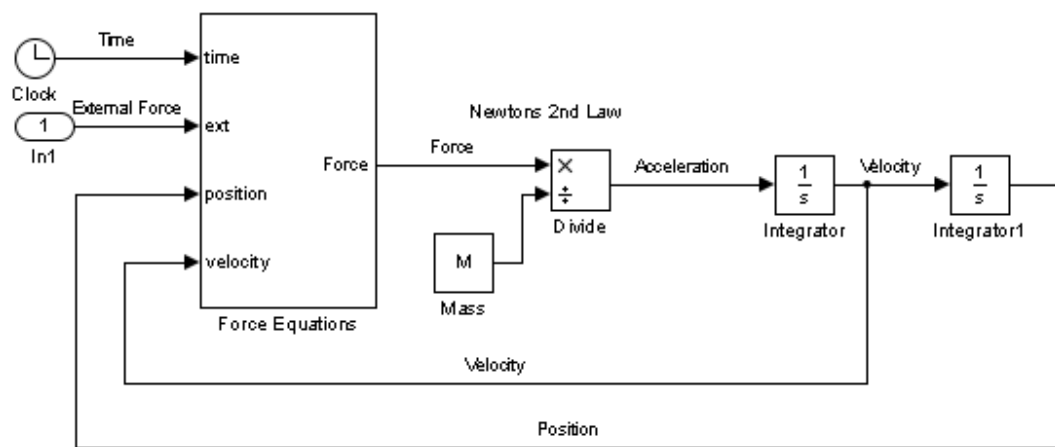


An Introduction to Using Simulink



An Introduction To Using Simulink.

Eric Peasley, Department of Engineering Science, University of Oxford
MATLAB and Simulink are registered Trade Marks of MathWorks Inc

Table of Contents

Introduction.....	4
The Solver.....	5
Sources Library.....	6
Scope Block.....	8
Maths Library.....	9
User Defined Functions.....	12
Look Up Tables.....	12
Signal Routing.....	13
Subsystems.....	15
Logic Operations.....	16
Modeling Continuous Systems.....	17
Dynamic Systems.....	18
Ordinary Differential Equations (ODE).....	19
Simultaneous Ordinary Differential Equation.....	20
Linear Systems.....	21
Modeling Discrete Systems.....	23
Discrete Transfer Functions.....	24
Simulink Online Documentation.....	26
Model Editing.....	27
Block Libraries.....	29

Introduction

This document is part of the Introduction to Using Simulink seminar. This seminar is designed for people that have never used Simulink. There are two components to the seminar. There are exercises in a separate document that will take you step by step through the tasks required to build and use a Simulink model. Once you get started using Simulink, you will find a lot of the functionality is self intuitive. Inevitably, there are things that need a bit more explanation. So the second part of the Seminar is a talk and demonstration. This document contains the notes for the talk.

It would be impossible to put everything about Simulink into such a short document, so I have concentrated on the parts of the package that I consider the most useful. I have also tried to highlight features that are not obvious to the casual user. The intention is that you use these notes as a reference when carrying out the exercises and when building your own models. Although these notes have their limits, I hope that they should be sufficient to get you started using the package and that they cover most of your modeling needs.

This is not a Simulink manual. Sooner or later you will need to know more detail about something within Simulink. This document is intended to be used in conjunction to the documentation available with in the package. Mathworks provide extensive online documentation for Simulink that can be accessed using the MATLAB help system. There is so much online documentation that not many people have the time or inclination to read all of it. So an aim of the Seminar is to emphasize the things that you ought to know about Simulink and to give you some idea about where you can find any other information that you require.

The Solver

Most of the time, you can just use the default settings to run your model. However you will find that sometimes you will want the model to use smaller steps, or fixed width steps. This is all configurable on the Solver page of the Configuration Parameters.

From the menu bar on your model select

Simulation Model Configuration Parameters

Then on the select menu on the left hand side, select **Solver**.

Simulation time

To the right at the top you will find the Simulation time box. I suggest that you leave the start time as zero. The stop time is identical to the stop time on the icon bar at the top of your model.

Solver Options

There are two types of solver. By default, a variable step solver is used. This will automatically adjust the step size as the model runs. If you are using variable step I suggest that you keep the default solver (**ode45**). Set the **Max step size** to a fixed value to improve the smoothness of any graphs if required.

Switching to a fixed width solver will be necessary for models with discrete components. If it also has no continuous components, change the solver to Discrete (no continuous states). I also suggest that you set the step size to a known value.

The fixed solvers are numbered in order of simplicity. ode1 being the simplest.

For more information about solvers, click on the **Help** button at the bottom of the configuration parameters window, while you are viewing the solver section. At the bottom of the page, select **Choosing a Solver**.

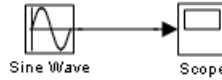
Zero-crossing options

At the bottom of the page you will find the Zero-crossing options box. You can disable zero crossing control if you think it will help.

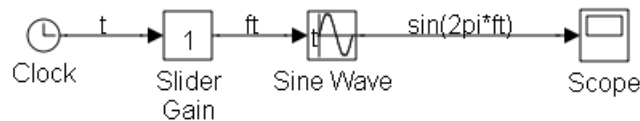
Sources Library

The Sine Wave Block

Most of the blocks in the source library are self expansionary. The basic sin wave block is easy to use. You just set the frequency and amplitude in the block parameters.



In the exercises we use the sine wave block to look at the frequency response of a system. However, repeatedly editing the block parameters to change the frequency of the sine wave is tedious. The alternative is to use an external time input to the block.



To do this, in the Sine Wave block parameters you set the **Time** parameter to **Use external signal** and set the **frequency** to 2π . You then connect a clock to the new input via a slider gain. The slider then sets the frequency of the sine wave.

From Workspace Block

This block is used for importing data into a Simulink model from the MATLAB workspace. A simple way of doing this is to place the data to be imported into a matrix, as shown below.

time	Signal 1	Signal 2
0	1	1
5	2	4
10	3	9
15	4	16
20	5	25

The variable name of the matrix is then entered into the block parameters of the block. Notice that the data in the above matrix is very coarse. Simulink will use interpolation to calculate values at any times between the given points.

The block will also read in data from a structure. The MATLAB script below will generate a structure to read into Simulink.

```
%Generate data to go into simulink

%Generate time vector
n = 21;
t = linspace(0,10,n);
t = t';           %transpose into a column

%Generate two sets of data
x = rand(n,1);
y = t.^2-50;

%Write data into structure
Mydata.time = t;           %Time vector







%Signal
Mydata.signals.values = [x y];
Mydata.signals.dimensions = 2;
```

To use enter **Mydata** into the **data** parameter of a **From Workspace** block.

Scope Block

Axes Properties

Right click on the axes, then select Axes Properties. This allows you to change the y limits of the axis and the axes title. The default title of an axis is the label of the signal being plotted. So it is probably better to label the signal than explicitly set the title here.

Scope Icons			
	Print		Magnify in x and y
	Parameters See below		Magnify in x only
	Auto Scale		Magnify in y only

Parameters / General

Click on the **Parameters** icon, then select the **General** tab. This allows you to set the **number of axes** and the **time range** of the graph.

Parameters / Data History

Click on the **Parameters** icon, then on the **Data History** tab. This allows you to save the scope data to the MATLAB workspace and also limit the size of the data stored. The data can be stored into an array or a structure. The function below plots the data stored in a structure.

```
function PlotScopeData(ScopeData)
%Plot graphs of data saved by a scope block

n = length(ScopeData.signals);    %Number of axes

t = ScopeData.time;               %time vector

for k = 1:n
    subplot(n,1,k);               %One subplot for each axes
    x = ScopeData.signals(k).values;
    plot(t,x);
    xlabel('time');
    ylabel(ScopeData.signals(k).label);
    title(ScopeData.signals(k).title);
end
```

The function can also be used to plot data from a **To Workspace** block.

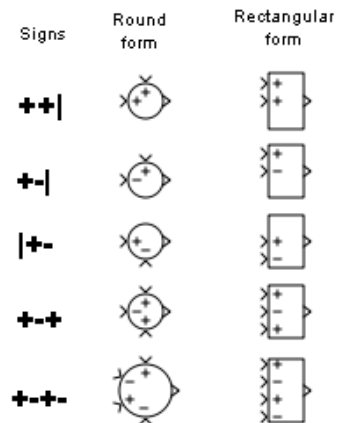
Parameters / Style

This allows you to change the colour and markers used to plot the data.

Maths Library

Sum, Add and Subtract Blocks

The add, subtract and sum blocks are essentially the same. By changing the **Icon shape** and **List of signs** in the block parameter you can convert one into the other. The diagram to the right shows examples of the what happens when you change the **List of signs** and the **Icon Shape**.



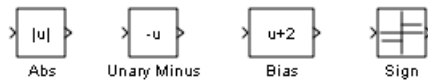
Product and Divide Blocks.

The product and divide blocks are also interchangeable. You can use a list of asterisks and forward slashes in the **number of inputs** block parameter to define the number and distribution of the inputs. For example, the block to the right was created by inserting `/*/**` into the number of inputs parameter.



Mathematical Functions

There are lots of blocks for different mathematical functions.



The **Abs** block finds the magnitude or absolute value of a signal, the **Unary Minus** block negates a signal and the **Bias** block adds a constant to a signal. The **Sign** block produces either plus or minus one depending on the sign (positive or negative) of the input.

There are also blocks which are a bit more general.

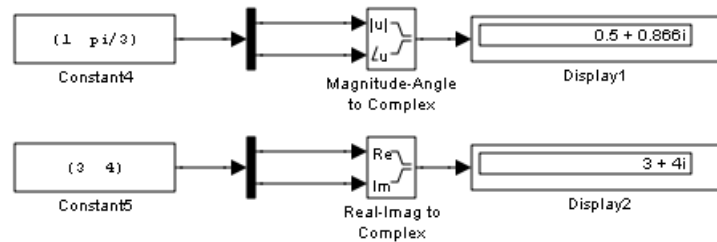


The **Math Function** block can perform many different functions, square, square root, log, reciprocal etc. A block parameter allows you to select which particular function you want. If you want a trig function then the **Trigonometric Function** block operates in a similar manner. The **Rounding Function** block rounds a number to an integer. There are four different MATLAB functions for doing this, floor, ceil, round and fix. You select the functionality you want. See the help on the MATLAB functions for more information.

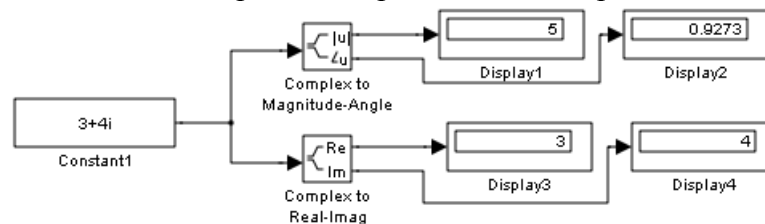
The **Polynomial** block allows you to evaluate a polynomial using the input as the argument. The coefficients of the polynomial are entered as a vector into a block parameter. This is similar to the MATLAB function polyval.

Complex Numbers

Signals in Simulink can be complex if desired. If it is conceivable that a block could work with complex numbers, it most probably will. There are special blocks for creating complex numbers.



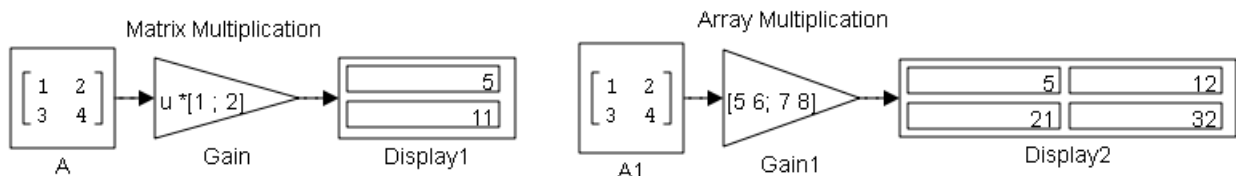
There are also blocks to extract the separate components of a complex number.



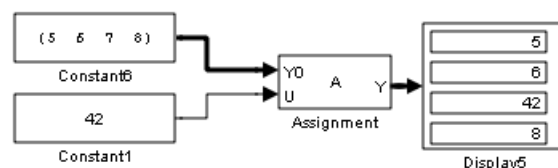
Unfortunately integrators don't work with complex inputs.

Matrices

As Simulink is MATLAB based, it is not surprising that there is extensive support for use of matrices and multidimensional arrays. For example, the basic gain block can be configured to perform matrix or array multiplication.



There are also special blocks for dealing with matrices. The **Assignment** block enables you to replace a particular element with another signal.

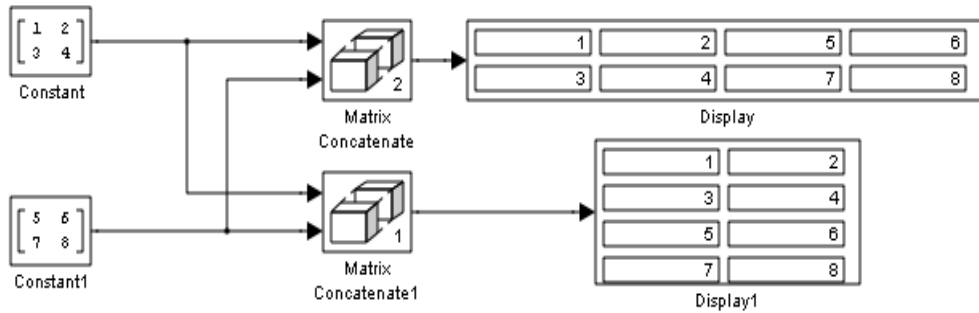


Here a block parameter is used to specify that the 3rd element in Y0 is to be replaced by what ever is on the input U.

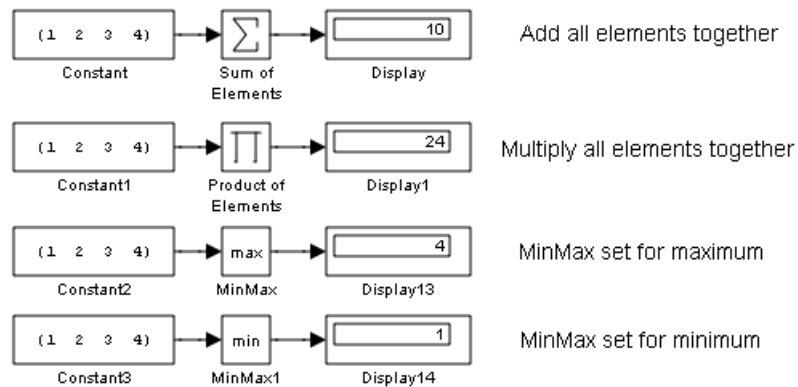
Note that if you enter a vector into constant block, by default it will produce a column vector, even if you enter a row vector. The way to overcome this problem is to untick the **Interpret vector parameter as 1-D** in the constant block parameters.

The **Permute** block swaps dimensions. In 2D it is like transpose, but it can also work with multiple dimensioned arrays.

There is also **Concatenate** that can add rows (dimension 1) or columns (dimension 2) to a matrix.

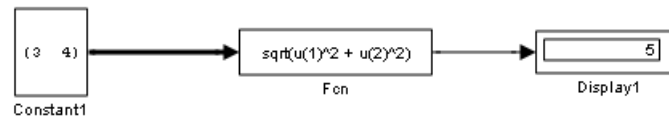


There are also blocks for operations on vector elements.

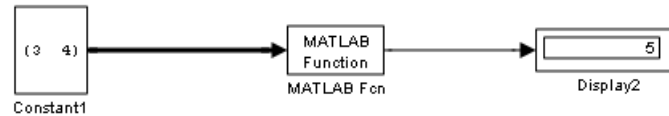


User Defined Functions

Sometimes you cannot find the exact function that you want. If this is the case, then I suggest that you first try the block called **Fcn**.



This allows you to enter a mathematical expression using a restrict set of operators and functions. To see what is allowed, click on the Help in the block parameters. If you need something a bit more complicated then you can write your own MATLAB function and use the block **MATLAB Fcn**.



This block has been configured to call the MATLAB function myabs below. x is the input and y is the output.

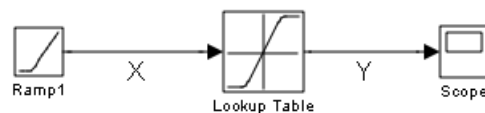
```
function y = myabs(x)

a = x.^2;
b = sum(a);
y = sqrt(b);
```

If you are using Simulink to create a program using the Real Time Workshop, then you will need to use the Embedded Matlab Function or an S Function. S functions are used to embed Matlab, C or Fortran into your model.

Look Up Tables

If all else fails, then you can use a look up table to define your function.

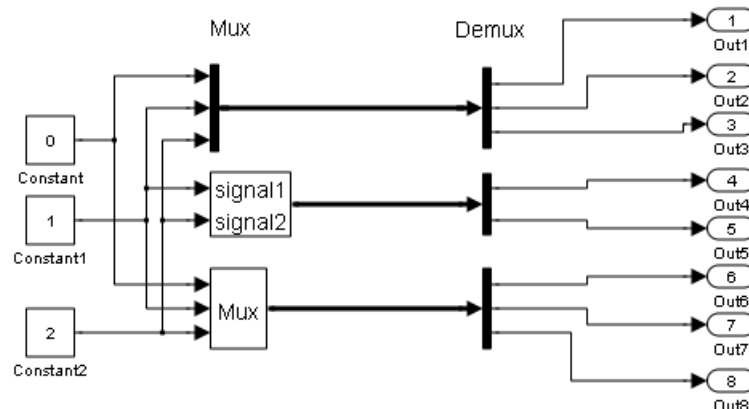


You need two vectors to do this. A vector of input values and a vector containing the table of corresponding output values. You enter these in to the appropriate block parameter. For intermediate values, the output is calculated using interpolation. Extrapolation is used to calculate values beyond the specified input range.

Signal Routing

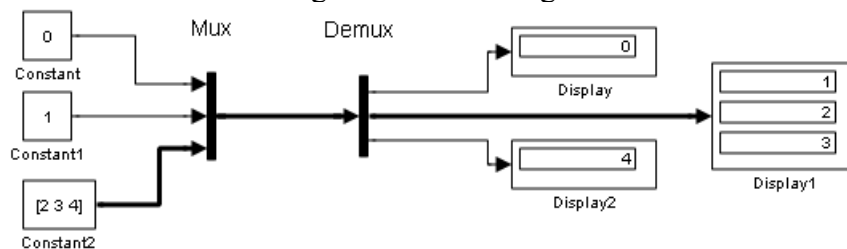
Mux and Bus

A **mux** is a wire that contains multiple signals. You can create a mux using the mux block.

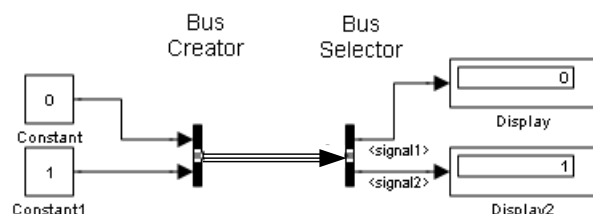


The three blocks under the word “Mux” are all mux blocks with different display options. All signals inside a mux must be of the same type and size. You can feed a mux signal directly into a mathematical block so that the block will operate on all signals in the mux.

A **demux** block separates out the individual wires in a mux. The **Number of outputs** block parameter of a demux block normally contains a single integer. If you enter a vector in the Number of outputs parameter, each number in the vector specifies the number of signals in a particular output. For example, [1 3 1], specifies three outputs. The first and last outputs are single wires. The middle output is a mux with three signals as in the diagram below.






A **bus** is like a wiring loom in a car. A convenient way keeping things tidy and not having a birds nest of wires running everywhere. It can contain signals of different type and sizes. That means that it may not be possible to feed a bus into a mathematical block. A bus and a mux are almost interchangeable, except when the bus contains different types of signal.



To see what signals you have, from the model menu bar select :-

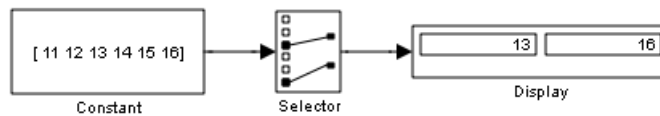
Display
Signals & Ports
Wide Nonscalar Lines

Scalar 
Mux 
Bus 

The picture to the right above shows how the signals will then be displayed.

Selectors

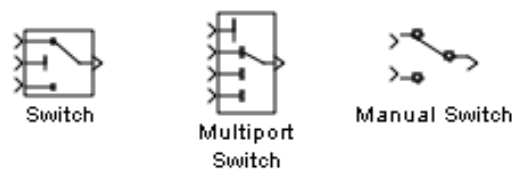
Selectors are the Simulink equivalent of subscripting in MATLAB.



The selector above has been configured to extract the 3rd and the 6th element of the input vector. This has been done by setting the number of inputs to 6 and the indexing to [3 6]. You can also configure selectors so that the indexes are obtained from an extra input. Selectors will also work with matrices and higher dimensional arrays.

Switches

Three types of switches.



The output of the **Switch** block is determined by the center pin. A comparison is carried out between the center input and an internal threshold value. You can select the comparison to be greater than, less than, equal to, etc. If the comparison is true, the output is switch to the top input, else the output is connected to the bottom input.

An integer on the top input of a **Multiport Switch** determines which of the other inputs will be connected to the output. If the top input is one the first of these inputs is selected. If the top input is two the second is selected etc, A **Vector Index** switch is essentially the same as a Multiport switch.

A **Manual Switch** is toggled by double clicking on the switch.


Subsystems

A subsystem is equivalent to a MATLAB script. A collection of blocks grouped together to carry out a particular task. You can create a subsystem from scratch by dragging in a Subsystem block from the Ports and Subsystems library, or you can select the blocks that you want to put into a subsystem and select


Diagram ► Subsystems & Model Reference ► Create Subsystem from Selection

from the model menu bar. If you need extra input and output ports, you will find them in the Ports and Subsystems library. They are also in the relevant source and sink library.

Enabled Subsystems

You can also add an enable input to your subsystem. To do this, drag the enable block  into your subsystem, from the Ports and Subsystems library. A one on the enable input enables the subsystem, a zero will disable it. The block parameters of the output ports allow you to configure them to either hold the current value or reset to a specified value when the subsystem is disabled.

Triggered Subsystems

You can also add a trigger input to your subsystem. The subsystem will then execute one step each time the appropriate signal is applied to the trigger input. You can configure the trigger input to react to a rising edge, a falling edge or both edges. 

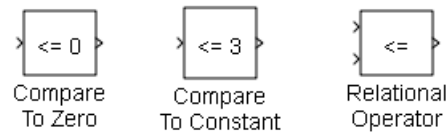
Program Control Subsystems (For, While, If and Case)

Within the Ports and Subsystems library, you will see subsystems that perform the Simulink equivalent of For and While loops and If and Case statements. The **For** and **While** loops will perform a number of loops for each step of the simulation. They each contain a control block that allows you to configure how many loops are performed.

It takes two sorts of block to implement an **If statement**. The If block itself is used to perform the test that determines what is going to happen. It produces two control signals, one for true the other for false. The control signals go to If Action subsystems. You put the blocks required for each condition into the appropriate Action subsystem. The **case statement** is performed in a similar way. There are two different blocks. The Case block itself determines which of the Case Action subsystems to perform.

Logic Operations

Comparison and Relational Operations



These three blocks compare a signal with either zero, a constant or another signal. The particular operation (greater than, equals to, greater or equals to etc) is selected by a block parameter. The output is a Boolean number.

Logic Operator

This block carries out the same operation as a Boolean logic gate. Which particular logic operation is selectable from a block parameters.

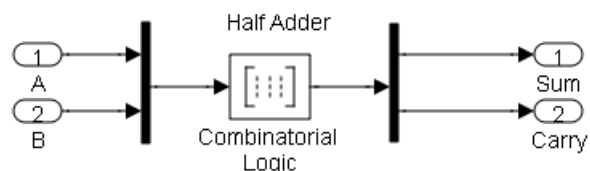


All the above are copies of the same block, but with different block parameter settings. The first has default rectangular icon shape. The rest have the icon shape set to distinctive. Most have only two inputs except for the third which has been set to 3 input. The first two are both AND gates, the rest are NAND, NOR and exclusive OR.

Combinatorial Logic Block

This block allows you to use a truth table. As an example, a half adder has the following truth table.

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



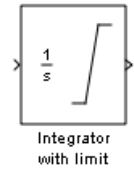
The left hand side of the truth table that defines the input values is always going to be the same depending only on the number of inputs. So you don't enter the left hand side. You just enter a matrix that defines what the outputs will do. For example, to implement the half adder above you enter [0 0 ; 1 0 ; 1 0 ; 0 1]

Modeling Continuous Systems

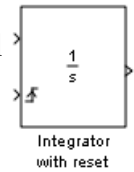
Integrators

There are various ways of configuring integrators with extra functionality.

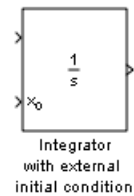
You can limit the output to a maximum and minimum value.



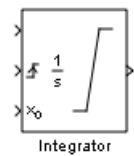
You can add an external reset signal, to force the output of the integrator back to its initial condition. You can configure the reset pin to act on a rising edge, falling edge or both.



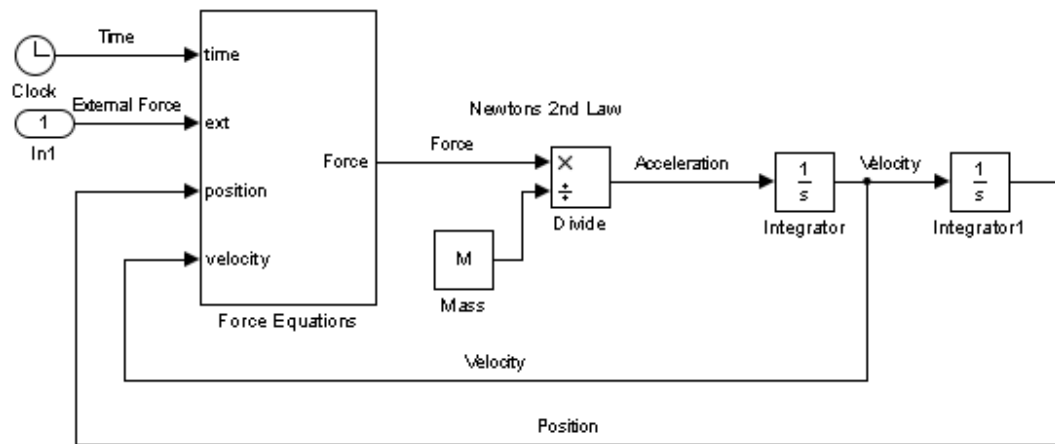
You can add an input for the initial condition.



Or a combination of the above.



Dynamic Systems



The above shows the general approach to modelling dynamic systems. You calculate the force, use Newton's 2nd law to calculate the acceleration, then integrate to get the velocity, then integrate again to obtain the position.

The above model is a general guide, it can get a bit more complicated. For example, the mass is required to calculate gravitational forces. It is also possible that the mass will be a function. For example, a rocket loses most of its mass as the fuel is burnt off.

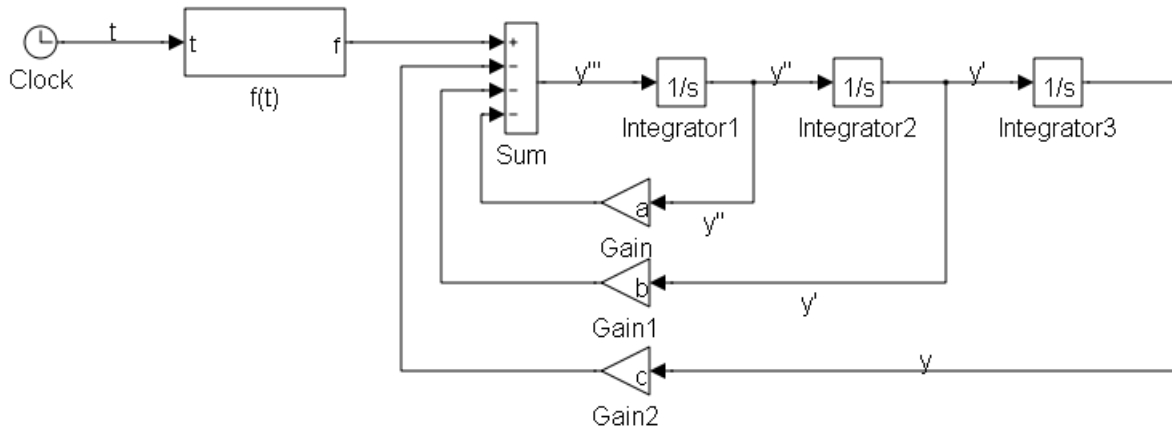
Ordinary Differential Equations (ODE)

The general rule for solving differential equations is to write the equation in terms of the highest differential. For example, consider the general second order equation below.

$$\ddot{y} + a \dot{y} + b y = f(t) \quad (1)$$

$$\ddot{y} = f(t) - a \dot{y} - b y$$

You then use integrators to obtain lower terms.

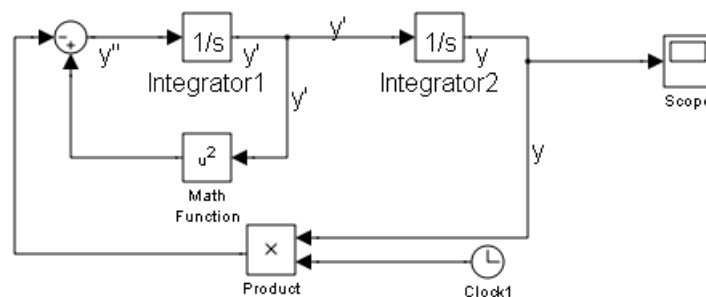


The right hand side of the equations is formed by feeding back these terms to form the expression required.

Notice that the model contains no differentiators, even though we are modeling a differential equation. Models with differentiators tend to produce a lot of noise, so are avoided if possible.

The next example is not linear or time invariant.

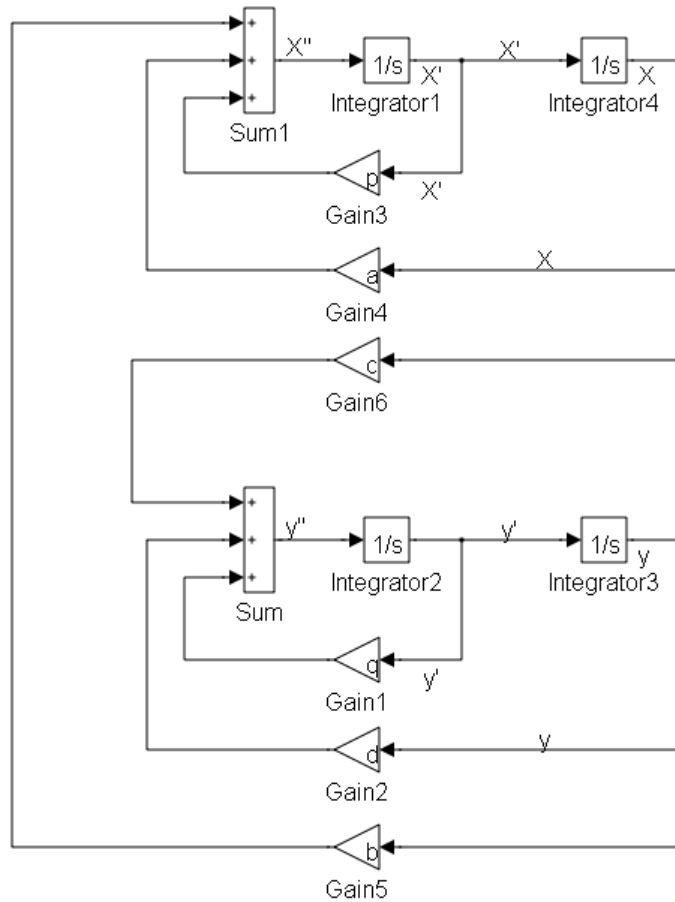
$$\ddot{y} = \dot{y}^2 - ty \quad (2)$$



Simultaneous Ordinary Differential Equation.

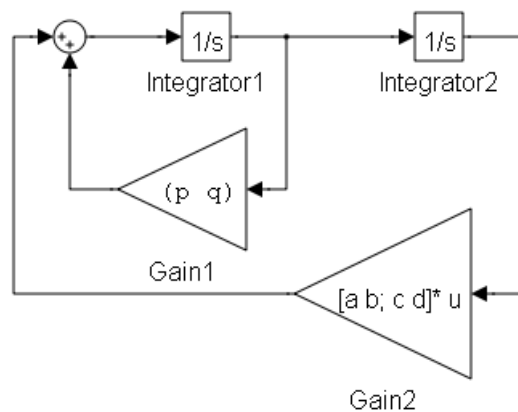
Of course Simulink is not limited to equations in one variable. Consider the simultaneous equation below.

$$\begin{aligned}\ddot{x} &= p \dot{x} + a x + b y \\ \ddot{y} &= q \dot{y} + c x + d y\end{aligned}\quad (3)$$



This can be simplified by using matrices.

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} p & \dot{x} \\ q & \dot{y} \end{bmatrix} + \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}\quad (4)$$



Linear Systems

Many systems can be modeled by linear, time invariant (LTI) differential equations, such as equation 5 below.

$$a_3 \ddot{y} + a_2 \dot{y} + a_1 \dot{y} + a_0 y = b_2 \ddot{x} + b_1 \dot{x} + b_0 x \quad (5)$$

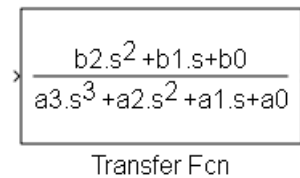
Where y is the output and x the input.

LTI systems can be represent by a transfer function.

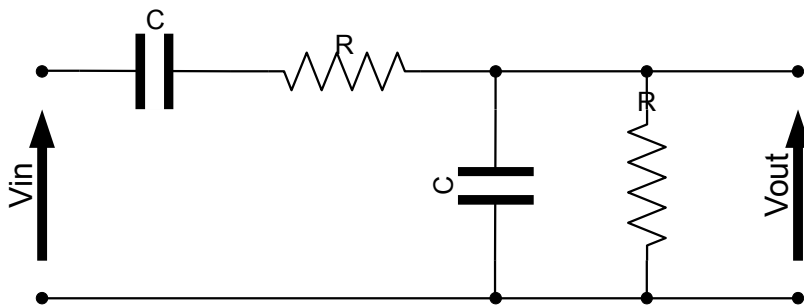
$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_2 s^2 + b_1 s + b_0}{a_3 s^3 + a_2 s^2 + a_1 s + a_0}$$

$$H(j\omega) = \frac{b_2 (j\omega)^2 + j\omega b_1 + b_0}{a_3 (j\omega)^3 + a_2 (j\omega)^2 + j\omega a_1 + a_0}$$

It is the convention in MATLAB to represent polynomial expressions with row vectors of the coefficients. So the numerator of the above transfer function is represented by $[b_2 \ b_1 \ b_0]$ and the denominator by $[a_3 \ a_2 \ a_1 \ a_0]$. Entering these two vectors to the appropriate block parameters of a transfer function block will produce the following block..



LTI transfer functions are used extensively in electronics to represent idealized electronic circuits. Take for example the circuit below.

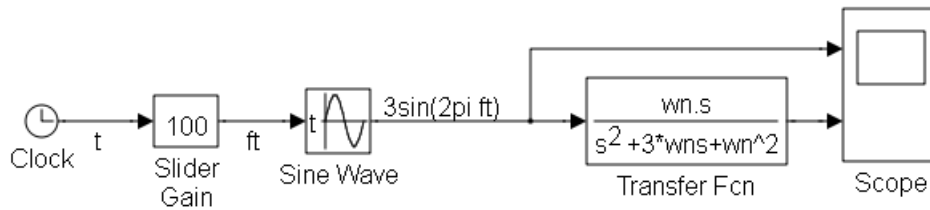


This circuit can be represented by the transfer function below.

$$H(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{s\omega_n}{s^2 + 3s\omega_n + \omega_n^2} \quad (6)$$

$$\text{Where } \omega_n = \frac{1}{RC}$$

The following model can be used to observe the behavior of this circuit in Simulink.



The numerator of transfer function was entered as $[wn \ 0]$ and the denominator was entered as $[1 \ 3*wn \ wn^2]$. Where wn is a predefined MATLAB variable.

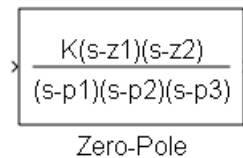
Poles and Zeros

An alternative way of representing a transfer functions is to use the pole, zero description. If you solve the numerator polynomial you get the zeros. So called because the transfer function is zero at that value. If you solve the denominator polynomial, you get the poles. They are called poles because if you plot the absolute value of a transfer function, it looks a bit like a tent, with the poles being the location of the tent poles.

The transfer function can then be represented in its pole zero form.

$$H(s) = K \frac{(s - z_1)(s - z_2)}{(s - p_1)(s - p_2)(s - p_3)} \quad (7)$$

Where p_i is a pole and z_i is a zero and K is a constant. You can model the transfer function in this form using a zero-pole block.



To configure this block you provide a vector for the numerator and the denominator. In this case the numerator is $[z1 \ z2]$ and the denominator is $[p1 \ p2 \ p3]$ and the gain is K .

NOTE

The MATLAB function **roots** will solve a polynomial, given the coefficients of the polynomial. The function **poly** does the opposite. Given the roots of a polynomial, it will return the coefficients of the polynomial.

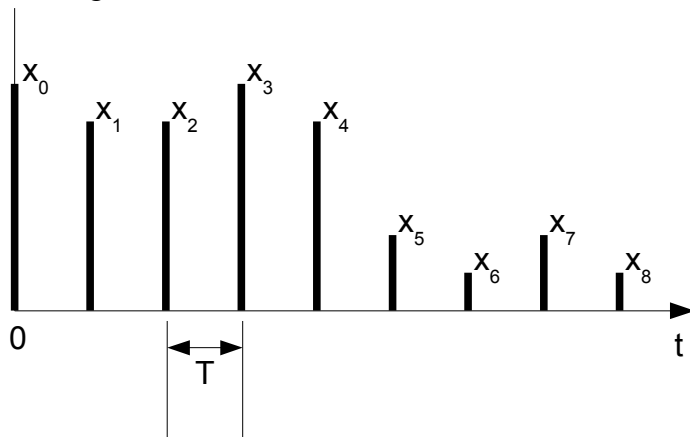
NOTE

The Signal processing toolbox provides a number of functions to provide the coefficients required to implement various filters. See help for **butter**, **cheby1**, **cheby2** and **besself**.

Also **freqs(B,A)** will plot the frequency response of a system. Where B is a vector of the numerator coefficients and A is a vector of the denominator coefficients.

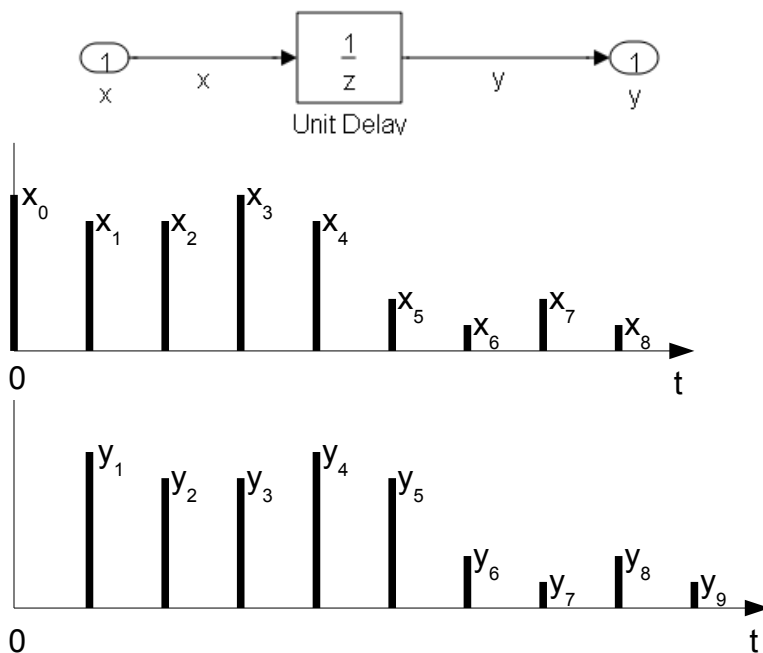
Modeling Discrete Systems

A discrete signal is a signal that has values only at discrete points in time. A sampled signal is a discrete signal.



The sample period is the time between two successive samples. It is normally denoted by T . The sample frequency(f_s) is $1/T$.

The fundamental component of a discrete system is a Unit delay. This delays the signal by one time period.



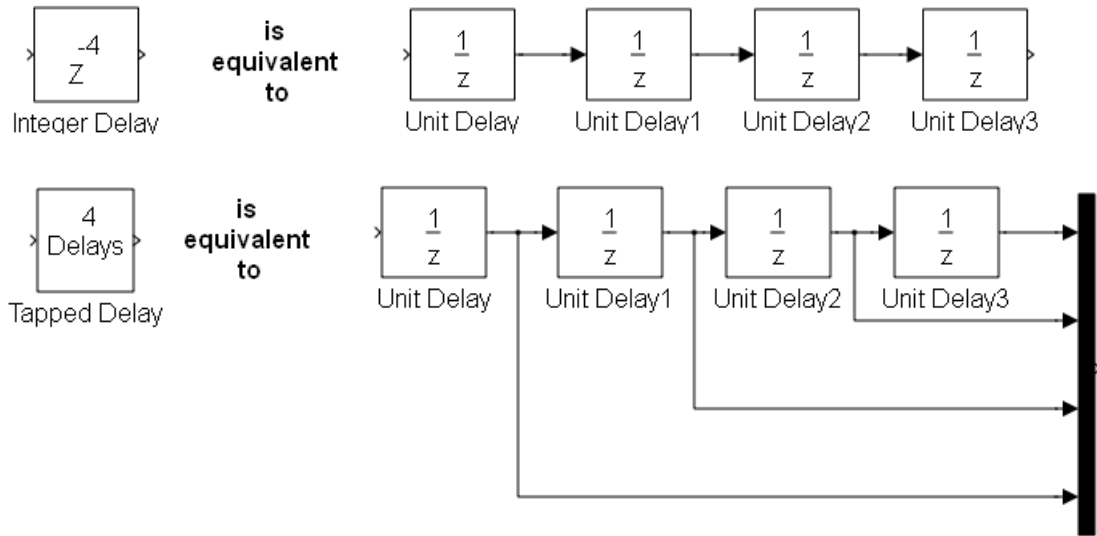
In general, $y_n = x_{n-1}$

The Z transform replaces each delay by one sample with a multiplication by z^{-1} .

$$Y(z) = z^{-1} X(z)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{z}$$

There are other blocks in the library that contain combinations of unit delays.



For each of the blocks above, a block parameter defines the number of unit delays.

Discrete Transfer Functions

Continuous systems are described by differential equations, discrete systems are described by recurrence equation. Equation 8 below is a typical recurrence equation.

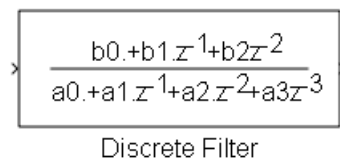
$$a_0 y_n + a_1 y_{n-1} + a_2 y_{n-2} + a_3 y_{n-3} = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} \quad (8)$$

Where x is the input and y the output. Each unit delay is replaced by z^{-1} in the Z transform.

$$(a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}) Y(z) = (b_0 + b_1 z^{-1} + b_2 z^{-2}) X(z)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}} \quad (9)$$

Which is the transfer function of a digital filter. This can be represented in a Simulink model by the discrete filter block.



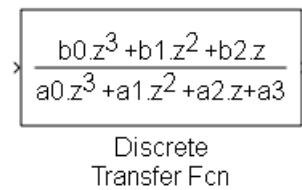
The above block was produced by setting the **numerator** to $[b_0 \ b_1 \ b_2]$ and the **denominator** to $[a_0 \ a_1 \ a_2 \ a_3]$ in the block parameters.

You will also need to set the sample time. The default sample time is one second. Set this to your sample period. Unfortunately, inherited sample time does not work. Many of the blocks in the discrete library are like this. If you find that your model is sampling at one second, regardless of the solver settings, then check the sample time of your blocks.

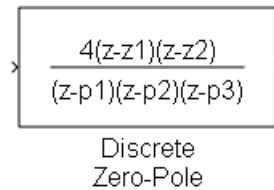
Equation 9 is defined in terms of z^{-1} . An alternative form is to write the transfer function in terms of z . If we multiply top and bottom of equation 9 by z^3 we get :-

$$H(z) = \frac{b_0 z^3 + b_1 z^2 + b_2 z}{a_0 z^3 + a_1 z^2 + a_2 z + a_3} \quad (10)$$

You can represent this with the Transfer Function block.



If you have the poles and zeros, an alternative is to use the Zero-Pole block.



Finite Impulse Response (FIR) digital filters do not have any poles. Their recurrence equation looks like this.

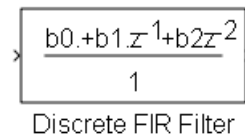
$$y_n = b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} \quad (11)$$

Which gives the following Z transform.

$$Y(z) = (b_0 + b_1 z^{-1} + b_2 z^{-2}) X(z)$$

$$H(z) = \frac{Y(z)}{X(z)} = b_0 + b_1 z^{-1} + b_2 z^{-2} \quad (12)$$

There is a block especially for FIR filters.



Simulink Online Documentation

The full Simulink documentation is available from the help menu. You can obtain this from the Matlab help, or you can go directly to the Simulink help. From the model menu bar select

Help ► Simulink ► Simulink Help

Block Documentation

The easiest way of obtaining the documentation for a particular block is to hit the help button in the block parameters. An alternative is to select the block and then select

Help ► Simulink ► Blocks & Blocksets Reference

from the model menu bar. If no block is selected when you do this, then you will be given an list of all the blocks. You can then select the documentation you want from this list. At the top of the list, on the right hand side you can choose to display by Category or in Alphabetic order.

Model Editing

This section contains the short cut keys that can be used to build and edit your model.
For more details select from the model menu bar

Help ► Keyboard Shortcuts

Object Selection

Task	Action
Select an object	Left Click
Add object to selection	Shift + Left Click
Select Multiple objects	Drag Box over objects
Move selected	Arrow keys
Delete selected	Delete key

Model Viewing

Task	Action
Zoom	Mouse Wheel
Pan	Press Mouse Wheel down and drag
Zoom In	Ctrl +
Zoom out	Ctrl -
Zoom to 100%	Alt 1
Fit to window	Space

Block Manipulation

Task	Action
Move (with Connections)	Drag with Left mouse button
Duplicate block	Drag with Right mouse button
Rotate selected blocks	Ctrl + r
Flip selected blocks	Ctrl + i

Wiring

Task	Action
Add a Wire	Drag from Input , Output or Wire End
Connect Two Blocks	Select Source block Ctrl + Select Destination block
Move a Segment	Drag with Left mouse button
Move a Point	Drag with Left mouse button
Add a Branch Line	Drag with Right mouse button

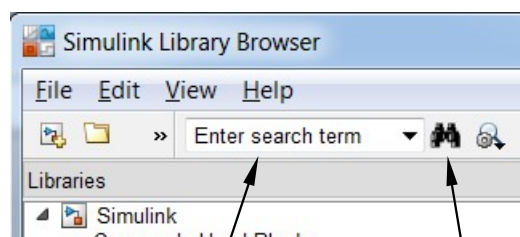
Labels and Annotation

Task	Action
Label a Signal	Double click on Wire
Add Annotation	Double click on White Space

Block Libraries

Block Library	Description
Source	Generators and Inputs
Sinks	Outputs
Math Operations	Mathematical Operators and Functions
User Defined Functions	User Defined Functions
Lookup Tables	Interpolation of tables
Signal Routing	Mux,Bus and Switches
Signal Attributes	Signal Types and Conversion
Ports and Subsystems	Subsystems
Continuous	Systems with Continuous States
Discrete	Systems with Discrete States
Discontinuities	Breaks, Gaps and Limits
Logic and Bit Operations	Boolean Operators and Functions
Commonly Used Blocks	Collection from all above

If you don't know which library a block is in, you can search the libraries by typing a part of the name of the block in white space on the Library Browser banner and hitting the binoculars icon.



Enter search
text here

Hit this icon
to search
for block