EEE3030 Signal Processing and Machine Learning

Semester 1 Report

Sahas Talasila *230057896*

Contents

Abstract

This report presents the findings and methodologies employed in the EEE3030 Signal Processing and Machine Learning course. It encompasses a comprehensive analysis of signal processing techniques, machine learning algorithms, and their applications in various domains. The report details the experimental setups, data analysis, and results obtained from implementing different models. Key insights and conclusions drawn from the study are also discussed, highlighting the effectiveness of the approaches used.

Plan for Report

1.1 Task 1 Plan

- Explain what the original file is, which is an AM signal with noise.
- Show this, showing the time domain output, where we can clearly see three peaks at x, y, z times.
- Show the frequency domain output using my fft code.
- Show calculations for sampling period and frequency resolution, explaining their importance in the context of FFT analysis.
- Explain the need for normalisation of the amplitude values.
- Show my fft code snippet.
- Explain why FFT is needed for analysis.
- Explain why windows are needed, show code snippets for different windows used.
- Show plots of different windowing methods and explain spectral leakage and also compare with unwindowed signals.
- Show calculations for fmin and fmax, talk about how I initially tried to use the proper methods, but then with jeff's help, I used the fact that the message bandwidth is 4kHz to determine the cut off frequencies for the bandpass filter.

1.2 Task 2 Plan

- Explain what an FIR filter is and why it is needed for bandpass filtering.
- Explain what bandpass filtering is and why it is needed in this context.
- Show the conditions needed for the filter design as a table, similar to the style in the pdf.
- Show how I designed the FIR filter (hand written calculations).
- Then show code snippet for FIR design in MATLAB and explain the code.
- Show the frequency response of the designed filter and explain what is going on.
- Show the frequency respones being applied to the AM signal.
- Show the code as well for the FIR filtering (convolution).

- Explain what convolution is and why it is needed
- Show the mixed signal in time domain, explain what is going on.
- Show the mixed signal in frequency domain, explain what is going on.
- Show the AM signal and explain its significance, and show how we can see the carrier frequency and sidebands.

1.3 Task 3 Plan

- Explain what the square law is and why we need it for task 3.
- Show the code snippet for squaring the filtered signal.
- Show the output of the squared signal in time domain and frequency domain.
- Explain what the plots mean (2fc and how we can approximate fc as a multiple).
- Show the code for carrier signal with phi as 0 and when we multiply it with the BP filtered signal. Explain that we have done that for ease and explain the code.
- Show the mixed signal in time domain and frequency domain. Explain what is going on in the plots.

1.4 Task 4 Plan

- Explain what an IIR filter is and why it is needed for low pass filtering.
- Show the conditions needed for the filter design as a table, similar to the style in the pdf.
- Show how I designed the IIR filter (hand written calculations) for Butterworth as well.
- Then show code snippet for IIR design in MATLAB and explain the code, and how it relates to the above.
- Show image with frequency response and explain why it is useful.
- Show the code for IIR filtering and plotting the output signal in time and frequency domain, then explain what is going on.

1.5 Task 5 Plan

- Explain the importance of phi and how it affects the output signal.
- Show the code snippet for the iterative phi increase and explain how the code works.
- Explain how SNR is calculated and its importance.
- Show the output signal in time domain and frequency domain for the best phi value obtained
- Explain the results obtained and compare with original message signal.
- Show the last output signal plot against original message signal plot.

Introduction

Results and Discussion

3.1 Task 1 Procedure, Results and Discussion

This section will cover the steps taken to read and analyse the provided AM signal with noise. The analysis includes time-domain and frequency-domain representations, as well as the application of various windowing techniques to mitigate spectral leakage.

3.1.1 Reading the Audio File and Statistics

Procedure

The provided AM signal was read using MATLAB's `audioread` function. The sampling frequency and duration of the signal were determined to facilitate further analysis. It is important to find the basic information about the audio file such as sampling frequency, frequency resolution, duration, number of samples etc. This can be clearly shown by using a few simple formulae, such as:

Time duration of signal:

$$T = \frac{1}{f_s} \tag{1}$$

Number of samples:

$$N = T \times f_s \tag{2}$$

Where f_s is the sampling frequency.

$$\Delta f = \frac{f_s}{N} \tag{3}$$

Where Δf is the frequency resolution.

This information is needed because it helps in understanding the characteristics of the signal and is essential for accurate FFT analysis. A time domain plot of the original AM signal can be shown as well, so that we can visually inspect the signal for any obvious features or anomalies.

This covers the first part of **Task 1**, as we have inspected the original signal in time domain and calculated important parameters.

```matlab
% Task 1: Time and Frequency Domain Analysis
clear; close all; clc;

% Load the AM signal
[signal, fs] = audioread('Sahas_Talasila.wav'); % Loading audio
   file
signal = signal(:);  % Ensure column vector

% Display basic information
fprintf('Sampling frequency: %d Hz\n', fs);
fprintf('Signal length: %d samples\n', length(signal));
fprintf('Duration: %.2f seconds\n', length(signal)/fs);
```

```
% %% Time Domain Plot

% This code has been commented out to reduce plotting during
   automated runs.

t = (0:length(signal)-1) / fs;  % Time vector in seconds

figure('Position', [100 100 1200 400]);
plot(t, signal, 'b', 'LineWidth', 0.5);
xlabel('Time (s)', 'FontSize', 12);
ylabel('Amplitude', 'FontSize', 12);
title('Received AM Signal - Time Domain', 'FontSize', 14);
grid on;
xlim([0 max(t)]);
```

Listing 1: MATLAB Code for Task 1: Time and Frequency Domain Analysis

This code snippet (Listing 1) demonstrates how to read the AM signal, extract its sampling frequency, and plot the time-domain representation of the signal. The time vector is calculated based on the length of the signal and the sampling frequency. There is a method to easily generate the AM signal in the time domain. From (FIGURE X), we can see that the sampling frequency is $96000Hz$, total sample count is 244104, frequency resolution of $0.39Hz$ and a duration of 2.54 seconds. The terminal output is show below and has been calculated using the above code snippet and *Equations 1, 2 and 3.*

These statistics can show us important information about the signal. Sampling frequency (f_s) indicates how many samples are taken per second, which is crucial for accurately capturing the signal's characteristics. The duration tells us how long the signal lasts, and the number of samples gives insight into the signal's resolution in both time and frequency domains.

Sample count can also help in determining the frequency resolution (Δf) when performing FFT analysis, which is essential for identifying frequency components within the signal.

Frequency resolution is particularly important when analyzing signals with closely spaced frequency components, as it determines the ability to distinguish between these components in the frequency domain. A smaller frequency resolution value means that the FFT can resolve finer details in the frequency spectrum (more bins, easier to differentiate difficult signals).

FIGURE SHOWING THE TERMINAL OUTPUT FOR SAMPLING FREQUENCY, DURATION ETC.

We can then see the time domain plot of the original AM signal, which shows the amplitude variations over time. This plot is crucial for visualizing the characteristics of the AM signal and identifying any potential issues such as noise or distortion. Some initial observations reveal a few key pieces of information about the signal. We can see that there are three distinct peaks in the signal at approximately 0.5s, 1.5s, and 2.3s.

These peaks likely correspond to the presence of the message signal within the AM waveform. The rest of the signal appears to be relatively low in amplitude, indicating that it is primarily noise.

FIGURE SHOWING THE TIME DOMAIN PLOT OF THE ORIGINAL AM SIGNAL.

3.1.2 Frequency Domain Analysis

Procedure

To analyze the frequency content of the AM signal, the Fast Fourier Transform (FFT) was applied. The FFT provides a frequency-domain representation of the signal, allowing for the identification of key frequency components. The amplitude spectrum was normalized to facilitate comparison and interpretation. For some background theory:

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the Discrete Fourier Transform (DFT). The DFT converts a signal from the **time domain**, where we see how the signal changes over time, to the **frequency domain**, where we see which frequencies are present in the signal. *Equation 4* below shows the mathematical representation of the DFT:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi kn/N} \tag{4}$$

Here, $x[n]$ is the time-domain signal and $X[k]$ is the frequency-domain representation.

Frequency domain analysis is needed because it allows the user to see the different frequency components that make up the signal. This is particularly useful for identifying periodic signals, noise, and other characteristics that may not be easily observable in the time domain, wherein we only saw the 3 peaks and noise that permeated through.

```matlab
%% Frequency Domain Analysis - Manual FFT Implementation
N = length(signal);   % Number of samples
df = fs / N;          % Frequency resolution

% Compute FFT
signal_fft = fft(signal);

% Create frequency vector (only positive frequencies)
f = (0:N/2) * df;
```

```
% Take single-sided spectrum (positive frequencies only)
signal_fft_single = signal_fft(1:N/2+1);

% Amplitude normalisation (convert to actual amplitudes)
signal_amplitude = abs(signal_fft_single) / N;
signal_amplitude(2:end-1) = 2 * signal_amplitude(2:end-1);  %
   Double non-DC components

% Convert to dB scale
signal_dB = 20 * log10(signal_amplitude + eps);  % eps prevents
   log(0)

fprintf('\nFrequency resolution: %.2f Hz\n', df);

% %% Plot Frequency Spectrum

% This code has been commented out to reduce plotting during
   automated runs.

figure('Position', [100 100 1200 500]);
plot(f/1000, signal_dB, 'b', 'LineWidth', 1);
xlabel('Frequency (kHz)', 'FontSize', 12);
ylabel('Magnitude (dB)', 'FontSize', 12);
title('Received AM Signal - Frequency Domain (Full Spectrum)', '
   FontSize', 14);
grid on;
xlim([0 fs/2000]);  % Full range to fs/2
```

This code snippet (Listing **??**) demonstrates the manual implementation of the FFT in MATLAB. It calculates the frequency vector, normalises the amplitude spectrum, and converts it to a decibel scale for better visualisation. This step is needed, because normalisation ensures that the amplitude values accurately reflect the signal's strength across different frequency components. Without normalisation, the amplitude values could be misleading, making it difficult to interpret the frequency content of the signal correctly. Likewise, the log scale helps in visualising a wide range of amplitude values, making it easier to identify significant frequency components. The frequency resolution calculated here is approximately $0.39Hz$, which aligns with the earlier calculation based on the sampling frequency and number of samples.

FIGURE SHOWING THE FREQUENCY DOMAIN PLOT OF THE ORIGINAL AM SIGNAL.

From **FIGURE X**, it is evident that the AM signal contains significant frequency components around the carrier frequency and its sidebands. The presence of these components confirms the AM nature of the signal, where the message signal modulates the amplitude of the carrier wave. However, there is also a considerable amount of noise present across the frequency spectrum, which could potentially interfere with the accurate demodulation of the message signal. This noise is likely due to various factors, including environmental interference and imperfections in the transmission process, but in this case, it has been artificially added in.

It is safe to assume that the carrier frequency is roughly around $16kHz$, with $f_{min} = 15.6kHz$ and $f_{max} = 16.4kHz$, because we can see the two sidebands around this frequency, which is typical for AM signals. But the bandwidth of the message signal is $4kHz$, which means that the sidebands can actually extend up to $\pm 4kHz$. This is the reason why I had to use trial and error to find the cut off frequencies for the bandpass filter in task 2, as the initial calculations did not yield good results due to the noise present in the signal.

3.1.3 Windowing and Spectral Leakage

Procedure

To mitigate spectral leakage in the FFT analysis, various windowing techniques were applied to the AM signal. Spectral leakage is a phenomenon that occurs when the signal being analysed is not perfectly periodic within the observation window, leading to discontinuities at the edges of the sampled signal. These discontinuities introduce artifacts in the frequency domain, causing energy from one frequency component to "leak" into adjacent frequencies. This can obscure the true frequency content of the signal and make it difficult to accurately identify and analyze specific frequency components. Window functions work by tapering the edges of the signal to zero, which minimizes discontinuities when the signal is treated as periodic during the FFT computation, making it a bit easier and improving accuracy. This tapering reduces the abrupt transitions at the boundaries, thereby decreasing spectral leakage.

Whilst I did compare different windowing methods such as Hamming, Hanning and Blackman windows, I found that the Hamming window provided the best balance between main lobe width and side lobe levels for this specific application. The Hamming window effectively reduced spectral leakage while maintaining a reasonable frequency resolution, making it suitable for analyzing the AM signal in this context. I have excluded the other windowing methods from the report for brevity.

The Hamming window is defined mathematically as follows:

$$w[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right), \quad 0 \leq n \leq N-1 \tag{5}$$

Where N is the total number of samples in the window.

```
%% Estimate Carrier Frequency from Spectrum

% Calculate noise floor and threshold
noise_hamming = median(dB_hamming);
threshold_hamming = noise_hamming + 20;  % 20 dB above noise

% Detect signal bandwidth
detect_hamming = find(dB_hamming > threshold_hamming);

if ~isempty(detect_hamming)
    fmin_detected = f(detect_hamming(1));
    fmax_detected = f(detect_hamming(end));

    % Round to nearest 100 Hz
    fmin_detected = round(fmin_detected/100) * 100;
    fmax_detected = round(fmax_detected/100) * 100;

    % Estimate carrier frequency (center of detected band)
    fc_estimated = (fmin_detected + fmax_detected) / 2;

    fprintf('\n--- Detected Signal Band ---\n');
    fprintf('fmin (detected): %.2f kHz (%.0f Hz)\n',
   fmin_detected/1000, fmin_detected);
    fprintf('fmax (detected): %.2f kHz (%.0f Hz)\n',
   fmax_detected/1000, fmax_detected);
    fprintf('fc (estimated): %.2f kHz (%.0f Hz)\n', fc_estimated
   /1000, fc_estimated);
end

%% Set Bandpass Filter Edges
% The message bandwidth B = 4kHz
% Therefore, bandpass filter should be fc +- 4kHz

fmin = fc_estimated - 3800;  % fc - 4kHz
fmax = fc_estimated + 3700;  % fc + 4kHz

fprintf('\n=== Bandpass Filter Design Parameters ===\n');
fprintf('Carrier frequency (fc): %.0f Hz (%.2f kHz)\n',
   fc_estimated, fc_estimated/1000);
fprintf('Bandpass filter edges:\n');
fprintf('  fmin = fc - 4kHz = %.0f Hz (%.2f kHz)\n', fmin, fmin
   /1000);
fprintf('  fmax = fc + 4kHz = %.0f Hz (%.2f kHz)\n', fmax, fmax
   /1000);
```

```
fprintf('  Bandwidth = %.0f Hz (%.2f kHz)\n', fmax-fmin, (fmax-
   fmin)/1000);
```

The code snippet (Listing **??**) demonstrates the application of the Hamming window to the AM signal before performing the FFT. The windowed signal is then analysed in the frequency domain to assess the effectiveness of the windowing technique in reducing spectral leakage. The code works by first creating the Hamming window based on the length of the signal. The window is then applied to the original signal through element-wise multiplication. After windowing, the FFT is computed, and the amplitude spectrum is normalised and converted to a decibel scale for visualisation. Afterwards, the code estimates the carrier frequency and sets the band-pass filter edges based on the detected signal bandwidth (but this has been changed to $\approx \pm B$, where $B = 4kHz$).

FIGURE SHOWING THE FREQUENCY DOMAIN PLOT OF THE AM SIGNAL WITH HAMMING WINDOW APPLIED.

From **FIGURE X**, it is evident that the application of the Hamming window has significantly reduced spectral leakage in the frequency domain representation of the AM signal. The frequency components are more distinct, and the overall noise floor appears lower compared to the unwindowed FFT analysis. This improvement is crucial for accurately identifying the carrier frequency and sidebands of the AM signal, as it allows for a clearer distinction between the signal components and the surrounding noise. The reduced spectral leakage enhances the reliability of subsequent analyses, such as filter design and demodulation, which are essential for recovering the original message signal from the noisy AM waveform.

3.2 Task 2

The main focus of Task 2 is the design and implementation of a bandpass FIR filter to isolate the desired frequency components of the AM signal. This section will detail the filter design process, including the selection of filter parameters, the implementation of the filter in MATLAB, and the analysis of the filtered signal in both time and frequency domains.

3.2.1 Filter Design and Output Verification

Procedure

The bandpass FIR filter was designed using the window method, specifically employing a Hamming window to shape the filter's impulse response.

The filter parameters were determined based on the estimated carrier frequency and the message bandwidth identified in Task 1. The filter design specifications are as follows:

Table 1: Bandpass FIR Filter Design Specifications

Parameter	**Value**
F_{min} (passband frequency)	$12kHz$
F_{max} (passband frequency)	$20kHz$
Stopband frequencies	$10kHz$, $22kHz$
Max Passband Ripple	$0.1dB$
Stopband Attenuation	$50dB$

The filter was implemented with a custom convolution function to apply the FIR filter to the AM signal. The frequency response of the designed filter was analysed to ensure it met the specified design criteria. The following shows the mathematical design of the FIR filter using the window method:

$$h[n] = h_d[n] \cdot w[n] \tag{6}$$

Where $h_d[n]$ is the ideal impulse response of the bandpass filter and $w[n]$ is the Hamming window function. The ideal impulse response for a bandpass filter is given by:

$$h_d[n] = \frac{2f_{max}}{f_s}\text{sinc}\left(\frac{2f_{max}n}{f_s}\right) - \frac{2f_{min}}{f_s}\text{sinc}\left(\frac{2f_{min}n}{f_s}\right) \tag{7}$$

The filter order was determined based on the desired stopband attenuation and transition width, ensuring that the filter effectively attenuated frequencies outside the passband while preserving the integrity of the desired signal components.

We can then substitute our requirements into the above equations to get the final filter coefficients, which can then be used in the convolution process to filter the AM signal.

```
%% Filter Specifications
% Passband edges (from Task 1)
fp1 = fmin;   % Lower passband edge (fc - 4kHz)
fp2 = fmax;   % Upper passband edge (fc + 4kHz)

% Stopband edges (assignment specification)
fstop1 = fmin - 2000;   % Lower stopband edge
```

```matlab
fstop2 = fmax + 2000;   % Upper stopband edge

% Performance requirements
passband_ripple_dB = 0.1;
stopband_atten_dB = 50;

% Transition bandwidth
transition_bw = fp1 - fstop1;  % = 2000 Hz

fprintf('\nFilter Specifications:\n');
fprintf('Passband: %.0f Hz to %.0f Hz\n', fp1, fp2);
fprintf('Stopband: DC-%.0f Hz and %.0f Hz-Nyquist\n', fstop1,
   fstop2);
fprintf('Transition bandwidth: %.0f Hz\n', transition_bw);
fprintf('Passband ripple: %.1f dB\n', passband_ripple_dB);
fprintf('Stopband attenuation: %.0f dB\n', stopband_atten_dB);

%% Estimate Filter Order
% Using Hamming window formula: N = 3.3 * fs / transition_bw
N_estimated = ceil(6.0 * fs / transition_bw);

% Make it odd for Type I FIR (symmetric)
if mod(N_estimated, 2) == 0
    N_estimated = N_estimated + 1;
end

fprintf('\nEstimated filter order: %d taps\n', N_estimated);

M = N_estimated;  % Filter length

%% Design Ideal Bandpass Impulse Response
% Normalize frequencies to [0, 1] where 1 is Nyquist (fs/2)
wc1 = 2 * fp1 / fs;   % Normalized lower cutoff
wc2 = 2 * fp2 / fs;   % Normalized upper cutoff

% Center point of filter
M_center = (M - 1) / 2;

% Ideal bandpass impulse response
h_ideal = zeros(1, M);
for i = 1:M
    if i == M_center + 1  % Handle n = 0 case (avoid division by
   zero)
        h_ideal(i) = wc2 - wc1;
    else
        % Sinc functions for bandpass
        n = i - M_center - 1;
        h_ideal(i) = (sin(pi * wc2 * n) - sin(pi * wc1 * n)) / (
   pi * n);
    end
end
```

```

fprintf('Ideal impulse response computed\n');

%% Apply Hamming Window
% Generate Hamming window
w_hamming = hamming(M)';

% Apply window to ideal impulse response
h_fir = h_ideal .* w_hamming;

fprintf('Hamming window applied\n');
fprintf('Final filter length: %d taps\n', length(h_fir));

%% Verify Filter Frequency Response
% Compute frequency response (use FFT with zero-padding)
N_fft = 8192;  % Zero-padding for smooth frequency response
H = fft(h_fir, N_fft);
H_mag = abs(H(1:N_fft/2+1));
H_dB = 20 * log10(H_mag + eps);

% Frequency vector
f_response = (0:N_fft/2) * fs / N_fft;

% Plot frequency response
figure('Position', [100 100 1000 600]);
plot(f_response/1000, H_dB, 'b', 'LineWidth', 1.5);
hold on;

% Mark specifications
yline(-passband_ripple_dB, 'g--', 'LineWidth', 1.5, 'Label', '
   Passband ripple');
yline(-stopband_atten_dB, 'r--', 'LineWidth', 1.5, 'Label', '
   Stopband spec');
xline(fp1/1000, 'k--', 'LineWidth', 1);
xline(fp2/1000, 'k--', 'LineWidth', 1);
xline(fstop1/1000, 'r--', 'LineWidth', 1);
xline(fstop2/1000, 'r--', 'LineWidth', 1);

xlabel('Frequency (kHz)', 'FontSize', 12);
ylabel('Magnitude (dB)', 'FontSize', 12);
title('FIR Bandpass Filter Frequency Response', 'FontSize', 14);
grid on;
xlim([0 fs/2000]);
ylim([-80 5]);

%% Apply FIR Filter via Custom Convolution
fprintf('\nApplying FIR filter via custom convolution...\n');

signal_filtered = custom_conv(signal, h_fir);

fprintf('Filtering complete. Output length: %d samples\n', length
    (signal_filtered));
```
Listing 2: MATLAB Code for Task 2: FIR Filter Design Code

Conclusion

explain what was done in the report and summarise the results obtained. Talk about any challenges faced and how they were overcome. Discuss any potential improvements or future work that could be done based on the findings of the report. Need to add some stuff on how I initially tried to find the upper and lower limits for the bandpass filter by looking at the fft of the AM signal but this did not work well as there was too much noise. So I had to use trial and error to find suitable cut off frequencies for the bandpass filter.

References

References that I have used in the report. (articles, MATLAB documentation, textbooks etc.)

Appendix

Include some flowcharts for code design if possible. Include entire code listings if possible or split for the tasks. (code snippet for task 1, task 2 etc.) Include conv.m and iir filter design code.