C.8 Comb filters

The purpose of this lab is to use a kind of filter called a comb filter to deeply explore concepts of impulse response and frequency response.

The lab uses Simulink, like lab C.6. Unlike lab C.6, it will use Simulink for discrete-time processing. Be warned that discrete-time processing is not the best part of Simulink, so some operations will be awkward. Moreover, the blocks in the block libraries that support discrete-time processing are not well organized. It can be difficult to discover how to do something as simple as an N-sample delay or an impulse source. We will identify the blocks you will need.

The lab is self contained, in the sense that no additional documentation for Simulink is needed. As in lab C.6, be warned that the on-line documentation is not as good for Simulink as for Matlab. You will want to follow our instructions closely, or you are likely to discover very puzzling behavior.

C.8.1 Background

To run Simulink, start Matlab and type `simulink` at the command prompt. This will open the Simulink library browser. The library browser is a hierarchical listing of libraries with blocks. The names of the libraries are (usually) suggestive of the contents, although sometimes blocks are found in surprising places, and some of the libraries have meaningless names (such as "Simulink").

Here, we explain some of the techniques you will need to implement the lab. You may wish to skim these now and return them when you need them.

Simulation Parameters

First, since we will be processing audio signals with a sample rate of 8 kHz, you need to force Simulink to execute the model as a discrete-time model with sample rate 8 kHz (recall that Simulink excels at continuous-time modeling). Open a blank model by clicking on the document icon at the upper left of the library browser window. Find the Simulation menu in that window, and select Parameters. Set the parameters so that the window looks like what is shown in figure C.9. Specifically, set the stop time to 4.0 (seconds), the solver options to "Fixed-step" and "discrete (no continuous states)," and the fixed step size to 1/8000.

Reading and Writing Audio Signals

Surprisingly, Simulink is more limited and awkward than Matlab in its ability to read and write audio files. Consequently, the following will seem like more trouble than it is worth. Bear with us. Simulink only supports Microsoft wave files, which typically have the suffix ".wav". You may obtain a suitable audio file for this lab at

```
http://www.eecs.berkeley.edu/~eal/eecs20/sounds/voice.wav
```

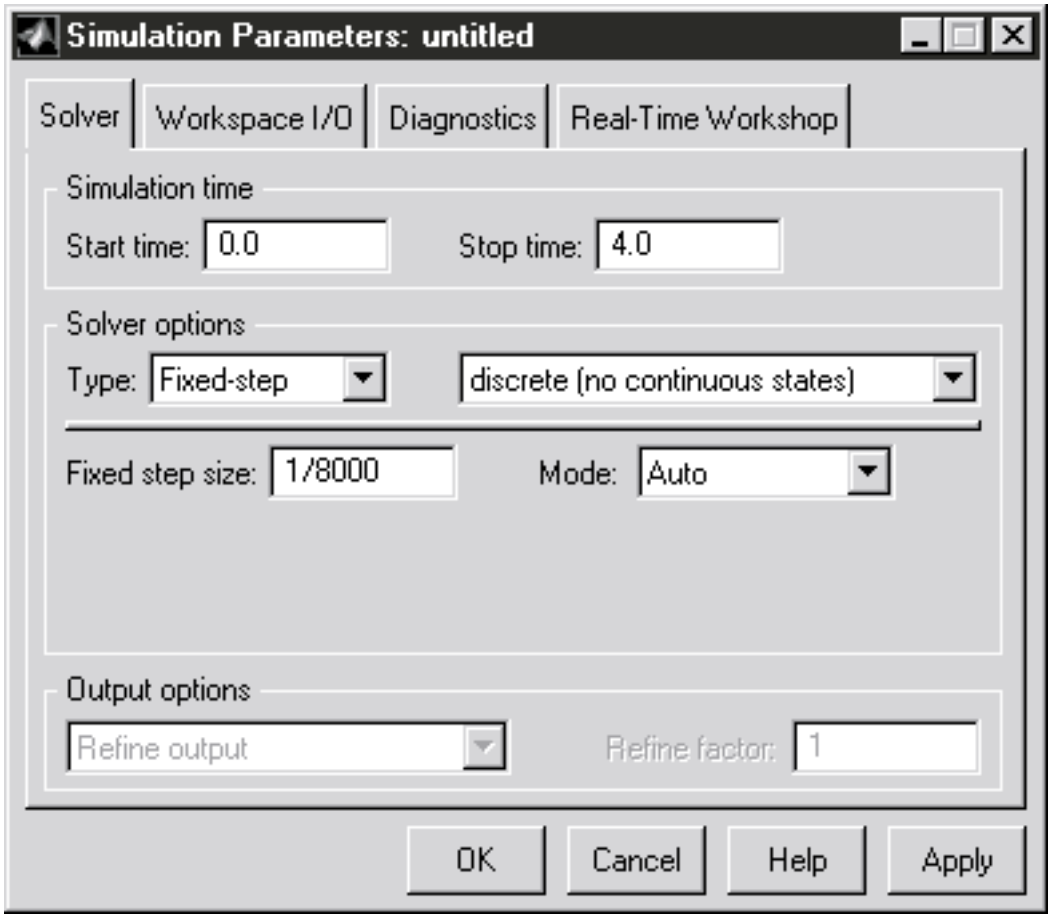

Figure C.9: Simulation parameters for discrete-time audio processing in Simulink.

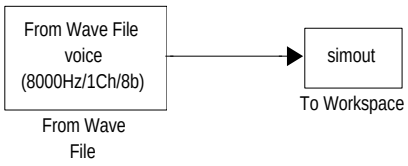

Figure C.10: Test model for Simulink audio.

In Netscape you can go to

```
http://www.eecs.berkeley.edu/~eal/eecs20/sounds/
```

and then right click on the `voice.wav` filename to bring up a menu, and choose "Save Link As..." to save the file to your local disk. It is best to then, in the Matlab command window, to change the current working directory to the one in which you stored the file using the `cd` command. This will make it easier to use the file.

To make sure we can process audio signals, create the test model shown in figure C.10. To do this, in a new model window with the simulation parameters set as explained in "Simulation Parameters" on page 462, create an instance of the block called `From Wave File`. This block can be found in the library browser under `DSP Blockset` and `DSP Sources`. Set the parameters of that block to

```
File name: voice.wav
Samples per frame: 1
```

The first parameter assumes you have set the current working directory to the directory containing the `voice.wav` file. The second indicates to Simulink that it should produce audio samples one at a time, rather than collecting them into vectors to produce many at once.

Next, find the `To Workspace` block in the Simulink block library, under Sinks. Create an instance of that block in your model. Edit its parameters to change the "Save format" to "Matrix". You can leave other parameters at their default values.

Connect the blocks as shown in figure C.10.

Assuming the simulation parameters have been set as explained in "Simulation Parameters" on page 462, you can now run the model by invoking the Start command under the Simulation menu. This will result in a new variable called `simout` appearing in the Matlab workspace. In the Matlab command window, do

```
soundsc(simout)
```

to listen to the voice signal.

Note that the `DSP Sinks` library has a block called `To Wave Device`, which in theory will produce audio directly to the audio device. In practice, however, it seems much easier to use the `To`

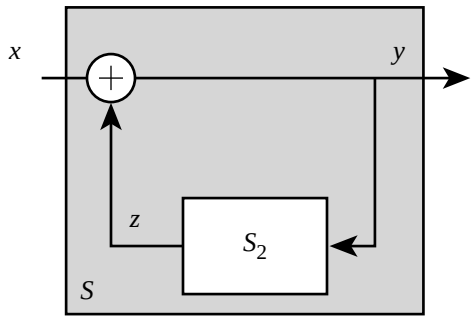

Figure C.11: Comb filter modeled as a feedback system.

`Workspace` block and the `soundsc` command. For one thing, `soundsc` scales the audio signal automatically. It also circumvents difficulties with real-time performance, platform dependence problems, and ideosyncrasies with buffering. However, if you wish to try the `To Wave Device` block, and can figure out how to get it to work, feel free to use it.

C.8.2 In-lab section

1. Consider the equation

$$\forall\, n \in \textit{Integers}, \quad y(n) = x(n) + \alpha y(n - N) \tag{C.8}$$

   for some real constant $\alpha < 1$ and integer constant $N > 0$. Assume the sample rate is 8 kHz. The input is $x(n)$ and the output is $y(n)$. The equation describes an LTI system where the output is delayed, scaled, and feb back. Such a system is called a **comb filter**, for reasons that will become apparent in this lab. The filter can be viewed as a feedback structure, as shown in figure C.11, where S_2 is a system with input y and output z. Give a similar equation describing S_2, relating y and z.

2. Implement in Simulink the comb filter from part (a). Provide as input the file `voice.wav` (see page 462). Send the output to the workspace, just like figure C.10, so that you can use `soundsc` to listen to the result. You will probably need the `Gain` and `Sum` blocks, which you can find in the Simlink, Math library. The delay in the feedback path can be implemented by the `Integer Delay` block, which you can find in the DSP Blockset, General DSP, Signal Operations library.

   Experiment with the values of N. Try $N = 2000$ and $N = 50$ and describe qualitatively the difference. With $N = 50$, the effect is called a sewer pipe effect. Why? Can you relate the physics of sound in a sewer pipe with our mathematical model? **Hint**: The speed of sound in air is approximately

$$331.5 + 0.6T \text{meters/second}$$

   where T is the temperature in degress celcius. Thus, at 20 degrees, sound travels at about 343.7 meters/second. A delay of $N = 50$ samples at an 8 kHz sample rate is equal to the time it takes sound to travel roughly 2 meters, twice the diameter of a 1 meter sewer pipe.

Experiment with the value of α. What happens when $\alpha = 0$? What happens when $\alpha = 1$? When $\alpha > 1$? You may wish to plot the output in addition to listening to it.

3. Modify your Simulink model so that its output is the first one second (the first 8001 samples) of the impulse response of the system defined by (C.8), with $\alpha = 0.99$ and $N = 40$.

   The simplest approach is to provide an impulse as an input. To do that, use the `Discrete Pulse Generator` block, found in the Simulink, Sources. This block can be (sort of) configured to generate a Kronecker delta function. Set its amplitude to 1, its period to something longer than the total number of samples (i.e. larger than 8001), its pulse width to 1, its phase delay to 0, and its sample time to 1/8000.

   You will also want to change the simulation parameters to execute your system for 1 second instead of 4.

   Listen to the impulse response. Plot it. Can you identify the tone that you hear? Is it a musical note? **Hint**: Over short intervals, a small fraction of a second, the impulse response is roughly periodic. What is its period?

4. In the next lab you will modify the comb filter to generate excellent musical sounds resembling plucked strings, such as guitars. As a first step towards that goal, we can make a much less mechanical sound than the impulse response by initializing the delay with random data. Modify your Simulink model so that the comb filter has no input, and instead of an input, the `Integer Delay` block is given random initial conditions. Use $\alpha = 0.99$ and $N = 40$, and change the parameters of the `Integer Delay` block so that its initial conditions are given by

   ```
   randn(1,40)
   ```

   The Matlab `randn` function returns a vector of random numbers (try `help randn` in the Matlab command window).

   Listen to the result. Compare it to the sound of the impulse response. It should be richer, and less mechanical, but should have the same tone. It is also louder (even though `soundsc` scales the sound).

C.8.3 Independent section

The comb filter is an LTI system. Figure C.11 is a special case of the feedback system considered in section 8.5.2, which is shown there to be LTI. Thus, if the input is

$$x(n) = e^{j\omega n}$$

then the output is

$$y(n) = H(\omega)e^{j\omega n}$$

where $H\colon \mathit{Reals} \to \mathit{Complex}$ is the frequency response. Find the frequency response of the comb filter. Plot the magnitude of the frequency response over the range 0 to 4 kHz using Matlab. Why is it called a comb filter? Explain the connection between the tone that you hear and the frequency response.

Instructor Verification Sheet for C.8

Name: _____ Date: _____

1. Found an equation for S_2, relating y and z.

   Instructor verification: _____

2. Constructed Simulink model and obtained both sewer pipe effect and echo effect.

   Instructor verification: _____

3. Constructed the impulse response and identified the tone.

   Instructor verification: _____

4. Created sound with random values in the feedback delay.

   Instructor verification: _____