C.6 Differential equations

The purpose of this lab is to experiment with models of continuous-time systems that are described as differential equations. The exercises aim to solidify state-space concepts while giving some experience with software that models continuous-time systems.

The lab uses Simulink, a companion to Matlab. The lab is self contained, in the sense that no additional documentation for Simulink is needed. Instead, we rely on the on-line help facilities. Be warned, however, that these are not as good for Simulink as for Matlab. The lab exercise will guide you, trying to steer clear of the more confusing parts of Simulink.

Simulink is a block-diagram modeling environment. As such, it has a more declarative flavor than Matlab, which is imperative. You do not specify exactly how signals are computed in Simulink. You simply connect together blocks that represent systems. These blocks declare a relationship between the input signal and the output signal.

Simulink excels at modeling continuous-time systems. Of course, continuous-time systems are not directly realizable on a computer, so Simulink must **simulate** the system. There is quite a bit of sophistication in how this is done. The fact that you do not specify how it is done underscores the observation that Simulink has a declarative flavor.

The simulation is carried out by a **solver**, which examines the block diagram you have specified and constructs an execution that simulates its behavior. As you read the documentation and interact with the software, you will see various references to the solver. In fact, Simulink provides a variety of solvers, and many of these have parameters you can control. Indeed, simulation of continuous-time systems is generally inexact, and some solvers work better on some models than others. The models that we will construct work well with the default solver, so we need not be concerned with this (considerable) complication.

Simulink can also model discrete-time systems, and (a bit clumsily) mixed discrete and continuous-time systems. We will emphasize the continuous-time modeling because this cannot be done (conveniently) in Matlab, and it is really the strong suit of Simulink.

C.6.1 Background

To run Simulink, start Matlab and type `simulink` at the command prompt. This will open the Simulink library browser. To explore Simulink demos, at the Matlab command prompt, type `demo`, and then find the Simulink item in the list that appears. To get an orientation about Simulink, open the help desk (using the Help menu), and find Simulink. Much of what is in the help desk will not be very useful to you. Find a section with a title "Building a Simple Model" or something similar and read that.

We will build models in state-space form, as in chapter 5, and as in the previous lab, but in continuous time. A continuous-time state-space model for a linear system has the form (see section 5.7)

$$\dot{z}(t) = Az(t) + bv(t) \tag{C.1}$$

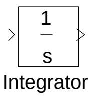
$$w(t) = cz(t) + dv(t) \tag{C.2}$$

where

- z: *Reals* $\to$ *Reals*^{N} gives the state response;
- $\dot{z}(t)$ is the derivative of z evaluated at $t \in$ *Reals*;
- v: *Reals* $\to$ *Reals* is the input signal; and
- w: *Reals* $\to$ *Reals* is the output signal.

The input and output are scalars, so the models are SISO , but the state is a vector of dimension N, which in general can be larger than one. The derivative of a vector z is simply the vector consisting of the derivative of each element of the vector.

The principle that we will follow in modeling such a system is to use an Integrator block, which looks like this in Simulink:



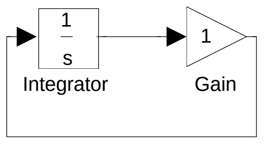
This block can be found in the library browser under “Simulink” and “Continuous.” Create a new model by clicking on the blank-document icon at the upper left of the library browser, and drag an integrator into it. You should see the same icon as above.

If the input to the integrator is $\dot{z}$, then the output is z (just think about what happens when you integrate a derivative). Thus, the pattern we will follow is to provide as the input to this block a signal $\dot{z}$.

We begin with a one-dimensional system ($N = 1$) in order to get familiar with Simulink. Consider the scalar differential equation

$$\dot{z}(t) = az(t) \tag{C.3}$$

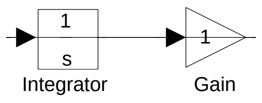
where $a \in$ *Reals* is a given scalar and z: *Reals* $\to$ *Reals* and $z(0)$ is some given initial state. We will set things up so that the input to the integrator is $\dot{z}$ and the output is z. To provide the input, however, we need the output, since $\dot{z}(t) = az(t)$. So we need to construct a feedback system that looks like this:



This model seems self-referential, and in fact it is, just as is (C.3).

Construct the above model. You can find the triangular “`Gain`” block in the library browser under “Simulink” and “Math.” To connect the blocks, simply place the cursor on an output port and click and drag to an input port.

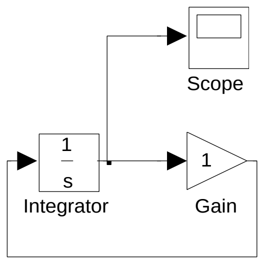
After constructing the feedback arc, you will likely see the following:



This is simply because Simulink is not very smart about routing your wires. You can stretch the feedback wire by clicking on it and dragging downwards so that it does not go over top of the blocks.

This model, of course, has no inputs, no initial state, and no outputs, so will not be very interesting to run it. You can set the initial state by double clicking on the integrator and filling in a value under "initial condition." Set the initial state to 1. Why is the initial state a property of the integrator? Because its output at time t is the state at time t. The "initial condition" parameter gives the output of the integrator when the model starts executing. Just like the feedback compositions of state machines in chapter 4, we need at least one block in the feedback loop whose output can be determined without knowing its input.

You will want to observe the output. To do this, find a block called "Scope" under "Simulink" and "Sinks" in the library browser, and drag it into your design. Connect it so that it displays the output of the integrator, as follows:



To make the connection, you need to hold the Control key while dragging from the output port of the integrator to the input port of the Scope. We are done with the basic construction of the model. Now we can experiment with it.

C.6.2 In-lab section

1. Set the gain of the gain block by double clicking on the triangular icon. Set it to -0.9. What value of a does this give you in the equation (C.3)?

2. Run the model for 10 time units (the default). To run the model, choose "Start" under the "Simulation" menu of the model window. To control the number of time units for the simulation, choose "Parameters" under the "Simulation" menu. To examine the result, double click on the Scope icon. Clicking on the binoculars icon in the scope window will result in a better display of the result.

3. Write down analytically the function z given by this model. You can guess its form by examining the simulation result. Verify that it satisfies (C.3) by differentiating.

4. Change the gain block to have value 0.9 instead of -0.9 and re-run the model. What happens? Is the system stable? (Stable means that if the input is bounded for all time, then the output is bounded for all time. In this case, clearly the input is bounded since it is zero.) Give an analytical formula for z for this model.

5. Experiment with values of the gain parameter. Determine over what range of values the system is stable.

C.6.3 Independent section

Continuous-time linear state-space models are reasonable for some musical instruments. In this exercise, we will simulate an idealized and a more realistic tuning fork, which is a particularly simple instrument to model. The model will be two-dimensional continuous-time state-space model.

Consider the state and output equations (C.1) and (C.2). Since the model is two dimensional, the state at each time is now a two-dimensional vector. The “initial condition” parameter of the Integrator block in Simulink can be given a vector. Set the initial value to the column vector

$$z(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{C.4}$$

The factor A must be a 2×2 matrix if the state is a two dimensional column vector. Unfortunately, the Gain block in Simulink cannot be given a matrix parameter. You must replace the Gain block with the MatrixGain block, also found in the “Math” library under “Simulink” in the library browser.

At first, we will assume there is no input, and we will examine the state response. Thus, we are only concerned at first with the simplified state equation

$$\dot{z}(t) = Az(t). \tag{C.5}$$

Recall that in chapter 2, equation (2.11) states that the displacement $x(t)$ at time t of a tine of the tuning fork satisfies the differential equation

$$\ddot{x}(t) = -\omega_0^2 x(t)$$

where ω_0 is constant that depends on the mass and stiffness of the tine, and and where $\ddot{x}(t)$ denotes the second derivative with respect to time of x (see box on page56). This does not have the form of (C.5). However, we can put it in that form using a simple trick. Let

$$z(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix}$$

and observe that

$$\dot{z}(t) = \begin{bmatrix} \dot{x}(t) \\ \ddot{x}(t) \end{bmatrix}.$$

Thus, we can write (C.5) as

$$\dot{z}(t) = \begin{bmatrix} \dot{x}(t) \\ \ddot{x}(t) \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix}$$

for suitably chosen constants $a_{1,1}$, $a_{1,2}$, $a_{2,1}$, and $a_{2,2}$.

1. Find $a_{1,1}$, $a_{1,2}$, $a_{2,1}$, and $a_{2,2}$ for the tuning fork model.

2. Use Simulink to plot the state response of the tuning fork when the initial state is given by (C.4). You will have to pick a value of ω_0. Use Simulink to help you find a value of ω_0 so that the state completes one cycle in 10 time units. Each sample of the state response has two elements. These represent the displacement and speed, respectively, of the tuning fork tine in the model. The displacement is what directly translates into sound.

3. Change ω_0 so that the state has a frequency of 440 Hz, assuming the time units are seconds. Change the simulation parameters so that you run the model through 5 complete cycles.

4. Change the simulation parameters so that you run the model through 1 second. Use the Simulink To Workspace block to write the result to the workspace, and then use the Matlab soundsc function to listen to it. **Note**: You will need to set the sample time parameter of the To Workspace block to 1/8000. You will also need to specify that the save format should be a matrix. For your lab report, print your block diagram and annotate it with all the parameters that have values different from the defaults.

5. In practice, a tuning fork will not oscillate forever as the model does. We can add damping by modifying the matrix A. Try replacing the zero value of $a_{2,2}$ with -10. What happens to the sound? This is called **damping**. Experiment with different values for $a_{2,2}$. Describe how the different values affect the sound. Determine (experimentally) for what values of $a_{2,2}$ the system is stable.

6. A tuning fork is not much of a musical instrument. Its sound is too pure (spectrally). A guitar string, however, operates on similar principles as the tuning fork, but has a much more appealing sound.

   A tuning fork vibrates with only one mode. A guitar string, however, vibrates with multiple modes, as illustrated in figure C.5. Each of these vibrations produces a different frequency. The top one in the figure produces the lowest frequency, called the **fundamental**, which is typically the frequency of the note being played, such as 440 Hz for A-440. The next mode produces a component of the sound at twice that frequency, 880 Hz; this component is called the **first harmonic**. The third produces three times the frequency, 1320 Hz, and the fourth produces four times the fundamental, 1760 Hz; these components are the second and third harmonics.

   If the guitar string is undamped, and the fundamental frequency is f_0 Hz, then the the combined sound is a linear combination of the fundamental and the three (or more) harmonics. This can be written as a continuous-time function y where for all $t \in$ *Reals*,

$$y(t) = \sum_{k=0}^{N} c_k sin(2\pi f_k t)$$

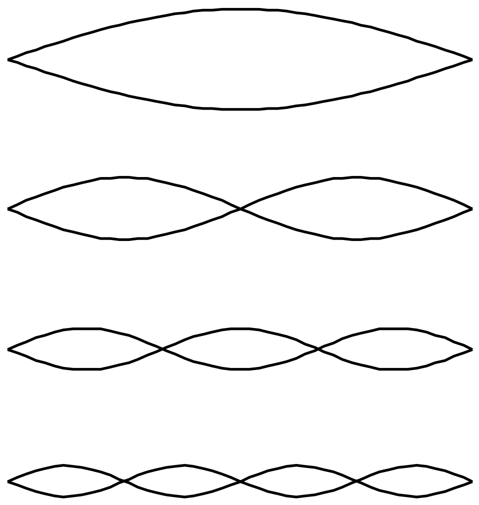


Figure C.5: Four modes of vibration of a guitar string.

where N is the number of harmonics and c_k gives the relative weights of these harmonics. The values of c_k will depend on the guitar construction and how it is played, and affect the **timbre** of the sound.

The model you have constructed above generates a damped sinusoid at 440 Hz. Create a Simulink model that produces a fundamental of 440 Hz plus three harmonics. Experiment with the amplitudes of the harmonics relative to the fundamental, as well as with the rates of decay of the four components. Note how the quality of the sound changes. Your report should include a printout of your model with the parameter values that you have chosen to get a sound like that of a plucked string.

Instructor Verification Sheet for C.6

Name: ______________________ Date: ______________________

1. Value of a.

   Instructor verification: ______________________

2. Plot of the state response.

   Instructor verification: ______________________

3. Formula for function z. Verified by differentiating.

   Instructor verification: ______________________

4. Formula for function z.

   Instructor verification: ______________________

5. Range of values for the gain over which the system is stable.

   Instructor verification: ______________________