Lab 1: Dynamic Simulation Using Simulink and Matlab

Objectives

In this lab you will learn how to use a program called Simulink to simulate dynamic systems. Simulink runs under Matlab and uses block diagrams to represent dynamic systems. You can think of Simulink as a tool that allows programming in a graphical manner. Instead of large amounts of code, you can simply drag-and-drop pre-made blocks into a "model" window and connect the blocks' inputs and outputs to create a program or simulation.

Once a system is defined, you can run a simulation by choosing options from the Simulink menu. The progress of the simulation can be monitored while the simulation is running, and the final results can be made available in the Matlab workspace when the simulation is complete.

Background and Theory

Steps Required to Simulate a System

The steps required to simulate a system using Simulink are listed below.

1. Write the governing equation(s) of the system.
2. Draw the block diagram representing the system dynamics on paper.
3. Create the corresponding block diagram in Simulink including additional source and sink blocks for handling inputs and outputs.
4. Set up and run the simulation.
5. Display and analyze the results.

You completed steps 1 and 2 in the Prelab assignment. The details required for completing steps 3 through 5 will be explained in the following sections.

Structure of a Block Diagram in Simulink

The overall block diagram in Simulink will have the form shown in Figure 1. This figure only represents the general form a block diagram. There are many source, system, and sink blocks from which to choose.

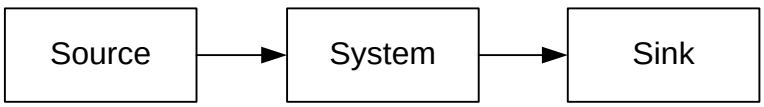

Figure 1: A typical setup of a Simulink model block diagram.

The block labeled *System* represents the dynamics of the system you want to simulate. Depending on the complexity of the system, this block will consist of one or more blocks selected from the Simulink library.

If the system you are simulating has a forced input, you will need to include a *Source* block to supply the desired input. The Simulink library contains several source blocks to generate a variety of forcing functions such as steps, ramps, and sinusoidal inputs.

The manner in which the output data is displayed or stored is specified using a *Sink* block. There are a variety of sink blocks to choose from in the Simulink library. For example, one sink block specifies that the data is to be stored on a disk, and another specifies that the system output is to be placed in the Matlab workspace.

Experimental Procedure

Exercise 1

Based on the block diagram you were asked to draw in Question 2 of the Prelab assignment, a corresponding block diagram can be drawn in Simulink. The block diagram that you are going to create in Simulink will have the following appearance:

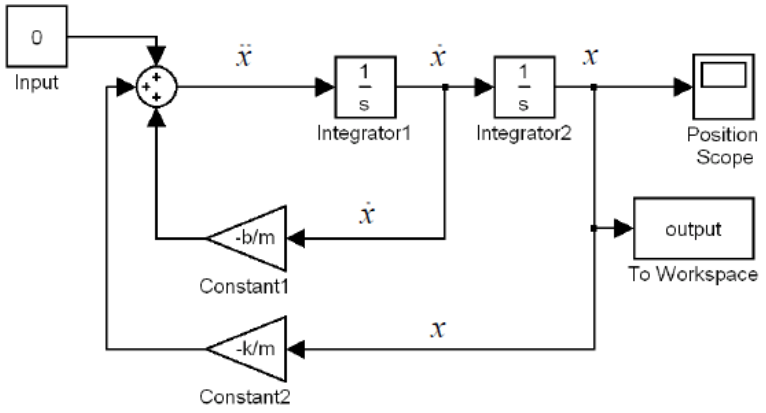

Figure 2: The completed block diagram with source, system, and sinks.

Note that the system is unforced and will not require a source block when creating the Simulink block diagram.

Invoke Simulink and Open New Window. To create the above block diagram in Simulink, invoke Simulink by opening Matlab and typing

>>simulink

This command displays a window titled Simulink which contains icons representing the block libraries. Now select *File* from the Simulink menu and choose *New* to open a new untitled window in which you can create the block diagram. **Note:** Simulink may open a new model window each time it is started.

Copy Blocks from Libraries. You are now ready to copy the individual blocks from the block libraries. Blocks are copied from the block libraries by opening a library and dragging the desired block into the new

window. Dragging a block is accomplished by placing the pointer on the desired block and then holding the left mouse button down as you move a copy of the block to the desired location.

Most of the blocks that are needed for this example are contained in the library titled *Linear*. To open this library, double-click on the icon using the mouse. This will display the contents of the library. Note that the library includes a summing junction, a gain block, and an integrator. Copy a summing junction, two integrators, and two gain blocks into the window, and arrange the blocks as shown in the block diagram above. The orientation of the gain blocks can be reversed by selecting the *Flip Block* option from the *Format* menu.

Only the blocks titled *Scope* and *To Workspace* remain to be copied into the new window at this point. These blocks are located in the library titled *Sinks*. Copy these blocks to the new window in the same manner as before. By including the *To Workspace* block, you are specifying that the output of the system be stored as a vector in the Matlab workspace after the simulation is complete. Including the *Scope* block allows the output to be viewed as the simulation is running.

Connect Blocks. Now you need to connect the blocks. The angle bracket (>) pointing out of a block represents the output port and the angle bracket pointing into a block represents the input port. First, connect the output of the summing junction to the input of the first integrator by dragging the output port of the summing junction to the input port of the integrator. A line with an arrowhead should now appear showing the connection. Connect the output of the first integrator to the input of the second integrator in the same manner.

Most of the remaining connections require that you connect line segments in order to route the connections around other blocks. As an example of how this is done, let us now connect the output of the second integrator to the input of the gain block having gain *-k/m*. To do this, click on the output port of the second integrator and drag it horizontally to create the first line segment and release the left mouse button. Now click on the end of this line segment and drag it vertically to create the second line segment and then release the left mouse button. To complete this process, connect the end of this line segment to the input of the *-k/m* gain block.

Note that the output of the first integrator must be split into two line segments. To do this, point to the line connecting the two integrators and click on the line using the right mouse button to split the line. Then drag the new line vertically and release the right mouse button to create the new line segment. Now you can proceed as before by connecting this output to the input of the *-c/m* gain block. At this point, you should be able to make all of the remaining connections on your own following the above procedures.

Set Block Parameters. After all of the connections are made, you will need to set the gains of the gain blocks to the proper values as well as set the initial conditions of the integrator. To set the gain of the *-c/m* gain block, double-click on the gain block to open the dialog box and enter a value of *-2.0*. In the same manner, set the gain of the *-k/m* gain block to *-40*. Setting the initial conditions of the integrators is done in a similar fashion. The output of the second integrator is position with the corresponding initial condition of 100mm. Open the dialog box of the second integrator and input *0.100*. In the same manner, set the initial condition for the first integrator equal to *zero* (the initial velocity).

The sink block labeled *To Workspace* must also be set up. Open the dialog box for the block labeled *To Workspace*, enter *position* for the variable name, and enter *2000* for the maximum number of rows (which is the maximum number of points that will be stored). Leave *Decimation* and *Sample Time* as default.

Set up Integration Routine. Now the integration routine parameters must be set. To do so, choose menu item *Simulation* and select *Parameters*. Choose *ODE45* in the solver options and set the *start time, stop time, relative* and *absolute tolerance* to *0, 10, 1e-3,* and *1e-6*, respectively. Here you have specified that the simulation will run from 0 to 10 seconds with the given tolerances.

Run Simulation and Plot Results. In order to view the output as the simulation is running, open the block titled *Scope* and place its window to the side of the current untitled window so that it can be seen as the simulation is running. To run the simulation, select menu item *Simulation* and choose *Start*. You should see the output in the Scope window as the result is generated. When the simulation is complete you will hear a beep. The results of the simulation are also available in the Matlab workspace. Go to the Matlab workspace and plot the time response by typing

>> plot(tout, position).

Save Block Diagram. You will want to save the block diagram since you will be asked to modify it in Exercise 2. To save the block diagram you have created, select menu item *File*, choose *Save* and then supply the requested filename. **Note:** Save the file in the directory C:\ME460.

Before proceeding to the following exercises, you should become familiar with the other types of blocks available in the block libraries. All of the blocks that you will need to complete this lab will be found in the block libraries titled *Sources, Sinks, Commonly Used Blocks,* and *Math Operations*. Browse through each of these libraries to become acquainted with the available blocks. The function of each block should be obvious from its title. However, if you are unsure about a block's function or its use, double-click on the block to open its dialog box and then select *Help* to get a detailed explanation of its use.

Find Inverse Laplace transform in Matlab by Using Impulse Response. To compare with the numerical integration results from Simulink, do an inverse Laplace Transform of the output response X(s) by using Matlab. In Matlab, there is a function called *impulse* which gives the impulse response (i.e. the output response for an impulse input $\delta(t)$) of a given transfer function with zero initial conditions). Since the Laplace Transform of an impulse is unity, the impulse response is simply the inverse Laplace transform of the transfer function. Therefore, if we treat X(s) as if it were a transfer function, the impulse response of "X(s)" would give us its Laplace inverse $x(t)$.

Type the following in Matlab as an M-file and run it several times to see how system response varies with damping constant. You don't need to type the comments.

```
>>m=1;                          %mass
>>k=40;                         %spring constant
>>c=2;                          % damping constant
>>x0=.1;                        % initial position
>>v0=0;                         % initial velocity
>>T=linspace(0,10,200);         %time vector from 0 to 10 sec., 200 points
```

```
>>num_x=[0,m*x0,c*x0+m*v0];      % numerator of the Laplace transform
>>den_x=[m,c,k];                 % denominator of the Laplace transform
>>x=impulse(num_x,den_x,T);      % inverse Laplace transfer function
                                 % Type "help impulse" to find how to use it
>>plot(T,x);                     %plot the time response x(t)
```

Compare the plots you get from Simulink and Matlab's *impulse* command.

Exercise 2

Using the block diagram you were asked to draw in Question 4 of the Prelab assignment, build a corresponding block diagram in Simulink. Note that the equation of motion for the pendulum with viscous damping is non-linear. However, this non-linearity can be easily handled in Simulink using the non-linear block called *Trigonometric Function* located in the *Math Operations* block library.

You should find that this block diagram is the same as the block diagram of the earlier problem with a few exceptions (the old gains of $-k/m$ and $-b/m$ are now $-g/l$ and $-b/ml^2$, respectively, and the $\sin(\theta)$ block has been inserted immediately before the $-g/l$ gain block) . Make the necessary modifications to the block diagram you used in Exercise 1, and set the gains and initial conditions to the correct values. Then run a simulation from 0 to 10 seconds, and plot and print your results.

Note that in order to make modifications in Simulink, any object (line or block) can be deleted by highlighting it and selecting menu item *Edit* and choosing *Cut*. Also, to change the position of a block, you can simply drag it from one location to another and its connections will remain intact. Finally, if you want to change the size of a block for readability, highlight the block and drag any corner until it is the desired size.

Now consider a linear approximation of $\sin(\theta)$ by θ. Modify your Simulink block diagram accordingly.

Run a dynamic simulation of this linear approximation with different initial conditions and compare to the nonlinear simulation. At what initial conditions does the linear approximation become poor? Plot and print the system response for that initial condition.

Exercise 3

For Exercise 3 you will develop linear and nonlinear Simulink models of a hydraulic motor system similar to the one you will be working with in future labs (see Figure 3). The system of equations which describe the motor system will be explained below. You will then need to linearize this set of equations and create Simulink model representations of both the linearized and nonlinear models.

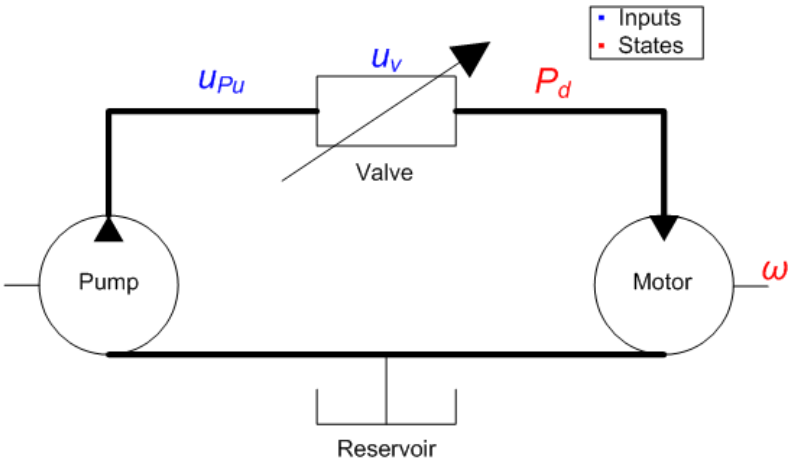

Figure 3: Hydraulic motor system with valve control.

For the system shown in Figure 3, there are two states: the motor speed and the pressure downstream of the valve. The two inputs are the valve command and the pressure upstream of the valve. The relationship between torque applied to the hydraulic motor and the speed of the motor can be captured well with a first order model. A differential equation for the motor speed is given below.

$$J \cdot \dot{\omega} = T - b \cdot \omega \tag{1}$$

Here J is the motor inertia, b is the friction coefficient, T is the torque applied to the motor, and ω is the angular velocity of the motor. The speed of the hydraulic motor and the flow rate of oil through the motor, q_m, are related by the displacement of the motor, D.

$$q_m = D \cdot \omega \tag{2}$$

Note that several simplifying assumptions have been made in this model. For example, leakage in the hydraulic motor has been ignored and it is assumed that there is no static friction in the motor. If leakage was included, the flow rate and motor speed would be related by a more complex function. These effects play a minor role in the physical system and are not necessary for deriving an accurate model of the motor, but it is important to always be aware of what assumptions you are making when modeling a physical system.

The torque applied to the motor is a function of the pressure drop across the motor, so one must first derive a differential equation for the pressure in the hose connecting the valve and the motor. This relationship is given by Eq. (3). Equation (4) relates the pressure in the hose to the torque applied to the motor.

$$\dot{P}_d = \frac{\beta}{V}\left(q_v - q_m\right) \tag{3}$$

$$T = D \cdot P_d \tag{4}$$

Here P_d is the pressure of the oil downstream of the valve, V is the volume of oil inside the hose connecting the valve and motor, β is the bulk modulus of the oil (this is a measure of the oil's resistance

to compression), and q_v is flow through the valve. Equation (4) relates the rate of change of pressure within the hose to the relative flow rate from the valve and through the hydraulic motor. When the flows from the valve and through the motor are equal, the pressure remains constant.

The last component of the system is the valve. The valve is the actuator with which one controls the motor speed for a given pressure upstream of the valve. Equation (5) relates the pressure drop across the valve and the valve command to the flow through the valve.

$$q_v = k \cdot u_v \sqrt{u_{P_u} - P_d} \qquad (5)$$

Here u_v is the valve command in volts, k is a coefficient relating the valve command to the valve opening, and u_{Pu} is the pressure upstream of the valve. This equation is known as the orifice equation and it can be derived from Bernoulli's equation[1].

Combine Eq. (1) through Eq. (5) into a system of 2 nonlinear dynamic equations and then linearize this system about the operating point: $u_{v,o}$, $u_{Pu,o}$, $P_{d,o}$, ω_o. Using the parameters given in Table 1, build a Simulink model of both versions (nonlinear and linearized) of the hydraulic motor model. Simulate both representations with an upstream pressure of 2.75 kPa and a step change in valve command from 0 to 2.5V.

Table 1: Hydraulic motor system parameters

Bulk Modulus (β)	240 MPa
Hose Volume (V)	30 cm^3
Displacement (D)	4 cm^3/rad
Friction Coefficient (b)	1 N.m.s
Inertia (J)	0.5 $kg.m^2$
k	2
$u_{v,o}$	2.5 V
u_{Pu}	2.75 MPa
$P_{d,o}$	0.5 MPa

Lab Report

1. Compare the plots you get from Simulink and Matlab's impulse command in Exercise 1. Are they similar? Do you expect them to be similar or not? Can you explain the differences, if any?

2. In Exercise 2, at which initial condition does the linear approximation become a poor one? In what ways do the approximations become poor?

3. How are the observations made in Question 2 (above) relevant to the design of controllers?

4. For Exercise 3, how did the linear and nonlinear models compare? How would their agreement change if they were simulated at a different upstream pressure?

Be sure to include your plots from all three exercises!

[1] White, F.M., Fluid Mechanics, 5th ed. McGraw-Hill, Boston: 2003.