Implementing time-frequency analysis using custom spectrogram program in MATLAB with applications to radar and biomedical signals processing

George Lin

August 7, 2023

1. Introduction

See project repository on github

Real world data such as speech, biomedical data, or radar data often has spectral characteristics which change over time. For this reason, time-frequency representations are often used, conveying both information in the time and frequency domains. This combined approach of utilizing joint time-frequency representations allow for richer representations of data that cannot be obtained through individual time or frequency domain analysis [2].

Particularly in training machine learning models, taking high dimensional time series data and transforming it into a lower dimensional time-frequency representation helps to reduce sparsity in the data, allowing for models to learn underlying patterns instead of noise. This leads to improved computational complexity, improved feature extraction, and allows for models to avoid over fitting to data [8]. Often times, time-frequency representations are also more interpretable as models are trained on smaller, and thus generally simpler, feature sets. Furthermore, time-frequency representations are more comprehensible to those with expert domain knowledge, leading to more explainable ML models [15].

Spectrograms provide a useful way to visualize time-frequency representations, primarily when the changes in frequency of a signal over time are relevant. They are often used in music analysis, but also see use in speech recognition, radar signal processing, and medical imaging. For example, they are used for feature extraction in order to train machine learning models for anger and stress detection in speech, often times being preferred over other time-frequency representations such as wavelet transforms and Wigner–Ville distributions [9]. Obtaining DCT coefficients from spectrogram plots can also be effective in speech recognition, though performance is generally worse than using MFCC features [18]. Spectrograms also provide useful representations that sees use in various radar settings such as spectrum sensing [10] and multicomponent signal decomposition where algorithms can be used for signal retrieval based on the spectrogram [1]. Spectrograms are also see use in biomedical applications for analyzing EEG and ECG data [21], capturing more information and allowing for deeper insights to be drawn [16] compared to individual time or frequency domain analysis.

Digital signal processing lies at an intersection in terms of applications for my interests in machine learning and radar systems. This motivated me to learn more about the field, specifically focusing on utilizing Fourier analysis for drawing insights from data. Looking at the various uses of spectrograms specifically for time-frequency analysis led me to take on this project to learn more on underlying mathematics and various DSP techniques.

A high level programming language chosen for implementing this program as the focus was on implementing signal processing techniques rather than implementing linear algebra in a low level language. MATLAB was chosen over python as it provides specialized tools for signals processing that I can compare my implementation against as well as more documentation for the specific functions I was using. I also generally prefer developing programs in MATLAB for numerical computations over Jupyter notebooks as it provides a better GUI for viewing variables matrix sizes.

This document delves into the mathematical principles behind spectrogram creation and provides a general overview of how this was implemented in MATLAB. Later, the results of my spectrogram are compared with the built in MATLAB spectrogram from the signal processing toolbox.
The main objectives of this work are to

1. Outline fundamental concepts pertaining to building a spectrogram, going over the STFT equation and how matrices are used to represent and manipulate relevant time, frequency, and amplitude data.

2. Outline the uses of window functions and overlapping for dealing spectral leakage resulting from computing the DTFT of a signal

3. Explain the implementation in MATLAB

4. Evaluate the results, specifically pertaining to noise and execution time.

2. Fundamental Concepts

Spectrograms are a way to visualize the STFT of a signal, with an alternative being a waterfall plot which can be used to give a sense of scale regarding relative amplitudes of frequency components. The following sections cover the principles behind how a spectrogram is created and some of the mathematical descriptions of Fourier analysis. First, the steps of creating a spectrogram are outlined, then specific details regarding design considerations are discussed, these being the use of windowing signals and overlapping windowed signals.

2.1 How to build a spectrogram

Spectrograms are created by splitting a time-varying signal into segments, segments are windowed, then an FFT is performed on each window to obtain frequency content. The STFT of a signal is by computing by summing the Fourier Transform on signal segment which is multiplied by a moving windowed segment $w[n-m]$.

$$STFT(x[n]) = \sum_{n=1}^{\infty} x[n]\omega[n-m]e^{-jwn} \quad (2.1)$$

The following image are the steps for how a short time Fourier transforms (STFTs) is created.

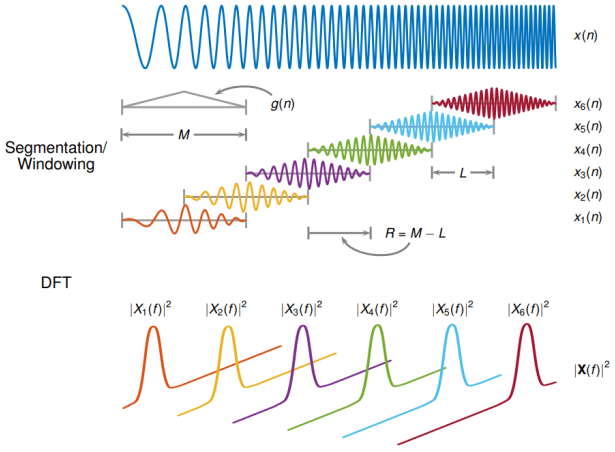

Figure 1: Process for creating a STFT [14]

In order to create a spectrogram, the DFTs in figure 1, which in are computed using a FFT, returns an n dimensional vector. This frequency content is represented as a coloured bar, with amplitudes corresponding to a color scale. This process repeats for each segment of $x[n]$, with these vertical strips of color being placed next to each other to create the full spectrogram.

For the purpose of this project, this data is most conveniently represented using matrices. As the window is moved across the entirety of the signal, 3 outputs are created.

$$\hat{f} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{bmatrix} \quad \hat{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_m \end{bmatrix} \quad \hat{A} = \begin{bmatrix} A_{11} & \cdots & A_{1m} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nm} \end{bmatrix} \quad (2.2)$$

Each column of $\hat{A}$ corresponds to the amplitudes of output of the m_{th} FFT performed and each row in $\hat{A}$ corresponds to the n_{th} frequency bin. The vector $\hat{f}$ increases by a constant Δf until it reaches the Nyquist frequency [11].

The number of frequency bins that are present will depend exclusively on the the size of the FFT, N. Due to symmetry in the FFT's output, generally the first half is taken. This makes the total number of frequency bins $N/2$. Using this result, the size of each frequency bin can be calculated as simply:

$$\Delta f = \frac{F_s}{N} \quad (2.3)$$

In the next two subsections, the justification for why windowing is used as well as why overlapping is used will be elaborated on.

2.2 Windowing functions

The DFT is the discrete counterpart to the continuous Fourier transform, which is defined over the entire real line from $(-\infty, \infty)$. When a DFT is applied to a segment of a signal, that segment is treated as being periodic, thus repeating infinitely in both the positive and negative directions.

When signal segment is not exactly periodic with the window, meaning the edges of the windows are different values, discontinuities are introduced as the signal is repeated due to the property previously discussed. These discontinuities are illustrated in figure two, where there's a sudden jump where the signal is repeated.

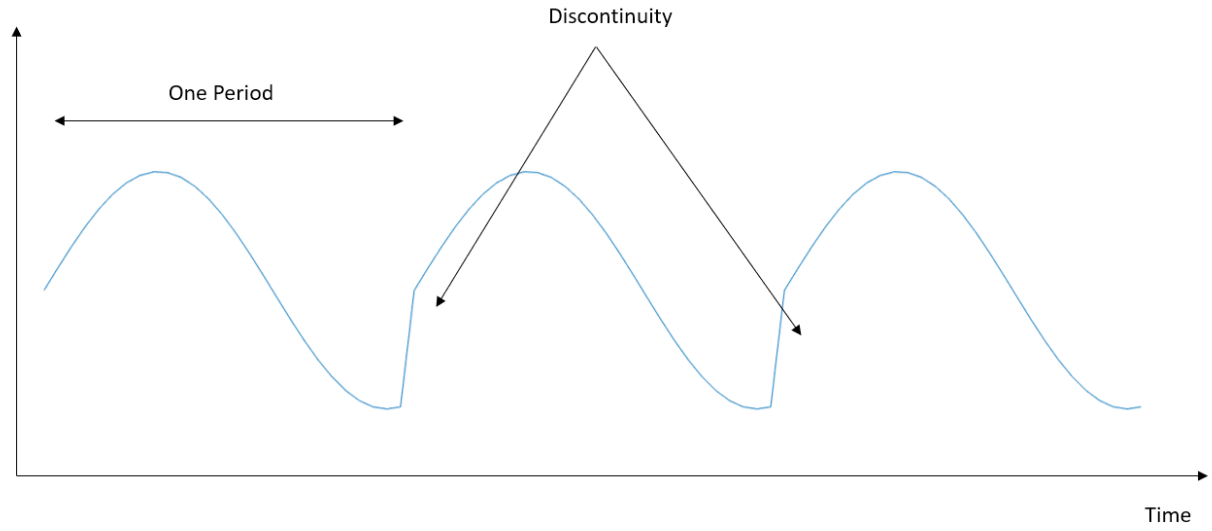

Figure 2: Discontinuities in signal end points

Because of these sharp discontinuities, the computed frequency spectrum will contain frequencies not present in the original signal. This is known as spectral leakage, leading to frequency content spreading into adjacent bins. For this project, this would ultimately result in a decreased frequency resolution in the spectrogram, lowering one's ability to pick up separate frequencies content, and thus the overall utility of the spectrogram.

To avoid this issue, a windowing function is used which will taper out the signal forcing both ends to zero, allowing spectral leakage to be reduced through eliminating discontinuities [7]. The figure below illustrates effect of spectral leakage and the importance of using window functions in order to reduce its effects.

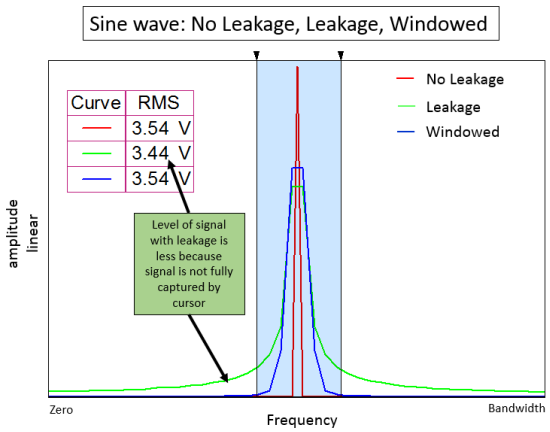

Figure 3: Comparison of frequency spectrums for a periodic sin wave [20]

When deciding on what window functions to use, the frequency response of each should be carefully considered depending on the specific signal processing application. The performance of a window function is evaluated in the frequency domain, with the main characteristics to consider being main lobe width, side lobe amplitudes, roll off.

Window functions with a narrow main lobe give higher frequency resolution, which is useful for when high frequency resolution is needed. Small side lobes reduce spectral leakage, which is especially important if there's strong interference near the frequency of interest, a window with a low maximum side lobe levels should be chosen. If interference is distant from the frequencies of interest, a window with high side lobe roll-off should be chosen [4].

For audio processing tasks, the Hann window is commonly used for its ability to provide high frequency resolution while having small sidelobes which helps with reducing spectral leakage [17]. For 95 % of cases, it's satisfactory, especially if the characteristics of the signal is unknown [4]. For this reason, this was the window function used in the spectrogram for testing purposes.

For a segment of size L, the coefficients of the a Hann window are given by [13]:

$$w(n) = 0.5\left(1 - cos\left(2\pi\frac{n}{N}\right)\right)\ 0 \leq n \leq N.$$

Where the $N = L - 1$.

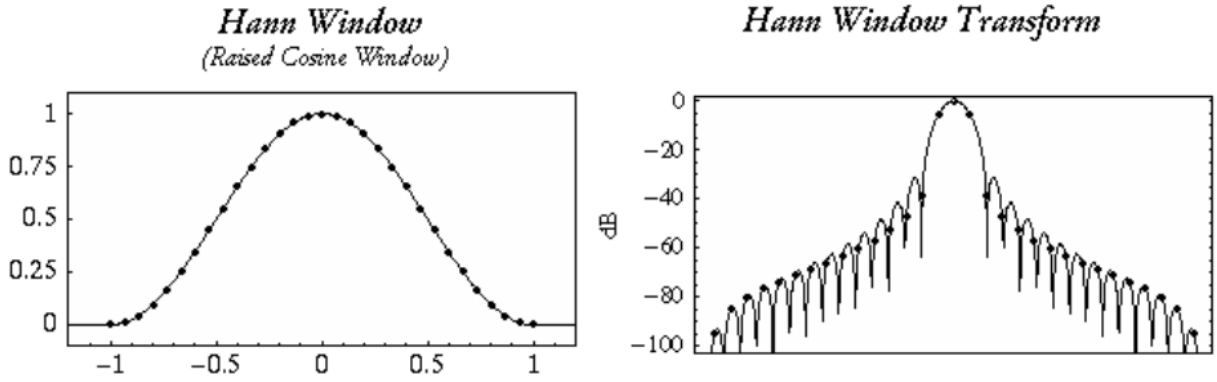

Figure 4: Hann Window and frequency response [3]

2.3 Overlapping

While windowing a signal comes with various drawbacks, most of which are outside the scope of this project, there is one consideration which is relevant, which is that windowing a signal will cause the frequency information at the tapered ends to be lost. This is one reason that spectral windows, which have symmetrical tapering, aren't used for analyzing impact and response signals [4].

To compensate for the frequency information lost at the end of a signal, signal segments are overlapped (see Fig. 1). Overlapping allows for the lost frequency content in one window to show up in the next, allowing for fuller recreation of the frequency spectrum of the signal. Furthermore, it increases temporal resolution by essentially allowing more DFTs to be performed. This also allows for transient events (which relies on high temporal resolution) to be detected. Reliable detection of transient events can be important in healthcare applications such as stroke monitoring. Specifically in systems using Transcranial doppler (TCD) ultrasound, inadequate overlap of FFT time windows resulted in more failed detections of embolic signals [12] which represent solid or gaseous particles in blood flow and are detected in a number of cardiovascular conditions [6].

3. Implementation

In this section, the steps involved in creating a spectrogram are outlined, First, the pseudocode for the program is introduced, then what the effects of each parameter of the spectrogram function is discussed. In section 3.3, the parameters in (2.2) are computed.

In order to create a spectrogram of a signal, the imagesc function in MATLAB was used as it allows 3 dimensional information to be plotted. An x-axis for time, a y-axis for frequency, and all scaled by a colour axis to represent the amplitude as a function of time and frequency. The imagesc function can take in 3 inputs: x, y, and C which correspond to the matrices $\hat{t}$, $\hat{f}$, $\hat{A}$ which are described in (2.2). The next sections will go into how the program works and how each individual section is calculated.

3.1 Pseudocode

First, sound data is loaded into MATLAB and the parameters of the STFT are configured. These are the size of the FFT windows to use, the type of windowing function, overlap size, and the sample rate of the audio file. In the following pseudocode, $x[n]$ corresponds to the n_{th} windowed signal and $w[n]$ corresponds to the window function. A_n corresponds to the n_{th} column of the amplitude matrix. Since the number of windows and the size of the FFT is known, this matrix can be initialized before the for loop to increase computation speed.

Algorithm 1 Spectrogram Program

```
Load sound data
Set STFT parameters
Compute number of windows, N
Compute windowing function, w[n]
Initialize Â

for N do
    sgn = x[n].*w[n] for the n_th window
    y = fft(sgn)
    A_n = |y|× window correction factor
end for

Create vector t̂
Create vector f̂
```

3.2 Computing the number of windows

A vector with N_x data points must be split into overlapped segments of length k with overlap p, where $\{N_x,\, k \in I\}$ and $\{p \in R \,|\, 0 \leq p < 1\}$.

To avoid any indexing issues, these vectors must be set up such for all potential values of p, n, and k, the total length of the overlapped segments is smaller or equal to the original vector.

A length of N overlapped segments is equal to k plus an additional $N-1$ non-overlapped segments. The length of a non-overlapped segments is $(1-p)k$. This gives the following formulation of the problem:

$$k + (N-1)(1-p)k \leq N_x$$

To solve for N - the number of windowed segments - the equation is rearranged to obtain

$$N = \left\lfloor \frac{N_x - k}{k(1-p)} + 1 \right\rfloor \tag{3.4}$$

where x is the total length of the signal, k is the size of the FFT, and the value p is the % overlap, expressed as a decimal.

3.3 Computing amplitude and frequency matrices

This following for loop will create a windowed segment of the original signal, x, compute the FFT on that windowed segment, and add it to the amplitude matrix after taking the 1st half of the FFT result. For a Hanning Window, 2 is used as the correction factor [19] due to the energy lost from windowing.

```
for n = 1:num_windows
    start_val = 1+floor((1-Overlap)*NFFT*(n-1));
    end_val = start_val + NFFT -1
    w_sg =  x(start_val:end_val) .* w;
    abs_fft = abs(fft(w_sg)) * 2;

    % Take first half of the FFT output
    amp(:,k) = abs_fft(1:NFFT/2);
end
```

The first two steps of this process are roughly outlined by Fig. 5, where a DFT is computed on segments of the window and the frequency-amplitude content is added to the matrix $\hat{A}$.

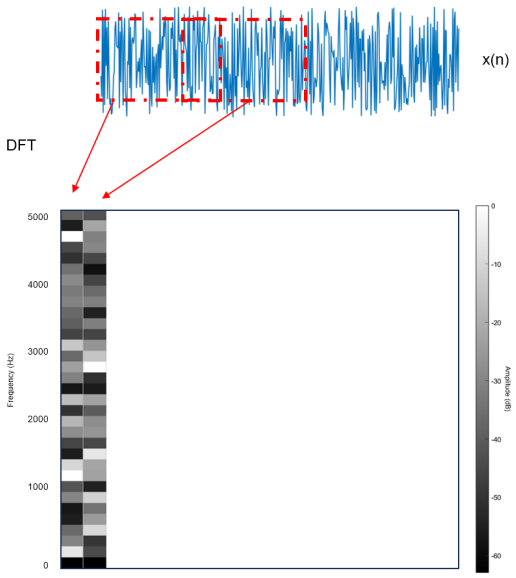

Figure 5: Caption

The frequency vector can be computed in MATLAB as

```
freq = 1:del_freq:nyquist_frequency;
```

where Δf is equal to the sample rate / size FFT. See (2.3).

$\hat{t}$ can be computed by simply creating a vector that increments from 0 to end time of the signal and width(time domain) of each frequency strip will be the total length of the signal in seconds divided by the number of windows, N.

```
end_time = length(x)/Fs;
del_time = end_time/N;
time = 0:del_time:end_time;
```

After computing these three matrices, they can be passed to imagesc to create the spectrogram. For the full program, see Appendix A.

4. Evaluating results

To evaluate the results of my code, my resulting spectrogram will be compared to the output using the built in function from the signal processing toolbox. The plots will be visually inspected to verify they produce the same results for the same parameters. Next, the noise and spectral leakage characteristics of my resultant spectrogram will be analyzed. Finally, the execution speeds will be compared to evaluate the computational efficiency of my algorithm.

4.1 Spectrograms evaluation

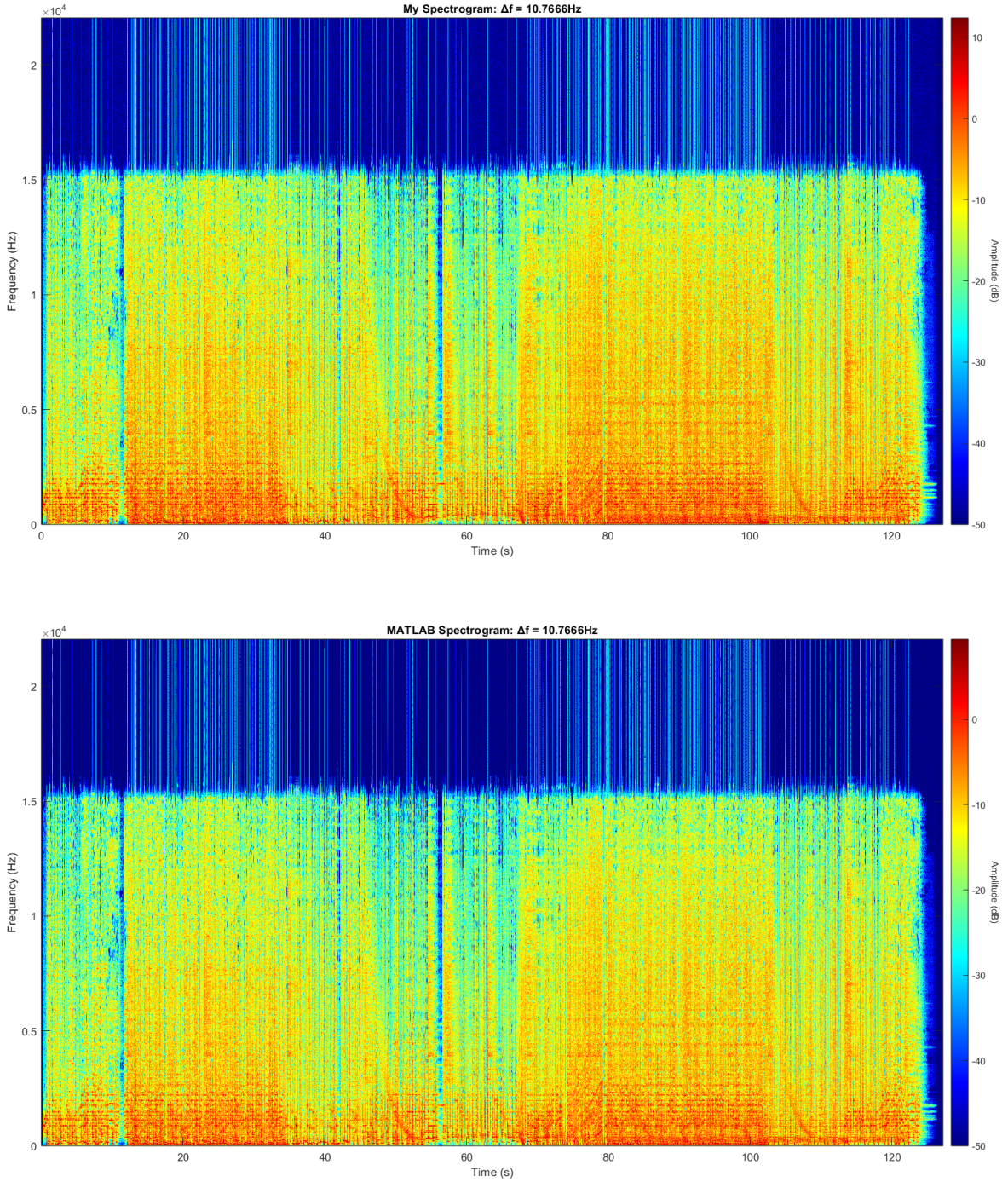

Figure 6: Spectrograms comparison for an FFT size of 4060 and 50 % overlap

Both spectrograms capture the major effects present in the audio file. For example, there are downward sweeping quadratic shaped curve at 110 seconds. This would sound like a pure tone that starts high and gradually lowers over the course of a few seconds. This audio effect is captured in both spectrograms so both would be usable for music analysis. However, in the MATLAB version, the one that occurs at 85 seconds is slightly more identifiable due lower noise of the resultand spectrogram. This allows for closely spaced frequencies to be identified more easily. The exact reason for why MATLAB's algorithm performs better is beyond the scope of this work, but it may be due to better reduction of spectral leakage. This assumption is mostly due to the fact in frequency bands which should be silent, my spectrogram appears brighter due to noise.

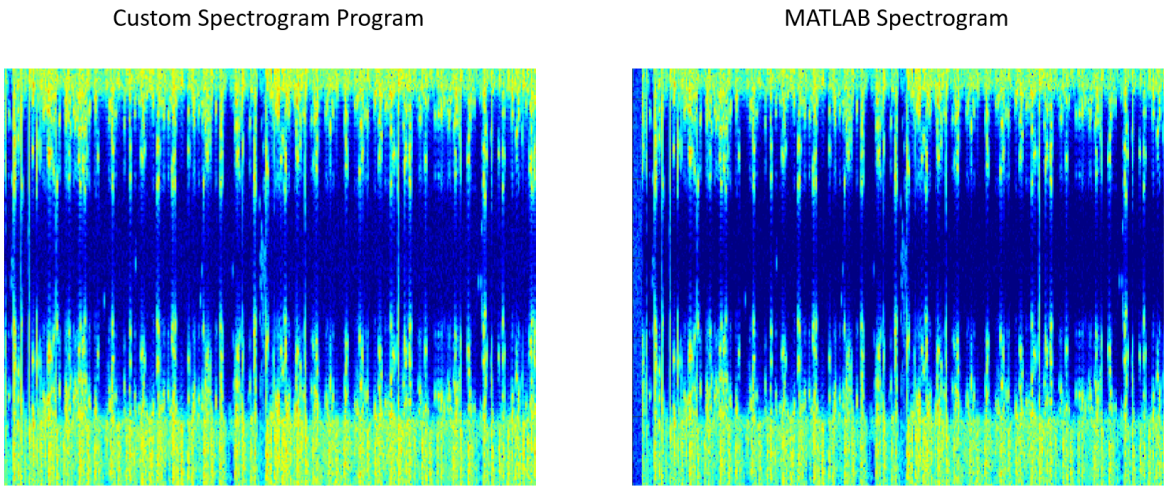

Figure 7: Noise Comparisons between custom made spectrogram and spectrogram from the Signal Processing Toolbox (SPToolbox)

The spill over of spectral leakage into adjacent frequency bins is seen pretty clearly in my implementation whereas from the SPToolbox sees lower levels of noise in silent areas. This leads to a loss in ability to differentiate closely space frequency components.

Some noise may have been introduced due to using mp3 audio files which is a lossy compression format, however the same file was used across tests so the increased noise comes from my implementation of the spectrogram. Outside of applications which require high fidelity, and low noise, my implementation is usable, if unoptimal.

One potential technique to address the issue of noise in the resultant spectrogram is to apply spectral subtraction. Spectral subtraction is a widely used method for noise reduction that involves estimating the noise profile from segments of the signal that are expected to contain only noise (such as silent portions). This estimated noise profile is then subtracted from the entire signal in the frequency domain, effectively reducing the noise component. Spectral subtraction, however, requires careful consideration of parameters, such as the noise estimation window and the subtraction factor, to avoid over or under-denoising.

4.2 Execution Time

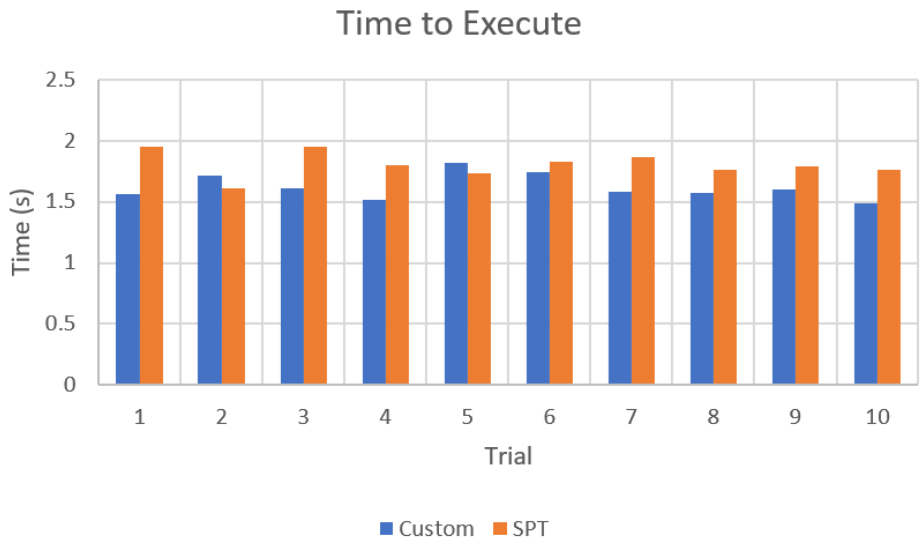

Figure 8: Execution times between custom vs MATLAB SPToolbox spectrogram

When it comes to execution time, the average over 10 trials was 1.620s vs 1.805s representing a 11% improvement in computational efficiency. This may have uses if computational efficiency is highly important but this does come at the cost of a nosier resultant spectrogram.

5. Conclusion

While I didn't start this project for the sake of developing something of utility, it was very promising to see that my spectrogram algorithm does have comparable robustness and faster execution compared to the one in the SPToolbox. The most valuable part of this project was the time I spent peaking under the hood of commonly tools that I use and delving into the nitty gritty details of how things work. As part of my effort to reverse engineer things and break things down into their fundamental components, I think this was a solid showing.

While the execution time was one aspect of this project that was successful, the amount of spectral leakage present in the final result is a cause for concern. My program relies exclusively on classical approach of windowing to reduce spectral leakage since these are computationally efficient, and additional techniques like spectral subtraction could be used to reduce noise. However, this doesn't address the issue of spectral leakage. One potential way to address the issue of spectral leakage is to average adjacent frequency columns in $\hat{A}$, where the amplitudes of each frequency will be influenced, by some factor α, by preceding and then succeeding columns which would have the effect of reducing noise. However, this would reduce temporal resolution and blur transient events, with the amount of blurring being influenced by how large α is.

In the future, I may look into exploring non-classical approaches such as the Minimum Variance Spectral Estimation (MVSE) [5], however I will likely also use other time-frequency visualizations and attempt to use libraries, either in MALTAB or python, to solve a more practical problem. Particularly, I am interested in satellite imaging, especially Synthetic Aperture Radar, and am looking to branch out to more electrical engineering style design work related to radar and satellite systems.

References

[1] Karol Abratkiewicz. Radar Detection-Inspired Signal Retrieval from the Short-Time Fourier Transform. *Sensors*, 22(16):5954, January 2022. Number: 16 Publisher: Multidisciplinary Digital Publishing Institute.

[2] Selin Aviyente, Edward M. Bernat, Stephen M. Malone, and William G. Iacono. Time-Frequency Data Reduction for Event Related Potentials: Combining Principal Component Analysis and Matching Pursuit. *EURASIP journal on advances in signal processing*, 2010:289571, January 2010.

[3] CCRMA. Windows package overview.

[4] Michael Cerna and Audrey F Harvey. The Fundamentals of FFT-Based Signal Analysis and Measurement.

[5] Mark Folwer. Minimum Variance Method EE 521 Lecture Slides. Binghamton University.

[6] Narcis Hudorović. Clinical significance of microembolus detection by transcranial Doppler sonography in cardiovascular clinical conditions. *International Journal of Surgery*, 4(4):232–241, January 2006.

[7] National Instruments. Understanding ffts and windowing.

[8] Weikuan Jia, Meili Sun, Jian Lian, and Sujuan Hou. Feature dimensionality reduction: a review. *Complex & Intelligent Systems*, 8(3):2663–2693, June 2022.

[9] Shalini Kapoor and Tarun Kumar. Fusing traditionally extracted features with deep learned features from the speech spectrogram for anger and stress detection using convolution neural network. *Multimedia Tools and Applications*, 81(21):31107–31128, September 2022.

[10] W. Max Lees, Adam Wunderlich, Peter Jeavons, Paul D. Hale, and Michael R. Souryal. Deep Learning Classification of 3.5 GHz Band Spectrograms with Applications to Spectrum Sensing. *IEEE transactions on cognitive communications and networking*, 5:10.1109/TCCN.2019.2899871, 2019.

[11] John W. Leis. *Digital Signal Processing Using MATLAB for Students and Researchers*. John Wiley & Sons, October 2011.

[12] Hugh Markus. Importance of Time-Window Overlap in the Detection and Analysis of Embolic Signals. *Stroke*, 26(11):2044–2047, November 1995. Publisher: American Heart Association.

[13] MATLAB. Hann (hanning) window.

[14] MATLAB. Short-time Fourier transform.

[15] Christoph Molnar. *Chapter 3 Interpretability | Interpretable Machine Learning*.

[16] Santiago Morales and Maureen E. Bowers. Time-frequency analysis methods and their application in developmental EEG data. *Developmental Cognitive Neuroscience*, 54:101067, April 2022.

[17] A. Nuttall. Some windows with very good sidelobe behavior. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(1):84–91, February 1981. Conference Name: IEEE Transactions on Acoustics, Speech, and Signal Processing.

[18] Vishal H. Shah and Mahesh Chandra. Speech Recognition Using Spectrogram-Based Visual Features. In Srikanta Patnaik, Xin-She Yang, and Ishwar K. Sethi, editors, *Advances in Machine Learning and Computational Intelligence*, Algorithms for Intelligent Systems, pages 695–704, Singapore, 2021. Springer.

[19] SIMCENTER. Window Correction Factors.

[20] SIMCENTER. Windows and Spectral Leakage.

[21] Weijie Wang, Gaopeng Zhang, Luming Yang, V. S. Balaji, V. Elamaran, and N. Arunkumar. Revisiting signal processing with spectrogram analysis on EEG, ECG and speech signals. *Future Generation Computer Systems*, 98:227–232, September 2019.

Appendix A: Program Code

```
clear, clc

% Load sound data
[data,Fs] = audioread('audio_file.MP3');
data = data (:,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% STFT Parameters

window_size = 4096;
FFT_size = window_size;
Overlap = 0.5;

% Aduio parameters
%   min_amplitude - sounds below this level(dB) are set to this value
%   freqMax - truncates values after this amount

min_amplitude = 50;
freqMax = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[amplitude,freqVec,timeVec, del_freq] = spec(data, FFT_size, Fs, Overlap);

amplitude = 10 * log10(amplitude/min_amplitude);
amplitude(amplitude < -min_amplitude) = -min_amplitude;

if freqMax > 0
    cutoff = floor(freqMax / (Fs/2) * length(freqVec) + 1);
    freqVec = freqVec (1:cutoff);
    amplitude = amplitude(1:cutoff,:);
end

% Graphing data
```

```
colormap('jet')
imagesc(timeVec,flipud(freqVec), amplitude)
title("My Spectrogram: \Delta f = " + del_freq + "Hz")
xlabel('Time (s)'); ylabel('Frequency (Hz)')
set(gca,'YDir','normal')

colorbar()
h = colorbar;
h.Label.String = "Amplitude (dB)";
h.Label.Rotation = 270;
h.Label.VerticalAlignment = "bottom";


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function definition
%   x  - Input data
%   NFFT  - FFT length
%   Fs - Sample frequency (Hz)
%   Overlap - buffer overlap
%   Energy loss term - corrects for the energy lost due to windowing

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function  [amp, freq, time, freq_increment] = spec(x, NFFT, Fs, Overlap)
    energy_loss_correction = 2;

    % Compute windows to analyze
    num_windows = floor((length(x) - NFFT) / (NFFT * (1 - Overlap)) + 1);

    w = hanning(NFFT);
    amp = zeros(NFFT/2, num_windows);

    for k = 1:num_windows
        start_value = 1 + floor((1-Overlap) * NFFT * (k-1));
        windowed_signal =  x(start_value:start_value + NFFT - 1) .* w;
        fft_output = abs(fft(windowed_signal)) * energy_loss_correction;
        amp(:,k) = fft_output(1:NFFT/2); % Take first half of the FFT output
    end

    max_freq = Fs/2;
    freq_increment = max_freq / (NFFT/2);
    freq = 1:freq_increment:max_freq;

    end_time = length(x)/Fs;
    time_increment = end_time/num_windows;
    time = 0:time_increment:end_time;
end
```