SIMULINK® TUTORIAL

The Intelligent Structures and Systems Laboratory
Department of Mechanical Engineering
The Ohio-State University
Columbus OH 43210.

Prepared by Gregory Washington and Arun Rajagopalan
Spring 2002.

TABLE OF CONTENTS

LIST OF FIGURES

Introduction: Concept of Dynamic System Simulation

Computers have provided engineers with immense mathematical powers, which can be used to simulate (or mimic) dynamic systems without the actual physical system. Simulation of Dynamic Systems has been proven to be immensely useful when it comes to system modeling and control design. This because it saves the time and money that would otherwise be spent in prototyping a physical system. Simulink is a software add-on to MATLAB® which is a mathematical tool developed by The Mathworks, (http://www.mathworks.com) a company based in Natick, MA. MATLAB is powered by extensive numerical analysis capability. Simulink® is a tool used to visually program a dynamic system (those governed by Ordinary Differential equations) and look at results. Any logic circuit, or a control system for a dynamic system can be built by using standard ***BUILDING BLOCKS*** available in Simulink Libraries. Various toolboxes for different techniques, such as Fuzzy Logic, Neural Networks, DSP, Statistics etc. are available with Simulink, which enhance the processing power of the tool. The main advantage is the availability of templates / building blocks, which avoid the necessity of typing code for various mathematical processes.

Concept of signal and logic flow

In Simulink, data/information from various blocks are sent to another block by lines connecting the relevant blocks. Signals can be ***generated*** and fed into blocks (dynamic / static). Data can be fed into functions. Data can then be dumped into ***sinks,*** which could be virtual oscilloscopes, displays or could be saved to a file. Data can be connected from one block to another, can be branched, multiplexed etc. In simulation, data is processed and transferred only at ***discrete*** times, since all computers are discrete systems. Thus, a SIMULATION time step (otherwise called an INTEGRATION time step) is essential, and the selection of that step is determined by the fastest dynamics in the simulated system. In the following sections, the different blocks that are available are explained. Figure 1 shows the overview of the Simulink libraries available. More toolboxes may be available based on what has been purchased. The latest version is Simulink 4.0, which is used with MATLAB 6.1 (Release 12.1).

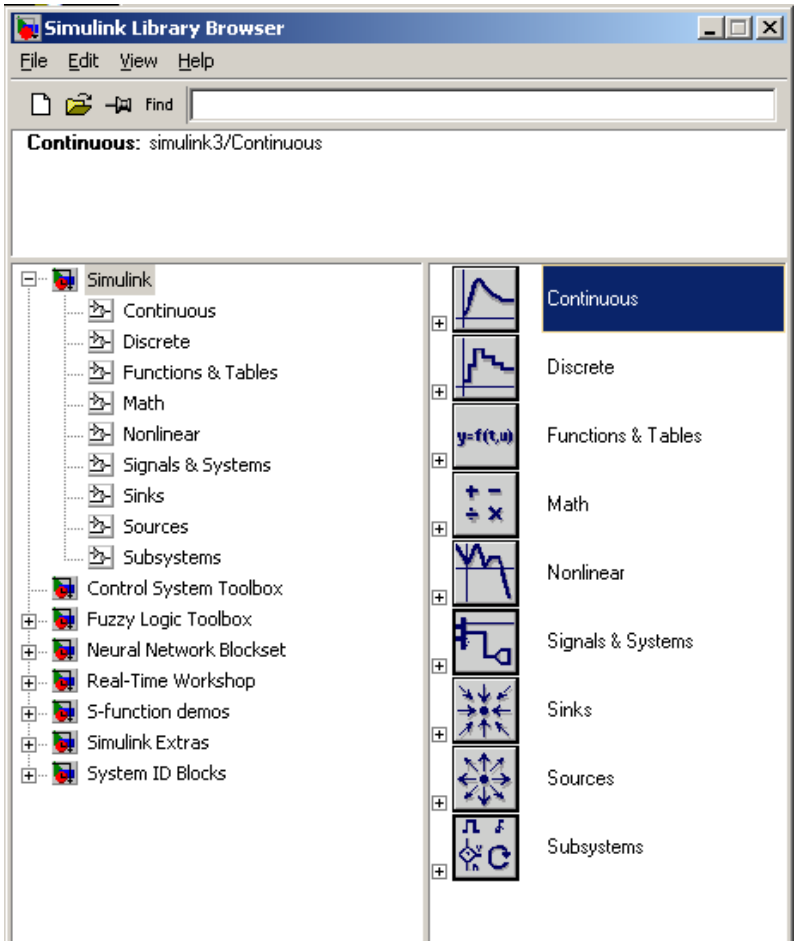

Figure 1: Simulink Library

Connecting blocks

To connect blocks, *left-click* and drag the mouse from the output of one block to the input of another block. Figure 2 shows the steps involved. Tips for branches and quick connections are provided at the end of this document.

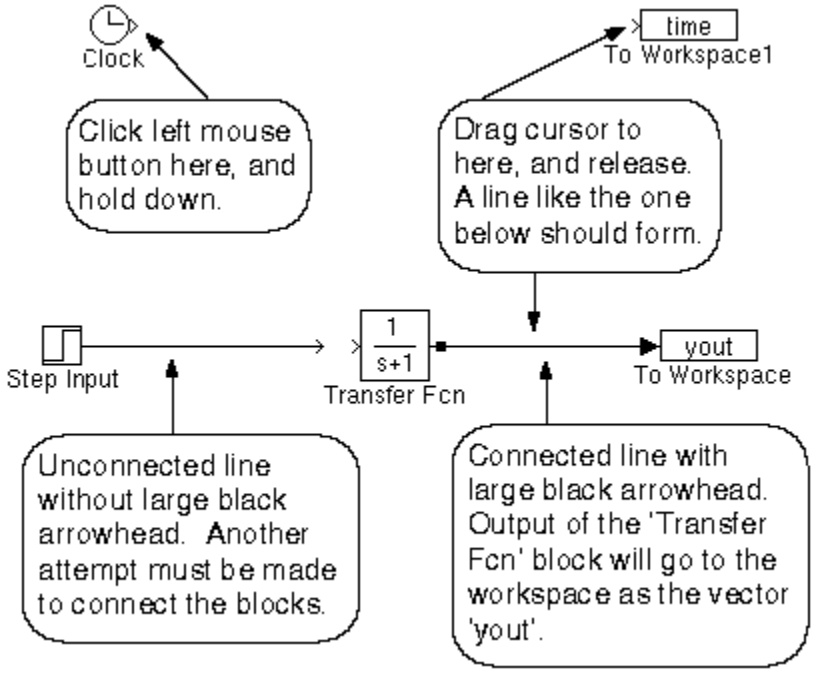

Figure 2: Connecting blocks

Sources and Sinks

The ***sources*** library contains the sources of data/signals that one would use in a dynamic system simulation. One may want to use a *constant* input, a *sinusoidal* wave, a step, a repeating sequence such as a *pulse train*, a *ramp* etc. One may want to test *disturbance* effects, and can use the random signal generator to simulate *noise*. The *clock* may be used to create a time index for plotting purposes. The *ground* could be used to connect to any unused port, to avoid warning messages indicating unconnected ports.

The ***sinks*** are blocks where signals are terminated or ultimately used. In most cases, we would want to store the resulting data in a file, or a matrix of variables. The data could be displayed or even stored to a file. The STOP block could be used to stop the simulation if the input to that block (the signal being sunk) is non-zero. Figure 3 shows the available blocks in the sources and sinks libraries. Unused signals must be terminated, to prevent warnings about unconnected signals.

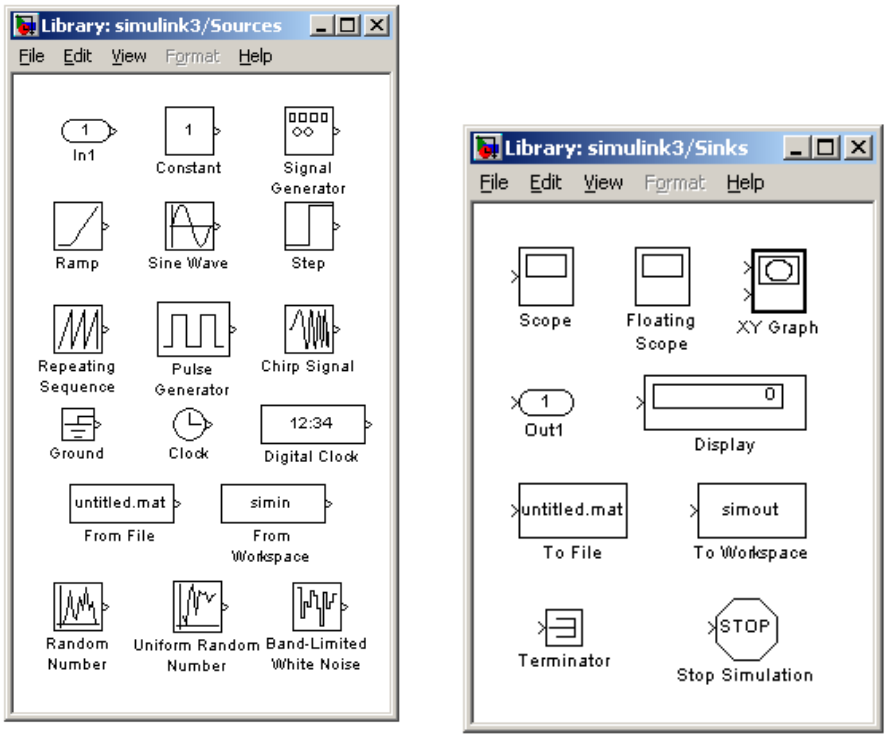

Figure 3: Sources and Sinks

Continuous and Discrete Systems

All dynamic systems can be analyzed as continuous or discrete time systems. Simulink allows you to represent these systems using transfer functions, integration blocks, delay blocks etc.

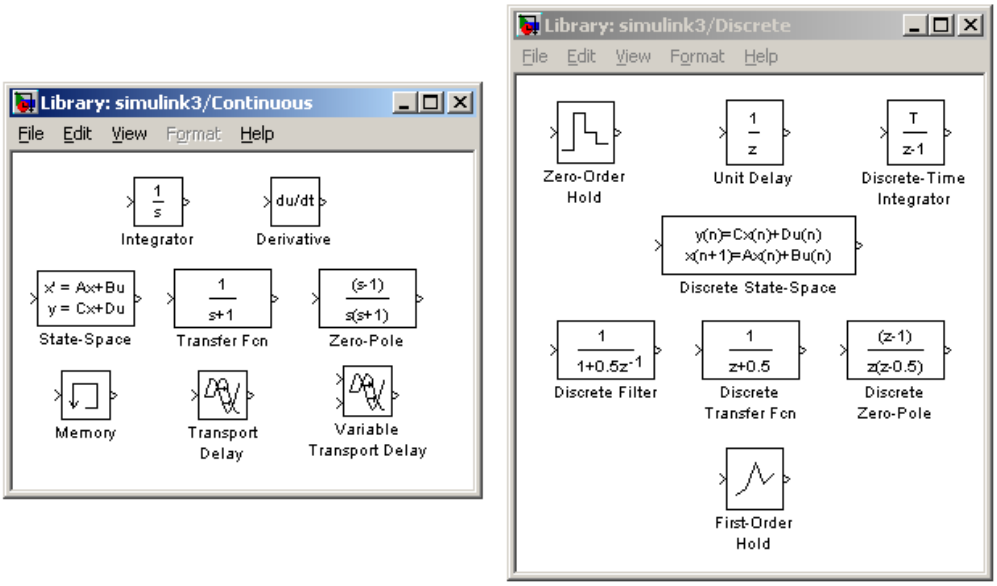

Figure 4: Continuous and Discrete Systems

Figure 4 shows the available dynamic systems blocks. Discrete systems could be designed in the Z-plane, representing difference equations. Systems could be represented in State-space forms, which are useful in Modern Control System design.

Figure 5 contains some advanced linear blocks, available in the "***Simulink Extras***" library. They contain certain advanced blocks, such as a PID control block, transfer functions with initial conditions, etc.

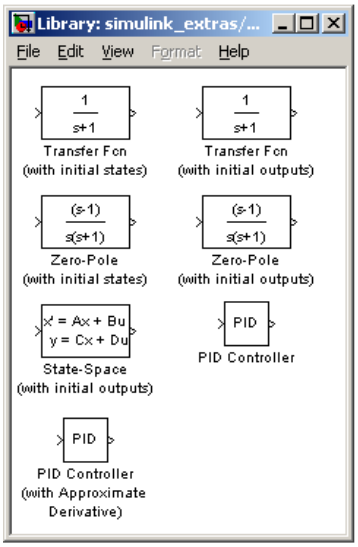

Figure 5: Advanced Linear Systems

EXAMPLE of a dynamic system: A mass-spring-damper system

The following section contains an example for building a *mass-spring-damper* system. The system can be built using two techniques: a state space representation, used in modern control theory, and one using conventional transfer functions. The mass-spring-damper system is a *second order* system, which is commonly encountered in system dynamics. Electrical *Resistance-Inductance-Capacitance* (RLC) circuits are also analogous to this example, and can be modeled as 2nd order systems.

The example is shown in Figure 6. A step input is used as the control input. (It is an open loop example). The top portion of the block contains the transfer function representation of the dynamic system. We can observe only the outputs, and cannot monitor the states. Also, initial conditions cannot be specified. (By using the special transfer function block in the Simulink\Extras toolbox, initial conditions can be specified). The bottom portion of the Simulink diagram shows the same 2nd order system in state space representation. The highest derivative (acceleration in our case) is represented as a function of the input and the other states. This input is integrated to form the next lower state. Initial conditions for each state can be specified in the integration block. States can be individually monitored and manipulated.

Consider a mass-spring damper with the following dynamic equation:

$$m\ddot{x} + c\dot{x} + k_s x = f \qquad \textbf{(1)}$$

where

x	Output variable
m	Mass
c	Damping coefficient
k	Spring stiffness
f	Control force (multiplied by a constant q_i)

Equation (1) can be represented in Laplace domain (as a transfer function) as follows:

$$\frac{X(s)}{F(s)} = \frac{K\omega_N{}^2}{s^2 + 2\zeta\omega_N s + \omega_N{}^2} \qquad \textbf{(2)}$$

where

ζ Damping coefficient $\zeta = \dfrac{c}{2\sqrt{k \cdot m}}$

ω_N Natural frequency $\omega_n = \sqrt{\dfrac{k}{m}}$

K Steady State gain (or Static sensitivity) $K = \dfrac{1}{k_s}$

In the state formulation the system is represented in terms of it's highest derivative:

From (1) $$m\ddot{x} = (f - c\dot{x} - k_s x) \quad \rightarrow \quad \ddot{x} = \frac{1}{m}(f - c\dot{x} - k_s x) \qquad \textbf{(3)}$$

or it can also be written in terms of it's damping and natural frequency as (with $q_i = 1$):

$$\ddot{x} = K\omega_N^2 f - 2\zeta\omega_N \dot{x} - \omega_N^2 x \qquad \textbf{(4)}$$

In our example below, with zero initial conditions, both the transfer function and the state representations provide similar results. In general both diagrams are NOT necessary. The **steps** for the state formulation are as follows:

1. Solve the differential equation in question for the highest derivative. If the equation is not normalized (as in the first of equation 3) the highest derivative may be multiplied by a term. You can divide all the values by that term as was done in the second part of equation 3. You should now have your single term with the highest derivative on the left side and the rest of the terms on the right side of the equation.
2. Draw a summer block. The block should have as many plusses and minuses as there are terms in the right side of the equation (in equation (3) we have 3 components and two of them are negative, thus we add 2 minus sings and 1 plus sign to our summer). The output of the summing block should equal the highest derivative term multiplied by a constant. You can now multiply or divide the constant out to get the derivative by itself.
3. Add integrators. The total number of integrators should equal the total number of derivatives that you want to remove. For example, if you have a second order mechanical system (like the one in equation 3) and you want position, you need to integrate twice. Put a block at the end for the output variable.
4. After each integrator, feed the signal back to its proper place on the summer. Immediately to the right of an integrator is a value equal to the integral of the value on the left. Be sure to use a gain block to multiply any value by its proper constant before feeding the value back.

Notice in the state formulation example that the lower derivatives (or states) are accessible (Internal Variables). This accessibility makes the state formulation a better methodology for dynamic systems classes. In addition, it is easier to adapt the system to nonlinear components. The transfer function methodology is simpler (only one block), but it is limited in is application.

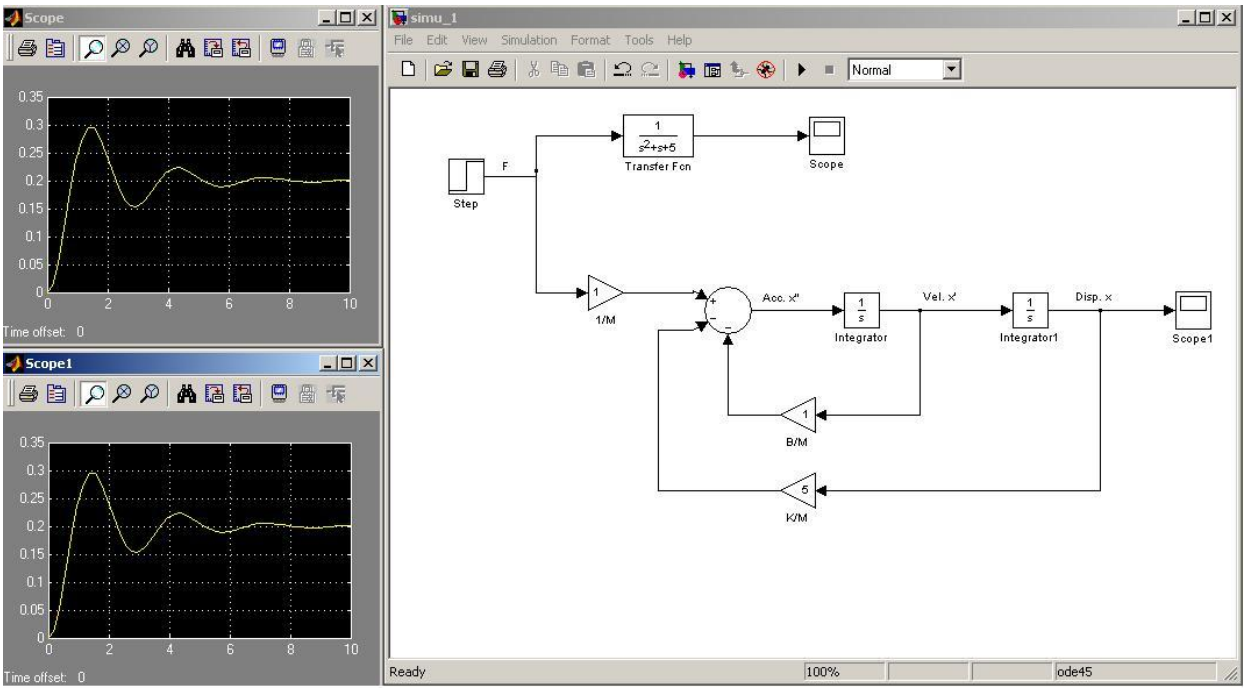

Figure 6: A mass-spring-damper system – an example of a 2nd order dynamic system.

Note that figure 6 does not show the spring and damper forces. This is rectified in figure 7. Note neither is incorrect, but figure 7 provides the engineer with more useful information. Figure 7 is the figure that is produced when one uses the first part of equation (3). Figure 6 is produced when one uses the second part of equation (3).

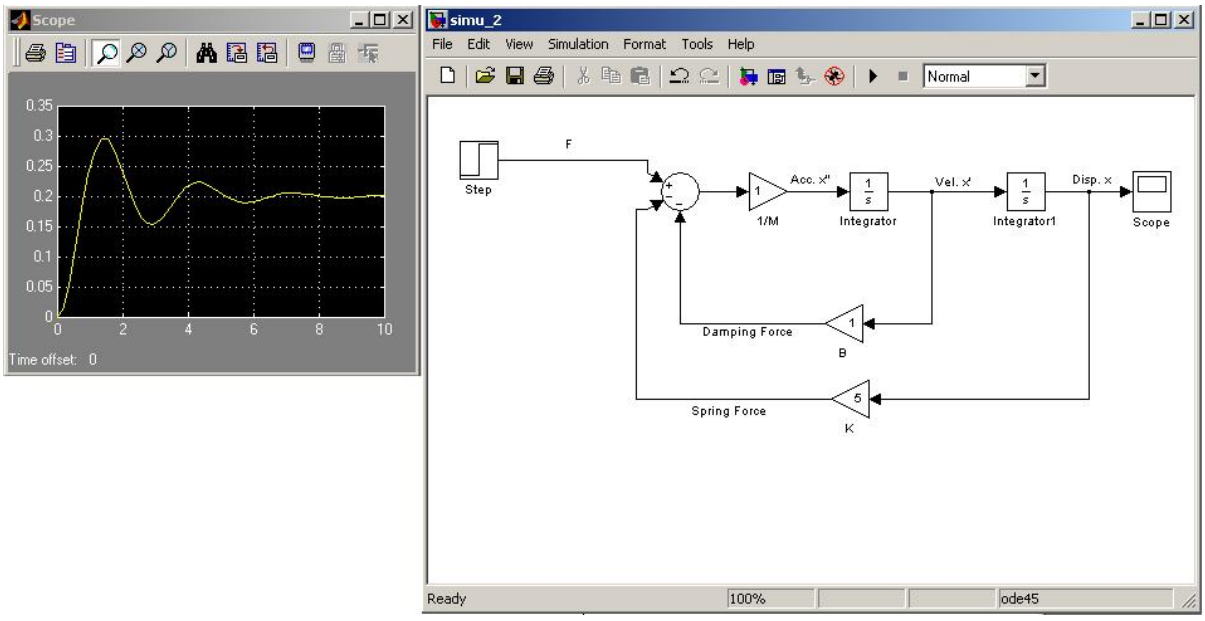

Figure 7: A mass-spring-damper system showing the spring and the damper forces.

Non-linear operators

A main advantage of using tools such as Simulink is the ability to simulate non-linear systems and arrive at results without having to solve analytically. It is virtually impossible to arrive at an analytical solution for a system having non-linearities such as saturation, signum function, limited slew rates etc. In simulation, systems are analyzed by numerical differentiation thus non-linearities are not a hindrance. Figure 8 shows some of the non-linear components that can be incorporated into a simulation. One such could be a *saturation* block, to indicate a physical limitation on a parameter, such as a voltage signal to a motor etc. *Manual switches* are useful when trying simulations with different cases. *Switches* are the logical equivalent of IF-THEN statements in programming. *Slew rates* using the rate limiter could control the rate of change of a physical parameter, such as the speed of a DC motor, etc.

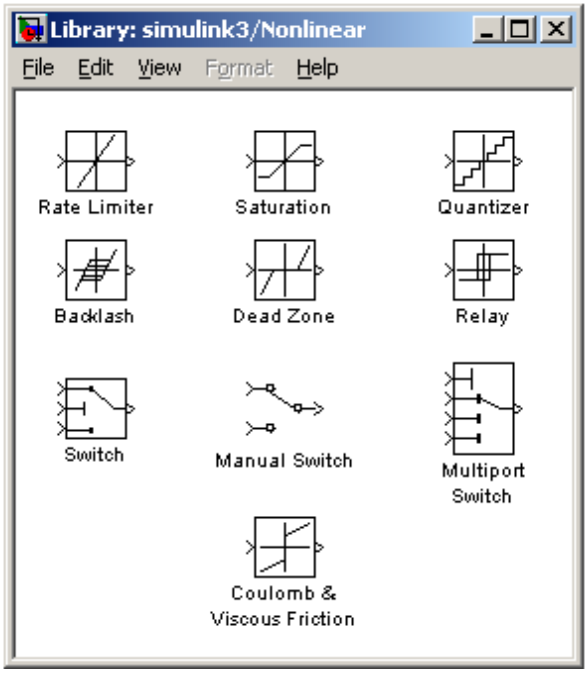

Figure 8: Non-linearities

EXAMPLE:

Here is an example using a non-linear block. Consider a sine wave of amplitude 1 (signal varies between +1 and –1). A ***saturation*** block is used to limit the output to an amplitude of 0.5 and the saturated and unsaturated (original) signals are compared. The example is shown in Figure 9. The saturated and unsaturated signals are clearly seen.

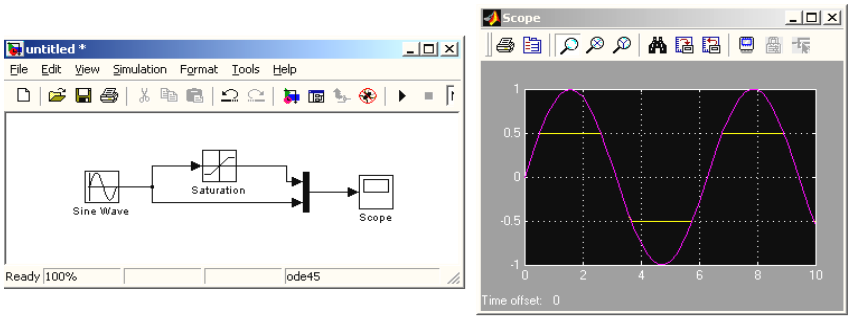

Figure 9: Example of a non-linear function (saturation)

EXAMPLE of a dynamic system: A mass-spring-damper system with Coulomb Friction

A mass-spring-damper system is created with ***Coulomb friction*** for the damper force. The Coulomb friction (from the non-linear library block) is represented as an offset at zero velocity. The offset for our example is given as 0.5 (with a slope of 1). The coding is shown in Figure 10. The output for a combination input = ramp(2t) + step + ramp (5t) is shown in Figure 11. The combination input is available as the ***repeating sequence*** in the sources library block. As expected, the Coulomb Friction creates undesired response in the output of the system.

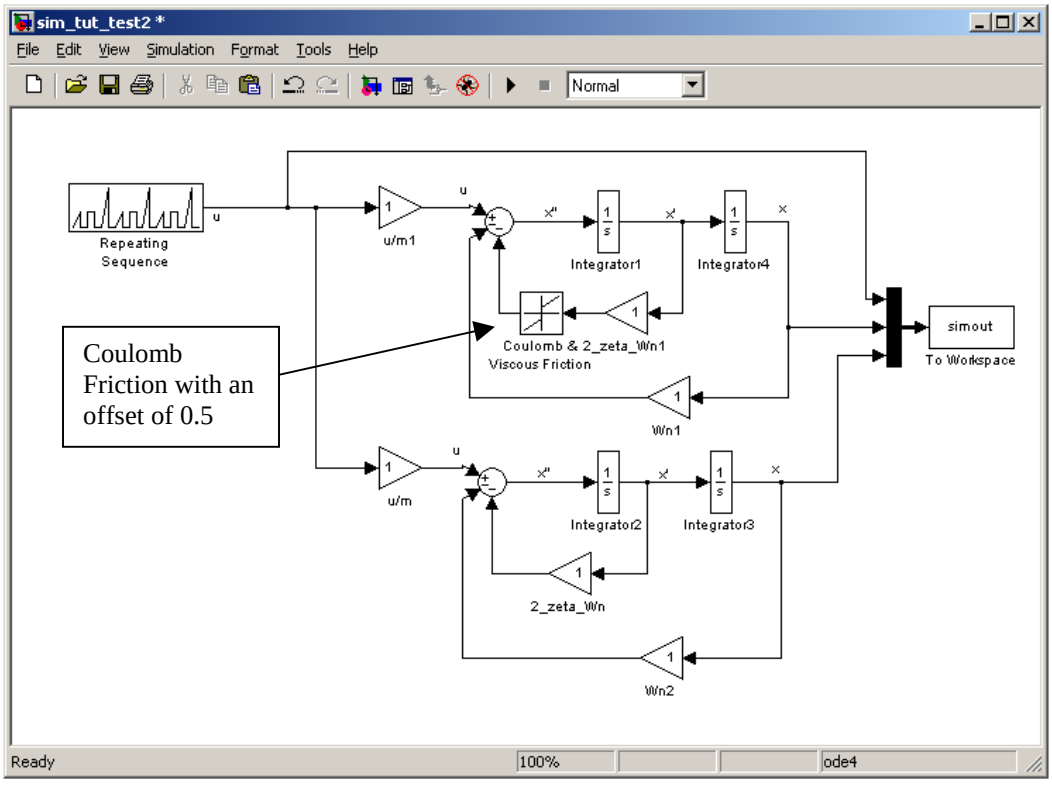

Figure 10: Mass-Spring-Damper system with Coulomb friction

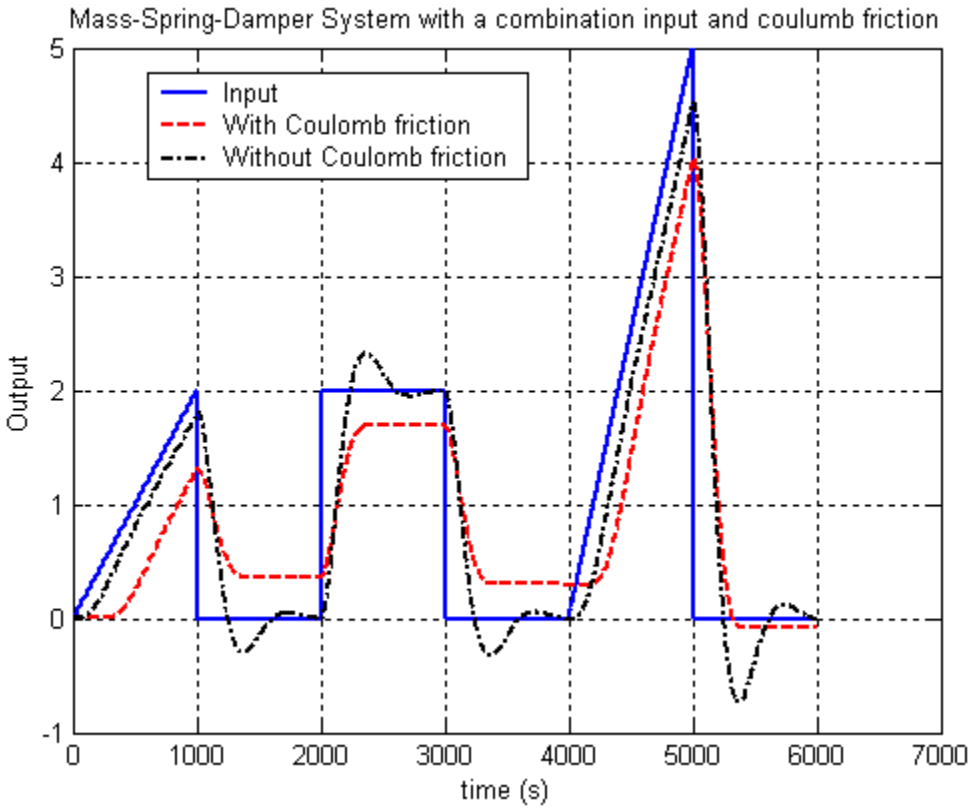

Figure 11: Output of mass-spring-damper system with coulomb friction

Using functions (written as M, C, etc..)

Functions written in M or in any other language such as C or Fortran could be used in conjunction with Simulink to enhance the computing power of Simulink. Custom code, if included, written as an 'M' or 'C' file, are evaluated at every simulation step. S-functions are Dynamic Linked Libraries (DLL) written in another language such as C, and then compiled using the MATLAB compiler 'MEX'. This is useful in large simulations, since a function written in 'C' runs much faster than a comparably programmed M-function. Also, for REAL-TIME simulations, only S-functions can be used, the reason again being high speed of processing.

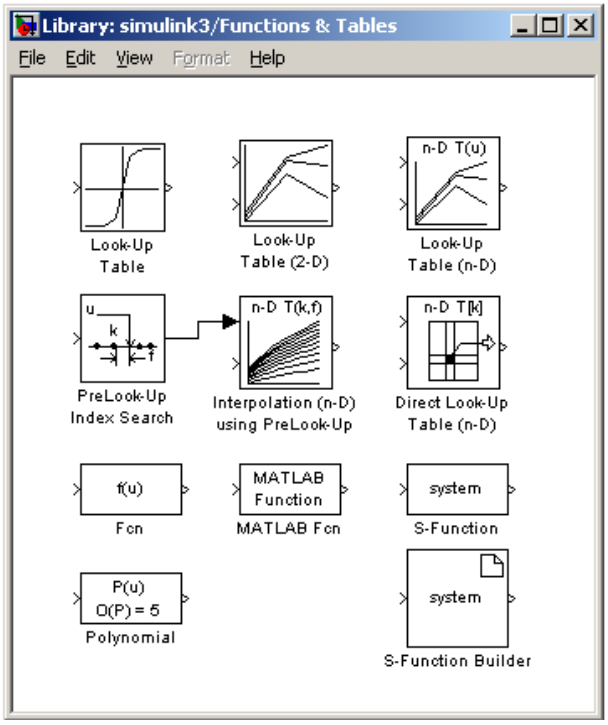

Figure 12: Functions and tables

Look-up tables are very useful in mapping different data points and functions. N-dimensional look-up tables are available. Figure 12 shows the various functions and tables used in Simulink.

EXAMPLE:

Look-up tables are used for producing outputs based on a pattern of inputs. If the pattern is known, then the data could be entered in a Look-up table, and linear interpolation is performed to produce the outputs based on the new set of inputs. Consider the simple example where you want to multiply 2 inputs and get the output. A 2-D look-up table is created in Simulink, and the values for 1, 2 and 3 as inputs are entered in the output block, as seen in Figure 13. The block is used to multiply 2 inputs, and the output is shown as follows (2 * 2.5 = 5):

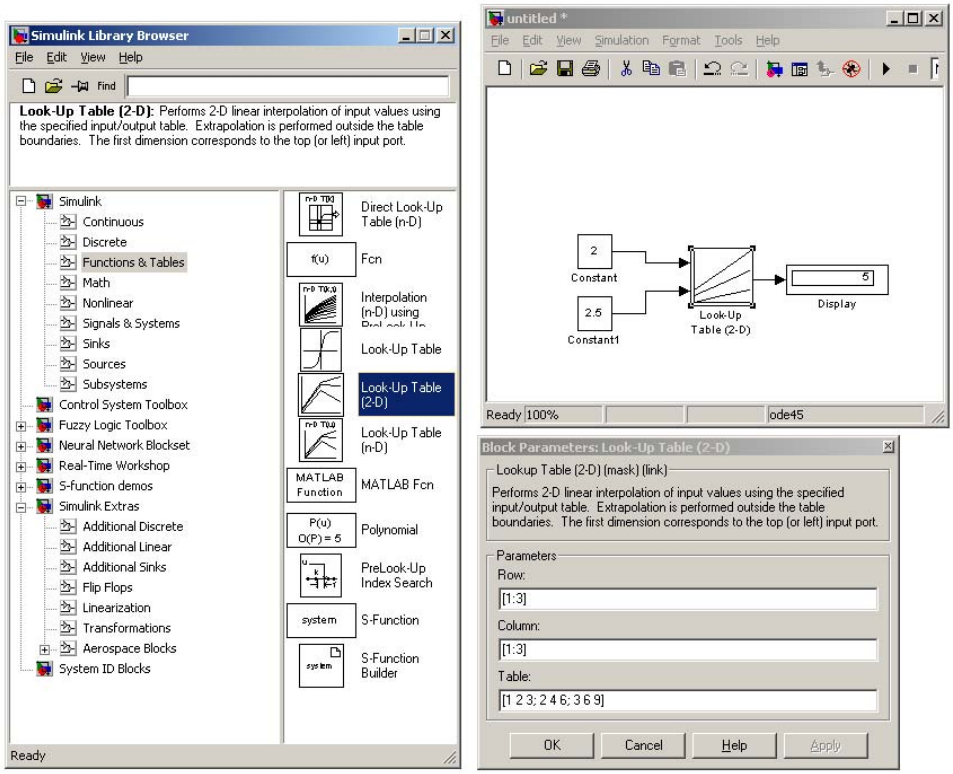

Figure 13: 2-D Look-up table example

The visualization of the 2-D look-up table is shown in Figure 14. Any 2-D surface can be represented as a look-up table if data exists for specific points on the inputs. 1-D and n-D look-up tables are also available in Simulink.

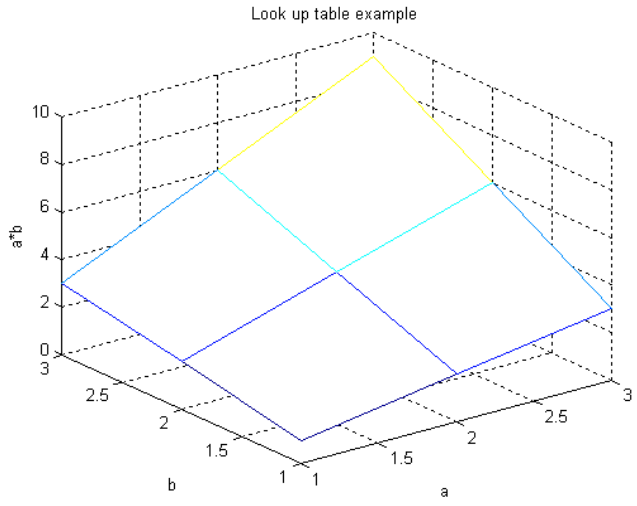

Figure 14: Visualization of the 2-D look-up table

EXAMPLE:

To show a more complete use of the look up table, let us consider a non-linear spring with the spring force defined as, $f = 10^5 x + 3 \times 10^{11} x^3$ attached to a mass that weighs 100 lbf.

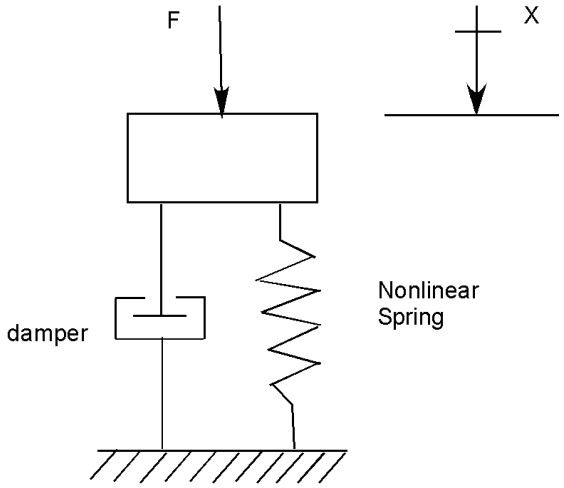

Figure 15: Mass-Spring-Damper System

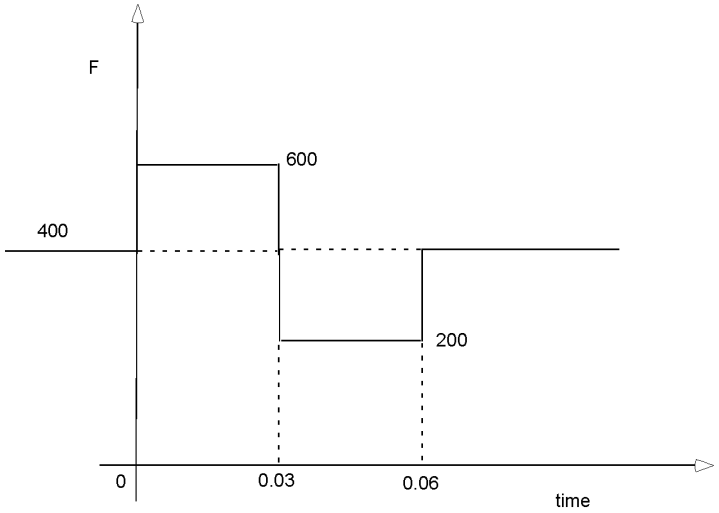

Figure 16: Variation of external force with time.

As an initial condition let us assume that there is an initial deflection of 0.001 inches due to an internal force of 400 pounds. The time variation of the external force is shown in the figure 16. The equation of the exact (nonlinearized) model is given by:

$$M\frac{d^2x}{dt^2} = -(10^5 x + 3\times 10^{11} x^3) - B\frac{dx}{dt} + F \quad \textbf{(5)}$$

To compare the nonlinearized model with the linearized one, let us linearize the spring force by Taylor series at F = 400 and x=0.001 inch. The linearized equation for the force is given by $f \approx 600 - 10^6 x$. Thus the linearized equation for the system is:

$$M\frac{d^2x}{dt^2} = -(10^6 x - 600) - B\frac{dx}{dt} + F \quad \textbf{(6)}$$

Here, the damper constant B is assumed to be 100lbf/(inch/sec).

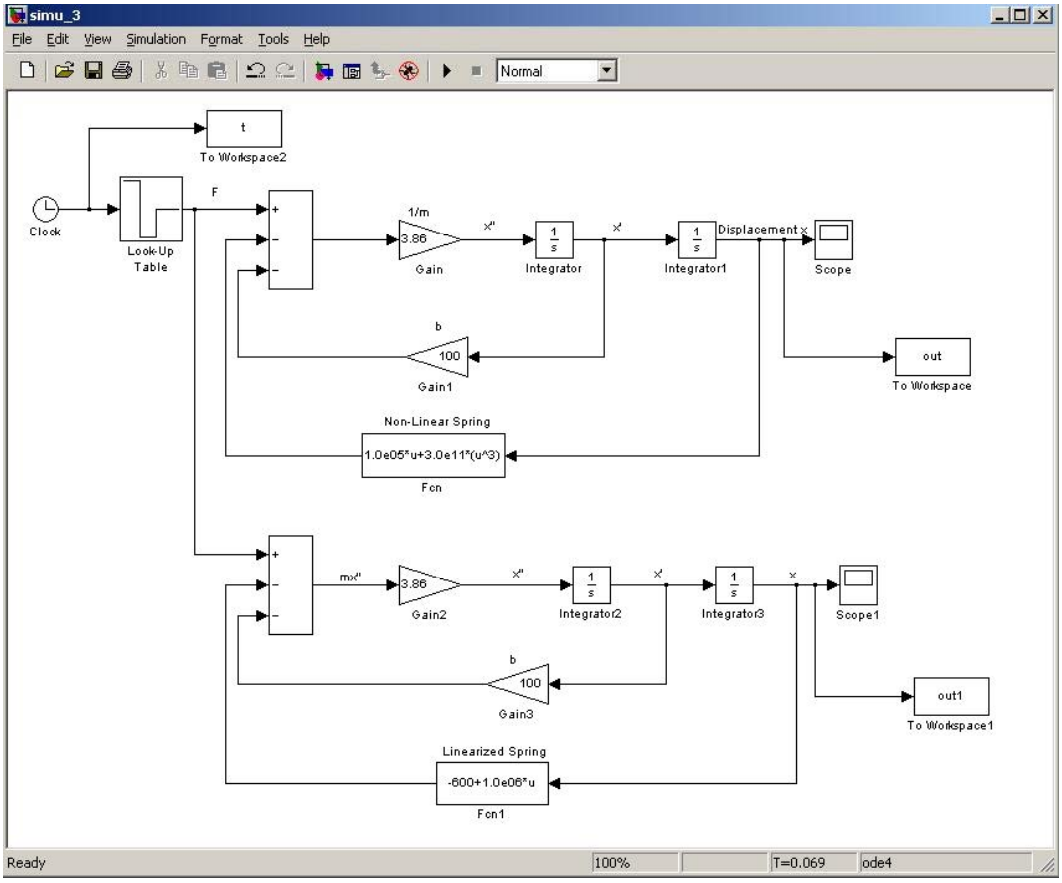

Figure 17: Simulink block diagram with linearized and nonlinearized spring system

Figure 17 is the Simulink block diagram used to solve both the equations in one run. The external force F is created by using the look up table. This force can also be created by using 3 step functions, a constant and a summer.

As always the Simulink model is started by the highest order derivative. Further integration of this derivative gives the velocity and the displacement parameter. So for convenience, arrange the differential equation such that the highest order derivative (2^{nd} in our case) is the only term on the left hand side of the equation. Then start drawing the block diagram using the velocity and position vectors according to the given equation.

Remember that we have a system with an initial deflection of 0.001 inch. So double click the integrator, whose output is the position vector and insert the value of 0.001 as the initial condition. Similarly, make sure that the initial condition of the integrator with the output as the velocity is zero for the present case.
The **Scope** block (see figure 3) is used to see the plots of the position vector x. We can use any number of these scopes to see the variation of any parameter. But to print the graphs it is a better idea to save the values using the **to Workspace** block (see figure 3) and plot it later in the command window. In the current case the values of the displacement vector are saved in the workspace with the variable name of 'out' and 'out1' for nonlinearized and linearized spring respectively.

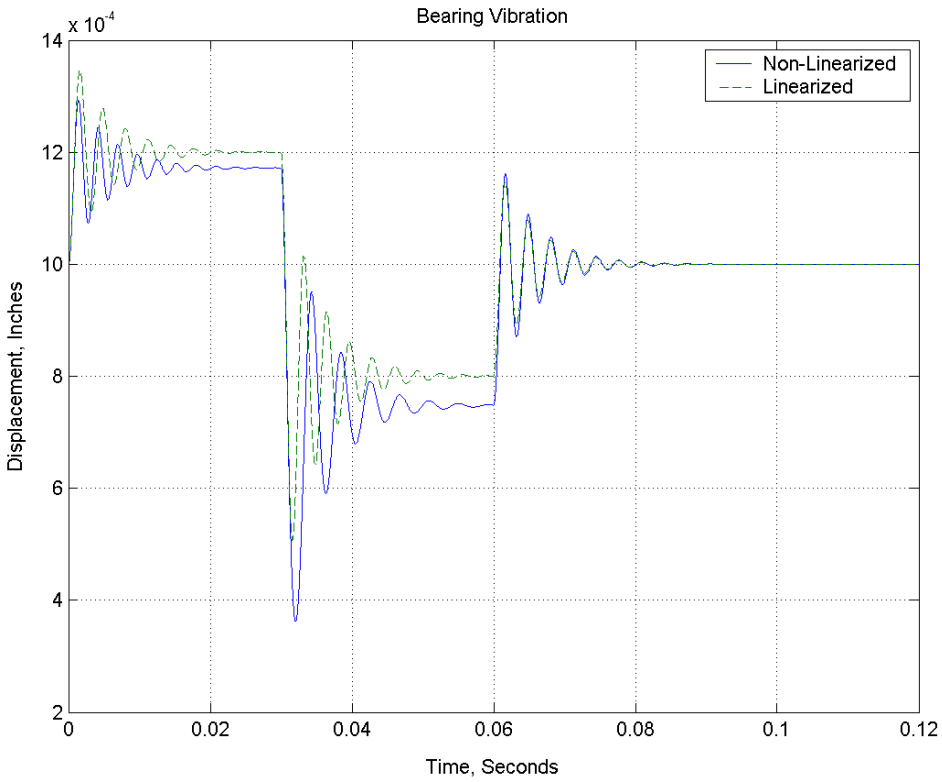

Figure 18: Figure showing the variation of displacement with time for linearized and nonlinearized spring system.

Mathematical operations

Mathematical operators such as products, sum, *logical operations* such as AND, OR, etc. can be programmed along with the signal flow. Matrix multiplication becomes easy with the *matrix gain* block. Trigonometric functions such as *sin* or *tan inverse (atan)* are also available.

Relational operators such as 'equal to', 'greater than' etc. can also be used in logic circuits. Figure 19 depicts the available mathematical tools in Simulink 4.0.

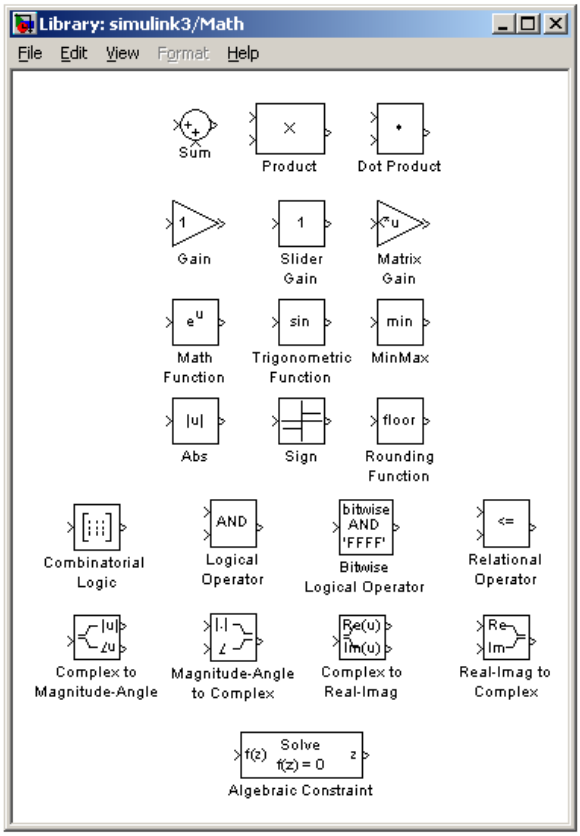

Figure 19: Mathematical tools

Signals & Data Transfer

In complicated block diagrams, there may arise the need to transfer data from one portion to another portion of the block. They may be in different subsystems. That signal could be dumped into a GOTO block, which is used to send signals from one subsystem to another. *Multiplexing* helps us remove clutter due to excessive connectors, and makes matrix (column/row) visualization easier.

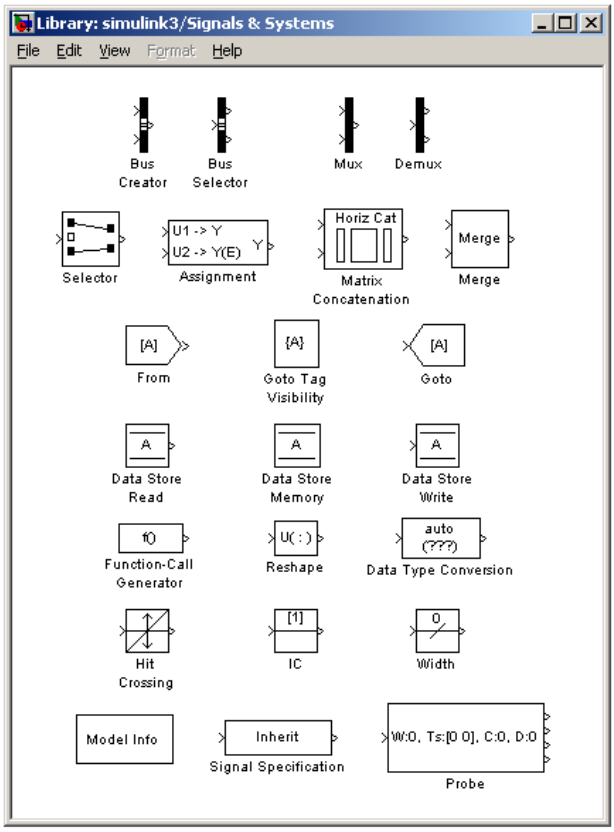

Figure 20: Signals and data transfer

Optimizing Visual appeal

Many times, when a complex Simulink diagram is built, the number of connectors and blocks on a particular level may prevent proper comprehension of the flow of logic. In such cases, one can create a hierarchical flow of blocks using subsystems, which help keep the block diagram simple and comprehendible.

Use of subsystems and masks

Masks are interfaces between the functionality of a subsystem and the user. For example, if there exists an algorithm that the programmer would like to hide from the user, or will be too confusing for the user, the programmer uses a mask and hides the algorithm after placing it in a subsystem.

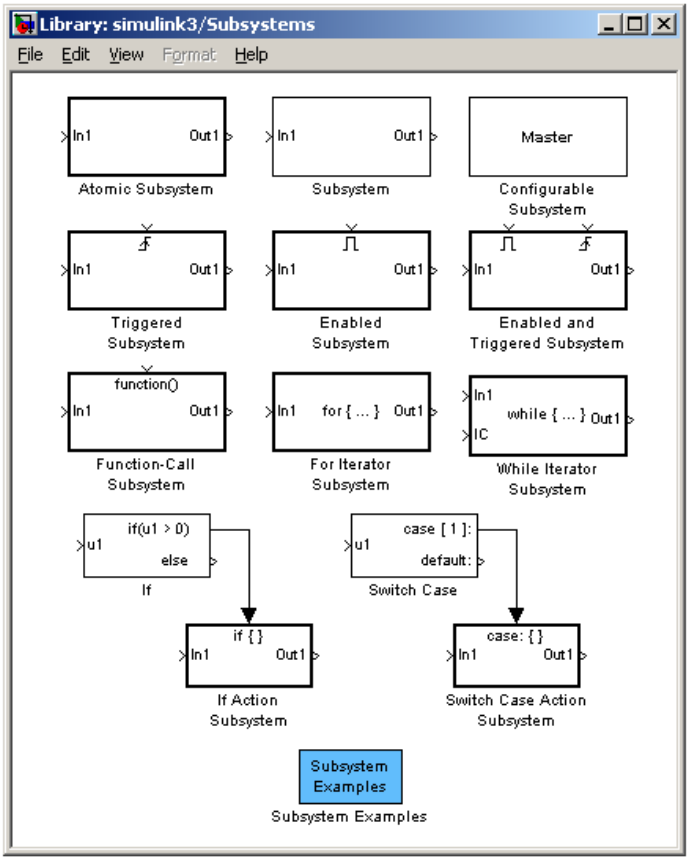

Figure 21: Subsystems

Example: (PID control block in Simulink\Extras)

The 'Simulink Extras' block, contains a PID controller. When double-clicked, it asks the user for the P, I and D gains of the system. The system inside (which can be observed by right-clicking on the block and clicking on 'Look under mask') is shown in Figure 22.

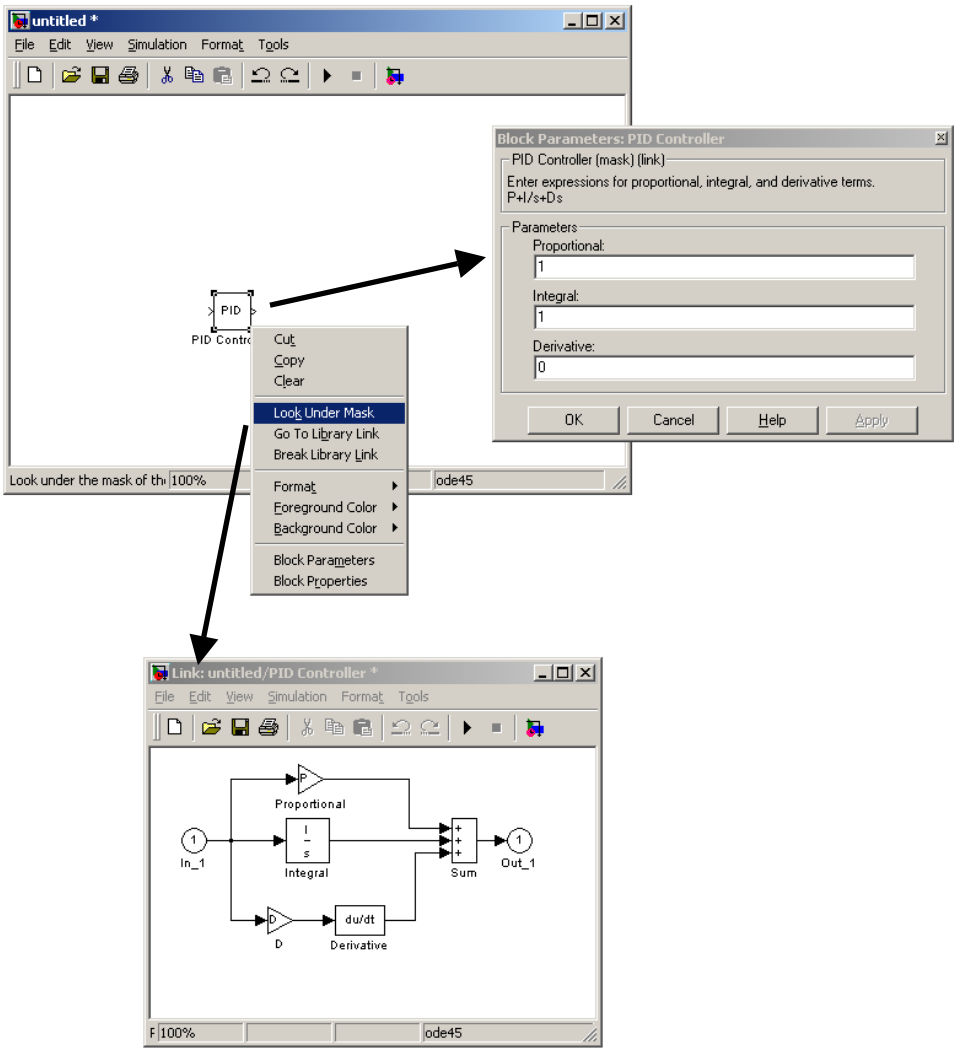

Figure 22: Masking example – PID control block

The following illustrations in Figure 23 show the components of the mask. There are spaces provided for typing help messages, sketching figures on the face of the block, accepting variables and creating prompts, etc.

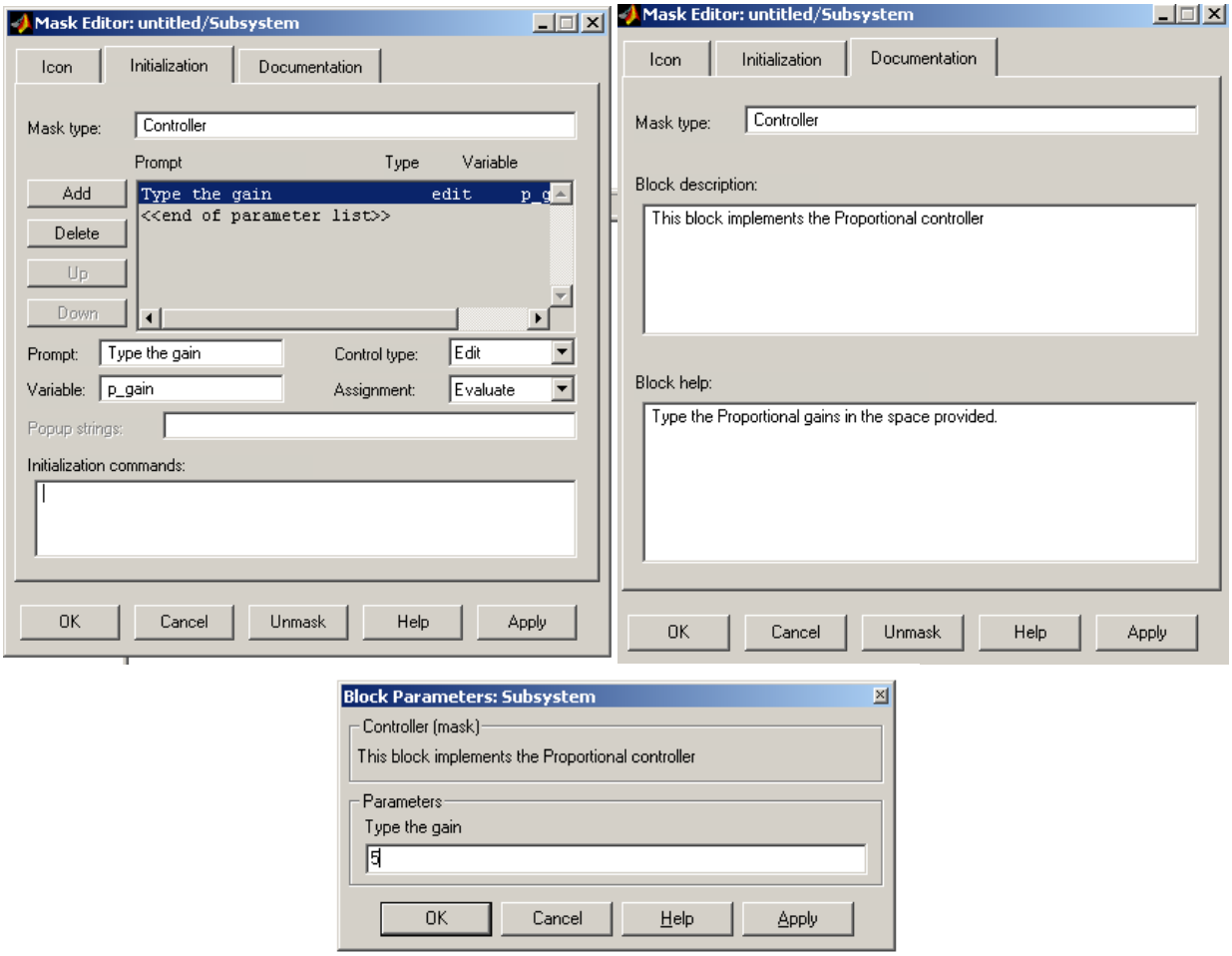

Figure 23: Programming the mask

EXAMPLE: Simplification of the block diagram

In case of complex block diagrams, cluttering of smaller blocks makes the block difficult to understand. In that case, based on functionality, blocks from the main window can be placed inside *sub-systems* and the subsystems make up the main block. Figure 24 shows an example of a *dynamic system* with a *feedback controller* and *actuator dynamics*. The three functional modules are now placed in their respective subsystems.

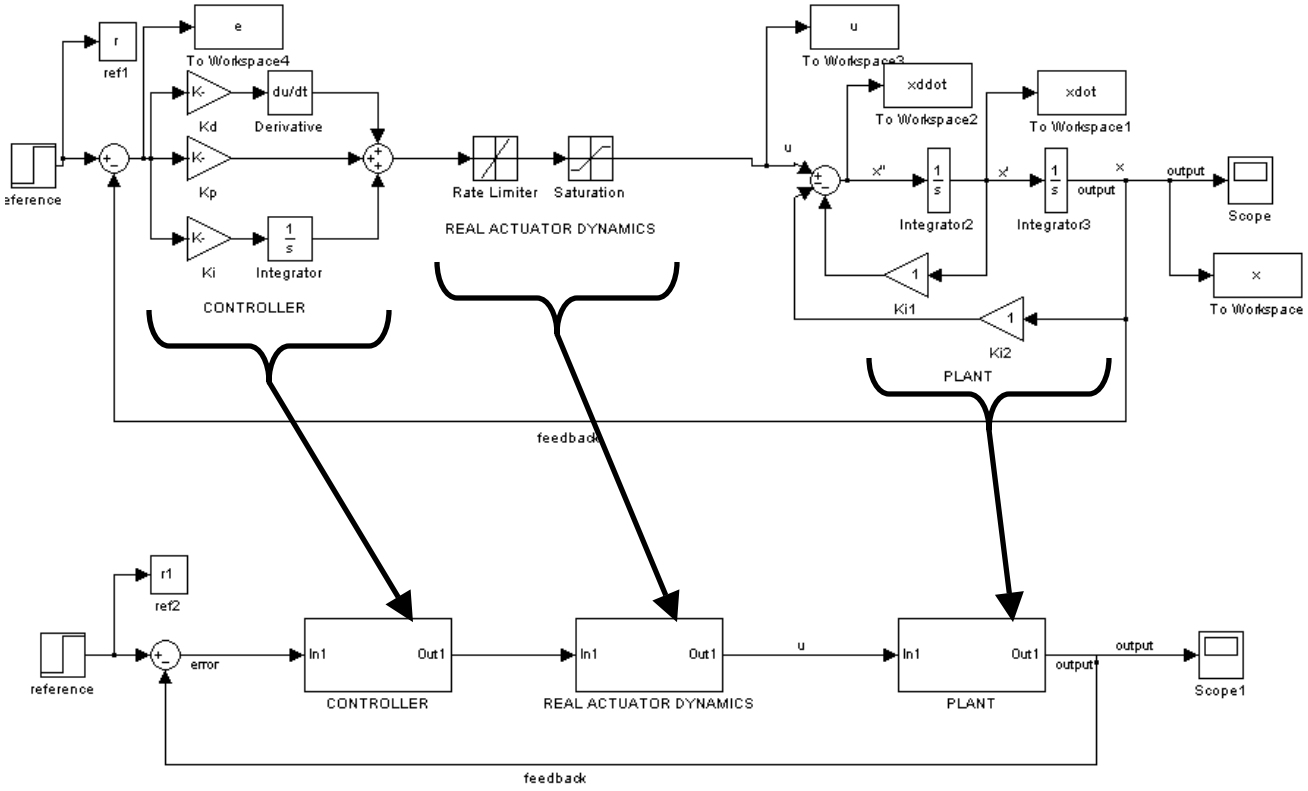

Figure 24: Simplification using subsystems

Making Subsystems

The following is the procedure for making subsystems such as the block in Figure 24.

1. Drag a subsystem from the Simulink Library Browser and place it in the parent block where you would like to hide the code. The type of subsystem depends on the purpose of the block. In general one will use the standard subsystem but other subsystems can be chosen. For instance, the subsystem can be a triggered block, which is enabled only when a trigger signal is received. Figure 25 shows the procedure for creating a subsystem block.

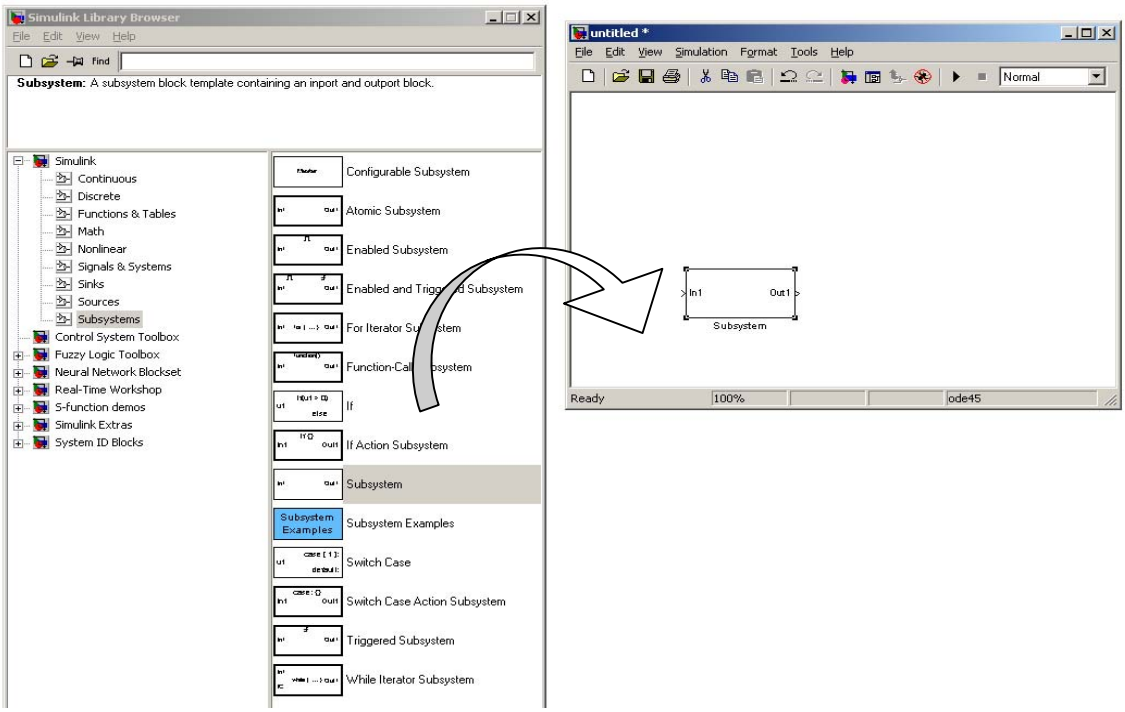

Figure 25: Create a subsystem

2. Open (double click) the subsystem and create input / output ***PORTS,*** which transfer signals into and out of the subsystem. The input and output ports are created by dragging them from the *Sources* and *Sinks* directories respectively. When ports are created in the subsystem, they automatically create ports on the external (parent) block. This allows for connecting the appropriate signals from the parent block to the subsystem. Figure 26 shows the creation of the input / output ports.

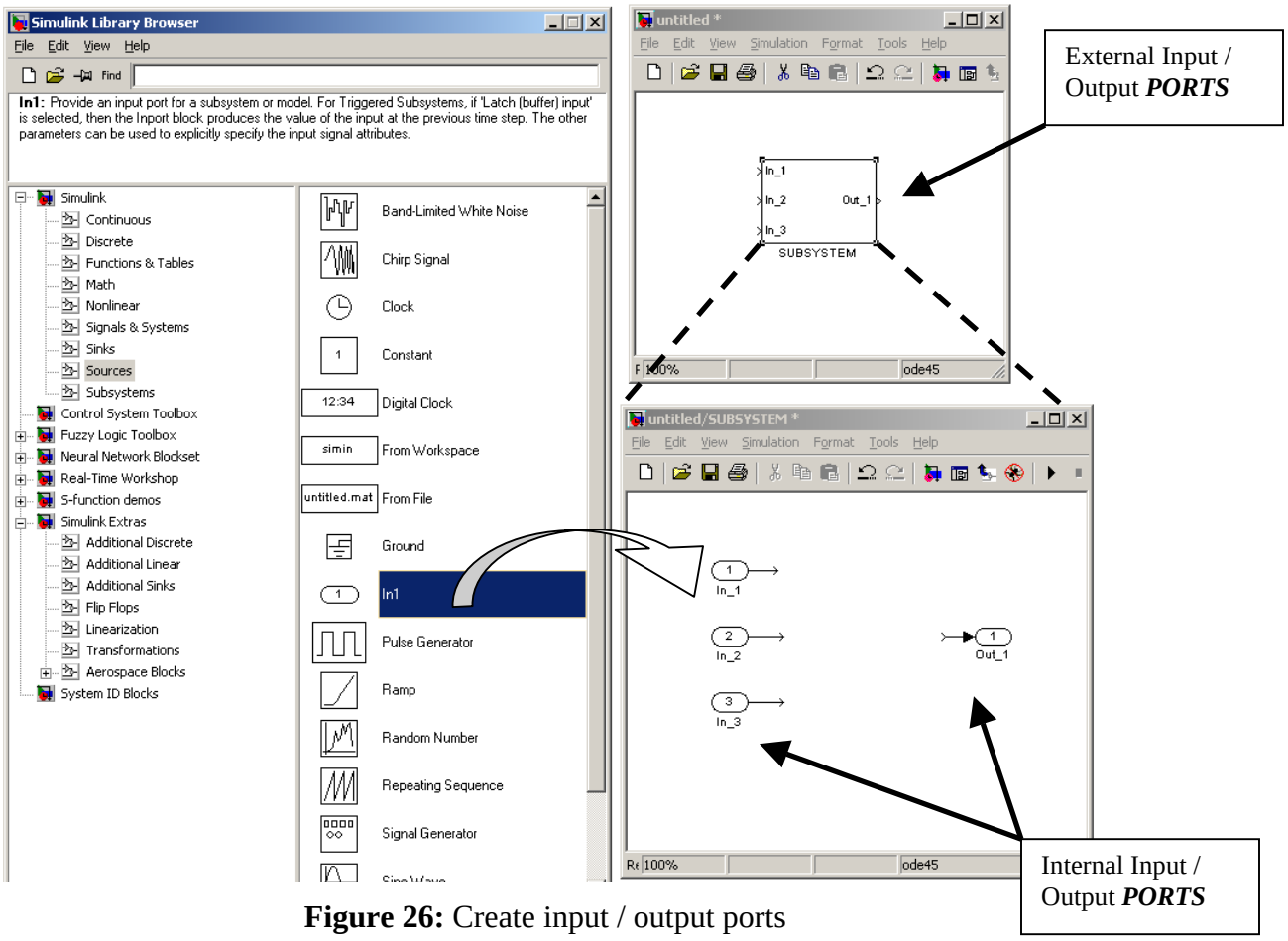

Figure 26: Create input / output ports

3. Once the subsystem is created create blocks or code to be enclosed. This is shown in the bottom part of Figure 27. These blocks contain the code that would be hidden from the parent block and they communicate with the parent block using the Input / output ***PORTS.*** Figure 27 shows how the hidden code uses the input output ports to communicate to the parent block.

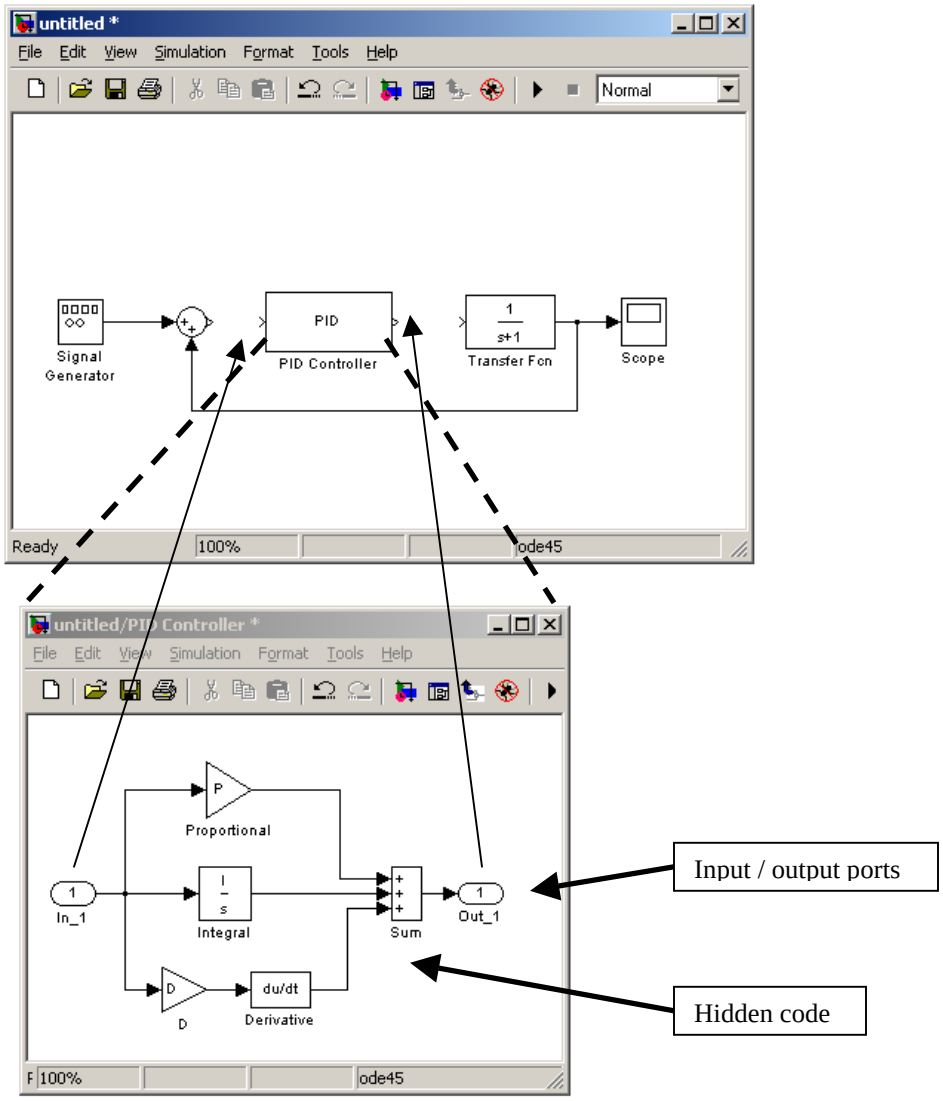

Figure 27: Create hidden code

The subsystem can then be masked if necessary.

Visual aids

The following visual aids can be used to provide more information about the simulated block.

- Sample time colors
  Based on the sampling rate of the system and their individual components, colors are assigned automatically to systems with different sampling rates.

- Signal Type
  Based on the type of signal, whether *double*, *Boolean* etc., signals ca be labeled, that help us identify what each signal represents.

- VECTOR Wide lines and line WIDTH
  The width of lines can be changed based on whether they transmit scalars or vectors. Wider lines represent vectors. The actual width (no. of multiplexed data signals) can also labeled next to the lines.

- Execution order
  Sometimes, it is useful to know the order of execution of the blocks in the Simulink diagram. This command places a number next to the specific block indicating its order of execution.

The commands are shown in Figure 28. A sample of the features is displayed in Figure 29.

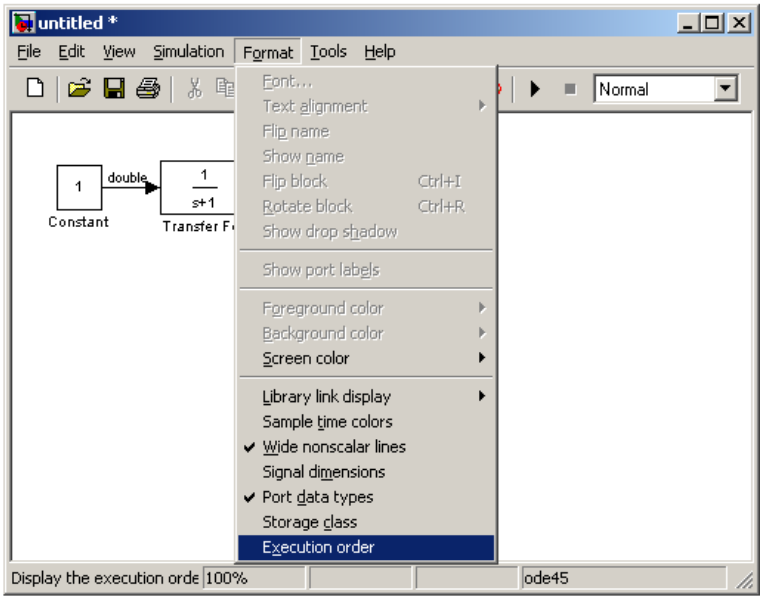

Figure 28: Setting block display features

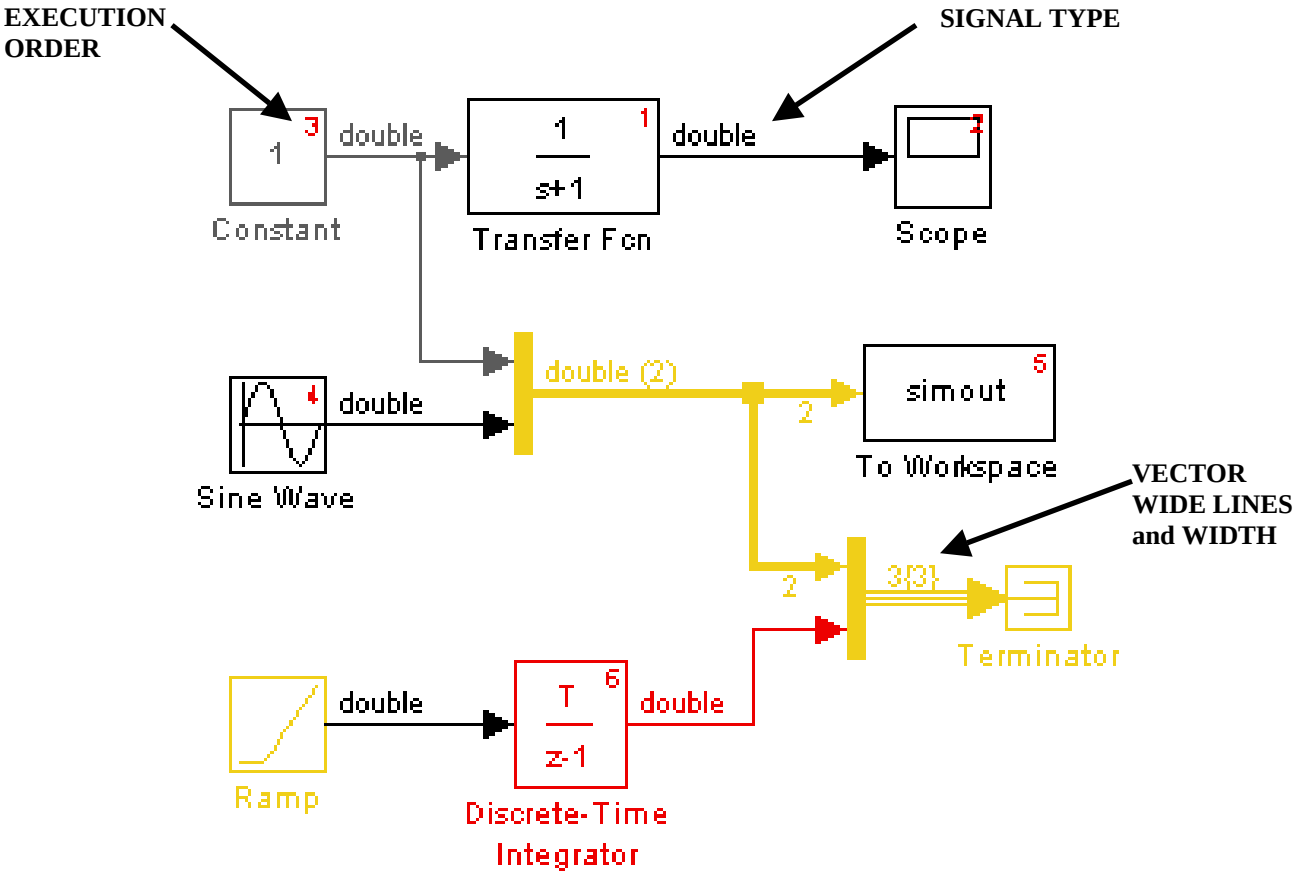

Figure 29: Example of block display options

Setting simulation parameters

Running a simulation in the computer always requires a numerical technique to solve a differential equation. The system can be simulated as a continuous system or a discrete system based on the blocks inside. The simulation start and stop time can be specified. In case of variable step size, the smallest and largest step size can be specified. A Fixed step size is ***recommended*** and it allows for indexing time to a precise number of points, thus controlling the size of the data vector. *Simulation step size* must be decided based on the dynamics of the system. A thermal process may warrant a step size of a few seconds, but a DC motor in the system may be quite fast and may require a step size of a few milliseconds.

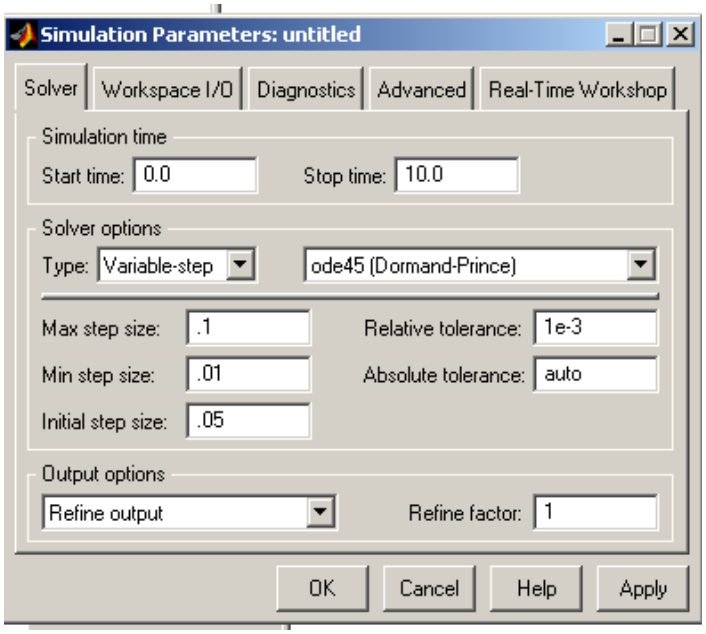

Figure 30: Simulation settings

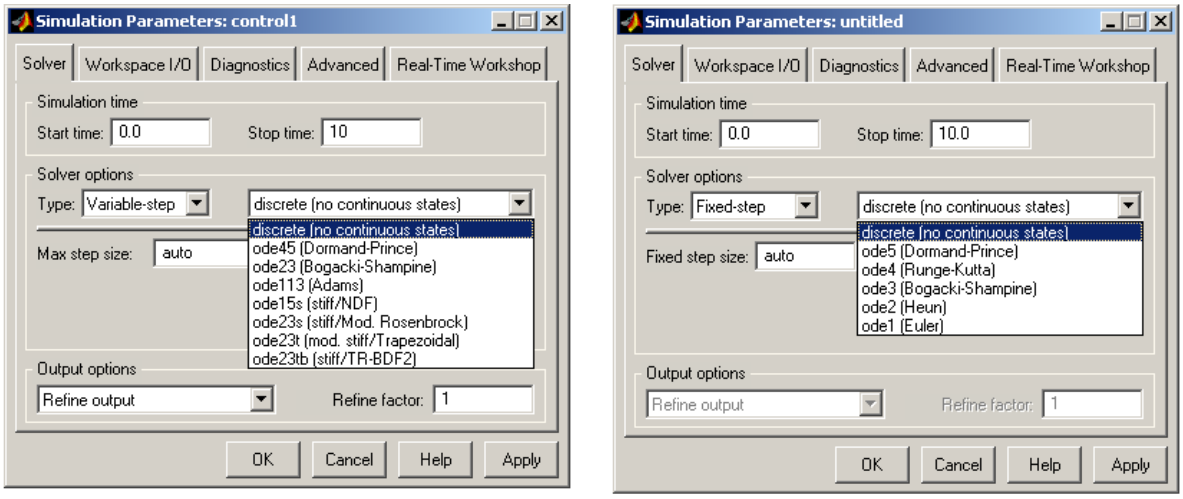

Figure 31: Available numerical methods for solving dynamic equations

Concept of Hardware in the Loop

Simulink's ***REAL TIME WORKSHOP*** (RTW) provides the ability to link Simulink to any hardware available, thus providing control capability directly from a high-level programming language like MATLAB/Simulink. This concept, known as Hardware-in-the-Loop (HIL) is used extensively in control development. The concept of Real-time control using hardware in the loop is explained below.

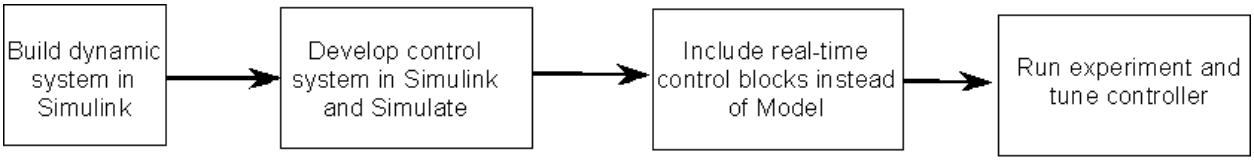

Figure 32: Concept of Hardware in the Loop

An example is shown below in Figure 33. A first order model is replaced by a Digital to Analog Converter (DAC) and an Analog to Digital converter (ADC) feeding information from and to the actual hardware. The DAC signal is sent to an actuator, and the ADC signal is acquired from a sensor. An example of a Real-time control system is dSPACE, who provide the hardware (data acquisition and connectivity boards) and the necessary hardware-software interfaces in Simulink. The interface blocks are available from a dSPACE library in Simulink.

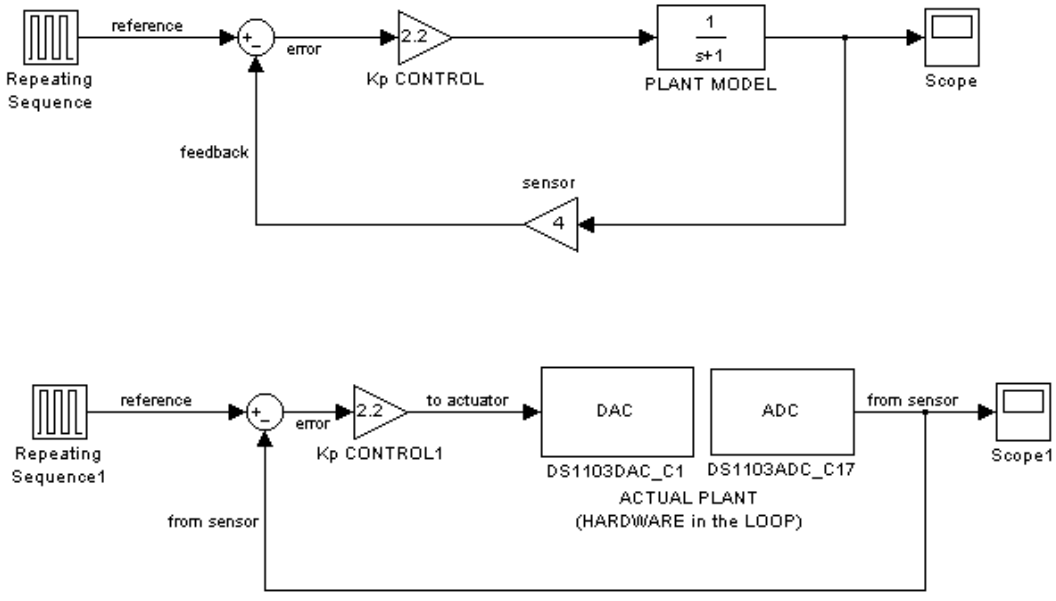

Figure 33: Example of Hardware in the Loop

Tips and Tricks

Here are some useful tips for working with Simulink.

1. To copy a block, right click and drop the block onto the target Simulink window.

2. LIBRARIES
   To create a template block, save the Simulink block as a library. In the future, you can copy the block from the library onto any Simulink block where it needs to be used. Changing the block structure or parameters in the library activates the changes in all the blocks where they may be used. If you want to break the link of a particular block from its library source, right-click and say "*Break Library Link*".

3. To create a branch from a signal, right click on the source signal at the point where you would like to branch to start, and drag it to the target location.

4. Always connect all open ports in a block diagram, to prevent warnings about unconnected ports. Ground (in Sources) and terminator (in Sinks) can be used to plug open ports.

5. Compatibility with older versions of MATLAB
   Simulink files saved in MATLAB 6 (Release 12) / Simulink 4 or MATLAB 6.1 (Release 12.1) / Simulink 4.1 may not be compatible with MATLAB 5.3 (Release 11) / Simulink 3.0 and earlier versions. To provide compatibility, specify the type when saving the Simulink block, as shown below in Figure 34.

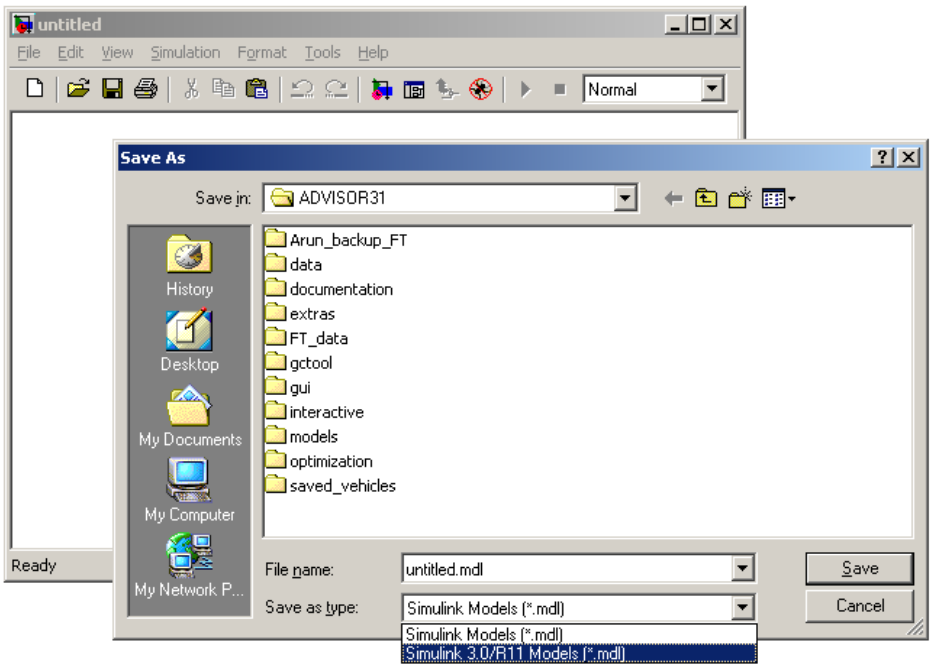

Figure 34: Providing compatibility with earlier versions of Simulink

Resources

The Mathworks Website (contains online documentation)
http://www.mathworks.com

Control System Analysis using MATLAB
http://rclsgi.eng.ohio-state.edu/matlab

Simulink Tutorial by T. Nuygen
http://www.messiah.edu/acdept/depthome/engineer/Resources/tutorial/matlab/simu.html

MATLAB/Simulink Resources
http://www.eng.fsu.edu/~cockburn/matlab/matlab_help.html

Simulink: A graphical tool for dynamic system simulation (by G.D. Buckner, NCSU)
http://www.mae.ncsu.edu/org/asme/webpages/tutorial1.pdf