

# EEE3030 Report

## Introduction

This assignment investigates the analysis and demodulation of a noisy double-sideband suppressed carrier (DSB-SC) AM signal using MATLAB. A personalised waveform sampled at 96 kHz is analysed in the time and frequency domains using the FFT, with appropriate scaling and windowing, to accurately estimate the AM signal bandwidth.

Based on this analysis, a finite-impulse-response (FIR) bandpass filter is designed using impulse-response truncation to isolate the AM signal and suppress out-of-band noise. The filter frequency response is verified, and the filtering operation is implemented using manual convolution. Carrier recovery is achieved by applying a square-law nonlinearity, identifying the spectral component at twice the carrier frequency, and then coherent demodulating with a locally generated carrier.

An infinite-impulse-response (IIR) fourth-order Butterworth lowpass filter is subsequently designed, verified, and implemented to recover the baseband message signal. Finally, the carrier phase is adjusted to maximise the output signal amplitude and signal-to-noise ratio, and the demodulated signal is output as an audio waveform to identify the three-letter spoken message.

## TASK 1 – Time and frequency domain analysis

The signal was read into MATLAB using the `audioread()` function and converted into a column vector for consistent processing. The signal was first examined in the time domain by plotting amplitude versus time as shown in Figure 1. The waveform exhibits high-frequency oscillations with a slowly varying envelope, characteristic of an amplitude-modulated (DSB-SC) signal and is corrupted by noise.

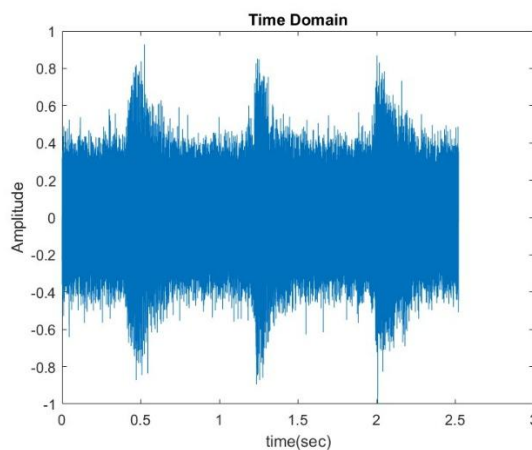


Figure 1: Time Domain of Received Signal

By observing the time domain signal, the length is 2.5 seconds. Hence, the number of samples,  $N$ , is 240k, given a sampling frequency of 96k.

Frequency-domain analysis was then performed using the discrete Fourier transform, implemented via the `fft()` function. The FFT output was normalised by the signal length and converted to a logarithmic magnitude scale in decibels. The frequency axis was correctly scaled using the sampling frequency, and the spectrum was plotted from 0 Hz to the Nyquist frequency ( $f_s/2$ ), as shown in Figure 2.

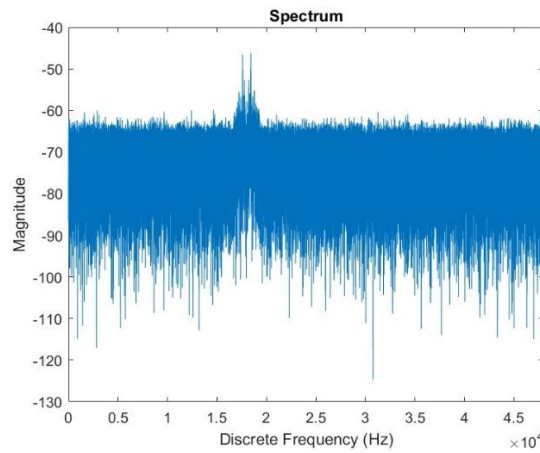


Figure 2: Frequency Domain

To investigate spectral leakage and frequency resolution, three window functions were manually applied: rectangular, Hanning, and Hamming. The windowed signals were transformed via the FFT, and their spectra were compared in Figure 3.

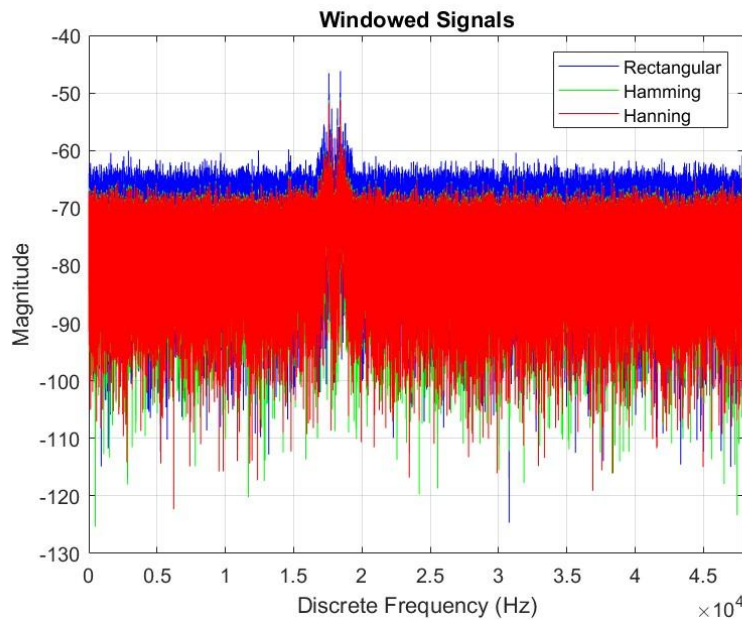


Figure 3: Windowed Signals for comparison

The rectangular window exhibits the highest spectral leakage, as evidenced by elevated sidelobe levels. Both the Hanning and Hamming windows significantly reduce sidelobe energy. From the windowed frequency spectra, the carrier frequency was clearly identified at approximately 18 kHz. Symmetric sidebands are present on either side of the carrier, consistent with amplitude modulation.

The significant spectral content extends from approximately 14 kHz to 22 kHz, corresponding to an estimated bandwidth of 8 kHz. These values were used to define the lower and upper passband edge frequencies,  $f_{\min} = 14\text{kHz}$  and  $f_{\max} = 22\text{kHz}$ , which will be needed for the bandpass filter design in Task 2.

Picture 1 below is the complete code for Task 1.

```

% TASK 1 - Time and frequency domain analysis
clear;
close all;
clear all;

[x,fs] = audioread('Vasiliki Sava.wav'); %Extract signal from audio
N = length(x); %Sampling period
Ts = 1/fs; %Calculates N using the duration of the signal
t = 0:Ts:(N-1)*Ts; %Generate discrete time values (nts)

%Plot time domain
figure; plot(t,x);
xlabel('Time(sec)'); ylabel('Amplitude'); title('Time Domain');

%Calculate FFT
x_fft = fft(x);
N = length(x);
h = (0:(N-1))/N; %Normalise frequency
f = h*fs;

%Normalise frequency
x_norm = abs(x_fft)/N;
x_db = 20*log10(x_norm);
figure; plot(f,x_db);
xlabel('Frequency (Hz)'); ylabel('Magnitude'); title('Frequency Spectrum');

%Generate N points Windows
ham_w = 0.5*(1+cos(2*pi*h/(N-1))); % Hamming
rect_w = ones(1, N); % Rectangular
ham_w = (0.54 - 0.46 * cos(2 * pi * (0:N-1) / (N - 1)))'; % Hamming

%Apply windows to signal
xh = x.*ham_w; %Hamming
xh_fft = fft(xh);
xh_norm = abs(xh_fft)/N;
xh_db = 20*log10(xh_norm);

%Rectangular
xr = x.*rect_w;
xr_fft = fft(xr);
xr_norm = abs(xr_fft)/N;
xr_db = 20*log10(xr_norm);

%Hamming
xm = x.*ham_w;
xm_fft = fft(xm);
xm_norm = abs(xm_fft)/N;
xm_db = 20*log10(xm_norm);

%Plot Windowed Signals for comparison
figure;
plot(f,xh_db,'b');
hold on;
plot(f,xr_db,'g');
plot(f,xm_db,'r');
hold off;
xlabel('Discrete Frequency (Hz)'); ylabel('Magnitude'); title('Windowed Signals');
legend('Rectangular', 'Hamming', 'Hamming');
grid on;
xlim([0 fs/2]);

%Estimations of the spectrum
fc = 18000;
Bw = 8000;
fmin = fc - Bw/2; % use an 8kHz bandwidth
fmax = fc + Bw/2;

```

Picture 1: Complete Code for Task 1

## Task 2 – Bandpass filter

In Task 2, a finite-impulse-response (FIR) bandpass filter was designed using the impulse-response truncation method. The passband edges were set to  $f_{\min} = 14\text{kHz}$  and  $f_{\max} = 22\text{kHz}$  based on the spectral analysis from Task 1. A transition width of 2 kHz was chosen, determining the filter lengths required for the Hamming and Blackman window designs. The Blackman window achieved higher stopband attenuation than the Hamming, exceeding 50 dB, with a slightly wider main lobe.

Window functions were then applied manually to reduce spectral leakage and control sidelobe levels. The resulting frequency responses of the filters were computed using the Fourier transform and are shown in Figure 4. The magnitude spectra confirm that both filters isolate the desired AM signal band while attenuating out-of-band noise.

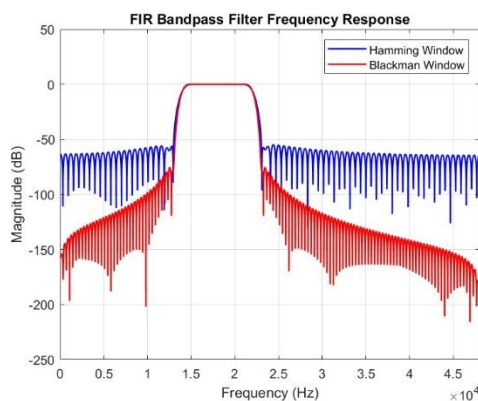


Figure 4: Frequency Response of Hamming and Blackman FIR Bandpass Filter

The filters were applied to the received AM signal via manual convolution as shown in Figure 5.

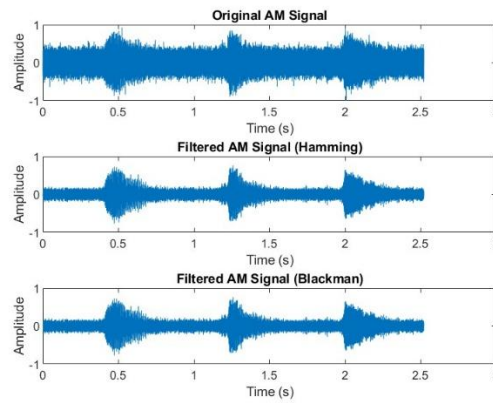


Figure 5: Time-Domain Comparison of Original and FIR-Filtered Signals

The time-domain outputs demonstrate that the filtered signals retain the AM waveform while reducing noise outside the passband. Frequency-domain analysis of the filtered signals was performed to verify passband preservation and stopband attenuation, as shown in Figure 6.

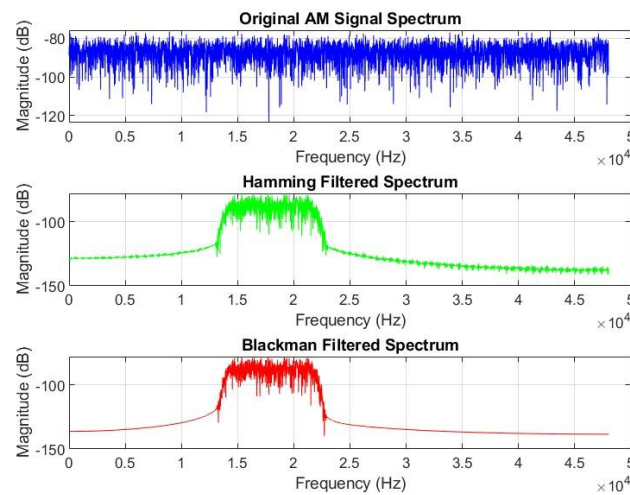


Figure 6: Frequency-Domain Comparison of Original and FIR-Filtered Signals

The spectra confirm that both FIR filters preserve the AM signal between 14–22 kHz and effectively remove unwanted spectral components. The Blackman-filtered signal appears slightly smoother due to stronger sidelobe suppression.

Picture 2 shows the complete code for Task 2

```

69 % TASK 2 - Bandpass Filter
70
71 df = 2000; % transition width
72
73 % Filter lengths
74 Nham = ceil(3.3*Fs/df); if mod(Nham,2)~=0, Nham=Nham+1; end
75 Nblc = ceil(5.5*Fs/df); if mod(Nblc,2)~=0, Nblc=Nblc+1; end
76
77 % Normalized frequencies
78 Fc1 = fmin/Fs;
79 Fc2 = fmax/Fs;
80
81 % Ideal impulse responses
82
83 % Hamming
84 n_h = 0:Nham-1;
85 a_h = (Nham-1)/2;
86 h_ideal_ham = 2*Fc2*sinc(2*Fc2*(n_h - a_h)) - 2*Fc1*sinc(2*Fc1*(n_h - a_h));
87
88 % Blackman
89 n_b = 0:Nblc-1;
90 a_b = (Nblc-1)/2;
91 h_ideal_blk = 2*Fc2*sinc(2*Fc2*(n_b - a_b)) - 2*Fc1*sinc(2*Fc1*(n_b - a_b));
92
93 % Windows (manual)
94
95 w_ham = 0.54 - 0.46*cos(2*pi*n_h/(Nham-1));
96 w_blk = 0.42 - 0.5*cos(2*pi*n_b/(Nblc-1)) + 0.08*cos(4*pi*n_b/(Nblc-1));
97
98 % Apply windows
99 h_ham = h_ideal_ham .* w_ham;
100 h_blk = h_ideal_blk .* w_blk;
101
102
103 N_fft = 8192;
104 H_ham = fft(h_ham, N_fft);
105 H_blk = fft(h_blk, N_fft);
106
107 f_resp = (0:N_fft/2) * Fs / N_fft;
108
109 H_ham_one = 20*log10(abs(H_ham(1:N_fft/2+1)) + eps);
110 H_blk_one = 20*log10(abs(H_blk(1:N_fft/2+1)) + eps);
111
112 figure;
113 plot(f_resp, H_ham_one, 'b', 'linewidth', 1.2); hold on;
114 plot(f_resp, H_blk_one, 'r', 'linewidth', 1.2);
115 xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
116 title('FIR Bandpass Filter Frequency Response');
117 legend('Hamming window', 'Blackman window');
118 grid on; xlim([0 Fs/2]);
119
120 % Manual FIR convolution
121 y_ham = zeros(1, N);
122 y_blk = zeros(1, N);
123
124 % Hamming filter
125 for n = 1:N
126     acc = 0;
127     for k = 1:Nham
128         if (n - k + 1) > 0
129             acc = acc + h_ham(k)*x(n - k + 1);
130         end
131     end
132     y_ham(n) = acc;
133 end
134
135 % Blackman filter
136 for n = 1:N
137     acc = 0;
138     for k = 1:Nblc
139         if (n - k + 1) > 0
140             acc = acc + h_blk(k)*x(n - k + 1);
141         end
142     end
143     y_blk(n) = acc;
144 end
145
146 % Time-domain comparison
147
148 figure;
149 subplot(3,1,1); plot(t,x); title('Original AM Signal'); xlabel('Time (s)'); ylabel('Amplitude');
150 subplot(3,1,2); plot(t,y_ham); title('Filtered AM Signal (Hamming)'); xlabel('Time (s)'); ylabel('Amplitude');
151 subplot(3,1,3); plot(t,y_blk); title('Filtered AM Signal (Blackman)'); xlabel('Time (s)'); ylabel('Amplitude');
152
153 % Frequency-domain comparison
154 % Compute FFT and one-sided magnitude in dB
155 X = 20*log10(abs(fft(x, N_fft)/N) + eps); % Original signal
156 Yh = 20*log10(abs(fft(y_ham, N_fft)/N) + eps); % Hamming filtered
157 Yb = 20*log10(abs(fft(y_blk, N_fft)/N) + eps); % Blackman filtered
158
159 % Keep only positive frequencies
160 X = X(1:N_fft/2+1);
161 Yh = Yh(1:N_fft/2+1);
162 Yb = Yb(1:N_fft/2+1);
163
164 % Plot
165 figure;
166 subplot(3,1,1); plot(f_resp, X, 'b'); title('Original AM Signal Spectrum'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); grid on;
167 subplot(3,1,2); plot(f_resp, Yh, 'g'); title('Hamming Filtered Spectrum'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); grid on;
168 subplot(3,1,3); plot(f_resp, Yb, 'r'); title('Blackman Filtered Spectrum'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); grid on;

```

Picture 2: Code for task 2

### Task 3 – Carrier Recovery and Mixing

In Task 3, the carrier recovery was performed using a square-law method. The bandpass signal was squared, which generates a spectral component at twice the carrier frequency ( $2f_c$ ). To reduce spectral leakage, a Hamming window was applied to the squared signal before performing a high-resolution FFT ( $N = 2^{16}$ ). Figure 7 shows the spectrum of the squared signal.

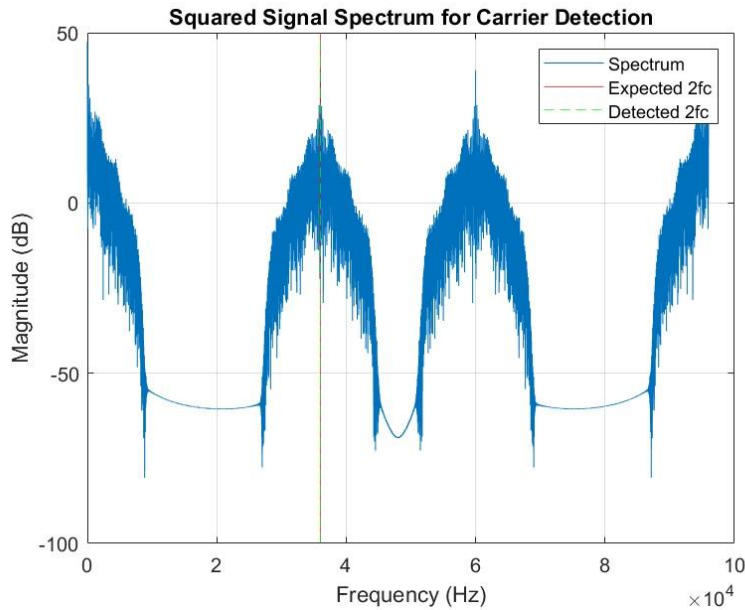


Figure 7: Squared Signal Spectrum for Carrier Detection

A clear peak near  $2f_c$  is observed, which was used to estimate the carrier frequency. The red line indicates the expected  $2f_c$  based on Task 1 and the green dashed line shows the detected  $2f_c$  peak. The peak frequency  $2f_c$  was divided by two to obtain the estimated carrier frequency  $f_c \approx 18$  kHz. This closely matches the expected carrier frequency from Task 1, confirming successful carrier detection.

Next, a local cosine carrier with the estimated frequency and an initial phase  $\phi = 0$  was generated. The bandpass signal was then multiplied by this local carrier to produce the mixed signal. This process shifts the AM signal to baseband, recovering the modulation signal while preserving its amplitude.

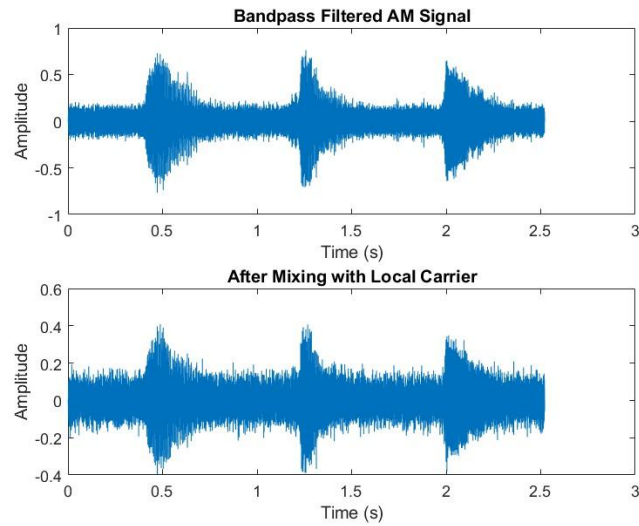


Figure 8: Time-Domain Signals Before and After Mixing

Picture 3 shows the complete code for task 3.

```

173 %% Task 3 - Carrier Recovery & Mixing
174
175 x_bp = y_hk;
176 x_sq = x_bp.^2;
177
178 % Apply Hamming window to reduce spectral leakage
179 h_win = hamming(length(x_sq));
180 x_sq_windowed = x_sq .* h_win;
181
182 % FFT with high resolution
183 N_fft = 2^16;
184 Xsq = fft(x_sq_windowed, N_fft);
185
186 % Frequency vector (0 to fs)
187 f_sq = (0:N_fft-1)*(fs/N_fft);
188
189 % Expected carrier from Task 1
190 expected_fc = 18000; % Hz
191
192 % Search range around 2*fc
193 search_range_low = 2*expected_fc - 5000; % Hz
194 search_range_high = 2*expected_fc + 5000; % Hz
195 idx_band = (f_sq >= search_range_low) & (f_sq <= search_range_high);
196
197 % Find the peak in squared signal spectrum
198 [~, idx_max] = max(abs(Xsq(idx_band)));
199
200 % Map peak back to frequency
201 f_2fc = f_sq(idx_max);
202 f_cfc = f_2fc/2;
203
204 % Compute estimated carrier frequency
205 fc_est = f_cfc / 2;
206 fprintf('Estimated carrier frequency fc = %.0f Hz\n', fc_est);
207
208 % Generate local carrier
209 t_bp = (0:length(x_bp)-1)/fs;
210 phi = 0; % Initial phase
211 local_carrier = cos(2*pi*fc_est*t_bp + phi);
212
213 % Mixing (multiply bandpass signal with local carrier)
214 x_mix = x_bp .* local_carrier;
215
216 % Time domain plots
217 figure;
218 subplot(2,1,1); plot(t_bp, x_bp); title('Bandpass Filtered AM Signal'); xlabel('Time (s)'); ylabel('Amplitude');
219 subplot(2,1,2); plot(t_bp, x_mix); title('After Mixing with Local Carrier'); xlabel('Time (s)'); ylabel('Amplitude');
220
221 % Frequency domain plot
222 Xmix = fft(abs(x_mix), N_fft);
223 f_fft = linspace(0, fs, N_fft);
224 figure;
225 plot(f_sq, 20*log10(abs(Xsq))+eps); hold on;
226 vline(2*expected_fc, 'r');
227 vline(f_2fc, 'g');
228 xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
229 title('Squared Signal Spectrum for Carrier Detection');
230 grid on;
231 legend('Spectrum', 'Expected 2fc', 'Detected 2fc');
232
233
234

```

Picture 3: code for task 3

## Task 4 – Lowpass filter

The goal of this filter is to remove high-frequency components, thereby reducing background noise. The 4<sup>th</sup> order Butterworth filter was designed with a cutoff frequency of 4 kHz. The cutoff frequency was normalised with respect to the Nyquist frequency ( $f_s/2$ ) to meet digital filter requirements and the IIR filter was applied manually using the difference equation. The filter's frequency response was manually computed from the FFT of its coefficients and is plotted in Figure 9.

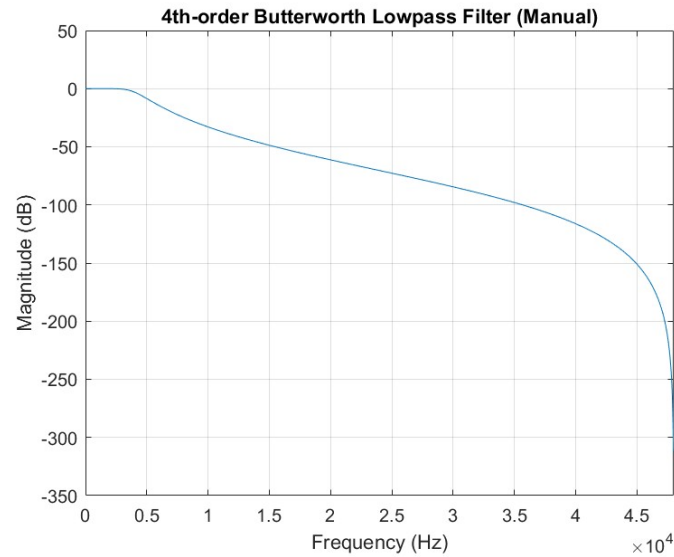


Figure 9: Frequency Response of 4th-order Butterworth Lowpass Filter

The filter's response shows a flat passband up to 4 kHz with smooth attenuation beyond the cutoff. The time-domain signals before and after filtering are shown in Figure 10 below.

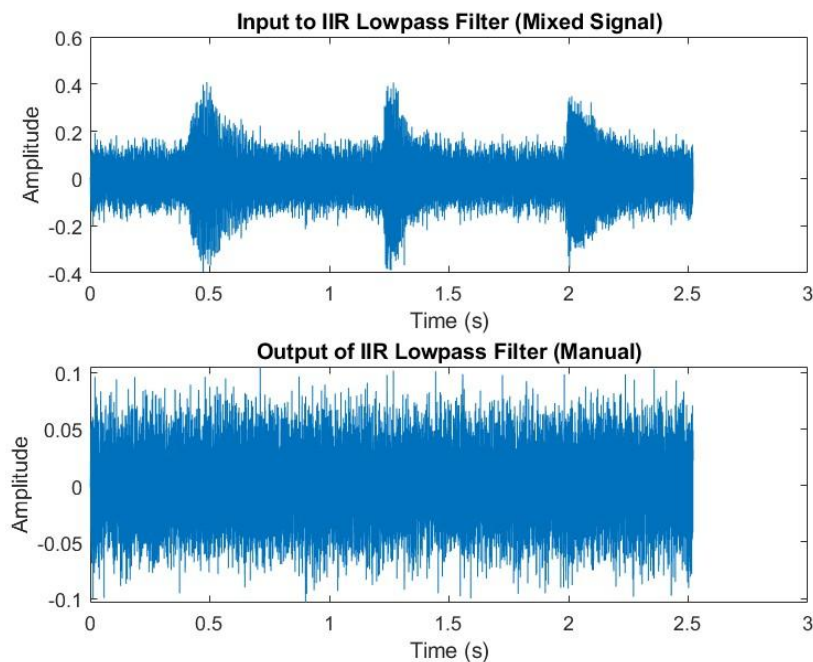


Figure 10: Time Domain Signal Before and After IIR Filtering

The filtered signal exhibits a clear demodulated waveform corresponding to the original modulation signal, while high-frequency oscillations from the carrier and noise are effectively removed. The FFT of the filtered signal was computed and plotted in Figure 11 below.

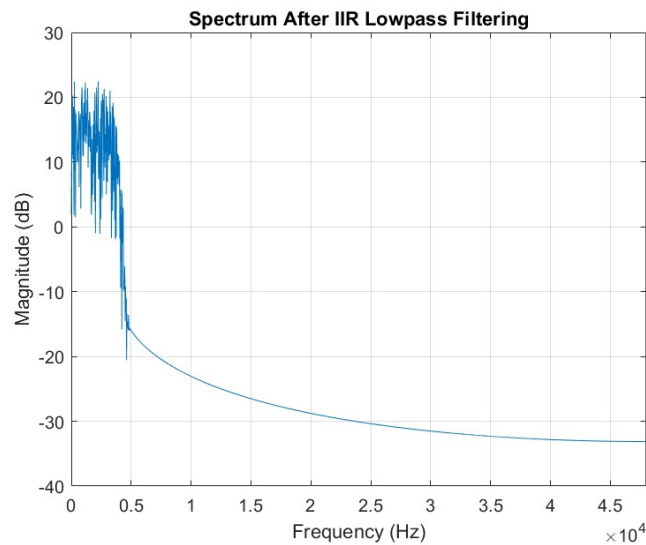


Figure 11: Spectrum of Signal After IIR Lowpass Filtering

Only frequency components up to 4 kHz remain, confirming that the lowpass filter has effectively removed unwanted high-frequency content while preserving the message signal.

Picture 4 below is the full code for task 4.

<pre> 235 % Task 4 - IIR Lowpass Filter (Butterworth) 236 237 signal = x_mix; 238 239 % Design Butterworth filter 240 order = 4; 241 fc_lp = 4000; 242 Wn = fc_lp/(Fs/2); 243 244 [b, a] = butter(order, Wn, 'low'); 245 246 % Verify frequency response manually 247 N_fft = 4096; 248 H = fft(b, N_fft) ./ fft(a, N_fft); 249 250 f_resp = (0:N_fft/2) * Fs / N_fft; 251 H_mag = 20*log10(abs(H(1:N_fft/2+1))) + eps; 252 253 figure; 254 plot(f_resp, H_mag); 255 xlabel('Frequency (Hz)'); 256 ylabel('Magnitude (dB)'); 257 title('4th-order Butterworth Lowpass Filter (Manual)'); 258 grid on; 259 xline([0 Fs/2]); 260 261 % Manual filtering using difference equation 262 y_iir = zeros(1, N); 263 264 for n = 1:N 265     acc_b = 0; 266     for k = 1:length(b) 267         if n - k + 1 &gt; 0 268             acc_b = acc_b + b(k) * signal(n - k + 1); 269         end 270     end 271 272     acc_a = 0; 273     for k = 1:length(a) 274         if n - k + 1 &gt; 0 275             acc_a = acc_a + a(k) * y_iir(n - k + 1); 276         end 277     end 278     y_iir(n) = acc_b - acc_a; 279 end 280 281 % Time-domain plots 282 figure; 283 subplot(2,1,1); 284 plot(t, signal); 285 title('Input to IIR Lowpass Filter (Mixed Signal)'); 286 xlabel('Time (s)'); ylabel('Amplitude'); 287 288 subplot(2,1,2); 289 plot(t, y_iir); 290 title('Output of IIR Lowpass Filter (Manual)'); 291 xlabel('Time (s)'); ylabel('Amplitude'); 292 293 % Frequency-domain plot 294 y_iir = fft(y_iir, N_fft); 295 Y_iir_mag = 20*log10(abs(Y_iir(1:N_fft/2+1))) + eps; 296 f_iir = (0:N_fft/2) * Fs / N_fft; 297 298 figure; 299 plot(f_iir, Y_iir_mag); 300 title('Spectrum After IIR Lowpass Filtering'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); 301 grid on; 302 xline([0 Fs/2]); 303 304 305 </pre>	<pre> % Input to IIR lowpass % cutoff frequency in Hz % normalized cutoff % one-sided frequency vector % FFT of filtered signal % one-sided magnitude % matching frequency vector </pre>
---	--

Picture 4: code for task 4

## Task 5– Audio signal output

The final stage of the demodulation process involves selecting the optimal carrier phase ( $\phi$ ) to maximise the amplitude and signal-to-noise ratio (SNR) of the demodulated message.

The carrier phase was adjusted in two stages, Coarse Phase and Fine Phase Search.

During the Coarse Phase Search, the phase was scanned over a coarse grid from 0 to  $\pi$  radians. For each phase, the bandpass-filtered signal from Task 2 was multiplied by a local carrier with that phase, followed by the manual IIR lowpass filtering (Task 4). The phase that produced the maximum RMS was selected as the coarse optimum.

During the Fine Phase Search, a finer search was performed in a small range ( $\pm 9^\circ$ ) around the coarse optimum. The same RMS metric was used to identify the final optimal phase. This two-stage approach ensures precise phase alignment for maximum demodulated signal strength.

The optimal carrier phase was determined to be:

$$\phi_{\text{best}} \approx 1.57 \text{ rad}$$

The bandpass-filtered signal was multiplied by the local carrier using the optimal phase, followed by IIR lowpass filtering to extract the final message signal. Figure 12 shows the time and frequency domain of the demodulated signal.

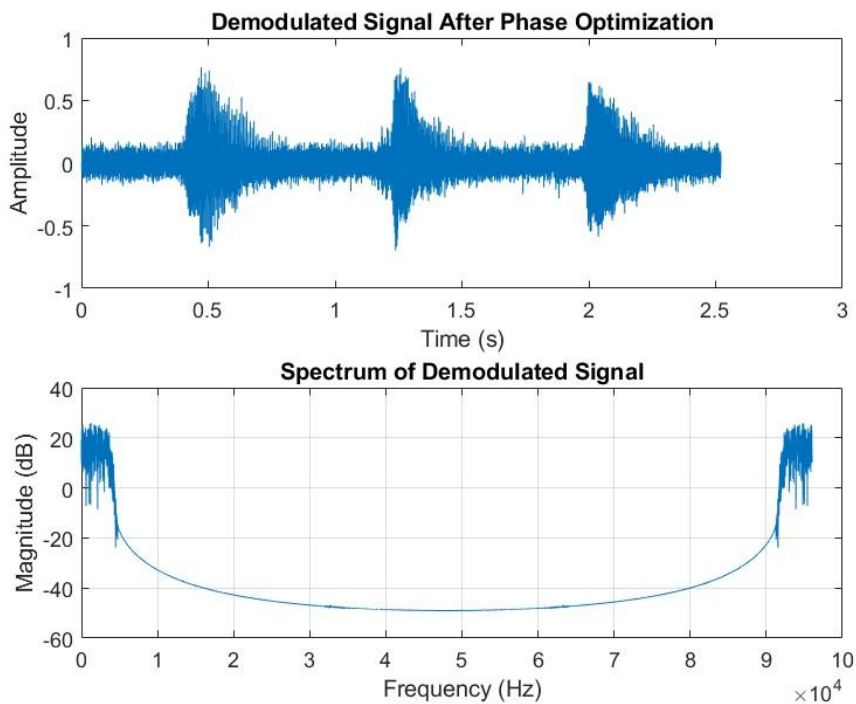


Figure 12: Time and Frequency Domains of Demodulated Signal

The spectrum of the demodulated signal was analysed to estimate the SNR. The speech band (300–3400 Hz) contains the desired message, while frequency components above 4 kHz represent noise. The estimated SNR was approximately -0.02 dB, confirming that the lowpass filtering and phase optimisation effectively enhanced the message signal quality.

The demodulated signal was finally normalised and played back using MATLAB's sound() function, allowing auditory verification of the three-letter spoken message. The signal was also saved as demodulated\_message\_Vasiliki\_Savva.wav for submission and review. The three letters are: **Y, J, V**.

Pictures 5 and 6 are the complete code for task 5.

```

386 % Task 5 - Improved Phase Adjustment and Playback
387 signal_lp = y_lir; % Output of manual IIR lowpass (Task 4)
388 y_final = zeros(size(signal_lp));
389
390 % Stage 1: Coarse Phase Search
391 phi_coarse = linspace(0, pi, 50); % coarse 50-point grid
392 rms_coarse = zeros(size(phi_coarse));
393
394 for p = 1:length(phi_coarse)
395     phi = phi_coarse(p);
396     local_carrier = cos(2*pi*fc_esttt_bp + phi);
397     mixed_signal = y_blk .* local_carrier; % y_blk = filtered bandpass from Task 2
398
399 % Apply same IIR Filter
400 % Manual IIR filtering (inline)
401 y_temp = zeros(1,N);
402
403 for n = 1:N
404     acc_b = 0;
405     for k = 1:length(b)
406         if (n - k + 1) > 0
407             acc_b = acc_b + b(k)*mixed_signal(n - k + 1);
408         end
409     end
410
411     acc_a = 0;
412     for k = 2:length(a)
413         if (n - k + 1) > 0
414             acc_a = acc_a + a(k)*y_temp(n - k + 1);
415         end
416     end
417
418     y_temp(n) = acc_b - acc_a;
419 end
420
421 rms_coarse(p) = sqrt(mean(y_temp.^2)); % RMS metric
422 end
423
424 [~, idx_best_coarse] = max(rms_coarse);
425 phi_best_coarse = phi_coarse(idx_best_coarse);
426
427 % Stage 2: Fine Phase Search Around Coarse Optimum
428 phi_fine = linspace(max(0, phi_best_coarse-pi/20), min(pi, phi_best_coarse+pi/20), 41); % +/- 9° around coarse
429 rms_fine = zeros(size(phi_fine));
430
431 for p = 1:length(phi_fine)
432     phi = phi_fine(p);
433     local_carrier = cos(2*pi*fc_esttt_bp + phi);
434     mixed_signal = y_blk .* local_carrier;
435
436 % Manual IIR filtering (inline)
437 y_temp = zeros(1,N);
438
439 for n = 1:N
440     acc_b = 0;
441     for k = 1:length(b)
442         if (n - k + 1) > 0
443             acc_b = acc_b + b(k)*mixed_signal(n - k + 1);
444         end
445     end
446
447     acc_a = 0;
448     for k = 2:length(a)
449         if (n - k + 1) > 0
450             acc_a = acc_a + a(k)*y_temp(n - k + 1);
451         end
452     end
453
454     y_temp(n) = acc_b - acc_a;
455 end
456
457 rms_fine(p) = sqrt(mean(y_temp.^2));
458 end
459
460 [~, idx_best_fine] = max(rms_fine);
461 phi_best = phi_fine(idx_best_fine);
462
463 % Apply best phase and filter
464 local_carrier_best = cos(2*pi*fc_esttt_bp + phi_best);
465 mixed_signal_best = y_blk .* local_carrier_best;
466 % Apply best phase and filter
467 y_final = zeros(1,N);
468
469 for n = 1:N
470     acc_b = 0;
471     for k = 1:length(b)
472         if (n - k + 1) > 0
473             acc_b = acc_b + b(k)*mixed_signal_best(n - k + 1);
474         end
475     end
476
477     acc_a = 0;
478     for k = 2:length(a)
479         if (n - k + 1) > 0
480             acc_a = acc_a + a(k)*y_final(n - k + 1);
481         end
482     end
483
484     y_final(n) = acc_b - acc_a;
485 end
486
487 fprintf('Optimal phase selected: %.4f rad (%.2f degrees)\n', phi_best, phi_best*180/pi);
488
489 % SNR check
490 signal_band = 300:3400; % speech band
491 ffft_y = abs(fft(y_final));
492 NFFT = length(ffft_y);
493 freq_axis = (0:NFFT-1)/NFFT*Fs;
494 speech_idx = freq_axis >= signal_band(1) & freq_axis <= signal_band(2);
495 noise_idx = freq_axis > 4000 & freq_axis <= Fs/2; % above speech band
496 SNR_est = 10*log10(sum(ffft_y(speech_idx).^2)/sum(ffft_y(noise_idx).^2));
497 fprintf('Estimated SNR: %.2f dB\n', SNR_est);
498
499 % Plot time and frequency domain of demodulated signal
500 figure;
501 subplot(2,1,1); plot(t_bp, mixed_signal_best);
502 title('Demodulated Signal After Phase Optimization'); xlabel('Time (s)'); ylabel('Amplitude');
503
504 subplot(2,1,2);
505 v_final_fft = fft(y_final, 8192);
506 f_fft = linspace(0, Fs, 8192);
507 plot(f_fft, 20*log10(abs(v_final_fft)/length(v_final_fft)));
508 title('Spectrum of Demodulated Signal'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
509 grid on;
510
511 % Normalize and Playback
512 x_playback = y_final / max(abs(y_final)) * 0.9;
513 sound(x_playback, Fs);
514 pause(length(x_playback)/Fs + 0.5);
515 fprintf('Audio playback complete.\n');
516
517 % Save audio file
518 audiowrite('demodulated_message_Vasiliki_Savva.wav', x_playback, Fs);
519 fprintf('Audio saved as demodulated_vasiliki_savva.wav\n');
520

```

Picture 5: code for task 5 – part 1

```

367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Picture 6: code for task 5- part 2

Link to full code: <https://github.com/vassavva/Signal-Processing>

