

The laboratories are intended to introduce students taking *SYSC 3600: Systems and Simulation* to some of the techniques used to simulate continuous-time linear systems; and, in conjunction with the lectures, to develop the students' understanding of the dynamic behaviour of linear systems.

---

## 1 Simulation

Simulation has been defined as “the act of representing some aspects of the real world by numbers or symbols which may be easily manipulated to facilitate their study”. It has also been defined as “the development and use of models for the study of the dynamics of existing or hypothesized systems”; and again as “the development and use of models to aid in the evaluation of ideas and the study of dynamic systems or situations”. Whichever definition one likes, simulation is the technique by which understanding of the behaviour of a physical system is obtained by making measurements or observations of the behaviour of a model representing that system.

Frequently, simulation studies are done to check and optimize the design of a system before its construction. Simulation is also used in model building where the object of the study is to determine a model which adequately represents a given system.

The word “model” is closely related in meaning to the word “analog” which has been used to describe types of calculating aids in which there is a close correspondence or relationship between parts of the original system and parts of the calculating aid. Many models consist of a set of equations which describe the behaviour of the system being modeled. Continuous-time systems are modeled by sets of differential equations. Typical examples of these sorts of systems form the basis of analysis in each of the engineering fields. The basic equations are often identical, even though the physical situations being modeled can be quite different in substance.

Setting up a model, based on mathematics, and solving it in a manner by which some feel or intuition or understanding can be gathered about the basic system itself is what simulation is all about.

Simulation is one of the most valuable techniques available to the engineer. Simulation studies are of great value in avoiding costly design errors, in ensuring safe designs, in choosing between alternative possibilities, for training and experience, especially with situations that cannot exist in reality without unacceptable risks and costs, and provide the engineer with the possibility of working with the systems that have not yet been completed, the ability to modify systems whose operation cannot be interrupted and to reproduce a wide range of operating conditions for systems which have failed.

## 2 Introduction to SIMULINK

SIMULINK is a software package developed by “The MathWorks Inc.”. It is a powerful tool to simulate complex dynamic systems. The SIMULINK software is really just a module within the MATLAB software package which is also sold by “The MathWorks Inc.”. MATLAB is a powerful mathematical toolbox which operates on vectors and matrices and has become one of the standard mathematical software packages for research engineers and scientists.

---

<sup>1</sup>This tutorial prepared for SIMULINK® v5.0 under Windows XP®.

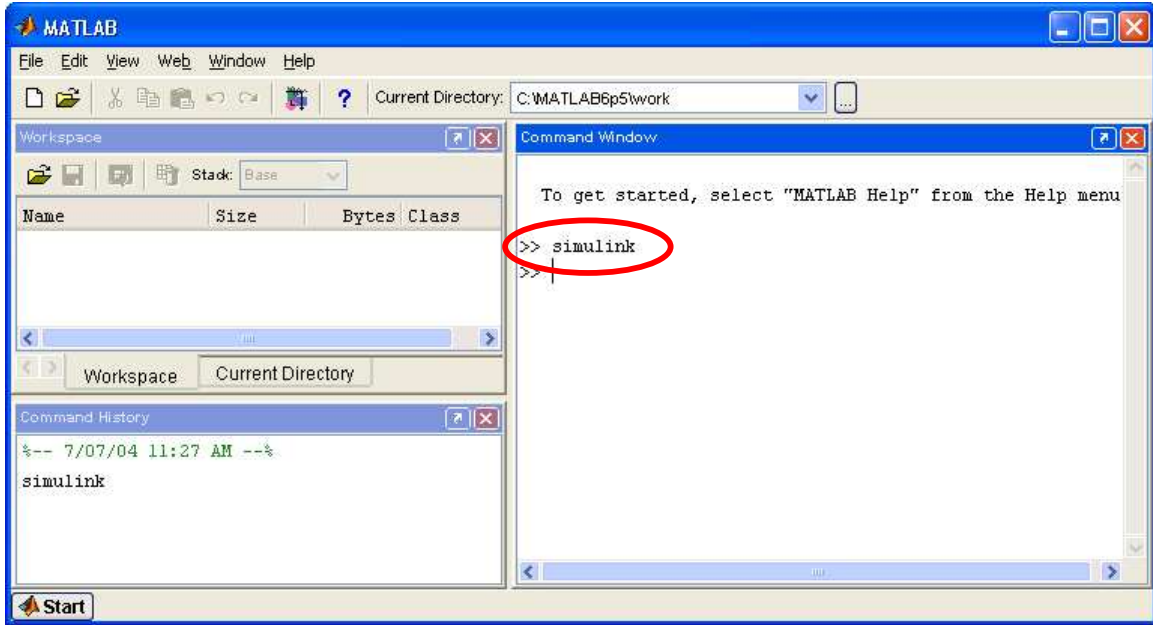


Figure 1: SIMULINK window.

The usefulness of SIMULINK is that one can draw out the *block diagram* or *simulation diagram* that describes a dynamic system and then simulate the system. SIMULINK will create the differential equations that represent the simulation diagram and one can then use the powerful MATLAB tools to analyze the system. For example, one can find the transfer function, the state space equations, eigenvalues and eigenvectors, poles and zeros, plot Bode diagrams, etc. For the purpose of these laboratories you will only use a small portion of the capabilities of MATLAB/SIMULINK.

To start SIMULINK, you must first launch MATLAB which will bring up a window similar to that shown in Fig. 1. After launching MATLAB you will notice that within the MATLAB window there is a Command Window where MATLAB commands can be executed. From within this Command Window, type

```
simulink
```

to start the SIMULINK portion of the MATLAB software package. Doing so should bring up the SIMULINK Library Browser window similar to that shown in Fig. 2<sup>2</sup>. The SIMULINK Library Browser window is organized to show the major categories of simulation blocks as well as other installed sets of specialized simulation blocks. For the labs, you will require simulation blocks from the following categories:

- (a) *Sources*: Sources are blocks that supply some form of input into your simulation. Within this course we will focus primarily on the step function, ramp function, sinusoids, and constants. Note that the impulse function is not available since it is a mathematical construct that is not physically realizable.
- (b) *Sinks*: Sinks are blocks that gather results from your simulation to be displayed as plots, values, or stored to file or to the MATLAB variable workspace for later processing.
- (c) *Continuous*: Consists of some specialized continuous-time functions such as the integrator, the differentiator (which we should normally avoid in our simulations), and transfer function blocks defined through the differential equation coefficients, poles/zeros, or state space representations.

<sup>2</sup>The appearance may differ for different versions of MATLAB as well as which additional toolboxes are installed.

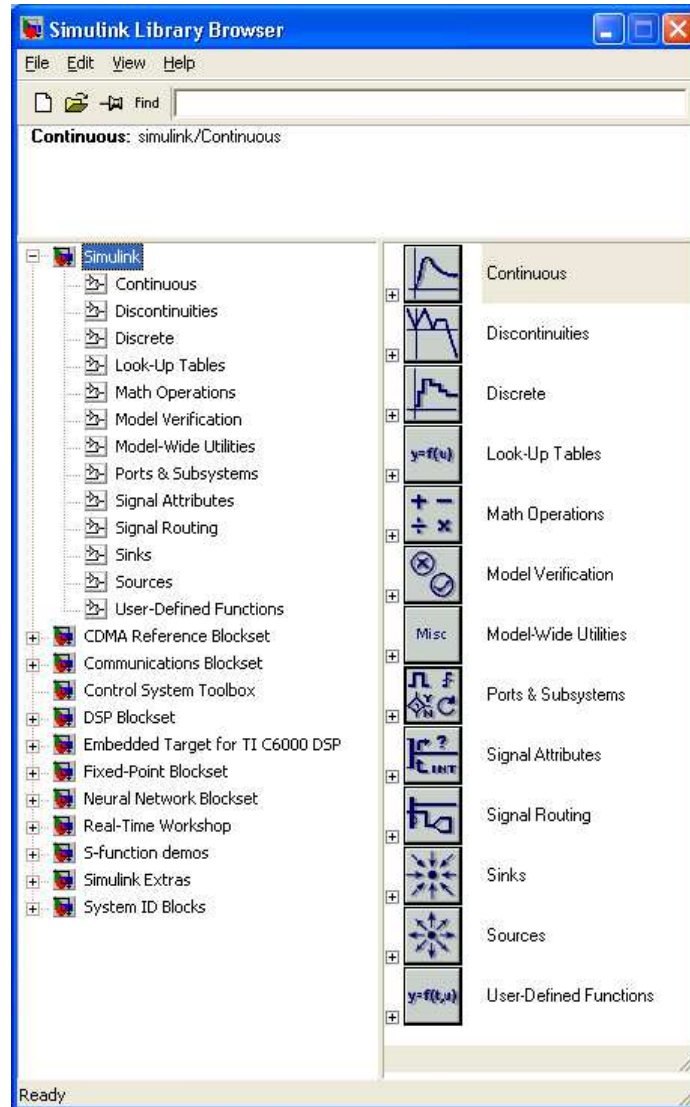


Figure 2: SIMULINK Library Browser window.

- (d) *Math*: Consists of the mathematical “glue” that will allow us to connect blocks to implement differential equations. The blocks of primary interest are the sum unit for summing multiple links and the gain for scaling the amplitude of a particular link connection.
- (e) *Signals & Systems*: Consists of blocks that will help manipulate and group other blocks as well as manipulate and group the signals. Some of these blocks can be used to make subsystems and others are useful for combining multiple streams of signals together for other uses such as plotting.
- (f) *Nonlinear*: Consists of some components for performing nonlinear operations. In this course we focus on linear elements and linear systems, but we will have at least one lab where some nonlinear components will be necessary.
- (g) *Discrete*: Consists of blocks for implementing discrete-time functions. The focus in this course is on continuous-time systems so little (if any) will be used from this section. The discrete-time blocks are important in areas such as digital signal processing when simulating digital filters.

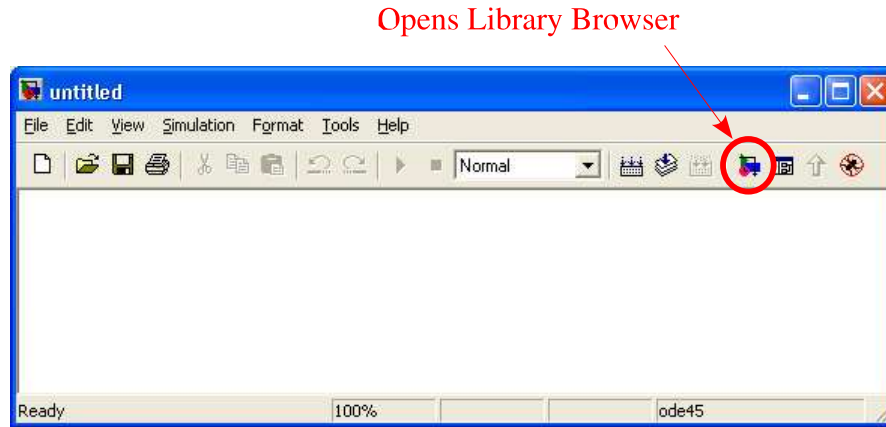


Figure 3: Empty window for the simulation model to be implemented.

### 3 SIMULINK Tutorial #1: Converting Celsius to Fahrenheit

Let's start by consider a very simple equation to solve so that we can learn some of the basics of SIMULINK. Consider the equation

$$^{\circ}\text{F} = \frac{9}{5}^{\circ}\text{C} + 32 \quad (1)$$

which describes how to convert temperatures from Celsius,  $^{\circ}\text{C}$ , to Fahrenheit,  $^{\circ}\text{F}$ . Let's build a simple SIMULINK model that will implement this equation.

#### 3.1 Building a SIMULINK model

An empty simulation model window must first be opened before we can setup our model to perform the Celsius to Fahrenheit scale conversion. A new simulation model is opened by selecting

File→New→Model

in the SIMULINK Library Browser window (see Fig. 2). A new window, as shown in Fig. 3, will open for your new simulation model and will be labeled "untitled". It is within this window that you will construct your simulation model. We will start by setting up blocks to perform the mathematics of Eq. 1 as shown in Fig. 4.

You can drag-and-drop any of the blocks listed in the SIMULINK Library Browser into your model. Start by finding the Gain and Sum blocks under the Math Operations section of the SIMULINK Library Browser. Then, find the Constant block under the Sources section and drag-and-drop it into your model. One convenient icon, as pointed out in Fig. 3, is the Library Browser icon which when pressed will raise the SIMULINK Library Browser window if it is hiding under other windows.

Now that you have the primary blocks placed in your model, you must connect them appropriately. You can do this by left-clicking and holding on an output port of a unit and dragging a connection to the input port of another unit to make the desired connection. Another way of doing this is to left-click on the output source block and then hold CTRL and left-click on the destination block. Connect the three units in your model such that they implement the math of Eq. 1 as shown in Fig. 4. Note that you can pick up and move the simulation blocks at any time and the connections will stretch as you move the block. Also, if you wish to delete a connection or delete a block, simply left-click on the connection/block and then press delete.

Most blocks within SIMULINK can be configured through its block parameters. Let's start with setting the block parameters for the Gain block. If you double-click on the Gain block the Block Parameters

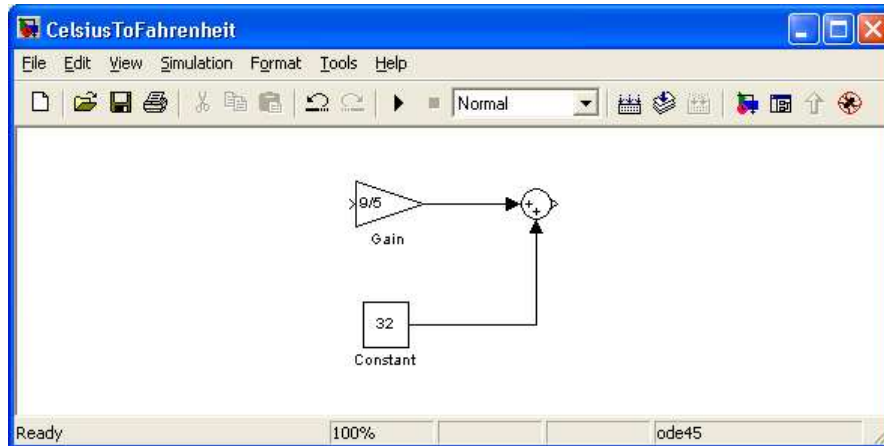


Figure 4: Math for implementing the Celsius to Fahrenheit converter.

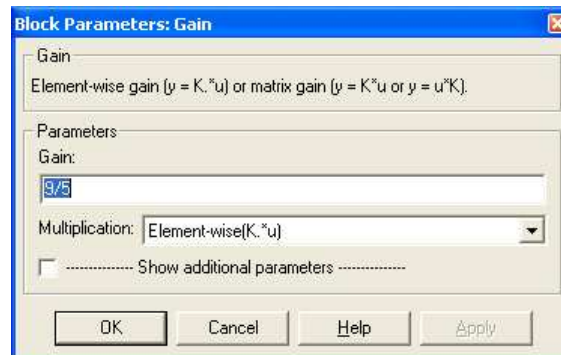


Figure 5: Block parameters for the Gain block.

window for the block will open as shown in Fig. 5. You can set the gain to a constant, or as shown in Fig. 5 to a simple expression. Set the gain of the block to  $9/5$ .

Now open the block parameter window for the Constant source block by double-clicking on it. Set the constant value to 32, which is the constant that must be added to implement Eq. 1. This block will now supply a value of 32 at its output.

The default setup for the Sum block is two summing inputs and one output, so we don't need to change it for implementing Eq. 1. If we needed to change the block parameters for the Sum block, you can similarly open its block parameters window by double-clicking on it which will open the window shown in Fig. 7. A sum unit always has a single output, but the number, type, and placement of the inputs can be changed. The default setting uses the string '|++' where each '+' indicates an input port and the '|' indicates a spacer. The type of port can be either a '+' (addition) or '-' (subtraction) port. The '|' indicates a spacer and has no function in the actual simulation, but is used to change the visual spacing of the ports around the sum unit. Using the spacer '|' in the appropriate spots may make your simulation model easier to read. For instance, a 4 input sum unit could be given as '++++' or similarly given as '+ ++|++' where extra spacing is put between the two middle ports. Having a sum unit where two ports sum and one port subtracts could be given as '+ +-|++' with appropriate spacing using '|' inserted to adjust how the unit is displayed with the model. More complicated sum units could be formed such as '+ +|-|++', etc. Experiment with different configurations, but return the sum unit back to '|++' before proceeding.

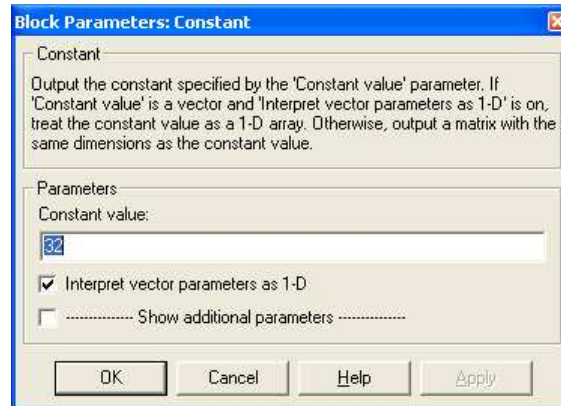


Figure 6: Block parameters for the Constant source block.

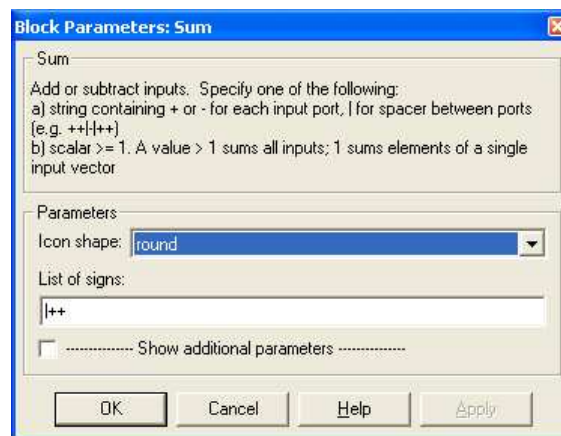


Figure 7: Block parameters for the Sum unit.

The simulation model shown in Fig. 4 implements the math required for the Celsius to Fahrenheit conversion. What we need now to complete the simulation model is some input values to represent the temperatures in Celsius and then a way of observing the output of the model so that we can see the temperatures converted to Fahrenheit. One simple approach would be to supply a constant input using the Constant block. Create another Constant block by either selecting it from under the Sources section of the SIMULINK Library Browser or by copying and pasting the Constant block already available in your model. Connect this Constant block to the input of the system as shown in Fig. 8.

For monitoring the output, we require an appropriate sink for the output. In this example, our input is a constant so we will only obtain a constant for the output. The Display block under Sinks in the SIMULINK Library Browser is appropriate in this case. Place the Display block into your model and connect it to the output of the Celsius to Fahrenheit converter as shown in Fig. 8.

Now that a model is setup, we can simulate the model. This is done by either pressing the simulation button in the model toolbar (the play button circled in red in Fig. 8) or by selecting

Simulation→Start

from the menu. Run the simulation a few times for different values of the input Centigrade temperature in the Constant block and verify that the conversion takes place correctly. Before continuing to the next section, you may wish to save your simulation model using

Runs the simulation

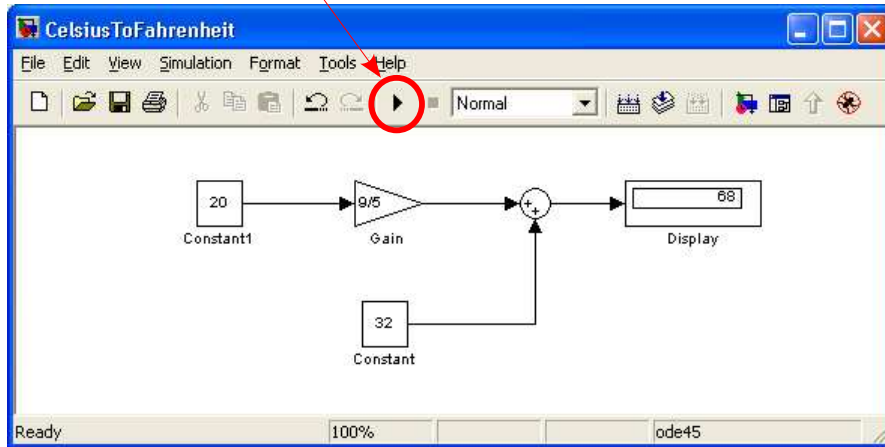


Figure 8: Celsius to Fahrenheit converter with constant input.

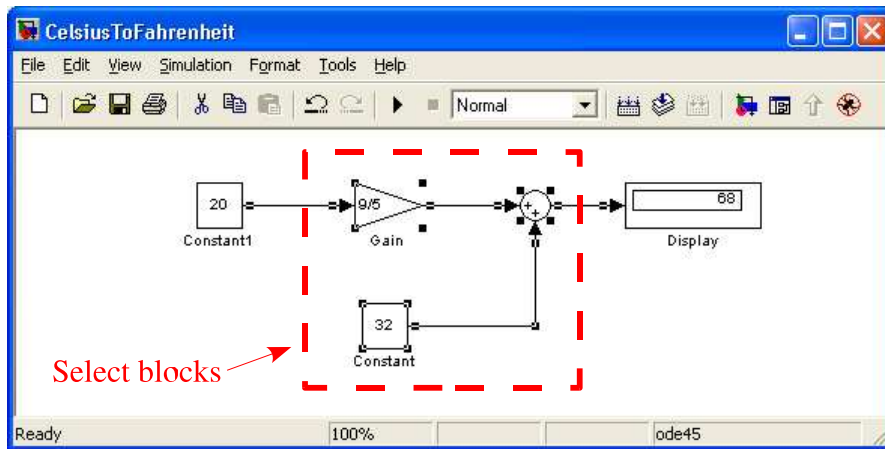


Figure 9: Selecting math portion of Celsius to Fahrenheit converter so that it can be made into a subsystem.

File→Save

from the menu.

### 3.2 Subsystems within simulation models

Since simulation models can become quite complex, it is often convenient to group blocks in a model into subsystems. We will create a subsystem for the math portion of the Celsius to Fahrenheit converter to demonstrate how this is done. Following Fig. 9, select the three blocks that implement the mathematics of Eq. 1. With the blocks selected, choose

Edit→Create Subsystem

from the menu. This will group the selected blocks into a subsystem as shown in Fig. 10. If you double-click on the newly created subsystem, a subsystem model window will open as shown in Fig. 11.

Notice that the math portion of the Celsius to Fahrenheit converter has simply been moved into the subsystem shown in Fig. 11. The only new elements are the input port and the output port required for

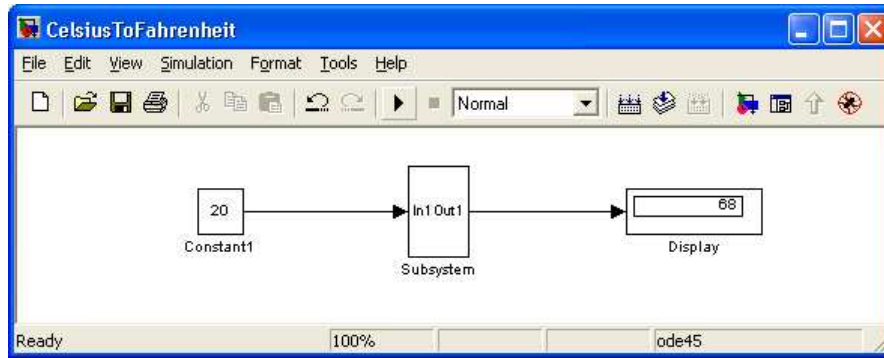


Figure 10: Celsius to Fahrenheit converter grouped as a subsystem.

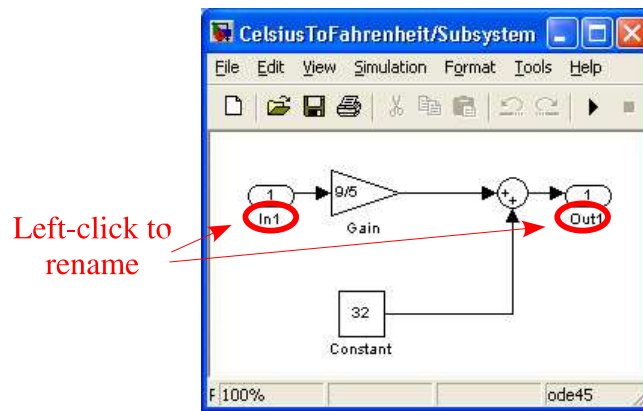


Figure 11: Celsius to Fahrenheit converter subsystem (before relabeling).

passing information into and out of the subsystem. By left-clicking on the name of the input port or output port you can change the names of the ports. For instance, in Fig. 12 the input port has been renamed to *C* and the output port to *F*. Similarly, the label under any block can be changed by left-clicking on the label and editing the label. Notice that this relabeling was also done to Fig. 10 to obtain Fig. 13.

Verify again that your simulation model functions correctly and that Celsius is properly converted to Fahrenheit. You may wish to *save* your model again at this point.

### 3.3 Simulations over time in SIMULINK

A simulation using only a constant input, such as in the last section, does not fully demonstrate the power of SIMULINK. Simulations are normally run with inputs that vary over time such that the response of the model can be observed over time.

#### 3.3.1 Inputting a signal and plotting the output

Let's change the input and output blocks for our model to see how simulations can be done over a certain time frame. First delete the *Constant* source block and the *Display* sink block from the model leaving only the Celsius to Fahrenheit converter subsystem. Next, drag-and-drop the *Signal Generator* block into your model from the *Sources* section of the SIMULINK Library Browser, and the *Scope* block from the *Sinks* section of the SIMULINK Library Browser. Connect these units to the Celsius to Fahrenheit converter as shown in Fig. 14.



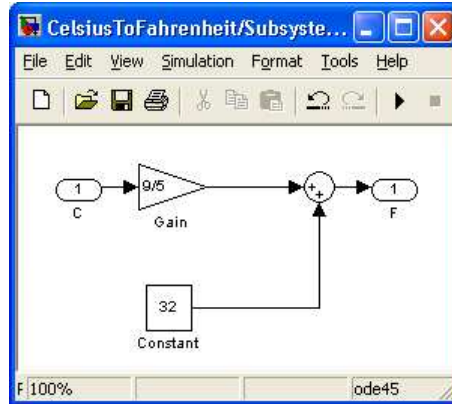


Figure 12: Celsius to Fahrenheit converter subsystem.

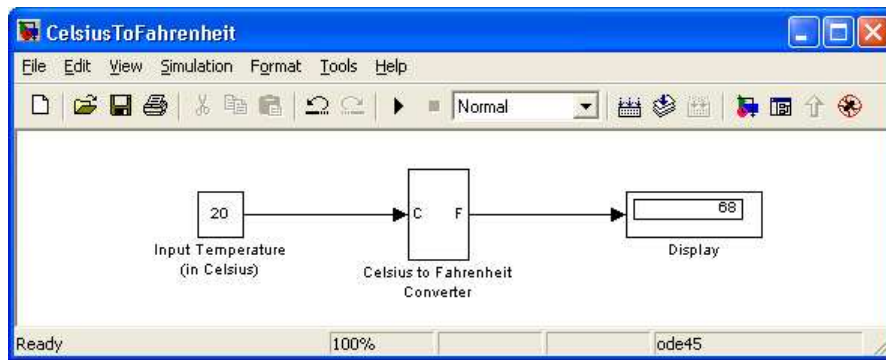


Figure 13: Celsius to Fahrenheit converter with constant input.

The `Signal Generator` block is similar to a function generator you might find in a hardware lab and the `Scope` block is similar to an oscilloscope. Double-click on the `Signal Generator` block to open its block parameters window as shown in Fig. 15. Four different types of signal waveforms can be generated: 1) sine, 2) square, 3) sawtooth, and 4) random. For each signal you can set the amplitude and frequency of the signal. Set the signal generator to be a sine wave with an amplitude of 4 and a frequency of 0.5 Hz.

You can now simulate the model by clicking on the simulation button. The simulation will now use the `Signal Generator`'s sine wave as input for the Celsius to Fahrenheit converter. To view the output of the simulation, double-click on the `Scope` block to bring up a window such as in Fig. 16. For the output plotted for the `Scope` block, the  $x$ -axis is the time axis and the  $y$ -axis is the signal value feed into the scope at that point in time.

### 3.3.2 Zooming on the `Scope`'s plot and controlling the axes

Often the plot given for the `Scope` is not properly scaled. If you left-click on the binoculars icon, the plot will autoscale in the  $x$ - and  $y$ -axis to view the entire signal plotted. The plot in Fig. 16 has been autoscaled to show the entire signal. If you need to zoom in on the plotted signal, you can use the magnifying glass icon and then select a boxed zoom region in the plot. You can also use either the  $x$ -axis or  $y$ -axis magnifying glass icons to zoom in on a portion of the output but only in the  $x$ - or  $y$ -axis directions, respectively.

Another useful aspect of the `Scope` block's plot is the `Save current axis settings` and `Restore saved axis settings` icons. These are useful if you need to zoom into a particular por-

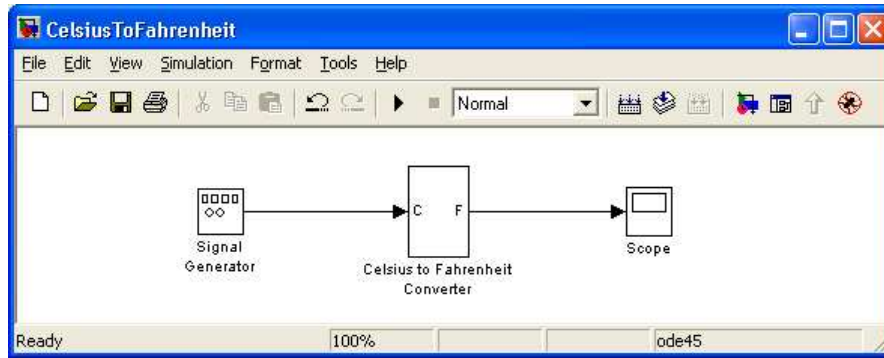


Figure 14: Celsius to Fahrenheit converter with signal generator input.

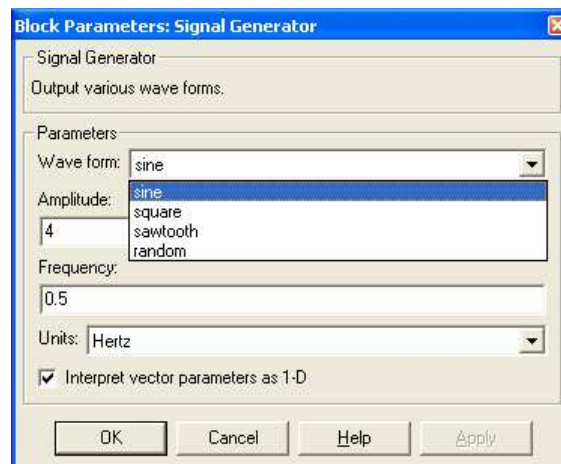


Figure 15: Block parameters for the Signal Generator source block.

tion of the plot and wish to come back to that zoomed setting later on. Try zooming into a portion of your plot and then click the `Save current axis settings` icon. Then, click the `Autoscale` icon (the binoculars icon) to zoom back out. Afterwards, click the `Restore saved axis settings` icon which will cause the plot to zoom back to the axis settings the last time you saved them. This is useful when you simulate your system with different settings and wish to monitor a certain zoomed portion of the results.

### 3.3.3 Changing the start time and stop time of the simulation

The default configuration for the simulation is for the simulation to run from time  $t = 0$  seconds to  $t = 10$  seconds. To set the start and stop times for the simulation, select

`Simulation`→`Simulation Parameters...`→`Model`

from the menu which will give you the window shown in Fig. 17.

As shown in Fig. 17, you can control the start time and the stop time of your simulation. Set the start time to -1.0 seconds and the stop time to 5.0 seconds and then run the simulation. Verify that the simulation results in the `Scope` show that the simulation time is now only from -1 to 5 seconds. You might have to press the binoculars icon again to autoscale the plot.

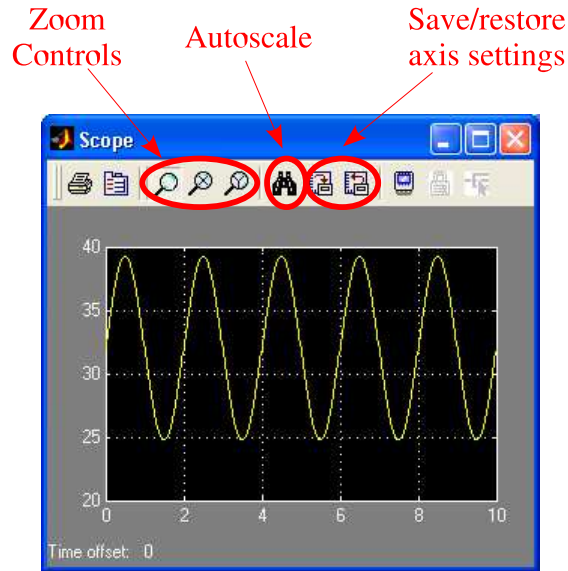


Figure 16: Scope output window.

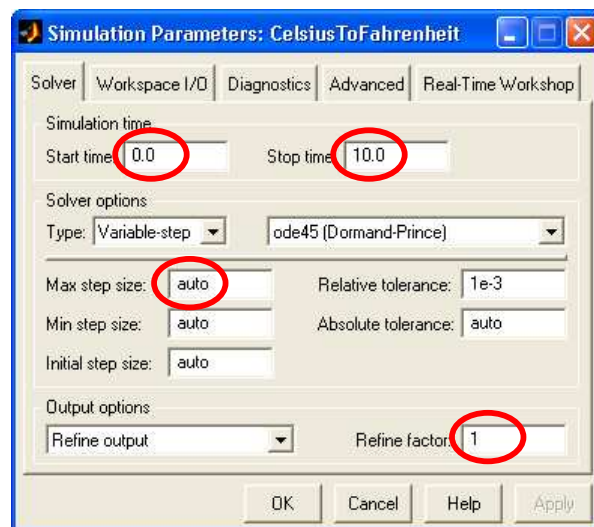


Figure 17: Simulation parameters configuration window.

### 3.3.4 Step size and refinement of sampling interval

After a simulation, you may notice that your resulting plots are not as fine detailed as perhaps expected. For instance, even though your input to the Celsius to Fahrenheit converter is a sine wave, the output appears more jagged than a sine wave. When simulating with continuous-time signals, SIMULINK can only calculate the output at a finite set of time instants. As such, simulations with continuous-time signals that move quickly (such as a sine wave) may produce results that don't appear smooth if SIMULINK chooses unsuitable time intervals between calculations.

The simulation parameters shown in Fig. 17 allow some control over how SIMULINK chooses the time intervals. By default, the Solver option type is set to Variable-step using the ode45 (Dormand-Prince) solver. The ode45 solver is an *ordinary differential equation* (ODE) solver based

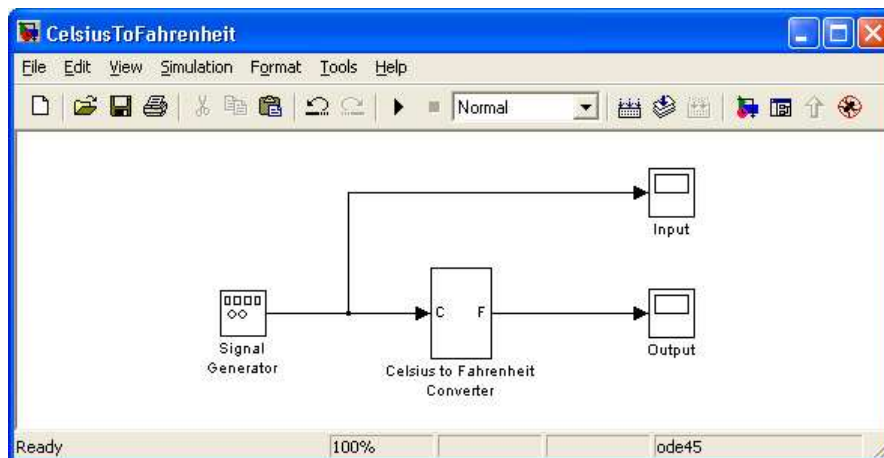


Figure 18: Using multiple `Scope` blocks to view both input and output signals.

on the Runge-Kutta algorithm and is appropriate for most simulations. These numerical ODE solver algorithms is past the scope of this course, but effectively the algorithm attempts to base the time interval step size according to the rate of change expected by the system.

In general, SIMULINK does not have *a priori* knowledge of the input source into the model (such as the `Signal Generator` block in this case with a sine wave) so some adjustments by the user might be necessary. One useful parameter to set is the `Max step size`. By default the `Max step size` is set to `auto` but you can set this to an appropriately small value to limit the maximum time interval between simulation calculations. Adjusting the `Max step size` appropriately will produce an output of sufficient detail. A preferred approach of refining the detail in the output is to increase the `Refine factor` as shown in Fig. 17. By default the `Refine factor` is set to 1 and uses SIMULINK's best guess at the required time interval. The value given for the `Refine factor` will increase the level of refinement (decrease the time interval) by that factor. For instance, if you wish to double the refinement factor from SIMULINK's best guess, you can enter 2 for the `Refine factor` parameter.

Adjust the `Refine factor` until you obtain a simulation result that appears reasonably smooth such as in Fig. 16.

### 3.4 Viewing multiple signals from your simulations

It is often useful to view a number of signals so that they can be compared. For instance, you might wish to monitor the input signal as well as the output signal. One approach is to have multiple `Scope` blocks within your simulation model as shown in Fig. 18. Using this approach, you can independently monitor various signals throughout your simulation model. You can even choose to monitor intermediary points within your model with a few `Scope` blocks.

It is sometimes useful to have the signals on the same plot for the purpose of comparison. For the `Scope` block, multiple signals can be combined together using the `Mux` block which multiplexes the signals into one structure that the `Scope` block will interpret as multiple signals. Each signal is then plotted within the `Scope`'s output window. The `Mux` block is located in the `Signal Routing` section of the SIMULINK Library Browser. Drag-and-drop the `Mux` into your model and double-click on it to open its block parameters, as shown in Fig. 19. From within the block parameters window for the `Mux` block you can set the number of inputs for the `Mux`. In this case we wish to monitor the input and the output, so set the number of inputs to two.

Now connect the model's input and output to the input of the `Mux` block and connect the output of the



Figure 19: Block parameters for the Mux block.

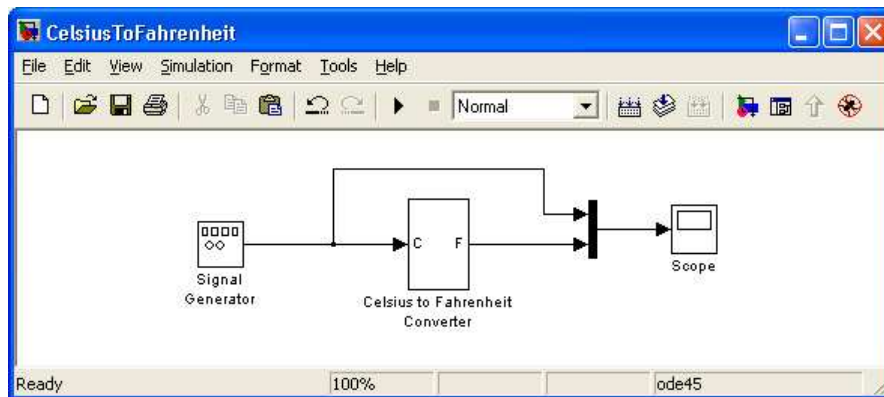


Figure 20: Using Mux to view both input and output signals on the same scope.

Mux block to the Scope block as shown in Fig. 20. When you now simulate the model, both the input and output will appear on the Scope's output window.

Rerun your simulation and monitor the input and output on the scope. You should be able to see how the input temperature in Celsius compares to the converted output temperature in Fahrenheit. Make sure you can obtain a smooth plot like that shown in Fig. 21.

### 3.5 Sharing data with MATLAB

It is often useful to share data/variables between SIMULINK and MATLAB. This sharing of data/variables might be used for configuring settings within your simulation model, to input other types of signals into your SIMULINK model, or to save output signals for further processing/plotting with MATLAB's more versatile plotting functions.

#### 3.5.1 Using variables from MATLAB within SIMULINK

A useful feature of SIMULINK is that expressions can often be used for some block parameters instead of simply a constant value. We saw in the Celsius to Fahrenheit converter subsystem in Fig. 12 that the constant 32 and the expression  $9/5$  were used when realizing Eq. 1.

Another approach of realizing Eq. 1 would be to define some variables with MATLAB that could then be used within the Celsius to Fahrenheit subsystem. Using the MATLAB Command Window (see Fig. 1), enter in the following commands.

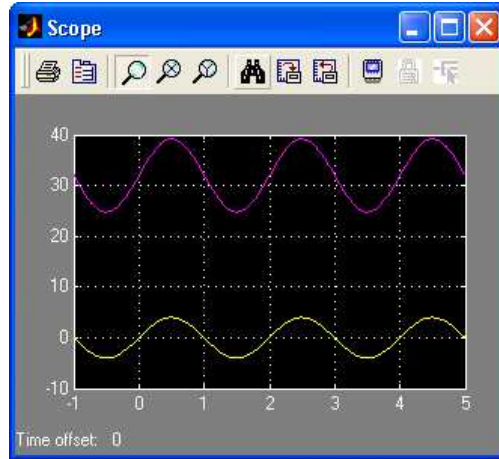


Figure 21: Scope output window with multiple traces.

```
scale=9/5
offset=32
```

These two commands will create the variables `scale` and `offset` for the Celsius to Fahrenheit converter. To see which variables are currently defined within MATLAB, type

```
whos
```

Note that there may be other variables already defined. You can remove all of the variables in MATLAB by typing

```
clear
```

but make sure you redefine the `scale` and `offset` variables before continuing this subsection.

With the `scale` and `offset` variables defined in MATLAB, you can use these variables within blocks such as the Gain block and the Constant block. Open the block parameters for the Gain block and replace the  $9/5$  expression with the `scale` variable name, and similarly change the 32 value in the Constant block to the `offset` variable. You should have a subsystem that looks like Fig. 22. Note that if the block is too small that the variable name may not show. You can always click on the block and resize it by dragging the corner anchors to make it large enough for the variable name to display.

An advantage of using MATLAB variables in this manner is that you can easily change the MATLAB variable to change the values used within the SIMULINK model. For instance, if we wished to convert from Celsius to Kelvin we would need to realize the following equation.

$$^{\circ}\text{K} = ^{\circ}\text{C} + 273.15 \quad (2)$$

We could use the subsystem given in Fig. 22 to calculate Eq. 2 if we set the MATLAB variables to

```
scale=1
offset=273.15
```

Using MATLAB variables can allow the design of SIMULINK models with more control over its configuration.

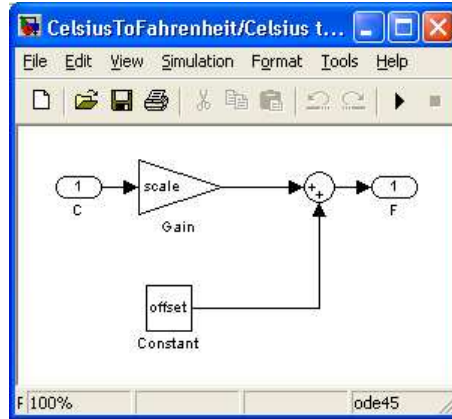


Figure 22: Celsius to Fahrenheit converter subsystem using variables from MATLAB.

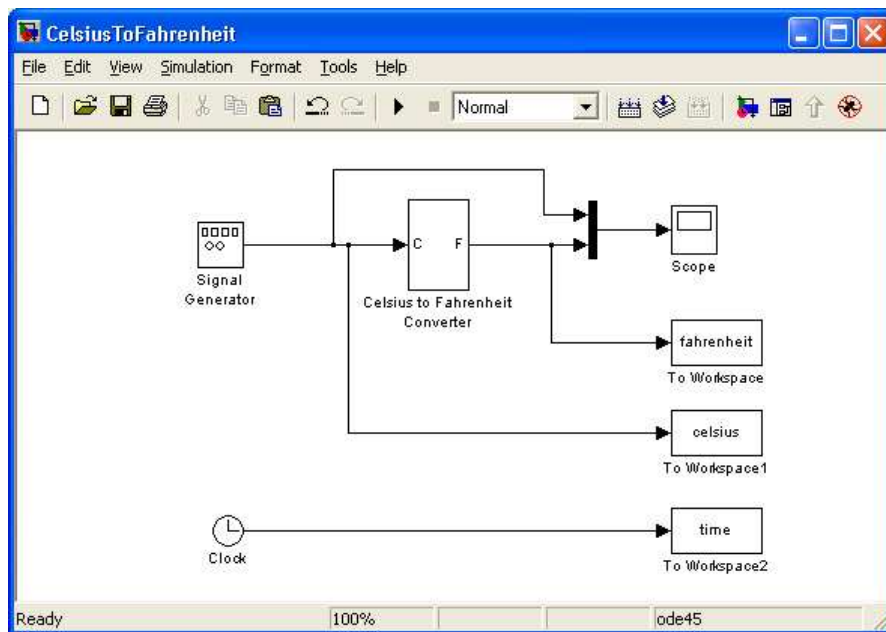


Figure 23: Simulation model with signals sent to the MATLAB workspace.

### 3.5.2 Outputting data to MATLAB

To output signals back to MATLAB’s variable workspace environment, use the To Workspace block from the Sinks section of the SIMULINK Library Browser. For instance, you can use the To Workspace block as shown in Fig. 24 to share the signal from the Signal Generator (*i.e.*, the input temperature in degrees Celsius) and the output of the Celsius to Fahrenheit converter.

Drag-and-drop two To Workspace blocks into your model and connect them to the Celsius and Fahrenheit signals as shown in Fig. 24. To configure the To Workspace blocks, double-click on them which will open a window similar to that shown in Fig. 24. Within the To Workspace block parameters window you can set the variable name that will be assigned within MATLAB for the simulation data passed to the To Workspace block. For instance, the temperature signal in Celsius is set to the variable `celsius` and the temperature signal converted to Fahrenheit is set to the variable `fahrenheit` in Fig. 23. The default Save format for the To Workspace block parameters is as a Structure. While this setting

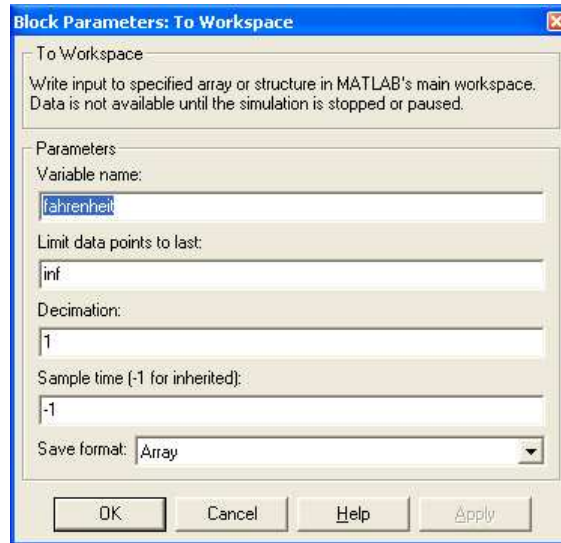


Figure 24: Block parameters for the To Workspace block.

can be used, it requires using the more complicated structure syntax in MATLAB. For our purpose, set the `Save format` to the straightforward `Array` format. A signal then input to that `To Workspace` block will be passed to MATLAB as a simple array of numbers.

Notice that the simulation model in Fig. 23 also uses the `Clock` block from the `Sources` section of the SIMULINK Library Browser and that the output of the `Clock` block is being passed to MATLAB through another `To Workspace` block. The `Clock` block outputs the time steps used in the SIMULINK simulation run. It is important to pass this information to MATLAB, since otherwise data such as the temperatures in Celsius or Fahrenheit passed to MATLAB have no corresponding time frame of when the data samples occurred. Since the time step interval between data samples might not be uniform (recall the `Variable-step` setting in Fig. 17), this time information must also be passed along with any simulation results. Setup the `Clock` block and connecting `To Workspace` block in your simulation model.

Now you can run your simulation and have the signals passed to MATLAB. Before running the simulation, be sure that all of your `To Workspace` blocks have the `Save format` set to `Array`. After running the simulation, you can now use commands from within MATLAB to process/plot your data. Using the MATLAB Command Window (see Fig. 1), enter in the following command.

```
whos
```

The `whos` command lists the current variables within the MATLAB variable environment. You should see variables for `celsius`, `fahrenheit`, and `time` as passed to MATLAB through the `To Workspace` blocks once you have run your simulation.

With the temperature and time variables passed to MATLAB, you can run the following commands to plot the data.

```
plot(time, celsius);
hold on;
plot(time, fahrenheit);
hold off;
title('Plot of temperature over time');
```



```
xlabel('Time');
ylabel('Temperature')
```

If you wish further help on any of the `MATLAB` commands, you can type

```
help command
```

to get details on the use of `command`. For instance, typing

```
help plot
```

will give you help on the `plot()` function in `MATLAB`.

### 3.5.3 Inputting data into SIMULINK

In a similar way that data can be passed from `SIMULINK` to `MATLAB`, data can also be passed from `MATLAB` into `SIMULINK`. For the Celsius to Fahrenheit converter, we could generate any type of input. The input data with `MATLAB` must be setup as a  $2 \times L$  matrix. The first row of the matrix must be a  $1 \times L$  vector containing the time reference of when the data sample was taken. The second row of the matrix is a  $1 \times L$  vector containing the data samples at the corresponding sample times.

For instance, we could generate a  $2 \times L$  matrix of temperature data as follows. This temperature data is just a sine wave that varies the temperature from  $-1^\circ\text{C}$  to  $+1^\circ\text{C}$ .

```
sindata(1,:) = 0:0.01:10;           % Times for each sample from
                                   % 0 to 10 in increments of 0.01
sindata(2,:) = sin(sindata(1,:)); % Temperature at time values
plot(sindata(1,:), sindata(2,:))
```

Other types of data could also be generated. To make the data a little more interesting, the following `MATLAB` code “generates” Celsius temperature data very similar to that measured from Day 2 to Day 7 on the Mars Pathfinder mission.<sup>3</sup> We can use this data as input into our Celsius to Fahrenheit converter.

```
% Times for each data sample
mpfdata(1,:) = linspace(2.4, 7.5, 234);

% Temperature varies as a sinusoid with some random fluctuations
mpfdata(2,:) = -30*sin(2*pi*mpfdata(1,:)) + 4*rand(1,234) - 45;
plot(mpfdata(1,:), mpfdata(2,:))
```

To check the variables created in the `MATLAB` environment, type

```
whos
```

You should have the variable `mpfdata` as a  $2 \times 234$  matrix where the first row of 234 elements are the time indices (real numbers ranging from about 2 to 7 indicating the time of day during the Mars Pathfinder

<sup>3</sup>Actual Mars Pathfinder temperature data can be found at <http://www.sce.carleton.ca/~rdanse/SYSC3600/mpf.mat>. After downloading this file, you can use the `MATLAB` command `load mpf` to load the  $2 \times 234$  matrix saved in the file.

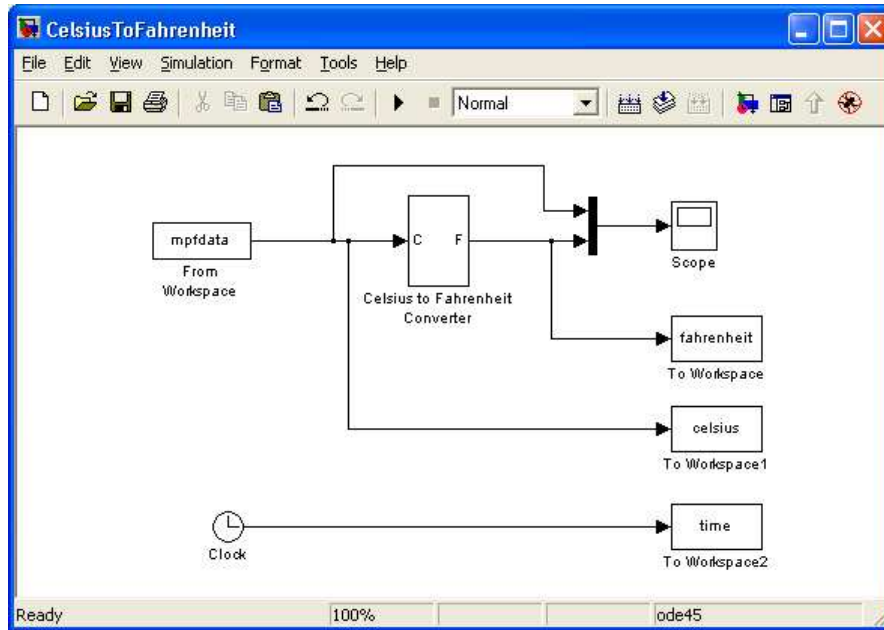


Figure 25: Simulation model with the input taken from the MATLAB workspace variable `mpfdata`.

mission) and the second row of 234 elements are the temperature readings from the Mars Pathfinder in centigrade.

With the temperature data setup in MATLAB, you can now setup your SIMULINK model to use this data as input data for the Celsius to Fahrenheit converter. In your model, delete the `Signal Generator` block and replace it with the `From Workspace` block as shown in Fig. 25 from the `Sources` section of the SIMULINK Library Browser.

Before simulating, you must first configure the `From Workspace` block to use the data from the `mpfdata` matrix. Double-click on the `From Workspace` block to open its block parameters window as shown in Fig. 26. For the `Data` parameter, enter the MATLAB variable name `mpfdata` and click `Ok`. With the data source set, you can now simulate the model and see the temperature at the Mars Pathfinder landing site in both degrees Celsius and Fahrenheit.

### 3.6 Other blocks/features in SIMULINK

A few extra blocks/features are briefly touched on in this section, including the `Product` block and the `Sine Wave` block. Note that the `Step` block and the `Manual Switch` block are discussed in Sec. 4 so are left out of this section.

#### 3.6.1 Flipping, rotating, scaling blocks

You should also notice that you can flip or rotate most blocks in SIMULINK. These options are found by right-clicking on any block and then selecting either

Format→Flip Block

or

Format→Rotate Block

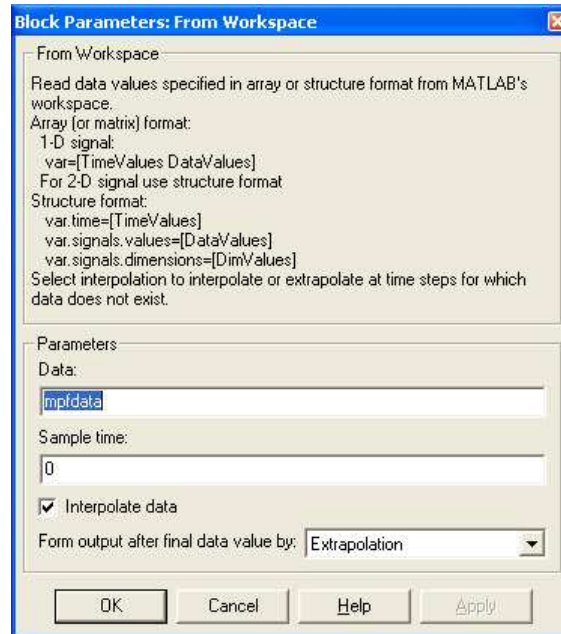


Figure 26: Block parameters for the From Workspace block.

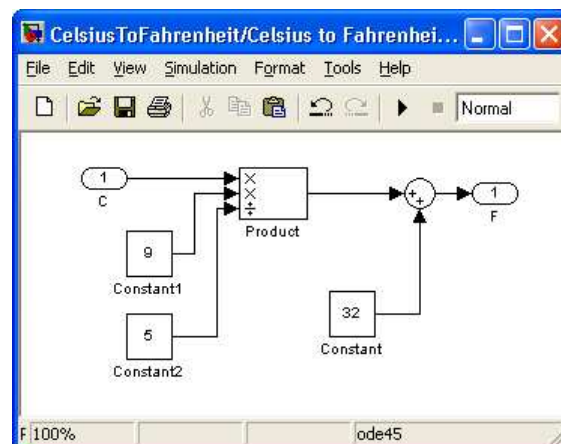


Figure 27: Possible SIMULINK subsystem realization for the Celsius to Fahrenheit converter.

from the pop-up menu. Flipping or rotating the block can be useful for improving the visual layout of your simulation model. For instance, we will see that many simulation diagrams have feedback connections with an associated gain. It will be useful to flip the `Gain` block such that this feedback connection can be neatly routed in the simulation model.

Scaling blocks can be done by clicking on the block and then dragging on one of the four corner anchors of the block. Again, this is only for aesthetic layout, but can help in reading the labels within the block.

### 3.6.2 Product block in SIMULINK

Another useful block is the `Product` block in the `Math` section of the SIMULINK Library Browser. The `Product` block allows you to *multiply* as well as *divide* input signals. An example use of the `Product` block for realizing the Celsius to Fahrenheit converter is given in Fig. 27.

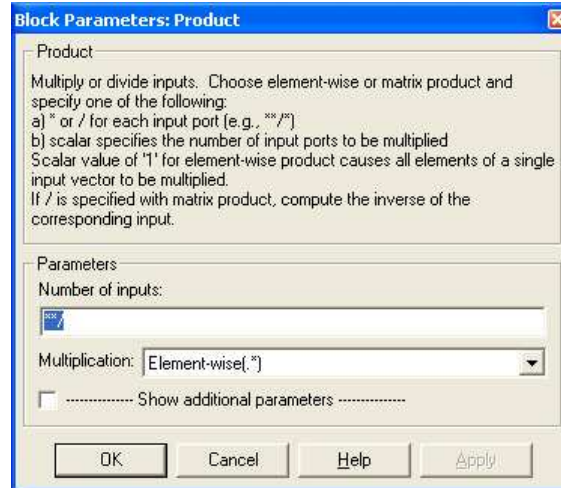


Figure 28: Block parameters for the `Product` block.

The block parameters for the `Product` block can be set by double-clicking on the block which opens the window shown in Fig. 28. Two ways exist for configuring the `Product` block. The first is to simply set the number of inputs to the block, where each input uses a *multiply* operator for combining the inputs. For instance, setting the number of inputs to 3 will cause the `Product` block to have 3 multiply inputs. If you require division, then the second option for configuring the `Product` block is required. Let's say you want a block, as in Fig. 27, where the first two inputs perform multiplication and the third input performs division. In this scenario, configure the block to `**/` which will form the two multiply inputs and the one division input.

### 3.6.3 Sine Wave block in SIMULINK

So far we have used the `Signal Generator` block to generate a sinusoid. One limitation of the sine waves generated by the `Signal Generator` block is that you cannot adjust the phase of the sine wave (at least in versions of `MATLAB` as of the writing of this tutorial). There also exists a `Sine Wave` block in the `Sources` section of the `SIMULINK Library Browser`. This block works like the sine option of the `Signal Generator`, but the phase of the sine wave can also be set.

## 4 SIMULINK Tutorial #2: Integrating a signal

One of the most important blocks needed for simulating systems is the integrator. An integrator, in the context of performing model simulation, performs a running integral in the form of

$$y(t) = \int_{-\infty}^t x(\lambda) d\lambda \quad (3)$$

where  $x(t)$  is the input into the integrator and  $y(t)$  is the output. Note that the running integral has limits from  $-\infty$  to  $t$  and not from  $-\infty$  to  $+\infty$  since the output  $y(t)$  depends only on the input  $x(t)$  from the past to the current time  $t$ , and not into the future compared to the current time  $t$  (*i.e.*, it is a causal block). Since time  $t$  progresses incrementally throughout the simulation, the running integral sums up the contributions of the signal  $x(t)$  over time.

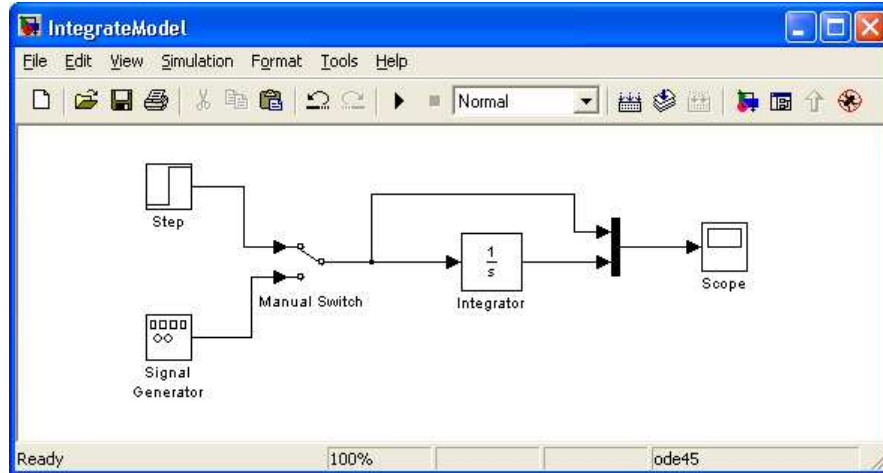


Figure 29: Simulation model to integrate a signal.

Let's build the simple model shown in Fig. 29 so that we can see how an integrator works in SIMULINK. Find the `Integrator` block in the `Continuous` section of the SIMULINK Library Browser and drag-and-drop it into a new model. Note that the `Integrator` block in SIMULINK is labeled with  $1/s$  since this is the Laplace transform representation for integration. Details of the Laplace transform will be discussed later in the course. Next, drag-and-drop the following items from SIMULINK Library Browser into your model.

- `Step` block from the `Sources` section,
- `Signal Generator` block from the `Sources` section,
- `Manual Switch` block from the `Signal Routing` section,
- `Mux` block from the `Signal Routing` section,
- `Scope` block from the `Sinks` section,
- and the `Integrator` block from the `Continuous` section if you haven't already done this step.

Arrange and connect the blocks as shown in Fig. 29.

Let's discuss the `Step` block and the `Manual Switch` block before starting the simulation. The `Step` block is source block to input a step function into our simulation. Double-clicking on the `Step` block opens its block parameters window such as shown in Fig. 30.

The `Step` block available in SIMULINK is more general than the typical step function, or Heaviside function, defined in mathematics. As we can see from the block parameters, the `Step` block's function in SIMULINK is defined as

$$Step(t) = \begin{cases} \text{Initial value}, & t < \text{Step time} \\ \text{Final value}, & t \geq \text{Step time} \end{cases} \quad (4)$$

where `Step time`, `Initial value`, and `Final value` are entries in the block parameters of the `Step` block. The typical definition for a step function in mathematics is given as

$$u(t) = \begin{cases} 0, & t < 0 \\ 1, & t \geq 0. \end{cases} \quad (5)$$

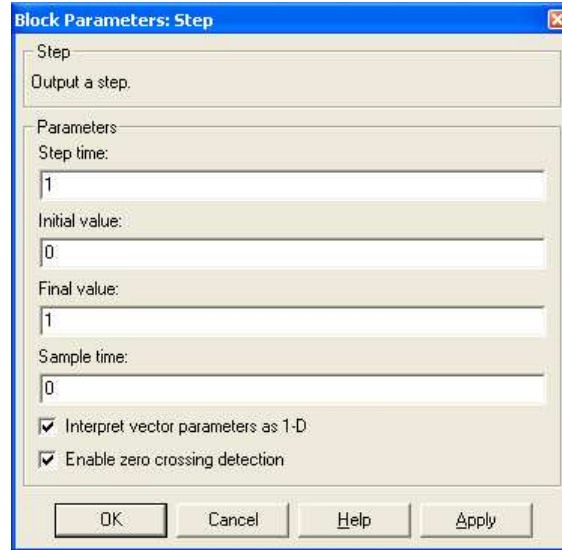
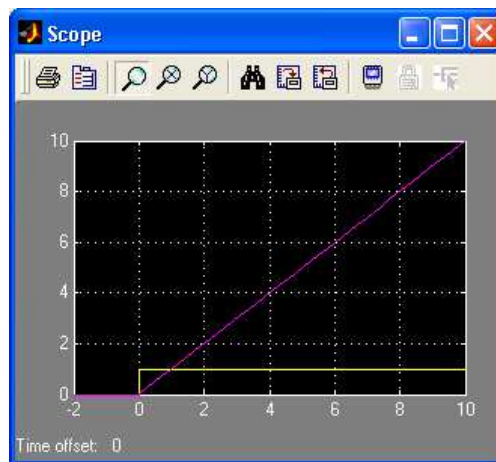


Figure 30: Block parameters for the Step block.

Figure 31: Results of integration of the step function  $u(t)$ .

If we wish to use the Step block in SIMULINK as the mathematical step function  $u(t)$ , we would set the Step time to  $t = 0.0$  seconds, the Initial value to 0.0 and the Final value to 1.0.

The Manual Switch is a simple block that allows us to manually select between two input signals. By default, the Manual Switch will select the top input. By double-clicking on the Manual Switch block, you can change whether the signal from the Step block or the signal from the Signal Generator block is passed through to the output of the Manual Switch.

With the simulation model from Fig. 29 constructed, try running the simulation using the following settings.

1. With a step function input as defined by Eq. 5 and run the simulation from  $t = -2.0$  to  $t = 10.0$  seconds. You should obtain the plot shown in Fig. 31.
2. With a step input with Step time = 3.0, Initial value = -1.0, Final value = 2.0 and run the simulation from  $t = 0.0$  to  $t = 10.0$  seconds. You should obtain the plot shown in Fig. 32.

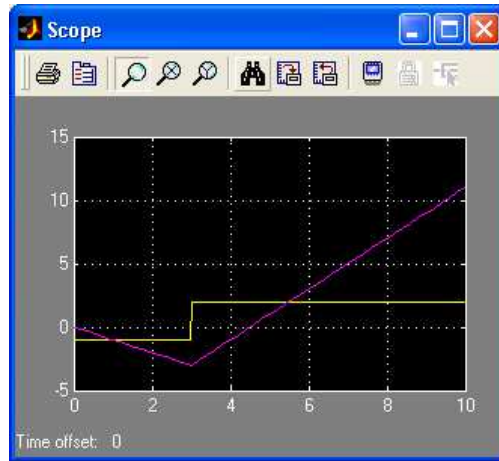


Figure 32: Results of integration from input of the Step block.

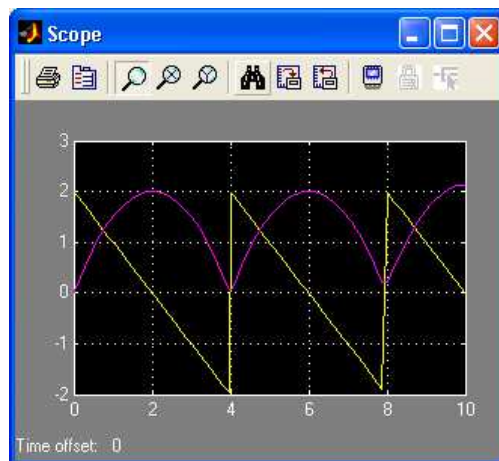


Figure 33: Results of integration of a sawtooth.

3. With a sawtooth input with amplitude of 2 and frequency of 0.25 Hz and run the simulation from  $t = 0.0$  to  $t = 10.0$  seconds. You should obtain the plot shown in Fig. 33.
4. With a sine input with amplitude of 2 and frequency of 0.25 Hz and run the simulation from  $t = 0.0$  to  $t = 10.0$  seconds. You should obtain the plot shown in Fig. 34. Note that performing this integration should give

$$\int A \sin(\omega t) dt = -\frac{A}{\omega} \cos(\omega t) + C. \quad (6)$$

Be sure you have set your simulation parameters appropriately to obtain fine enough detail to show smooth curves.

Verify that you do indeed get the expected result of a cosine. What is the value of  $C$  in this simulation?

#### 4.1 Using initial conditions in an integrator

A difficulty with working with the running integral in Eq. 3 is that the limit starts at time  $t = -\infty$ . This limit requires that the input function  $x(t)$  be known back to  $t = -\infty$ . For most practical simulations, we

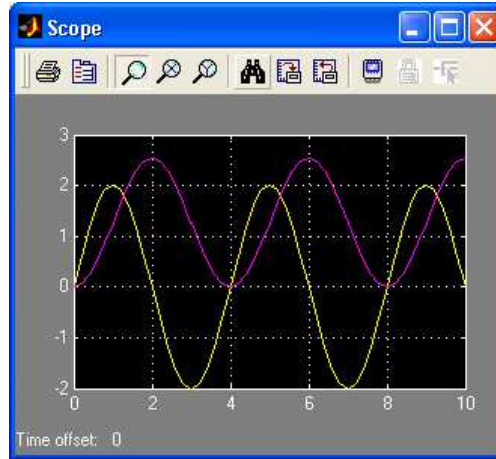


Figure 34: Results of integration of a sine function.

will only know the input  $x(t)$  for some time  $t \geq t_0$ , and the input  $x(t)$  for  $t < t_0$  is unknown. To handle this scenario, we can rewrite the running integral in Eq. 3 as

$$y(t) = \int_{-\infty}^{t_0^-} x(\lambda) d\lambda + \int_{t_0^-}^t x(\lambda) d\lambda \quad (7)$$

$$= y(t_0^-) + \int_{t_0^-}^t x(\lambda) d\lambda \quad \text{for } t \geq t_0 \quad (8)$$

where  $t_0$  is the *start time* of the simulation and  $y(t_0)$  is known as the *initial condition* (IC) for the running integral. Effectively, by writing the running integral as in Eq. 8 we have separated the result into an integral from  $t_0$  to  $t$  over  $x(t)$  that we can evaluate and set  $y(t_0^-)$  as the value we would have obtained had we integrated  $x(t)$  from  $-\infty$  to  $t_0$ . This value  $y(t_0^-)$  might be determined experimentally or might be determined by the current setup of the system just before starting the simulation.

Note that in Eq. 8 we use  $t_0^-$  instead of just  $t_0$  to indicate that we start the running integral from the *left side* of  $t_0$  instead of the *right side* of  $t_0$  (i.e.  $t_0^+$ ). Using  $t_0^-$  allows for any discontinuity in  $x(t)$  at  $t = t_0$  to be correctly incorporated into the running integral (such as with the step function  $u(t - t_0)$  or with the impulse function  $\delta(t - t_0)$ ).

SIMULINK allows ICs within its integrator such that equations like Eq. 8 can be simulated with some IC  $y(t_0^-)$ . Double-clicking on the Integrator block will bring up the block parameters window shown in Fig. 35. The IC for the Integrator block can be set as either an internal IC source or an external IC source. When the initial condition source in the block parameters window is set to internal, then your IC  $y(t_0^-)$  can be entered into the Initial condition entry in the block parameters window. By default, this IC entry is set to zero.

You can also use an external initial condition source, such as shown in Fig. 36. You can select using an external initial condition source from the block parameters window for the Integrator block. With the IC set as an external source, an extra port appears on the Integrator block to accept the IC. You can set a constant source input, such as shown in Fig. 36, to input the IC  $y(t_0^-)$ . A Constant block or some other source block can be used to externally set the IC for the integrator. This approach is particularly useful if you are creating a subsystem containing an Integrator block and wish to have control over the IC of the integrator from outside of the subsystem.

Set the initial condition to  $y(t_0^-) = 10$  (either with an internal or external source) and run the integration from the previous section for the sine wave input. How does the results differ from that shown in Fig. 34?



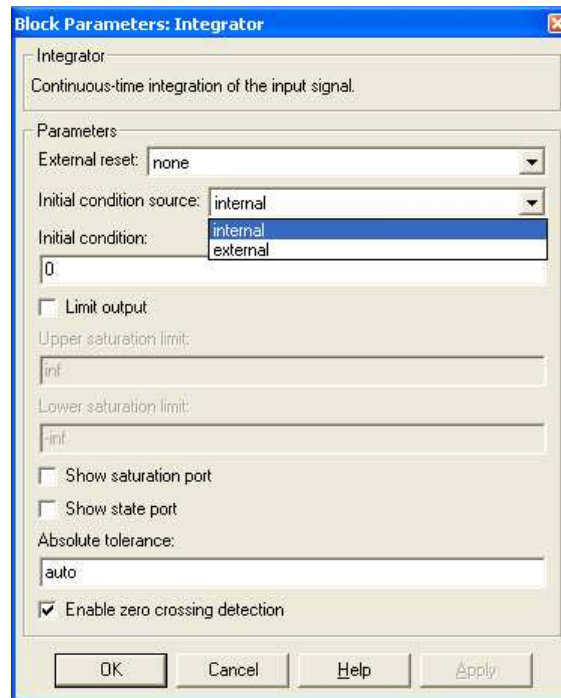


Figure 35: Block parameters for the Integrator block.

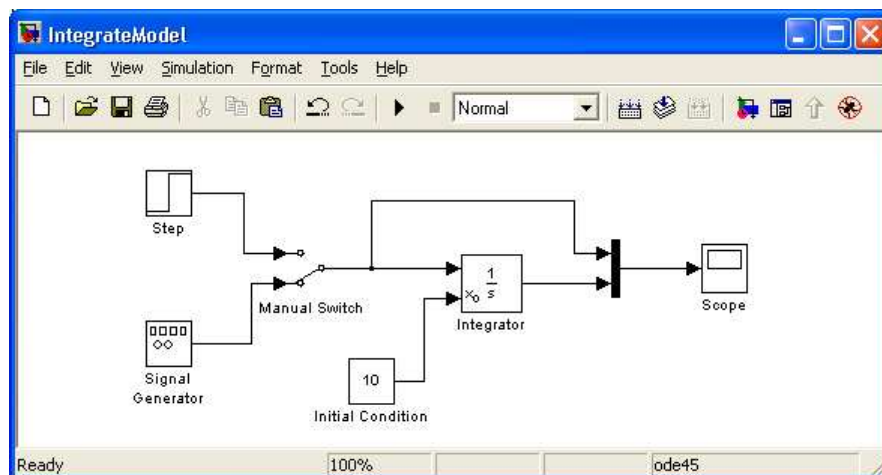


Figure 36: Simulation model to integrate a signal using external initial conditions on the Integrator block.