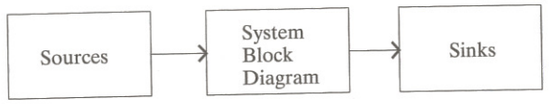


FIGURE 4.1: Elements of a Simulink model

of sinks are graphs, oscilloscopes, and output files. Sink blocks are found in the Sinks block library.

Frequently, Simulink models lack one or more of these three elements. For example, you might wish to model the unforced behavior of a system initially displaced from its equilibrium state. Such a model would have no inputs, but it would have system blocks (Gain blocks, Integrators, etc.) and probably sinks. It is also possible to build a model that has sources and sinks, but no system blocks. Suppose that you need a special signal composed of the sum of several functions. You could easily generate the signal using Simulink source blocks, and send the signal to the MATLAB workspace or to a disk file.

4.2 OPENING A MODEL

In Chapter 3, we discussed creating a new model by clicking the new window icon from the Simulink toolbar. We also discussed saving a model using **File:Save**. To use an existing model, click the open file icon from the menu bar of the Simulink window and select the file.

The default directory will be the current default directory in the MATLAB session from which you started Simulink. If you change to a different directory using the Simulink file menu, the default directory will not change. It will remain the same as the current directory in the MATLAB session, so if you wish to open another model in the same directory as the model you previously opened, and that model was not in the directory currently active in the MATLAB session, you will have to navigate once again to the appropriate directory. Therefore, it is frequently convenient to change to the directory containing the Simulink model in the MATLAB session (using the `cd` command), then open the model using the Simulink file open button.

Although it is not necessary, we recommend that you set up a directory for your Simulink models that is separate from the MATLAB directory structure. This will make it easier to keep track of your files. It will also eliminate the possibility of overwriting or deleting your files if you upgrade MATLAB and Simulink in the future.

An alternative method for opening an existing model is to type the name of the model as a command at the MATLAB prompt. MATLAB will search for the model via the MATLAB path, starting in the current directory. For example, if the model is named `examp.mdl`, enter the command `examp` at the MATLAB prompt.

4.3 MODEL WINDOW

You have several options for customizing the Simulink model window to suit your personal preferences. The model window for a simple system is shown in Figure 4.2. The toolbar contains convenient shortcuts for a number of common functions, such as saving and running the model. The functions of each button on the toolbar are

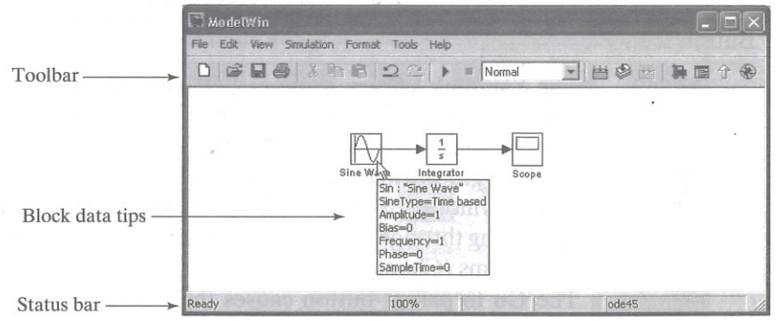


FIGURE 4.2: Model window

TABLE 4.1: Model Window Toolbar Buttons

Button Icon	Function
	The **New model** button creates a new empty model window.
	The **Open model** button opens the **File:Open model** dialog box.
	The **Save model** button saves the model. If the model has not been previously saved, the **Save model** button opens the **File:Save as** dialog box.
	The **Print** button opens the **File:Print model** dialog box.
	The **Cut** button deletes a selection and places it on the clipboard. The **Cut** button is dimmed when nothing is selected.
	The **Copy** button places the current selection on the clipboard. The **Copy** button is dimmed when nothing is selected.
	The **Paste** button places the contents of the clipboard in the model window. The **Paste** button is dimmed if nothing is on the clipboard.
	The **Undo** button reverses the effect of the last copy, cut, delete, or add editing operation. Simulink provides 101 levels of Undo operations.
	Redo reverses the last Undo operation.
	The **Start** button is a shortcut for model window menu choice **Simulation:Start**.
	The **Stop** button is a shortcut for model window menu choice **Simulation:Stop**.
	The **Build all** button is displayed if Real-Time Workshop is installed. See Chapter 15.
	The **Update diagram** button is displayed if Real-Time Workshop is installed. See Chapter 15.
	The **Build subsystem** button is displayed if Real-Time Workshop is installed. See Chapter 15.

TABLE 4.1: Model Window Toolbar Buttons (Cont)

Button Icon	Function
	The **Simulink library browser** button opens the Simulink library browser window. It is equivalent to selecting the Simulink library browser button on the MATLAB toolbar.
	The **Toggle browser** button opens a model browser pane in the model window. The browser provides a convenient means for navigating through complicated models, particularly those involving subsystems (Chapter 7).
	The **Go to parent** button causes the parent system to the current subsystem to be displayed as the active model window, another convenient means with which to navigate in complex models. This is not active in the main model window.
	The **Debug** button enables debugging using the graphical debugger user interface.

shown in Table 4.1. The status bar fields display the current model window zoom factor, the current simulation time (visible while the simulation is in progress), and the currently selected solver. The toolbar and status bar can be hidden using **View:Toolbar** and **View:Status** Bar from the model window menu bar.

4.3.1 Zooming

The model window menu bar **View** menu supplies commands to control the zoom factor of the model window. **View:Zoom In** allows you to view a portion of the model window in more detail, and **View:Zoom Out** presents a wider perspective. **View:Fit System to View** zooms the window so that the model fills the model window. If any block or signal line is selected, there will be a menu choice **View:Fit Selection to View** that zooms the window so that the selection fills the model window. Finally, **View:Normal (100%)** sets the zoom factor to 100%.

4.3.2 Block Data Tips

Block data tips (when enabled) are displayed in a pop-up window that appears when the cursor is stationary on a block for more than approximately one second. The Block data tips window can be configured to display a number of items of information about the block, such as block parameters and a user description character string. You can configure the contents of the Block data tips window using **View:Block Data Tips** submenu commands.

4.3.3 Simulink Block Library

The default method to access Simulink blocks is via the Simulink Library Browser (Figure 3.4). An alternative is to open the Simulink Block Library by right-clicking the Simulink Library icon in the Simulink Library Browser window and clicking the single menu choice **Open the 'Simulink' Library**. The Simulink Block Library is

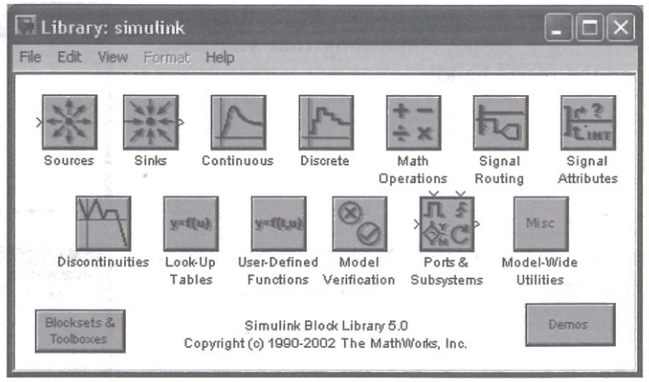


FIGURE 4.3: Simulink block library

a special type of model window called a *block library*, to be discussed in detail in Chapter 7. The Simulink Block Library is shown in Figure 4.3.

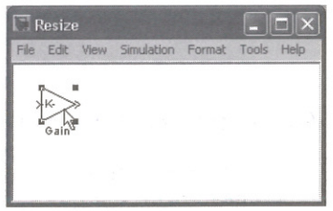
4.4 MANIPULATING BLOCKS

In Chapter 3, you learned how to drag a block from a block library to a model window and to flip a block. In addition to those basic operations, you can resize, rotate, copy, and rename blocks. In this section, we will discuss these and several other block manipulation operations. To prepare to practice the operations illustrated here, start Simulink, or, if Simulink is already started, open a new model window.

4.4.1 Resizing a Block

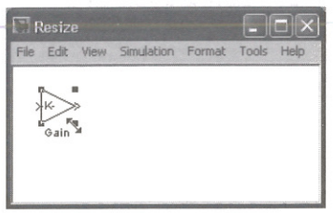
Frequently, resizing a block can improve the appearance of a model. To resize a block, proceed as follows:

Open a block library, and drag a block to the model window. Here, we're using a Gain block from the Math block library. We've set block parameter **Gain** to the value 1227.86. Because this value will not fit on the block icon, the displayed value is `-K-`.

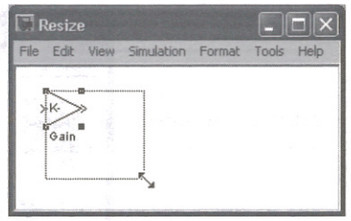


To resize a block, first select the block, causing the handles to appear.

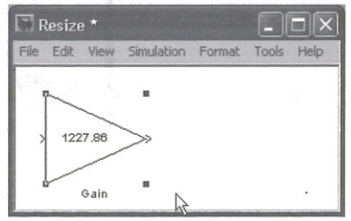
Click the desired handle, and, continuing to depress the mouse button,



drag the handle to resize the block. Notice that the cursor changes shape, confirming that you have grasped the resize handle.

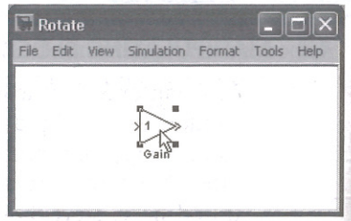


Release the mouse button. Now, the block icon is large enough to display the value of **Gain**.

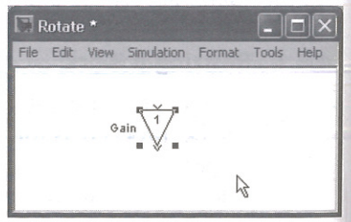


4.4.2 Rotating a Block

Occasionally, you will want to rotate a block. Select the block, then choose **Format:Rotate**.



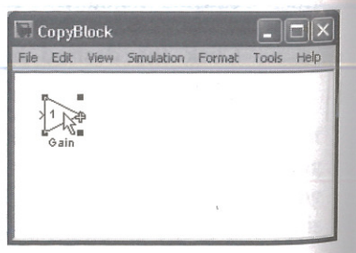
The block will rotate 90 degrees clockwise.



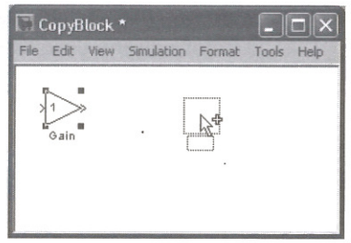
4.4.3 Copying a Block within a Model

You will frequently want to copy a block from within a model. For example, after you have resized a block, you will probably want all similar blocks to appear identical. Rather than attempting to resize each block, you can copy the block you resized.

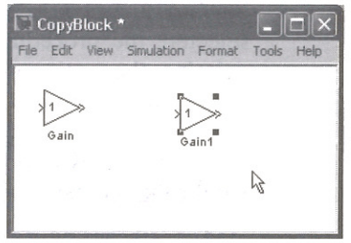
To copy a block within a model, depress and hold the Control key, then click the block.



Drag the copy to the desired position.



Release the mouse button, completing the copy operation.



If you have a two- (or three-) button mouse, dragging by using the right mouse button is equivalent to dragging with the Control key depressed.

An alternative is to click the block, select **Edit:Copy** or the **Copy** button from the model window toolbar, or press Control-C. Then, choose **Edit:Paste** or the **Paste** button from the model window toolbar, or press Control-V.

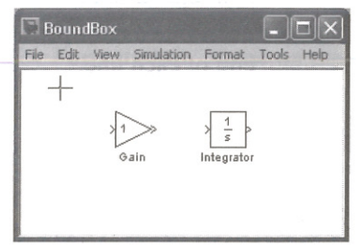
4.4.4 Deleting Blocks

To delete a block, select the block, and then press the Delete key. An alternative is to select the block, then choose **Edit:Clear** from the model window menu bar or **Cut** from the model window toolbar. To delete the block and place it on the Clipboard, select the block, then choose **Edit:Cut** or the **Cut** button from the model window toolbar, or press Control-X.

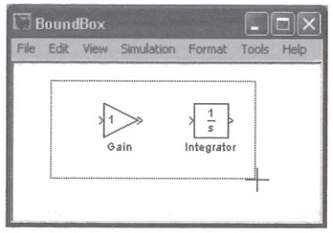
4.4.5 Selecting Multiple Blocks

You can select multiple blocks and move, copy, or delete them as a group. There are two ways to select multiple blocks. The first is to depress and hold the Shift key while clicking each block in the group. The other method is to use a bounding box as described next.

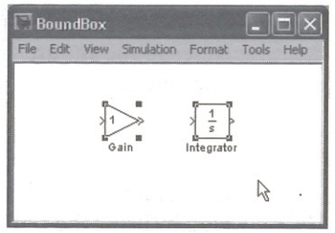
Click and hold the mouse button outside the blocks.



Drag the bounding box that appears so as to enclose all the desired blocks.



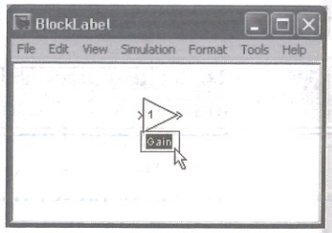
Release the mouse button. All blocks in the bounding box are now selected. Once the blocks are selected, the procedures for moving, copying, or deleting the entire group are the same as the corresponding procedures for a single block. For example, to delete the group, press the Delete key or choose **Edit:Cut** or **Edit:Clear** from the model window menu bar.



4.4.6 Changing a Block Label

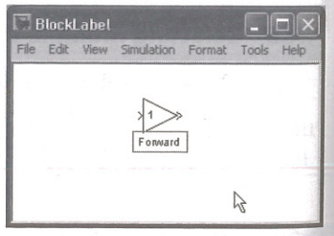
Simulink supplies a default label for each block as you place the blocks in the model window. For example, the first Gain block will have the default name "Gain", the second gain block will be labeled "Gain1", and so on. Change the block label as described next.

Click the block label. An editing cursor will appear. You can position the cursor anywhere in the label by clicking, and you can move the cursor by using the cursor movement keys.

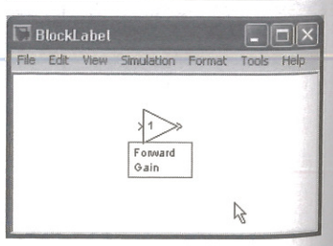


To replace a label, select it, then double-click it. The label will be highlighted as shown here.

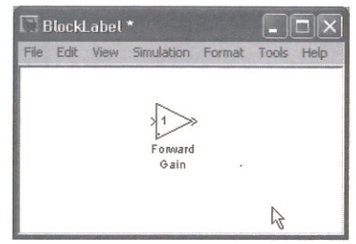
Type the new label.



To create a multiple-line label, press Return at the end of each line.



Click away from the block to accept the label.

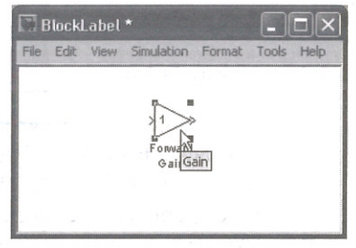


Each block in a window must have a unique name of at least one character.

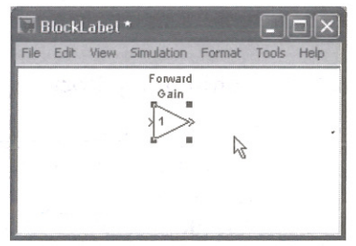
4.4.7 Changing Label Location

You can move the block label from below to above the block as described below.

Select the block.



Choose **Format:Flip Name**. An alternative is to click the name and drag it to the desired position. If the block is rotated 90 degrees, the block label will be to the left or the right of the block.



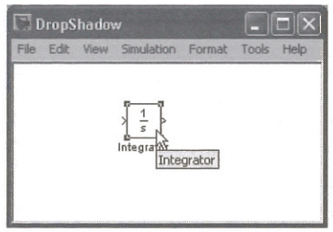
4.4.8 Hiding a Label

It is sometimes desirable to hide the name of a block. For example, because the shape of a Gain block uniquely identifies the purpose of the block, and the value of the gain is displayed on the block, a cluttered model might be improved by hiding the names of the gain blocks. To hide the name, select the block, and choose **Format:Hide Name**. Note that this does not change the name of the block in any way—it simply makes the name invisible. If the name of a block is hidden, when the block is selected, **Format:Hide Name** is replaced by **Format:Show Name** on the **Format** pull-down menu.

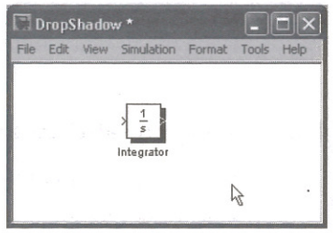
4.4.9 Adding a Drop Shadow

If you want to call special attention to a block, you can apply a drop shadow.

Select the block.



Choose **Format:Show Drop Shadow** from the model window menu bar. If a block is configured with a drop shadow, the menu choice will change to **Format:Hide Drop Shadow** for that block, permitting you to remove the drop shadow if desired.



4.4.10 Using Color

You can set the foreground and background colors for each block or a selected set of blocks. Foreground color is the color of a block's outline and labels. Background color is the color of the body of the block. To change the color of a block, select the block, and then choose **Format:Foreground Color** or **Format:Background Color**. You can also change the color of the model window background using **Format:Screen Color**.

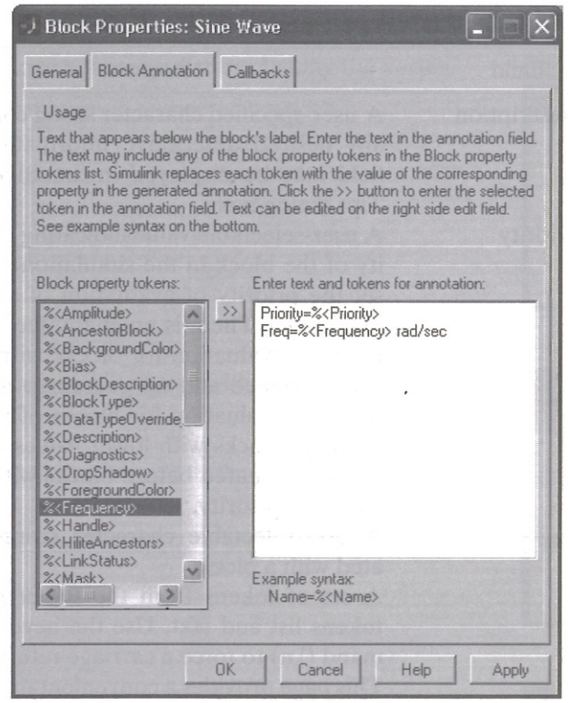
4.4.11 Configuring Blocks

In Chapter 3, we discussed setting block parameters. In the example in Chapter 3, the numeric parameters were set to constants. However, configuration parameters do not have to be constants. They can be any valid MATLAB expression and may use variables that will be defined in the MATLAB workspace when the model is executed. This is a very useful capability. Later, we'll discuss executing a Simulink model from a MATLAB script. By making certain parameters variables, the parameters can be changed by the script.

In addition to setting block parameters using the block parameters dialog box, you can set additional block characteristics using **Edit:Block Properties** (also available by right-clicking a block). The Block Properties dialog box consists of three tabbed pages: **General**, **Block Annotation**, and **Callbacks**. The Block Properties dialog box **Block Annotation** page for a Sine Wave block is shown in Figure 4.4 along with the corresponding block. The three tabbed pages contain five fields, as described in Table 4.2.

4.5 SIGNAL LINES

In Chapter 3 we discussed drawing signal lines using the Simulink automatic routing capability and drawing signal lines in segments. In this section we'll discuss techniques to edit signal lines.

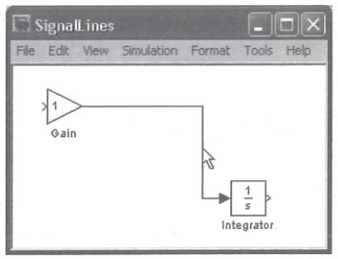


(a) Dialog box (b) Corresponding block

FIGURE 4.4: Block Properties dialog box **Annotation** tab

4.5.1 Moving a Segment

To move a line segment, click the segment.



The cursor will change shape.

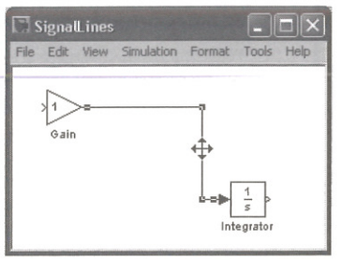
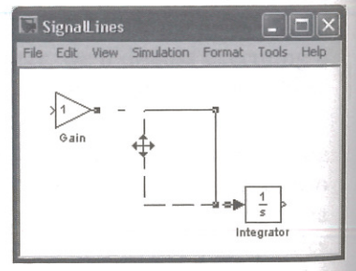


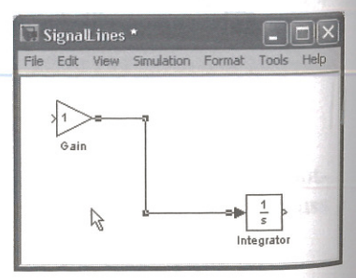
TABLE 4.2: Block Properties Dialog Box Fields

Page	Field	Purpose
General	Description	A user-specified character string containing any desired descriptive information. This field may be displayed in the Block Data Tips dialog box.
	Priority	A user-selectable value indicating the priority of the block in the simulation. You can set the priority of evaluation of all blocks in a model. There is no guarantee as to the order of evaluation for blocks with no priority assigned and no guarantee as to the order of evaluation for blocks of the same priority. Blocks with lower values of priority are evaluated before blocks with higher values of priority.
	Tag	A user-selectable character string associated with a block.
Annotation		A set of tokens from the **Block property tokens** list and text. Use the newline command (\n) to force a carriage return.
Callbacks		This page provides a convenient means with which to assign callback functions to a block. The list of available callbacks is in the **Callback functions list**. Details on callback functions are presented in Chapter 9.

Keeping the mouse button depressed, drag the segment to the desired location.

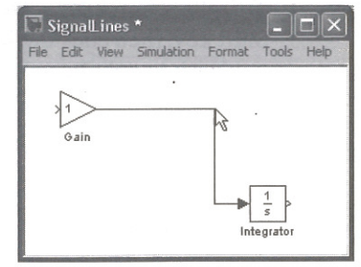


Release the mouse button.

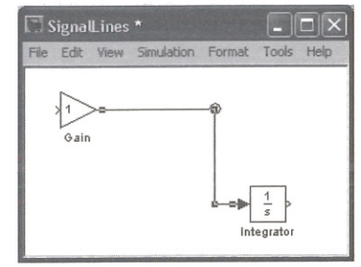


4.5.2 Moving a Vertex

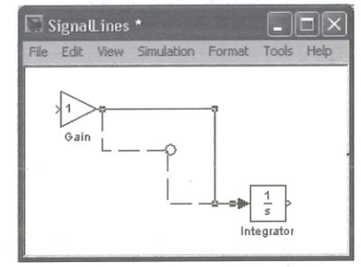
To move a vertex, start by clicking the vertex.



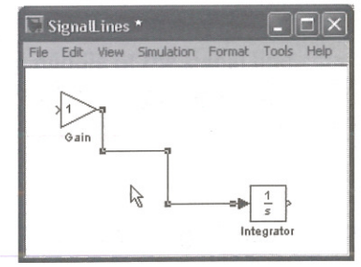
The cursor will change shape to a circle.



Drag the vertex to the desired location.



Release the mouse button to complete the operation.

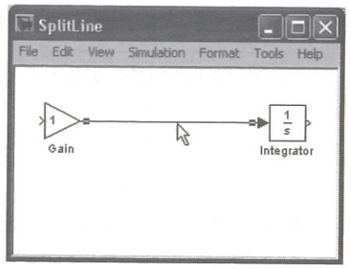


4.5.3 Deleting a Signal Line

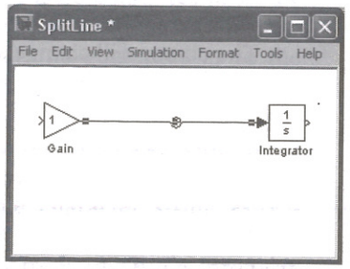
To delete a signal line, select the signal line by clicking it. Press the Delete key, or choose **Edit:Clear** or **Edit:Cut** from the model window menu bar.

4.5.4 Splitting a Signal Line

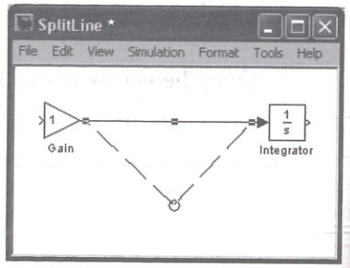
To split a line, first select the line. Depress the Shift key, then click the line at the point at which you wish to split it.



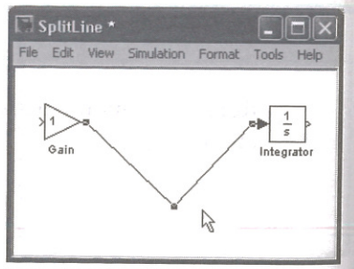
The cursor will change shape to a circle, and the segment will split into two segments.



Drag the new vertex to the desired location.



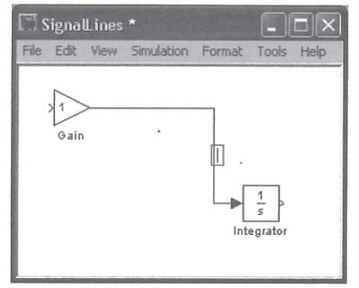
Release the mouse button to complete the operation.



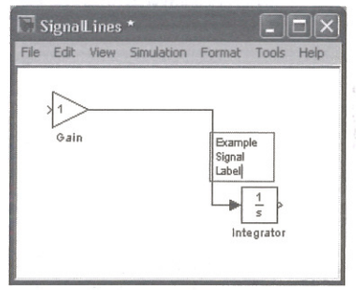
4.5.5 Labeling a Signal Line

Each signal line may have a label. The label can be positioned at either end of the signal line and on either side. Signal line labels, unlike block labels, do not have to be unique.

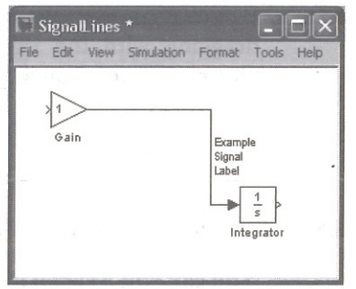
To add a label to a signal line, double-click the line, causing an editing cursor to appear near the line. Be sure that you click the line itself, not just near the line. Double-clicking away from a line will result in an annotation (to be discussed later) rather than a signal line label.



Enter the label. As with block labels, press Return at the end of each line to enter a label consisting of more than one line of text.

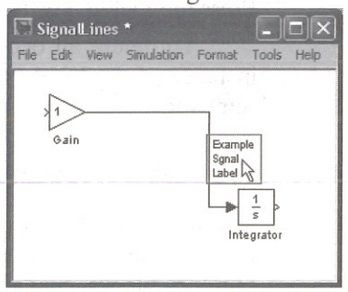


Click away from the line to complete entering the label. The label will snap to a position near the center of the line.

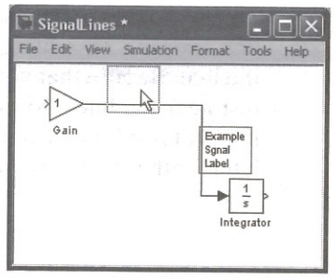


4.5.6 Moving or Copying a Signal Line Label

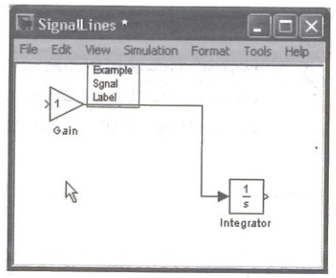
You can move a signal line label to either end or the middle of the signal line. To move a signal line label, click the label.



Drag the label to the desired location near the signal line.

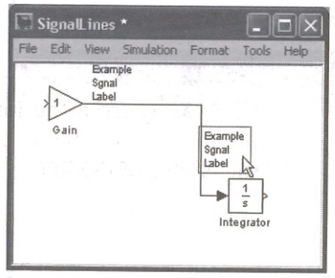


Release the mouse button. The label will snap into position.



To copy a signal line label, depress the Control key while dragging the label to the new position.

If your mouse has two or three buttons, dragging with the right mouse button is an alternative to dragging while depressing the Control key.



4.5.7 Editing a Signal Line Label

You can edit a signal line label using the same techniques used to edit a block label. All occurrences of the label will be changed.

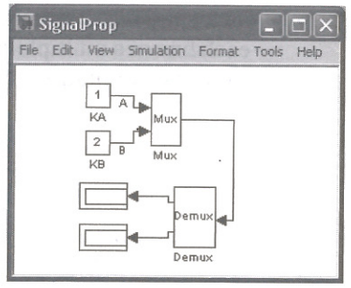
If there are multiple occurrences of a signal line label, you can delete a single occurrence. Depress the Shift key, then click on the occurrence you wish to delete. Press the Delete key to complete the process.

To delete all occurrences of a signal line label, delete all characters in one instance of the label. When you click away from the label, all occurrences will be removed.

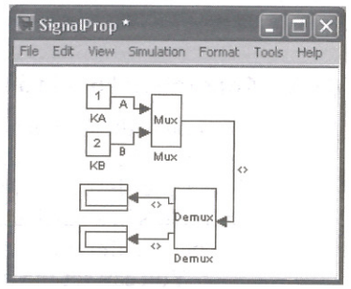
4.5.8 Signal Label Propagation

Signal line labels can propagate through several blocks in the Connections block library. Among these are the Mux and Demux, Goto and From, and Inport and Outport blocks. Signal line propagation provides an accurate means with which to determine the exact content of a signal line. This can make a model easier to understand and can also be useful in debugging. The process is illustrated next.

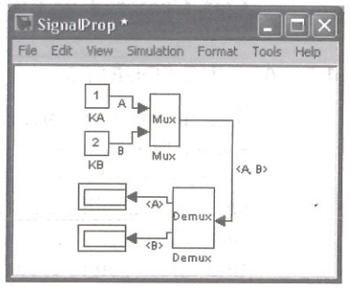
Here, we have a model in which two scalar signals produced by Constant blocks are combined to form a vector signal by a Mux block. The Demux block splits the vector signal into two scalar signals that are displayed by Display blocks. Notice that the model is shown before running the simulation (the Display blocks initially display 0). First, label the source signal lines. Here, the outputs of the constant blocks KA and KB are labeled A and B.



Label each signal line onto which you want the labels propagated with the single character <. Here, we configured all three signal lines to propagate the labels. Note that the > character is automatically added by Simulink.



Choose **Edit:Update Diagram** from the menu bar of the model window. Here, the signal line leaving the Mux block contains two signals, so the label is changed to <A,B>, showing both. The Demux block separates the vector signal into components, so the signal lines leaving it each contain only one signal, as shown by the labels. In both cases, the propagated labels are enclosed in angle brackets to distinguish them from simple labels.

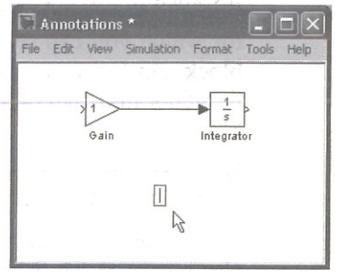


4.6 ANNOTATIONS

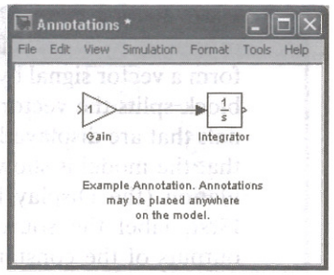
You can add annotations to a model to make it easier to understand. You can also change the font used in an annotation to add emphasis.

4.6.1 Adding Annotations

Double-click at the location where you want the center of an annotation to be. An editing cursor will appear.



Enter the annotation. Press Return at the end of each line of a multiple-line annotation. Click away from the annotation to complete the process.



You can move and copy annotations using the same procedures used to move and copy blocks.

4.6.2 Annotation Formatting

To change the font of an annotation, select the annotation. Choose **Format:Font** from the menu bar of the model window. A font selection dialog box will be displayed. Select the desired font, then press **OK**; then click away from the annotation. All characters in a particular annotation will be the same font, but different annotations can be in different fonts.

You can set the justification for an annotation using **Format:Text alignment** to **Left**, **Centered**, or **Right**.

4.7 ADDING SOURCES

The inputs to a model are called *sources*, located in the Sources block library. A source block has no inputs and at least one output. Detailed documentation for each block in the Sources block library is available via the online Help system. In this section, we will mention several of the more commonly used blocks in the Sources block library. Then, we will briefly discuss From Workspace and From File blocks that allow you to create any time-dependent signal you can describe mathematically in MATLAB, and then use that signal as a Simulink input.

4.7.1 Common Sources

Many of the input signals used in modeling dynamical systems are available in the Sources block library. The Constant block produces a fixed constant signal, the magnitude of which is set in the block dialog box and displayed on the block icon. The Step block produces a step function. You can set the time of occurrence of the step and the signal magnitude before and after the step. There is a Sine Wave block for which you can set the amplitude, phase, and frequency. The Signal Generator block can be set to produce sine, square, or sawtooth. More complex signals can be generated by combining the signals from multiple source blocks using a Sum block.

EXAMPLE 4.1 Unit impulse function

A signal that is useful in determining the behavior of dynamical systems is the unit impulse, also known as the delta function or Dirac delta function. (For a detailed discussion, see

Meirovitch [1].) The unit impulse $\delta(t-a)$ is defined to be a signal of zero duration, having the properties:

$$\delta(t-a) = 0, \qquad t \neq a$$

$$\int_{-\infty}^{\infty} \delta(t)dt = 1$$

Although the unit impulse is a theoretical signal that cannot exist, it is a close approximation to real impulse signals that are common. Physical examples are collisions, such as a wheel hitting a curb or a bat hitting a ball, or near-instantaneous velocity changes, such as firing a bullet from a rifle. Another use for the unit impulse is the assessment of a system's dynamics. The motion of a system forced by a unit impulse is due purely to the dynamics inherent in the system. Thus, you can use the impulse response of a complex system to determine its natural frequencies and modes of vibration.

You can approximate a unit impulse function using two Step blocks and a Sum block, as shown in Figure 4.5. The idea is to produce, at the desired time a, a very

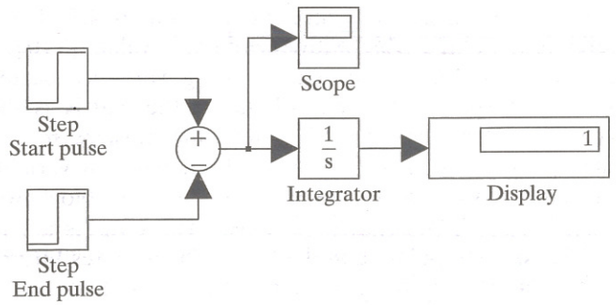


FIGURE 4.5: Generating a unit impulse function

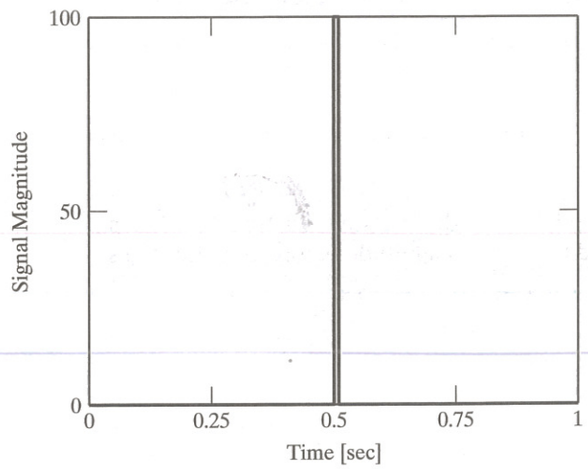


FIGURE 4.6: Unit-impulse signal

short duration d pulse of a magnitude M, such that $Md = 1$. The trick is in deciding on the proper value of d. It must be short relative to the fastest dynamics of the system. If it is too short, it can cause numerical problems, such as excessive round-off error. If it is too long, it will not adequately simulate a true impulse. Usually, an acceptable compromise can be found with a little experimentation.

The model in Figure 4.5 is set to simulate a unit impulse at 0.5 seconds with a pulse of 0.01 seconds duration and magnitude of 100. The Step block labeled Step Start pulse is configured as follows: **Step time** is 0.5, **Initial value** is 0, and **Final value** is 100. The Step block Step End pulse is configured as follows: **Step time** is 0.51, **Initial value** is 0, and **Final value** is 100. The simulation is configured to stop at 1 second. A plot of the output of the Sum block (sent by the Scope block to the MATLAB workspace) is shown in Figure 4.6. The Integrator block computes the time integral of the output of the Sum block, which is displayed using a Display block, and which has the desired value (1).

4.7.2 From Workspace Block

The From Workspace block permits you to design a custom input signal. The block and its dialog box are illustrated in Figure 4.7. The block configuration parameter is a matrix table with the default value `[T,U]`. The input must be in the form of a MATLAB matrix, using variables currently defined in the MATLAB workspace. The first column of the matrix is the independent variable that corresponds to simulation time and must be monotonically increasing. The subsequent columns are values of the dependent variables corresponding to the independent variable in the first column. The block will produce as many outputs as there are dependent variables. The outputs are produced by linearly interpolating or extrapolating in the table. Check boxes **Interpolate data** and **Hold final data value** modify the behavior of the block. For details, refer to the block help screen.

EXAMPLE 4.2 From Workspace block usage

To illustrate the use of the From Workspace block, suppose that we wish to generate a signal defined as

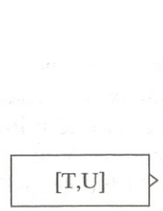
$$u(t) = t^2$$

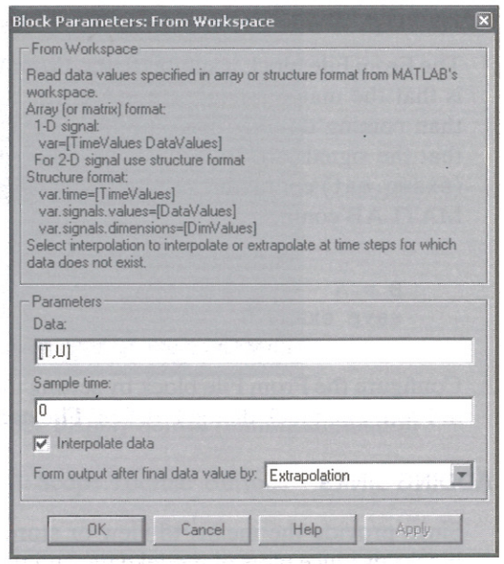
Listing 4.1 illustrates an M-file that produces a suitable input table.

Listing 4.1: M-file to create input table for From Workspace block

```
%Generate a signal for a From Workspace block
t = 0:0.1:100 ;    %Independent variable
u = t.^2      ;    %Dependent variable
A = [t',u']   ;    %Form table
```

Note that the M-file must be saved with a name different from the Simulink model file. To use this table, you must first execute the M-file from within MATLAB, creating the table `A`. Configure the From Workspace block by replacing `[T,U]` in the From Workspace dialog box with the name of the table (`A`).

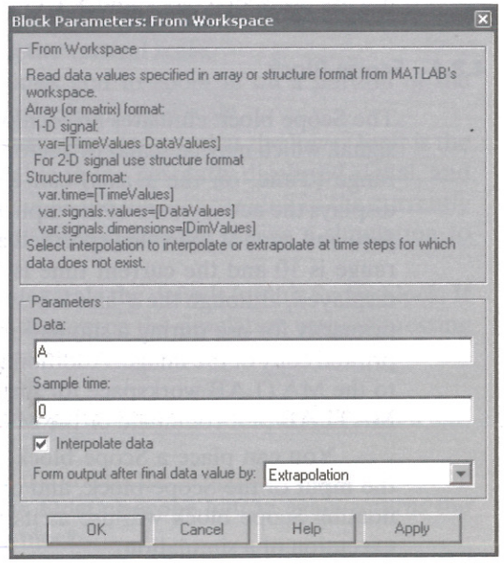




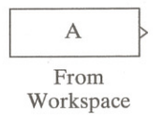
(a) From Workspace Block (b) From Workspace dialog box

FIGURE 4.7: From Workspace block and dialog box

Set Matrix table to `A`, the name of the table created in the MATLAB workspace. Click Close.



The block icon will display the name of the table.



4.7.3 From File Input Block

The From File block is similar to the From Workspace block. The primary difference is that the matrix is stored in a file in MATLAB matrix file (`.mat`) format rather than coming directly from the MATLAB workspace. An additional difference is that the signals are stored in rows rather than in columns. You can produce a file (`examp.mat`) containing the matrix produced by the M-file in Listing 4.1 using the MATLAB commands:

```
B = A'
save examp B
```

Configure the From File block by setting the full filename (for example, `examp.mat`) in From File block dialog box field **File name**.

4.8 ADDING SINKS

Sinks provide the means to view or store model data. The Scope and XY Graph blocks produce plots of model data, and the Display block produces a digital display of the value of its input. The To Workspace block saves a signal to the MATLAB workspace, and the To File block saves a signal in MATLAB `.mat` file format. The Stop block causes a simulation to stop when its input is nonzero. A detailed reference for each of these blocks is available in the online Help system. We will discuss the Scope block and XY Graph block in some detail in this section.

4.8.1 Scope Block

The Scope block emulates an oscilloscope. The block shows a segment of the input signal, which may be scalar or vector. Both the vertical range (y-axis) and horizontal range (Time, on the x-axis) can be set to any desired values. The vertical axis displays the actual value of the input signal. The horizontal axis scale always starts at zero and ends at the value specified as time range. So, for example, if the horizontal range is 10 and the current time is 100, the input data for the period 90 to 100 is displayed, although the x-axis labels will still be 0 to 10. The Scope block is intended primarily for use during a simulation, but the block has the capability to produce a printed copy of the image. Additionally, the Scope block will send the signals it plots to the MATLAB workspace for further analysis or plotting using, for example, the MATLAB plot command or the MATLAB `simplot` command.

You can place a Scope block in a model without connecting a signal line to the input of the Scope block, and configure the block as a floating Scope block. A floating Scope block will use as its input any signal line that you click during the execution of a simulation.

Figure 4.8 illustrates a Scope block. Note that there is a toolbar that contains eleven icons along the top of the Scope block window. These buttons allow you to zoom in on a portion of the display, autoscale the display, save a configuration for future use, and open a scope properties dialog box. Described in Table 4.3 are the functions of each button.

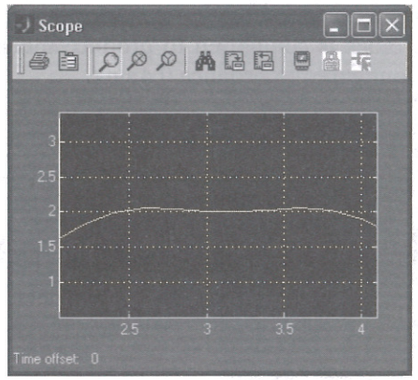


FIGURE 4.8: Scope block display before running simulation

TABLE 4.3: Scope Toolbar Buttons

Button Icon	Function
	Click the **Printer** button to print the scope plots.
	Click the **Parameters** button to open the Scope parameters dialog box. This dialog box allows you to set the default scales for the Scope block and to send the Scope data to the MATLAB workspace.
	The **Zoom** button allows you to enlarge a region of the display.
	The **Zoom X** button allows you to zoom in on a portion of the display without changing the vertical scale.
	The **Zoom Y** button allows you to zoom in on a portion of the display without changing the horizontal scale.
	Autoscale changes the vertical scale such that the lower limit is the same as the minimum value in the currently displayed signal, and the upper limit is the same as the maximum value in the currently displayed signal. You can click **Autoscale** during a simulation to rescale the display.
	Save axis makes the current scale the default for this Scope block. If you change the scale and then rerun the simulation without pressing Save axis first, the scale will revert to the current default when the simulation starts.
	Restore axis causes the Scope to return to the currently saved axis settings after zooming.
	The **Floating scope** button converts the Scope block into a Floating Scope block. Pressing the **Floating scope** button again returns the block to a normal Scope block.
	The **Lock Axes selection** button toggles axis selection for Floating Scope blocks. The button is not active for normal Scope blocks.
	The **Signal selection** button opens the signal selection dialog box for Floating Scope blocks. The button is not active for normal Scope blocks.

Zooming the Scope Display. Consider the model shown in Figure 4.9. The top Sine Wave block is configured to produce the signal $\sin t$, and the other Sine Wave block is configured to produce $0.4 \sin t$.

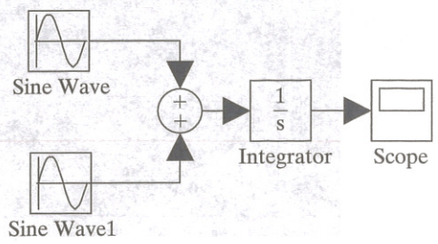
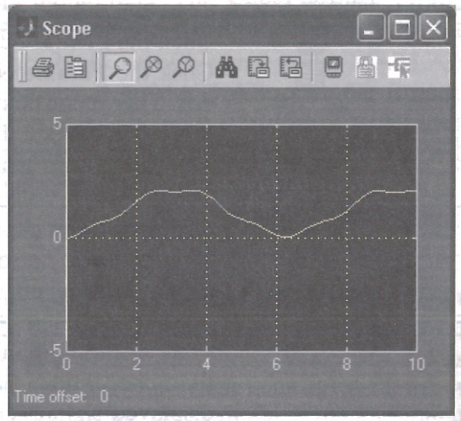
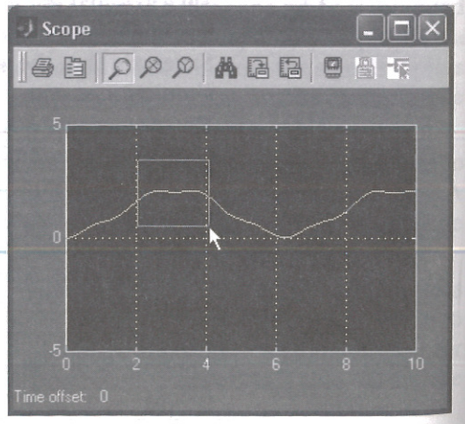


FIGURE 4.9: First-order system with sinusoidal input

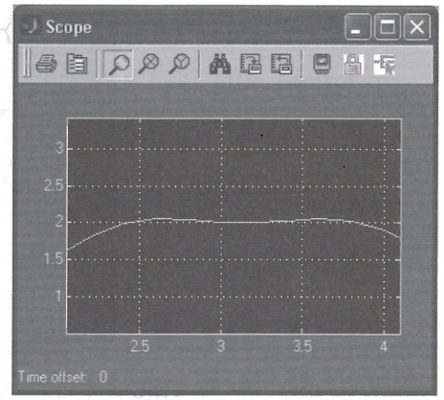
Open the Scope block by double-clicking it. Run the simulation, resulting in the display shown here. At this point, you could rescale the display by clicking the **Autoscale** button. Instead, zoom in on the peak between 2 and 4 seconds, as follows.



Click the **Zoom** button. Then, enclose the area you wish to zoom in on using a bounding box.



The Scope scales will change to include only the area you zoomed in on.



Scope Parameters Dialog Box. The **Parameters** button opens the Scope Parameters dialog box, which has two pages. The **General** page (Figure 4.10) has fields to set the number of axes and time range and to control the spacing between plotted points. The **Data history** page (Figure 4.11) has fields to control the size of the scope data buffer and to send the displayed data to the MATLAB workspace.

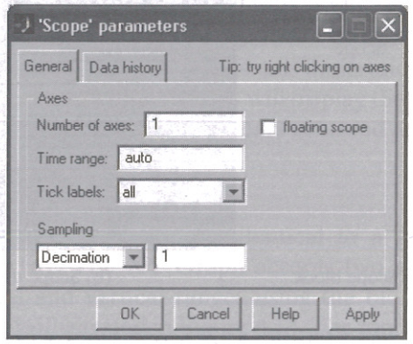


FIGURE 4.10: Scope parameters **General** page

The **General** page (Figure 4.10) has two sections. The Axes section controls the number and configuration of axes. **Number of axes** controls the number of axes displayed in the scope window and the number of inputs to the Scope block. There will be one input for each axis. For example, setting **Number of axes** to 2 produces the scope block and display shown in Figure 4.12. Checking **floating scope** converts the Scope block to a Floating scope block, which has no inputs. A floating scope block displays a signal line that is selected during the execution of a simulation. Note that in order to use a Floating scope block with many Simulink blocks, it is necessary to disable **Signal storage reuse** in the **Simulation:Parameters** dialog box Advanced page, to be discussed in Section 4.10. **Time range** controls the scale of the time axis. If set to `auto`, the scale will range from zero to the final simulation time. If set to a number greater than zero, the time range will be zero to the value entered.

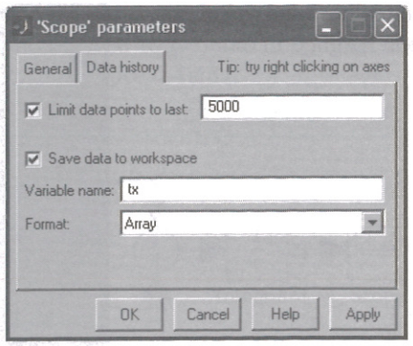
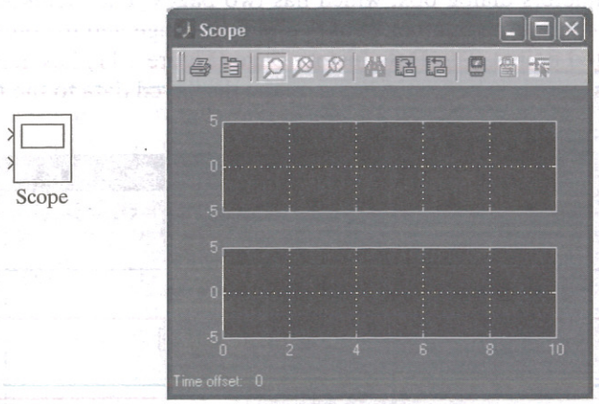


FIGURE 4.11: Scope parameters **Data history** page



(a) Scope block (b) Scope display

FIGURE 4.12: Scope block with two axes

Tick labels is a drop-down menu that can be set to all to place time ticks on each time axis, none, or bottom axis only for ticks on the x-axis only.

The Sampling section of the **General** page contains a drop-down list containing two choices: **Decimation** and **Sample time**. If **Decimation** is selected, the corresponding data field is set to a decimation factor that must be an integer. If **Decimation** is selected and set to 1 (the default), every point in the block input is plotted. If **Decimation** is set to 2, every other point is plotted, and so on. If **Sample time** is selected, the absolute spacing between plotted points must be entered in the data field.

The Scope block stores the input points in a buffer. The size of the buffer can be set using the **Data history** page. Check **Limit points to last** and enter a value to specify the size of the buffer (default is 5000). Autoscaling, zooming, and saving scope data to the workspace all work with this buffer. Thus, if **Limit rows to last** is

set to 1000, and the simulation produces a total of 2000 points, only the final 1000 points are available when the simulation stops.

Although you can print a scope display, you have little control over its appearance. Additionally, the Scope printing capability is not intended to support placing the plot in a word-processing document. However, you can send the Scope data to the MATLAB workspace, and use the extensive plotting capabilities MATLAB provides. To send the Scope data to the MATLAB workspace, select **Save data to workspace** and enter the name of a MATLAB variable. When the simulation stops, the data displayed on the Scope will be stored in the MATLAB variable. Drop-down menu **Format** controls the type and contents of the MATLAB variable. If **Matrix** is selected, there will be one column for the time values and one column for each signal input to the Scope block. Thus, if the signal entering the Scope block is a vector signal with two components, the MATLAB variable will be a matrix with three columns and a number of rows equal to the number of time points displayed on the Scope. If **Structure** is selected, the output will be a MATLAB structure, but no time vector will be stored. Choose **Structure with time** to add a time vector to the structure. If you send the Scope data to the MATLAB workspace, you may use the MATLAB `simplot` command to produce an editable MATLAB plot that resembles the Scope display.

Setting y-Axis Limits. To set y-axis limits, right-click the scope display. In the dialog box that appears, choose **Axis properties**. Another dialog box will appear. This dialog box has fields in which to enter the axis limits and to specify an axis label. The default special string `%<SignalLabel>` will display the signal label of the input signal, if the signal has a label. The label appears centered at the top of the axis.

4.8.2 XY Graph

The XY Graph block produces a graph identical to a graph produced by the MATLAB command `plot`. The XY Graph accepts two scalar inputs. You must configure the horizontal and vertical ranges using the block dialog box. The XY Graph block dialog box is illustrated in Figure 4.13. The top input port is the x-input, and the bottom input port is the y-input.

4.9 DATA TYPES

Most programming languages, including MATLAB, provide multiple data types. A data type is a computer representation of a number, both the amount of storage allocated to the number and the format of the representation. MATLAB and Simulink provide long and short integers, single-precision and double-precision floating point, and Boolean data types. In Table 4.4, the available data types are listed. The size (in bytes) of each data type is dependent upon the computer and operating system used. See the MATLAB online documentation for details.

The default data type for all Simulink signals and block parameters is double precision floating point. Therefore, it is never necessary for you to use the other available data types. Often, however, using the other data types can improve the fidelity of a simulation. For example, a model of a control system that uses Boolean data, analog to digital converters, or finite word length

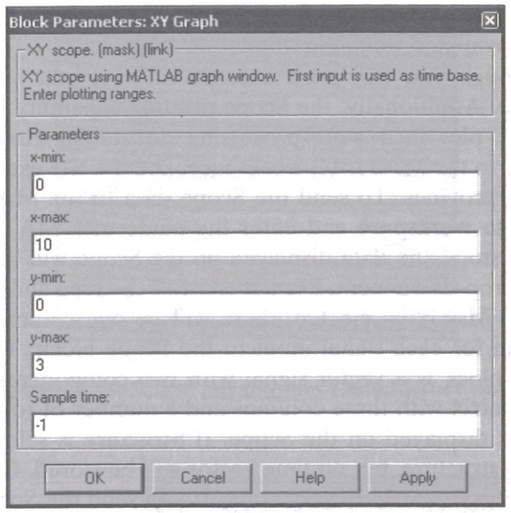


FIGURE 4.13: XY Graph block dialog box

TABLE 4.4: Simulink Data Types

Type	Meaning
double	Double-precision floating point: This is the default Simulink data type.
single	Single-precision floating point: Usually requires one-half the storage of a double-precision floating point.
int8	Signed 8 bit integer.
uint8	Unsigned 8 bit integer.
int16	Signed 16 bit integer.
uint16	Unsigned 16 bit integer.
int32	Signed 32 bit integer.
uint32	Unsigned 32 bit integer.
boolean	Logical data: False is 0, True is 1. If the Data Type Conversion block is set to **boolean**, it converts any nonzero input to 1.

filters can be improved through the use of appropriate data types. Also, if a model will be used with Real-Time Workshop, significant performance improvements can result from using more compact data types, where possible. Additionally, the Combinatorial Logic and Logical Operator blocks (see Chapter 6) can be configured via the **Simulation:Parameters** dialog box to require Boolean signals.

In a Simulink model, signals and block parameters can be set to any data type. To set a block parameter to a particular data type, use the type-casting notation

type(value), where type is any of the data types listed in Table 4.4, and value is the value assigned. For example, to set a block parameter to a 16-bit integer with the value 25, enter in the block dialog box field for the parameter.

```
int16(25)
```

To convert a signal to a particular data type, use the Data Type Conversion block from the Signal Attributes block library.

You can configure a model such that the data types at each block port are displayed. Choose **Format:Port Data Types** from the model window menu bar.

4.10 CONFIGURING THE SIMULATION

A Simulink model is essentially a computer program that defines a set of differential and difference equations. When you choose **Simulation:Start** from the model window menu bar, Simulink solves that set of differential and difference equations numerically, using one of its differential equation solvers. Before you run a simulation, you may set various simulation parameters, such as the starting and ending time, simulation step size, and various tolerances. You can choose among several high-quality integration algorithms. You can also configure Simulink to acquire certain data from the MATLAB workspace, and to send simulation results to the MATLAB workspace.

To illustrate the basic idea, consider the model shown in Figure 4.14. This model represents the differential equation

$$\dot{x} = 2$$

We set the Integrator block **Initial condition** field to 1. Next, we choose **Simulation:Parameters** from the model window menu bar, and set **Start time** to 0, and set **Stop time** to 5. Then choose **Simulation:Start** from the model window menu bar. Simulink will numerically evaluate the value of the integral to solve

$$x(\tau) = 1 + \int_0^\tau 2\,dt$$

and plot the value of $x(\tau)$ along the interval from 0 to 5. A numerical integration algorithm that solves this kind of problem (frequently called an initial value problem) is referred to as an *ordinary differential equation solver*, or just *solver* for the sake of brevity.

To set simulation parameters, choose **Simulation:Parameters** from the menu bar of the model window, which opens the Simulation parameters dialog box

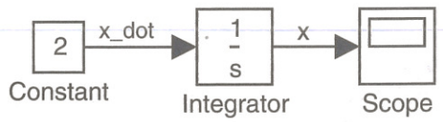


FIGURE 4.14: Simulink model of first-order system

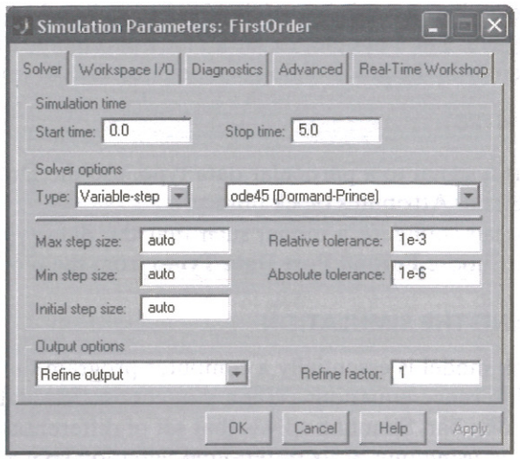


FIGURE 4.15: Simulation parameters dialog box

(Figure 4.15). The Simulation parameters dialog box contains four tabbed pages: **Solver**, **Workspace I/O**, **Diagnostics**, and **Advanced**. If Real-Time Workshop is installed, there will be a fifth tabbed page titled **Real time Workshop**. The **Solver page**, illustrated in Figure 4.15, selects and configures the differential equation solver. The **Workspace I/O** page contains optional parameters that permit you to acquire simulation initialization data from the MATLAB workspace and to send certain simulation data to the MATLAB workspace. The **Diagnostics** page is used to select diagnostic modes, which are useful for troubleshooting certain simulation problems. We'll discuss each of these four pages in detail, next.

4.10.1 Solver Page

The **Solver** page consists of three sections. The first, Simulation time, contains fields in which to enter the start and stop times. **Start time** defaults to 0, and **Stop time** defaults to 10. The Solver options section contains fields to select the differential equation solver (numerical integration algorithm) and to set parameters that control the integration step size. The solvers are grouped in two categories: variable-step and fixed-step. Several different integration algorithms are available for each category. If a variable-step solver is chosen, there are fields to select the maximum integration step size, the initial integration step size, and absolute and relative tolerances. If a fixed-step solver is chosen, there is a single field in which to enter the step size. The Output options section controls the time spacing of points in the simulation output trajectory.

Solver Type. Simulink provides several ordinary differential equation solvers. The majority of these solvers are the result of recent numerical integration research, and they are among the fastest and most accurate methods available. Detailed descriptions of the algorithms are available in the paper by Shampine [2] and Hosea [3].

It is generally best to use the variable-step solvers, as they continuously adjust the integration step size to maximize efficiency, while maintaining a specified accuracy. The Simulink variable-step solvers can completely decouple the integration step size and the interval between output points, so it is not necessary to limit the step size to get a smooth plot or to produce an output trajectory with a predetermined fixed step size. The available solvers are listed in Table 4.5. We will discuss them in more detail in Chapter 13.

TABLE 4.5: Simulink Solvers

Solver	Characteristics
ODE45	Excellent general-purpose one-step solver. Based on the Dormand–Prince fourth/fifth-order Runge–Kutta pair. ODE45 is the default solver, and it is usually a good first choice.
ODE23	Uses the Bogacki–Shampine second/third-order Runge–Kutta pair. It sometimes works better than ODE45 in the presence of mild stiffness. It generally requires a smaller step size than ODE45 to get the same accuracy.
ODE113	Variable-order Adams–Bashforth–Moulton solver. Because ODE113 uses the solutions at several previous time points to compute the solution at the current time point, it may produce the same accuracy as ODE45 or ODE23 with fewer derivative evaluations, and thus perform much faster. It is not suitable for systems with discontinuities.
ODE15S	Variable-order multistep solver for stiff systems. It is based on recent research using numerical difference formulas. If a simulation runs extremely slowly using ODE45, try ODE15S.
ODE23S	Fixed-order one-step solver for stiff systems. Because ODE23S is a one-step method, it is sometimes faster than ODE15S. If a system appears to be stiff, it is a good idea to try various stiff solvers to determine which one performs best.
ODE23T	This is a one-step solver for stiff systems, based on trapezoidal integration. ODE23T and ODE23TB are variants of the same method. ODE23T is somewhat faster but also less stable.
ODE23TB	This is a variant of ODE23TB using backward differentiation formulas for error estimation. This version is more stable than ODE23T but is also somewhat slower.
Discrete	This is a special solver for systems that contain no continuous states.
ODE5	It is a fixed-step version of ODE45.
ODE4	Class fourth-order Runge–Kutta formulas using a fixed step size.
ODE3	Fixed-step version of ODE23.
ODE2	Fixed-step second-order Runge–Kutta method, also known as Heun's method.
ODE1	Euler's method using a fixed step size.

Output Options. The Output options section of the **Solver** page works in conjunction with the variable-step solvers to control the spacing between points in the output trajectory. Output options do not apply to the fixed-step solvers. The **Output options** field contains a list box with three choices: `Refine output`, `Produce additional output`, and `Produce specified output only`.

Choose `Refine output` to force the solver to add intermediate points between the solution points for successive integration steps. Simulink computes the intermediate points using interpolation, which is much faster than using reduced integration step size. `Refine output` is a good choice if the output trajectory needs to appear smoother, but there is no need for a fixed spacing between points. If `Refine output` is chosen, there will be an additional input field labeled **Refine factor**. **Refine factor** must be an integer. Simulink divides each integration step into **Refine factor** output steps, so, for example, if **Refine factor** is set to 2, the midpoint of each integration step will be added to the output trajectory.

`Produce additional output` permits you to force Simulink to include certain time points in the output trajectory, in addition to the solution points at the end of each integration step. If `Produce additional output` is selected, there will be an additional field labeled **Output times**. This field must contain a vector listing the additional times for which output is requested. For example, if it is necessary to include the output at 10-second intervals, and the value of **Start time** is 0 and **Stop time** is 100, **Output times** should contain `[0:10:100]`.

Choose `Produce specified output only` if it is necessary to produce an output trajectory containing only specified time points. For example, you may wish to compare several trajectories to evaluate the effect of changing a parameter. If `Produce specified output only` is selected, there will be an additional field labeled **Output times**, which must contain a vector of the desired output times.

4.10.2 Workspace I/O Page

The **Workspace I/O** page (Figure 4.16) permits you to acquire simulation input from the MATLAB workspace, and to send output directly to the MATLAB workspace. The page consists of three sections: Load from workspace, Save to workspace, and Save options. We'll discuss each section in detail.

Simulink Internal State Vectors. Before discussing the Load from workspace and Save to workspace sections of the **Workspace I/O** page, we should briefly discuss Simulink internal state variables. A Simulink model can be thought of as a set of simultaneous first-order, possibly nonlinear, differential and difference equations. In addition to the state variables associated with each integrator block, there are implicitly specified state variables associated with transfer function blocks, state-space blocks, certain nonlinear blocks, certain discrete blocks, and many of the blocks in the Extras block library. It is frequently useful to have access to a model's state variables, and Simulink provides mechanisms to facilitate this. Use of the **Workspace I/O** page is probably the easiest method with which to access a model's state variables. Accessing a model's state variables, and, in particular, identifying all of a model's state variables, are discussed further in Chapter 8.

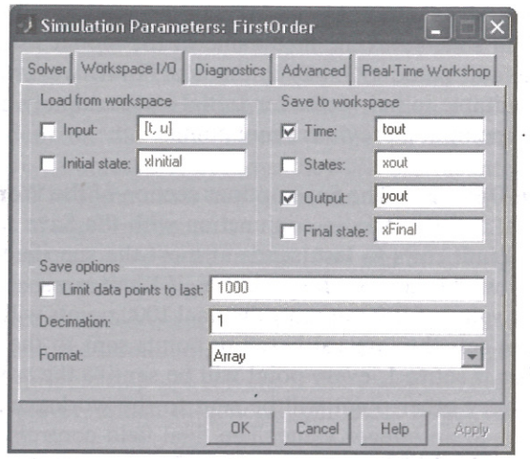


FIGURE 4.16: **Workspace I/O** page

Load From Workspace. Selecting the **Input** check box causes Simulink to take the input time points and values of the input variables from the MATLAB workspace. **Input** works in conjunction with the Inport block found on the Ports and Subsystems block library. Inport blocks may be configured to accept scalar or vector data. Set the name of the time and input matrices in the Input field. The first matrix (default name `t`) is a column vector of time values, and the second matrix (default name `u`) consists of one column for each input variable, with a row corresponding to each row in the time matrix. If there is more than one Inport block, the columns of the input matrix are ordered corresponding to the number assigned to the Inport blocks. So, the first column corresponds to the lowest numbered Inport block, and the last column corresponds to the highest numbered Inport block. For a vector Inport block, there must be one column in the `u` matrix for each element of the input vector.

Selecting the **Initial state** check box of the Workspace I/O forces Simulink to load the initial values of all internal state variables from the MATLAB workspace. All Simulink state variables have default initial values, in most cases 0. States associated with Integrator blocks may be initialized to any value using the block's dialog box. Specifying initial states on the Workspace I/O page overrides any default initialization values, including initial values set in an Integrator block's dialog box. **Initial state** sets initial value of a model's state vector to the values in the specified input vector (default name `xInitial`), which must be defined in the MATLAB workspace when the simulation begins. The initialization vector must be of the same size as the model's state vector but may be either a row or column vector.

Save to Workspace. The Save to workspace section contains four fields, each activated with a check box. **Time** sends the independent variable to the specified workspace matrix (default name `tout`). **States** sends all of the model's state variables to the specified MATLAB workspace matrix (default name `xout`). **Output** works in

conjunction with Outport blocks in a manner analogous to Inport blocks, discussed above. **Final state** saves the final value of the model's state vector in the specified matrix (default name `xFinal`) in the MATLAB workspace. The output of **Final state** is a suitable initial vector for **Initial state** and may be used to restart a model at the final point of a previous simulation.

Save Options. The Save options section of the **Workspace I/O** page contains three fields, and it works in conjunction with the Save to workspace section. The first field, **Limit rows to last**, sends at most the specified number of points to the MATLAB workspace. So, for example, if **Limit rows to last** is checked and set to the default value of 1000, at most the final 1000 points will be sent to the workspace. **Decimation** sets the interval between points sent to the MATLAB workspace. If **Decimation** is set to 1, every point will be sent to the workspace. If **Decimation** is set to 2, every other point will be sent to the workspace, and so on. **Decimation** must be set to an integer value. The final field controls the format of the output variables in the MATLAB workspace. The default, `Matrix`, saves the data in a standard MATLAB matrix. There are also options for saving the data in the form of MATLAB structures with or without a time vector.

4.10.3 Diagnostics Page

The Diagnostics page (Figure 4.17) allows you to select the action taken for various exceptional conditions and also includes options to control automatic block output consistency checking, bounds checking, and simulation profiling. There are three choices for the response to each of the exception conditions. The first choice, `None`, instructs Simulink to ignore the corresponding exception. The second choice, `Warning` (the default choice), causes Simulink to issue a warning message each time the corresponding exception occurs. The final choice, `Error`, causes Simulink to abort the simulation and issue an error message whenever the corresponding

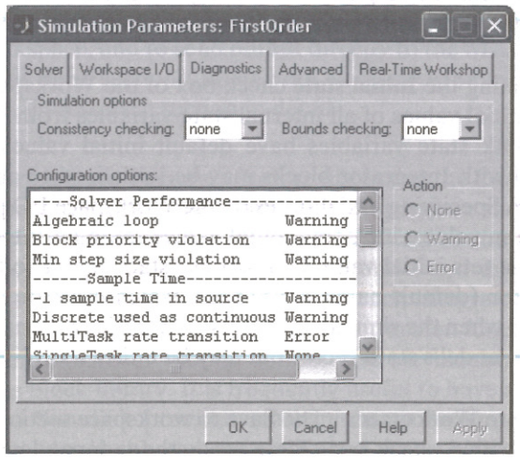


FIGURE 4.17: **Diagnostics** page

exception occurs. The exceptions are detected when a model is executed. A brief explanation for each exception is available from the Simulink Help browser. We'll briefly discuss a few of the more common exceptional conditions here.

An **Algebraic loop** is an exception in which a block's input at a given instant of time is dependent on the same block's output at the same instant of time. Algebraic loops are troublesome, because they can significantly reduce the speed of a simulation, and in some cases can cause the simulation to fail to execute. A detailed discussion of algebraic loops is presented in Chapter 13. It is usually best to set the algebraic loop response to `Warning`. If an algebraic loop is discovered, and if performance is acceptable, change the response to `None`.

A **Min step size violation** occurs when the solver attempts to use an integration step size smaller than the minimum. It is not possible to change the minimum step size for any of the variable step size solvers. If this exception occurs, you can change to a higher-order solver, which in general will use a larger integration step size. Your other choice is to increase the absolute and relative tolerances on the **Solver** page. **Min step size violation** should always be set to `Warning` or `Error`, because it indicates the simulation is not producing the expected accuracy.

Unconnected inputs, outputs, and signal lines are almost always errors. Therefore, the corresponding options should be set to `Warning` or `Error`. If it is necessary to leave an input or output unconnected, use Ground and Terminator blocks.

A **Data overflow** exception occurs when the magnitude of a signal or internal variable exceeds the limit for its data type. Data overflow is almost always a serious error, and therefore, **Data overflow** should be set to `Warning` or `Error`.

Consistency checking is a debugging feature that detects certain programming errors in custom blocks. Consistency checking is not needed with standard Simulink blocks, and it causes a simulation to run much slower. Ordinarily, **Consistency checking** should be set to `None`.

4.10.4 Advanced Page

The **Advanced page** (Figure 4.18) allows you to control certain simulation options that affect the speed of execution and memory usage. Included are options to disable zero crossing detection and to relax the rules for Boolean-type checking in certain blocks. These options are set to either `On` or `Off`.

A number of Simulink blocks exhibit discontinuous behavior. For example, the output of the Sign block, located in the Math Operations block library, is 1 if its input is positive, 0 if its input is zero, and -1 if its input is negative. Thus, the block exhibits a discontinuity at zero. If a variable step size solver is in use, Simulink will adjust the integration step size when the input to a Sign block is approaching zero, so that the switch occurs at the right time. This process is called *zero crossing detection*. You can determine whether a block invokes zero crossing detection by referring to the Characteristics table for the block in the online Help system.

Zero crossing detection improves the accuracy of a simulation, but it can cause a simulation to run slowly. Occasionally, a system will fluctuate rapidly about a discontinuity, a phenomenon called *chatter*. When this happens, the progress of the simulation can effectively stop, as the integration step size is reduced to a very small value. If your model runs slowly and includes one or more blocks with intrinsic

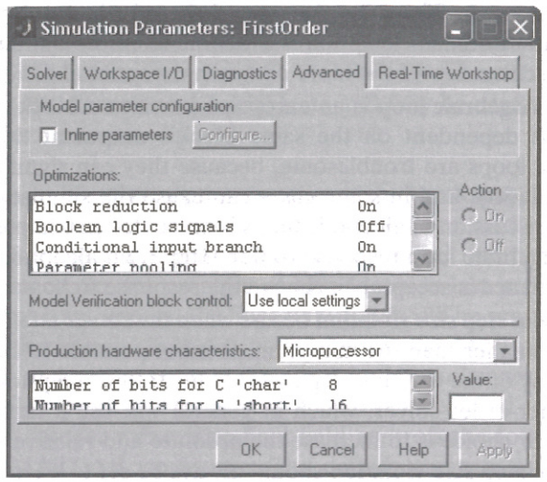


FIGURE 4.18: **Advanced** page

zero crossing detection, disabling zero crossing detection can significantly increase the speed of a simulation. However, this can also adversely affect the accuracy of a simulation, so it is best used only as a tool to verify that chatter is occurring. If disabling zero crossing detection dramatically improves the speed of a simulation, you should locate the cause of chatter and correct the problem.

As a consequence of the method by which Simulink optimizes storage, memory buffers are not maintained for every signal line. Ordinarily, this causes no problem. However, in order for floating Scope blocks to display a signal, there must be a buffer allocated for the signal line. Turn Signal storage reuse to `Off` to disable this optimization feature. This should only be done during development and debugging, as there is a performance and memory penalty.

A number of Simulink blocks, for example logic blocks, expect Boolean signals and will produce an error message if the input signals are not Boolean. Setting `Boolean logic signals` to `Off` suppresses these error messages. This feature is provided to ensure compatibility with earlier versions of Simulink, which had only one data type, double-precision floating point. If `Boolean logic signals` is set to `Off`, logic blocks will accept floating point signals. In this case, the value 0.0 is interpreted as False, and anything else is interpreted as True.

4.11 RUNNING A SIMULATION

You can control the execution of the model using the **Simulation** pull-down menu on the model window menu bar or by using the model window toolbar. To start the simulation, select **Simulation:Start** or click the **Start** button. You can stop the simulation at any time using **Simulation:Stop** or the **Stop** button on the model window toolbar. Choose **Simulation:Pause** to temporarily halt execution. Then, choose **Simulation:Continue** to resume or **Simulation:Stop** to permanently halt

execution of the model. It is also possible to run a simulation from the MATLAB command line; this method will be discussed in Chapter 8.

While the simulation is executing, you can change many parameters. For example, you can change the gain of a Gain block, choose a different solver, or change integration parameters, such as minimum step size. You can also select a signal line that will become the input to a floating Scope block. This permits you to check various signals as the simulation progresses.

4.12 PRINTING A MODEL

There are a variety of options for printing Simulink models. The simplest is to send the model directly to a printer. However, it is also possible to embed the model in a word-processing document or other file, either by copying it to the clipboard (Microsoft Windows) or saving it as an Encapsulated PostScript file.

4.12.1 Printing to the Printer Using Menus

The fastest way to get a printed copy of a model is to send it directly to the printer. On Microsoft Windows, you can choose **File:Print** or click the **Print** button on the model window toolbar. The Print Model dialog box (Figure 4.19) will be displayed. In addition to standard print controls, the Print Model dialog box contains an Options section that controls which parts of a hierarchical Simulink model (see Chapter 7 for information on hierarchical models) are printed. There are also options to produce a log of printout pages and to add frames to each page.

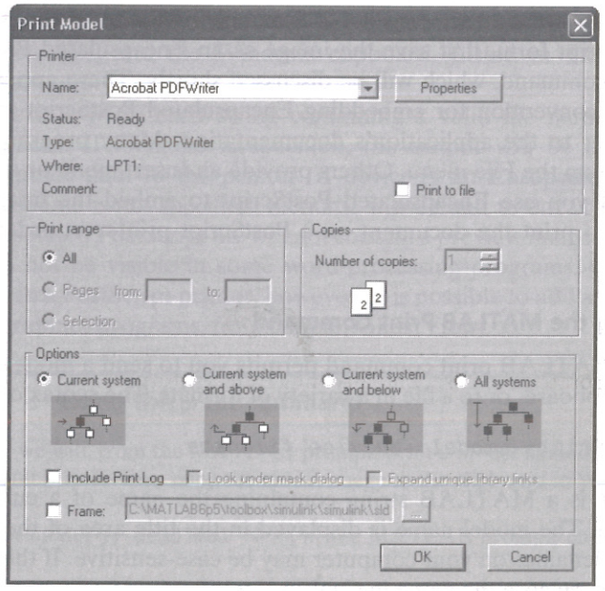


FIGURE 4.19: **Print Model** dialog box

Select **Frame** to add a background frame to each printout page. When **Frame** is selected, the frame file field is available. A frame file contains all information needed to produce the frame. Frame files can be created and edited using the frame file editor, which may be accessed by entering the command

```
frameedit
```

at the MATLAB prompt. For details on using the frame editor, enter the command

```
doc frameedit
```

at the MATLAB prompt.

4.12.2 Embedding the Model in a Document

Modern documentation applications such as word processors, presentation programs, desktop publishing programs, drawing programs, and even spreadsheets, allow you to insert Simulink model images in documents. Once you have embedded an image in a document, you can use the graphics capabilities of the documentation application to resize the image and to further annotate the model. Simulink model images may be embedded as bitmaps, as Windows metafiles, or as Encapsulated PostScript files. To embed an image as a metafile using Microsoft Windows, select **Edit:Copy Model** from the model window menu bar. The entire model window will be copied to the clipboard. Next, make the target document active, and use the procedure for the documentation application to embed the image. For most Microsoft Windows programs, the procedure is to choose **Edit:Paste Special**, then select Metafile (sometimes listed as Picture).

To embed the image of a Simulink model in a document in Encapsulated PostScript form, first save the image as an Encapsulated PostScript file using the print command, which will be discussed shortly. There appears to be no standard menu convention for embedding Encapsulated PostScript images, so you'll need to refer to the application's documentation. Many programs provide an Import choice on the File menu. Others provide an Insert menu or a Graphics menu. Note that if you use Encapsulated PostScript to embed the image, you will probably have to print the document on a PostScript printer in order for the image to be printed.

4.12.3 Using the MATLAB Print Command

The MATLAB print command permits you to send a model image to a printer, to the clipboard, or to a file in a variety of formats. The syntax of the print command is

```
print -smodel -ddevice filename
```

model is a MATLAB string containing the name of a currently open Simulink model. The model name is displayed in the title area of the model window. Note that filenames on your computer may be case-sensitive. If the model name contains spaces, enclose the name in single quotes:

```
print -s'Spring Mass System' -ddevice filename
```


TABLE 4.6: Device Codes for the `Print` Command

Device	Description
`ps`	PostScript
`psc`	Color PostScript
`ps2`	Level 2 PostScript
`psc2`	Level 2 color PostScript
`eps`	Encapsulated PostScript (must go to a file)
`epsc`	Color Encapsulated PostScript (must go to a file)
`eps2`	Encapsulated Level 2 PostScript (must go to a file)
`epsc2`	Color Encapsulated Level 2 PostScript (must go to a file)
`win`	Current printer
`winc`	Current printer, color
`meta`	Metafile format (saved to clipboard if no file name is supplied)
`bitmap`	Bitmap format (saved to clipboard if no file name is supplied)
`setup`	Same as selecting **File:Printer Setup** from the model window menu bar

If the model name contains a carriage return, that is, if it is shown in the window title in two lines, represent the carriage return as its ASCII code (13), enclose each line in single quotes (' '), and the whole name in brackets ([ ]):

```
print -s['Damped' 13 'Spring Mass System'] -ddevice
filename
```

device is a MATLAB string that specifies the type of output device. Devices include printers, files, and the Windows clipboard. Listed in Table 4.6 are the available device types.

filename is a MATLAB string containing a valid file name, and it is an optional argument. If filename is specified, the output will be directed to the specified file rather than to the printer. If device is an Encapsulated PostScript format, and filename is not specified, filename will default to `Untitled.eps`. Note that the Encapsulated PostScript file will not contain a preview image, and therefore, the image will not be visible in some word-processing programs. The image will print correctly to a PostScript printer, however. It is possible to add a preview image using certain graphics programs, for example, GhostView.

EXAMPLE 4.3 Using the `print` command

In this example, we will, from the MATLAB prompt, print a model named `xy-demo.mdl` to the current default printer, then copy the model image to the clipboard in Windows Metafile format.

First, open model `xy-demo.mdl`. Next, at the MATLAB prompt, enter the following command:

```
print -s'xydemo'
```

The model will be printed. Next, enter the command

```
print -s'xydemo' -d'meta'
```

The model is now copied to the clipboard. It can be pasted into a word-processing document or spreadsheet.

4.13 MODEL BUILDING SUMMARY

In Table 4.7, the model-building procedures discussed in Chapter 3 and Chapter 4 are summarized. The following terms are used:

Drag	Click and hold the left mouse button; drag object to new location.
Shift-click	Holding down the Shift key, click with the left mouse button.
Shift-drag	Holding down the Shift key, drag with the left mouse button.
Control-drag	Holding down the Control key, drag with the left mouse button (Windows only).
Right-drag	Drag using the right mouse button (Windows only).

TABLE 4.7: Summary of Model-building Operations

Operation	Method
Select object (block or signal line)	Click on the object with the left mouse button.
Select another object	Shift-click the additional object (or click the center mouse button on X Windows).
Select with bounding box	Click with the left mouse button at the location of one corner of the bounding box. Continuing to depress the mouse button, drag the bounding box to enclose the desired area.
Copy block from block library or another model	Select block, then drag it to the model window.
Flip block	Select block, then **Format:Flip Block**. Shortcut: Control-F.
Rotate block	Select block, then **Format:Rotate Block**. Shortcut: Control-R.
Resize block	Select block, then drag handle.
Add drop shadow	Select block, then **Format:Show Drop Shadow**.
Edit block name	Click on name.
Hide block name	Select name, then **Format:Hide Name**.
Flip block name	Select name, then **Format:Flip Name**. Shortcut: Drag name to new location.
Delete object	Select object, then **Edit:Clear**. Shortcut: Delete key.
Copy object to clipboard	Select object, then **Edit:Copy**. Shortcut: Control-C.
Cut object to clipboard	Select object, then **Edit:Cut**. Shortcut: Control-X.

TABLE 4.7: Summary of Model-building Operations (Cont)

Operation	Method
Paste from clipboard	**Edit:Paste**. Shortcut: Control-V.
Draw signal line	Drag from output port to input port.
Draw signal line in segments	Drag from output port to first bend. Release mouse button. Drag from first bend to second bend, and so on.
Branch from signal line	Control-drag from point of branch. Shortcut: Right-drag, starting from the point of branch.
Split signal line	Select line. Shift-drag the new vertex.
Move line segment	Select and drag segment.
Move line segment vertex	Select signal line and drag vertex marker.
Label signal line	Double-click line, then enter text.
Move signal line label	Drag the label to the desired location.
Copy signal line label	Control-drag the copy of the label to desired location. Shortcut: Right-drag to desired location.
Delete one occurrence of signal line label that has multiple occurrences	Shift-click label, then press Delete key.
Delete all occurrences of signal line label	Select label, then delete all characters in the label.
Propagate signal label	Label signal line onto which you want label propagated with single character <. Then choose **Edit:Update Diagram**.
Add annotation to model	Double-click at location of annotation, and type text.

4.14 SUMMARY

In this chapter, we described procedures for editing and annotating Simulink models. We also discussed configuring and printing a Simulink model. In the next two chapters, we will discuss using these procedures to model continuous, discrete, and hybrid systems.

REFERENCES

[1] Meirovitch, Leonard, *Introduction to Dynamics and Control* (New York: John Wiley & Sons, 1985) 16–17.

[2] Shampine, Lawrence F., and Reichelt, Mark W., "The MATLAB ODE Suite," The MathWorks, Inc. (Natick, Mass., 1996). This technical paper is available directly from The MathWorks. It provides a detailed discussion of the MATLAB differential equation solvers that are available from within Simulink.

[3] Hosea, M.E., and Shampine, Lawrence F., "Analysis and Implementation of TR-BDF2," *Applied Numerical Mathematics* 20, 1–2, (1996), 21–37.