SIMULINK TUTORIAL

Introduction: Concept of Dynamic System Simulation

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

Concept of signal and logic flow

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

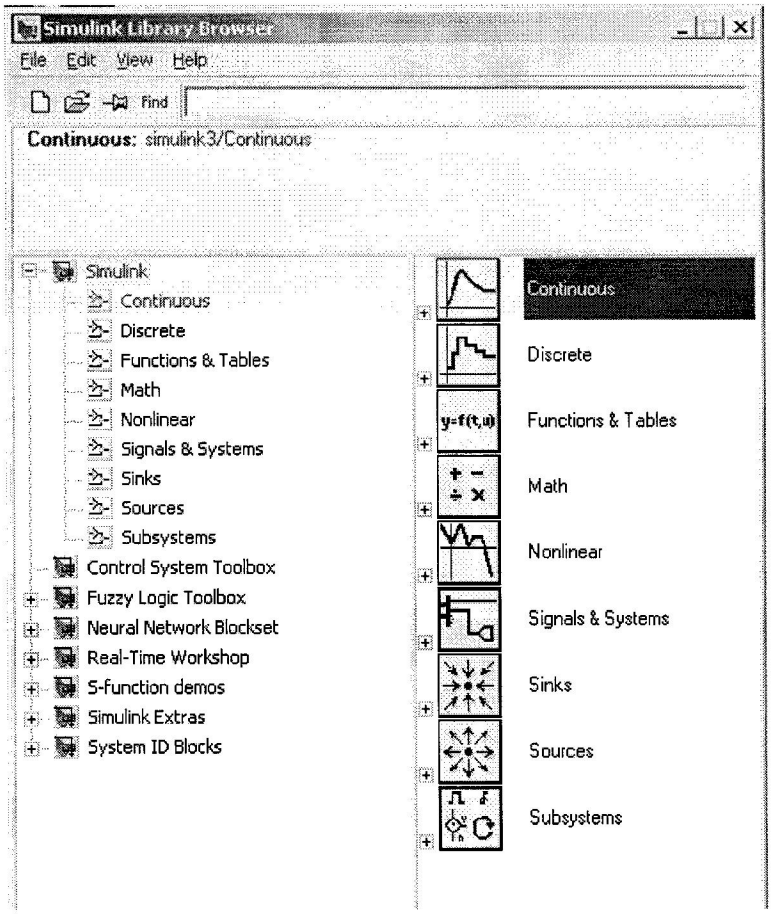

Figure 1: Simulink Library

Connecting blocks

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

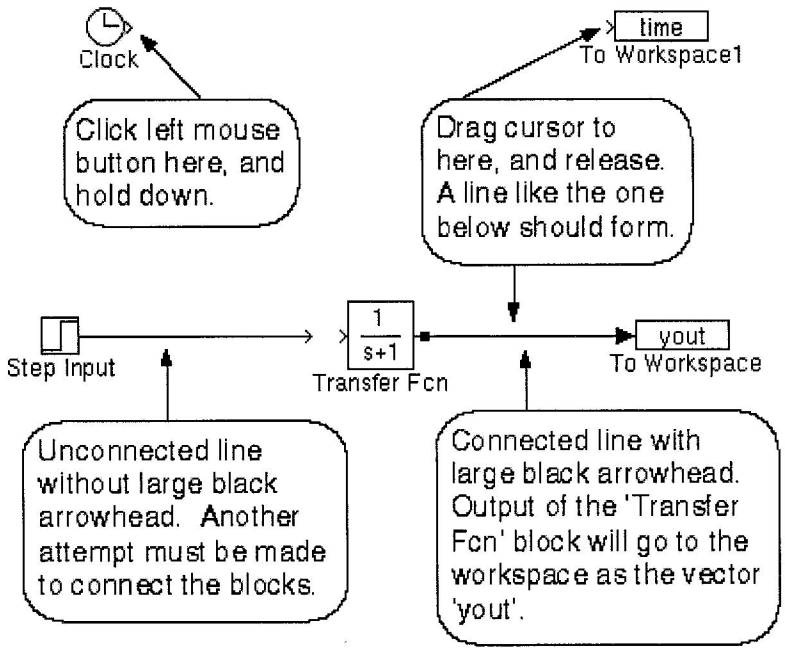

Figure 2: Connecting blocks

Sources and Sinks

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

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

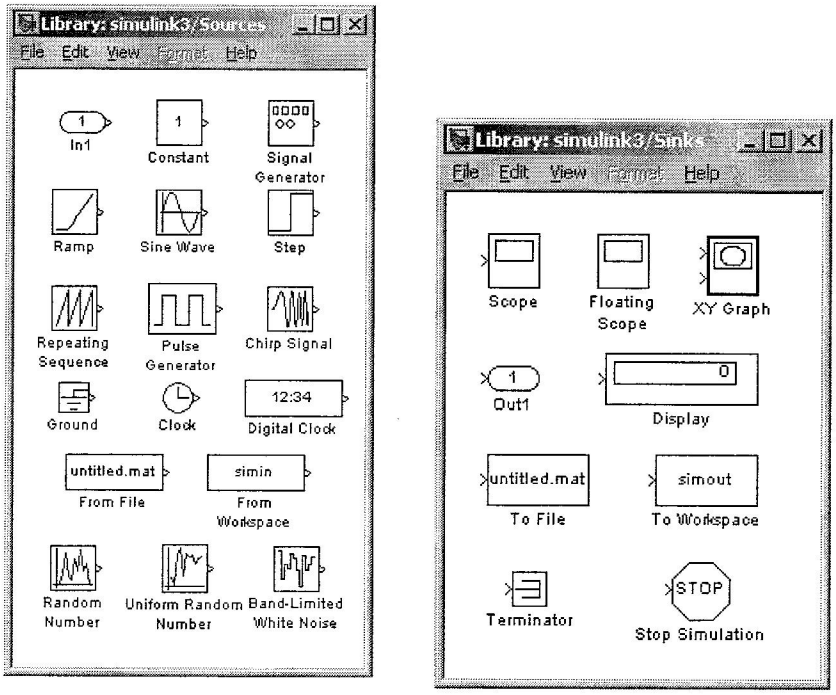

Figure 3: Sources and Sinks

Continuous and Discrete Systems

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

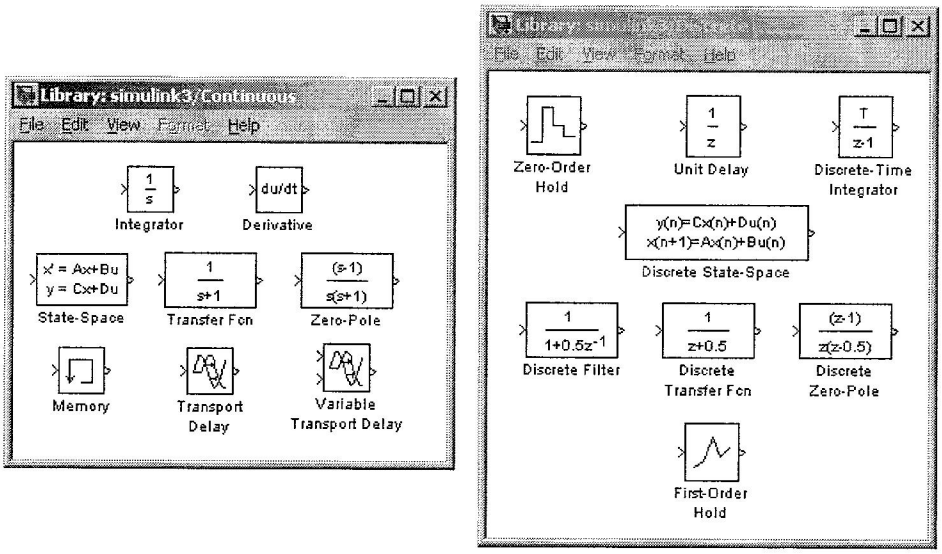

Figure 4: Continuous and Discrete Systems

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

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

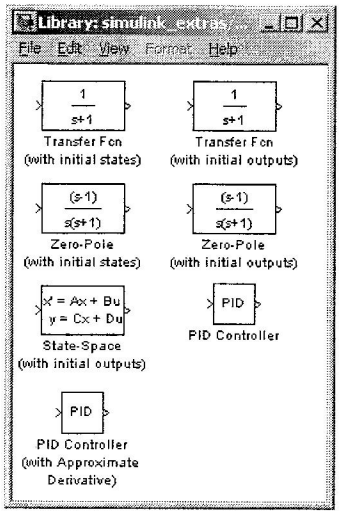

Figure 5: Advanced Linear Systems

MODELING FIRST AND SECOND ORDER SYSTEMS IN SIMULINK

First and second order differential equations are commonly studied in Dynamic Systems courses, as they occur frequently in practice. Because of this, we will discuss the basics of modeling these equations in Simulink. The first example is a low-pass RC Circuit that is often used as a filter. This is modeled using a first-order differential equation. The second example is a mass-spring-dashpot system. This system is modeled with a second-order differential equation (equation of motion). To better understand the dynamics of both of these systems were are going to build models using Simulink as discussed below. You should build both models first, then run them so you can compare how each system responds to the same input.

Modeling a First Order Equation (RC Circuit)

The RC Circuit is schematically shown in Fig. 1 below.

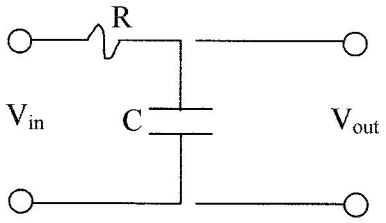

Fig. 1. The RC Circuit.

The differential equation for this is as show in (1) below.

$$\dot{x} = \frac{1}{RC}[f(t) - x] \tag{1}$$

Where $\dot{x}$ (**xdot**) is the time rate of change of the output voltage, **R** and **C** are constants, **f(t)** is the forcing function (Input voltage), and **x** is the output voltage. We are now going to take this piece by piece. First, we examine what is in the brackets and we notice that we are subtracting the term **x** from the term **f(t)**. If we imagine that each of these terms outputs a signal, we can model this relationship as shown in Fig. 2 below.

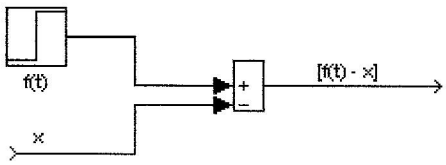

Fig. 2. Summing the input signals.

Now we notice that the bracketed term **[f(t) – x]** is multiplied by a constant **1/RC**. We do this in Simulink by passing the signal through a Gain block as in Fig. 3 below. Because all the terms on the right-side of xdot are accounted for, we know that the output signal must be equal to the left side of the equation, which is **xdot**.

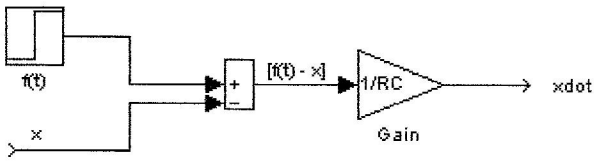

Fig. 3. Applying a gain to the output.

However, we are interested in **x,** not **xdot**. How can we take the **xdot** output signal and get an **x** output? The answer is to use an integrator block as in Fig. 4 below.

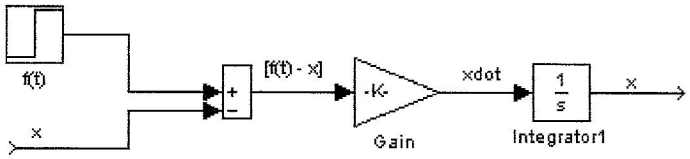

Fig. 4. Integrating the output signal.

Now, we have the desired **x** output, but we notice that **x** is also an input of the system. In our model above, the input **x** branch is a "dead" branch. In other words, there is no real signal going in there. How can we make **x** both an output and input of the system? The answer is to use a feedback loop by tapping the output **x** signal and feeding it back into the system at the input point. After some manipulation of the lines, your model should look like Fig. 5 below.

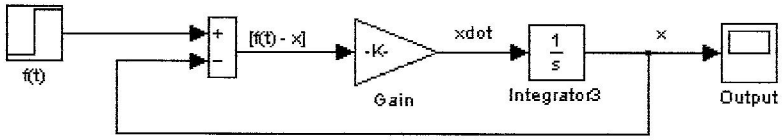

Fig. 5. The finished Simulink model.

After adding a Scope block, you are ready to set the block parameters, and run the simulation. But first, take the time to build the second-order model as described in the following section.

Modeling a Second Order Equation (Single Degree of Freedom System-SDOF)

The mass-spring-dashpot is a basic model used widely in mechanical engineering design to model real-world mechanical systems. It is represented schematically as shown in Fig. 6 below.

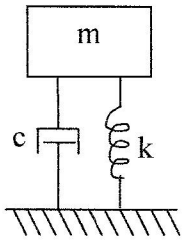

Fig. 6. The SDOF Mass-Spring-Dashpot.

The response of this system is governed by the equation of motion which is a second-order differential equation, and is shown in (2) below

$$\ddot{x} = \frac{1}{m}[f(t) - c\dot{x} - kx] \tag{2}$$

Where $\ddot{x}$ (**xddot**) is the acceleration of the mass **m**, $\dot{x}$ (**xdot**) is the velocity, **x** is the displacement, **f(t)** is the forcing function (input force), **c** is the damping coefficient, and **k** is the spring constant.

To model this system we start by looking at the terms in the bracket. There are three input signal lines: **f(t)**, **xdot**, and **x**. We also notice that both **xdot** and **x** are multiplied by constants. As in the case of the first-order model; this can be done in Simulink by using a Gain block. The signals are then summed together as in Fig. 7 below.

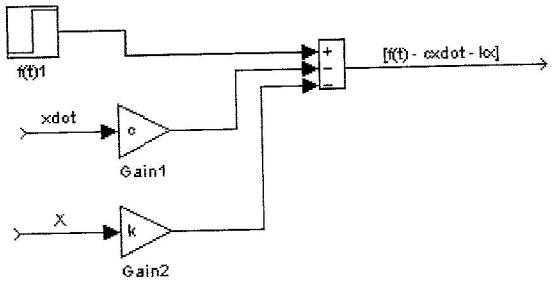

Fig. 7. Summing the input signals.

Now, we notice that the bracketed term **[f(t)- cxdot- kx]** is multiplied by the constant **1/m**. We do this in by passing the signal through a Gain block as in Fig. 8 below.

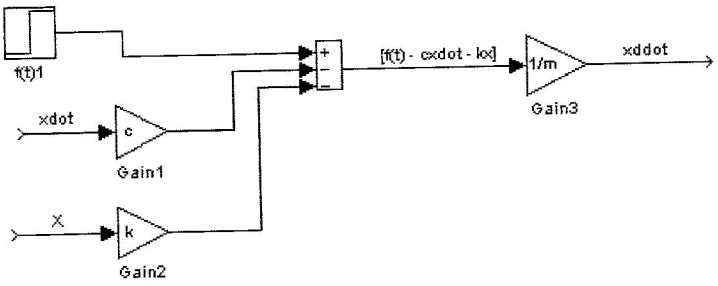

Fig. 8. Applying a gain to the output.

Again, we are interested in the displacement **x**, not **xddot**. To get the **x** output signal we integrate the **xddot** output signal twice, as in Fig. 9 below.

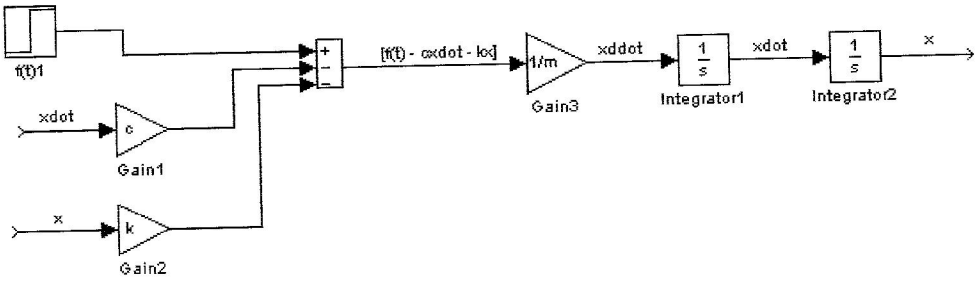

Fig. 9. Integrating the output signal twice.

We can see from the model that **xdot** and **x** are both outputs and inputs of the model. Therefore, we need to use a feedback loop for each of these signals as discussed in the First Order model example. The resulting model should look like Fig. 10 below.

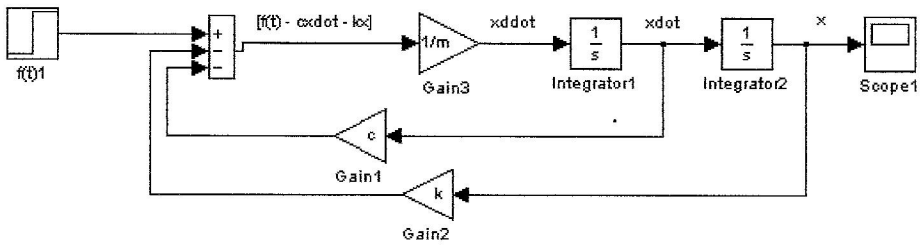

Fig. 10. The finished Simulink model for a SDOF system.

Before running the model be sure to include a ***Scope*** block for viewing the results. Now that the model is complete, we are ready to run the simulation and look at the results.

Running Simulation and Analyzing data

Having completed the Simulink Models for both the first and second order systems, it is now time to run a simple simulation and look at the results. We will start first with the first-order system, and then show the simulation and results for the second-order system.

First Order Differential Equation (RC Circuit)

Double-click on the ***Integrator*** block and set the initial conditions to zero as shown in Fig. 11 Leave the remaining fields as they are.

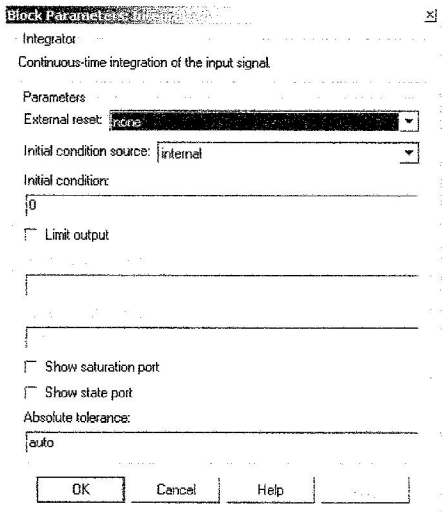

Fig. 11. Integrator block parameters

Now, double click on the ***Gain*** block and set the gain to the values shown in Fig. 12 below.

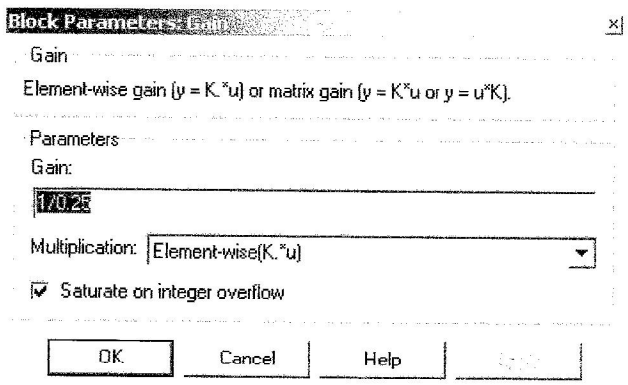

Fig. 12. Gain block parameters

Open the ***Step*** block to familiarize yourself with the parameters. Now, run the simulation and open the ***Scope*** block. Your results should be similar to Fig. 13 below.

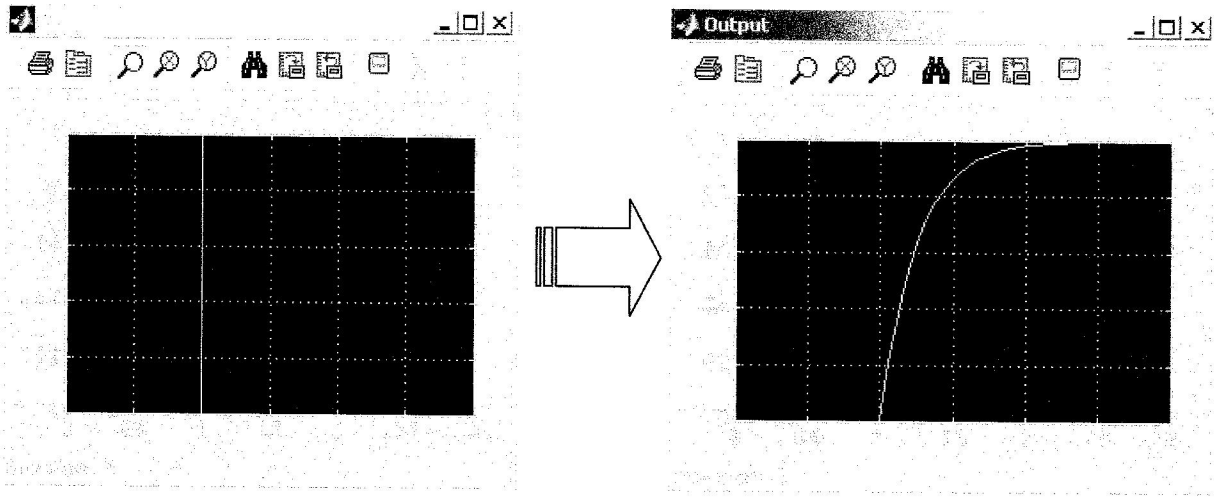

Fig. 13. Step input and resulting output from first-order system.

Notice how the response of the system lags the step input. This response is governed by the time constant of the system ($\tau = RC$). Other tutorials in this series will address the significance of this issue.

Second Order Differential Equation (Single Degree of Freedom System-SDOF)

For the SDOF model the initial conditions for both integrator blocks should be set to zero. As for the gain blocks, refer to Fig. 10 and use the following values $\mathbf{m = 2}$, $\mathbf{c = 2}$, and $\mathbf{k = 4}$. Now run the simulation and open the scope block. Your results should be similar to Fig. 14 below.

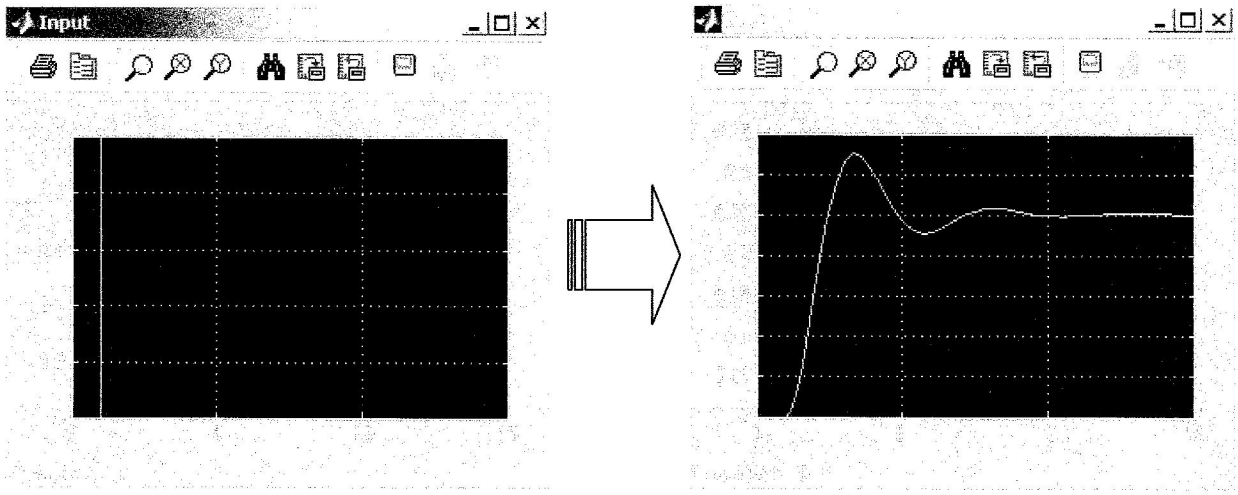

Fig. 14. Step input and resulting output from second-order system.

As you can see, things get a little more interesting for this case because the system is capable of oscillatory response, in fact, that can be seen in Fig. 14 above. The system response in this case is "under-damped." It is important to note however, that with different values of **m**, **c**, and **k** the response of the second-order system can be made to mimic a first-order response. When this is the case, the system is referred to as "critically-damped." Can you determine what values of **m**, **c**, and **k** will make this happen?

Finally, you are encouraged to play around with different values for both the first and second order systems to develop an understanding of how each of these systems will respond. An understanding of first and second order systems is critical for continued work in dynamic systems.