A Description of Project 1

Zhou Kaining 3180101148

1 General Introduction

This whole project, lab assignment #1, was accomplished by me alone. The Matlab code and files are in the .rar file titled as '3180101148 Zhoukaining Lab1'. The structure of code is as follows.

For Problem 1 & 2, I integrate the code into one .m file, titled as 'main_Problem1_2.m' in the directory named 'computer explorations toolbox', which is the main script. For the sake of a better presentation effect, I divide the code into coherent, relevant but relatively independent code sections which begin with '%%'. For Problem 3, I write the main script, 'main_Problem3.m' with other necessary and auxiliary functions to finish the most exercises. Just like Problem 1 & 2, the 'main_Problem3.m' is separated by section, too. The last exercise is carried out by a independent script called 'knzhou.m'. All the files are included in a directory 'Files for Problem3\M-Files'.

Please run the script 'main_Problem1_2.m' for Problem 1 & 2; run 'main_Problem3.m' for Problem 3. For all problems, please run the code section by section from the first one, then sequentially, in their current directory. The output is in the form of graphics and characters in the command line as well as the sounds. Running the whole script one time is not recommended, for the previous output figures will be covered by the next and the videos to be played will be in disorder.

2 Problem 1

In order to run the 'main_Problem1_2.m' and solve the problems correctly, I modify the display color of the lines in 'pzplot.m', and use the command 'clf' instead of 'clg' in 'dpzplot.m'. Please check up on that. Open the 'main_Problem1_2.m', change the directory to 'computer explorations toolbox' in Matlab, then start to run the first section.

2.1 Problem 1.1

(a). To enhance the uniformity and scalability of the solution, I use a *cell* to store vectors of different length and dimension. The labeled pole-zero diagram is shown in Figure 1.

(b). By calling function **pzplot**, ROC can be derived as Figure 2 shows. Because the system is rational, ROC must cover $j\omega$ axis. Thus, ROC of all three functions is $Re\{s\} > -1$.

(c). Perform Laplace Transform on the differential equation. Thus, we obtain

$$sY(s) - 3Y(s) = s^2X(s) + 2sX(s) + 5X(s)$$

$$H(s) = \frac{Y(s)}{X(s)} = \frac{s^2 + 2s + 5}{s - 3}$$

so correspondingly, $b = \begin{bmatrix} 1 & 2 & 5 \end{bmatrix}, a = \begin{bmatrix} 1 & -3 \end{bmatrix}$. And we solve the ROC of this causal system: $Re\{s\} > 3$. The diagram is illustrated in Figure 3.

(d). The function **pzplot** takes in a argument called 'ROC' which is a point in ROC of one system function. We know that poles are excluded from ROC and form the boundary of ROC, so we should choose regions separated by poles to cover the point 'ROC'. As inferred, the function **pzplot** selects the nearest poles to that point from both left and right sides. The chosen poles constitute the periphery of ROC. By this means ROC is determined.

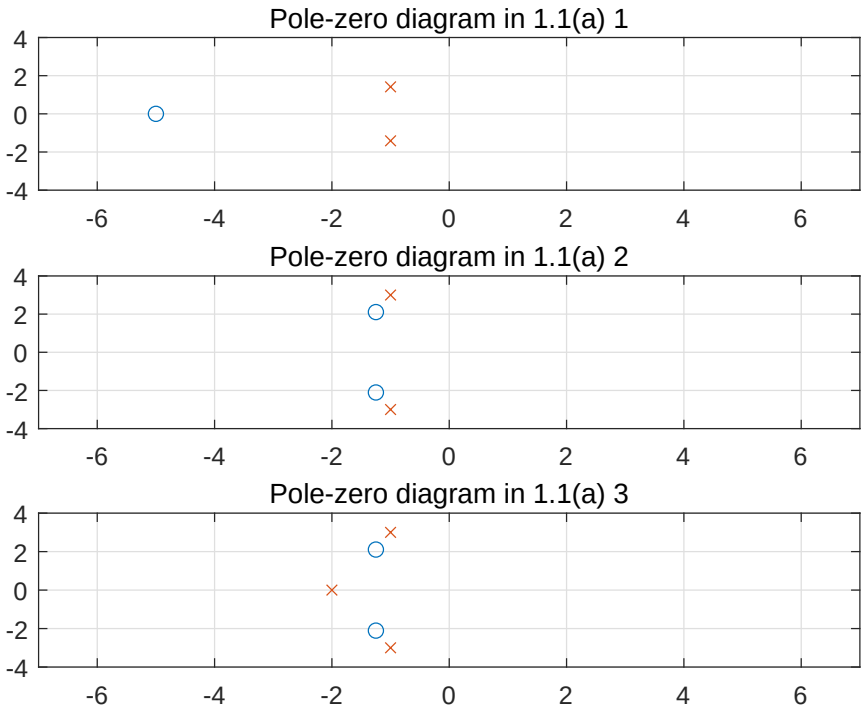


Figure 1: Pole-zero diagram in Problem 1.1(a).

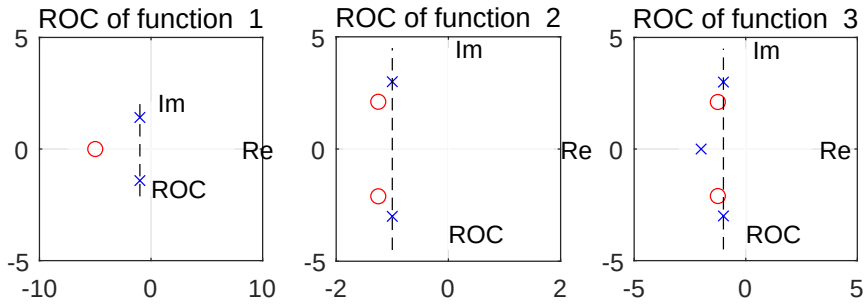


Figure 2: Pole-zero diagram in Problem 1.1(b) with different ROC presented.

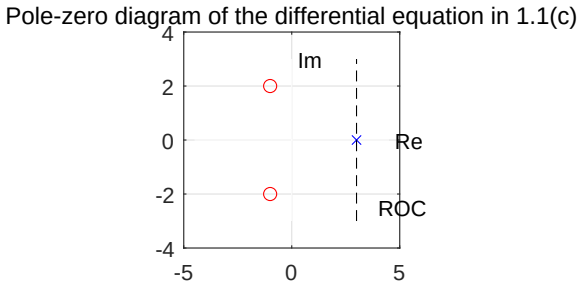


Figure 3: Pole-zero diagram in Problem 1.1(c).

2.2 Problem 1.2

(a). Obviously $b = \begin{bmatrix} 1 & -1 \end{bmatrix}, a = \begin{bmatrix} 1 & 3 & 2 \end{bmatrix}$. Enter them into the function **dpzplot**. The zero-pole diagram is shown as Figure 4.

(b). Do the Z transform first and obtain

$$Y(z) + z^{-1}Y(z) + 0.5z^{-2}Y(z) = X(z)$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 + z^{-1} + 0.5z^{-2}}$$

$b = \begin{bmatrix} 1 \end{bmatrix}, a = \begin{bmatrix} 1 & 1 & 0.5 \end{bmatrix}$. The zero-pole diagram is shown as Figure 5.

(c). Similar to the previous question, we can get

$$H(z) = \frac{1 + 0.5z^{-1}}{1 - 1.25z^{-1} + 0.75z^{-2} - 0.125z^{-3}}$$

$b = \begin{bmatrix} 1 & 0.5 \end{bmatrix}, a = \begin{bmatrix} 1 & -1.25 & 0.75 & -0.125 \end{bmatrix}$. The zero-pole diagram is shown as Figure **??**.

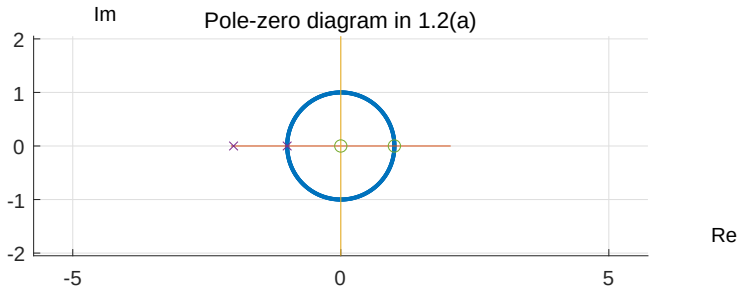


Figure 4: Pole-zero diagram in Problem 1.2(a).

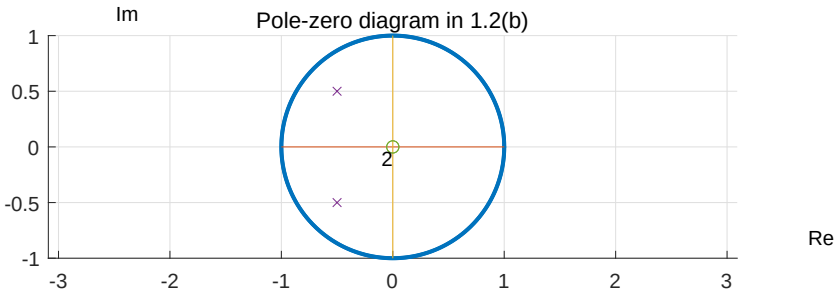


Figure 5: Pole-zero diagram in Problem 1.2(b).

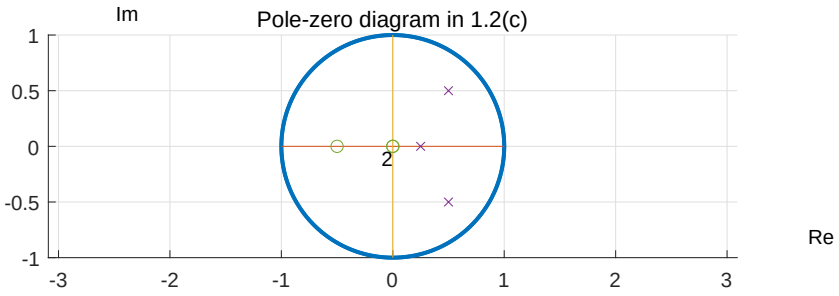


Figure 6: Pole-zero diagram in Problem 1.2(c).

3 Problem 2

3.1 Problem 2.1

(a). One of the properties of convolution reveals that only when the filter and the pattern to be found are centrosymmetric can the convolution meets maximum. What is more, the parameter n where convolution equals the maximum, is related to the delay of signals. So two things should be determined: the values of filter and their coordinates across time. The former is clear that the filter must be centrosymmetric to pattern, in this case, it must be $p[n] = \begin{bmatrix} 1 & -1 & -1 \end{bmatrix}$. The latter can be converted to finding the coordinates of filter's first non-zero entry.

That deserves more elaborate thinking. Beyond the scenario given by problems, we should deal with several occurrences of the patterns and different n given which indicates the location of patterns. Out of this consideration, to be more scalable and generic, I write the code which enables to find the number of patterns ever appear (in this question, the number is 1), and calculate the first coordinate of non-zero time step of filter (in this question it is -1) using given n (point centered on the desired patterns) and the first index of a maximum. Then I show the filter with values and time axis simultaneously with a loop. The matched filter is given by Figure **??**.

(b). Due to the requirement that white pixels and black pixels contribute positively to the answer, I transform the smiley image into binary image consisting of 1s (indicating white pixels) and -1s (indicating black pixels). As is explained above, the filter is centrosymmetry of the smiley pattern. There is a difference of convolution between 2D and 1D. That means we have to use padding to control the shape of output. Here I use the 'same' padding for I want to specify the precise coordinate of the smiley's nose. Because the nose is located at the center of the smiley, so, after using 'same' padding and function 'find', the return vector contains the coordinate I want. The coordinate is printed on the command interface, and the left-hand of Figure **??** is the smiley found among raw 'findsmiley.jpg'.

(c). The advanced task is to find smileys with artificial noise. Here I set excessive noise of that $mu = 60$ and $sigma = 60$. The right-hand of Figure **??** depicts the smiley found among the image with noise. We can see the program still output the right location which proves the code and algorithm quite robust. The cause is that we use binarization algorithm which largely attenuates the disturbance of the noise for it is reduced to either -1 and 1 according to a given threshold.

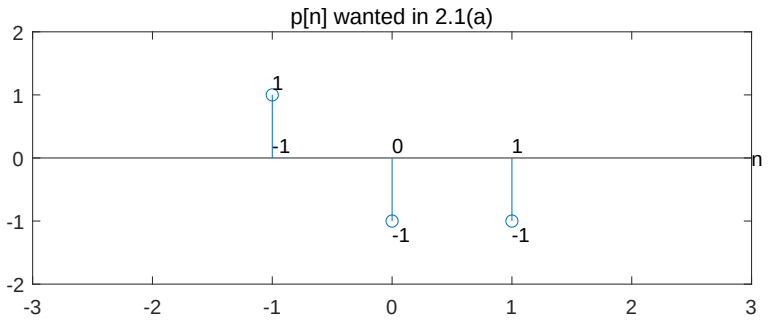
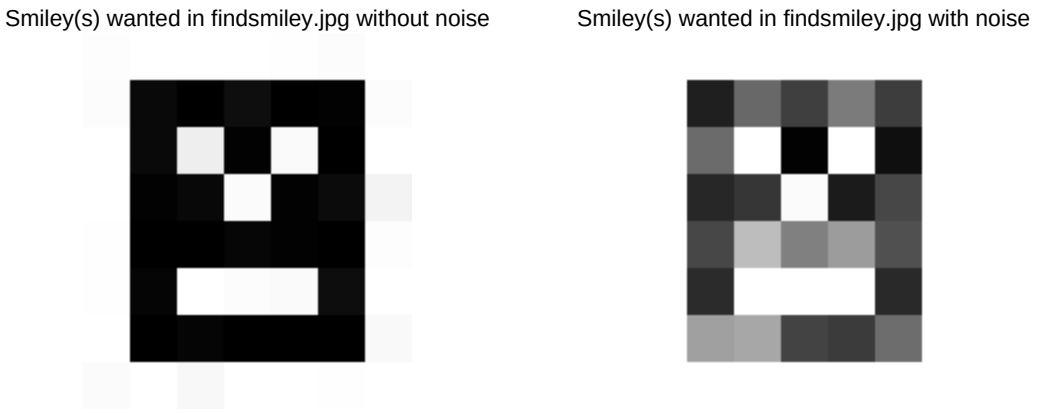


Figure 7: A matched filter in 2.1(a).



(a) 2.1(b) (b) 2.1(c)

Figure 8: Smiley found without and with noise in 2.1.

4 Problem 3

Exercise 1: I create several hierarchal cells to store the data in the same way for more designability. WThe wavesforms of each audio is displayed in Figure **??**. The volume is determined by the amplitude of the sound wave, I predict that the sound which has larger amplitude across time of duration has larger volume.

Exercise 2: By playing each audio, the pridiction above proves correct.

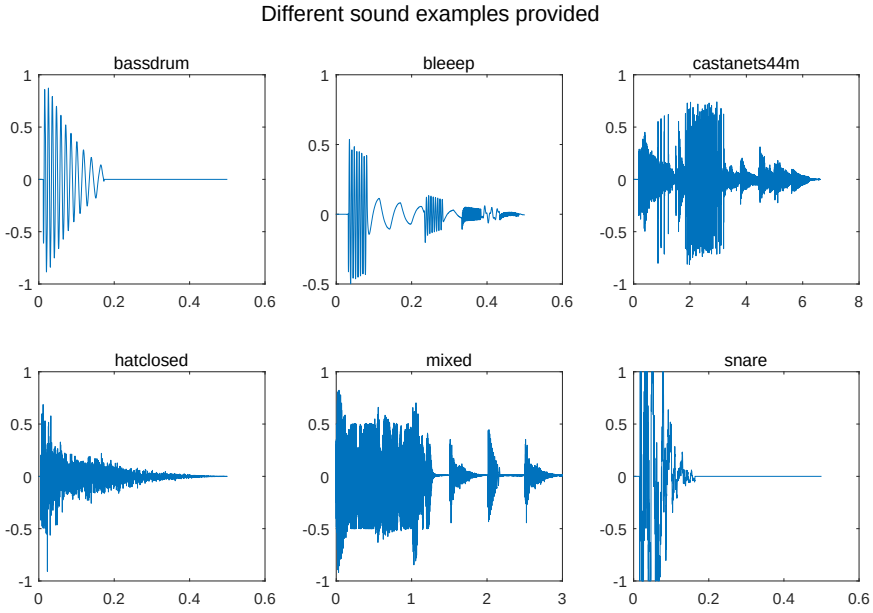


Figure 9: Depict the audios with plots.

Exercise 3: I select one audio randomly. For the chosen sound, I use function 'flipud' to reverse the samples, and plot the original and reversed ones together in Figure **??**. To play the audio sequentially and automatically, I use function 'audioplayer' rather than 'sound'.

Exercise 4: Please examine 'timescale.m'. I set the parameter $p = 2$ and $q = 3$ and call the function respectively. The faster version has speed $1.5\times$ and a higher pitch, while the slower one has speed $0.67\times$ and a lower pitch. Apparently that the pitch is highly related to frequency, which is changed by the function 'timescale'. Figure **??** presents three waveforms parallelly.

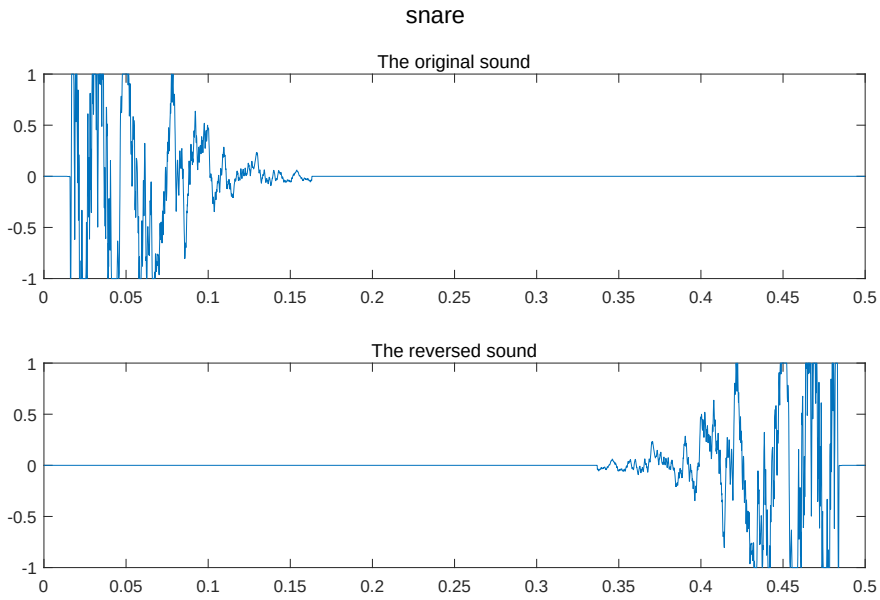


Figure 10: Two type of sounds.

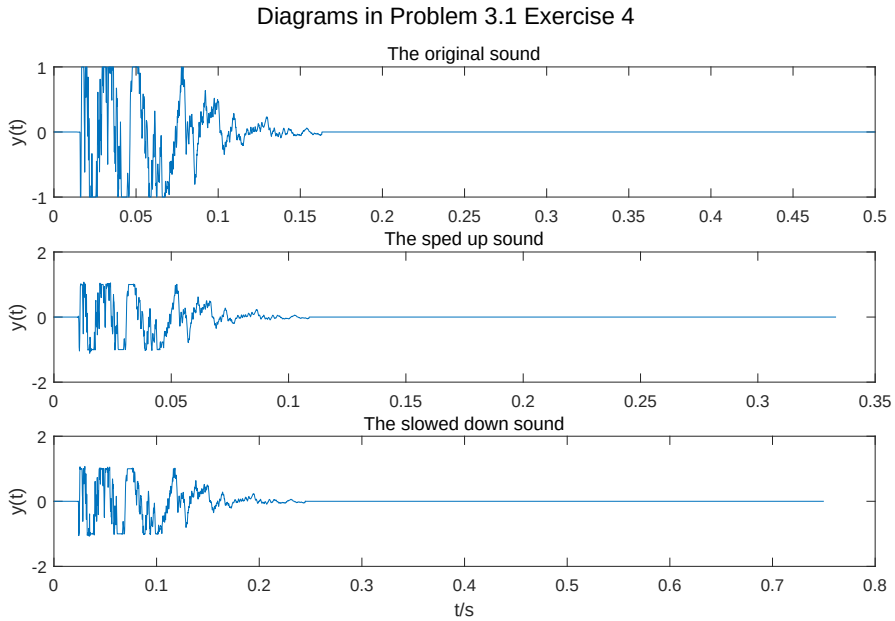


Figure 11: Different scales of time.

Exercise 5: Please examine 'fade.m'. The 'fade' function controls the amplitude of the wave. I define the fade process in a linear space. It takes one or two argument(s). By default, the input sample will end up 0;

if the second argument, 'level', is taken in, first we should clip it to the range from 0 to 1, then shrink the amplitude linearly. Figure **??** illustrates the fade process of sinusoidal input signal.

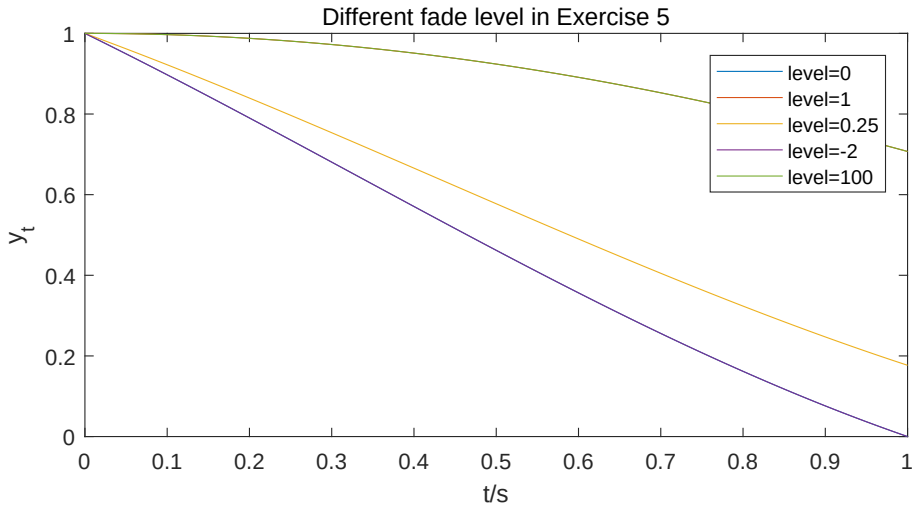


Figure 12: Fade function, and the input is defined as $y = \cos(0.25\pi t)$.

Exercise 6: Please examine 'repeat.m'. The 'repeat' function is used as an instrument that repeats the sound N times. If $N < 1$, function will not do anything; if $N \geq 1$, function will concatenate the original sound for N time(s) as Figure **??** shows.

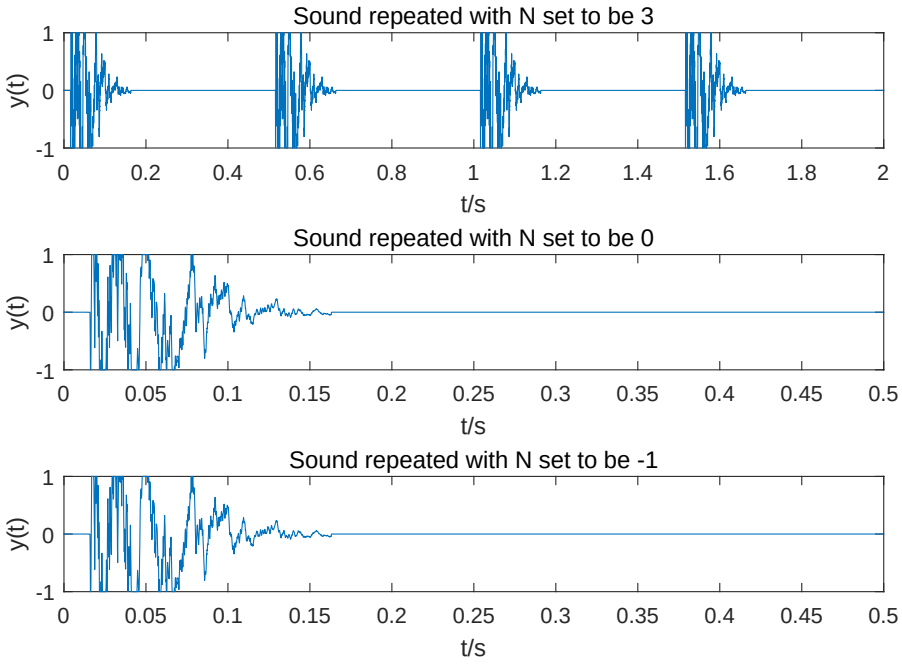


Figure 13: Repeat for $N = 3, 0, -1$ time(s).

Exercise 7: Please examine 'delay.m'. The 'delay' function's effect depends on the argument given by the users. If the parameter passed in, 'delay', is a positive value, it will delay the whole audio for *delay* seconds;

on the contrary, the audio will be advanced and the prior sound of *delay* seconds will be cut. More intuitive presentation is in Figure **??**.

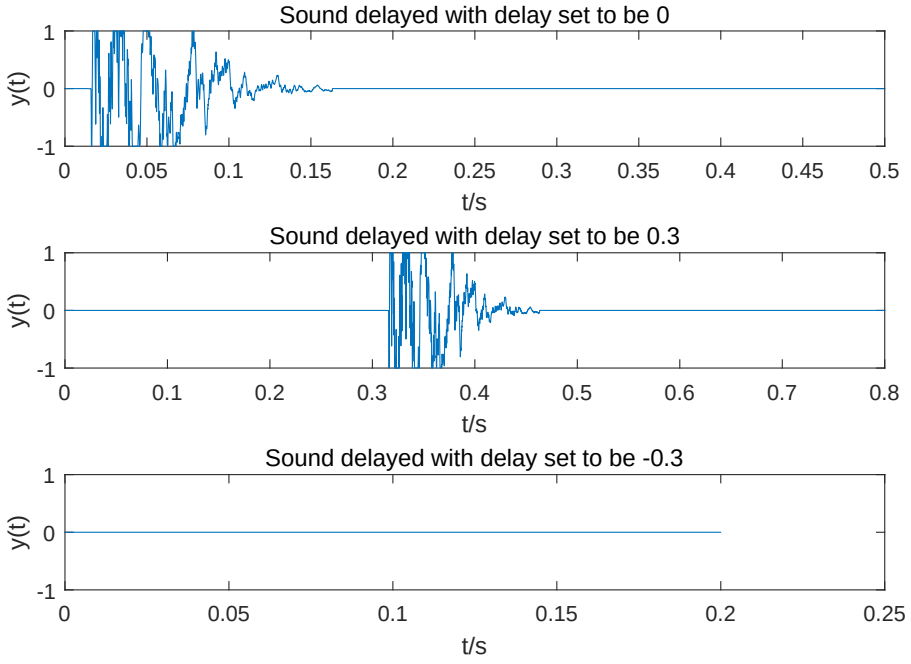


Figure 14: The original and delayed versions. The first diagram is the original wave. If delayed by a negative amount, some former samples will disappear.

Exercise 8: Please examine 'mix.m'. To mix two independent sounds need take into consideration whether the mixed amplitude is beyond -1 and 1, what to do if so and if two audios are not of the same length. The specific solution is in the code. For the first two original sounds and mixed one are shown in Figure **??**.

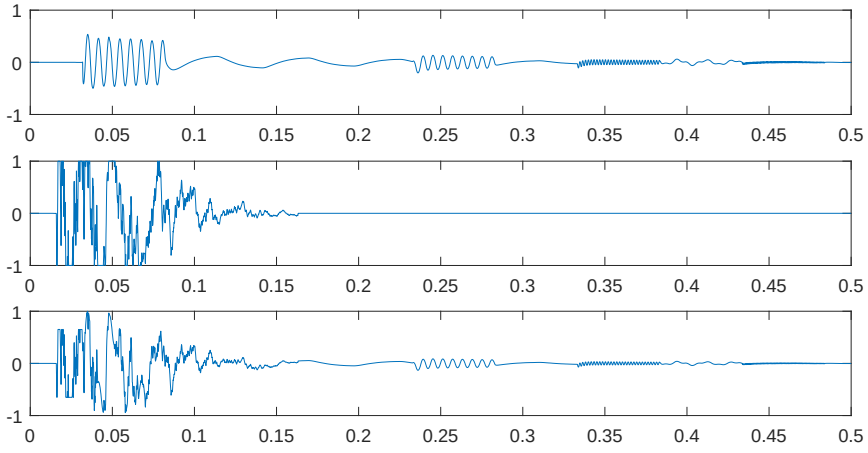


Figure 15: After aligning and reducing the mixed samples.

Exercise 9: I adapt a piece of music named 'Bad Apple!!' in Exersice 9 in a relative independent script named

'knzhou.m'. Run it and the audio 'Bad Apple!!' will be played automatically. The .wav file was written in advance to 'Bad Apple!!.wav' under the current directory.

To construct a music, I must define three fundamental elements: sampling rate, samples per count, windowing function (ADSR envelope), they are the dominant hyper-parameters. Based on that, a series of parameters can be determined, like the duration and waveform of each count. The notes in different range have different frequencies, which are determined by the physical vibrations. Besides notes, I add rests and noise which are also the components of my music. The illustration of ADSR can be visualized as Figure **??** displays. I design it and try to simulate the timbre of a piano.

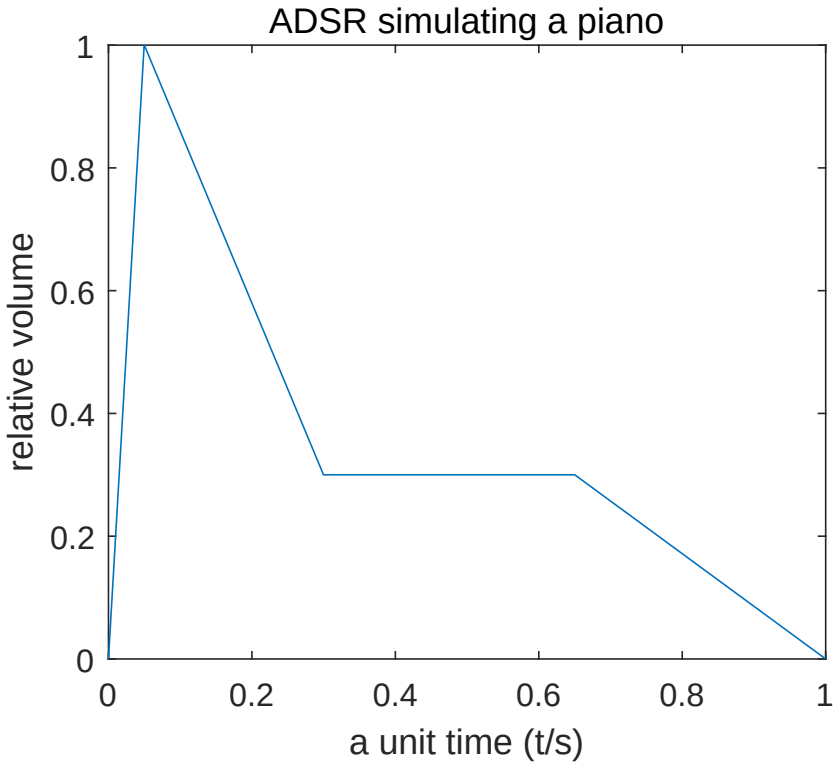


Figure 16: ADSR used in the music.

This melody has two parts: treble and basso, stored in the overall cell *bars*, and three sections: one prelude, two periods, stored in three tensors individually. Firstly I employ cells to store the bars, and then concatenate them to corresponding tensors mentioned above.

At the meantime, I utilize prior functions created by following the instruction of the questions like 'fade', 'repeat', 'delay', 'timescale', which make my music more vivid and protean. Finally I combine all three tensors into one melody and play it.

5 Contribution

Contributor: Zhou Kaining.