Due on the beginning of the day (0:00AM) specified on the course calendar. Late submission will not be graded.

Deliverables:

1. Codes that can reproduce all your results and a readme file with instructions on how to run your code. It is your responsibility to make all your codes readable. Detailed comments and meaningful variable names are highly recommended.
2. A report with answers, figures, and explanations. It is highly recommended that you export/save your figures with decent resolution.
3. Any other files required.

Submission Instruction:

1. Put all your files in one folder.
2. Compress this folder and name it <firstname>_<lastname>_HW5.zip. For example, *Zhiyao_Duan_HW5.zip*
3. Submit via Blackboard to the appropriate entry.

Matlab Problems

Programming Tips: In this assignment, you will write classes in Matlab. In Matlab, the scope within a class member function does not include the properties of the class automatically, which is different from many statically typed language like C++/Java/etc., you have to refer them by a object reference, which is the first argument of a member function.

1. (3 points) Fractional Delay Line

In this part, you will implement and evaluate both linear interpolation and first order all-pass interpolation for building a fractional delay line.

(a) (1 point) Fill in blanks in *DelayLine.m*. This class implements a delay line with fixed delay. It is required that you implement a specialized circular buffer internally for the delay line. In the *process* method, 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 point) Modify your *DelayLine.m* to make it capable of taking a non-interger length of delay in its constructor. Name the new class *FractionalDelayLine.m*. The constructor now has the signature:

```
function obj = FractionalDelayLine(delayLineLength, interpolationMethod)
```

where *interpolationMethod* takes two string values: *linearInterp*, and *firstOrderAllpass*. *fractionalDelayLineTest.m* also provides a unit test for this class. Feel free to add test cases to the unit test. In your implementatation, 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*, set $\Delta \in [0.3, 1.3]$ if possible; you will need to process the case where the total delay length < 0.3 .

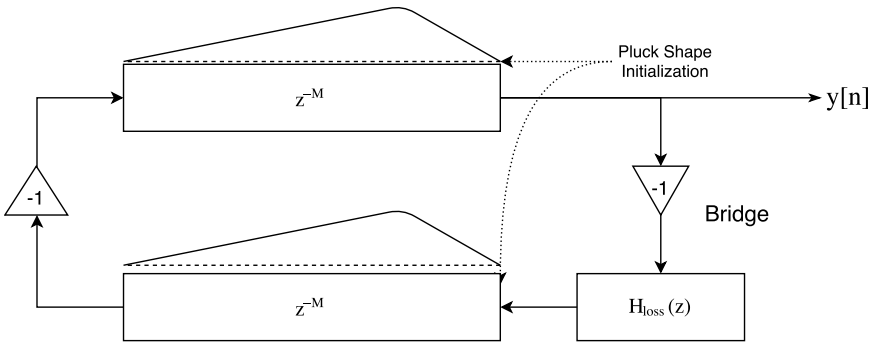


Figure 1: A simple Plucked string model

(c) (1 point) Plot various magnitude responses and phase delays for both linear interpolation and first order all-pass interpolation where $delayLineLength \in [0, 2]$ using the class you have finished in the previous problem. Discuss when it is appropriate to use these low-order methods.

2. (2 points) A Simple Plucked String

In this problem, you will implement a simple plucked string model, which is shown in figure 1.

(a) (0.5 points) Complete the function specified in *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 in this function.

(b) (0.5 points) Complete the class *LossFilter.m*, which implements the simplest loss filter in the diagram:

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

where $0 < g < 1$ and $-1 < a \leq 0$. In the *process* method, each time one sample is processed and the state is updated.

(c) (1 point) Generate one A4 (440hz) note according to the signal diagram. 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.

Hint: pay attention to the orientation of the two delay lines. You are free to add some random noise to the initialization for some variations[1].

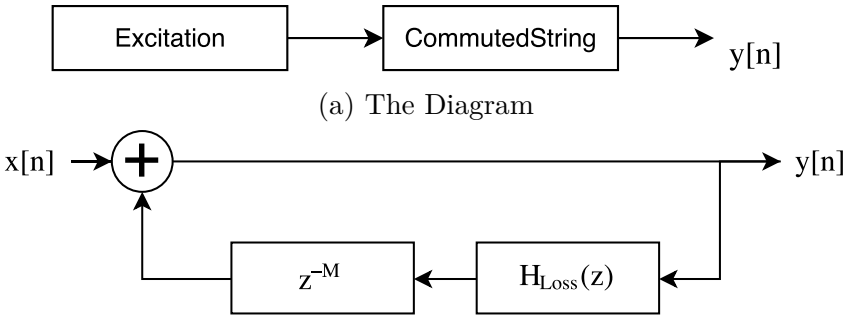
3. (2 points + 2 bonus points) Commuted Synthesis

In this problem, you will implement a basic commuted synthesis model, which is shown in Figure 2, for generating violin pizzicato sound. You will use classes developed in previous problems.

(a) (0.5 points) Fill in blanks in *CommutedString.m*

(b) (0.25 points) Estimate the fundamental frequency of *pizz_A4_mono.wav*. In this simple situation (one stable pitch for an entire file), you can merely pick up the first peak

[1]The first synthesis algorithm of this kind, the *Karplus-Strong algorithm*, only initializes with pure random noises.



(a) The Diagram

(b) The Commuted String

Figure 2: A basic commuted synthesis Model

in the autocorrelation function after the first zero-crossing, or calculate the mean distance between first k peaks in the autocorrelation function.

(c) (0.25 points) Use your estimated fundamental frequency and arbitrary values of a and g to initialize the commuted string. Generate the impulse response and plot the spectrogram of both the string response and the real sound. Tune a and g manually until the decay time observed in the spectrogram roughly matches the real sound.

(d) (1 point, *optional*) Try to implement the method described in `https://ccrma.stanford.edu/realsimple/phys_mod_overview/Loop_Filter_Estimation.html` to determine a and g. Does this method work in this case? Can this method serve as an automated method for estimating the loop (loss) filter? Why?

(e) (0.25 points) Extract the excitation signal from *pizz_A4_mono.wav* (The most handy method is Matlab's *deconv* function. Note that this function may require zero padding to obtain a result because this function is purely algebraic. Alternatively, you can implement the MSE deconvolution algorithm, where we can apply regularization). You can truncate and window the extracted excitation to make it easy to use and to make it free of some obvious artifacts in the time domain.

(f) (0.75 point) Now you can generate sound with different pitches using the same excitation and parameters you obtained previously. Generate a chromatic scale from A3 to A5, with standard tuning ($A4 = 440Hz$). Repeat the process of tuning a and g, extracting the excitation, generating the scale, until the generated notes sound similar to the real instrument sound. Save your final audio clip as *Generated_A3_A5.wav*.

(g) (0.25 points, *optional*) Have you noticed some harmonic artifact in you generated sound? Give an explanation. **Hint:** Examine the spectrogram of both excitation and the generated sound.

(h) (0.75 points, *optional*) Implement one way of fixing the harmonic artifacts. Save your final chromatic scale as *Generated_A3_A5_2.wav*. Illustrate the effect by comparing before/after spectrograms.

Hint: You may remove unwanted harmonics with spectral processing; instead of forcing a range of bins to be zero, flatten them with a level determined by their neighbors. You can have multiple choices to set the value. Feel free to use other methods.

(3 points) Lab Problems

Please refer to *OMAP StartUp* on the course website for instructions on how to compile and run codes on the DSP board. Try *GEL Tutorial* on the course website for how to make a slider for debugging purpose.

(a) (2 points) Implement the simple plucked string model on the DSP board, which corresponds to Problem 1-2. **You Matlab implementation should be a direct reference for writing these codes. You should finish them first.** In practice, you can implement on your computer with minimal standard C/C++. With properly designed API which should resemble the one in the Matlab implementation, this code should be able to be called from codes for the DSP board without effort.

(b) (1 point) Create 5 strings with each tuned to a different frequency. For each step, Output the summation of output samples from all these 5 strings. Use a slider (see the Gel file example) to select which one to sound. When you want to excite a string, add the shape to the delay line of that string.