Simulink Basics Tutorial

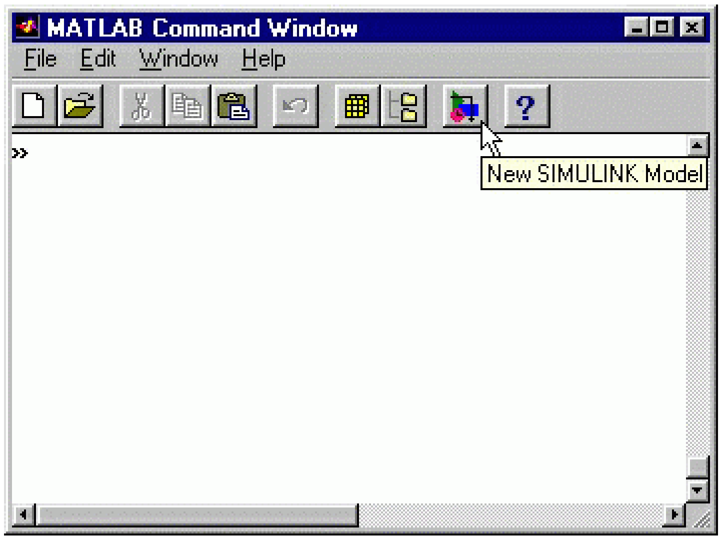
Simulink is a graphical extension to MATLAB for modeling and simulation of systems. In Simulink, systems are drawn on screen as block diagrams. Many elements of block diagrams are available, such as transfer functions, summing junctions, etc., as well as virtual input and output devices such as function generators and oscilloscopes.   These virtual devices will allow you to perform simulations of the models you will build. Simulink is integrated with MATLAB and data can be easily transferred between the programs. In this tutorial, we will apply Simulink to the examples of modeled systems, then build controllers, and simulate the systems

Starting Simulink

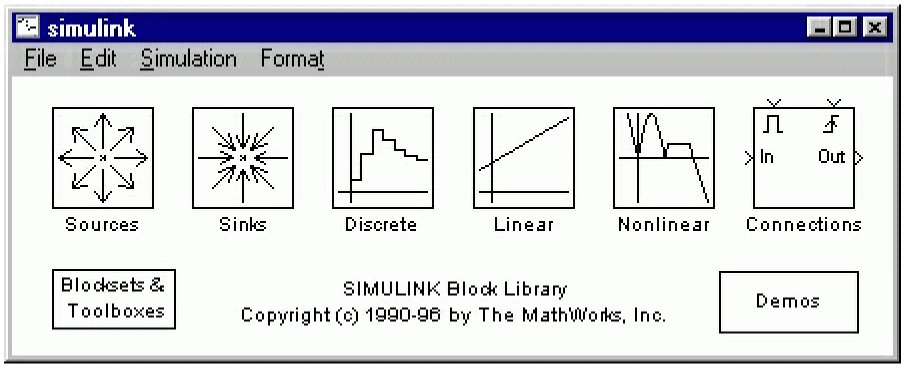
Simulink is started from the MATLAB command prompt by entering the following command:

```
        simulink
```

Alternatively, you can hit the New Simulink Model button at the top of the MATLAB command window as shown below:



When it starts, Simulink brings up two windows. The first is the main Simulink window, which appears as shown or similar to this as different versions of the software are found:



The second window is a blank, untitled, model window. This is the window into which a new model can be drawn.

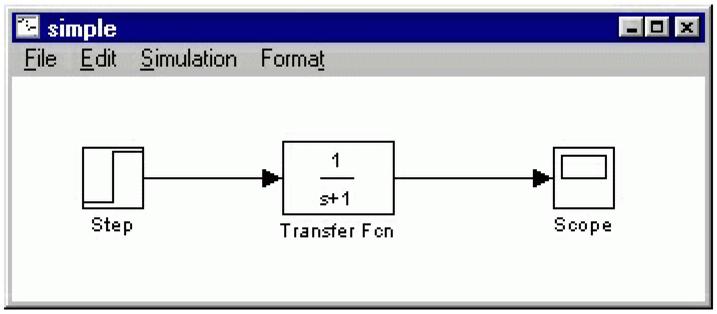
Model Files

In Simulink, a model is a collection of blocks which, in general, represents a system. In addition, to drawing a model into a blank model window, previously saved model files can be loaded either from the **File** menu or from the MATLAB command prompt. As a first exercise we will be building a model.

You can open a file in Simulink by entering the following command in the MATLAB command window. (Alternatively, you can load this file using the **Open** option in the **File** menu in Simulink, or by hitting Ctrl+O in Simulink.)

```
       File name.mdl
```

For our purposes we will create the following model in simulink. A new model can be created by selecting **New** from the **File** menu in any Simulink window (or by hitting Ctrl+N).   Using the information below, create the following model.



Basic Elements

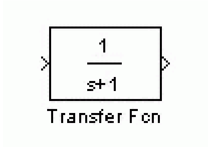
There are two major classes of items in Simulink: **blocks** and **lines**. Blocks are used to generate, modify, combine, output, and display signals. Lines are used to transfer signals from one block to another.

Blocks

There are several general classes of blocks:

- Sources: Used to generate various signals
- Sinks: Used to output or display signals
- Discrete: Linear, discrete-time system elements (transfer functions, state-space models, etc.)
- Linear: Linear, continuous-time system elements and connections (summing junctions, gains, etc.)
- Nonlinear: Nonlinear operators (arbitrary functions, saturation, delay, etc.)
- Connections: Multiplex, Demultiplex, System Macros, etc.

Blocks have zero to several input terminals and zero to several output terminals. Unused input terminals are indicated by a small open triangle. Unused output terminals are indicated by a small triangular point. The block shown below has an unused input terminal on the left and an unused output terminal on the right.

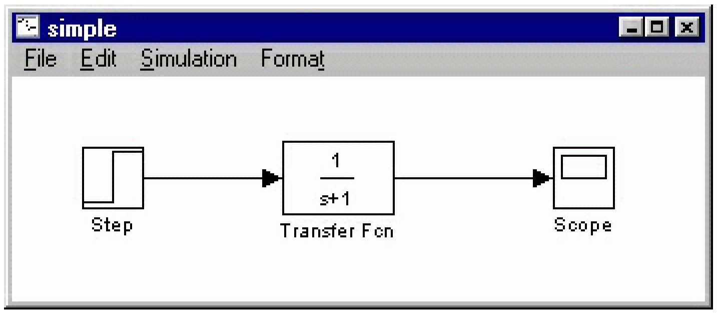


Lines

Lines transmit signals in the direction indicated by the arrow. Lines must always transmit signals from the output terminal of one block to the input terminal of another block. On exception to this is a line can tap off of another line, splitting the signal to each of two destination blocks, as shown.   Lines can never inject a signal *into* another line; lines must be combined through the use of a block such as a summing junction.

A signal can be either a scalar signal or a vector signal. For Single-Input, Single-Output systems, scalar signals are generally used. For Multi-Input, Multi-Output systems, vector signals are often used, consisting of two or more scalar signals. The lines used to transmit scalar and vector signals are identical. The blocks on either end of the line determine the type of signal carried by the line.

Simple Example

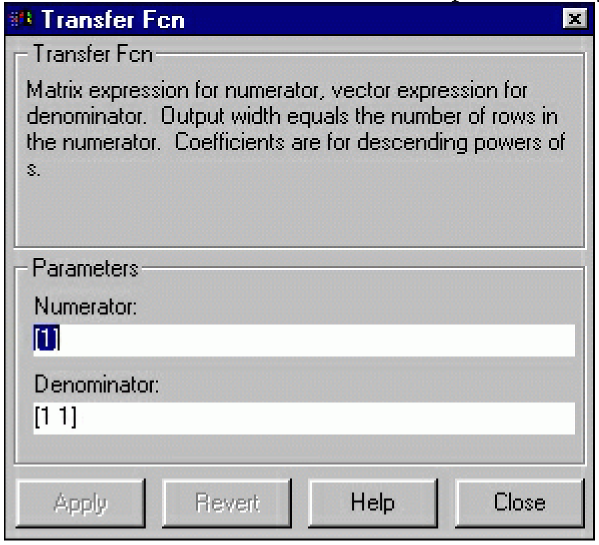


The *simple* model (from the model file section) consists of three blocks: Step, Transfer Fcn, and Scope. The Step is a **source block** from which a step input signal originates. This signal is transferred through the **line** in the direction indicated by the arrow to the Transfer Function **linear block**. The Transfer Function modifies its input signal and outputs a new signal on a line to the Scope. The Scope is a **sink block** used to display a signal much like an oscilloscope.

There are many more types of blocks available in Simulink, some of which will be discussed later. Right now, we will examine just the three we have used in the *simple* model.

Modifying Blocks

A block can be modified by double-clicking on it. For example, if you double-click on the "Transfer Fcn" block in the *simple* model, you will see the following dialog box.



Transfer Fcn

Transfer Fcn

Matrix expression for numerator, vector expression for denominator. Output width equals the number of rows in the numerator. Coefficients are for descending powers of s.

Parameters

Numerator:

[1]

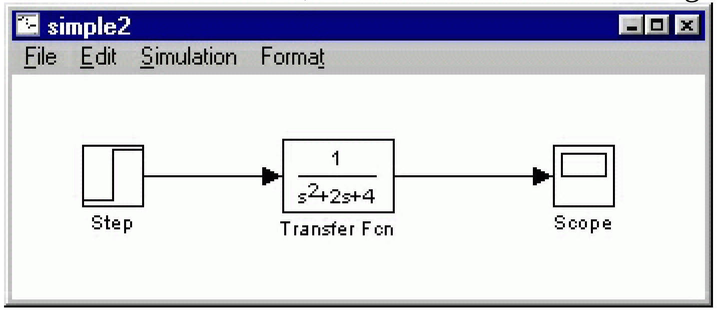
Denominator:

[1 1]

This dialog box contains fields for the numerator and the denominator of the block's transfer function. By entering a vector containing the coefficients of the desired numerator or denominator polynomial, the desired transfer function can be entered. For example, to change the denominator to s^2+2s+4, enter the following into the denominator field:

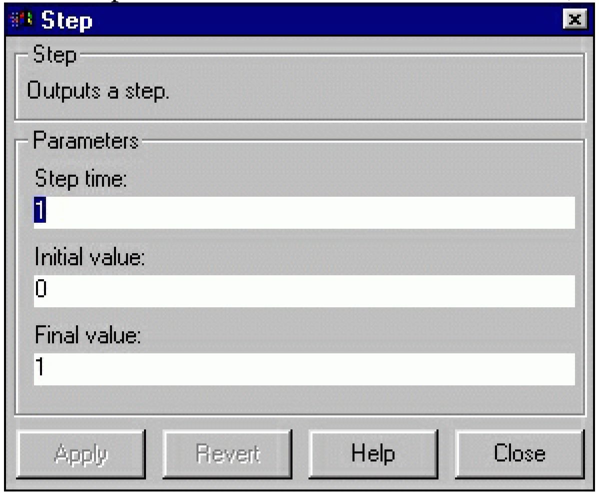
```
        [1 2 4]
```

and hit the close button, the model window will change to the following,



which reflects the change in the denominator of the transfer function.

The "step" block can also be double-clicked, bringing up the following dialog box.



Step

Step

Outputs a step.

Parameters

Step time:

1

Initial value:

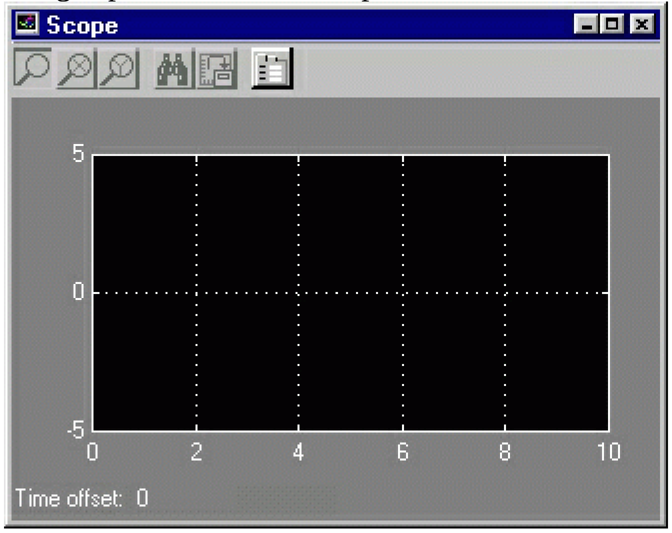
0

Final value:

1

The default parameters in this dialog box generate a step function occurring at time=1 sec, from an initial level of zero to a level of 1. (in other words, a unit step at t=1). Each of these parameters can be changed. Close this dialog before continuing.

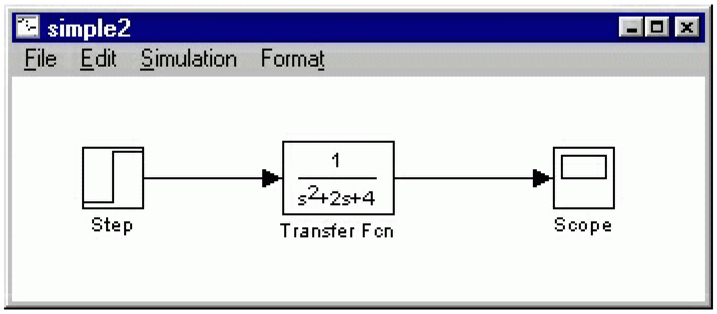
The most complicated of these three blocks is the "Scope" block. Double clicking on this brings up a blank oscilloscope screen.



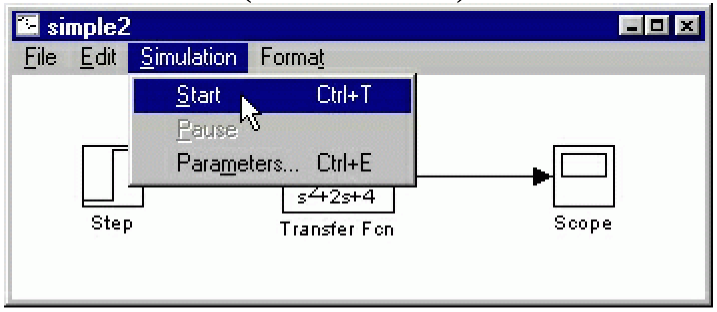
When a simulation is performed, the signal which feeds into the scope will be displayed in this window. Detailed operation of the scope will not be covered in this tutorial. The only function we will use is the autoscale button, which appears as a pair of binoculars in the upper portion of the window.

Running Simulations

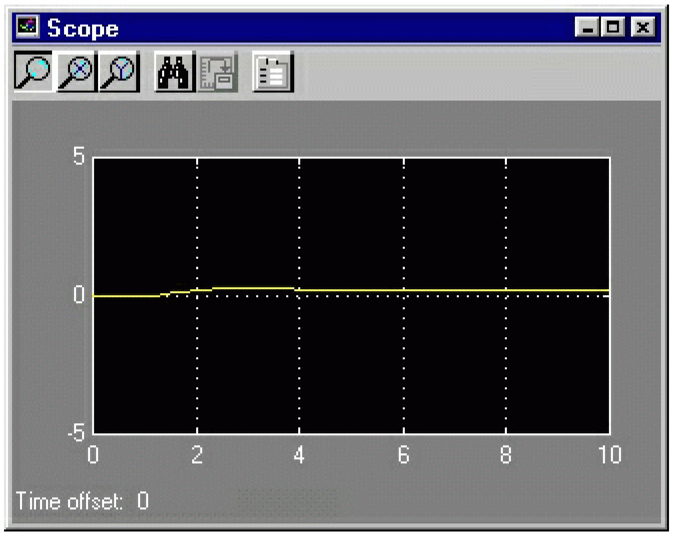
To run a simulation, we will work with the following model. Using the techniques developed above creates the following model:



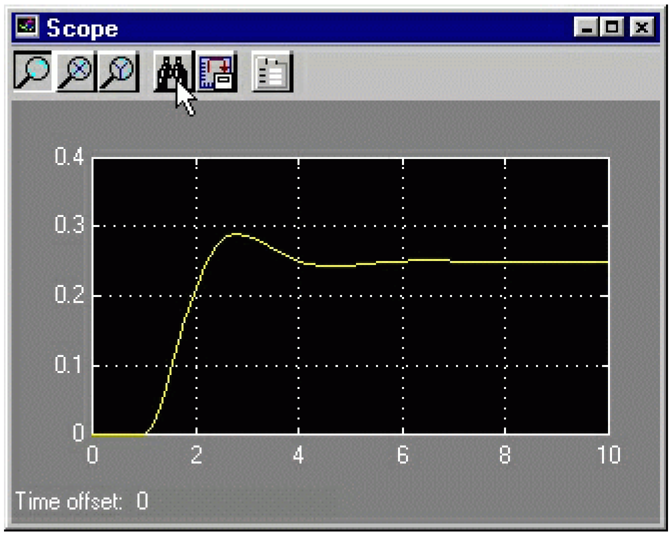
Before running a simulation of this system, first open the scope window by double-clicking on the scope block. Then, to start the simulation, either select **Start** from the **Simulation** menu (as shown below) or hit Ctrl-T in the model window.



The simulation should run very quickly and the scope window will appear as shown below.



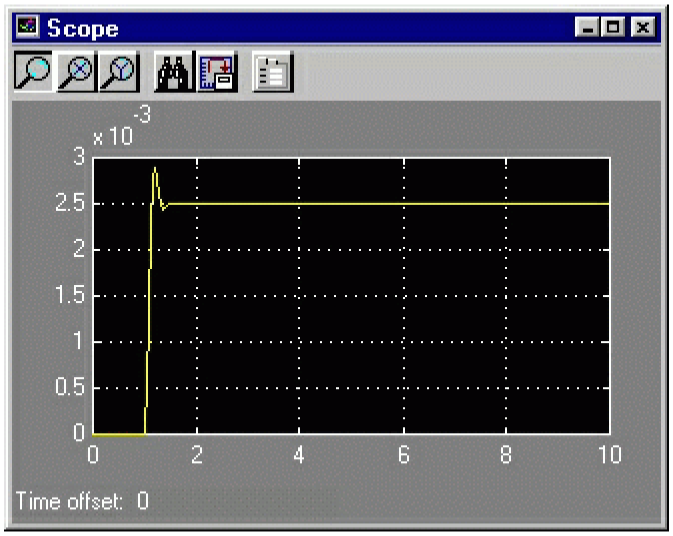
Note that the simulation output (shown in yellow) is at a very low level relative to the axes of the scope. To fix this, hit the autoscale button (binoculars), which will rescale the axes as shown below.



Note that the step response does not begin until t=1. This can be changed by double-clicking on the "step" block. Now, we will change the parameters of the system and simulate the system again. Double-click on the "Transfer Fcn" block in the model window and change the denominator to

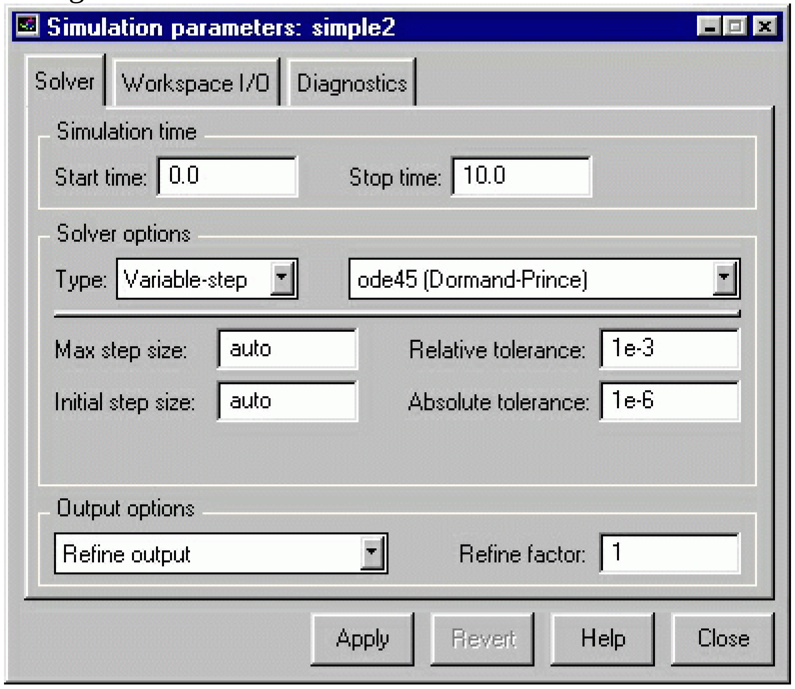
```
        [1 20 400]
```

Re-run the simulation (hit Ctrl-T) and you should see what appears as a flat line in the scope window. Hit the autoscale button, and you should see the following in the scope window.



Notice that the autoscale button only changes the vertical axis. Since the new transfer function has a very fast response, it compressed into a very narrow part of the scope window. This is not really a problem with the scope, but with the simulation itself. Simulink simulated the system for a full ten seconds even though the system had reached steady state shortly after one second.

To correct this, you need to change the parameters of the simulation itself. In the model window, select **Parameters** from the **Simulation** menu. You will see the following dialog box.



Simulation parameters: simple2

Simulation time

Start time: 0.0 Stop time: 10.0

Solver options

Type: Variable-step ode45 (Dormand-Prince)

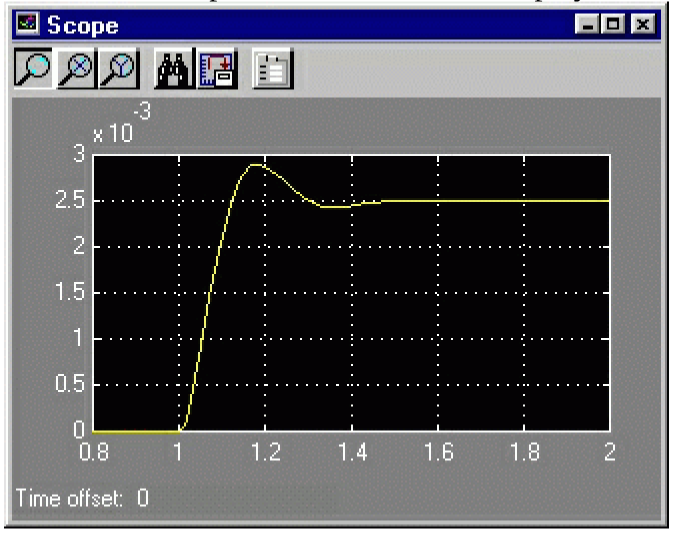
Max step size: auto Relative tolerance: 1e-3

Initial step size: auto Absolute tolerance: 1e-6

Output options

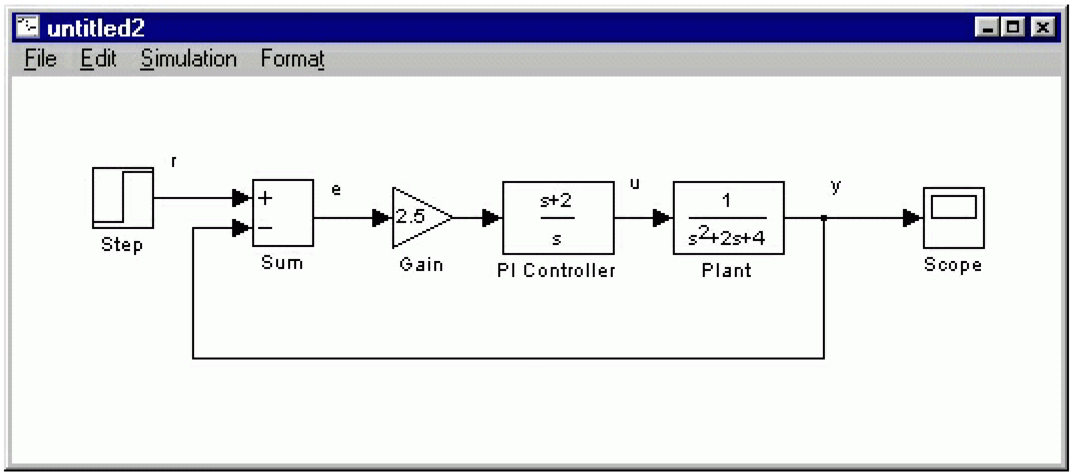
Refine output Refine factor: 1

There are many simulation parameter options; we will only be concerned with the start and stop times, which tell Simulink over what time period to perform the simulation. Change **Start time** from 0.0 to 0.8 (since the step doesn't occur until t=1.0. Change **Stop time** from 10.0 to 2.0, which should be only shortly after the system settles. Close the dialog box and rerun the simulation. After hitting the auto-scale button, the scope window should provide a much better display of the step response as shown below.



Building Systems

In this section, you will learn how to build systems in Simulink using the building blocks in Simulink's Block Libraries. You will build the following system.

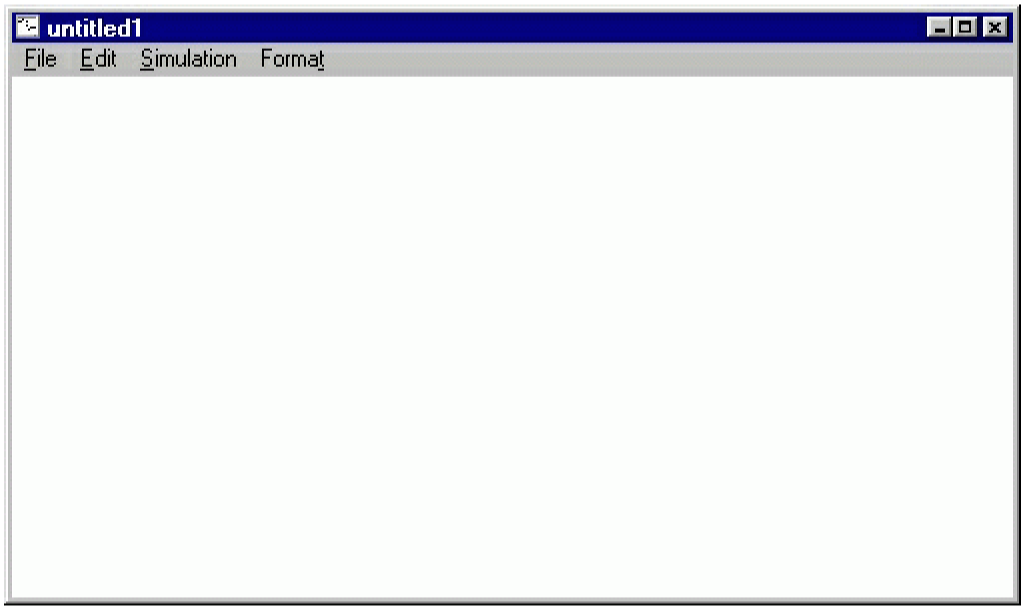


First you will gather all the necessary blocks from the block libraries. Then you will modify the blocks so they correspond to the blocks in the desired model. Finally, you will connect the blocks with lines to form the complete system. After this, you will simulate the complete system to verify that it works.

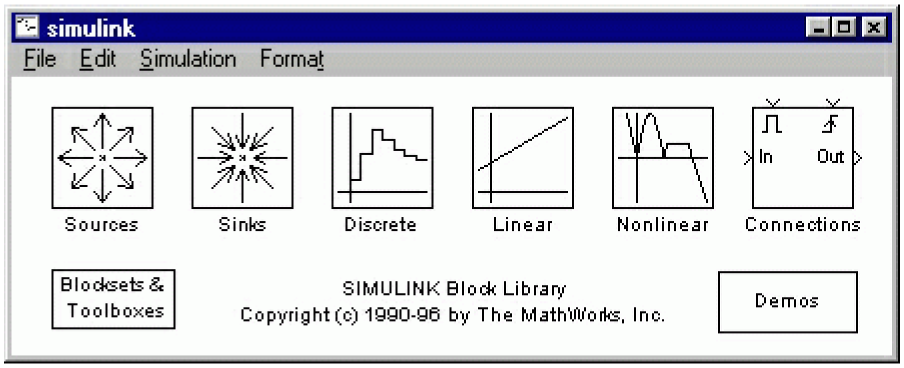
Gathering Blocks

Follow the steps below to collect the necessary blocks:

- Create a new model (**New** from the **File** menu or Ctrl-N). You will get a blank model window.

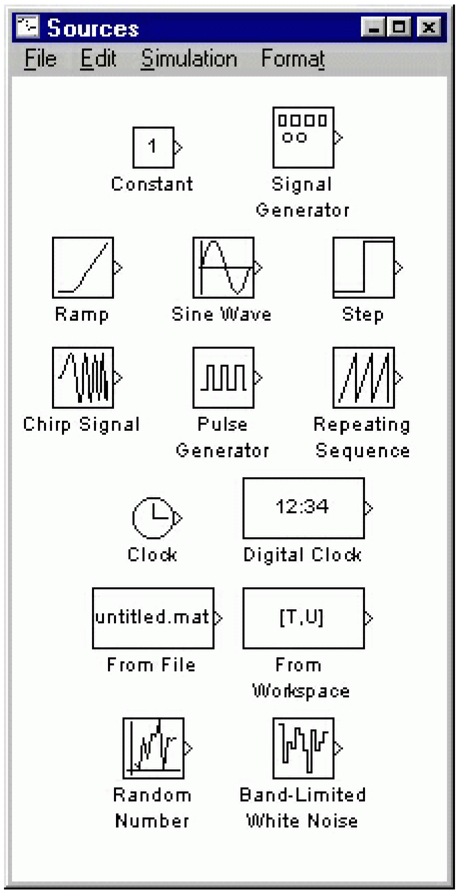


- Double-click on the Sources icon in the main Simulink window.

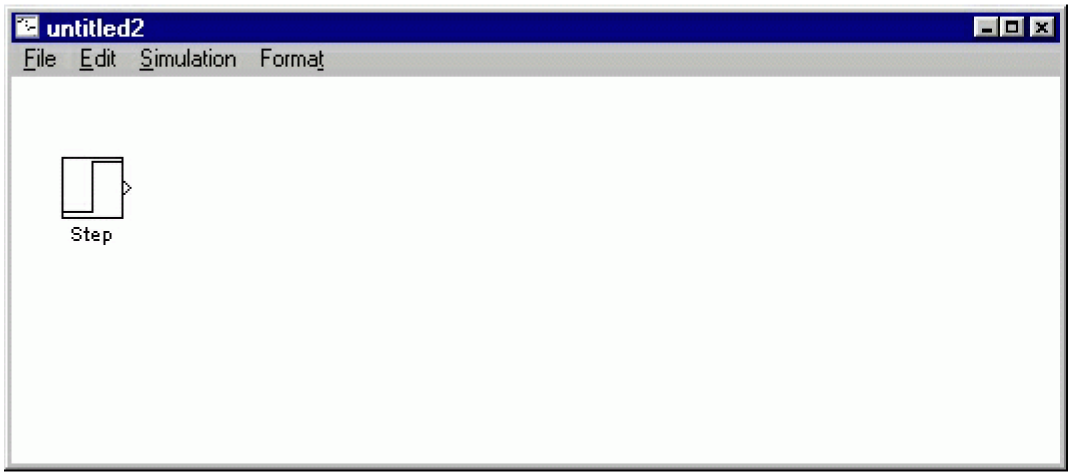


SIMULINK Block Library
Copyright (c) 1990-96 by The MathWorks, Inc.

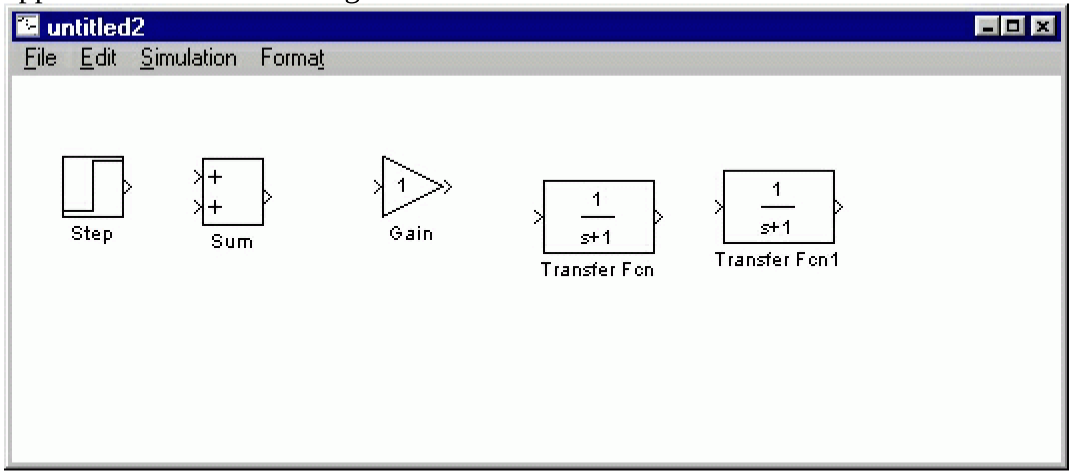
This opens the Sources window which contains the Sources Block Library. Sources are used to generate signals. Click here for more information on block libraries.



- Drag the Step block from the sources window into the left side of your model window.

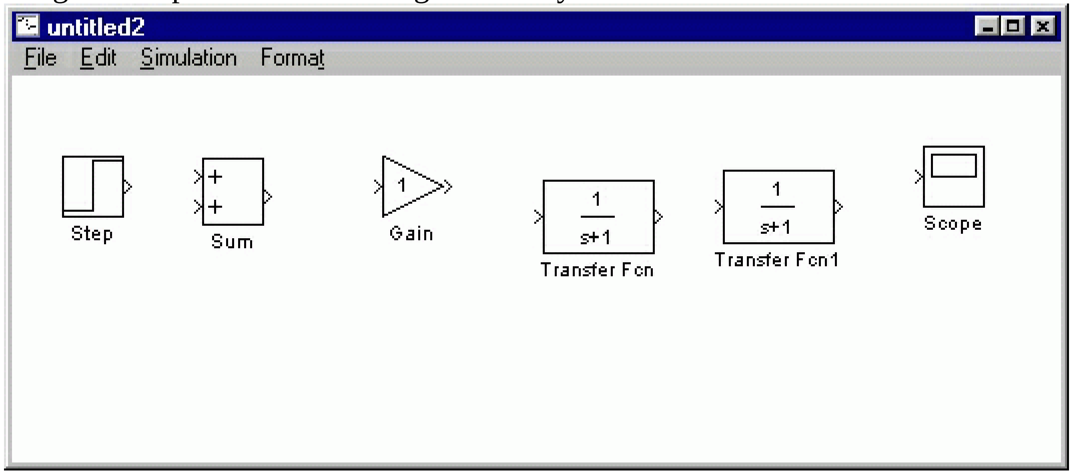


- Double-click on the Linear icon in the main Simulink window to open the Linear Block Library window.
- Drag the Sum, Gain, and two instances of the Transfer Fcn (drag it two times) into your model window arranged approximately as shown below. The exact alignment is not important since it can be changed later. Just try to get the correct relative positions. Notice that the second Transfer Function block has a 1 after its name. Since no two blocks may have the same name, Simulink automatically appends numbers following the names of blocks to differentiate between them.



- Double-click on the Sinks icon in the main Simulink window to open the Sinks window.

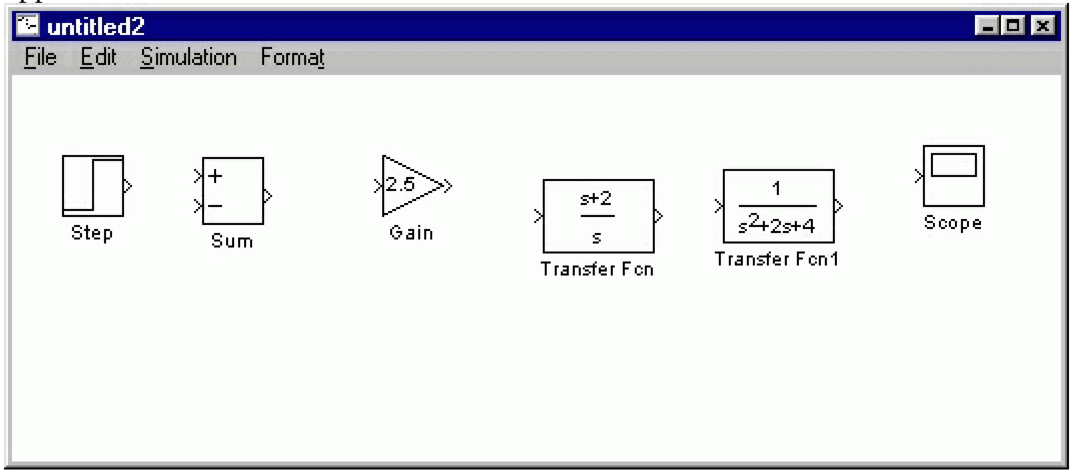
- Drag the Scope block into the right side of your model window.



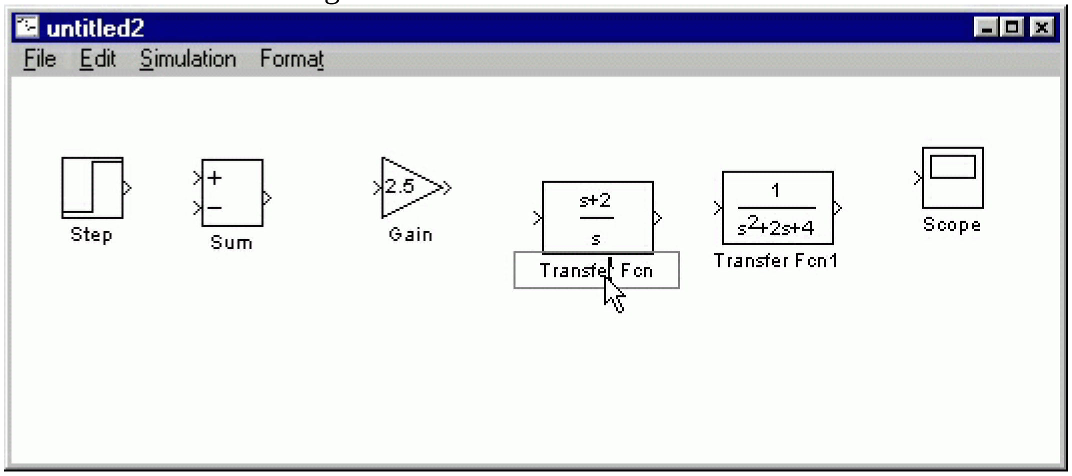
Modify Blocks

Follow these steps to properly modify the blocks in your model.

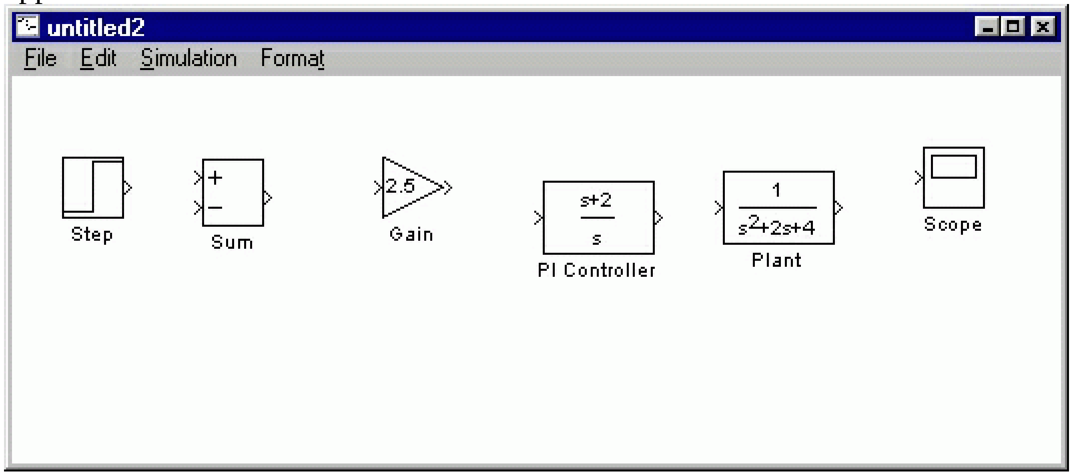
- Double-click your Sum block. Since you will want the second input to be subtracted, enter +- into the list of signs field. Close the dialog box.
- Double-click your Gain block. Change the gain to 2.5 and close the dialog box.
- Double-click the leftmost Transfer Function block. Change the numerator to [1 2] and the denominator to [1 0]. Close the dialog box.
- Double-click the rightmost Transfer Function block. Leave the numerator [1], but change the denominator to [1 2 4]. Close the dialog box. Your model should appear as:



- Change the name of the first Transfer Function block by clicking on the words "Transfer Fcn". A box and an editing cursor will appear on the block's name as shown below. Use the keyboard (the mouse is also useful) to delete the existing name and type in the new name, "PI Controller". Click anywhere outside the name box to finish editing.



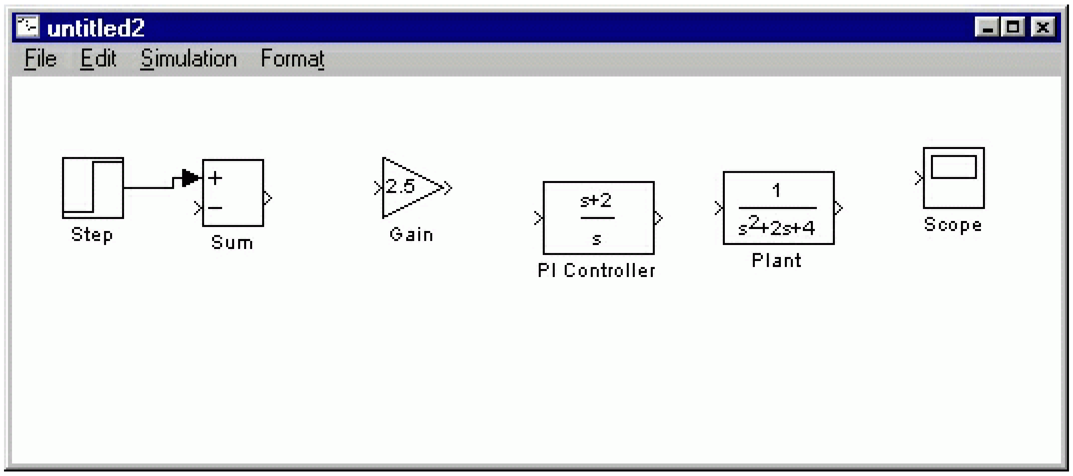
- Similarly, change the name of the second Transfer Function block from "Transfer Fcn1" to "Plant". Now, all the blocks are entered properly. Your model should appear as:



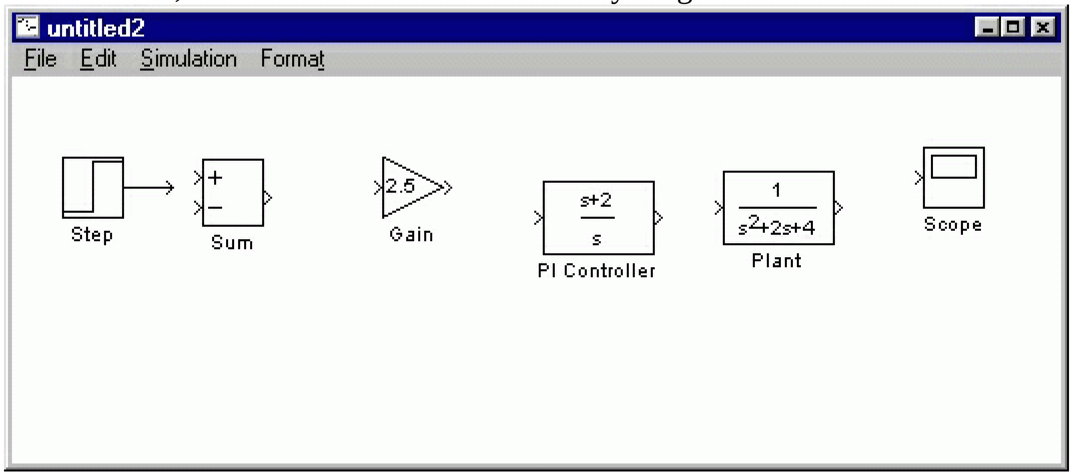
Connecting Blocks with Lines

Now that the blocks are properly laid out, you will now connect them together. Follow these steps.

- Drag the mouse from the output terminal of the Step block to the upper (positive) input of the Sum block. Let go of the mouse button only when the mouse is right on the input terminal. Do not worry about the path you follow while dragging, the line will route itself. You should see the following.

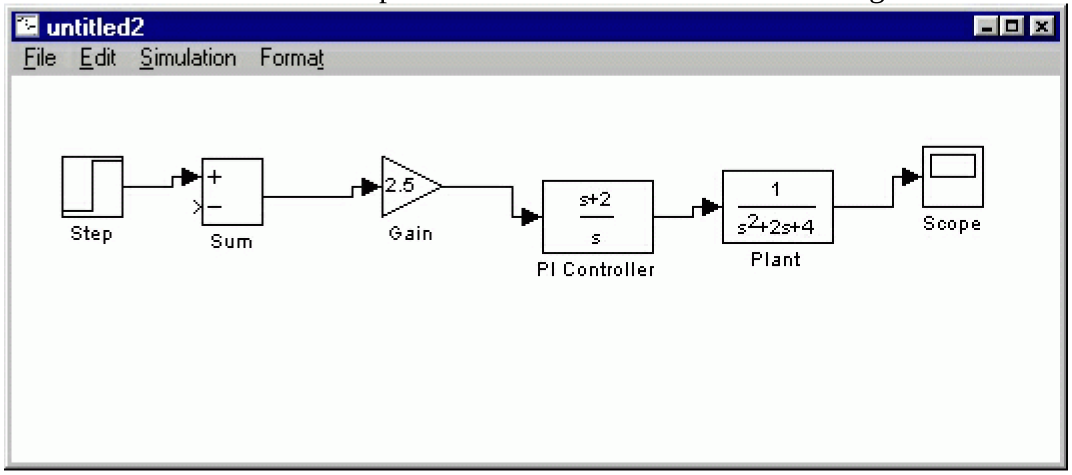


The resulting line should have a filled arrowhead. If the arrowhead is open, as shown below, it means it is not connected to anything.



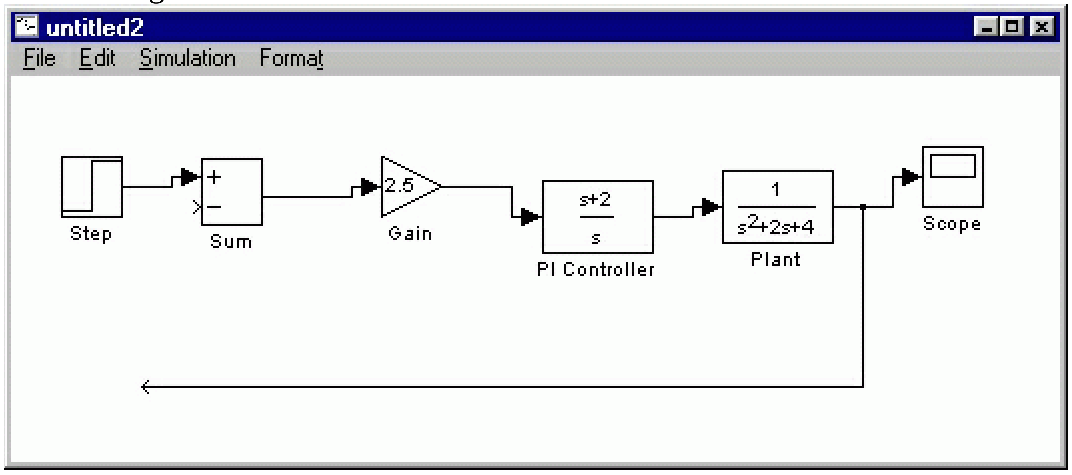
You can continue the partial line you just drew by treating the open arrowhead as an output terminal and drawing just as before. Alternatively, if you want to redraw the line, or if the line connected to the wrong terminal, you should delete the line and redraw it. To delete a line (or any other object), simply click on it to select it, and hit the delete key.

- Draw a line connecting the Sum block output to the Gain input. Also draw a line from the Gain to the PI Controller, a line from the PI Controller to the Plant, and a line from the Plant to the Scope. You should now have the following.



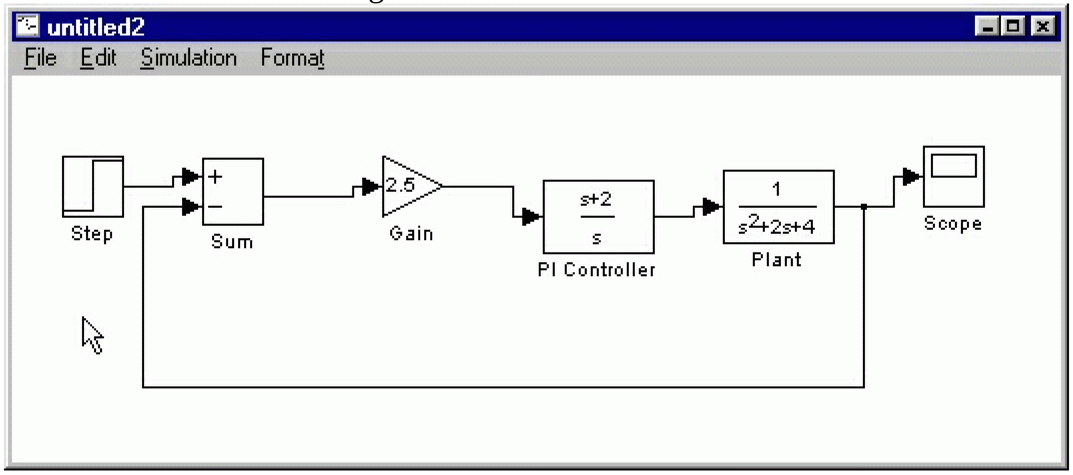
- The line remaining to be drawn is the feedback signal connecting the output of the Plant to the negative input of the Sum block. This line is different in two ways. First, since this line loops around and does not simply follow the shortest (right-angled) route so it needs to be drawn in several stages. Second, there is no output terminal to start from, so the line has to tap off of an existing line.

  To tap off the output line, hold the Ctrl key while dragging the mouse from the point on the existing line where you want to tap off. In this case, start just to the right of the Plant. Drag until you get to the lower left corner of the desired feedback signal line as shown below.

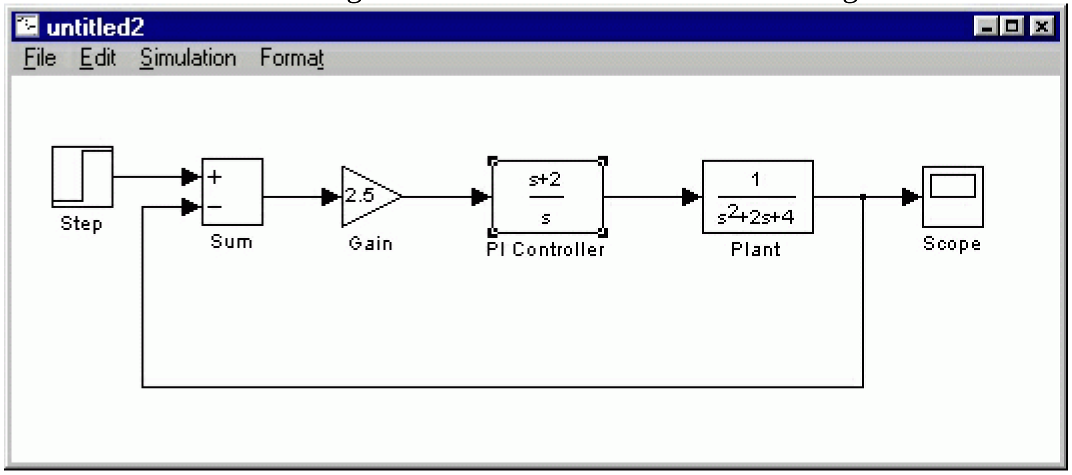


Now, the open arrowhead of this partial line can be treated as an output terminal.

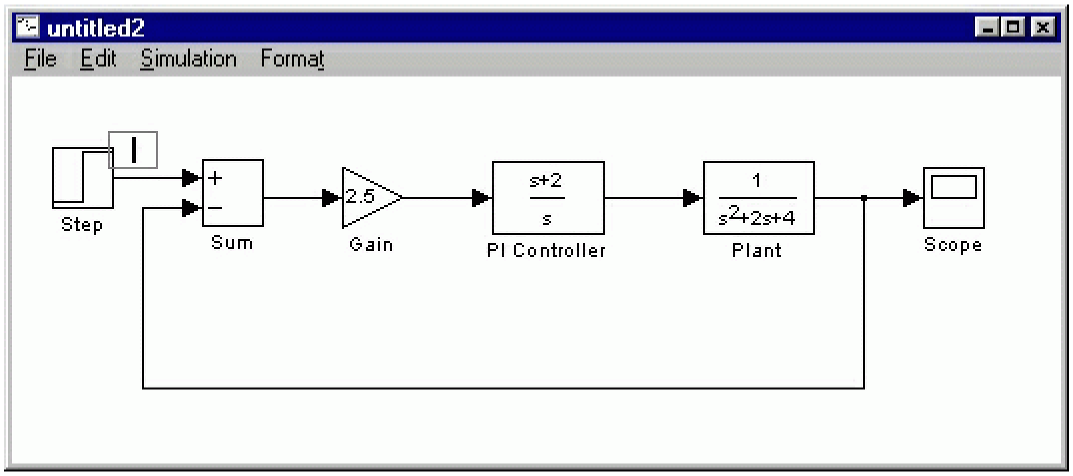
Draw a line from it to the negative terminal of the Sum block in the usual manner.



- Now, you will align the blocks with each other for a neater appearance. Once connected, the actual positions of the blocks does not matter, but it is easier to read if they are aligned. To move each block, drag it with the mouse. The lines will stay connected and re-route themselves. The middles and corners of lines can also be dragged to different locations. Starting at the left, drag each block so that the lines connecting them are purely horizontal. Also, adjust the spacing between blocks to leave room for signal labels. You should have something like:

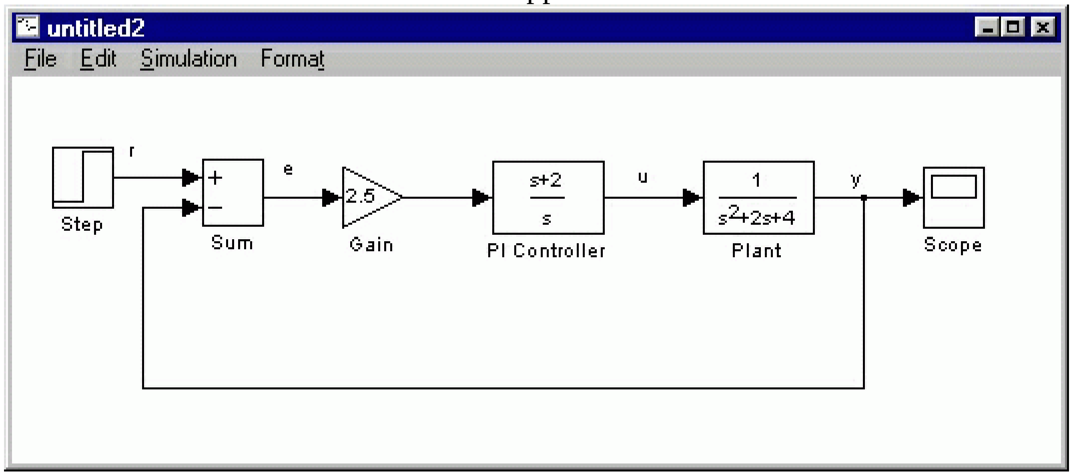


- Finally, you will place labels in your model to identify the signals. To place a label anywhere in your model, double click at the point you want the label to be. Start by double clicking above the line leading from the Step block. You will get a blank text box with an editing cursor as shown below



Type an r in this box, labeling the reference signal and click outside it to end editing.

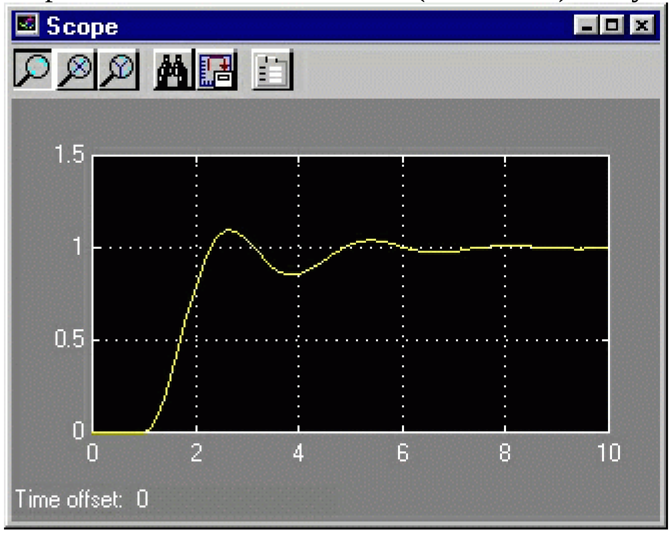
- Label the error (e) signal, the control (u) signal, and the output (y) signal in the same manner. Your final model should appear as:



- To save your model, select **Save As** in the **File** menu and type in any desired model name. The completed model can be found here.

Simulation

Now that the model is complete, you can simulate the model. Select **Start** from the **Simulation** menu to run the simulation. Double-click on the Scope block to view its output. Hit the autoscale button (binoculars) and you should see the following.



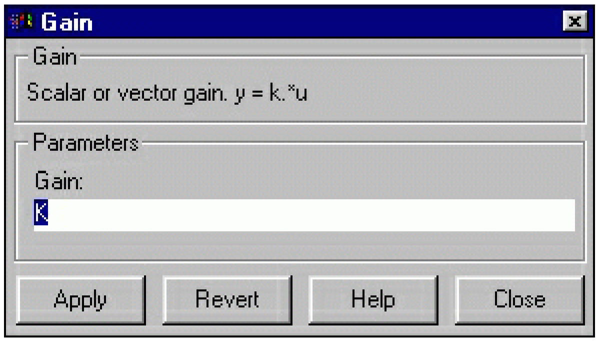
Taking Variables from MATLAB

In some cases, parameters, such as gain, may be calculated in MATLAB to be used in a Simulink model. If this is the case, it is not necessary to enter the result of the MATLAB calculation directly into Simulink. For example, suppose we calculated the gain in MATLAB in the variable K. Emulate this by entering the following command at the MATLAB command prompt.

```
K=2.5
```

This variable can now be used in the Simulink Gain block. In your simulink model, double-click on the Gain block and enter the following in the Gain field.

```
K
```


Gain

Gain

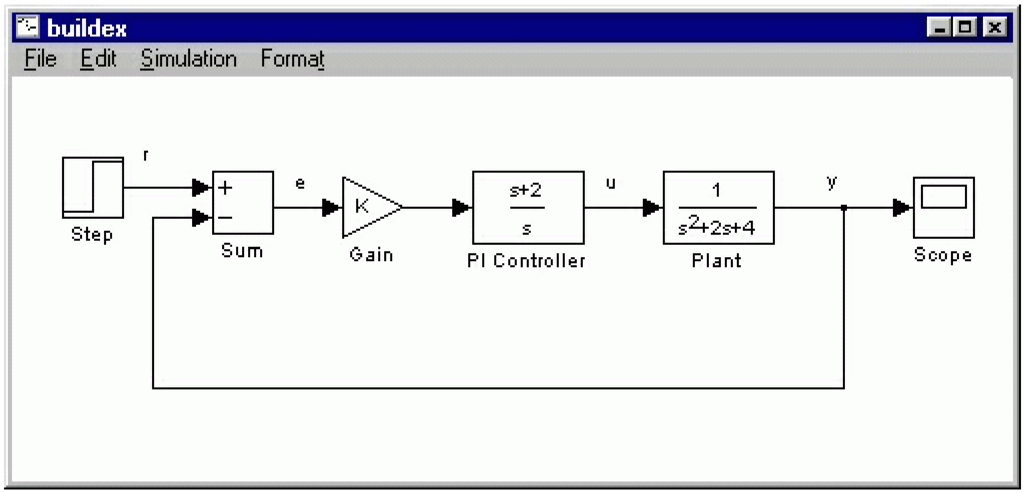
Scalar or vector gain. y = k.*u

Parameters

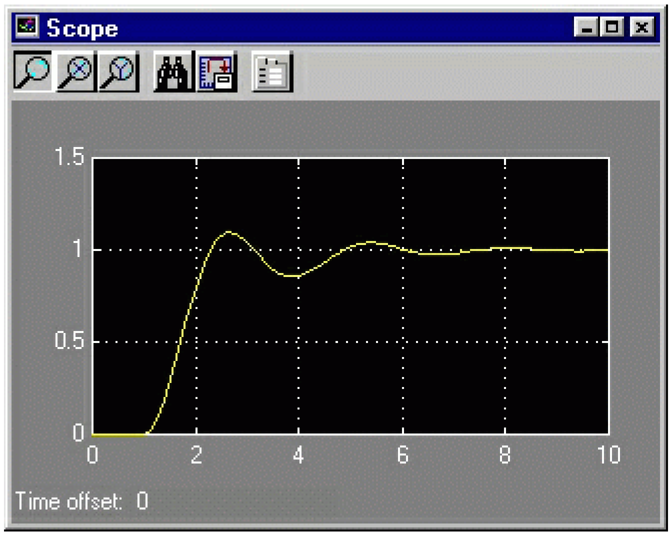
Gain:

K

Close this dialog box. Notice now that the Gain block in the Simulink model shows the variable K rather than a number.



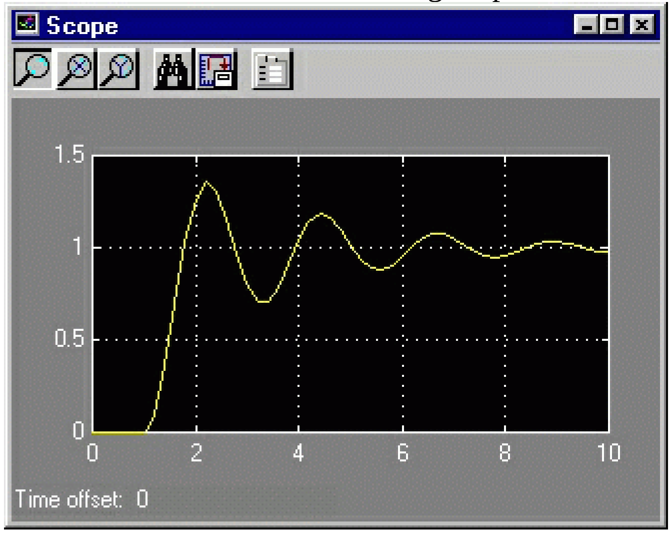
Now, you can re-run the simulation and view the output on the Scope. The result should be the same as before.



Now, if any calculations are done in MATLAB to change any of the variable used in the Simulink model, the simulation will use the new values the next time it is run. To try this, in MATLAB, change the gain, K, by entering the following at the command prompt.

```
K=5
```

Start the Simulink simulation again, bring up the Scope window, and hit the auto-scale button. You will see the following output which reflects the new, higher gain.



Besides variable, signals, and even entire systems can be exchanged between MATLAB and Simulink. For more information see below

Simulink Basics Tutorial - Interaction With MATLAB

Defining Block Parameters Using MATLAB Variables
Exchanging Signals With MATLAB
Extracting Models From Simulink Into MATLAB

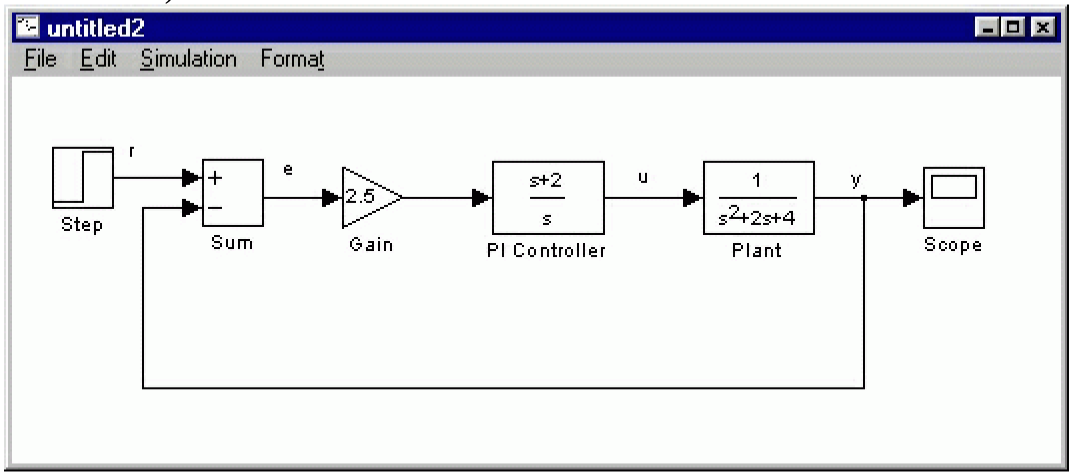
We will examine three of the ways in which Simulink can interact with MATLAB.

- Block parameters can be defined from MATLAB variable.
- Signals can be exchanged between Simulink and MATLAB.
- Entire systems can be extracted from Simulink into MATLAB.

Block Parameters from MATLAB Variables

Often, a controller will be designed in MATLAB and verified in a Simulink model. Normally, numerical parameters such as gains and controller transfer functions are entered into simulink manually by entering the numbers in the block dialog boxes. Rather than enter numbers directly, it is possible to use MATLAB variable in Simulink block dialog boxes.

For example, bring up the Simulink model built in the Basics tutorial (or click here to download it.)



In this case, the complete controller transfer function is:

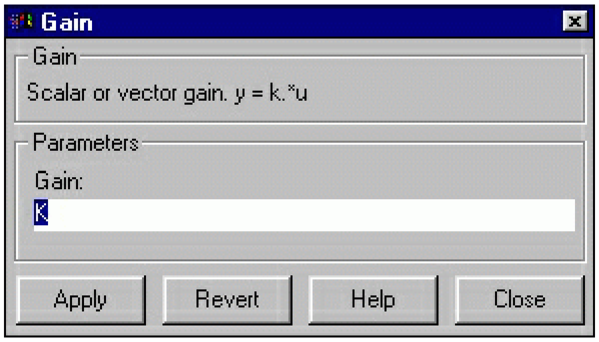
```
          s+2
     2.5 -----
           s
```

Suppose this transfer function were generated by some computation in MATLAB. In this case, there would most likely be three variable, the numerator polynomial, the denominator polynomial, and the gain. Enter the following commands in MATLAB to generate these variable.

```
      K=2.5
      num=[1 2]
      den=[1 0]
```

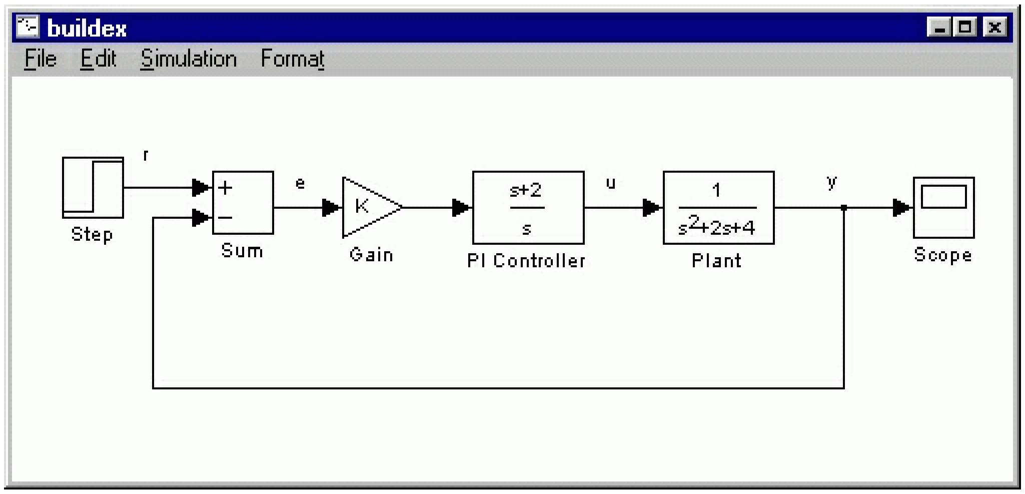
These variable can now be used in the blocks in Simulink. In your simulink model, double-click on the Gain block. Enter the following in the Gain field.

```
      K
```



Gain
Scalar or vector gain. y = k.*u
Parameters
Gain:
K

Close this dialog box. Notice now that the Gain block in the Simulink model shows the variable K rather than a number.

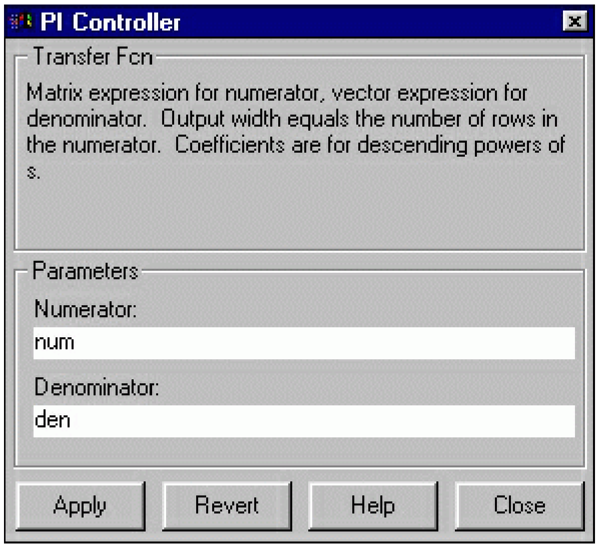


Double-click on the PI Controller block. Enter the following into the Numerator field.

```
num
```

Enter the following into the Denominator field.

```
den
```

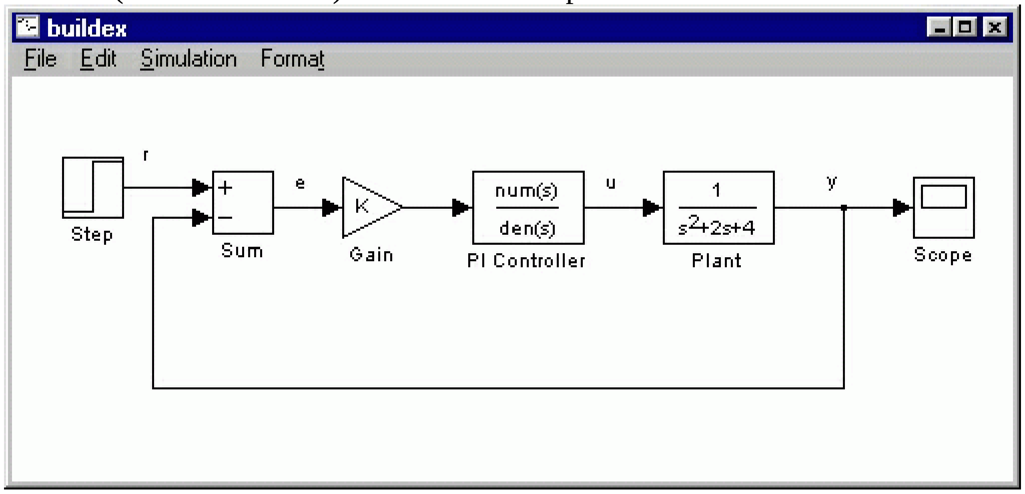


Matrix expression for numerator, vector expression for denominator. Output width equals the number of rows in the numerator. Coefficients are for descending powers of s.

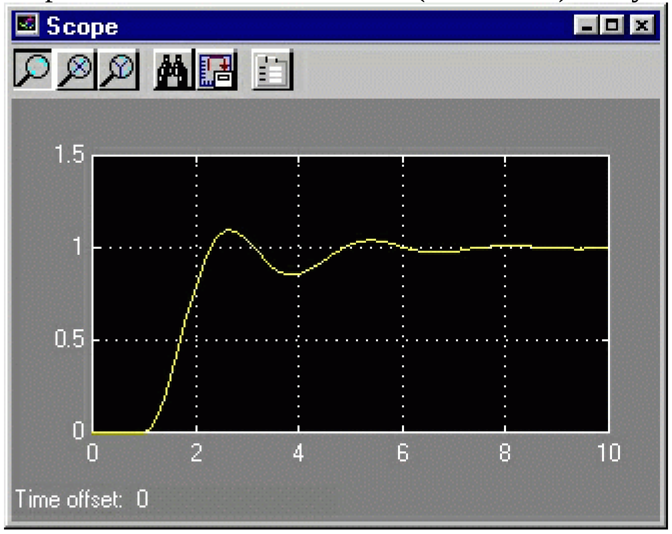
Numerator: num

Denominator: den

Close this dialog box. Notice now that the PI Controller block shows the variable num and den (as functions of s) rather than an explicit transfer function.



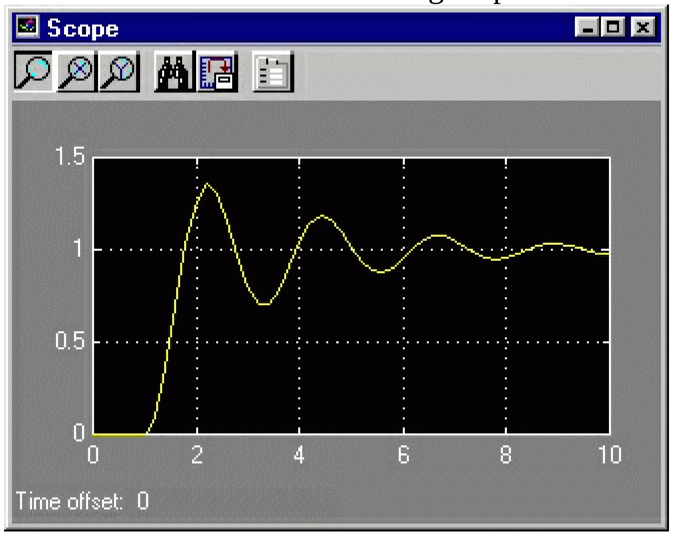
You can simulate the model with the MATLAB variable parameters. Select **Start** from the **Simulation** menu to run the simulation. Double-click on the Scope block to view its output. Hit the autoscale button (binoculars) and you should see the following.



Now, if any calculations are done in MATLAB to change any of the variable used in the Simulink model, the simulation will use the new values the next time it is run. To try this, in MATLAB, change the gain, K, by entering the following at the command prompt.

```
        K=5
```

Start the Simulink simulation again, bring up the Scope window, and hit the autoscale button. You will see the following output which reflects the new, higher gain.



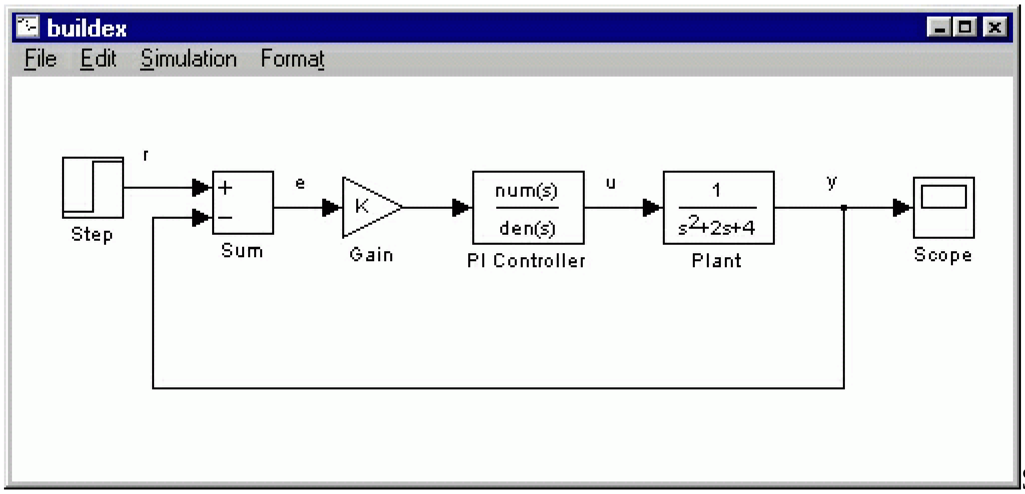
To download the model with MATLAB variable parameters, click here.

Exchanging Signals with MATLAB

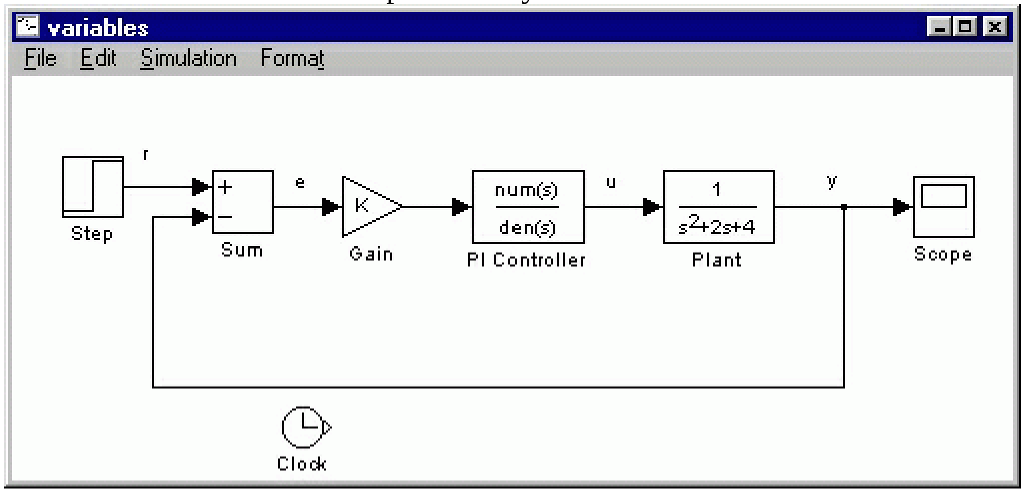
Sometimes, we would like to use the results of a Simulink simulation in the MATLAB command window for further calculations and plotting. Less often, we would like to generate signals in MATLAB which we then use as inputs in a Simulink model. These tasks are accomplished through the use of the To Workspace Sink Block and the From

Workspace Source Block. We will only transfer signals from Simulink to MATLAB. Doing the reverse is a very similar process.

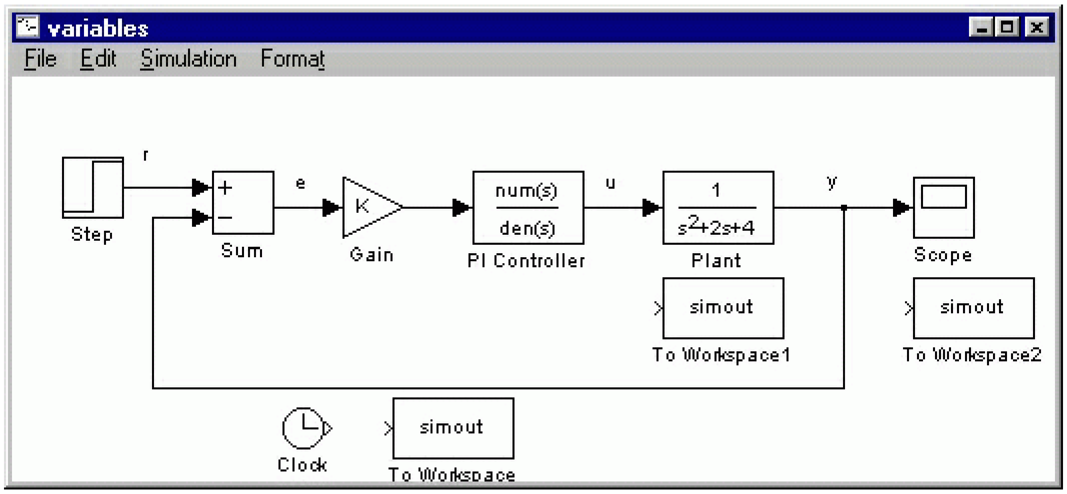
The To Workspace Sink Block saves a signal as a vector in the MATLAB Workspace. Open the model which you used previously in this tutorial or click here to download the model. Be sure that the variable K (=5), num (=[1 2]), and den (=[1 0]) are defined in MATLAB.



Suppose we would like to use both the output signal and the control signal for calculations in MATLAB. We will save these two variables as well as a time signal from our Simulink model. First, you need to generate a time signal. Open the Sources window by double-clicking the Sources icon in the main Simulink window. Drag the Clock block from the Sources window to the lower portion of your Simulink model.

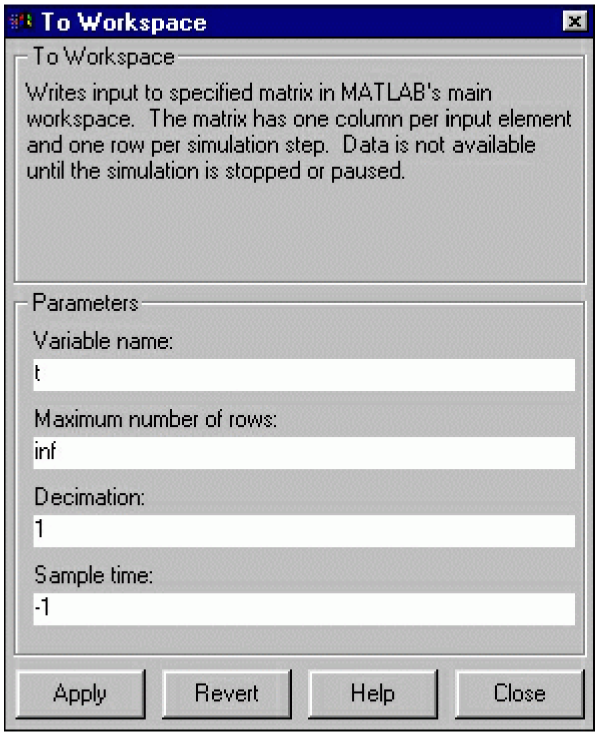


Now, open the Sinks window and drag three instances of the To Workspace block to your Simulink window, arranged approximately as shown below.



Before connecting these blocks to the rest of your system, first you will name the variable to which they output. The lower To Workspace block will output the time signal to the MATLAB variable. Double-click on this block and enter the following in the Variable Name field.

t



To Workspace

Writes input to specified matrix in MATLAB's main workspace. The matrix has one column per input element and one row per simulation step. Data is not available until the simulation is stopped or paused.

Parameters

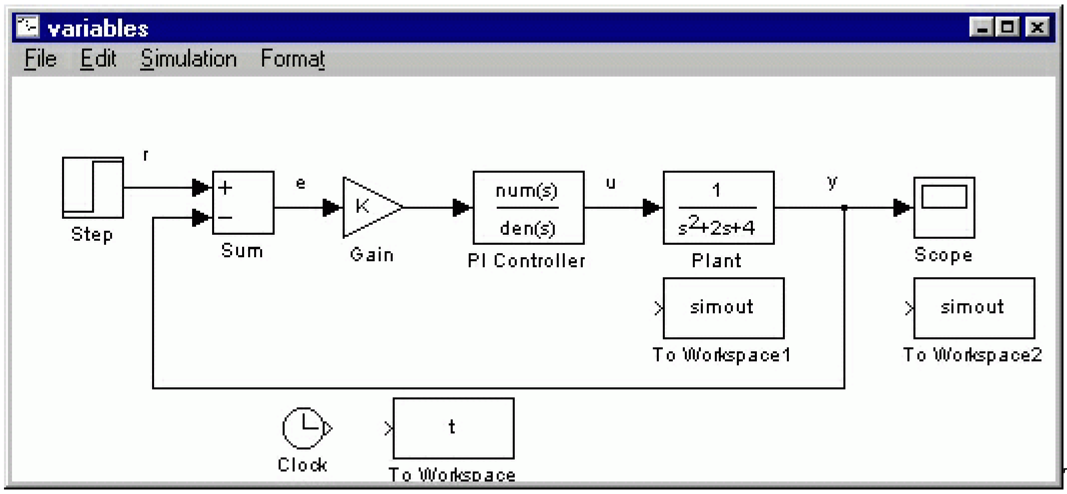
Variable name: t

Maximum number of rows: inf

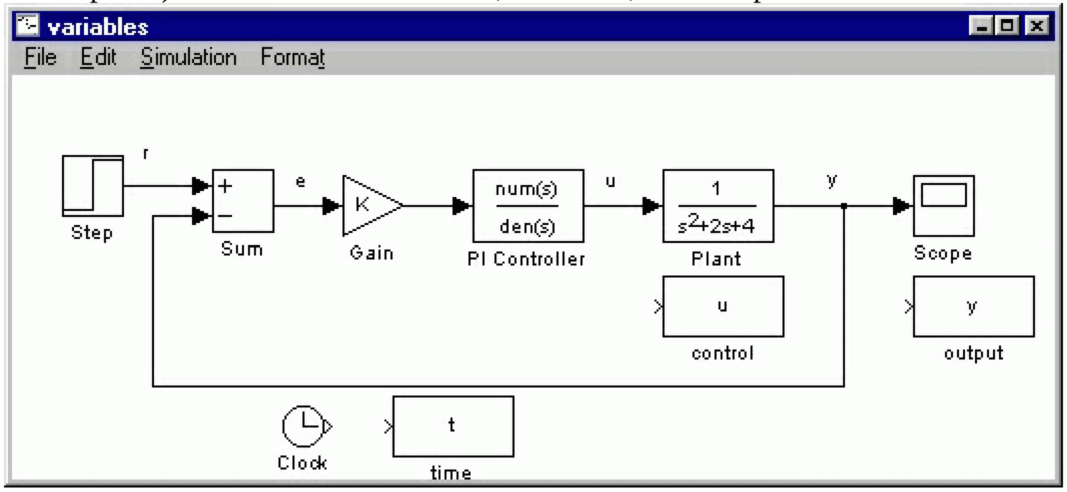
Decimation: 1

Sample time: -1

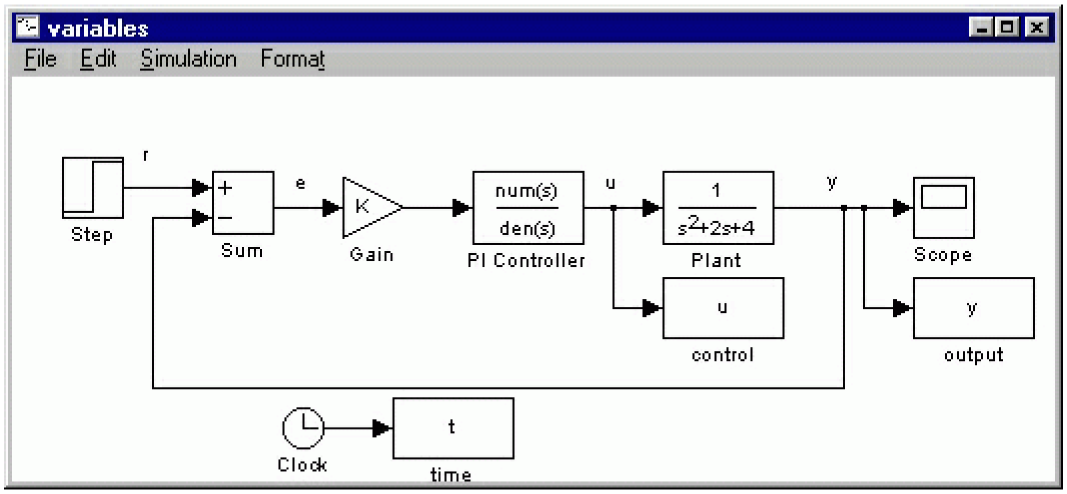
Close the dialog box. Notice that the lower To Workspace block shows a t.



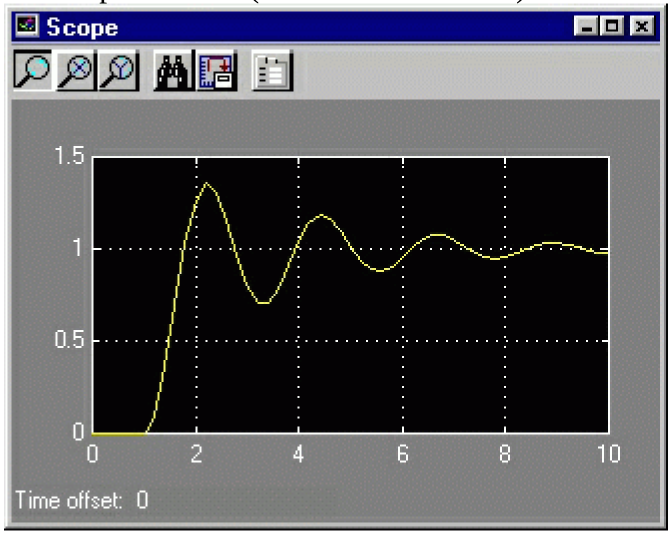
The To Workspace block near the Plant block will output the control signal to the MATLAB variable u. Edit this block to output to the variable u. The last To Workspace block will output the output signal to the MATLAB variable y. Edit this block to output to the variable y. Also, for better clarity, change the labels (by clicking on the exiting labels "To Workspace") of these blocks to "time", "control", and "output".



Now, you will connect these blocks to the rest of your system. Draw a line from the Clock block to the time (t) block. Tap a line off of the control signal (the line between the PI Controller block and the Plant block) and connect it to the control (u) block. Remember, to tap off an existing line, hold the Ctrl key while drawing the line. Tap a line off the output signal line (the line which enters the Scope block) and connect it to the output (y) block. Your system should appear as follows.



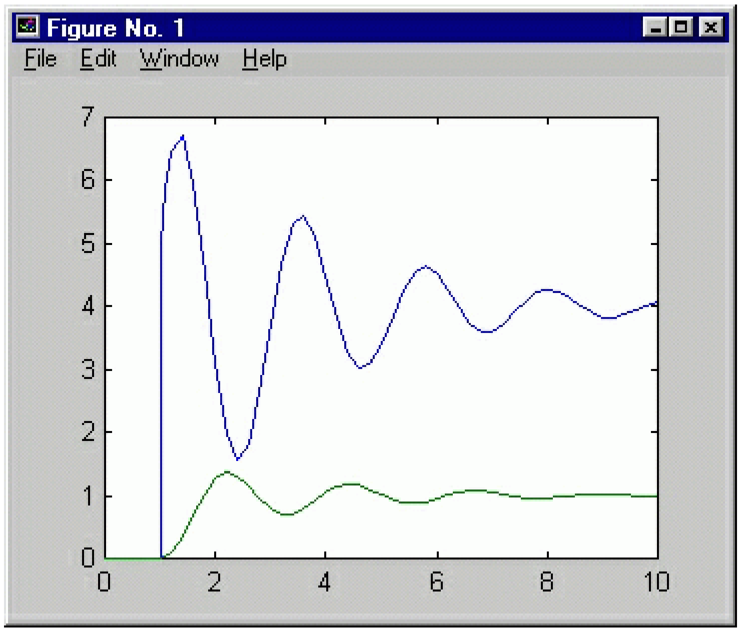
Start the simulation (**Start** from the **Simulation** menu). You can still view the output in the Scope window (remember autoscale).



You can now examine the outputted variable in the MATLAB window. Plot u and y vs. t by entering the following command.

```
        plot(t,u,t,y);
```

Note that it is important to plot each of these variables against the time vector generated by Simulink, since the time between elements in the signal vectors u and y may be unequal, particularly near a discontinuity such as the step input. Your plot of u (blue) and y (green) should appear as follows.



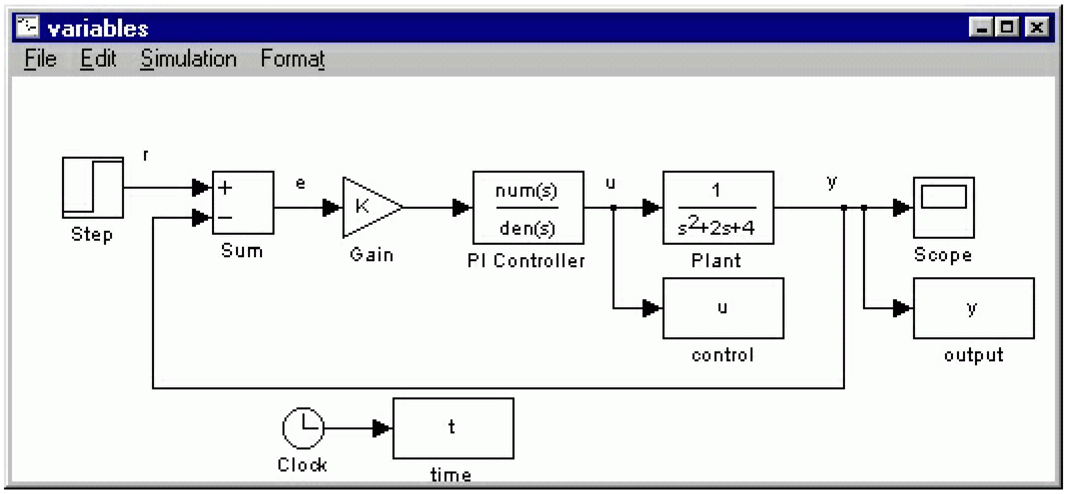
To download the model with outputs to MATLAB variable, click here.

Extracting Models From Simulink into MATLAB

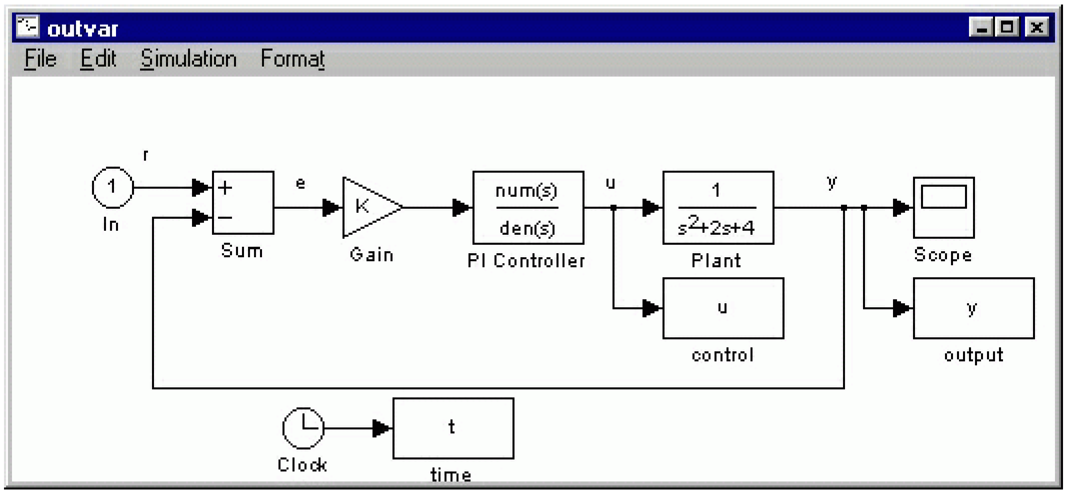
Sometimes, we may build a complicated model in simulink and would like to derive either a transfer function or a state space model of the entire system. In order to do this, you first need to define the input and output signals of the model to be extracted. These virtual signals can be any signal in a model, for example, if we can generate an input-to-output transfer function or a disturbance-to-error transfer function. These signals are defined using the In and Out Connection Blocks.

Once the input/output model is defined, the Simulink model must be saved to a .mdl file. This file is then referenced in the MATLAB command window by the *linmod* command.

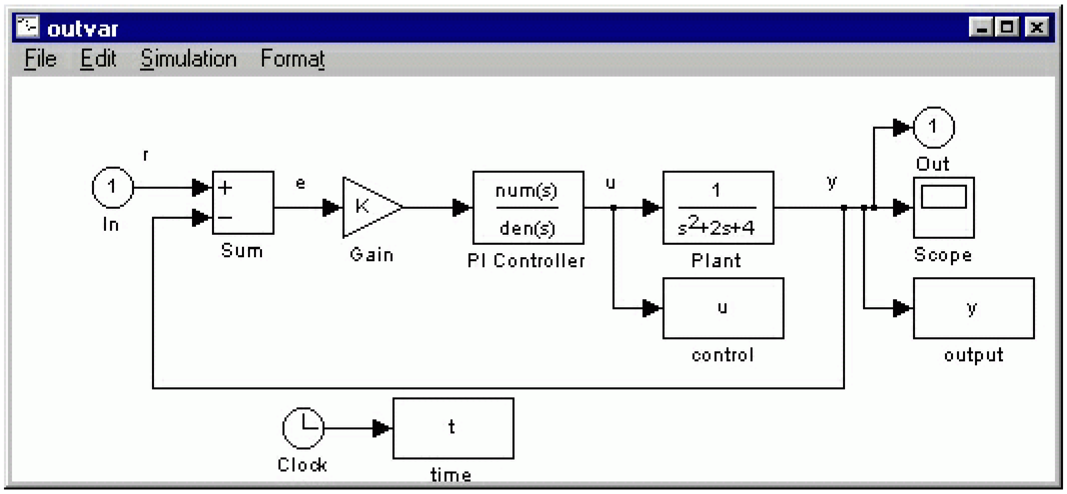
To demonstrate this, bring up your model from the previous section of this tutorial (or click here to download it). Be sure that the variable K (=5), num (=[1 2]), and den (=[1 0]) are defined in MATLAB.



You will be extracting a closed-loop reference-to-output model. Therefore, The virtual input will be put in place of the step input to the system. First, delete the Step block (click on it and hit the delete key). The previous line will remain with an open input terminal where it used to connect to the Step. Open the Connections window from the main Simulink window. Drag an In Block from the Connections window to your model window in place of the Step block you just deleted. Move the In block until the output terminal of the In block touches the open input terminal of the left over line. The line should attach to the In block.



The virtual output does not need to replace an existing block - the signal can be tapped off an existing line. Drag an Out block from the Connections window and place it just above the Scope block. Tap a line off the output signal (hold Ctrl) and connect it to the out block.



Now, save this model under a new name. Call it **mymodel.mdl**. You can download a version here.

At the MATLAB prompt, enter the following command to extract a state-space model from your model file.

```
      [A,B,C,D]=linmod('mymodel')
```

You should see the following output.

```
      A =

          -2    -9     2
           1     0     0
           0    -5     0


      B =

           5
           0
           5


      C =

           0     1     0


      D =

           0
```

This can, of course, be converted to a transfer function with the following command.

```
      [numcl,dencl]=ss2tf(A,B,C,D)
```

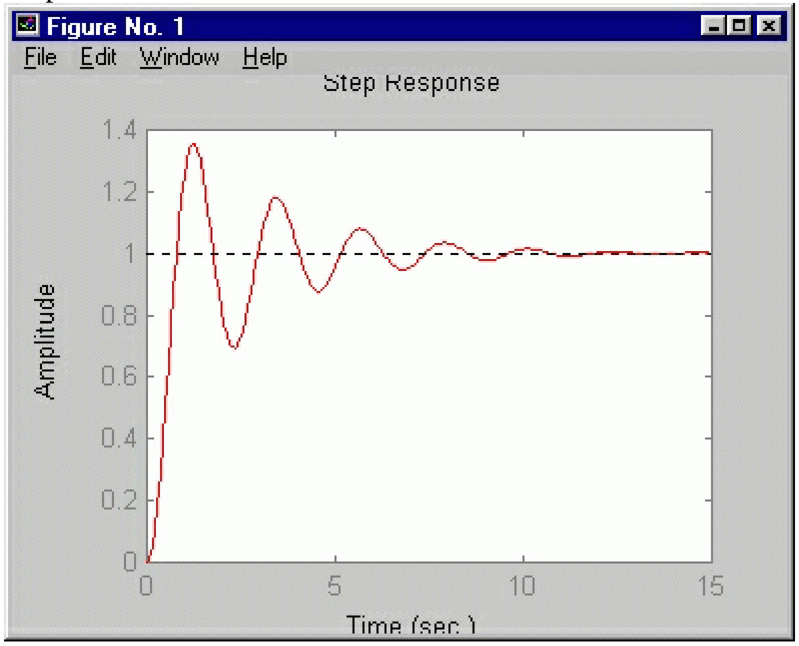
You should get the following output.

```
      numcl =

               0          0    5.0000   10.0000


      dencl =
          1.0000    2.0000    9.0000   10.0000
```

To verify that the model transfered properly, you can obtain a step response of the extracted model.

```
        step(numcl,dencl)
```

You should see the following plot which is similar to the previous Simulink Scope output.



The modeling information is continued below:

Simulink Modeling Tutorial

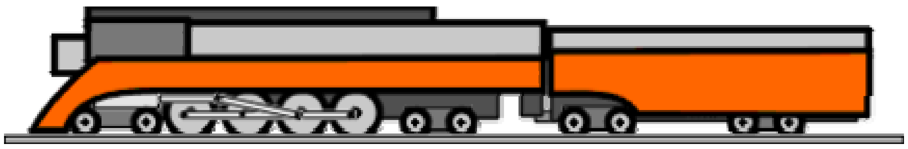
Train system
Free body diagram and Newton's law
Model Construction
Running the Model
Obtaining MATLAB Model

In Simulink, it is very straightforward to represent a physical system or a model. In general, a dynamic system can be constructed from just basic physical laws. We will demonstrate through an example.

Train system

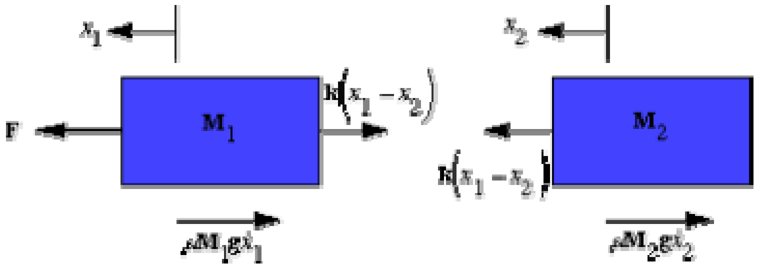
In this example, we will consider a toy train consisting of an engine and a car. Assuming that the train only travels in one direction, we want to apply control to the train so that it has a smooth start-up and stop, along with a constant-speed ride.

The mass of the engine and the car will be represented by M1 and M2, respectively. The two are held together by a spring, which has the stiffness coefficient of k. F represents the force applied by the engine, and the Greek letter, mu (which will also be represented by the letter u), represents the coefficient of rolling friction.



Free body diagram and Newton's law

The system can be represented by following Free Body Diagrams.

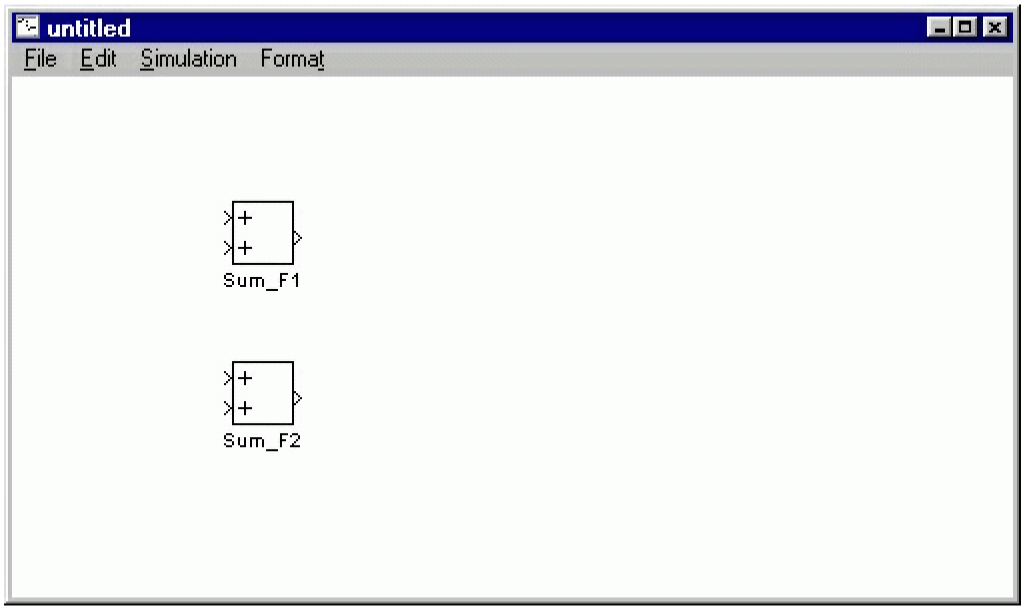


From Newton's law, you know that the sum of forces acting on a mass equals the mass times its acceleration. In this case, the forces acting on M1 are the spring, the friction and the force applied by the engine. The forces acting on M2 are the spring and the friction. In the vertical direction, the gravitational force is canceled by the normal force applied by the ground, so that there will be no acceleration in the vertical direction. We will begin to construct the model simply from the expressions:

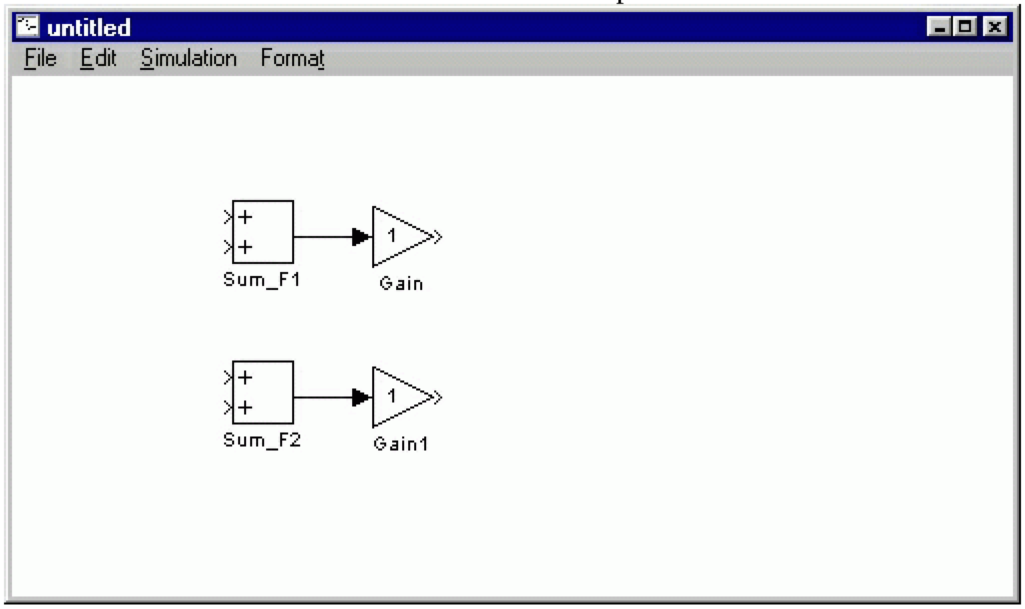
```
Sum(forces_on_M1)=M1*x1''
Sum(forces_on_M1)=M1*x1''
```

Constructing The Model

This set of system equations can now be represented graphically, without further manipulation. First, we will construct two copies (one for each mass) of the expressions sum_F=Ma or a=1/M*sum_F. Open a new model window, and drag two Sum blocks (from the Linear library), one above the other. Label these Sum blocks "Sum_F1" and "Sum_F2".



The outputs of each of these Sum blocks represents the sum of the forces acting on each mass. Multiplying by 1/M will give us the acceleration. Drag two Gain blocks into your model and attach each one with a line to the outputs of the Sum blocks.



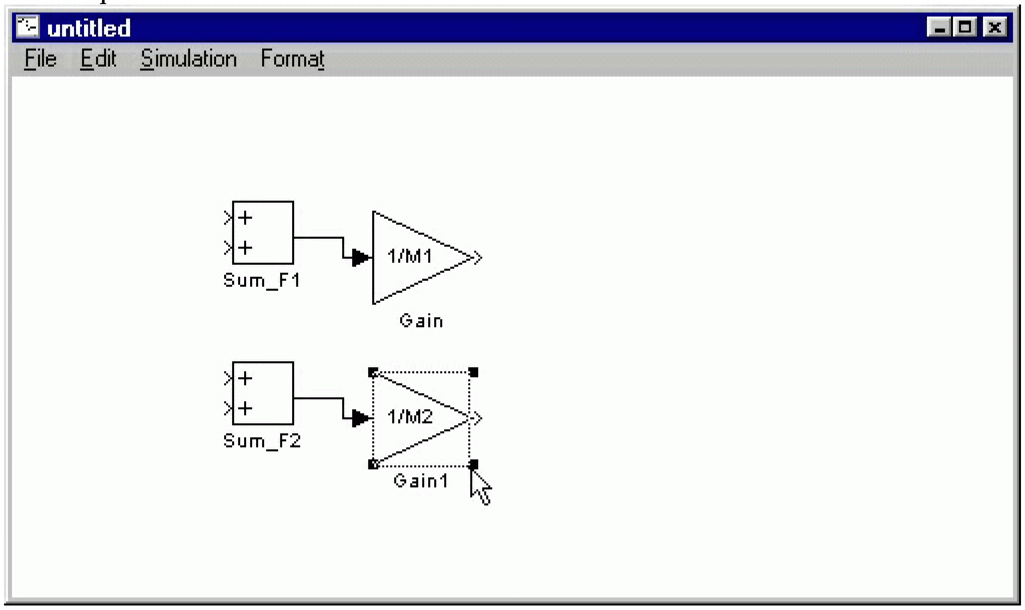
These Gain blocks should contain 1/M for each of the masses. We will be taking these variable as M1 and M2 from the MATLAB environment, so we can just enter the variable in the Gain blocks. Double-click on the upper Gain block and enter the following into the Gain field.

```
        1/M1
```

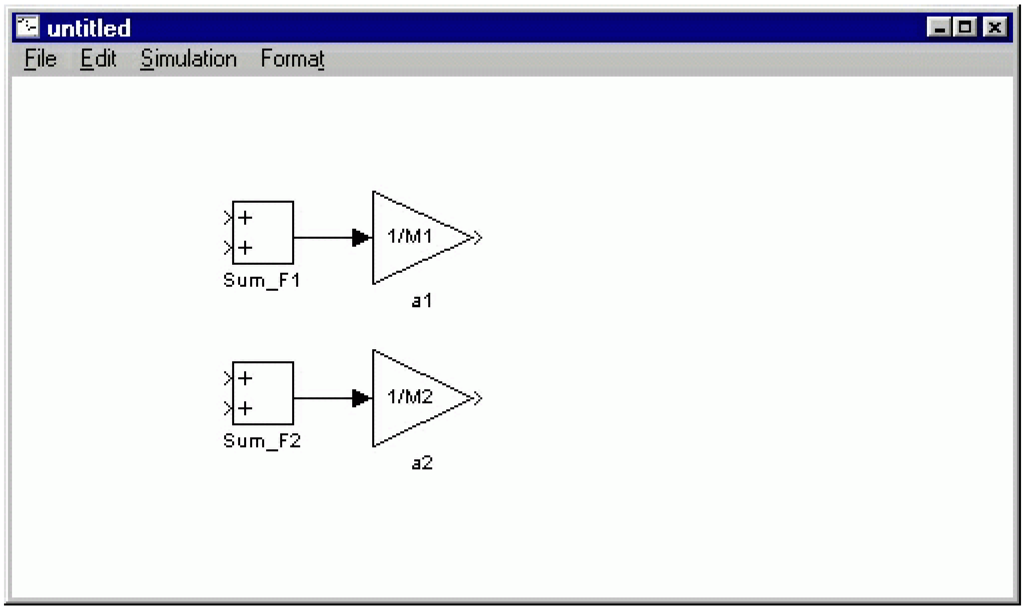
Similarly, change the second Gain block to the following.

```
        1/M2
```

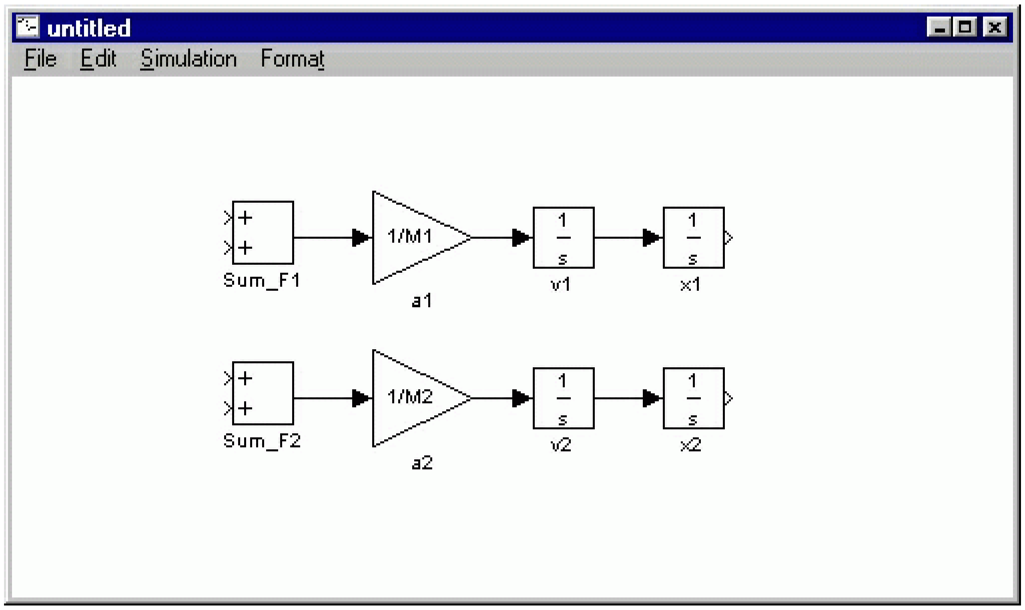
Now, you will notice that the gains did not appear in the Gain blocks, and just "-K-" shows up. This is because the blocks are two small on the screen to show 1/M2 inside the triangle. The blocks can be resized so that the actual gain can be seen. To resize a block, select it by clicking on it once. Small squares will appear at the corners. Drag one of these squares to stretch the block.



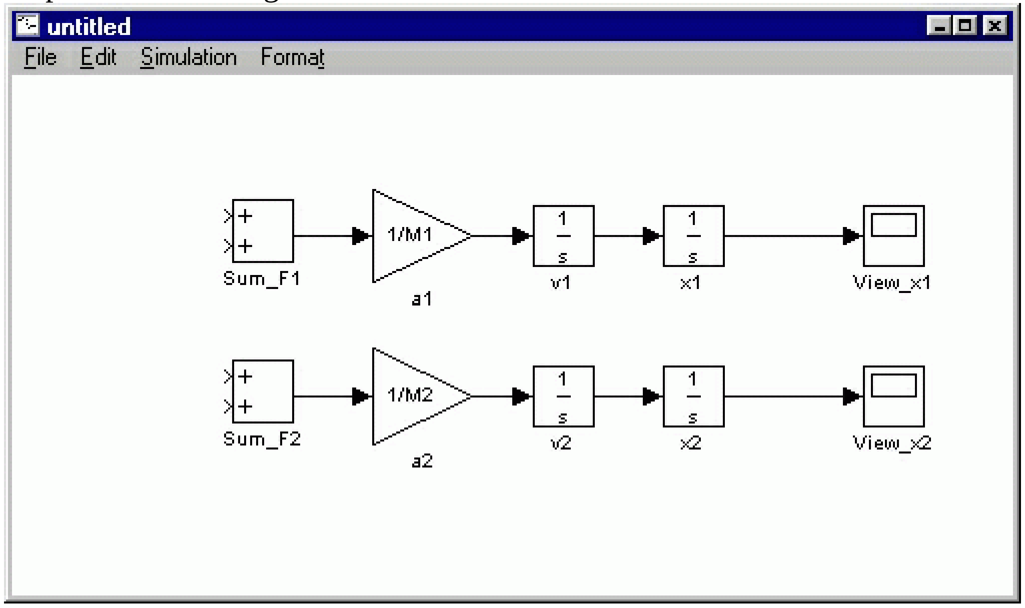
When the Gain blocks are of sufficient size to display the actual gains, re-align them with the signal line output from the Sum blocks. Also, label these two Gain blocks "a1" and "a2".



The outputs of these gain blocks are the accelerations of each of the masses. We are interested in both the velocities and the positions of the masses. Since velocity is the integral of acceleration, and position is the integral of velocity, we can generate these signals using integrator blocks. Drag two integrator blocks into your model for each of the two accelerations. Connect them with lines in two chains as shown below. Label these integrators "v1", "x1", "v2", and "x2" since these are the signals these integrators will generate.



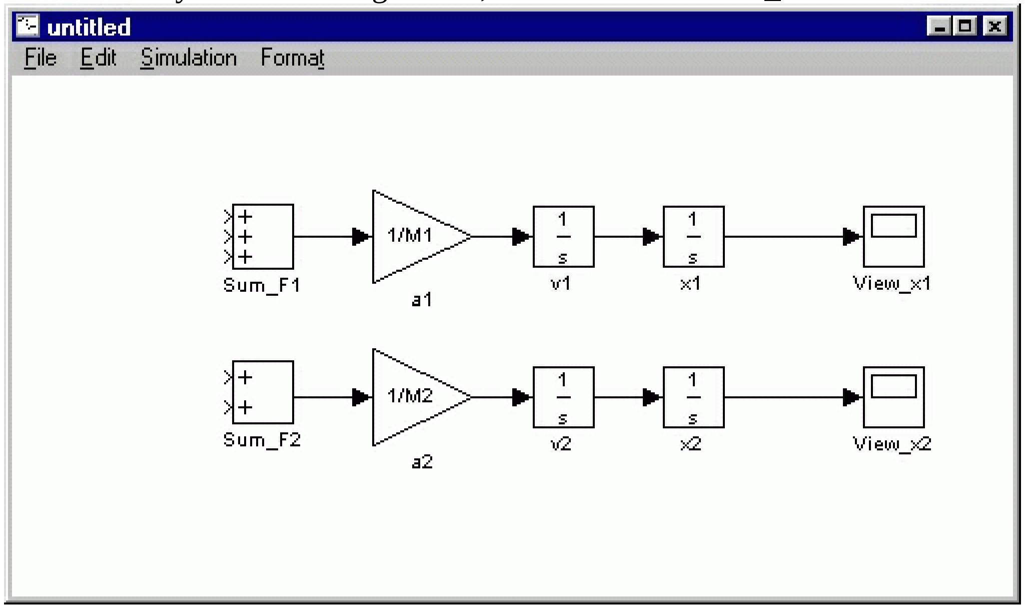
Now, drag two Scopes from the Sinks library into your model and connect them to the outputs of these integrators. Label them "View_x1" and "View_x2".



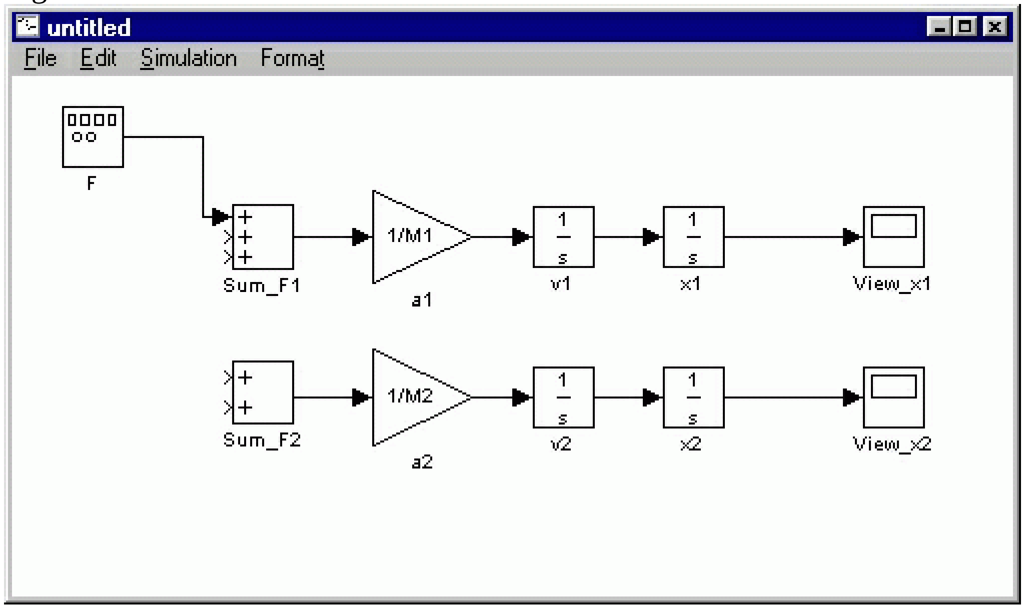
Now we are ready to add in the forces acting on each mass. First, you need to adjust the inputs on each Sum block to represent the proper number (we will worry about the sign later) of forces. There are a total of 3 forces acting on M1, so change the Sum_F1 block's dialog box entry to:

+++

There are only 2 forces acting on M2, so we can leave Sum_F1 alone for now.



The first force acting on M1 is just the input force, F. Drag a Signal Generator block from the Sources library and connect it to the uppermost input of the Sum_F1 block. Label the Signal Generator "F".



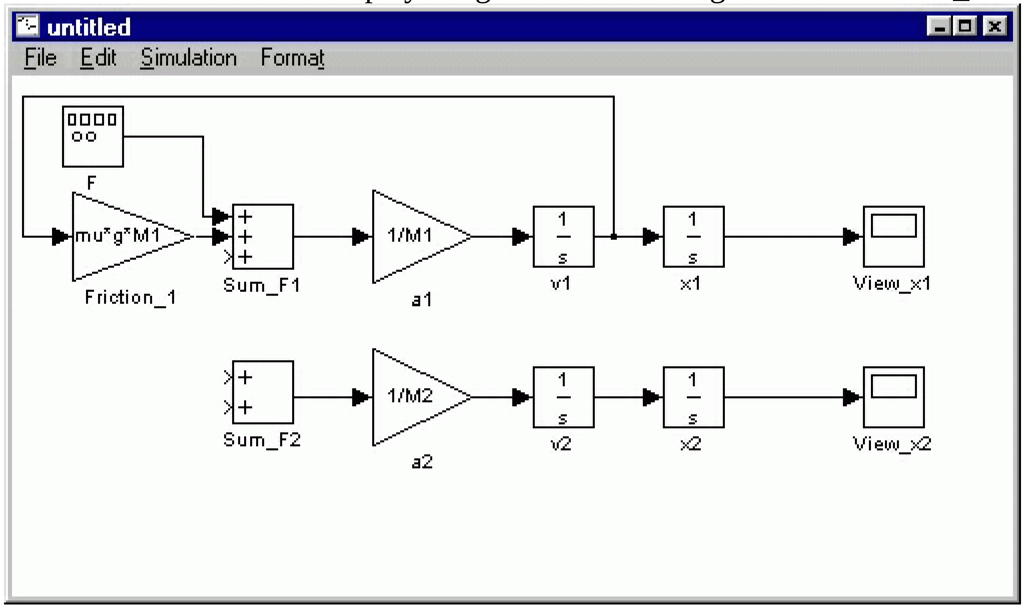
The next force acting on M1 is the friction force. This force is equal to:

```
F_friction_1=mu*g*M1*v1
```

To generate this force, we can tap off the velocity signal and multiply by a gain, mu*g*M1. Drag a Gain block into your model window. Tap off the line coming from the v1 integrator and connect it to the input of the Gain block (draw this line in several steps if necessary). Connect the output of the Gain block to the second input of Sum_F1. Change the gain of this gain block to the following.

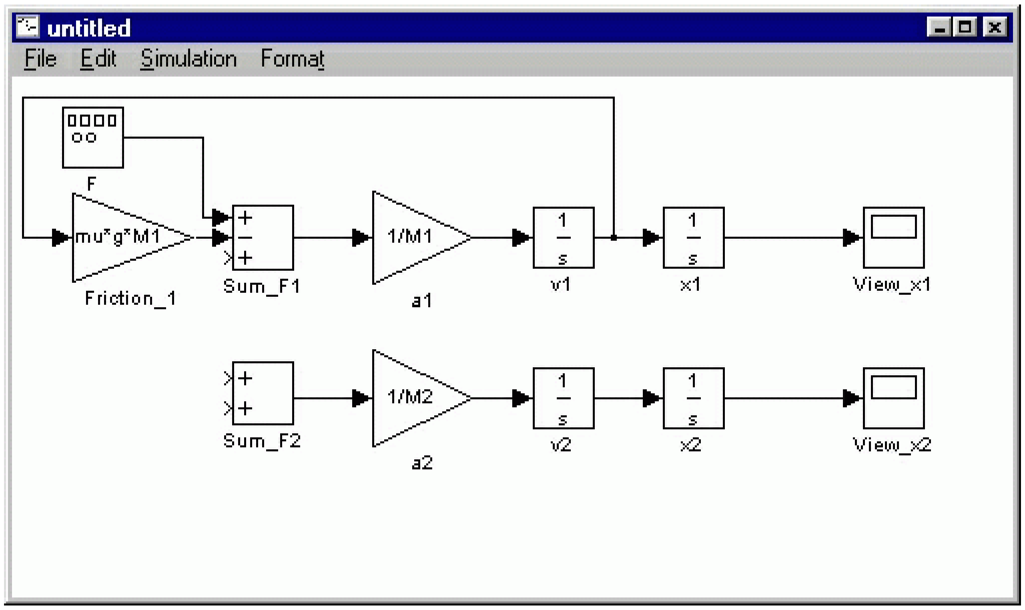
```
        mu*g*M1
```

Resize the Gain block to display the gain and label the gain block Friction_1.



This force, however, acts in the *negative* x1-direction. Therefore, it must come into the Sum_F1 block with *negative* sign. Change the list of signs of Sum_F1 to

```
+-+
```



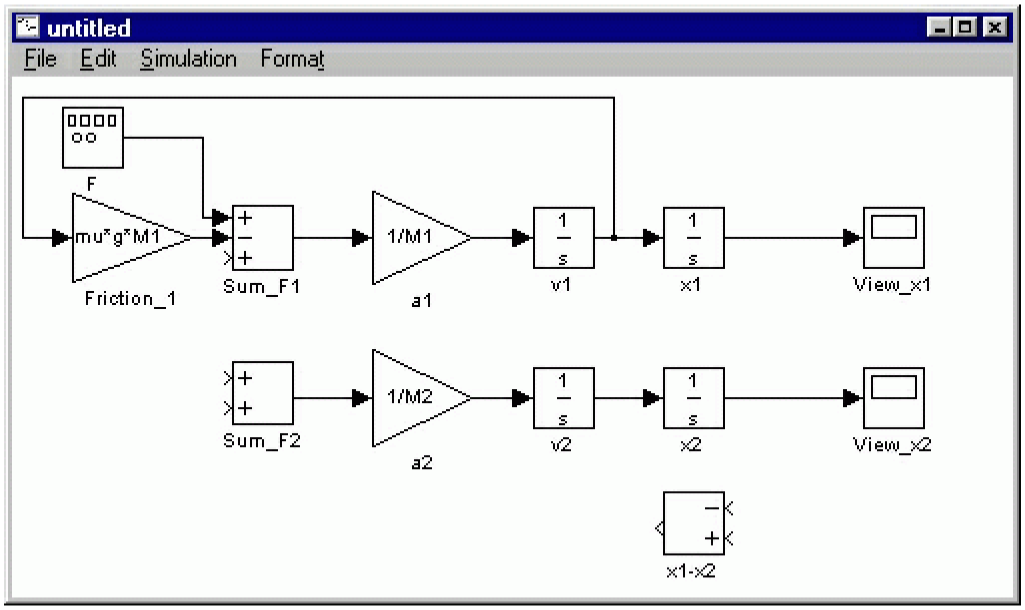
The last force acting on M1 is the spring force between masses. This is equal to:

```
k*(x1-x2)
```

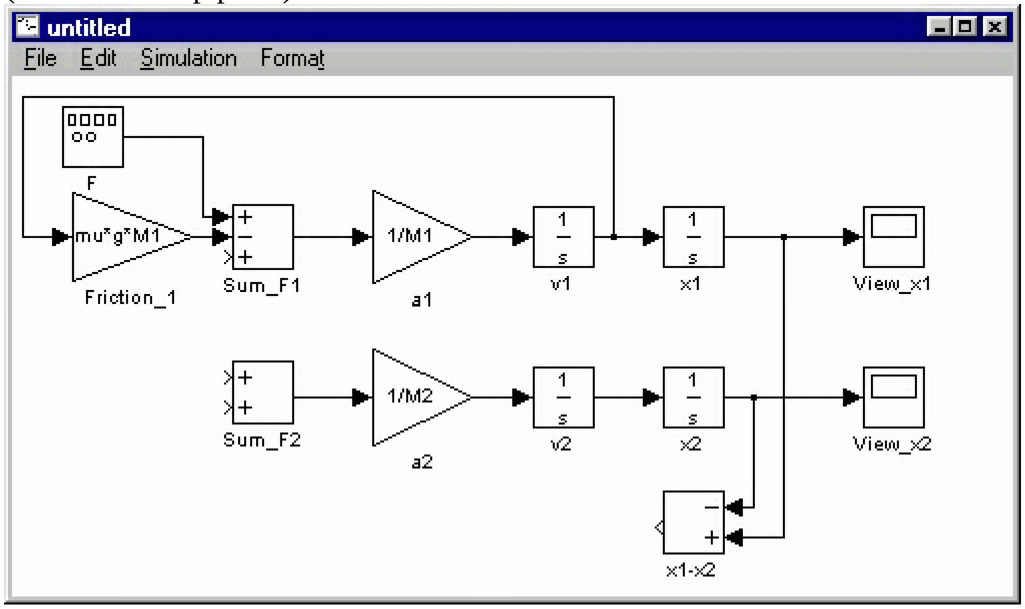
First, we need to generate (x1-x2) which we can then multiply by k to generate the force. Drag a Sum block below the rest of your model. Label it "(x1-x2)" and change its list of signs to

```
-+
```

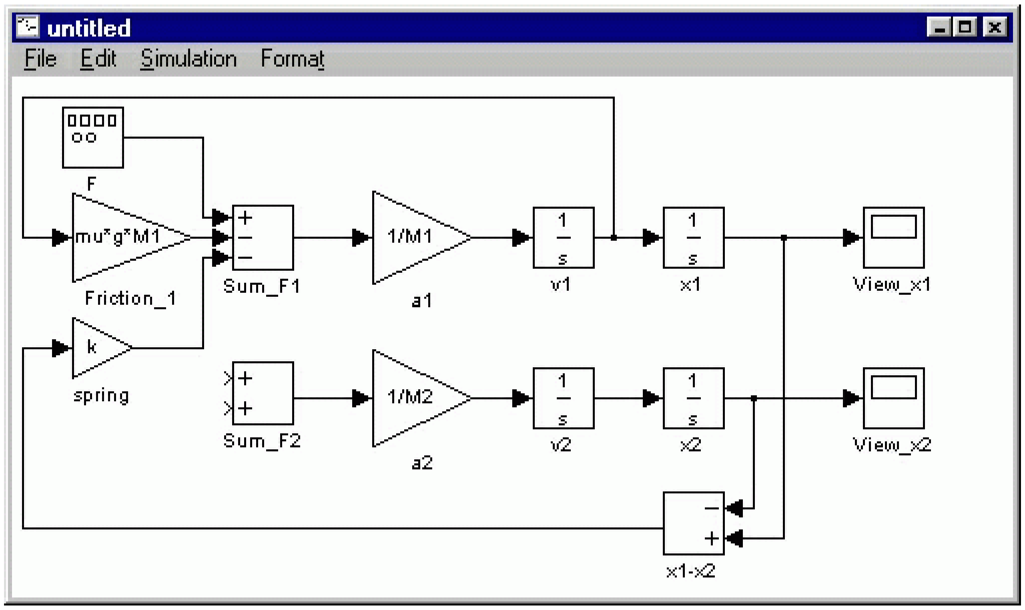
Since this summation comes from right to left, we need to flip the block around. Select the bloc by single-clicking on it and select **Flip** from the **Format** menu (or hit Ctrl-F). You should see the following.



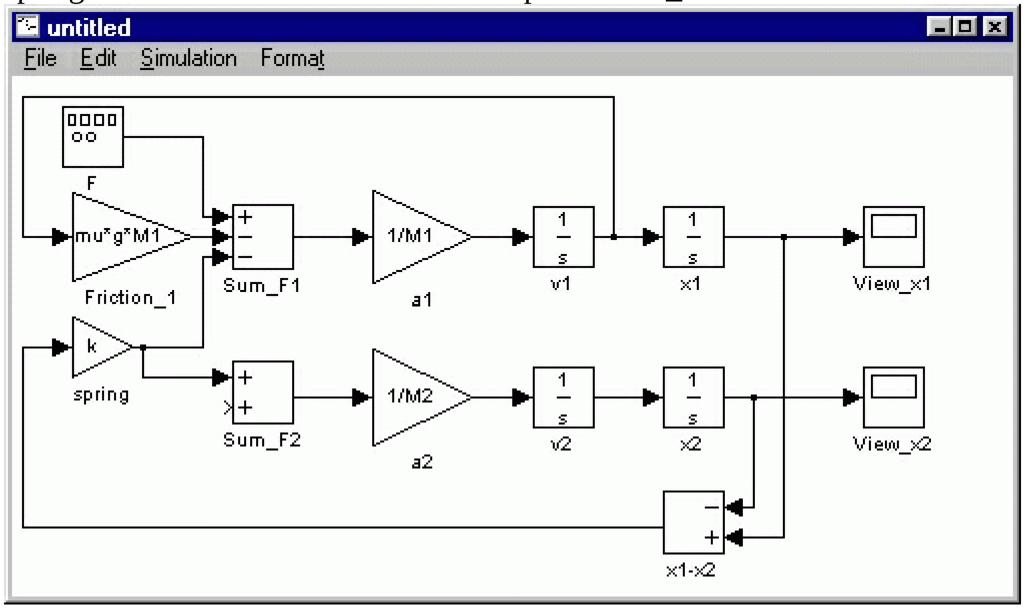
Now, tap off the x2 signal and connect it to the negative input of the (x1-x2) Sum block. Tap off the x1 signal and connect it to the positive input. This will cause the lines to cross. Lines may cross, but they are only actually connected where a small block appears (such as at a tap point).



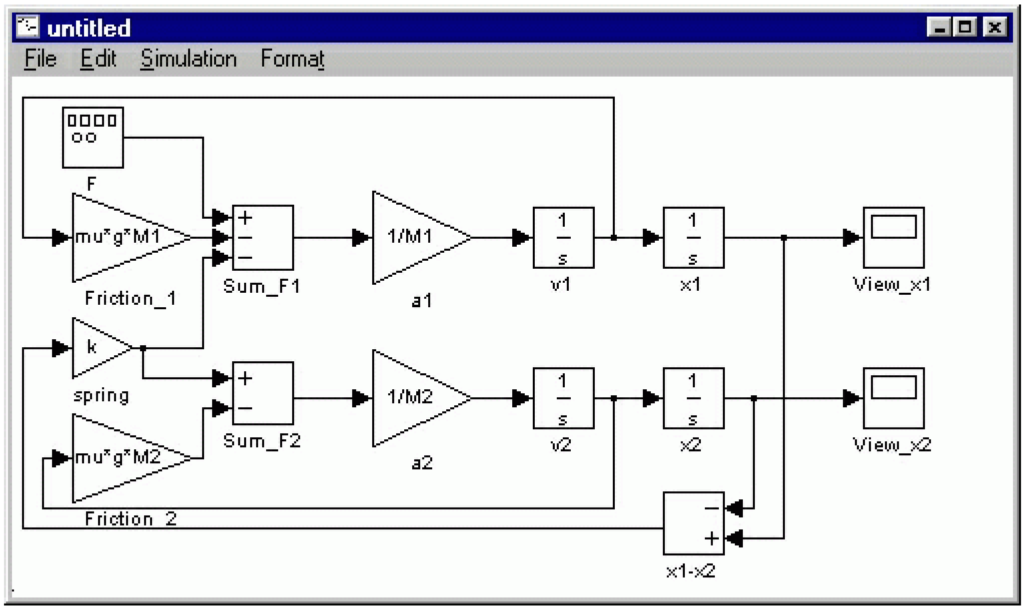
Now, we can multiply this position difference by the spring constant to generate the spring force. Drag a Gain block into your model to the left of the Sum blocks. Change it's value to k and label it "spring". Connect the output of the (x1-x2) block to the input of the spring block, and the output of the spring block to the third input of Sum_F1. Change the third sign of Sum_F1 to negative (use +--).



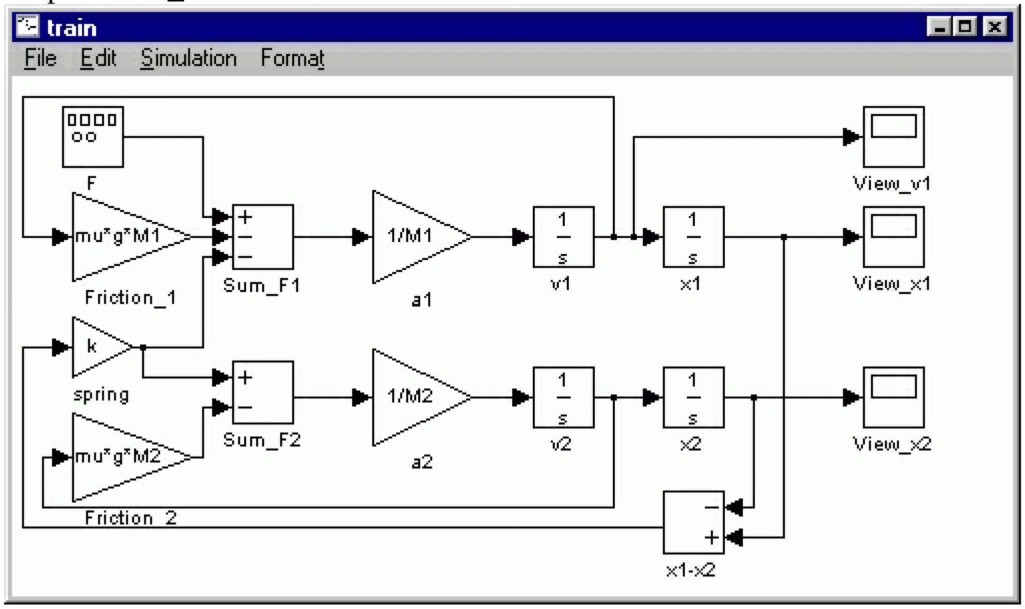
Now, we can apply forces to M2. For the first force, we will use the same spring force we just generated, except that it adds in with positive sign. Simply tap off the output of the spring block and connect it to the first input of Sum_F2.



The last force to add in the the friction on M2. This is done in the exact same manner as the friction on M1, tapping off v2, multiplying by a gain of mu*g*M2 and adding to Sum_F2 with negative sign. After constructing this, you should have the following.



Now the model is complete. We simply need to supply the proper input and view the proper output. The input of the system will be the force, F, provided by the engine. We already have placed the function generator at the input. The output of the system will be the velocity of the engine. Drag a Scope block from the Sinks block library into your model. Tap a line off the output of the "v1" integrator block to view the output. Label the scope "View_v1".



Now, the model is complete. Save your model in any file you like. You can download the completed model here.

Running the Model

Before running the model, we need to assign numerical values to each of the variable used in the model. For the train system, let

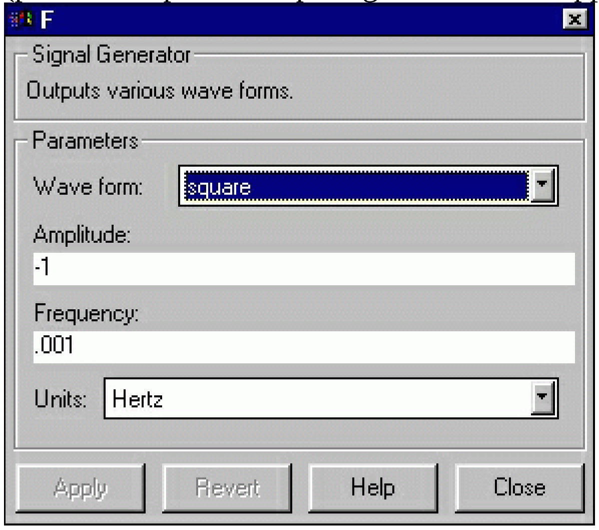
- M1 = 1 kg
- M2 = 0.5 kg
- k = 1 N/sec
- F= 1 N
- u = 0.002 sec/m
- g = 9.8 m/s^2

Create an new m-file and enter the following commands.

```
M1=1;
M2=0.5;
k=1;
F=1;
mu=0.002;
g=9.8;
```

Execute your m-file to define these values. Simulink will recognize MATLAB variable for use in the model.

Now, we need to give an appropriate input to the engine. Double-click on the function generator (F block). Select a square wave with frequency .001Hz and amplitude -1 (positive amplitude steps negative before stepping positive).



F

Signal Generator

Outputs various wave forms.

Parameters

Wave form: square

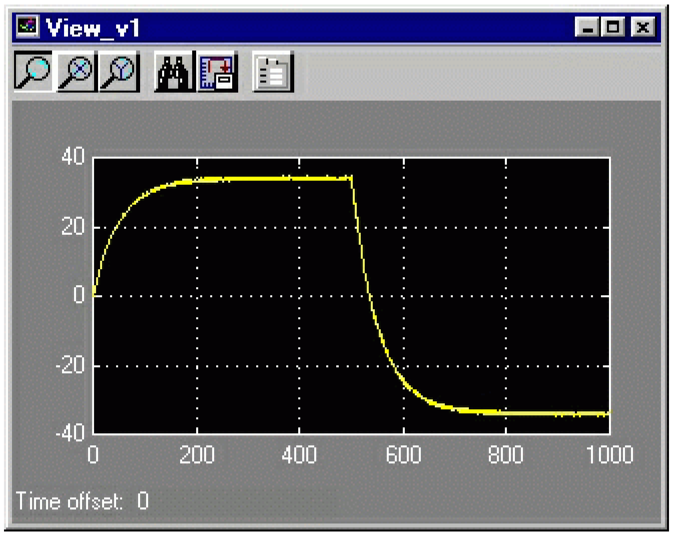
Amplitude: -1

Frequency: .001

Units: Hertz

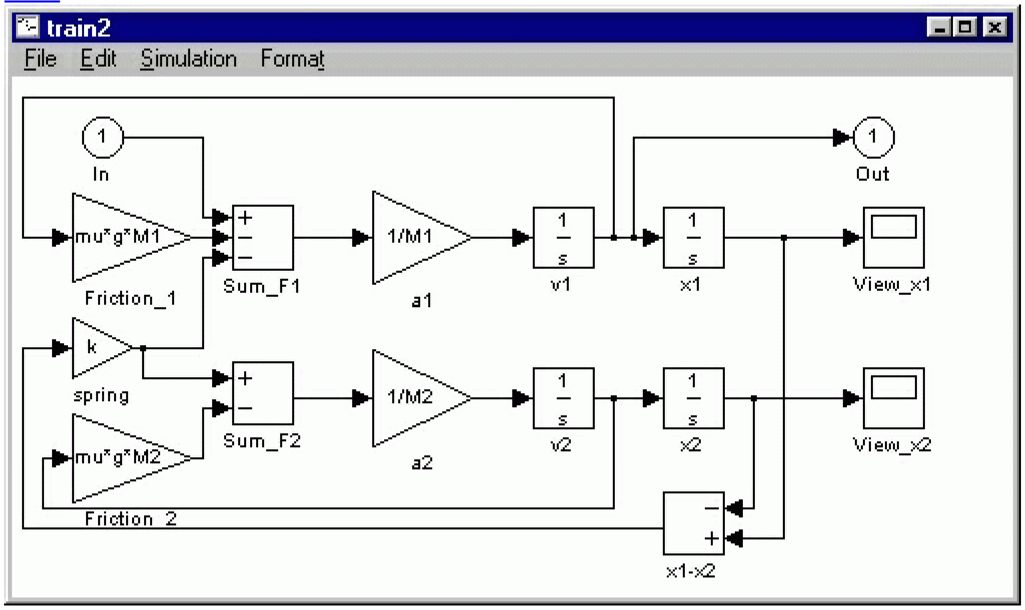
The last step before running the simulation is to select an appropriate simulation time. To view one cycle of the .001Hz square wave, we should simulate for 1000 seconds. Select **Parameters** from the **Simulation** menu and change the Stop Time field to 1000. Close the dialog box.

Now, run the simulation and open the View_v1 scope to examine the velocity output (hit autoscale). The input was a square wave with two steps, one positive and one negative. Physically, this means the engine first went forward, then in reverse. The velocity output reflects this.



Obtaining MATLAB Model

We can now extract a MATLAB model (state-space or transfer function) from out simulink model. In order to do this, delete the View_v1 scope and put an Out Block (from the Connections library) in its place. Also, delete the F function generator block and put an In Block (from the Connections library) in its place. The in and out blocks define the input and output of the system we would like to extract. For a detailed description of this process and other ways of integrating MATLAB with Simulink, click here.



Save this model as **train2.mdl** or download our version here. Now, we can extract the model into MATLAB. Enter the following command at the MATLAB command window to extract a state-space model.

```
        [A,B,C,D]=linmod('train2')
```

You should see the following output which shows a state-space model of your Simulink model.

```
A =

   -0.0196         0    1.0000   -1.0000
         0   -0.0196   -2.0000    2.0000
         0    1.0000         0         0
    1.0000         0         0         0


B =

     1
     0
     0
     0


C =

     1     0     0     0


D =

     0
```

To obtain a transfer function model, enter the following command at the MATLAB command prompt.

```
[num,den]=ss2tf(A,B,C,D)
```

You will see the following output representing the transfer function of the train system.

```
num =

         0    1.0000    0.0196    2.0000    0.0000


den =

    1.0000    0.0392    3.0004    0.0588    0.0000
```

These models are equivalent (although the states are in different order) to the model obtained by hand in the MATLAB tutorials.