MATLAB / Simulink Workshop
Modeling & Simulation of Dynamic Systems

Prof. Nolan Tsuchiya
Cal Poly Pomona
ntsuchiya@cpp.edu

Contents

This page is intentionally left blank.

1 Introduction

Simulink is a tool used for modeling, simulation, and analysis of dynamic systems. It uses graphical block diagramming to greatly simplify simulations. While most simulations can be hard-coded in an m-file using solvers such as ODE45, the Simulink package allows for a much faster and more intuitive way to run simulations due to its graphical layout. Hopefully, the benefits of using Simulink will become apparent as you progress through this lesson.

This module is divided into three main sections. First, we will discuss what modeling of dynamic systems really means and motivate the importance of being able to simulate these models using MATLAB. An introduction to the Simulink environment along with a couple of simple examples will carry you to the end of the first section. In the second section, we will use a concrete example to look at some more advanced features of Simulink and bridge the gap between MATLAB and Simulink to emphasize the integration between programs. Finally, we will model and simulate a more complicated dynamic system and animate the results. We will conclude by touching upon some tips and common issues that arise when using Simulink.

1.1 Dynamic Systems

What is a *dynamic system*? What does *modeling* refer to? What can Simulink actually *do* for me? These are some of the questions I hope to answer before we jump into some concrete examples.

Any object, contraption, device, or scheme, whether mechanical, electrical, biological, chemical, ecological, geographical, etc.. can be thought of as a dynamic system with inputs and outputs. A mass-spring system is a familiar dynamic system where the input may be an applied force, and the output may be the mass position. A low-pass filter circuit is a dynamic system, where the output voltage is the result of passing an input voltage through the electronics of the circuit. Animal population in the wild is a dynamic system, where numerous input influences can affect the output population. Global economics is a vastly complicated dynamic system with countless inputs and outputs. The idea is that virtually anything can be thought of as a dynamic system in one way or another.

The defining factor that makes a system *dynamic* is that the present output is dependent on present *and past* inputs.

1.2 Modeling

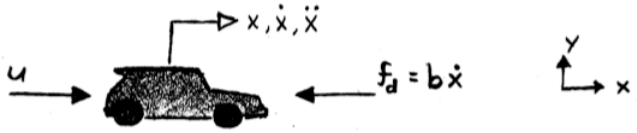
The process of modeling can be thought of as converting a real-life system to a mathematical relationship using our knowledge of the physical world. For example, Newton's Laws help us to model mechanical systems, while in electrical circuits, Kirchoff's current and voltage laws combined with Ohm's law allow us to mathematically describe (model) a circuit. For our purposes here, we will define a model as a mathematical relationship describing how one parameter (the input) affects another (the output). Because the definition of a dynamic system in the previous section is so general, it is wise to focus our attention on an example in order to discuss the modeling process.

1.2.1 Example - Modeling an Automobile

Imagine getting into your car and driving off from rest. Suppose you depress the gas pedal all the way to the floor (you are in a hurry). This is known as a *step input*. Does the car instantly reach its top speed and stay there? Without too far a stretch, we can probably guess *no*. Rather, our intuition tells us that

the car will gradually accelerate and eventually the wind resistance will match that of the engine's power and the car's velocity will stabilize at some terminal velocity. The *way* by which the car reaches some target velocity is inherent in the system's *dynamics*.

To model such a system, let's consider a free-body diagram of the vehicle:



It is clear from the diagram that, in addition to the driving force, u, provided by the engine, we are also considering wind resistance by adding a drag force, $f_d = b\dot{x}$, proportional to the car's velocity. Applying Newton's law in the horizontal direction, we have:

$$\sum F_x = ma_x$$
$$u - f_d = m\ddot{x}$$
$$u - b\dot{x} = m\ddot{x}$$

which is a model for the input force, u, to the car's position, x. However, for this example, we are interested in the car's velocity as the output rather than the car's position. Substituting $\dot{x} = v$ and $\ddot{x} = \dot{v}$ yields a model relating the input force, u, to the velocity, v:

$$u - b\dot{x} = m\ddot{x}$$
$$u - bv = m\dot{v}$$

Or in standard form:

$$\boxed{\dot{v} = -\frac{b}{m}v + \frac{1}{m}u} \tag{1}$$

The previous example illustrates how we have choices when deciding how to model a system. Had we not included the drag force, this would have greatly simplified the model, but the accuracy of the model would have suffered equally. In fact, without the drag force, the model could be a different model altogether - perhaps a rudimentary spacecraft thruster model!

As the complexity of the model increases and the engineer accounts for more variables (friction, heat, drag, aerodynamics, etc..) the model accuracy increases. While we can never exactly achieve it, a perfect model would precisely represent the system. Consequently, simulating the model would tell the design engineer exactly how the system would behave in real life. For this reason, the benefits of an accurate simulation cannot be understated. Consider a multi-million dollar space mission, where the spacecraft must be designed to withstand extreme and uncertain conditions. Having the ability to simulate various scenarios on a computer before the actual mission occurs can (and does) greatly improve success rates for these types of missions as the engineers can see the results of certain events and adjust for them before the actual spacecraft is launched.

In any case, before we can actually simulate our vehicle velocity model in Simulink, we need to learn the basics of the program.

1.3 Simulink Basics

To open Simulink, simply type 'simulink' in the MATLAB command window and press Enter. The Library Browser should open.

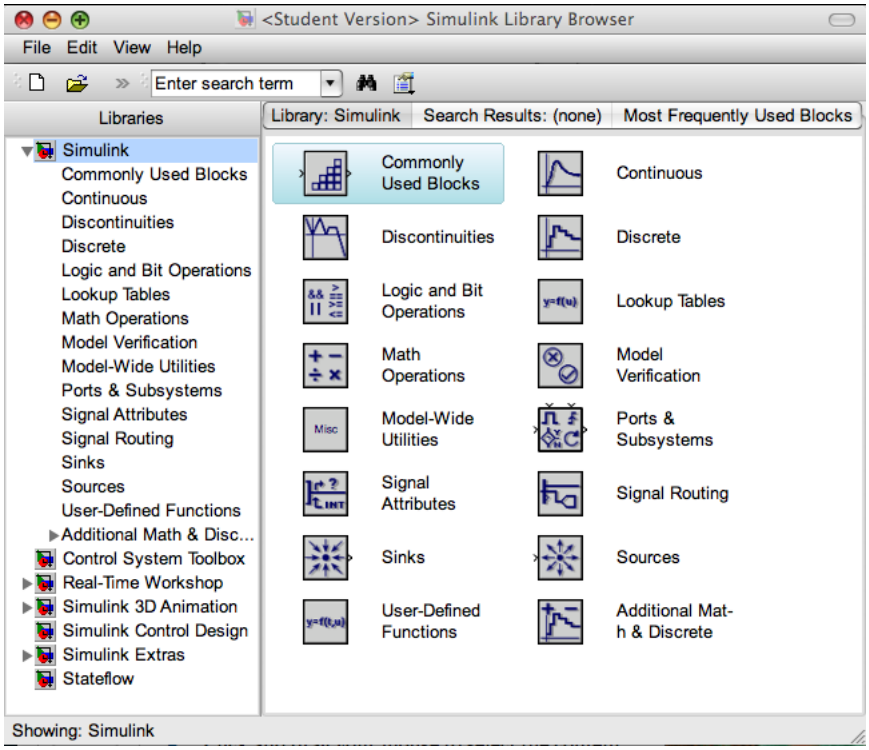


Figure 1: Simulink Library Browser

The library browser in Figure 1 is where the all of the Simulink blocks are contained. These blocks are used to construct the block diagrams that will represent your models.

To begin a new model, go to File > New > Model. You will see a blank white window open. This is the space into which you can drag the blocks from the library and construct your model. Saving a model in Simulink generates an .slx file.

1.3.1 Example - Sine Wave

From the library browser, drag a **Sine Wave** block from the 'Sources' menu onto your new model. Then, drag a **Scope** block from the 'Sinks' menu to the model. Connect the blocks by dragging a line from one block to the other. Alternatively, you can select the **Sine Wave** block, hold Ctrl, then click the **Scope** block.

Double click the **Sine Wave** block and change the amplitude to 3. Double click the **Scope** block to view the scope axes. To begin the simulation, go to Simulation > Start in the model window.

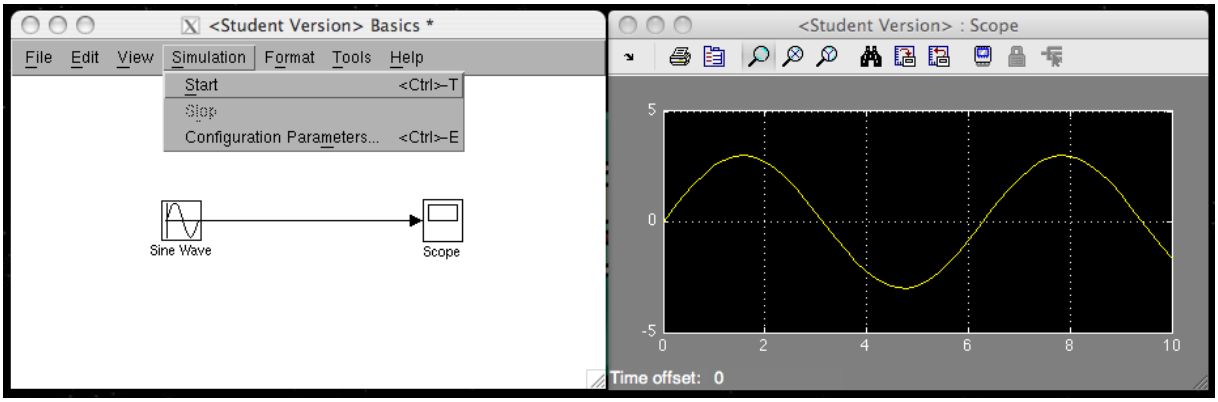


Figure 2: Sine Wave Model

The result should look like the scope in Figure 2.

1.3.2 Example - Interference

Build the following block diagram shown in Figure 3. In addition to the block used in the previous example, you will need an **Add** block found in the 'Math Operations' menu of the library browser. Create two sine waves with the same amplitude and frequency and add them together. What is the result? Then add two sine waves with the same amplitude but phases that differ by π radians. What is the result? Is this what you expect?

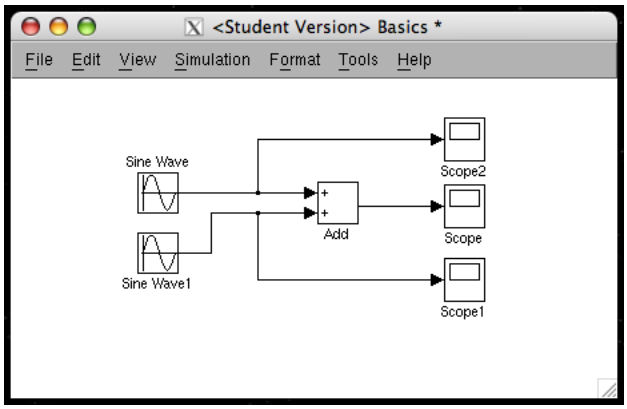


Figure 3: Two Sine Waves

1.3.3 Example - Integration

A very useful element in Simulink is the **Integrator** block. It is this numerical integration tool that allows for such easy simulation of the ODEs that represent a vast majority of the system models we deal with. Create the block diagram shown in Figure 4 below. Integrate a constant and verify it's function by using the **Step** input block found in the 'Sources' menu and the **Integrator** block found in the 'Continuous' menu. The **Step** block simulates a constant value for $t \geq t_0$ and is 0 for $t < t_0$. The step time, t_0, can be specified by double clicking the **Step** block. Does the integrator seem to do it's job?

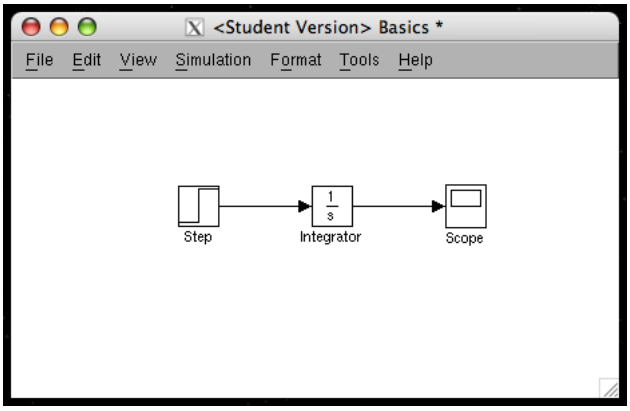


Figure 4: The Integrator Block

2 Integration Between Simulink and MATLAB

Thus far, you have used Simulink as a stand-alone program and you saw the results of your block diagrams via the **Scope** block. However, the integration between MATLAB and Simulink is quite seamless and intuitive and ought to be mastered by any engineer who uses Simulink.

In the previous section we derived an ODE (model) for a simplistic vehicle velocity system. By the end of this section, you will simulate this system and plot various results entirely from MATLAB, using Simulink only to build the initial block diagram.

2.1 The 'To Workspace' Block

Using the **Scope** block is convenient for quickly viewing a signal, however, the data is not saved for future use. The **To Workspace** block addresses this issue and any signals that are sent to it get passed to the MATLAB Workspace where they can be saved and later analyzed.

2.1.1 Example - Sending a Signal to the MATLAB Workspace

Try replacing the **Scope** block from the integrator block diagram with a **To Workspace** block as shown below.

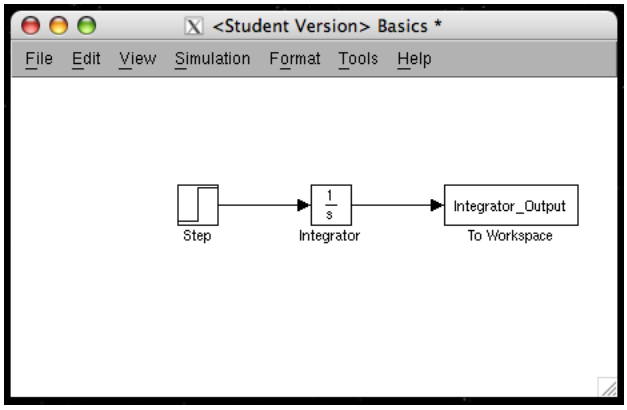


Figure 5: Sending a Signal to the Workspace

For ease of analysis and plotting, be sure to set the 'Save Format' to 'Array' by double clicking the **To Workspace** block and selecting from the drop down menu show below. Note that you can also rename the output variable here by typing in your desired name in the 'Variable Name:' text field.

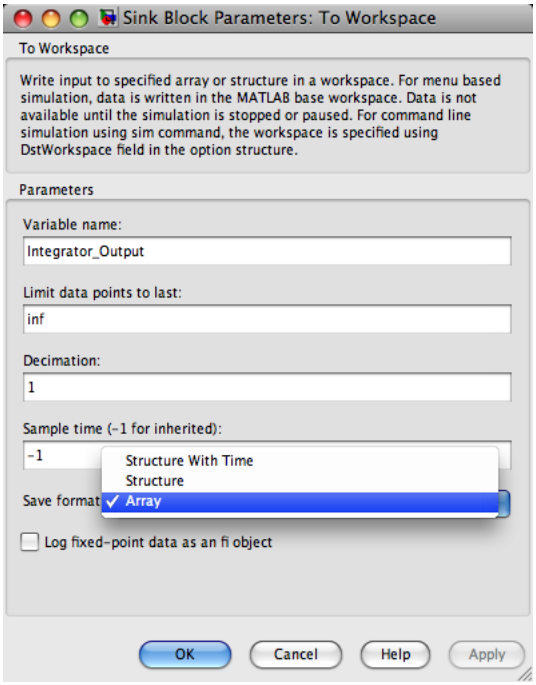


Figure 6: Specifying the Save Format

When sending signals to the Workspace, it is wise to also send a *time vector* to the Workspace. This becomes a necessity when you wish to plot certain data sets vs. time. Add a **Clock** to this block diagram and send it directly to the Workspace by adding another **To Workspace** block and renaming it appropriately.

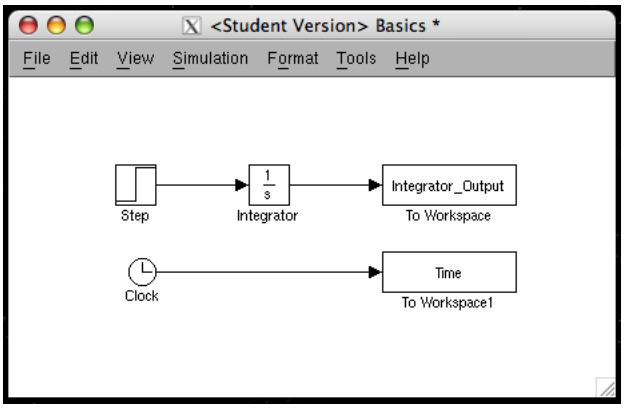


Figure 7: Adding a Clock to Generate a Time Vector

In the MATLAB command window, type 'clear' to erase all variables currently in your Workspace. Next, run this simulation from Simulink. **Note where the data from the simulation went. It should show up as two arrays in your Workspace.** Then, use what you have learned about plotting to generate the following plot of **Integrator Output vs. Time**

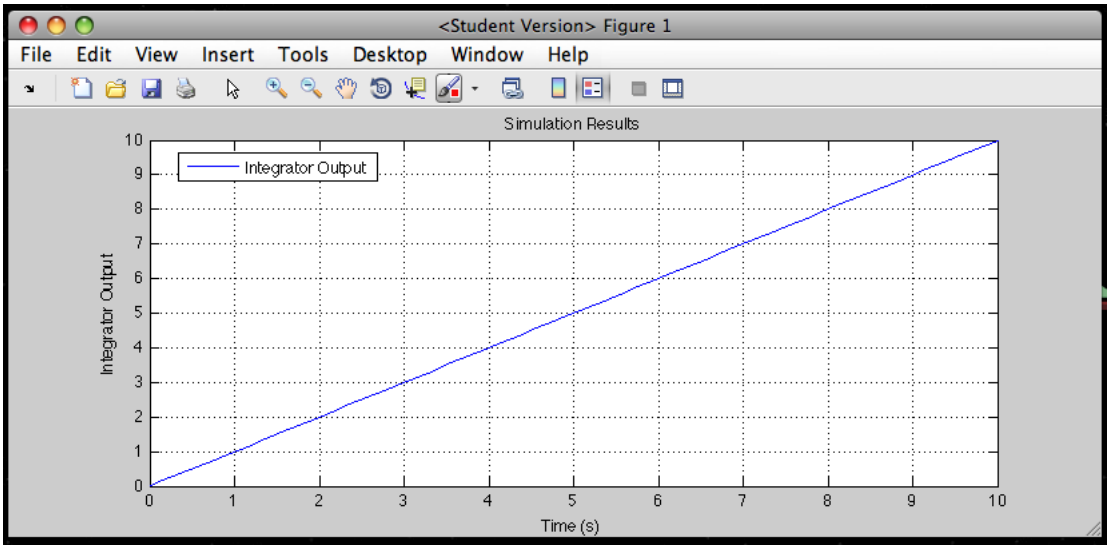


Figure 8: Plotting the Simulation Results

2.2 Referencing Data From The Workspace

You just learned how to send data from a simulation to the MATLAB Workspace for future analysis. However, Simulink can also grab data from the MATLAB Workspace and use it in a model. This is helpful when you need to specify many constants and be able to adjust them quickly.

2.2.1 Example - Calling Data from the MATLAB Workspace

Again, clear the Workspace of all variables. Then build the following block diagram. The new block here is the **Gain** block. Its purpose is to multiply the incoming signal by its *gain*, or constant multiplier. Set the gain to 'a' and run the simulation. What happens?

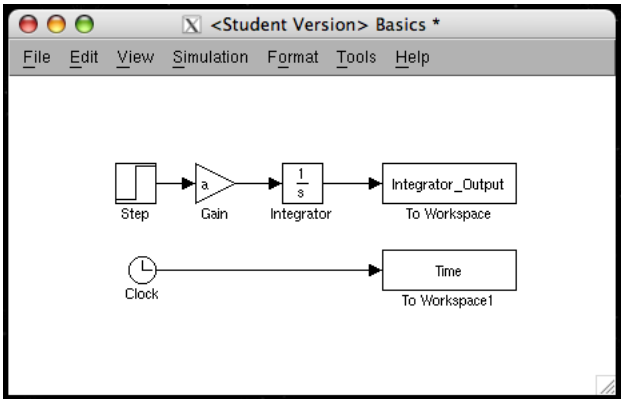


Figure 9: Adding a Gain Block

Now, in the MATLAB Command Window, type '$a = 3$.' Then run the simulation again. If you have done it correctly, the simulation should run without trouble.

This should illustrate how Simulink can call variables from the Workspace while running a simulation.

2.3 Simulating From MATLAB

Finally, you can simulate a model directly from MATLAB to avoid switching back and forth between programs. This is especially helpful because it allows you to define parameters, initialize vectors, run the simulation, and plot the data all from running a single m-file.

2.3.1 Example - Simulating from MATLAB

In the command window, type 'sim(*model name*)' and hit Enter. Note: the argument in the 'sim' command is the name of your .mdl file, typed *without* the '.mdl.' Your simulation should run as normal, but you didn't need to actually interact with the Simulink model.

3 Automobile Velocity

You now have all of the tools necessary to model an automobile's dynamics, simulate the system, and analyze the data you collect.

3.1 Activity

For this problem you will first build and save a block diagram that represents the automobile's dynamics as defined in equation (1). You will simulate the case where the car begins at rest and is then subject to a constant force of 10,000N at time $t = 1$. The parameters, b and m should be left in symbolic terms. You will then need to write and execute a single m-file that simulates the model. The m-file should:

- Clear the Workspace of all variables.
- Define the necessary variables. Begin with $m = 850kg$ and $b = 95kg/s$.
- Run the simulation for 180 seconds.
- Plot the vehicle's velocity vs. time, including appropriately labeled axes, a legend, and a title.
- Plot the vehicle's acceleration vs. time in a separate figure - Hint: You will need to add something to your model so that you can send an acceleration vector to your workspace.

Hints for creating your block diagram:

- Begin with a summing junction (**Add**) block. View the output of the block as the left side of equation (1) and view the input(s) to the block as the terms on the right side of equation (1).
- You can flip and rotate blocks by right clicking them and exploring the 'format' menu.
- The input should be a constant of 10,000N, beginning at time $t = 1$. Hint: recall the **Step** block.
- In order to run the simulation for 180 seconds, you can go to Simulation > Configuration Parameters > Solver, and set the 'Stop Time' to 180.

Question: What is the vehicle's terminal velocity?

3.1.1 Dynamic Twist - A Hill

Suppose that, after driving on flat roads for a while, the vehicle then encounters an incline such that the ground angle (from horizontal) is represented by the following profile:

$$\theta(t) = \begin{cases} 0 & , t < 60s \\ \pi/6 & , t \geq 60s \end{cases}$$

Your tasks are to:

- Re-write the dynamic equation to include this additional parameter.
- Modify your block diagram to include this parameter.
- Simulate the system for 180s using $g = 9.8m/s^2$ and the same m, b, and $u(t)$ from above.

Question: What is the vehicle's terminal velocity while traveling up the hill?

4 The Pendulum

We will now tackle a more complex dynamic system - the pendulum. However, before we begin, there is one final piece of information that you need in order to be successful.

4.1 Specifying Integrator Initial Conditions

When you double click an **Integrator** block, you will see a drop down menu called *'Initial Condition Source'*. You can select whether you would like to define the initial conditions from within the block (*'internal'*), or form an external source, such as the MATLAB Workspace (*'external'*).

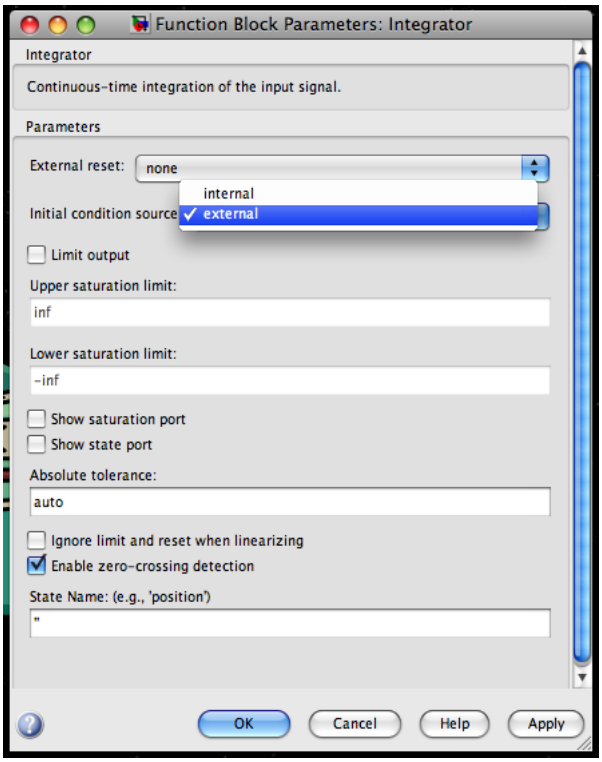


Figure 10: Specifying Initial Conditions

In the previous examples, you used the integrator default settings of *'internal'* and zero initial conditions. For the following pendulum example, we will want to play with various initial conditions specified from the Workspace.

When you select an *'external'* source, you will notice that a second input port shows up on the integrator block. To specify your initial conditions from the Workspace, simply connect a **Constant** block to this port and give it a symbolic value that you can define in your Workspace such as x_0 shown in the Figure below.

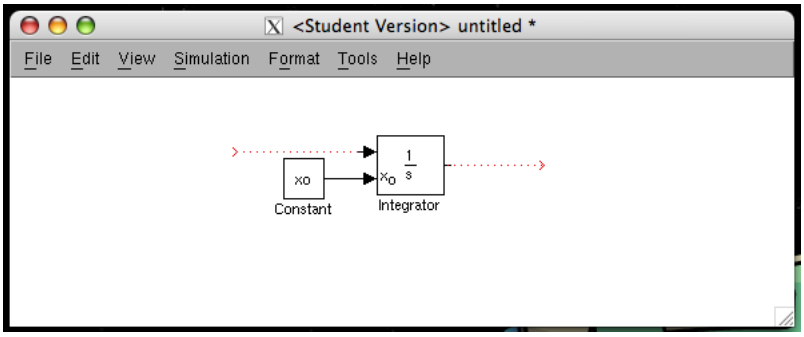
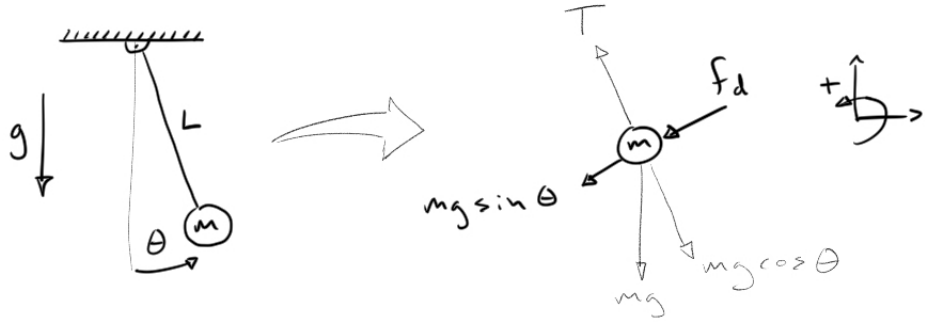


Figure 11: External Initial Condition Source

4.2 Modeling a Swinging Pendulum

Consider the free body diagram below:



In this example, we are modeling a drag force, f_d, proportional to the mass' velocity, $f_d = bv = bL\dot{\theta}$ that always opposes the motion of the pendulum. Additionally, gravity (g), is taken into account. The string of length L is modeled as massless and rigid, and the mass, m, is assumed to be a point mass. Applying Newton's law for rotation ($\sum \tau = I\alpha$) yields the ODE that describes this dynamic system:

$$\boxed{\ddot{\theta} = -\frac{b}{m}\dot{\theta} - \frac{g}{L}\sin(\theta)} \tag{2}$$

4.3 Activity

Your task is to generate a block diagram for this system and write an m-file that does the following:

- Clear the Workspace of all variables.
- Define the necessary variables. Begin with $m = 6$, $r = 1$, $b = 5$, and $g = 9.8$.
- Define the (2) initial conditions that represent initial angle, and initial angular velocity. Begin with $\theta_0 = \frac{\pi}{4}$ and $\dot{\theta}_0 = 0$ to make sure your simulation makes sense.
- Run the simulation for 20 seconds.
- Plot the mass' angle vs. time, including appropriately labeled axes, a legend, and a title.
- Plot the mass' angular velocity vs. time in a separate figure.

Question: With the initial condition for angular displacement, $\theta_0 = \frac{\pi}{4}$, find an appropriate initial positive angular velocity ($\dot{\theta}_0 > 0$) such that the pendulum does one full "loop" then settles at a final value of $\theta_f = 2\pi$.

4.4 Animating the Results

Once you have completed the tasks above, you may use the following to animate your model and see the pendulum in action!

You can *pick-off* a line from your θ signal and feed it into the block diagram given below. This block diagram uses simple trigonometry to generate the $x-$ and $y-$ position of the mass based on the angle and given radius. Thus, it sends two additional signals to the Workspace.

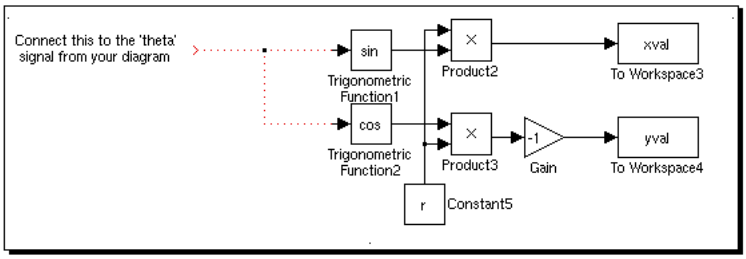


Figure 12: Block Diagram to Generate x-y Position

Then, the following code takes those two signals and animates them by plotting the x and y coordinates sequentially. You may simply copy this code into your m-file. You may need to modify it slightly to make it integrate into your existing code.

```
% Animation code

ax_scale = r*1.1;  % Specify axis limits for later plotting
tic          % Start a timer
dt = .01;    % Specify a fixed sample time
figure()     % Open a figure
% set(gcf,'PaperUnits','centimeters','position', [125 350, 700, 700])

for i = 1:length(xval)     % For each data point in the simulation, do...

t_loopstart=tic();          % Start a timer for the loop
    plot(xval(i), yval(i),'b.', 'markersize',45);     hold on;

    x1 = [0 xval(i)];  y1 = [0 yval(i)];  line(x1,y1);

    xx = [-.25 .25];    yy = [0 0];
    line(xx,yy,'linewidth',5,'color', 'k');             hold off;
    axis equal;
    axis([-ax_scale, ax_scale, -ax_scale, ax_scale]);

    xlabel('x (m)');     ylabel('y (m)')
    el_time=toc(t_loopstart);   % record elapsed time for loop
    pause(dt-el_time);          % pause for constant sample time
```

```
end
toc;          % end timer
```

If you've implemented everything properly, you should see an animation come up when you run your m-file. The following is a screenshot of the pendulum in action:

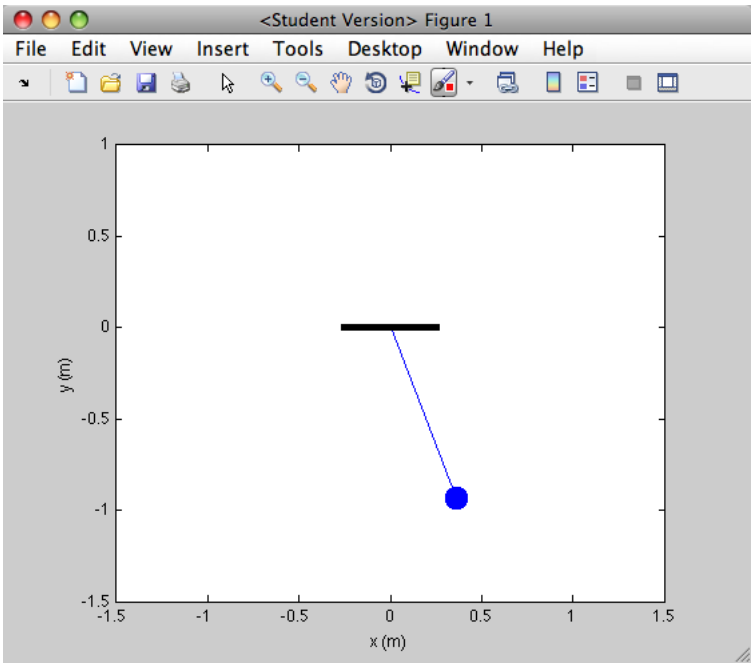


Figure 13: Screenshot of Pendulum Animation

5 The Double Pendulum and Chaotic Motion

In mathematics, the field of study called *chaos theory* deals with the behavior of dynamic systems that are highly sensitive to initial conditions. Often referred to as the "butterfly effect," chaotic systems have vastly different outcomes due to tiny differences in initial conditions. The result is that predicting long-term outcomes of the system is nearly impossible, despite the system being deterministic in nature.

5.1 Equations of Motion

One common example of a chaotic system is the double pendulum system:

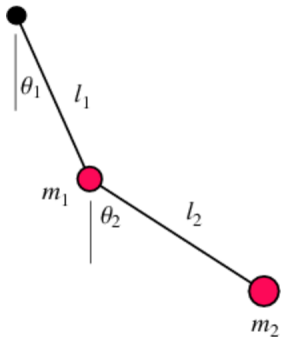


Figure 14: Double pendulum

In this formulation, the masses are treated as point-masses and the rods are considered massless, rigid wires. The derivation of the equations of motion requires a bit more complex math than a simple application of Newtons law; they are based on the Euler-Lagrange formulation (details can be found here: http://scienceworld.wolfram.com/physics/DoublePendulum.html).

Nonetheless, we still get two differential equations of motion that describe the system:

$$\ddot{\theta}_1 = -\frac{m_2 L_2 \ddot{\theta}_2 \cos(\theta_1 - \theta_2)}{(m_1 + m_2) L_1} - \frac{m_2 L_2 \dot{\theta_2}^2 \sin(\theta_1 - \theta_2)}{(m_1 + m_2) L_1} - \frac{g \sin(\theta_1)}{L_1} \tag{3}$$

$$\ddot{\theta}_2 = -\frac{L_1 \ddot{\theta}_1 \cos(\theta_1 - \theta_2)}{L_2} + \frac{L_1 \dot{\theta_1}^2 \sin(\theta_1 - \theta_2)}{L_2} - \frac{g \sin(\theta_2)}{L_2} \tag{4}$$

5.2 Activity

Your task is to generate a block diagram for this system and write an m-file that does the following:

- Clear the Workspace of all variables.
- Define the necessary variables. Begin with $m_1 = m_2 = 2$, $L_1 = L_2 = 0.5$, and $g = 9.8$ (this model assumes no drag).
- Define the (4) initial conditions that represent initial angle, and initial angular velocity for each link. Begin with $\theta_1(0) = \frac{\pi}{2}$, $\theta_2(0) = \pi$, and $\dot{\theta_1}(0) = \dot{\theta_2}(0) = 0$ to make sure your simulation makes sense.
- Run the simulation for 20 seconds.

- Plot the mass angle vs. time for m_1 and m_2, including appropriately labeled axes, a legend, and a title.
- Animate your results using the same technique as the previous single pendulum example.

5.2.1 Challenge Problem - Demonstrate Chaotic Motion

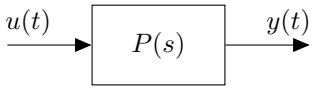
Run three (3) simultaneous double pendulum simulations with initial conditions that differ by a tiny amount ($\approx 2\%$). Animate the results simultaneously to prove the chaotic nature of the double pendulum system.

5.3 A note on SimScape / SimMechanics

The Mathworks has developed a dynamic simulation blockset within Simulink called SimScape. Within this package are several specific blocks used to simulate mechanical, electrical, automotive, etc... systems. SimMechanics is one of these specialized blocksets, and it allows for block diagram creation without the need for the governing ODEs. You can build mechanical systems using the predefined elements within the blocks, such as masses, springs, dampers, joints, supports, etc... The simulation results are identical to building up the block diagram from the ODEs (as we have done in this workshop), but it is much easier to implement. Still, having the knowledge to implement a simulation in the "traditional" way via the ODEs is valuable in that you are required to implement a fundamental understanding of how to simulate an ODE, and hence a dynamic system.

6 Electrical Circuits and the Frequency Response

The frequency response of a system is a vital concept when it comes to understanding and characterizing dynamic systems. Consider the dynamic system $P(s)$:



In simple terms if the system $P(s)$ is linear and time-invariant (LTI), then a sinusoidal input

$$u(t) = \sin(\omega t)$$

will result in a sinusoidal output of the same frequency ω, but it may be scaled by a constant and / or phase-shifted, i.e.

$$y(t) = A\sin(\omega t + \phi)$$

The scaling factor A is referred to as the *magnitude*, or *gain*; it is the amount by which the input has been scaled in the output. The angle ϕ is referred to as the *phase-shift*; it is the amount that the output sinusoid is phase-shifted with respect to the input. A key point is that both of these parameters are functions of the frequency, so it is more appropriate to label them:

$$\text{Magnitude: } A(\omega)$$
$$\text{Phase: } \phi(\omega)$$

You can then plot the magnitude and phase as functions of ω and see how A and ω change as the frequency of the input sine wave increases; **Key point: This pair of plots is called the Bode plot**

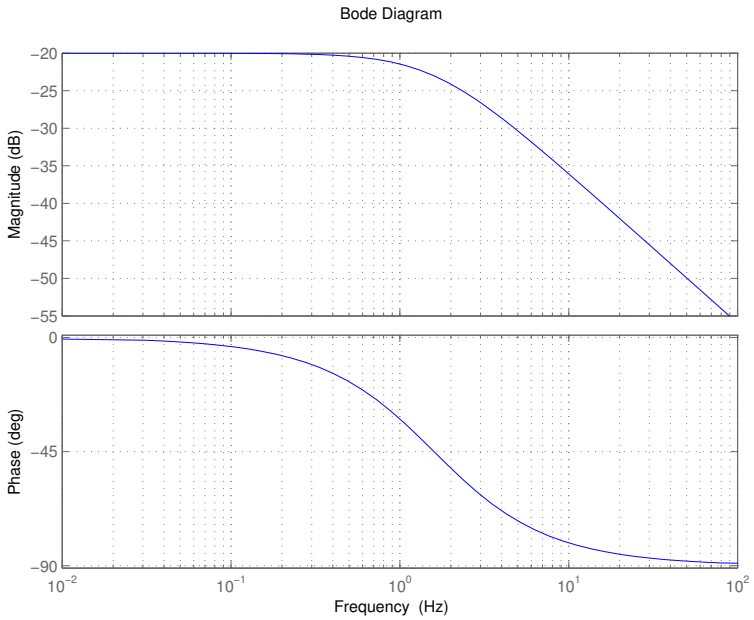
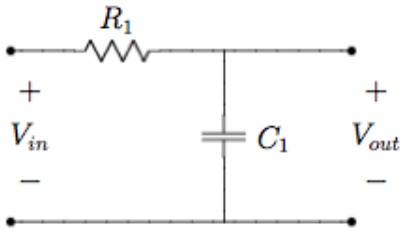


Figure 15: Bode plots (magnitude and phase) for a typical low-pass filter

6.1 Electrical Systems - Modeling a Low-Pass Filter

While the principles of frequency response are critical in mechanical systems, it tends to be easy[1] to think about frequency response in terms of electrical systems because it is easy to apply sinusoidal (AC) signals. One basic electrical filter is shown below:



6.2 Activity

- Model the Low-Pass Filter (LPF) circuit above by deriving an ODE that relates V_{in} to V_{out} (Hint: Follow the current through the circuit and use Ohm's Law as well as Kirchoff's Voltage and Current Laws. Recall for a capacitor $i = C\frac{dv}{dt}$).
  $\left(\textit{ans. } \dot{V}_{out} = -\frac{1}{R_1 C_1} V_{out} + \frac{1}{R_1 C_1} V_{in} \right)$
- Generate a block diagram that represents the ODE above with $R_1 = 1$ and $C1 = 1$.
- Simulate for $V_{in} = \sin(\omega t)$, where $\omega = 0.2$ and $\omega = 200$.

Question: Why is this called a *Low-Pass* Filter?