

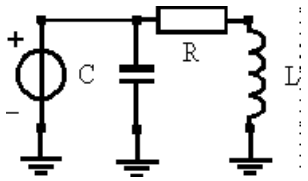
Example

The method will be illustrated by an example. We will convert an electrical circuit into a bond graph model.

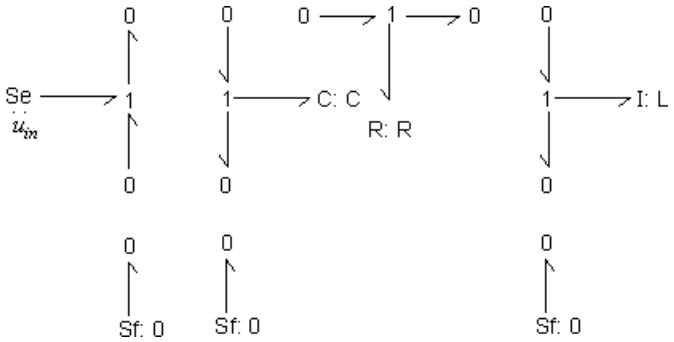
1. Create an iconic diagram



2. Split up reference points.

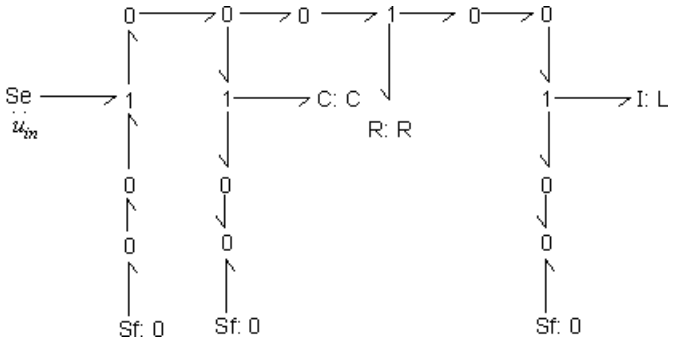


3. Replace elements by bond graph equivalents. Note that we have arranged the orientation of the bonds (the direction of the half arrow) as much as possible in the direction of the power flow from the source to the load elements.

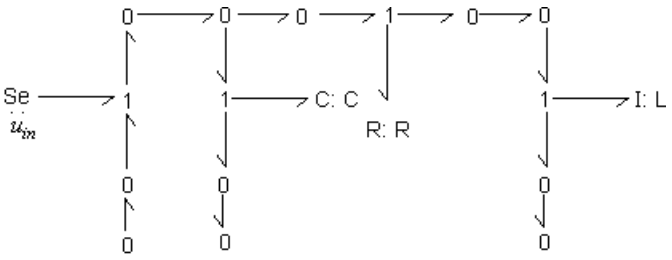


4. Replace every knot that has been left by a 0 junction. In this model no knots have been left. Step 4 is therefore not applicable.

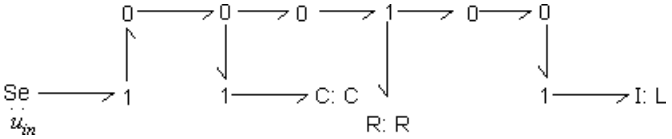
5. Connect all elements and junctions with bonds, according to the layout of the electrical system. A comparison between the iconic diagram and the bond graph model clearly shows the symmetry between both representations. It is obvious that any change or addition in the iconic diagram, can be easily implemented in the bond graph model.



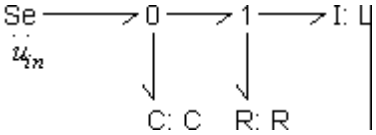
7. Simplify the resulting bond graph model. First we eliminate the Sources (Se elements) with zero output.



8. Eliminate loose junctions.



9. Eliminate junctions.



The simplified model does not show a direct symmetry with the iconic diagram. It does give a clear insight in the flows of power from the voltage source to the other elements.

10.2.1 Iconic Diagrams to Bond Graphs (Mechanical 3 Domain)

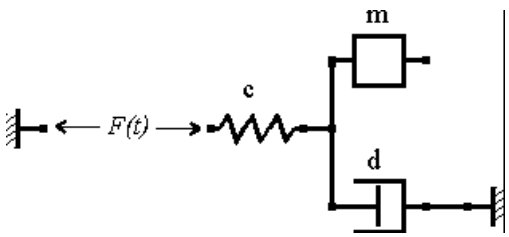
In the mechanical domain, both translational and rotational, terminals and knots can be replaced by 1 junctions. With this knowledge and the description of various icons a table can be created that shows icons of mechanical elements and their equivalent bond graph elements. The method of creating a bond graph model is:

1. Create an iconic diagram of the mechanical system.
2. Reference points or grounds are split up as much as possible.
3. Replace every element of the diagram by its bond graph equivalent using the conversion table.
4. Replace every knot that has been left by a 1 junction.
5. Connect all elements and junctions with bonds, according to the layout of the mechanical system.
6. Simplify the resulting bond graph model, using the given set of simplification rules.

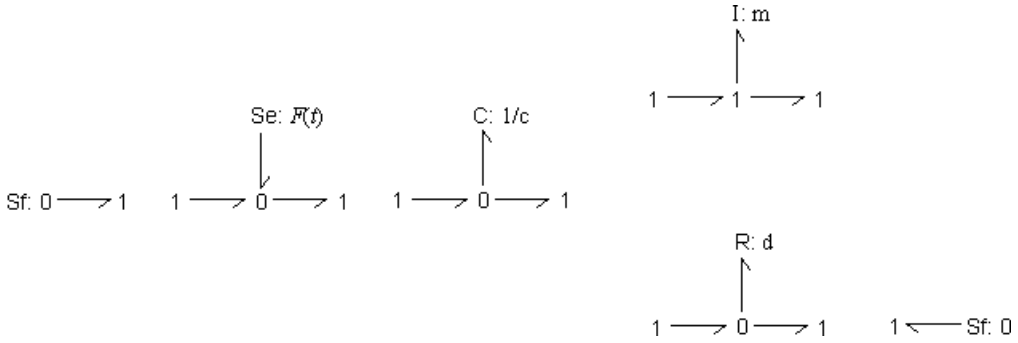
Example

The method will be illustrated by an example. We will convert an mechanical structure (translational) into a bond graph model.

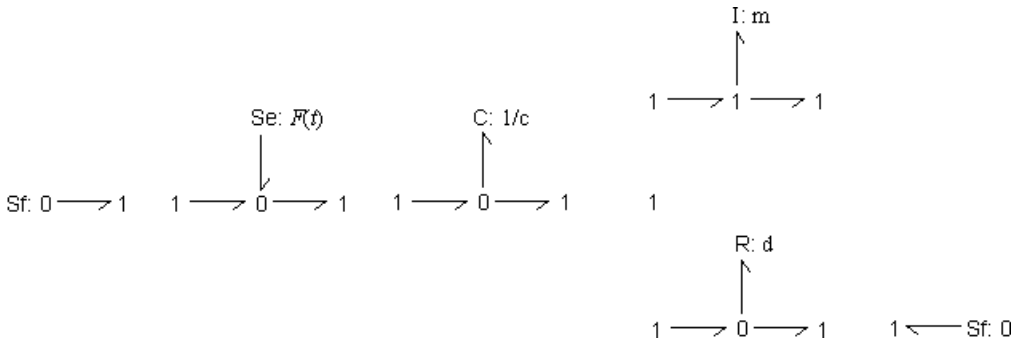
1. Create an iconic diagram



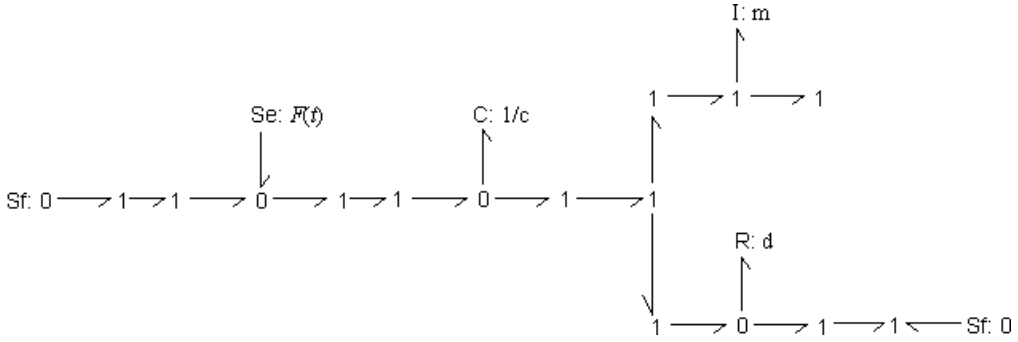
2. Split up reference points. In this diagram this is not possible.
3. Replace elements by bond graph equivalents. Note that we have arranged the orientation of the bonds (the direction of the harpoon) as much as possible in the direction of the power flow from the source to the load elements.



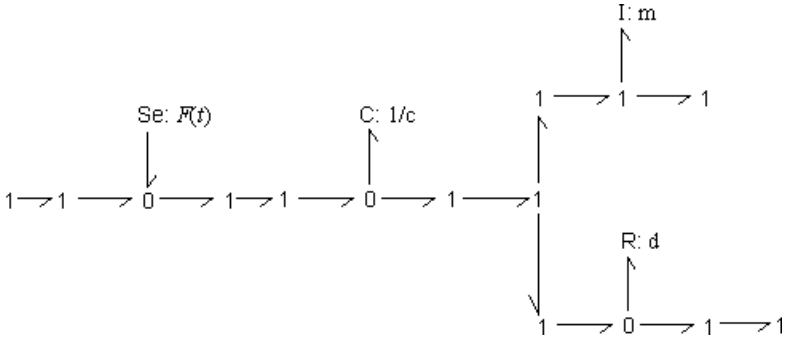
4. Replace every knot that has been left by a 1 junction.



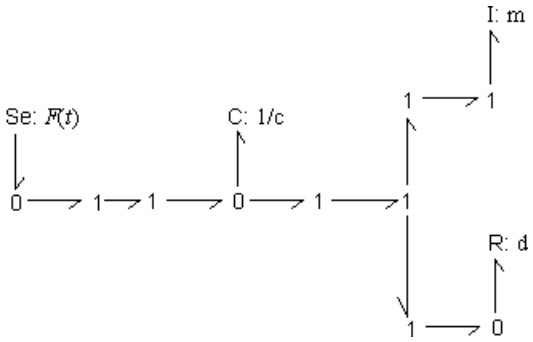
5. Connect all elements and junctions with bonds, according to the layout of the mechanical system. A comparison between the iconic diagram and the bond graph model clearly shows the symmetry between both representations. It is obvious that any change or addition in the iconic diagram, can be easily implemented in the bond graph model.



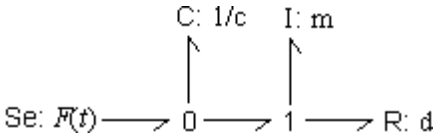
6. Simplify the resulting bond graph model. First we eliminate the Sources (Se elements) with zero output.



7. Eliminate loose junctions.



8. Eliminate junctions.

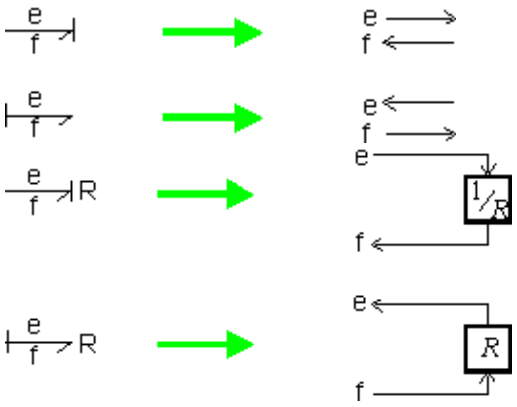


The simplified model does not show a direct symmetry with the iconic diagram. It does give a clear insight in the flows of power from the force source to the other elements.

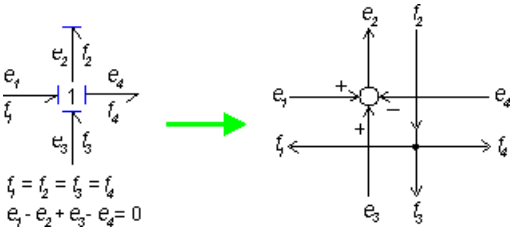
10.2.1 From Bond Graph to Block Diagram 4

With a causal bond graph model, equivalent block diagram models can easily be derived. To create a block diagram, the following steps have to be performed.

- Change bonds by equivalent bilateral signals.
- Replace elements by corresponding block diagram symbols. Use the correct effort and flow description which can be found in the element tables. As an example both descriptions for the R-element are given.

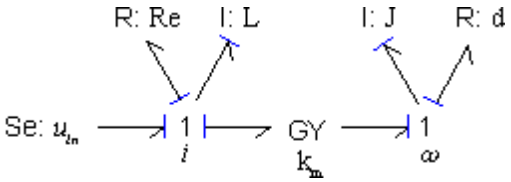


- Replace the junctions by signal summation points and signal splitters. If correct, the bonds have already be replaced by effort and flow signals. Out of these signals and the junction description you can derive the effort and flow equations for the junction. As an example the conversion for a 1 junction is shown.

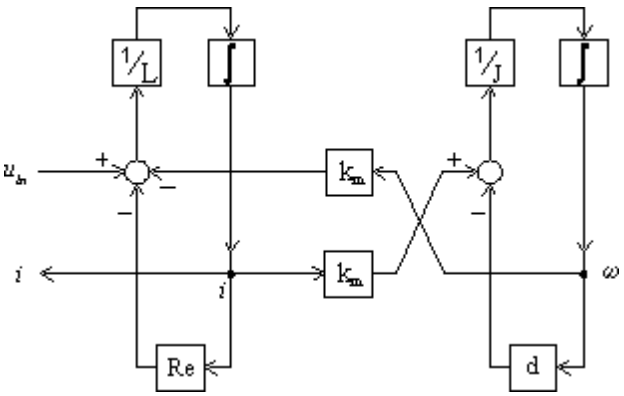


Example

As an example in the figure below a causal bond graph model is shown.



Using the given set of rules and the element descriptions an equivalent block diagram models is found, which is shown below.

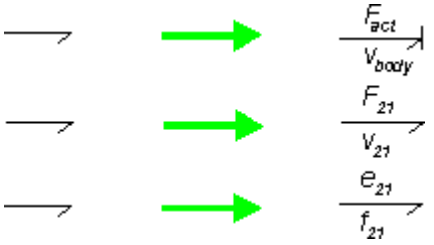


The resulting block diagram model can be simplified by combining blocks and elimination of loops. Out of the block diagram, easily a set of dynamic equations can be deduced.

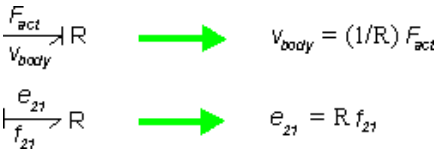
10.2.1 From Bond Graph to Equations 5

With a causal bond graph model, equivalent dynamic equations can easily be derived. To create the dynamic equations, the following steps have to be performed.

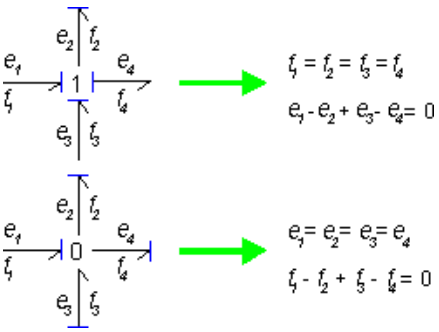
- Denote for every bond its effort and flow pair. You can use names that are obvious or methodically use numbers. Some examples are shown at the right.



- Replace elements by corresponding dynamic equations. Use the correct effort and flow description which can be found in the element tables. As an example descriptions for the R-element are given.



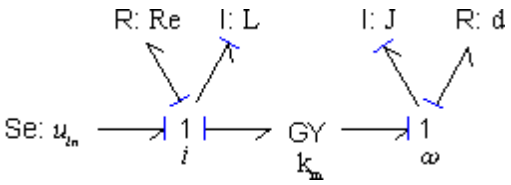
- Replace the junctions by the correct equations. For 0 junctions the efforts are equal. All flows of the bonds pointing towards the 0 junction should be added and all flows of the bonds pointing from the 0 junction should be subtracted. For 1 junctions the flows are equal. All efforts of the bonds pointing towards the 0 junction should be added and all efforts of the bonds pointing from the 0 junction should be subtracted. Some examples are shown at the right.



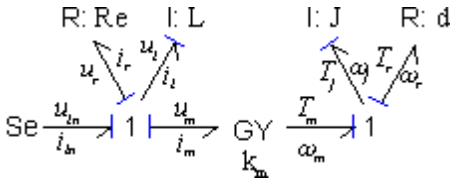
- Rearrange the equations by removing redundant variables.

Example

As an example in the figure below a causal bond graph model is shown. We will derive the set of dynamic equations out of this model.



1. Denote all efforts and flows. This is shown below:



2. Write the dynamic equations of the elements and junctions:

elements	**equations**
Se	u_{in} = constant , i_{in} = free
R	$u_r = Re\ i_r$
I	$i_l = (1/L)\ int(u_l)$
1 junction	$i_{in} = i_r = i_l = i_m$
	$u_{in} - u_r - u_l - u_m = 0$
GY	$u_m = k_m\ \omega_m$
	$T_m = k_m\ i_m$
I	$T_j = (1/J)\ int(\omega_j)$
R	$T_r = d\ \omega_r$
1 junction	$\omega_m = \omega_j = \omega_r$
	$T_m - T_l - T_r = 0$

3. Reduce the amount of equations. We replace the equalities of the 1 junctions: *iin = ir = il = im= i* and *ωm = ωj = ωr= ω*.

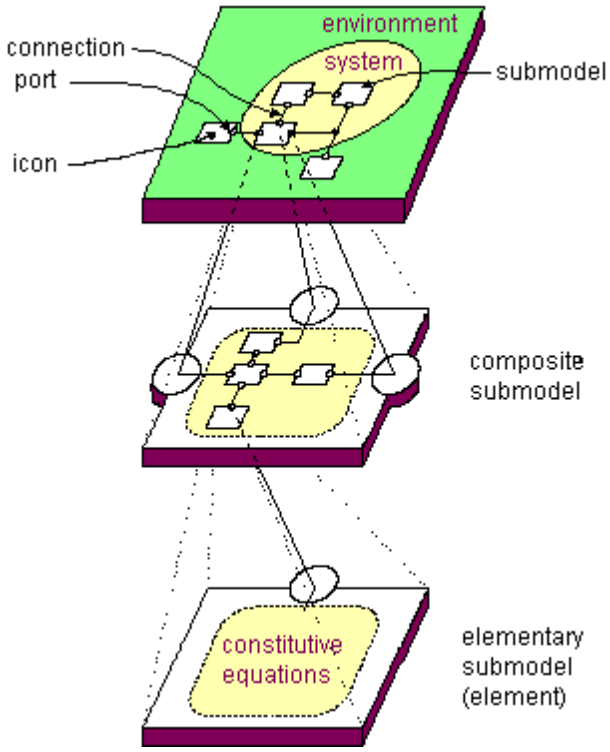
elements	**equations**
Se	u_{in} = constant
R	$u_r = Re\ i$

I	$i = (1/L)\ int(u_l)$
1 junction	$u_{in} - u_r - u_l - u_m = 0$
GY	$u_m = k_m\ \omega$
	$T_m = k_m\ i$
I	$T_j = (1/J)\ int(\omega)$
R	$T_r = d\ \omega$
1 junction	$T_m - T_l - T_r = 0$

The resulting dynamic equations can be used for simulation. 20-sim can automatically extract the dynamic equations out of a bond graph model. The equations can be shown using the Show Equations command or used in the Simulator for simulation.

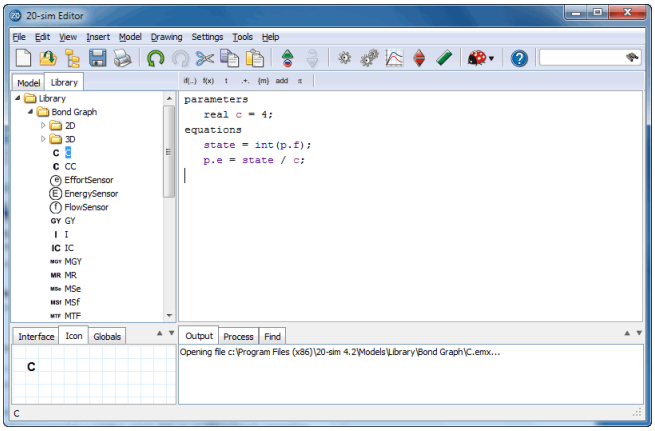
10.2.1 Ports 6

A port is a location where an element can exchange information (in case of a signal port) or power (in case of a power port) with its environment. So, it is the model port that defines the connection with the element. A port is an important concept, as it allows you to describe the properties of the bonds that can be connected to the element, i.e., its direction, size, domain, etc. Ports can be defined in 20-sim using the *Interface Editor*.



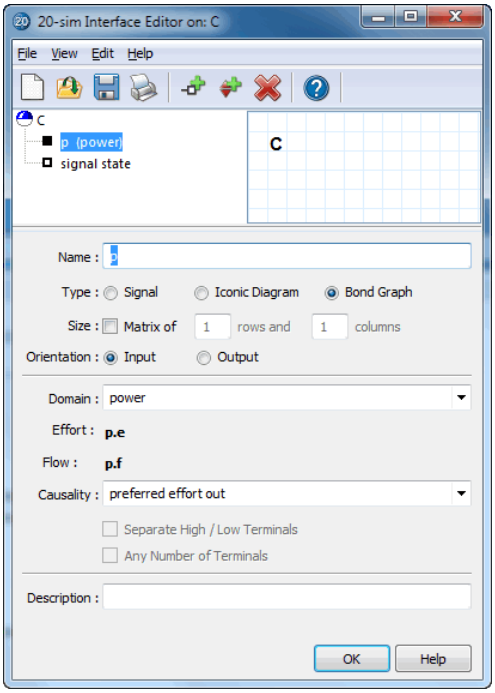
Port Variables

All ports of a submodel are shown in the Interface tab. In the figure below a standard *C-element* is shown with one power port *p*. Each port has an effort and a flow variable. In 20-sim these variables are be denoted with the extensions *.e* and *.f.* You can see an example in the figure below where equations are defined using the variables *p.e* and *p.f.*



Port Properties

Ports can be added and defined in the 20-sim Interface Editor. The *Interface Editor* of the *C-element* is shown below.



Bond graph ports have several properties:

- *Name*: The name of the port.
- *Type*: Next to bond graph ports, 20-sim also knows iconic diagram ports and signal ports

- *Orientation*: The orientation of a connected bond (indicated by the half arrow).
  - *fixed in orientation*: The bond will point towards the element.
  - *fixed out orientation*: The bond will point from the element to another element.
- *Rows/Columns*: The standard size of a port and corresponding bond is 1 but you can also define ports with larger sizes.
- *Domain*: The domain of the port.
- *Causality*: The preferred causality of the port variable (effort and flow). You have to defined here what should be the input variable (effort or flow) and what should be the output variable (effort or flow).

10.2.1 Creating your own Elements 7

In 20-sim you can easily create your own bond graph elements. The process of creation consists of three parts:

1. For any bond connected to your submodel an internal port has to be defined. For each port you have to specify some data:
   - The physical domain.
   - The size.
   - The orientation.
   - The causality.
2. Create the icon for the component. This can be done with a specialized drawing editor.
3. Create the element description. This can be done by (differential) equations or by a bond graph.

10.2.1 Bond Graph Literature 8

A vast number of publications and books on bond graph modeling have been issued. Here is only referred to some well known books. A comprehensive collection of bond graph literature can be found on the Internet pages of prof. F. Cellier (http://www.ece.arizona.edu/~cellier/bg.html) and the mirror site at the Glasgow University (http://www.eng.gla.ac.uk/bg/).

Alan J. Blundell (1982)
Bond Graphs for Modelling Engineering Systems
Ellis Horwood Publishers, Chichester, United Kingdom, and Halsted Press, New York, 151p.

Peter C. Breedveld, Geneviève Dauphin-Tanguy (1992)
Bond Graphs for Engineers

Elsevier Science Publishers, Amsterdam.

François Edouard Cellier (1991)
Continuous System Modeling
Springer-Verlag, New York, ISBN 0-387-97502-0, 755p.

Dean C. Karnopp, Ronald C. Rosenberg (1968)
Analysis and Simulation of Multiport Systems - The Bond Graph Approach to Physical Systems Dynamics
M.I.T. Press.

Dean C. Karnopp, Ronald C. Rosenberg (1974)
System Dynamics: A Unified Approach
John Wiley, New York.

Dean C. Karnopp, Donald L. Margolis, Ronald C. Rosenberg (1990)
System Dynamics: A Unified Approach - 2nd Edition
John Wiley, New York.

Lennart Ljung, Torkel Glad (1994)
Modeling of Dynamic Systems
Prentice Hall, Englewood Cliffs, N.J.

Henry M. Paynter (1961)
Analysis and Design of Engineering Systems
M.I.T. Press, Cambridge, Mass.

Ronald C. Rosenberg, Dean C. Karnopp (1983)
Introduction to Physical System Dynamics
McGraw-Hill, New York.

Nobuhide Suda (1988)
System Dynamics
Koronasha, Tokyo. (in Japanese)

Jean U. Thoma (1982)
Grundlagen und Anwendungen der Bonddiagramme
Girardet Taschenbuch GT20, Essen, Germany.

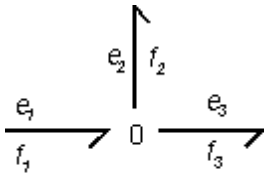
Jean U. Thoma (1989)
Simulation by bond graphs - Introduction to a Graphical Method
Springer-Verlag, New York.

10.2.1 Standard Elements 9

0 and 1 junctions

Junctions couple one or more elements of a model in a power continuous mode: no energy is stored or dissipated. Examples are a series junction or parallel junction in an electrical network, a fixed connection between two mechanical parts etc. Two types of junctions exist: 0 junctions and 1 junctions.

0 junction

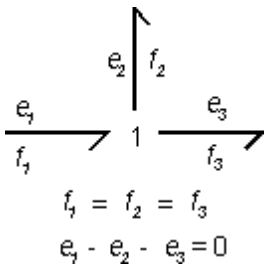


$$e_1 = e_2 = e_3$$

$$f_1 - f_2 - f_3 = 0$$

The 0 junction represents a coupling where all efforts of the connected bonds are equal. As a consequence of the property of power continuity the sum of the flows must be equal to zero. The orientation of the bonds determines the sign of the flow summation: all flows of the bonds pointing towards the 0 junction should be added and all flows of the bonds pointing from the 0 junction should be subtracted. This summation corresponds to the *Kirchhoff current law* for electrical networks. The equality of the efforts, limits the causality. Only one bond may have an "*effort-in*" causality (the stroke pointed towards the 0 junction). All other bonds must have (seen from the 0 junction) an "*effort-out*" causality (stroke pointing from the 0 junction). In other words: the 1 junction has a *constraint causality.*

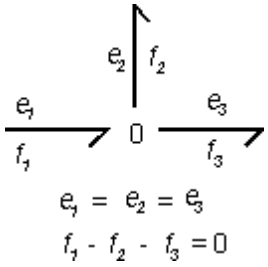
1 junction



The 1 junction is the dual form of the 0 junction (effort and flow are opposite). The 1 junction represents coupling where all flows of the connected bonds are equal. As a consequence of the property of power continuity the sum of the efforts must be equal to zero. The orientation of the bonds determines the sign of the flow summation: all efforts of the bonds pointing towards the 1 junction should be added and all efforts of the bonds pointing from the 1 junction should be subtracted. This summation corresponds to the *Kirchhoff voltage law* for electrical networks. The equality of the flows, limits the causality. Only one bond may have an "*effort-out*" causality (the stroke pointing from the 1 junction). All other bonds must have (seen from the 1 junction) an "*effort-in*" causality (stroke pointing to the 1 junction). In other words: the 1 junction has a *constrained causality.*

0 junction

Suppose we have the following 0 junction with one bond (1) pointing towards the junction and two bonds (2 and 3) pointing from the junction.



1. For a 0 junction the efforts are always equal! This means:

   e1 = e2 = e3

2. The junction is power continuous. For the figure above this means:

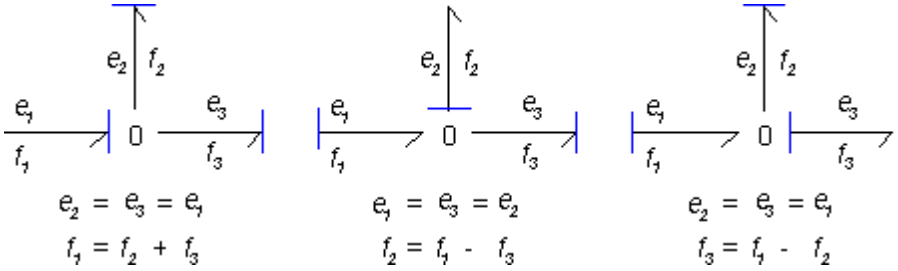
   e1×f1 = e2×f2 + e2×f2

3. Combining 1 and 2 yields:

   f1 - f2 - f3 = 0

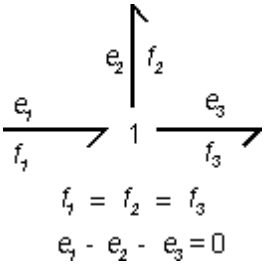
The last equation can also be derived with a rule of thumb: All flows of the bonds pointing *towards* the 0 junction should be added and all flows of the bonds pointing *from* the 0 junction should be subtracted.

The first equation, limits the causality. Only one bond may have an *effort-in* causality (the stroke pointed towards the 0 junction). All other bonds must have (seen from the 0 junction) an *effort-out* causality (strokes pointing from the 0 junction). For the example junction this means three possible causal forms can exist:



1 junction

Suppose we have the following 1 junction with one bond (1) pointing towards the junction and two bonds (2 and 3) pointing from the junction.



1. For a 1 junction the flows are always equal! This means:

   f1 = f2 = f3

2. The junction is power continuous. For the figure above this means:

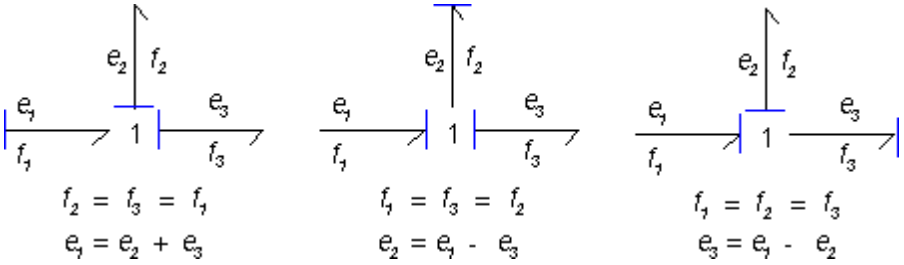
   e1×f1 = e2×f2 + e2×f2

3. Combining 1 and 2 yields:

   e1 - e2 - e3 = 0

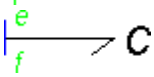
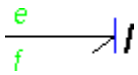
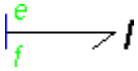
The last equation can also be derived with the rule of thumb: All effort of the bonds pointing *towards* a 1 junction should be added and all efforts of the bonds pointing *from* the 1 junction should be subtracted.

The first equation, limits the causality. Only one bond may have an *flow-in* causality (the stroke pointed from the 1 junction). All other bonds must have (seen from the 1 junction) a *flow-out* causality (strokes pointed towards the 1 junction). For the example junction this means three possible causal forms can exist:



Buffers

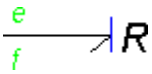
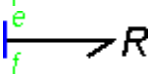
Buffers are bond graph elements that can store energy. There are two types of buffers: *C-elements* and *I-elements*. The table below shows the effort and flow descriptions that belong to these elements. The parameters "C" and "I" are the *bufferconstants*, which determine a linear buffer behavior. Examples of C-elements are a mechanical spring and an electrical capacitor. Examples of an I-element are a mechanical inertia and an electrical inductance.

C-element	**bond graph element**	**equation**
effort-in causality		*f* = C d*e*/d*t*
effort-out causality		*e* = (1/C) int(f)
I-element	**bond graph element**	**equation**
effort-in causality		*f* = (1/I) int(e)
effort-out causality		*e* = I d*f*/d*t*

Buffer elements do not fix the direction of the effort and flow. Both effort-in as well as effort-out causality is allowed. With simulation however, we prefer to avoid differentiation. in other words, with the C-element the effort-out causality is *preferred* and with the I-element the effort-in causality is *preferred*.

Resistance

A resistance, R, dissipates free energy. This energy of any arbitrary domain is transported **irreversibly** to the thermal domain. This means the power towards a resistance is always positive. In the table below the effort and flow description of the resistance is shown. Examples of a resistance are a mechanical damper and an electrical resistor.

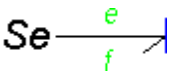
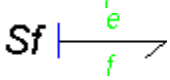
R-element	*bond graph element*	*equation*
effort-in causality		f = (1/R) *e*
effort-out causality		e = R f

The direction of effort and flow is not restricted for a resistance and there is no preferred form for the equation. In other words: the R-element has a *indifferent causality*.

Sources

Sources represent the interaction of a systems with its environment. There are two types of source elements: effort sources (Se) and flow sources (Sf).

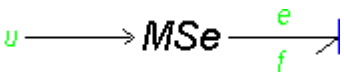
In the table below the effort and flow description of the source elements is shown. Examples of a flow source are an electrical current source and a hydraulic pump that generates a constant flow of liquid. Examples of an effort source are an electrical voltage source and a constant mechanical force.

element	**bond graph element**	**equation**
effort source		*e* = c, *f* = free
flow source		*f* = c, *e* = free

The direction of effort and flow for sources are restricted. In other words: the Se-element (effort-out) and Sf-element (effort-in) have a *fixed causality*.

Modulated Sources

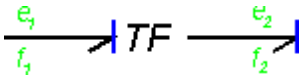
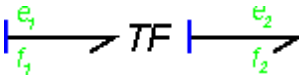
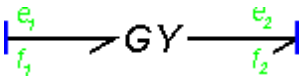
Next to the standard effort and flow sources, that generate a constant effort and flow, bond graph modeling also allows the use of so called **modulated sources**. In these sources the resulting effort or flow are equal to a (fluctuating) value provided by an input signal. Two types of modulated sources are know: the modulated effort source **MSe** and the modulated flow source **MSf**. In the table below the effort and flow description of the source elements is shown.

element	*bond graph element*	*equation*
modulated effort source		$e = u$, f = free
modulated flow source		$f = u$, e = free

The direction of effort and flow for sources are restricted. In other words: the MSe-element (effort-out) and MSf-element (effort-in) have a *fixed causality*.

Transformers and Gyrators

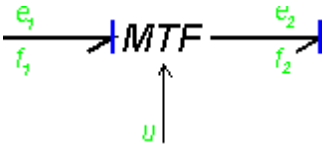
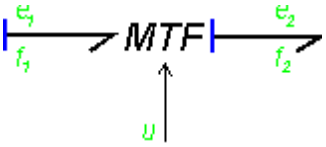
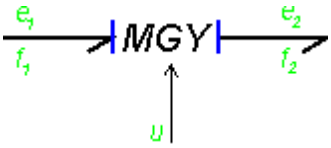
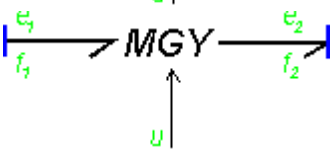
Transformers and gyrators are bond graph elements that can convert energy ideally, as well in one physical domain as well as between one physical domain and another. A transformer is denoted by the mnemonic code **TF** and a gyrator by the code **GY**. In the table below the effort and flow equations are shown. The parameters *r* and *n* are the *transformation ratio* and *gyration ratio*. An examples of a transformer is a mechanical gear. An example of a gyrator is a DC-motor.

TF-element	**bond graph element**	**equation**
effort-in causality		$f_1 = (1/n)\ f_2$ $e_2 = (1/n)\ e_1$
effort-out causality		$f_2 = n\ f_2$ $e_1 = n\ e_2$
GY-element	*bond graph element*	*equation*
effort-in causality		$f_1 = (1/r)\ e_2$ $f_2 = (1/r)\ e_1$
effort-in causality		$e_1 = r\ f_2$ $e_2 = r\ f_1$

The directions of the flow and effort are partly fixed for a transformer. Like the 1 junction and the 0 junction, the causality is *constraint*. For a transformer, an effort-in causality on the incoming bond results in an effort-out causality on the outgoing bond and vice-versa. For a gyrator, an effort-in causality on the incoming bond results in an effort-in causality on the outgoing bond and vice-versa.

Modulated Transformers and Gyrators

Next to transformers and gyrators wit a fixed transformation ratio and gyration ratio, in bond graphs also modulated transformers (**MTF**) and modulated gyrators (**MGY**) are supported. In these models the transformation ratio of gyration ratio are equal to a (fluctuating) value provided by an input signal. In the table below the effort and flow equations are shown.

MTF-element	***bond graph element***	***equation***
effort-in causality		$f_1 = (1/u)\, f_2$ $e_2 = (1/u)\, e_1$
effort-out causality		$f_2 = u\, f_2$ $e_1 = u\, e_2$
MGY-element	*bond graph element*	*equation*
effort-in causality		$f_1 = (1/u)\, e_2$ $f_2 = (1/u)\, e_1$
effort-in causality		$e_1 = u\, f_2$ $e_2 = u\, f_1$

The directions of the flow and effort are partly fixed for a modulated transformer. Like the 1 junction and the 0 junction, the causality is *constraint*. For a transformer, an effort-in causality on the incoming bond results in an effort-out causality on the outgoing bond and vice-versa. For a gyrator, an effort-in causality on the incoming bond results in an effort-in causality on the outgoing bond and vice-versa.

10.3 Iconic Diagrams

10.3.1 Dynamic Systems

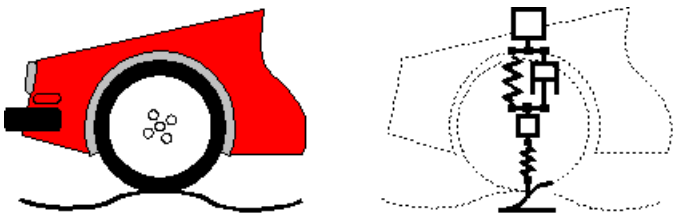
An important aspect in mechatronic systems is the dynamic behavior, i.e. the behavior of the system as function of time. Especially in systems that display swift changes or systems that should behave accurately, the dynamic behavior is important. It is therefore useful to predict the dynamic behavior of a system. Modeling and simulation is useful for making such predictions.

There are various methods of modeling and simulating dynamic systems. A well known one is the *Lumped Parameter Method*.

With the *Lumped Parameter Method*, the dynamic behavior of a system is concentrated in discrete points. The interaction of these points gives us insight in the behavior of the real system. The more discrete points are used the more accurate the model will be.

There are various ways to represent lumped parameter models. Well known representations are iconic diagrams, differential equations, block diagrams and bond graphs.

Let's consider the suspension of a car. A lumped parameter model starts with the identification of the various lumps (or parts or components) of this system. We start with the car body. It is supposed to be a rigid body and therefore this part is represented with the icon of a mass. The suspension of the car is represented by a spring damper combination. The wheel is considered to have a significant mass and to be elastic. This component is therefore represented by a mass and spring icon. Finally the road is modeled by path generator function. The resulting ideal physical model (IPM) is shown below.



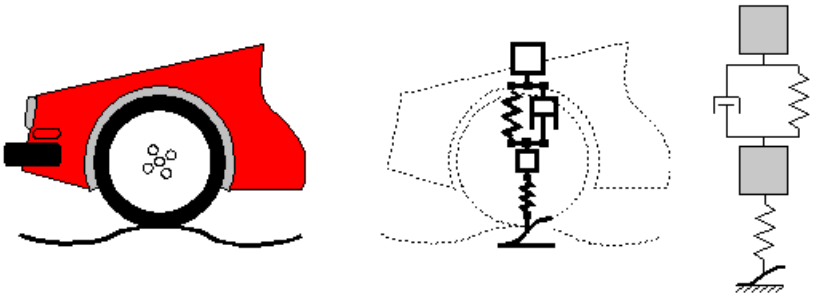
Iconic diagram of a car suspension.

10.3.2 Iconic Diagrams

If we can represent the various parts of an ideal physical model by predefined models or components modeling is easy. In 20-sim a large library of components is available allowing you to create a model just by selecting the proper components and connect them according to the ideal physical model. To make modeling even more easy, the components are displayed in the editor by icons which look like the corresponding parts

of the ideal physical model. In 20-sim these kind of models are called iconic diagrams (sometimes other simulation tools refer to this kind of modeling as component based modeling). 20-sim can automatically compute simulation code out of an iconic diagram and start a simulation.

Look again at the car suspension example. In the picture below at the right an iconic diagram is shown that has been entered in 20-sim. As you can see, the iconic diagram similar to the ideal physical model. Each component of the iconic diagram represents a physical process. For example the mass component represents the law of motion (F = m*a) and the spring represents the linear spring equation (F = k*x). The connections between the components in the iconic diagram represent ideal energy transfer between those components, i.e. no energy is stored, generated or dissipated.



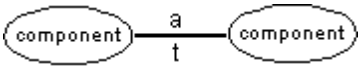
A iconic diagram model of a car suspension.

An iconic diagram describes a physical system as a number of physical concepts (the components) connected by energy flows (the connections).

10.3.3 Across and through

A connection between two components transfers energy from one component to the other. This flow of energy can be described in many ways. In 20-sim a uniform approach is chosen:

The flow of energy between two components of an iconic diagram can be characterized by two variables, of which the product is power. These variables are called *across* (a) and *through* (t).



Across and though as variables of a connection.

The across and though variables make up a combination that is typical for a physical domain. For example voltage and current are used for electrical networks and force and velocity are used for mechanical (translation) systems. The table below shows the variables for the domains that are currently supported in 20-sim.

Domain	**across (a)**	**through (t)**

power	across *a*	through *t*
mechanical (translation)	velocity *v* [m/s]	force *F* [N]
mechanical (rotation)	angular velocity *omega* [rad/s]	torque *T* [Nm]
pneumatic	pressure *p* [Pa]	volume flow *phi* [m^3/s]
thermal	temperature *T* [K]	entropy flow *dS* [J/Ks]
electric	voltage *u* [V]	current *i* [A]
hydraulic	pressure *p* [Pa]	volume flow *phi* [m^3/s]
magnetic	magnetomotoric force *i* [A]	flux change *df/dt* [V]
pseudothermal	temperature *T* [K]	heat flow *dQ* [W]

The across and through variables for several domains.

The most general domain is power. Connections of this domain can be used for all domains. For the thermal domain there is one pair of across and through, T and dS, that multiplies to power. The pair T and dQ however is more often used but does not multiply to power. Therefore the domain with these variables is called pseudothermal.

There is a direct relation between the across and through variables of an iconic diagram and the effort and flow variables of a bond graph:

Domain	**Non-mechanical domains**	**Mechanical Domains**
across variable	effort	flow
through variable	flow	effort

The relation between across and through variables and effort and flow variables.

Note

- 20-sim propagates domains. If a connection of the general power domain is tied to a component of the electrical domain, it automatically becomes an electrical connection. The other end of the connection can then only be tied to another component of the electrical domain. In this way it is prevented to connect components of different physical domains.
- If you want to use a connection of a domain that is not supported or has variables other than across and through, use the general power domain. The across and through variables can then be used as an alias of your own variables.

10.3.4 Connections

When two components in an Iconic Diagram are connected, one component will always compute the across variable, while the opposite component will always determine the through variable.

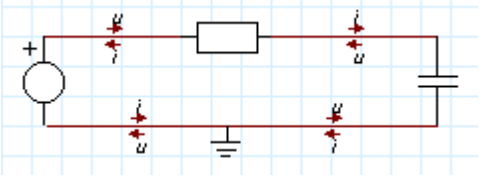
We can therefore also interpret a connection as a bilateral signal connection (across-signal and through-signal), of which the directions are opposite to each other.

With "direction" we mean the computational direction, just like signals in a block diagram. In a block diagram the across and through variables, which together form the flow of power, are not shown as a couple. This breaks up the symmetry between the physical system and the the structure of the model.

A connection as a bilateral signal flow.

The interpretation of a connection as a bilateral signal flow, does not fix the individual direction of the across variable and the through variable. It only means the direction of the across variable and through variable are opposite.

The specific computational direction of across and and through variables is called causality. In 20-sim an advanced algorithm detects the possible causalities of each submodel and tries to combine these in such a way, that optimal simulation code will be generated. The result can be inspected using the *Causality Info* command from the *View* menu. The result of this command is shown in the example below. The computational direction of forces and velocities is clearly shown for all the connections.

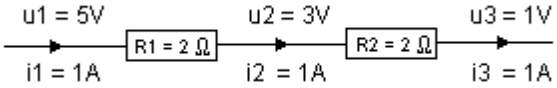


10.3.5 Orientation (Across)

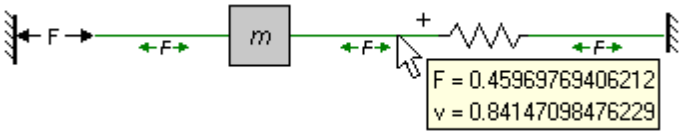
When simulating, you have to know how across and through variables are oriented to make a correct interpretation of what is happening. 20-sim can show the orientation when you select the *Orientation Info* option of the *View* menu.

Across variables in the Top Level

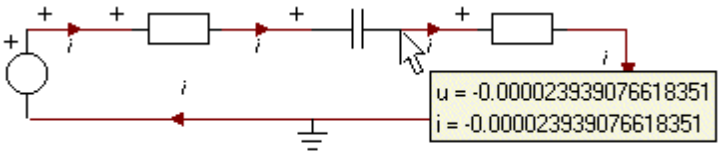
Across variables in the top level of a model are always defined with respect to a single global reference. In the picture below an electrical iconic diagram is shown with three voltages (u1 = 5 V, u2 = 3 V and u3 = 1 V). This means these voltages are all defined with respect to the same reference, e.g., the ground u = 0 V.



All variables values of a connection can be easily inspected during simulation by placing the mouse pointer on top of that connection as shown below. The connection between the mass and the spring (see picture below) shows an across value v = 0.84 m/s. This means the point where the mass and spring are connected has a global velocity of v =0.84 m/s.



In the figure below the connection between the capacitor and the resistor shows a negative voltage of u = -2e-5 V.

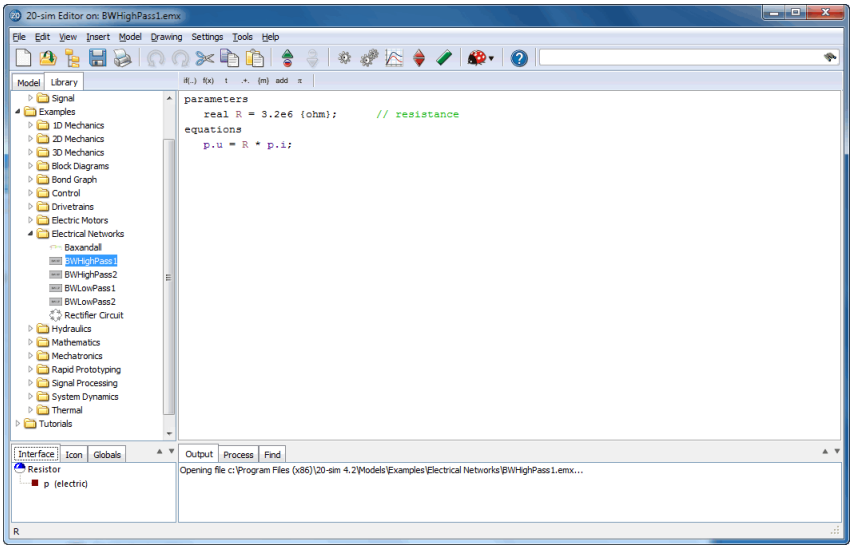


Across variables in Submodels

In submodels you have to inspect how across variables are defined. For example an ideal electrical resistor (see figure below) uses separate high and low terminals. The internal equations are:

```
p.u = p_high.u - p_low.u;
p_high.i = p_low.i = p.i;
p.u = R * p.i;
```

where p.high and p_low are the ports to the connections at both sides of the resistor. p_high.u and p_low.u are voltages defined with respect to the global reference, but p.u is the voltage difference between the high and low port and therefore not defined with respect to the global reference.



The definition of across variables depends on the definition of number of terminals of a port, which is shown in the Type Editor. To inspect the ports, put the mouse pointer on top of the Type section and select **edit type** from the right mouse menu.

Note

- The geometrical layout of the iconic diagram does not have a meaning. If a resistor is drawn upside down, its current is still defined with respect to the resistor (flowing towards or from). If a mass is drawn upside down, its force is still defined with respect to that mass (pulling or pushing).
- To show or hide the orientation of an Iconic Diagram select or deselect the *Orientation Info* option of the *View* menu.

10.3.6 Orientation (Through)

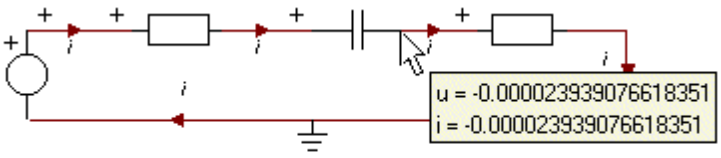
When simulating, you have to know how across and through variables are oriented to make a correct interpretation of what is happening. 20-sim can show the orientation when you select the *Orientation Info* option of the ***View*** menu.

Through variables in connections

Through variables in connections are always defined with respect to the components they are connected with. 20-sim automatically assigns an orientation for these through variables. This orientation can be made visible by selecting the *Orientation Info* command of the *View* Menu.

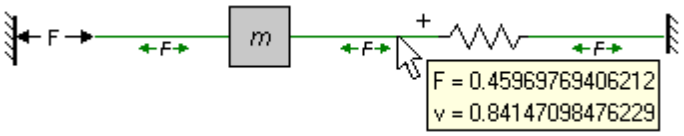
Non-mechanical

Through variables that are not of the mechanical domain are always shown with an arrow in the connection. The arrow indicates the direction of the positive flow. The figure below shows a negative current: i = **-**2e-5 A. This means the current flows in the opposite direction of the arrow.



Mechanical

Through variables of the mechanical domain are always shown with two arrows. The arrows indicate the direction of the positive force. The figure below shows a positive force: F = 0.45 N. This means the spring pushes with a force of F = 0.45 [N] upon the mass and the mass pushes with a force of F = 0.45 N upon the spring (note: a negative force would mean pulling instead of pushing).



This interpretation of forces in 20-sim is only valid when a positive velocity defined from the left to the right, and from the bottom to the top!

Through Variables in Submodels

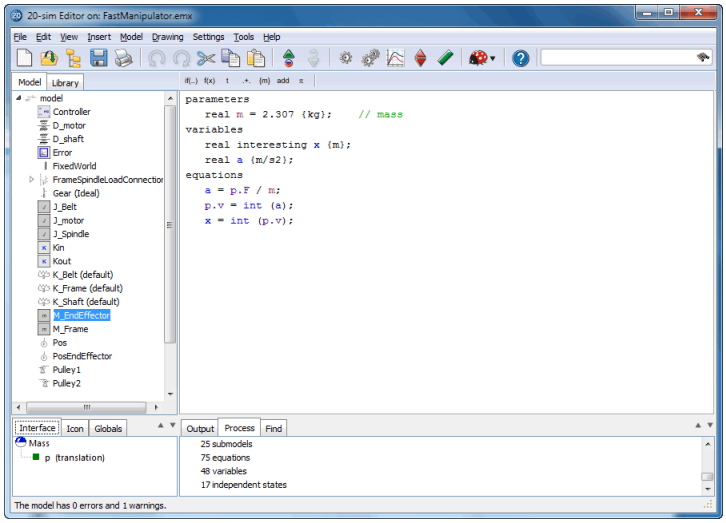
In submodels, you have to use the Type Editor to find out how through variables are defined. For example an ideal mass (see next figure) uses the option any number of terminals for its port (inspect the port in the Type Editor). The port equations are:

```
p.F = p1.F + p2.F
p.v = p1.v = p2.v;
```

The internal equations are:

```
p.v = (1/m)*int(p.F);
x = int(p.v);
```

where *p1* and *p2* are the ports to the connections at both sides of the mass. *p1.v* and *p2.v* are velocities defined with respect to the global reference. However, *p.F* (see also in the figure below) is the sum of the forces applied on both sides.



The definition of through variables depends on the definition of number of terminals of a port, which is shown in the Type Editor. To inspect the ports, put the mouse pointer on top of the Interface tab and select *Edit* from the right mouse menu.

Note

- The geometrical layout of the iconic diagram does not have a meaning. If a resistor is drawn upside down, its current is still defined with respect to the resistor (flowing towards or from). If a mass is drawn upside down, its force is still defined with respect to that mass (pulling or pushing).
- To show or hide the orientation of an Iconic Diagram select or deselect the *Orientation Info* option of the *View* menu.

10.3.7 Global Reference

20-sim iconic diagram models do not contain geometrical information. It is not important how a model is drawn, but how the elements are connected. To make a correct interpretation of the simulation results, the definition of references has to be known.

Across

Across variables in the top level of a model are always defined with respect to a single global reference. For all domains this global reference is equal to 0. The interpretation of this reference is up to the user. For example a zero pressure could be the absolute vacuum or the air pressure at ground level. For most physical domains, unless specified otherwise, a standard interpretation is used. This is shown in the table below.

Domain	Across (a)	Global Reference
power	across *a*	zero

mechanical (translation)	velocity *v* [m/s]	zero velocity
mechanical (rotation)	angular velocity *omega* [rad/s]	zero angular velocity
pneumatic	pressure *p* [Pa]	air pressure at ground level
thermal	temperature *T* [K]	absolute minimum temperature
electric	voltage *u* [V]	*zero voltage*
hydraulic	pressure *p* [Pa]	air pressure at ground level
magnetic	magnetomotoric force *i* [A]	zero
pseudothermal	temperature *T* [K]	absolute minimum temperature

Through

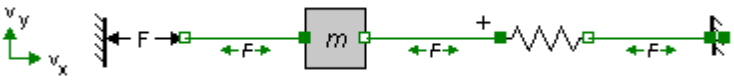
Through variables in connections are always defined with respect to the components they are connected with.

Non-Mechanical

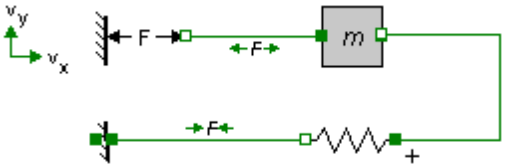
Through variables that are not of the mechanical domain are always shown with an arrow indicating the direction of the positive flow.

Mechanical

Through variables of the mechanical domain are always shown with two arrows indicating the direction of the positive force. Since mechanical variables are often geometrically interpreted, these arrows assume a positive velocity from the left to the right and from the bottom to the top.



See for example the system shown above. Look at the connection between the spring and the fixed world. Most users will correctly interpret a positive force as one that "makes the mass slow down". The same model is shown below. To prevent an incorrect interpretation such as a positive force "makes the mass accelerate", the arrows are pointing inwards.



This interpretation is only valid when you assume that a positive velocity is from the left to the right and from the bottom to the top!

10.3.8 Causality

The connection between two components describes the flow of power from one to the other. This power flow is always described by across and through variables. For example an electrical connection can be described by the variables voltage *u* and current *i* (u*i = power). Each Iconic Diagram component creates a causal relation between these variables. E.g. an electric resistor can be described by the equation:

*u = i*R;*

where *R* is the resistance. Here the voltage is a function of the current. In 20-sim we say the component has a voltage out causality. This causality is not fixed. We can easily invert the equation:

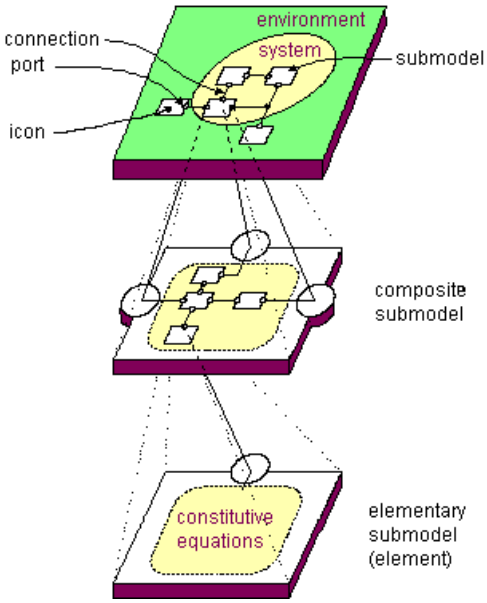
i = u/R;

Now the component has a current out causality. Which of the two equations will be used for the simulation depends on the other components.

In 20-sim an advanced algorithm detects the possible causalities of each model and tries to combine these in such a way, that optimal simulation code will be generated. You can help this algorithm by manually setting the causality of ports.

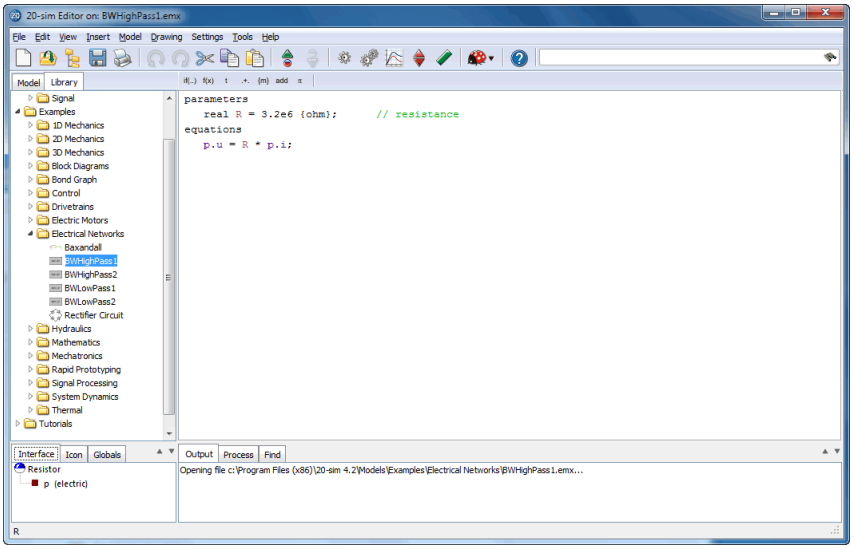
10.3.9 Ports

A port is a location where a component can exchange information (in case of a signal port) or power (in case of a power port) with its environment. So, it is the port that defines the connection with a component. A port is an important concept, as it allows you to describe the properties of the connection that can be made to the component, i.e., its direction, size, domain, etc. Ports can be defined in 20-sim using the Type Editor.



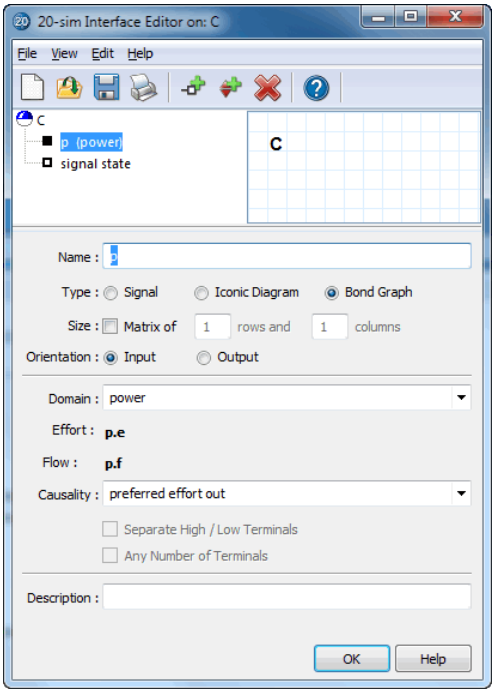
Port Variables

All ports of a submodel are shown in the Type window of the 20-sim Editor. In the figure below an electrical Resistor is shown with one power port *p*. Each port has an across and through variable. In 20-sim these variables are denoted with the extensions *.a* and *.t.* You can also use the names that are specific for a certain domain. For the electrical domain the extensions *.u* and *.i* are used. You can see an example in the figure below (Implementation section) where equations are defined using the variables *p.u* and *p.i.*



Port Properties

Ports can be added and defined in the 20-sim Type Editor. You can open the *Interface Editor* in the Interface tab (right mouse menu). The *Interface Editor* of the Resistor is shown below.



Port Properties

Iconic Diagram Ports have several properties:

- *Name*: The name of the port.
- *Type*: Next to Iconic diagram ports, 20-sim also knows bond graph ports and signal ports.
- *Size*: The standard size of a port and corresponding connection is 1 but you can also define ports with larger sizes.
- *Orientation*: The orientation of a port defines how the though variable is connected.
  - *Input*: A positive through variable will act upon a component in the positive direction (mechanical domain) or flows into the component (non-mechanical domain).
  - *Output*: A positive through variable will act upon a component in the negative direction (mechanical domain) or flows from the component (non-mechanical domain).

Note: The port orientation and the orientation of the corresponding connection can be made visible in 20-sim by arrows.

- *Domain*: The physical domain of the port.
- *Causality*: The causality of the port variable (across and through). You have to define here what should be the input variable (across or through) and what should be the output variable (across or through).

By default, an iconic diagram port is a port where power can be exchanged between a component and its environment in terms of an across variable and a through variable. Such a port is represented by one terminal. However, there are two special cases where it is desirable to define an iconic diagram port that has more than one terminal. These are the Separate High / Low Terminals port and the Any Number of Terminals port. These options are only for advanced users!

- *Separate High / Low Terminals*: Select the port to have two terminals. Only for experienced users!
- *Any Number of Terminals*: Select the port to have any number of terminals. Only for experienced users!

10.3.1 Ports with more than one Terminal 0

By default, an iconic diagram port is a port where power can be exchanged between a component and its environment in terms of an across variable and a through variable. Such a port is represented by one terminal (connection point). However, there are two special cases where it is desirable to define an iconic diagram port that has more than one terminal.

Separate High / Low Terminals

Consider, as an example, an mechanical spring. The power that flows into such a component is uniquely determined by the velocity difference between the two terminals of the component and the common force that acts upon both ends. Because of this one could say that there is one port, whose power is determined by one across value (the velocity difference) and one through value (the common force), but is represented by two terminals. To support this, 20-sim allows you to define a special type of iconic diagram port by indicating that is has Separate High / Low Terminals. The two terminals of the connection are named high and low. If the port is named p, the formal equations are:

fixed in orientation

p.t = p1.t = p2.t
p.a = p_high.a - p_low.a

fixed out orientation:

p.t = p1.t = p2.t
p.a = - p_high.a + p_low.a

In 20-sim these equations are automatically derived.

Any Number of Terminals

Consider, as an example, a mass. A characteristic of this component is that you can connect many springs and dampers to it. Implicitly, one expects that the net force (i.e., the summation of the forces applied by the connected components) will be applied to the mass, and that it will have a single velocity. To support this, 20-sim allows you to define a special type of iconic diagram port by indicating the kind to be Any Number of Terminals. The terminals of the connection are named 1, 2, 3 etc.. If the port is named p, the formal equations are:

p.a = p1.a = p2.a = p3.a = ...
*p.t = sign(p1)*p1.t + sign(p2)*p2.t + sign(p3)*p3.t + ...*
sign = 1 when p1 has a fixed in orientation etc.
sign = -1 when p2 has a fixed out orientation etc.

In 20-sim these equations are automatically derived.

10.3.1 Creating Iconic Diagrams 1

When you understand the concepts of across and through, including orientation, constructing an iconic diagram is easy:

1. Create an ideal physical model.
2. Select the components that represent the various parts of the ideal physical model and drag and drop them to the 20-sim editor. Iconic Diagram components can be found in the model library. Some standard components are shown in the tables of this tutorial.
3. Connect the components according to the ideal physical model.
4. Compile the model and run a simulation. For a good interpretation of the plots, you must know that across variables are always defined with respect to a global reference and through variables are always defined with respect to the components.

10.3.1 Creating your own Components 2

In 20-sim you can easily create your own components. The process of creation consists of three parts:

1. For any connection an internal port has to be defined. For each port you have to specify some data:
   - The physical domain.
   - The size.
   - The orientation.
   - The causality.
2. Create the icon for the component. This can be done with a specialized drawing editor.
3. Create the component description. This can be done by (differential) equations or by an iconic diagram.

Index

- - -

- 197, 212

- * -

* 147, 196

- . -

./ 198
.^ 199
.cse 420
.csv 420
.emxz 23

- / -

/ 199

- [-

[229

-] -

] 229

- ^ -

^ 201

- + -

+ 147, 197, 212

- < -

< 209
<= 209
<> 210

- = -

== 210

- > -

> 211
>= 211

- 0 -

0 536, 1146
0 junction 1147
0-junction 536, 537

- 1 -

1 525, 526, 1146, 1148
1-3-5-7-9 Polynomial 390, 402
1D 774
1-junction 525, 526

- 2 -

20sim 2, 6, 7, 9, 10, 15, 198
20-sim 2, 6, 7, 9, 10, 13, 14, 15, 198
20-sim Filter Editor 363
20-sim library 12
20-sim Model 255
20-sim Scenery 255
255 296
2D 774
2-D 773
2D Library 773
2D motion 773
2D Springs 793
2D-Animation 292
2D-modelling 809
2D-point-X 813
2DSmallRotation library 773
2dtable 923

- 3 -

3 degrees of freedom 809
32-bit machine code 45
3-4-5 Polynomial 390, 402
3D Animation 83
3D Animation Properties 274
3D Animation Window 273
3D Body 807
3D Library 807
3D mechanics 2
3D Mechanics Editor 239
3D Points 807
3D Representation 247
3DAnimationBeam 289
3D-body 813
3D-mass 807

- A -

A 592
ABCD matrix 355

abort 221
Abs 128
Absolute 104, 105, 196, 926
absolute tolerance 361
absolute value 128
AC motor 690, 696
AC synchronous motor 436
acceleration 398, 790, 856
AccelerationActuator 689, 820
AccelerationActuator-Relative 689, 820
AccelerationSensor-Absolute 761, 867
AccelerationSensor-Relative 762, 869
accumulator 683
acmotor 696
ACMotor-TorqueLoop 690
Across 67, 1154
Activation 7
activation function 325
Activation Function Type 327
Active 89
activity 537
Actuated Joints 261
actuator 705, 706, 707, 708, 709, 810, 811, 823, 824, 829, 830, 831
Actuator Model 959
Actuator Properties dialog 270
Actuators 793
AD 1016
Adams-Bashford 104
Add 920
Add New 34
Add to Favorites 85
Add to Input Probes 85
Add to Output Probes 85
Add/Delete 89, 466
Addition 197
Additional Outputs 255
adjoint 168, 169
Advanced Scripts 512
After the run 455
algebraic 85, 128
algebraic loop in 45
algebraic loop out 45
Algebraic loops 50, 103, 232
algebraic solver 103
Algebraic variables solved 27
Alias 71
alias variable 45
Alias Variables 85
All Runs 83
Allow model updates 27
Allow Symbol Prefixes 71
Always Look up Vector 292
Always perform brute force simulation 27
Ambient color 295
Ambient Light 295
amplifier 575
Amplitude 319, 377
amplitudesensor 1063
Analog 1019
analog filter 363
Analog to Digital Convertor 1016
Analyze Causality 32
And 207, 1048, 1056
angle 728
angular acceleration 728
angular velocity (constant) 707, 708
angular velocity (variable) 708, 709
angular velocity = 0 723, 838
Animation Properties 274
ANSI-C Code 455
ANSI-C Function 455
antisym 170
Anti-Windup 959
Any Number of Terminals 65, 1166
Apply Regularization 333
arccos 185
arccosh 186
Archive 22
arcsin 185

arcsinh 187
arctan 188
arctanh 188
Array 230
Array Division 198
array multiplication 198
Array Power 199
Artificial intelligence 323, 328
As Constraint Joint 261
atan2 189
Attempting Real-Time simulation 97
Attenuate 901
Attenuation 302
Attributes 274
Author 27
Auto Causality 27
Auto Indent 13
Auto Recovery Filename 19, 30
Average 1065, 1090
Averaging Integration 914
Avi 279

- B -

B 296
back plane 292
Background Color 281
Background Image 281
Backgrounds 44
Backlash 710, 831, 934
BackPropagation Networks 324
Backward Difference transformation 363
Backward Differentiation Formula 105
Backward Euler 103
Band 348
Band Pass 348
Band Stop 348
Bandwith 1034, 1036
Base 10 Logarithm 136
basic libraries 592
Basic Script 511
basis function 328
BDF 105
bearing 711, 723
BeltPulley 736, 874
Bessel 348
bilateral signal 1156
bilinear transformation 363
Bipolar Sigmoid 327
Bipolar Sigmoid Activation Function Scale 327
bitand 201
bitclear 202
bitcmp 203
bitget 203
bitmap 299
bitset 205
bitshift 206
bitxor 206
Block Diagram 1137
Block Diagrams 1016
blue 296
Bode Plots 319, 377
body points 807
Body Properties Dialog 257
body without rotational inertia 812
Bond Graph 17, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 539, 540, 541, 542, 544, 545, 546, 547, 548, 549, 550, 551, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 1132, 1137, 1139
Bond Graph Literature 1144
Bond Graph Models 1128
Bond Graph Models,Simplification 1128
Bond Graph,Equations 1139
Bond Graph,Iconic Diagram 1132
Bond Graphs 15, 1123, 1131, 1132, 1134
Bond Graphs,Iconic Diagrams 1132, 1134
Bonds 1125, 1126
Boolean 122, 124, 125
Boolean And 207
Boolean Not 212
Boolean Or 208

Break 711
Breakpoint Editor 89
Breakpoints 89
Bridge 579
Broydon Fletcher Goldfarb Shanno 476
Brush DC Motors 431
Brushless DC Motors 434
Brute Force 83
Bryant angles 257, 296
B-spline 328
B-spline Network 328
bsplinenetwork 952
Buffers 1149
Built-in Compiler 77
Bulk modulus 592
Butterworth 348, 1034, 1036
BW 1034, 1036
by 218

- C -

c 455, 515, 516
C-2 551
C-3 539
cam 736, 875
Cam Motion Profiles 390
Cam Wizard 385, 390
Camera 292
Camera Settings 251
CamProfile.dll 736, 875
cams 385, 736, 875
Capacitor 567
Capacitor.emx 567
carriage 793
Case 220
Catch up with lost time 97
Causal analysis 32
causal conflict 32
Causal Form 46, 233
causal order 46, 233
Causal Strokes 32, 1156
Causality 46, 64, 226, 227, 233, 1129, 1156, 1162
CCCS 584
C-code Folders 19, 30
C-Code Generation 19, 30, 455
Cd 592
ceil 129
C-elements 1149
center of stiffness 796
CentrifugalPump 615
changing mass 856
Chebychev 348
check 254
Check Complete Model 26
Check Energetic Behavior 32
Check for Model Updates 43
Check Model 255
checkvalve 635
checkvalve-states 659
Choosing a Motor 451
Chrome 299
Circle 292
circle elements 283, 286, 287, 302
Clear 83
Clear After Every Run 466
Clock 1057
Clock Signal 1049
Clock-Continuous 1049
clock-discrete 1017
clockinterrupt 100
closed chain mechanism 252
Closed Forms 302
Clutch 717
cmabender 564, 821
CMAStretcher 565, 822
code 13, 109
code block 110
Code Blocks 51, 236
code generation 2
Cogging 445

collect 180
Collision 835
Collision-Relative 833
colon 229
Color 70, 296
Color Syntax Highlighting 13
Column 37
Columns 171, 229, 1162
Combining of divisions 26
Combining of multiplications 26
Combining splitters 26
Comments 115
Commercial Controllers 964
Company 27
Comparison of Friction Models 1120
compensator 308, 347
Compiling 45
Compiling Models 45
complete set of equations 54, 238
Component Properties 239
Components 1167
computational direction 1156
Cone 287
Confirmation on Delete 19, 30
connecting of joints 254
Connecting Sensors 268
Connecting Submodels 24
connection 597, 729, 859
connection mode 24
Connections 1156
Constant 295, 1068
constant current 589
Constant Values 116
constant voltage 591
Constants 14, 76, 116, 118
constraint 64, 129
constraint equal 64
Constraint Joint 252
constraint not equal 64
Constraint Settings 261
Constraint variables solved 27
constraints on motion 780
Continue 89
Continuous Descent 476
continuous pulse 390, 402
Continuous-Discrete Tranformations 363
Continuous-Time 35
Continuous-Time and Discrete-Time Models 35
contra-rotating shafts 741
ControlledLinearSystem 946
ControlledSystem 947
Controller 343, 344, 345, 346, 347, 986, 987, 988, 1009, 1010, 1011, 1012
Controller Design Editor 308
controller output 954
controller variable 954
Controller-P 969, 993
Controller-PD_p 970, 993
Controller-PD_s 971, 994
Controller-PI 971, 995
Controller-PI_sp 972, 996
Controller-PI_sp_aw 973, 996
Controller-PI_sp_aw_u0 974, 997
controller-pi_sp_aw_u0_tr 975, 998
Controller-PID_p 976, 999
Controller-PID_p_sp 977, 1000
Controller-PID_p_sp_aw 978, 1001
Controller-PID_p_sp_aw_u0 979, 1002
controller-pid_p_sp_aw_u0_tr 980, 1003
Controller-PID_s 981, 1005
Controller-PID_s_sp 982, 1005
Controller-PID_s_sp_aw 983, 1006
Controller-PID_s_sp_aw_u0 984, 1007
controller-pid_s_sp_aw_u0_tr 985, 1008
controllerwizard 969
convection 889
Convert 1132
Copy 85
Copy from States 466
Copy Specified 85

Copy States 83, 92
correct order of execution 51, 236
cos 190
cosh 190
Cosine 926
Cosine Wave 1085
cost function 504
CostFunction 1013, 1014
CounterbalanceValve 661
counter-moving ports 879
counting 229
coupling 755
CPS 383
crankrod 738, 877
Create Datafile 466
Created 27
Creating 1131, 1167
Creating Components 1167
Creating Elements 1144
Creating,Bond Graphs 1131
Creating,Iconic Diagrams 1167
cross 171
CrossingBoth 1028
CrossingDown 1028
CrossingUp 1028
crossover yerk 390, 402
Cube 284
Cubic 390, 402
cumulative power spectral density 383
current 589
current controlled current source 584
Current time 355
CurrentSensor.emx 568
CurrentSource 589
CurrentSource.emx 589
CurrentSource-CCCS 584
CurrentSource-CCCS.emx 584
curve fitting 482
custom libraries 17
CV 954
cyclic 390, 402
Cyclic Motions 390, 402
Cycloid 1073
Cycloidal 390, 402
Cylinder 286, 299, 598
CylinderChamberA 601
cylinderchamberb 602
cylinderdouble 603
cylindersingle 607
CylinderSingleSpringReturn 605
CylinderSpringReturn 599

- D -

D Controller 344
DA 1019
da-delay 1018
Damper 722, 796, 837, 862
data 148
Data Files 413
Data Input Wizard 1069
Data Types 124
Datafile 931, 1074
DataFromFile 1069
Davidson Fletcher Powel 476
DB 927
DC Motors 452
DCmotor 566, 697
DC-motor 566, 697
DCmotor.emx 566, 697
Ddt 129
deactivate license 9
deactivation 9
DeadZone 922
deal voltage source 590
Debugger 88
declarations 14
Default Lights and Cameras 292
Default Line Thickness 19, 30
Default Shared Y-Axis 19, 30
default value 76
default values 119

degree of freedom 809
degrees of freedom 773, 776
deinstallation 9
Delay 902, 1020
delay-n 1019
Delay-Pade 902
Delay-Step 902
Delay-Time 903
delay-variabletime 903
demonstration 6
Demux 904
denominator 312, 368
Department 27
Dependent 85
dependent rate 45, 48, 235
dependent state 45, 48, 235
Dependent states are transformed 27
Dependent/Algebraic 85
Derivative 130
Derivative action 957
Derivative Gain Limitation 907, 957
Derivative time 957
description 27, 70, 76
Desired Area of Operation 451
det 171
Determinant 171
diag 172
difference equations 310, 366
Differential 741, 1021
differential equations 310, 366
differential form 48, 235
DifferentialGear 739
Differentiate 906
Differentiate- 905
Differentiate-Calculus 905
Differentiate-Default 908
Differentiate-FO 906
Differentiate-SVF 907
Differentiation 907, 908
Differentiator 344
Diffuse color 295
Digital 1019
digital filter 363
Digital linear controllers 342
Digital linear filters 342
Digital to Analog Convertor 1019
Diode 569, 579
Diode Bridge 579
Diode-Ideal 569
direct 181
Direct acting 955
direct search 476
Direction Up Vector 257
Directional Light 295
discharge coefficient 592
Discrete 363
Discrete Differential 1021
Discrete Integral 1022
Discrete Sample time 310, 366
Discrete System 99, 140
Discrete Time Interval 99
DiscreteDifferential 1021
DiscreteIntegral 1022
Discrete-Time 35
Discrete-Time Models 35
discrete-time system 363
disk space 5
DisplacementMotor 609
displacementmotor-leakage 618
displacementpump 610
displacementpump-leakage 620
dissipative element 522, 530
Distribute curves at end of simulation 27
disturbances 308
Div 199, 200
Divide 918
Division 199
DLL 149, 151
dlldynamic 151
dly 130

Do 218, 219
Documentation 514
Documentation Editor 41
DoFromMatlab 1040
Domain 67
domain changes 533
Domain name 70
Domains 66, 67, 70
Domains,Quantities and Units Editor 66
DoMatlab 223, 1042
DoMatlab-Final 1041
DoMatlab-Initial 1041
double acting cylinder 603
Double Click Function 19, 30
DoubleSwitch-Level 570
drag and drop 17, 254
Drawing Rules 776
DTypeFlipFlop 1057
DTypeFlipFlop-Continuous 1050
DTypeFlipFlop-Discrete 1057
Duplication 304
During the run 455
Duty Cycle 445
Dynamic Backgounds 44
Dynamic DLL's 159
Dynamic Error Budgeting 383
Dynamic Model 419
Dynamic Systems 1122, 1153
dynamic variables 85

- E -

Eddy Current 445
Edit Condition 89
Edit Implementation 34
Edit Modes 248
Edit window 244
Editing Domains 70
editor 2, 12, 13
Effort 67, 1123
effort in 226
effort source 523, 531
Effortincausality 226
EffortSensor 517, 1126
Eigen Frequencies 317, 375
Electric 67, 564
Electric Switch 570, 580
electrical inductance 574
Elements 229, 1144
Eliminating double differences 26
Eliminating junctions 26
Eliminating nodes 26
Else 215, 217
Enable XMLRPC Interface 19, 30
Encoder 763
encrypted 22
end 214, 215, 217, 218, 219
End Angle 385
end plane 281
end position 283
End Time 398
end_time 390, 402
Endless 77, 97
Energy 32, 518
EnergySensor.emx 518
Equal 64, 146, 209, 210, 211
equation 13
equation editor 12, 13, 14
equation model 13
Equations 54, 109, 110, 238, 1139
Equations interpreted as code 27
Equations,Bond Graph 1139
Error 954
error message 26
Errors 26, 88
Euler 103
Euler Parameters 257, 296
event 143, 144, 145, 1028
Event delta 77
eventdown 144
Events 115
eventup 144

Every 19, 30
Examples 17
Execution 51, 236
Exp 131
exp10 131
exp2 132
Expand Vectors/Matrices 85
Expand Vectors/Matrices: 85
Explode 25
Exponential 131, 132
Exponential Window 910
Export 41
Export to Matlab 10, 39
Export to Simulink 39
Exporting Simulations 94
External DLL 503
External Tracking 959
eye 172

- F -

F3 17
fading colors 84
false 122
Fast Fourier Transform 352
Faulhaber 2006 427
favorites 85
femto 71
FFT Analysis 352
FFT plot 352
FFT settings 352
FFT Window 355
File Data 148, 166
File Input 931, 1074
FileInput 148
Filename 1069
Files of Type 22
Filter 342, 348, 352, 908, 957, 1033, 1034, 1036
Filter Editor 342, 908
Filtering 363
Final Equations 110
finalequations 109
Find 84
Find again 17
Find box 17
Find tab 12, 17
Finish 77
first 182
First Order Filter 349, 351
Fixation of bodies 780
Fixed Causality 46, 233
fixed in orientation 1162
fixed out 64
fixed out orientation 1162
fixed position 56
fixed sample time 91
Fixed Sampletime 100
fixed time step 91
fixed world 723, 787, 810, 838
FixedWorld 723, 838
Flash 279
Flat 298, 299
Flexible Components 793
FlipFlop 1050, 1054, 1055, 1057, 1061, 1062
Flipping 807
Floating License 6, 7, 9
floating mechanism 257
floor 132
Flow 67, 1123
flow conductance 592
flow in 227
flow source 524, 532
flowcontrolvalve 636
flowcontrolvalve-states 662
Flowincausality 227
FlowSensor 518, 632, 1126
FlowSource 611
flowsource-leakage 622
Fonts 19, 30
For 218
For To Do 218
Force 810, 811, 823

force (constant) 823
force (variable) 824
ForceActuator 824
ForceActuator-Relative 824
Force-Relative 823
forces 792
ForceSensor-Relative 870
Fork 878
Form 46, 233
Forward Euler transformation 363
Fourier Analysis 352
fourthreewaydirectionalvalve 637
fourthreewaydirectionalvalve-states 664
FourThreeWayProportionalValve 639
FourThreeWayProportionalValve-States 666
frame 296
frame hierarchy 282
frame rate 279
Frames 292
free 6
freeze 787, 810
frequency 99, 352, 382
frequency range 352
Frequency Response 359
Frequency Sweep 1081
frequencyevent 145, 1031
Friction 522, 530, 723, 838
friction literature 1121
Friction Phenomena 1113
FrictionRelative 843
FrictionSimple 848
FrictionSimple-Relative 852
From Matlab 76
FromMatlab 224, 1042
front plane 292
Full Screen 96
Functions 14, 127

- G -

G 296, 592
Gain 343, 370, 909, 955
Gain Margin 319, 377
Gain-Phase 382
Gauss 183, 1075, 1091
Gaussian Noise 183, 1075, 1077, 1091
gdi+ 2
Gear 741
gear box 741
gear ratio 741, 752
gearbox 752
General Modified Sine with Constant Velocity 390, 402
General Properties 281, 298
Generate 20-sim Model 255
Generic Filter 348
Geneva Mechanism 390, 402
Ghost Modes 248
giga 71
Global 57, 59, 123
Global Maximum 84
Global Minimum 84
Global parameters 57, 76, 123
Global Reference 1160
Global Relations Editor 59
global variables 57, 85, 123
globals 57
Globals tab 59
Go Down 20
Go Up 20
Gouroud 298
gradient search 476
Graph Animation 305
graphical editor 12, 15
gravity 856
green 296
green input and output lines 35
Ground 573, 890
Ground.emx 573
GY 519
GY-2 553, 555
GY-3 540

GY-element 1151
Gyrator 519, 522, 532
Gyrators 1151
- H -
halt 221
Has Damping 261
Has Minimum 261
Has Spring 261
Head Sensor 633
heatcapacity 891
heatflow 896
heatflowsensor 900
Help 1
Help Page 27
Hidden 85, 116, 119, 120, 124
Hidden layer 325
Hidden layers 324
hidden variable 45
Hierarchical Models 20
hierarchy 20, 282
high 65
High / Low Terminals 65, 1166
High frequency noise 957
High pass 348, 349
High Pass First Order Filter 349
High Pass Second Order Filter 349
high-frequency measurement noise 957
High-frequency roll-off filter 347
Hinge X-rotation 261
Hinge Y-rotation 261
Hinge Z-rotation 261
Histogram 492
H-matrix (4x4) 268
Hold 140, 1023
homogeneous 172
How to use the Parameters 417
html document 41
HTTP 19, 30
Hydraulic 67
hydraulic accumulator 683
Hydraulic Powersensor 633
HydraulicInertia 596
Hysteresis 445, 935, 942
Hz 99, 1034, 1036
- I -
I 520
I Controller 344
I-2 554
I-3 541
IC-2 521
Icon 20
Icon Editor 12, 55
Icon Size 19, 30
Icon tab 12
Iconic Diagram 1132
iconic diagram models 15
Iconic Diagram Ports 1162
Iconic Diagram,Bond Graph 1132
Iconic Diagrams 17, 564, 1132, 1134, 1153
Iconic Diagrams,Bond Graphs 1132, 1134
ideal capacitor 567
ideal current source 589, 590
ideal cylinder 598
ideal DC-motor 566, 697
ideal electrical inductance 574
ideal electrical resistor 579
ideal mass 856
ideal physical model 1122, 1153
ideal voltage source 591
Identified with 99
I-elements 1149
If 214, 215, 217
If Then 214
If Then Else 215, 217
Image 281
images 15
imaginary 382
impacts 860
Implementation 20
Implementations 34

Implode 25
Import 76, 89
Import Data 382
Import Gain-Phase 382
Import Real-Imag 382
Importing Simulations 95
included toolboxes 6
indentation 282
independent rate 45
independent state 45
indexer 736, 875
indexers 736, 875
indifferent 64
Indifferent Causality 46, 233
Inductor 574
Inductor.emx 574
Inertia 728
Initial 104, 105
Initial Equations 110
initial output 964
Initial Values 76
Initial Weights Fill Scale 327
initial yerk 390, 402
initialequations 109
Initialize variables at start of simulation 27
Initialize Variables on Nan 88
inner 173
Input Equations 45
Input from File 166, 1074
Input is not used 27
input noise 383
input probes 85
input signal 63
insert a submodel 24
Inspect Models 20
installation 7, 507
Installing 20-sim 7
instructions 45
int 133
Integer 124, 126
Integral 133, 1022
Integral Form 48, 235
integral time 956
Integrate 911, 915
Integrate Data 352
Integrate-ExpWindow 910
Integrate-FO 911
integrate-folimited 912
Integrate-Limited 913
Integrate-RectWindow 914
Integrate-Reset 914
Integration Error 104, 105
Integration Methods 77, 97
Integrator 344
interacting form 957
Interesting 85, 120, 124
interesting variable 45
interface 13, 62
Interface Editor 12, 62
Interface tab 12
intermediate points 24
Intermittent 390, 402
Interpreter Code 45
Interval 914
Introduction 410, 505
introduction to friction 1112
inverse 173, 916
inverseH 174
inverting equations 46, 233
Invertor 1050, 1058
IPM 1122, 1153
ISA form 957
iterate 115

- J -

Jacobian Matrix 268
jerk 385
Join Parameter Variation 466
Joystick 1070
Jump 17, 936
Jump and Rate Limiter 936

Junction 1148
Junctions 1146
- K -
Keyboard 1071
keyboard keys 277
keys 277
Keywords 27, 116
kilo 71
kind 85
- L -
Lag Filter 350
laminar flow 592, 628, 630
LaminarResistance 628
language 13
Language Reference 107, 109, 110, 116, 124, 127, 195, 214
Laplace variable 312, 368
Larger Than 211
Last Run 83
layers 324
Lead Filter 350
Learn after Leaving Spline 328, 333
Learn at each Sample 328, 333
learning rate 327, 328, 333
left-hand frames 292
left-handed 298
Less Than 209
Lever 879, 887
lever ratio 879
Libraries 19, 30
library 2, 12, 17, 257
Library File 27
Library Folders 19, 30
library names 19, 30
Library Path 27
library paths 19, 30
Library tab 12, 246
License 7, 9
License Activation 7
Likes 64
Likes Causality 46, 233
limint 134
Limit 134, 937
limitations 6
Limited Integration 134, 913
limited stiffness 807
Limits 442
Line 283
Line Climber 476
Linear 295
Linear Actuators 793
linear differential equations 310, 366
linear feedback anti windup 959
Linear Motors 438
Linear Slides 793
Linear System 916, 1100
Linear System Editor 363, 916, 1100
linear time- invariant models 363
Linearization Explained 358
Linearization moment 27
Linearization Tolerances 361
Linearization Type 27
Linearize at 355
Linearize Model 355
linearized symbolically 773
LinearSystem 1023
linsolve 174
Literature 965
Load Scene 279
Load Weights at Start of Simulation 327, 333
Localhost 19, 30
Locate License File 9
locks 277
Log 135, 927
log10 136
log2 136
Logarithm 136
Logical Nand 1052
Logical-And 1048, 1056
Logical-Invertor 1050

Logical-Nor 1053
Logical-Or 1053
Logical-Xor 1051
Loop 3D Animation on replay 27
Loop Flushing Valve 642
Loops 50, 232
low 65
Low Pass 348, 351, 1034, 1036
Low Pass First Order Filter 351
Low Pass Second Order Filter 351
Low Terminals 65, 1166
LowPassFilter-BW2Hz 1034
lowpassfilter-bw4 1036
LowPassFilter-BW4Hz.emx 1036
lowpassfilter-fo 1037
lowpassfilter-so 1039
Lumped Parameter Method 1122, 1153

- M -

Machine Code 45
Magnetic 67
Main Model 20, 27
Main Model / Submodel 27
main reference frame 282
major 122
major step 52
Manager 27
Manipulate 277
Manual output 964
Mark 84
mask 363
Mass 856
Material 302
matlab 10, 39, 85, 222, 223, 224, 355, 455, 466, 505, 509
matlab simulink connection 10
Matlab-Code folders 19, 30
Matlab-Code Generation 19, 30
Matrices 37, 85, 228, 229
Matrix 198, 229, 230
Matrix Declaration 228
Matrix is assigned a scalar 27
Matrix Notation 229
Matrix Operators 230
Matrix Use 230
max 175
maximum 104, 105, 944, 1065
Maximum allowed lost time 97
maximum continuous torque 445
Maximum Efficiency 450
Maximum number of variables to show in varlist (0=all) 19, 30
maximum phase current 434
maximum phase to phase voltage 434
Maximum Power 450
Maximum Step Size 105
Maximum Value 85
Maxon 421
Maxon motors 421
Maxwell reciprocity 521
Mean 1065, 1090
mean time interval 1095
measured variable 954
measurement 308, 954
Mechanical 67, 688
mechanism 736, 875
mechanisms 385, 736, 875
mega 71
Melt equal junctions 26
Mesh 298
Message Log 73
Message window 248
M-file 39
MGY 522, 1152
MGY-2 555
MGY-3 542
MGY-element 1152
micro 71
Migrating from Older Versions 241
milli 71
min 176

Minimize/Maximize 470
Minimum 945, 1066
Minimum / Maximum 466, 470
Minimum Value 85
minor steps 52
Minus 920
Mixed models 35
MLP Network 325
mlpnetwork 953
Mod 200
Modal 858
ModalSummer 859
Model contains algebraic loops 27
Model contains algebraic variables 27
Model contains constraint variables 27
Model contains dependent states 27
Model Help 27
Model Hierarchy 12, 76
Model Hierarchy: 85
Model Layout 108
model libraries 17
model library 15
Model Settings 27, 250
Model tab 12, 245
Model Template Folders 19, 30
modeling 515
models 20, 515
Modified Sine 390, 402
Modified Sine with Constant Velocity 390, 402
modified sine with contstant velocity 390, 402
Modified Trapezoidal 390, 402
Modulated dissipative element 522
Modulated effort source 523
Modulated Elements 1126
Modulated flow source 524
Modulated gyrator 522
Modulated Gyrators 1152
Modulated Sources 1151
Modulated transformer 524
Modulated Transformers 1152
ModulatedCurrentSource 590
ModulatedCurrentSource.emx 590
modulatedflowsource 611
modulatedflowsource-leakage 623
modulatedheatflow 896
modulatedpressuresource 612
ModulatedTemperatureSource 897
ModulatedVoltageSource 590
ModulatedVoltageSource.emx 590
Modulus Operator 200
Momentum Constant 327
Monitor 921, 1085
Monte Carlo analysis 83, 492
More 279
motion 254, 385, 402
motion profile 385
Motion profile parameters 402
Motion Profile Wizard 398, 401
MotionProfile-Wizard.emx 1072
Moving 277
Moving Objects 277
MovingAverage 1066
MR 522
MR-2 556
MR-3 544
MSC% 390, 402
MSC50 390, 402
MSe 523
MSe-2 557
MSe-3 545
MSe-element 1151
MSf 524
MSf-2 558
MSf-3 545
MSf-element 1151
msum 176
MTF 524, 1152
MTF-2 558
MTF-3 546
MTF-element 1152

mul 147
Multi Layer Perceptron 325
multi-bond 37
multi-connection 37
Multi-Dimensional Models 37
Multi-Dimensional Ports 37
Multi-Line Tabbing 13
multiple port restrictions 64
Multiple Run 83
Multiple Run results 83, 466
Multiple Run Wizard 465
multiple-run 83
multiplication 69, 196, 198
multiplication factor 71
Multiplications 71
Multiply 901, 918
MultiplyDivide 147, 918
multi-signal 37
Mux 919

- N -

Name 27, 85, 1162
names 76, 116, 119, 120
Naming Conventions 989
Nand 1052, 1059
nano 71
Natural Logarithm 135
Negate 919
net energy 32
net power 32
net power flow 32
network 323, 328
Network is Discrete 333
Network Name 327, 333
Neural Networks 323, 328
neuron 325
neurons 323, 328
new in 20-sim 2
New Simulation Plot 90
Newton Raphson 476
next 141
Next Camera 292
Next Local Maximum 84
Next Local Minimum 84
Nichols 321, 379
Nichols Chart 321, 363, 379
No Emphasising Threshold 19, 30
Node 597, 729, 859
Node.emx 575
Noise 183, 184, 1075, 1078, 1091, 1092
Nominal Operating Point 445
Non-actuated 261
non-interacting form 957
Nor 1053, 1060
norm 176
Normal Force 1112
Normal stop 221
norminf 177
Not 212
Not Equal 64, 210
Notch Filter 351, 352
NTC 899
Number 65, 1166
Number Hidden Neurons 327
Number of points to simulate before updating plot 27
Number of Splines 333
numerator 312, 368
numerical linearization 361
Numerical Output 466
Numerical Values 84
Nyquist 322, 378
Nyquist Diagram 322, 378
Nyquist Plot 363

- O -

Object Attributes 274
Object Tree 274
objects 274, 277, 290
octave 505, 507
octaveforge 507
offset 299, 956

OK 466
Omega (3x1) 268
Omega and Velocity (3x1) 268
Omega-X 268
Omega-Y 268
Omega-Z 268
one 1073
One Junction 526, 534, 1146, 1148
One Step 83
One Step Simulation 83, 88
One_in 64
One_out 64
OneJunction 525
Only Frames 279
OpAmp 575
OpAmp.emx 575
Open 22
open chain mechanism 252
open end 788
Open Model 22
Operating Point 92, 355
operating system 5
operational amplifier 575
Operators 14, 195
optimization 470, 476
Optimization Method 470
Optimization Methods 476
Optimization Results 470
optimization run 83
Optimize Divisions 27
Optimize Duplicate Expression 27
Optimize Equation Structure 27
Optimize Static Expressions 27
Optimizing Equation Structure 45
Or 208, 1053, 1060
order 333, 390, 402
Order of Excecution 51, 236
Order of Execution 51, 236
Orientation 64, 65, 268, 296, 1126, 1156, 1158, 1162
Orientation Info 1156
orifice 628
orifice area 592
Origin 299
OrthoGraphic 292
oscillating 390, 402
oscillating signal generators 390, 402
Other Motors 453
output 968, 992
Output After Each 77, 97
output delay 310, 366
Output Equations 45
Output Filename 255
Output is not used 27
Output position 261
output probes 85
Output Sigma 383
output signal 63
Output tab 12
overshoot 318, 376

- P -

P 969, 986, 1009
P Controller 343, 986, 1009
p_vapour 592
Pack 22, 23, 39
Packed Files 23
Pade Time Delay 902
Pair-wise transfer function 352
Parallel 295
Parallel Form 957
Parameter is not used 27
parameter sweep 83, 466
Parameter Sweeps 466
parameters 76, 119, 413
Parameters/Initial Values Editor 76
Parasitic 687
ParasiticVolume 687
Partial Cubic 390, 402
Partial Trapezoidal 390, 402
Pass Band 348

PD 986, 1010
PD Controller 344, 986, 1010
peaks 352
penumbra 295
Permanent Magnet Motorsunction 430
Perpendicular Search 476
Perspective 292
phase 319, 377, 382
Phase Margin 319, 377
phase to phase inductance 439
phase to phase resistance 439
Phased Sine Wave.emx 1086
Phong 298
physical domain 67
Physical Domains 67
physical model 1122, 1153
physical systems 515
pi 971, 987, 1011, 1073
PI Controller 345, 987, 1011
pico 71
pictures 15
PID 988, 1012
PID Compensator 347
PID Control 954
PID Controller 345, 346, 347, 988, 1012
PID-1 Controller 345
PID-2 Controller 346
piezo actuator 564, 565, 821, 822
PilotOperatedCheckValve 644
PilotOperatedCheckValve-States 671
planar motion 773
Plane Distance 292
PlanetaryGear 744
plant 308, 954
Plot Properties Editor 79
plot window 2
Plus 920
PlusMinus 147, 920
Pneumatic 67
point mass 778
point model 782
Polack Ribiere 476
Pole Zero 320, 380
Pole Zero Diagram 320, 380
pole zero notation 314, 372
Poles 314, 363, 372, 916
Poles and Zeros (including root locus) 363
Polynomial 390, 402
Polynomials 312, 368
Port is not used 27
Port Names 111
Port Properties 1141, 1162
Port Relations tab 64
Port Variables 67, 1162
Ports 62, 65, 1141, 1162, 1166
port-size 37
position 257, 268, 296, 398, 789, 817, 856
Position (3x1) 268
Position Sensor 789, 817
Position/Orientation 268
PositionActuator 701, 826
PositionActuator-Relative 698, 825
PositionSensor-Absolute 767, 870
PositionSensor-Relative 768, 871
Position-X 268
Position-Y 268
Position-Z 268
positive direction 776
Possible loss of data at type conversion 27
Potentiometer 578, 768
Power 32, 67, 201, 792, 928
power flow 32
Power Interaction Port 261
Power ports 111
power sensor 528
power spectral density 352, 383
powerflow 792
PowerSensor 633, 769
PowerSplitter 529
powertport 67

pre-act 957
Predefined Constants 118
Predefined Variables 122
preferred 64
Preferred Causality 32, 46, 233
Preferred out 64
Prefilter 308, 352
prefix 71
Prefix Minus Sign 212
Prefix Plus Sign 212
prefixes 69
Prepare Scripting Folder 510
Pressing Mode 24
pressure 592
Pressure Compensator 646, 672
pressurereducingvalve 650
pressurereducingvalve-states 675
pressurereliefvalve 648
pressurereliefvalve-states 676
pressuresensor 634
pressuresource 613
previous 142
Previous Local Maximum 84
Previous Local Minimum 84
Previous Runs 83
process 954
process output 954
Process tab 12
process variable 954
processor 5
professional 6
profile 390, 398, 402
Profiles 279, 390, 402
Programming Language 455
Progress Bar Update Interval (ms,0=don't show) 19, 30
Project 27
Projection 292
Properties 247, 274, 1162
Property Page 19, 30
Proportional Band 955
proportional gain 955
PSD 383
Psensor 529, 1126
Pseudo Domain 70
Pseudo Pneumatic 67
PseudoHydraulic 67
PseudoThermal 67
PseudoThermalH 67
PTC 899
Pulse 390, 402, 1077
PV 954
pwm 938

- Q -

Qsensor 530, 1126
Quadratic 295
quanties 76
Quantisize-Round 1025
Quantisize-Truncate 1026
quantities 66, 69, 70, 71
Quantities and Units Editor 66
Quantities Mismatch 27
QuantitiesAndUnits.ini 66
Quantity 71, 85, 116, 119, 120
Quantity name 71

- R -

R 296, 530
R-2 559
R-3 547
RackPinionGear 745, 880
radiation 892
Radius 288
Raising Power 928
Ramp 183, 390, 402, 1078
ran 183
Random 122, 183, 1078, 1092
random form 46, 233
random integers 1094
random noise 183, 1077
random numbers 1077

Random Seed 184, 1077
range 229
ranges 229
Rate 48, 235, 939, 957
Rate Limiter 936, 939
Rates 85
ratio 519, 522
raw Pseudo bonds As Dashed Bonds 27
Ray Tracing 279
Read Datafile 83
real 124, 126, 382
Real Time 3D Animation 83
Real Time Toolbox 454
Real-Imag 382
Real-Time simulation 97
Rectangular Window 914
Rectifier 579
red 296
Reduction Tolerance 363
Reference Body is Floating option 257
Reference Frame 282, 296
Registration/Update License 9
Regularization 333
Relative 104, 105
relative tolerance 361
Relay 940
RelayHysteresis 942
Remove redundant equations 27, 45
render 298
Rendering 298
Repeat 219
Repeat Until 219
Replace parameters when this model is used to update another model 27
Replay 83
Representation 247
Request License 9
requirements 5
resample 91
Reserved Words 113
reset 956
reset time 956
Resettable Integration 137, 914
resint 137
Resistance 1150
Resistor 522, 530, 579, 897
Resistor.emx 579
resolution 352
resonance frequencies 352
resonant frequencies 352
resonant peaks 352
Restore Default 19, 30
Return Angle 385
Return Time 398
return_time 390, 402
reverse acting 955
rewrite equations 48, 235
rewriting equations 46, 233
rho 592
right hand frames 292
ripple 348, 431
rise time 318, 376
Root Locus Gain 370
root locus plot 320, 380
Root Mean Square 1067
Rotation 67
Rotation (3x3) 268
rotation speed 770
rotational inertia 728, 778
rotational spring 729
round 137
Rounding 1025
Row 37
Rows 177, 229, 1162
Rows/Columns 1141, 1162
RS232 19, 30
Run 83
Run Number 84
Run Properties Editor 77
run tasks 505

Runge Kutta 2 104
Runge Kutta 4 104
Runge Kutta Dormand Prince 8 105
Runge-Kutta-Fehlberg 104
Running a 1-D B-Spline Network 334
Running a Simulation 83
Runtime 110

- S -

s 99, 363
s&

shock absorber 860
Show Equations 54, 238
Show Grid 27
Show Names For New Submodel 19, 30
Show results after processing 27
Show Variables: 85
Shuttle Valve 652, 678
SI Symbol 71
sidops 107, 198
SIDOPS+ 107
Sigma 1027
Sigmoid 325
Sign 138, 928, 931
Signal 17
Signal Ports 111
SignalGenerator-Cycloid 1073
SignalGenerator-FileInput 1074
SignalGenerator-GaussianNoise 1075, 1091
SignalGenerator-Pulse 1077
SignalGenerator-Ramp 1078
SignalGenerator-Random 1078, 1092
SignalGenerator-RandomInteger 1094
SignalGenerator-Step 1080
SignalGenerator-StepTime 1081
SignalGenerator-Sweep 1081
SignalGenerator-Time 1084
SignalGenerator-VariableBlock 1095
SignalMonitor 921, 1085
Signals 1126
Silent stop 221
Simple Transformer 533
Simplification 26, 1128
Simplification,Bond Graph Models 1128
Simplify Model 26, 1128
Simulation Code 45
simulator 2
Simulink 10, 39, 455
Simulink S-function 455
sin 191
Sine 390, 402, 929
Sine Sweep 1081
Sine Wave 1086, 1087
single 6
single acting cylinder 599, 601, 605, 607
single license 7
Single-Step Simulation 83, 88
sinh 192
SISO 312, 363, 368
size 37, 279
skew 178
Skip 89
SkyHookDamper 862
Slides 793
Small Rotations 773, 775
Smart Contraint Solving 106
Smooth Line 24
Snap to Grid 27
Solve algebraic variables 27
Solving Differential Equations 48, 235
source 523, 524, 531, 532
Sources 1150
SP 954, 968, 992
Specular color 295
Sphere 285, 299
spindle 748, 883
Spiral 288
spline function 328
Splitter 921
Spot Angle 295
Spot Exponent 295
Spot Light 295
Spring 729, 863
Spring Damper 252
Spring Damper Joint 252
SpringDamper 730, 864
spring-damper 730
Springs 793
sqr 138
sqrt 139
square 138, 291, 929

Square Root 139
Square Root with Sign 930
Square Sign 931
Square Wave 1087
squareroot 930
stall torque 440
Stand-alone ANSI-C 455
standard 6
standard deviation 383, 1067, 1098
standard deviations 383
Standard Elements 1126
standard form 957
StandardCameras_Orthographic.scn 292
Star and Delta Networks 439
Start 77
Start Angle 385
Start of simulation 355
start plane 281
start position 283
Start Time 398
start_time 390, 402
state 48, 235
State and Time Events 115
state events 77, 115
State Space 916
State Space Models 310, 363, 366
State Variable Filter 907
Statements 14, 214
States 85
States/Rates 85
state-space description 355
Static and Dynamic Phenomena 1116
Static Backgrounds 44
steady state 92
Steady State Gain 370
steady state value 318, 376
Steepest Descent 476
steepness 440
Step 184, 318, 376, 1080
Step Response 318, 363, 376
Step Size 103, 104, 105
Step-by-Step 89
StepMotor 703
Steps 466
STF 533
STF-2 562
STF-3 550
stiffness 796, 807
Stop 83, 97
Stop Angle 385
Stop band 348
Stop Time 398
stop_time 390, 402
Stopsimulation 221
Storage element 515, 520
Straight Lines 24
String 124, 127
Stroke 385, 390, 398, 402
strokes 32
structural connection 729, 859
Submodel 24, 27, 1144
Submodel Name to be used in 20-sim 255
Submodels 20
Subtract 920
Subtract DC-component 352
Subtraction 197
sum 147
swapbytes 213
Sweep 466, 1081
Switch 220, 570, 580, 943, 944
Switch Case 220
Switch-Break 943
Switch-Default 943
switching 534
Switch-Level 580
Switch-Make 944
Switch-Maximum 944
Switch-Minimum 945
sym 178
Symbol 71

Symbol Prefixes 71
Symbolic Linear Systems 365
Symbolic Linearization 361
Symplectic gyrator 532
Syntax Highlighting Threshold 19, 30
system 954
System Gain 370
system output 954

- T -

Table 166, 931
Table of Contents 41
Tabular Function 931
Tachometer 770
tan 194, 933
tanh 193
Tank 685, 686
Tank with resistance 686
TankNoRes 685
Tapping Mode 24
targets 455, 458
Targets.ini 455, 458
taskbar 12, 14, 15
TCP 19, 30
tdelay 139
Tecnotion 425
Tecnotion motors 425
temperaturesensor 900
temperatureresource 898
tera 71
Terminal 65
terminal inductance 439
terminal resistance 439
Terminals 56, 65, 1166
Texture 299
Texture Offset 299
texture wrap 299
TF 535
TF-2 563
TF-3.emx 550
TF-element 1151
Then 214, 215, 217
Thermal 67, 889
Thermal Duty Cycle 445
Thermal Model 454
ThermalConductance 893
ThermalResistor 894
thermistor 899
ThreeDBody 807
ThreeDFixedWorld 810
ThreeDForceActuator 811
ThreeDForceActuator-Relative 810
ThreeDMass 812
ThreeDPoint-X 813
ThreeDPositionSensor-Y 817
threedspringdamper 818
ThreeDZeroForce 819
throttling band 955
Through 67, 1154
tilde 178
Time 122, 1084
Time Delay 139, 903
Time Domain Toolbox 465
Time Events 115
Time window 914
timeevent 145, 1032
timer 100
Timing 77
timingbelt 750, 884
Title 27
To 218
To Matlab 76, 85
ToDoMatlab 1043, 1044
Toggle Full Screen 96
tokens 455, 461
Tolerance 363, 470
Tolerances 27, 355
ToMatlab 222, 1047
ToMatlab-Plot 1044
ToMatlab-Store 1045
ToMatlab-Timed 1046

ToMatlab-TimedPlot 1046
ToMatlab-TimedStore 1047
toolbox 2, 6
Torque 705, 792
torque (constant) 705
torque (variable) 706, 707
torque constant 440
torque f 430
Torque.emx 705
TorqueActuator 707
TorqueActuator-Relative 706
Torque-Relative 705
TorqueSensor-Relative 771
Torus 287
trace 179
tracking time constant 959
train 323, 328
Training 328
Training a 1-D B-Spline Network 338
transducer 770
Transfer Function 352, 916
Transfer Functions 312, 363, 368, 1099
TransferFunction 1102
TransferFunctionWithDeadTime 1107
Transform Dependents states 27
transformation 363, 775
transformer 524, 533, 535
Transformer.emx 582
Transformers 1151
Translation 67
translational spring 863
Transmission 752, 887
Transmission-Universal 888
transparency 299
transpose 180
Trapezoidal 390, 402
Triangle Wave 1089
Trigger Type 1055
Trigger-type Flipflop 1062
TriggerTypeFlipFlop-Continuous 1055
TriggerTypeFlipFlop-Discrete 1062
troubleshooting 10
trunc 140
Truncate 1026
Truncation 933
Tube 287, 288
turbulent flow 592, 628, 631
turns 288, 768
Tustin 363
two port 516
TwoDAccelerationSensor 790
TwoDBody 778, 809
TwoDFixedWorld 787
TwoDForceSensor 792
TwoDLinearActuator-X 802
TwoDLinearSlide-X 798
TwoDPoint-X 782
TwoDPositionSensor 789
TwoDPowerSensor 792
TwoDSpring 796
TwoDVelocitySensor 790
TwoDZeroForce 788
Two-Terminal 65, 1166
TwoTwoWayDirectionalValve 654
twotwowayproportionalvalve 656
twotwowayproportionalvalve-states 682
TwoTwoWayValve-States 680
Type 85, 1162
Type conversion found 27
Type Editor 1141, 1156, 1162
type of motion 398
Types 76, 116, 119, 120

- U -

umbra 295
unattended installation 10
Unbalance 733
Uniform distribution 1075, 1091
uninstalling 9
Unipolar Sigmoid 327
Unit 76, 85, 116, 119, 120

Unit Conversion when SI disabled 27
Unit Conversion when SI enabled 27
Unit Delay 130, 142
Unit is unknown 27
Unit missing for variable when SI disabled 27
Unit name 71
Unit Symbol 71
units 66, 69, 71, 76
Units Editor 66
Universal Joint coupling 755
Universal Notch Filter 352
Unlit 298
Unlit Flat 298
Unpack 22, 23, 39
Until 219
Unwrap Phase 319, 377
Updates 43
use 411
Use as Result 466
Use BDF Integration Scheme 105
Use Built-in Compiler 77
Use Newton Solver 105

- V -

Value 76, 85
Values 76, 116, 119, 466
vapour pressure 592
Variable 85
Variable Chooser 85
Variable is never given a value 27
Variable is not used 27
Variable is set but not used 27
Variable Multiple set 27
Variable name 71
Variable Pulse 1096
variable sampletime 100
variable voltage 590
variabledisplacementmotor 613
variabledisplacementmotor-leakage 624
variabledisplacementpump 614
variabledisplacementpump-leakage 626
variablelaminarresistance 630
variableorifice 631
Variables 85, 120, 122
Variables: 85
variance 1068, 1075, 1091, 1098
Variation Analysis 498
VCVS 586
Vector 229, 230, 294
Vectors 37, 85, 228, 229, 299
Velocity 268, 398, 708, 790, 829
Velocity (3x1) 268
velocity (constant) 707, 708, 829
velocity (variable) 708, 709, 830, 831
VelocityActuator 709, 831
VelocityActuator-Relative 708, 830
VelocityActuator-Relative.emx 708
Velocity-Relative 707, 829
VelocitySensor-Absolute 771, 872
VelocitySensor-Relative 772, 873
Velocity-X 268
Velocity-Y 268
Velocity-Z 268
Version 6, 27
View Modes 249
viewer 6
Vode Adams 105
voltage 590, 591
voltage constant 440
voltage controlled voltage source 586
VoltageSensorCurrentSensor.emx 584
VoltageSource.emx 591
VoltageSource-VCVS 586
VoltageSource-VCVS.emx 586
Volume 687
Volume-Constant 687

- W -

Warning 222
WaveGenerator-Cosine 1085
WaveGenerator-PhasedSine 1086
WaveGenerator-Saw 1086

WaveGenerator-Sine 1087
WaveGenerator-Square 1087
Wavegenerator-SquareExp 1088
WaveGenerator-Triangle 1089
weight 328
Welcome 1
Wet and Dry Friction 1114
While 219
While Do 219
White Space 114
Window 910, 914
Windows Media Video 279
WireFrame 281, 298
Working Point 355
WormGear 756
wrap 299
Writing Comments 115
Writing DLL's 151
Writing Dynamic DLL's 159
Writing your own Scripts 514

- X -

X-files 290
XMLRPC 19, 30
XML-RPC 506
Xor 208, 1051, 1063
X-rotation 261
X-translation 261
X-value 84
X-Y-X Euler 257

- Y -

Y-Axis 299
yellow 276
yerk 390, 398, 402
YouTube 279
Y-rotation 261
Y-translation 261
Y-value 84

- Z -

z 363
Z-1 1020
Z-Axis 299
zero 1090
Zero Junction 534, 1146
Zero Order Hold 140, 1023
ZeroForce 867
zerojunction 536, 537
Zeros 314, 363, 372, 916
Zeros and Poles 363
ZeroTorque 735
zip-file 23
Zooming 292
Z-rotation 261
Z-translation 261
Z-X-Z-Euler 296

20-sim is a modeling and simulation program that runs under Microsoft Windows. With 20-sim you can simulate the behavior of dynamic systems, such as electrical, mechanical and hydraulic systems or any combination of these.

20-sim fully supports graphical modeling, allowing to design and analyze dynamic systems in a intuitive and user friendly way, without compromising power. 20-sim supports the use of components. This allows you to enter models as in an engineering sketch: by choosing components from the library and connecting them, your engineering scheme is actually rebuilt, without entering a single line of math!

9 789079 499182 >

Controllab Products B.V. Hengelosestraat 500, 7521 AN Enschede, The Netherlands
T +31(0)85 773 18 72, E info@controllab.nl, **www.20sim.com**