BME 252 Project

Dr. Alaaeldin Elhady Ahmed

Spring 2023

University of Waterloo

This project utilizes your understanding of spectral analysis, filter design, and sampling theory. It is designed to be both fun and practical. Part A of this project trains you to practical uses of your new skills with a biomedical application. It is worth 25% of the project mark (5% of your total mark for the course). Part B is worth 75% of the project mark (15% of your total mark for the course), and it builds on the skills you learnt in Part A. It is a fun application that allows you to create a robot-voice synthesizer. Your words will sound like they are spoken by a retro sci-fi robot. The same technique is implemented in biomedical applications like cochlear implants, making the project both practical and fun.

All deliverables are listed in **bold**.

Part A: The EMG Filter

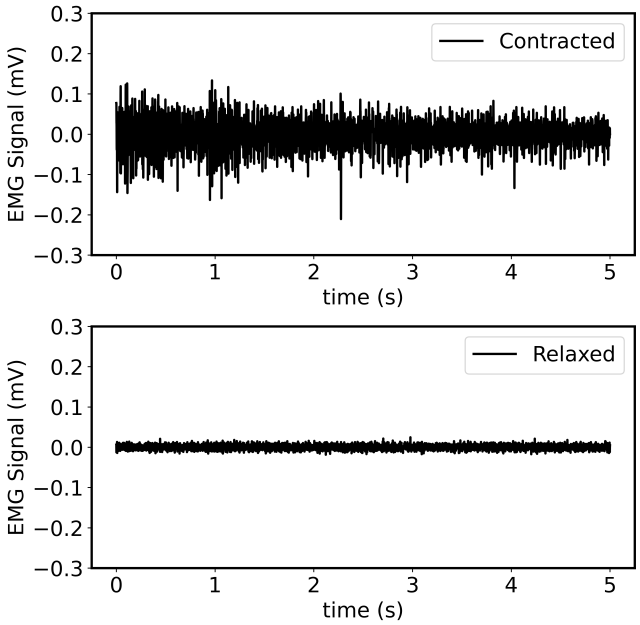
The purpose of this part is to condition the signal of a typical EMG so that it can be used to make a decision on whether the muscle was contracted or not. Typical problems that affect EMG signals are interference and noise. Therefore, we must filter both out, and then design a metric to determine the status of the muscle.

A.1 Contracted/Relaxed Metric

An EMG signal could be used to detect if a muscle is contracted or not. One way is to evaluate the root mean square value (RMS) of the EMG signal. The RMS of a discrete signal is calculated using the following formula:

$$RMS = \frac{1}{N}\sqrt{\sum_{n=0}^{N}(x[n])^2}$$

For a pure-tone sine wave, the average of the signal is zero. However, its RMS is approximately 0.707 times its amplitude. The RMS of a signal is a measure of the power (actually, it's the square root of the power) of the signal.



The EMG of a contracted muscle would have a larger RMS than a relaxed one. Therefore, we can check if the RMS crosses a certain threshold. This can thus act as a metric to help us decide if the EMG belongs to a contracted muscle or not.

A.2 Datasets

- Given are two EMG datasets for a contracted and a relaxed muscle
- The file EMG_Datasets.csv is a spreadsheet with three columns
  - The time in seconds
  - The EMG of a relaxed muscle in mV
  - The EMG of a contracted muscle in mV
- The source of this dataset was obtained from Physionet

A.3 Power Line Interference (10%)

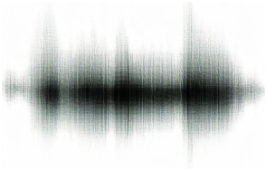
One common issue is the interference from the 60Hz power line. This signal 'creeps' into most biomedical signals. If it is loud enough, it can elevate the RMS. That is, if the interference is strong enough, it will contaminate and overlap with the EMG signal and thus increase the RMS of the signals to a level that triggers the contraction metric. Therefore we must filter it out.

- **Design** a filter (type, order, cutoff) to remove the 60 Hz interference signal without disrupting the original EMG (as much as possible)
- **Justify** your design choices

The filter type includes its Frequency Response Category (low pass, high pass, ...) and Filter Design Methodology (Chebyshev, Butterworth, ...). You must choose the appropriate type, order, and cutoff frequencies. **Write the transfer function** of your filter(s). You can do trial and error until you are satisfied with the results.

A.4 Background Noise (10%)

Another typical issue is noise from the environment, electronics, or the human body. A typical EMG signal exists in the range of 0.1 Hz to 450 Hz. Therefore, any noise that exists outside this range should also be filtered out.



- **Design** a filter (type, order, cutoff) to remove the noise outside this band, without disrupting the original EMG (as much as possible)
- **Justify** your design choices

Similarly, you must choose the appropriate type, order, and cutoff frequencies. **Write the transfer function** of your filter(s). You can do trial and error until you are satisfied with the results.

A.5 Procedure (5%)

Design filters to deal with the interference and the noise, as detailed above. **Highlight** your findings and **justify** your choices. **State** the type, order, cutoff(s), and transfer function $|B(j\omega)|^2$ of all used filters. For each signal/dataset:

- **Apply** the *same* set of filters on both datasets
- **Plot** the signals in time domain before and after the filters
- **Plot** the spectrum before and after the filters
- **Find** the RMS of each signal before and after the filters (in time-domain)

Answer this question: Did the 60 Hz corrupt our ability to decide if the muscle is contracted or not? Use the plots, and RMS values you calculated in answering your question.

Choose a suitable threshold to be used to decide if the muscle is contracted or not. A successful project submission should have a threshold that can decide if the muscle is contracted or not when looking at the RMS of the filtered EMG signal(s).

Part B: Robo-Voice Synthesizer

The purpose of this project is to create your own Robotic Voice Synthesizer. This tool should transform human voices into robot-like tones. It focuses on developing techniques and algorithms to replicate the characteristics of robotic speech.

B.1 Dataset

This time you should provide the dataset, from your own voice. In case you are unable to do so, contact me (the instructor) directly and I will share a recording you can use. Only do so, if you have a valid reason why you cannot use your voice.

- **Record** your voice saying your favourite quote or movie line
- The duration must be between 5 and 10 seconds
- You can use your phone, laptop, or dedicated mic,...
- Save your format as .wav (or convert it using any free online tool)
- In your report, **state** the quote you will use

B.2 Preprocessing (5%)

- **Load** the stream from the .wav into your program
- **Identify** the sampling rate (1/sample time)
- If the sampling rate is $>$ 16KHz, **down-sample** to 16kHz
- If it is a stereo recording (2 channels), keep only one of them and discard the other
- **Plot** the time stream and **include** the original .wav file in your report submission

Down-sampling means sampling discrete-time data at a lower rate. It is the process of reducing the sampling rate of a signal, resulting in fewer samples over a given time period, which can help conserve memory or reduce computational requirements while sacrificing some signal resolution. It involves discarding samples from a signal by selecting every Nth sample.

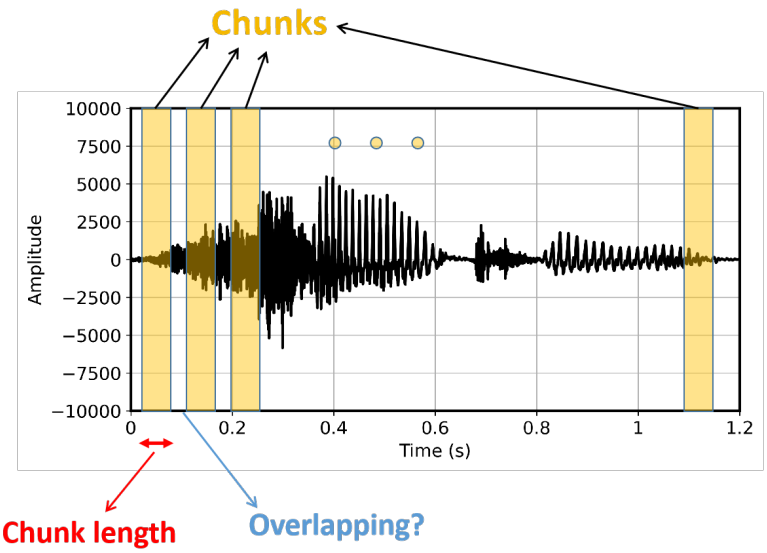
Example: $y[n] = x[2n]$ discards every other sample and thus gives an output that is a down-sampled version of $x[n]$. The ratio of the new to the old sampling rate is 1 to 2, that is the sampling rate is halved.

B.3 Procedure

Time-Segmentation (20%)

The first step is to segment the data in the time domain into smaller chunks. Each chunk must be smaller than the time it takes you to pronounce a single letter. Typically the duration of these chunks should be in milliseconds. A longer time chunk/segment will merge parts of each letter (Consonants, Vowels), and thus distort your words. A very short segment duration will increase your processing time and memory beyond reason.

- **Divide** the stream into successive short "chunks"
  - The number and length/duration of the "chunks" are an open design question
  - Try changing them and **justify** your final choice
  - Hint: The chunks should have equal duration, and it should be measured in milliseconds
  - Should the chunks be overlapping? Should they be consecutive, or would you leave gaps? What would happen if it is or is not? Try and see. **Report** on the effect of each case (overlapping or not, consecutive or with gaps)



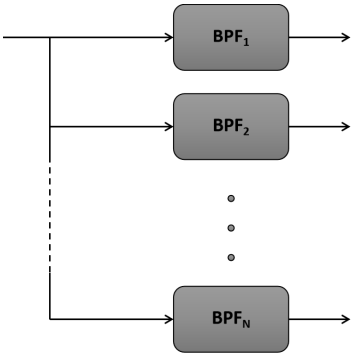
Freqeuncy-Domain Analysis (20%)

Freqeuncy-Domain (spectral) analysis of each segment reveals the dominant frequencies within that segment. For example, if the segment happened to contain the letter 's', you would find components near the 1 kHz and 5 kHz. That means if a robot were to say 's', it would create (synthesis) sine waves at 1 kHz and 5 kHz.

First, let us analyze the Freqeuncy-Domain. Instead of doing an FFt and looking at it each time, let's automate the process. We will segment the Freqeuncy-Domain into 'bands' using band-pass filters.

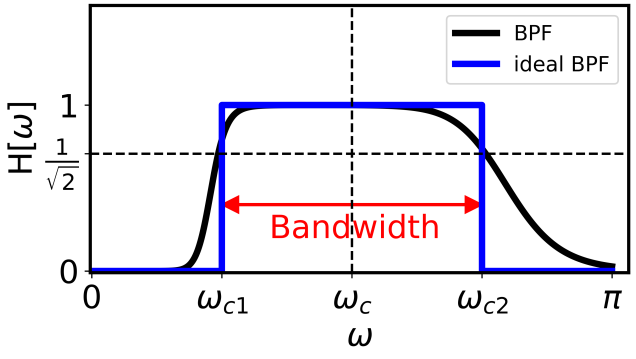
On each "chunk":

- **Apply** a bank/array of band-pass filters (BPFs)
- Open questions here are the center frequency, filter type and order, and bandwidth of each filter as well as the number of filters (N) in the bank
- Also, are the segregated frequency bands of the filters overlapping in the frequency domain? Do they have gaps? What would you think is best? **Justify** your choice.
- Try varying your parameters and **justify** your final choice



Here, we create parallel BPFs to segment the frequency domain into N-Bands. Each band should be narrow enough to avoid merging important spectral features, but not too narrow it overloads your processor and memory. *The bandwidth of each band should be in the order of Hz or 10s of Hz.*

A BPF has a center frequency and a bandwidth. The filter band should be doneusing a loop. Do not construct each filter manually. The loop should go over the center frequency of each BPF, and they can all have the same bandwidth. Don't copy-paste the code a million times. Use loops/tables/...

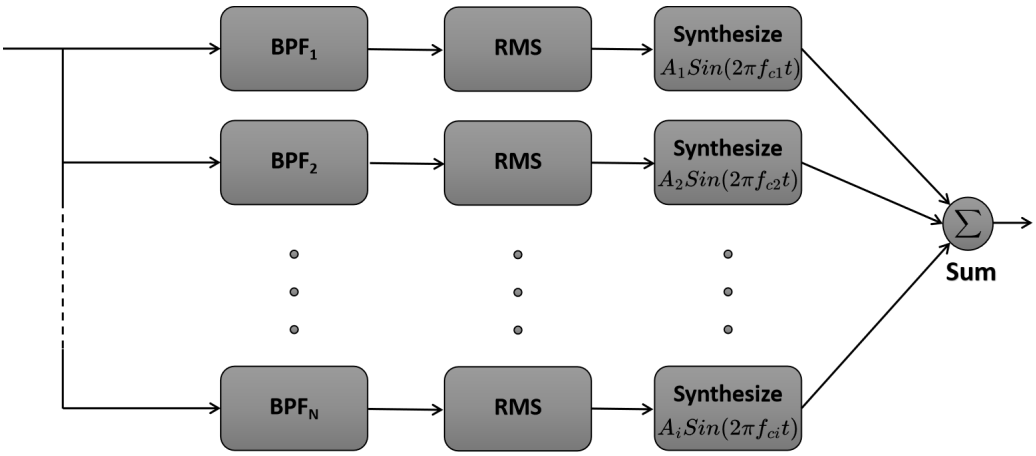


Right now, the output of each BPF should contain only the part of the original wave that is within its bandwidth. If we go back to our letter 's' example, most of the BPF outputs will be very small (zero or noise), except the BPFs centered around the 1 kHz and 5 kHz. Therefore, if get the RMS of the output of each BPF, they will all be approximately zero, except the BPFs centered around the 1 kHz and 5 kHz.

Synthesis (15%)

Now that we have the frequency domain segmented into bands, let's calculate the RMS of each band.

- **Get** the RMS of each output = A_i,
  where i is the individual filter number in the filter bank
- **Synthesize** the sine-waves:
  $$A_i Sin(2\pi f_{ci} t)$$
  where f_{ci} is the center frequency of band i. Synthesis means creating a signal by building it from its components.
- **Add up** (superimpose) the synthesized waves from all the bands



In this step, you calculate the RMS A_i of each band (band number i), then use it as an amplitude for a synthetic sine wave, hence the name synthesizer.

First, we construct our custom time domain as an array with the duration of your current chunk and use the sampling rate 16 kHz. Alternatively, you can use the time-domain potion of your original chunk. Then, you build an array using the equation $A_i Sin(2\pi f_{ci} t)$. This array looks like a sine wave with the amplitude A_i and it *should have the frequency of the current BPF band center frequency*.

This should also be done in a loop. It could also be done in the same loop you used to apply the BPFs. The last step here is to add or Superimpose the waves from all the bands into a single time stream.

Finally, move on to the next "chunk", or segment, in your time domain data. This (again) should be done in a loop. This is a different loop. This loop should go over your time chunks. The output of each loop should be concatenated (joined in order). This is the opposite of the time-segmentation step.

B.4 Evaluate your work (15%)

- Listen to the output stream
- Can you hear your sentence intelligibly?
- This is an important part of your evaluation (**The sentence must be intelligibly recognized**)
  - If not, go back and re-adjust your choices
  - If you absolutely can't, discuss what you think are the reasons
- **Save** it as a .wav file and make sure it works
- **Plot** the full output stream and **include** the output .wav file in your submission
- **Submit** your code file
- *When writing code, use comments to explain what you are doing on every single line of code!*
- In your report, **explain** your steps and **justify** ALL your design choices
- Feel free to add other effects, like echo or reverb. Explain your steps.

Coding hints

Below are the relevant functions for the various programming languages, with links to the 'help' function. You do NOT have to use these, and you can design and apply other filters if you would like.

- MATLAB has a butter(...) function
- Python has the scipy.signal library with a signal.butter(...) function
- Mathemtica has a ButterworthFilterModel[...] function

Project Submission

- **Highlight** your findings and **justify** your choices
- **State** the **type**, **order**, **cutoff**(s), and **transfer function** $|B(j\omega)|^2$ of all used filters
- This is an individual project (No group submissions)
- You can only use MATLAB, Python, or Mathematica (click for help)
- **Upload** one zip file to the Dropbox folder on Learn
- The zip file should have:
  - A single PDF file
  - Your input file (as .wav file type only), for part B of the project.
  - Your output files (as .csv for part A and .wav file for part B types only)
  - Your code file (as .m, .nb, .py, or .ipynb only)
- Do not waste time on introductions/conclusions. A (very) short **discussion** section at the end of each part is all that is required (Discuss your findings, limitations, ...)
- Hint: Start with part A first

Good Luck!

I hope you have fun doing this project