Assignment: Homework 6

What to Hand In

1. Your code with appropriate comments and a readme file. You are responsible for making us understand and know how to run your code.
2. A report with answers, figures, and explanations.
3. Any sound files asked for in the problems.

How to Hand It In

1. Put all your files (Matlab portion only) in one folder. Compress this folder and name it <firstname>_<lastname>_HW6 .zip. For example, "Zhiyao_Duan_HW6.zip".
2. Submit via blackboard to the appropriate entry. You can make multiple attempts but only the last attempt will be graded.
3. Demonstrate your DSP portion to the TAs during the lab session.

When to Hand It In

It is due immediately at the beginning of the day (i.e., 0:00AM) specified on the course calendar. **Late assignments will receive a grade deduction of 20% of the full grade each day.**

Problems (10 points in total)

1. (3.5 points) Fractional Delay Line. In this part, you will implement and evaluate both the linear interpolation method and the first-order all-pass interpolation method for building a fractional delay line.
   a. (0.5 points) Fill in blanks in *DelayLine.m*. This class implements a delay line with a fixed delay. It is required that you implement a specialized circular buffer internally for the delay line. In the *process* method/function, each time one sample is processed and the state is updated. A script *delayLineTest.m* can be used to test some basic cases of your implementation. It also shows you the basic usage of the intended interface. Currently, it only checks if a correct delay is produced. Feel free to add test cases to the unit test.
   b. (1 points) Fill in blanks in *FractionalDelayLine.m*. This class implements a fractional delay line with a fixed delay. You need to implement the *process* method/function for both the linear interpolation method and the first-order all-pass interpolation method. A script *fractionalDelayLineTest.m* provides a unit test for this class. Feel free to add test cases to the unit test.
   Note: In your implementation, you will need to split a large delay quantity into an integer and a fractional part. Also, for firstOrderAllpass interpolator, the fractional part works better (more flat phase delay) when $\Delta \in [0.3, 1.3]$, therefore, for firstOrderAllpass, adjust the split between integer part and fractional part to make $\Delta \in [0.3, 1.3]$.
   c. (2 points) Implement the fractional delay line on the DSP board. The Matlab implementation could be a direct reference for this C/C++ implementation. This is why we used object-oriented programming in Matlab.

2. (4.5 points) A Simple Plucked String Model. In this problem, you will implement a simple plucked string model shown in Figure 1.

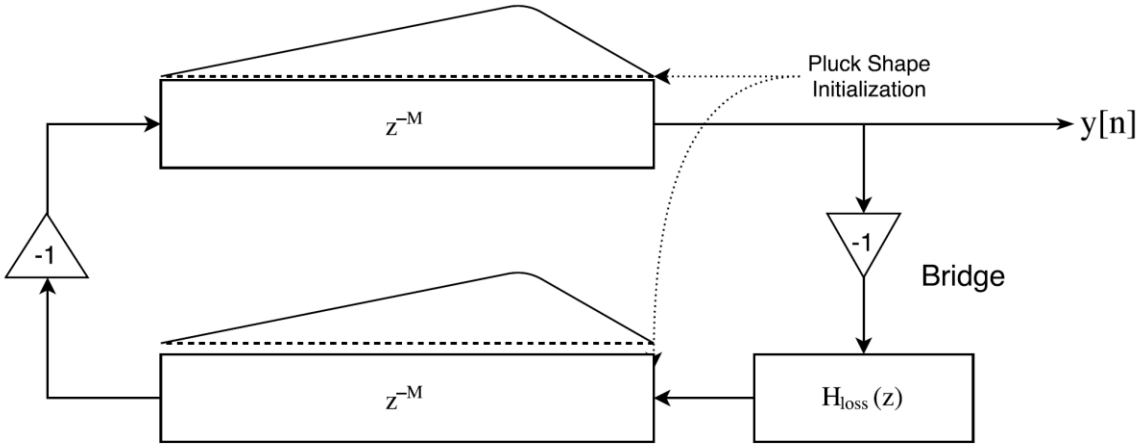


Figure 1. A simple plucked string model.

a. (0.5 points) Complete the function *genPluckShape.m*, which generates the plucking shape given its position ($pos \in [0,1]$) and the length (in samples) of the shape. For simplicity, the plucking position is rounded to integer samples in this function.
b. (0.5 points) Complete the class *LossFilter.m*, which implements the simplest loss filter:

$$H_{loss}(z) = g\frac{1+a}{1+az^{-1}},$$

where $0 < g < 1$ and $-1 < a \leq 0$. In the *process* method/function, each time one sample is processed and the state is updated.
c. (1 point) Generate a 1-second long A4 (440 Hz) tone with a sampling rate of 44.1 kHz according to the signal diagram in Figure 1. Save it as note_440hz.wav. You can pick any a and g for the loss filter that you like and you are encouraged to try different combinations of a, g, pos, and the round-trip delay line length $2M$. You will need to initialize your two delay lines with your generated pluck shape.
Note: Pay attention to the directions of the two delay lines, which are opposite. You are free to add some random noise to the shape initialization for timbral variations.
d. (1.5 points) Implement the simple plucked string model on the DSP board. Your Matlab implementation could be a direct reference for this C/C++ implementation.
e. (1 points) Create 3 strings with each tuned to a different frequency to form a chord of your favorite (Free free to create 4 or more strings if you like more complex chords). For each string, use a slider with two positions (i.e., a switch) to control whether the string will be plucked or not. In the interrupt function, activate the strings whose switches are set on every two seconds. To activate a string, simply re-set the pluck shape to the triangle as in Figure 1. Enjoy the chords you made.
Note: Follow the Gel file example to set up the sliders.

3. (2 points) Commuted Synthesis. In this problem, you will implement a basic commuted synthesis model, which is shown in Figure 2, for generating a violin pizzicato sound. You will use classes developed in previous problems.
   a. (0.5 points) Fill in blanks in *CommutedString.m*.
   b. (0.5 points) Create a commuted string object for 425.4019 Hz, which is the pitch estimated from the included *pizz_A4_mono.wav*. Set the parameters a and g appropriately so the impulse response (i.e., output of the commuted string model taking an impulse input) has roughly the same energy decay with that of *pizz_A4_mono.wav*. Plot this impulse response and the waveform of *pizz_A4_mono.wav* for 0.5 seconds.
   c. (1 point) Extract the excitation signal from *pizz_A4_mono.wav* by de-convolving it with the impulse response of the commuted string model in (b). Then use this excitation signal and the commuted string model to generate 25 semitones from A3 to A5, each of 1 second long. Concatenate these notes together and save the file as *Generated_A4.wav*. Comment on the synthesized notes. Did you hear any artifacts? Why?
   Note: For de-convolution, you can use the *deconv* function in Matlab.

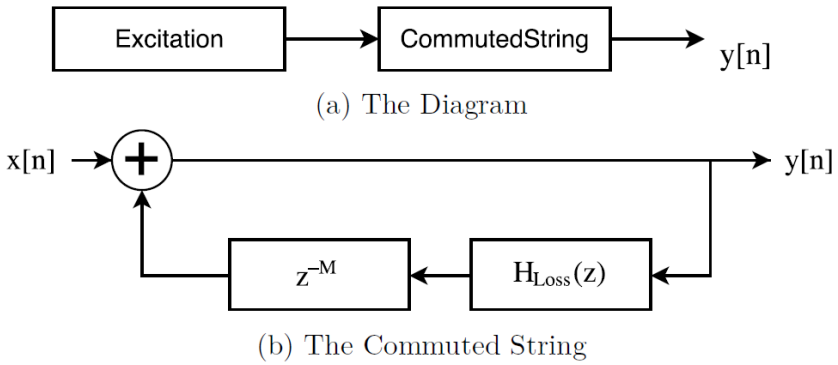


Figure 2. A basic commuted synthesis model.