Signals and Systems

Computer Assignment 2

Sampling, Nyquist Boundary, and Playback Speed Manipulation

Authors: Armin Maddah; Alireza Tavazo; Amirreza Nadi Chaghadari; Amirabbas Ghadiri

Release Date: 1404/03/04

Due Date: 1404/03/16

Table of Contents

Overview

This project is mainly about sound processing, and has three major parts, as can be seen in the table of contents:

- Introduction to MATLAB's sound I/O functions
- Analyzing Nyquist boundary
- **Manipulating playback speed**

The first part helps you get started with MATLAB. It introduces the functions you could use to import your sound track into a local variable, playing it, and also writing it back to your hard drive.

In the second part, you will reduce the sampling rate of the audio. As you have already learned during this course, there is a limit to how low the sampling frequency can go, and it is called the Nyquist boundary. You will confirm that as the sampling frequency drops bellow a certain threshold, the sound track loses its quality significantly. You will also compare the Fourier transform of the down-sampled signal[1] with that of the original one, in order to confirm what you've learned throughout this course with a real-world example.

The third part will be the challenging one. You will use different methods to adjust the audio's playback speed, and analyze the effect they leave on the signal. You will be asked to research and study a very important concept called **Overlap and Add**, and to implement and test it on your sound track. But, you will try some simpler methods to manipulate the playback speed before using Overlap and Add method. Doing so helps you realize their problems and clears the advantages of this fairly simple method in your minds.

Please read the description carefully, before trying to implement it.

Good Luck!

[1] The resulting signal after reducing the sampling rate

1- Introduction to MATLAB

1-1- Introduction, Downloading, and Installing

The paragraph bellow is quoted from MATLAB's official website:

"MATLAB is a programming and numeric computing platform used by millions of engineers and scientists to analyze data, develop algorithms, and create models."

As mentioned above, MATLAB is an extremely powerful platform for programming and numerical calculations. Its name, MATLAB, stands for Matrix Laboratory[2]. The reason is that in MATLAB, all local variables are matrixes[3].

As its name suggests, MATLAB is greatly optimized to do matrix calculations. This means you should try to present your calculations in the form of matrix operations as much as possible.

MATLAB provides a free version, allowing 20 hours per month of free use and access to ten of its commonly used products. You could access this online version through this hyperlink[4]. For those of you who are unable or unwilling to use the online version, you could use this link[5] to download use this tool. Unfortunately, there is currently no way you could use a licensed version of MATLAB in our country. Therefore, you need to crack the software before using it. Please pay attention to the instruction of the introduced website while doing so. The recommended version for you to install is the **2024a** version, but your free to install any of the other versions, as long as they suit your system.

[2] Many mistake the name for Mathematics Laboratory!
[3] Tensors, in general form
[4] https://www.mathworks.com/products/matlab-online.html
[5] https://soft98.ir/software/engineering/1291-%D8%AF%D8%A7%D9%86%D9%84%D9%88%E2%80%8C%D8%AF%E2%80%8C-%D9%85%D8%AA%D9%84%D8%A8-%DA%86%D9%87%D8%A7%D8%B1-1e.html

1-2- Importing, Playing, and Saving Sound Track

In this project, you need a sound track to work with. **The sound track you use is expected to have these conditions:**

1. **It has to be your own voice.**
2. **You must introduce yourself and address your student ID in it.**
3. **The duration must exceed 10 seconds, but stay below 20 seconds.**

you are free to say whatever you want in the voice track, as long as it does not break any laws or morals and passes the conditions above.

1-2-1. Importing the Track

Use MATLAB's "**audioread**" function to import your sound track into MATLAB's workspace and save it into a local variable[6]. You can find the proper syntax of this function on the web, along with the audio formats it supports. Please note that the function returns two things: the audio's samples and its sampling rate. **Please make sure to save the sampling frequency it gives in another local variable, for it is needed for the rest of the assignment**.

Once you use this function, check out your workspace to find the variable. What is the dimension of the matrix? Use MATLAB's "**size**" function to see its dimensions and their sizes.

The sound you just loaded has the data for both channels, which makes it a stereo sound. In order to make the rest of your progress easier, **use the code bellow to turn it into a mono sound:**

```
S = S(:,1);
```
[7]

1-2-2. Playing Audio in MATLAB

Use MATLAB's "**sound**" function to play the audio you imported in your workspace. Please make sure to use the same sampling rate the original audio file has, which is provided to you as one of the outputs of the **audioread** function[8].

Please note that you always have to normalize your audio before using this function, since it assumes the values given to it are in range of -1 to 1. To do the normalization, you could use the code bellow, or any other method you know.

```
S = S/max(abs(S));
```

[6] You're free to name the local variable anything you want, as long as the syntax allows it. Just make sure not to save it in MATLAB's default **ans** variable, which occurs if you call the function and not assign it to anything. Visit https://www.mathworks.com/help/matlab/ref/ans.html for more explanation if you needed.

[7] Replace S with the name of the variable you used for importing your sound track

[8] You were asked to save it in a local variable in the previous part.

1-2-3. Writing Back to Hard Drive

Use MATLAB's "**audiowrite**" to write your track back to a file. Please make sure to use the same sampling rate the original audio file has, which is provided to you as one of the outputs of the **audioread** function.

Please note that you always have to normalize your audio before using this function, since it assumes the values given to it are in range of -1 to 1. To do the normalization, you could use the code bellow, or any other method you know.

```
S = S/max(abs(S));
```

Use your own audio player to listen to the file you just saved. Does it match your original file?

2- Down-Sampling and Nyquist Boundary Analysis

For this part, you need to down-sample the signal. Since you do not have access to the ADC[9] of your recording device, you must somehow handle it in the code!

To do that, you need to drop some of the samples. Search the web for the appropriate syntax and avoid using loops for this part. you could use the syntax below:

```
n = 5000;
m = Fs;
K = n/m;
Fs_new = Fs_Old*K;
new_signal = S(1:1/K:end);
```

This code starts from the first sample in S, and with 1/K steps, saves those samples in new_signal variable. If 1/K is not integer, it will automatically group the samples and do what we need it to do. For example, if 1/K is 2.5, it includes 5 3 samples and drops 2.

Let's say you want to drop the sampling rate to $\frac{m}{n}; n \geq m$ of its original. You would have to take these steps:

1. Delete $\frac{n-m}{n}$ of the samples[10], or use proper indexing to copy $\frac{n}{m}$ of the samples into a new variable.
2. Play the results using MATLAB's **sound** function, with $F_{s_{new}} = \frac{n}{m} \times F_{s_{old}}$, where $F_{s_{old}}$ is the sampling rate of the original file you were supposed to save.

Based on the sampling rate of your original file[11], find the appropriate $\frac{n}{m}$ ratio to get the frequencies in the table below, and take the steps explained above to generate a down-sampled track.

Frequency $[KHz]$	40	20	10	5	2	1

[9] Analog to Digital Convertor
[10] The values inside the vector in which you imported your audio file are called samples
[11] It is normally $48000^{[Hz]} = 48^{[KHz]}$

3- Manipulating Playback Speed

In this part, the goal is to manipulate the playback speed of the sound track. You will be asked to use different approaches to reach this goal. First, you will try to fasten the sound and then, you will try to slow it down. The methods you will be using are:

1. Manipulating playback frequency (F_s in **sound** function)
2. Down-sampling without changing F_s
3. Frame dropping
4. Overlap and Add Method

3-1- Manipulating Playback Frequency

Use MATLAB's **sound** function to play the original sound. As you already know, this function takes a second argument: F_s. This is the playback frequency. Pass $2 \times F_{s_{org}}$ as to the function this argument, where $F_{s_{org}}$ is the sampling rate of the original sound track. Describe the results. How did this method affect the signal? Briefly explain why.

3-2- Down-Sampling[12] Without Changing F_s

You have already down-sampled the signal in the previous part. But back then, you also changed F_s. This could be described as taking less samples, but waiting more for each sample as well. Therefore, the audio's total duration and consequently, playback speed were conserved. This part is different: you are to drop the samples without changing F_s.

Using the method given to you is the second part, drop $\frac{1}{2}$ of the samples ($\frac{m}{n} = \frac{1}{2}$). Doing so will drop the number of samples by 50%, reducing half of the total duration and thus, increasing the playback speed by a factor of 2. Now use MATLAB's **sound** function to play this new signal. **Please note that this time, $F_s = F_{s_{org}}$.** Describe the results. How did this method affect the signal? Using what you've learned about Fourier transform and frequency-domain analysis, explain the effect.

[12] Also know as **sample dropping**

3-3- Frame Dropping

In order to understand this method, we need to define what a frame is. A **Frame**, is a group of adjacent samples, on which we do all of our processes. When we use frames, we break the signal into smaller groups and name them frames. These frames are now the smallest signal unit we will access for our processes. Meaning that we use to be able to manipulate each and every sample individually, while now we will process all of the samples inside a single frame together.

This method is a lot like that of step 3-2. The only difference is that this time, the smallest signal unit we have is the frames. So, in order to speed up the playback by a factor of 2, we need to drop half the samples. Follow the steps bellow for this part:

1. Separate the signal into frames with size $N = 2000$. **Please note that the frames must have no overlap**. **Also, you are now allowed to use a for loop for this separation[13].**
2. Drop one out of every two adjacent frames. (for example, drop frames 2nd, 4th, 6th, …)
3. Put frames back together and rebuild the signal
4. Play it using MATLAB's **sound** function to play it, with the same sampling frequency as the original track

Describe the quality of the output. Is it good enough?

Repeat the three steps above for $N = 1000$ to $N = 10000$, with step size of 1000. Use a for loop to generate the signals, and instead of using **sound** function to play them, use **audiowrite** function to save write them into separate files. You could use the syntax bellow to name your files:

```
audiowrite(sprintf("Audio_For_N=%d", N), Frame_Dropped_Signal, Fs)
```

The **sprintf** function works similar to **printf** function of **C** programming language. Except that it does not print the results. Instead, it returns the value in string format and can be passed to other functions as their input. **Please note that you do not have to use the same naming format as this one. This is only a recommendation.**

Compare the results and report the one you think is the least harmed.

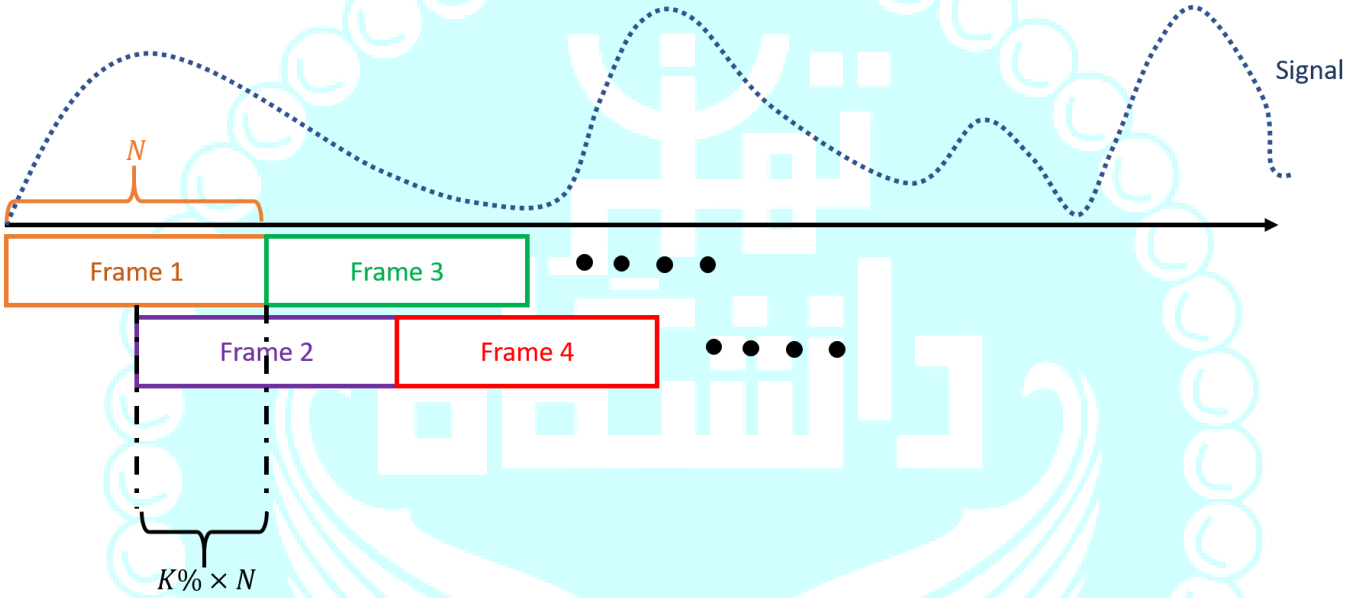
[13] Hint: Search the web for MATLAB's **reshape** function!

3-4- Overlap and Add Method

The previous method was not effective because removing frames that have no overlap with each other, causes the voice to lose its continuity. In order to solve this problem, we will **Overlap the Frames**, **Multiply Them by a Proper Window**, and **Add Them Together by a Proper Shift**. These concepts will be explained thoroughly.

3-4-1. Overlapping the Frames

In order to prevent the discontinuity, we could choose the frames in a way that they have overlap with each other. The image bellow will enlighten the concept.



Where N is the frame length and K is the overlap ratio, represented in percentage. In this part, we assume $K = 50\%$. But, you will be asked to sweep it later on.

3-4-2. Windowing[14]

These frames are being added together. If we just put the samples in the frames and add them up, the result will differ from the original signal. We solve this problem using a method called windowing. A window is basically a frame made by samples of another function what solves this problem if we multiply it against our signal's frames.

Take Frame 1 and Frame 2 of the image above. If we multiply the frames to a specific window named $h(n)$ for example. Now let's analyze the result of $(Frame\ 1 \times h(n)) + (Frame\ 2 \times h(n))$ from $n = \frac{N}{2}$ to $n = N$. **Please note that the**

[14] Multiplying the frame against a specific window of the same size

window gets shifted when it is multiplied into a frame[15], meaning that its first sample always gets multiplied by each frame's first sample.

$$\big(Frame\ 1 \times h(n)\big) + \big(Frame\ 2 \times h(n)\big)\ for\ n \in \left[\frac{N}{2}, N\right] =$$

$$x(n) \times h(n) + x(n) \times h\left(n - \frac{N}{2}\right)$$

$$= x(n) \times \left(h(n) + h\left(n - \frac{N}{2}\right)\right)$$

If we desire to get the original signal from this summation, we have:

$$for\ n \in \left[\frac{N}{2}, N\right] : x(n) \times \left(h(n) + h\left(n - \frac{N}{2}\right)\right) = x(n)$$

$$\Rightarrow for\ n \in \left[\frac{N}{2}, N\right] : h(n) + h\left(n - \frac{N}{2}\right) = 1$$

There are many functions that pass this condition, including triangular function ($\lambda\left(n - \frac{N}{2}\right)$), but we also need the window to smooth the signal in its boundaries. The reason is that when we drop some of the frames to get faster playback, we will get discontinuities and a good window is a window that filters them for us.

The most well-known window is the famous **Hann** window, which is the samples of the signal below:

$$hann_N(t) = \sin^2\left(\frac{\pi t}{N}\right)$$

If we take N samples from this signal, we get:

$$hann_{N(n)} = \sin^2\left(\frac{\pi n}{N}\right)\ for\ n \in [0, N]$$

3-4-3. Adding with Proper Shift

In order to generate a signal from its frames, we need to shift each frame by $\frac{N}{2}$ and add it to the signal.

$$x(n) = \sum_{k=0}^{L} F_k(n - kN) \times h(n - kN)$$

Where L is the original signal's length and $F_k(n)$ and $h(n)$ are k[th] frame and the window, respectively. **Please note that both $\boldsymbol{F_k(n)}$ and $\boldsymbol{h(n)}$ are zero for $\boldsymbol{n \notin [0, N]}$.**

[15] See the equation below the paragraph

3-4-4. Implementation

Take the original sound track and generate your frames, with a length of $N = 2000$. You will be asked to sweep N later, so try to use variables instead of fixed values. Also, if the number of the samples is not dividable by N, insert the proper number of zeros to the signal's end to solve that problem

Generate the frames, with $K = 50\%$, meaning that each frame starts at the middle of the previous one. Save your frames in a 2-dimensional matrix, where each **column** represents one frame.

Generate the window by taking samples of the introduced $hann_N(t)$. Search the web for the syntax if you need to.

Plot the window you generated.

Use MATLAB's **hann** function to generate another window, and plot it in the same figure as the window you generated through sampling $hann_{N(t)}$ and compare the outputs.

Shift your window's samples by $\frac{N}{2}$ to the right, and add the results together and plot the summation results. Please note that the dimensions of the vectors being added together must fit. Therefore, you need to insert $\frac{N}{2}$ zeros to the end of the original window and $\frac{N}{2}$ zeros to the beginning of the shifted one. You could use the code below:

```
window = ... % Do it yourself!
shifted_window = [zeros(1,N/2) ; window]
window2 = [window; zeros(1,N/2)]
added = window2 + shifted_window
```

What are the results? Does it match your expectations? Explain why using the theoretical insights of part 3-4-2.

Multiply each frame into the window. **Please note that both window and frames are $1 \times N$ matrixes, or vertical N-sized vectors.**

Remove one out of every two adjacent frames.

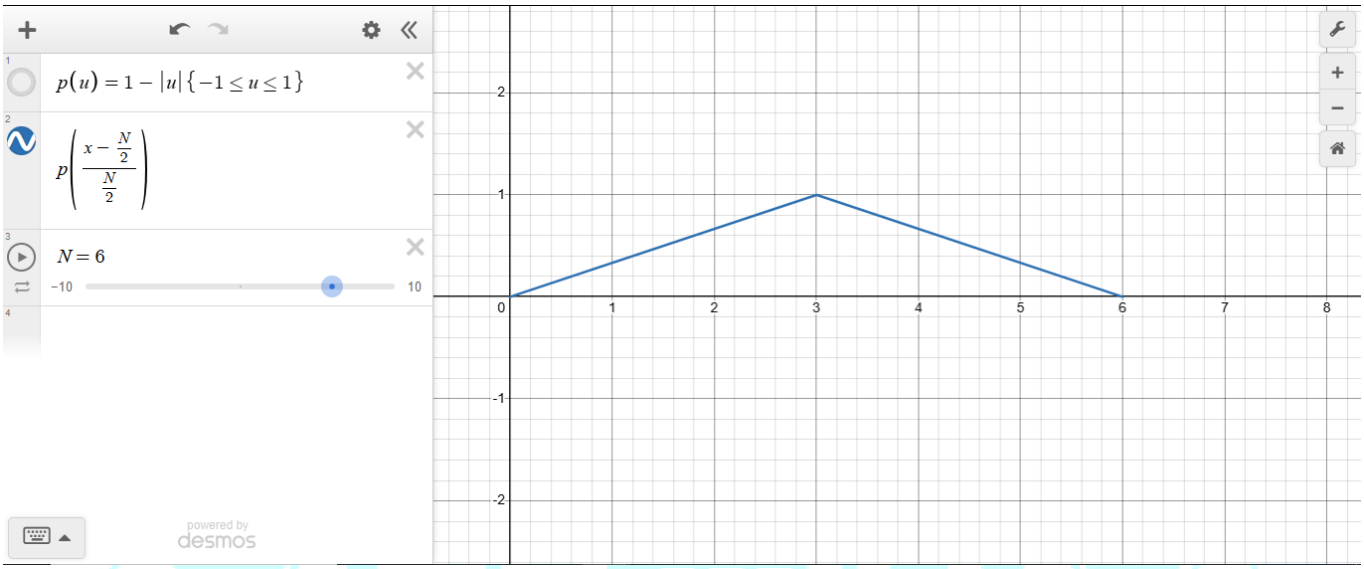
Give the proper shift to the remaining frames and generate the new signal. Please note that the length of this signal must be about half that of the original one.

Use MATLAB's **sound** function to play the results, with the same F_s as the original signal. How does it sound? Write the results into a file using **audiowrite** function.

Repeat the steps above for $N = 5000$ and $N = 7000$. Which one do you think is the best?

Based on the results of the previous steps, reason that why too large or too small values of N are not good.

For $N = 5000$, change the window from $hann_{N(n)} = \sin^2\left(\frac{\pi n}{N}\right)\ for\ n \in [0, N]$ to $\lambda_N(n) = \lambda\left(\frac{n-\frac{N}{2}}{\frac{N}{2}}\right)$. And repeat the steps. This window looks like this in continues form:



…THE END…

Deliverables

- A sound track, passing the conditions explained
- The MATLAB code for each and every part
- A report that includes explanation of your code, the theory behind each part, and the answer to the questions represented in this computer assignment.

Upload Format

Upload only one zip/rar file including the codes and the report, named according to the template below:

SS_CA2_LastName_SID.zip

OR

SS_CA2_LastName_SID.rar

PLEASE NOTE THAT NOT FOLOWING THIS NAMING TEMPLATE WILL RESULT IN PENALTY!