

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.

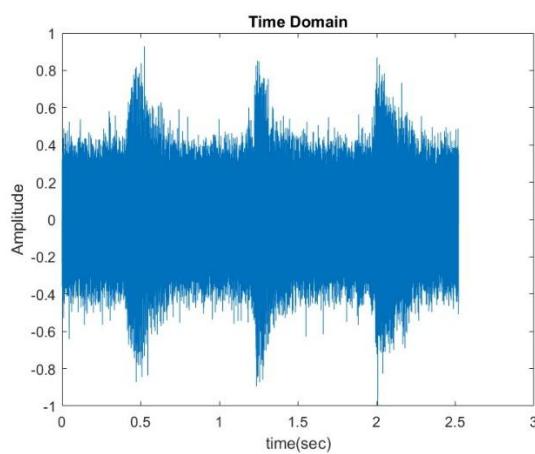


Figure 1: Time Domain of Received Signal

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.

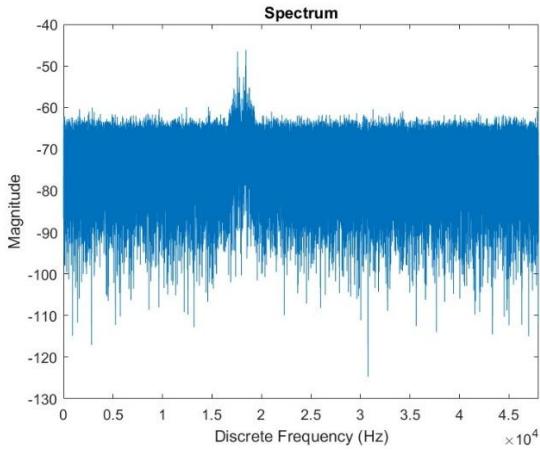


Figure 2: Frequency Domain

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.

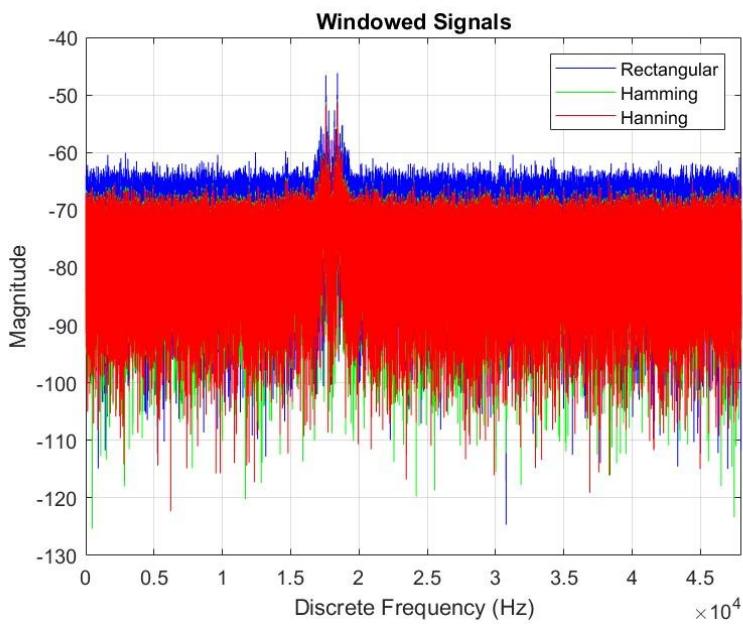


Figure 3: Windowed Signals for comparison

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, $f_{\text{min}} = 14\text{kHz}$ and $f_{\text{max}} = 22\text{kHz}$, 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
clc;
close all;
clear all;

[x,fs] = audioread('Vasiliiki Savva.wav'); % Extract signal from audio
T = 1/fs;
N = length(x);
t = 0:t:(N-1)*T; % Calculates N using the duration of the signal
%Generate discrete time values (ns)

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

xplot Magnitude FFT
X_fft = fft(x);
k = (0:N-1)';
f = k*(T/N); % Normalizes Frequency

X_norm abs(X_fft/N);
X_norm = abs(X_norm);
figure; plot(f,X_norm);
xlim([0 fs/2]);
xlabel('Discrete Frequency (Hz)'); ylabel('Magnitude'); title('Frequency Spectrum');

%Generate N points window
ham_w = 0.5*(1-(cos(2*pi*k/(N-1)));
rec_w = 0.5*(1-(cos(2*pi*k/(N-1)));
ham_w = (0.5+0.4j)*cos(2*pi*(0:N-1)/(N-1));
rec_w = 0.5+0.4j; % Hamming Window
% Rectangular Window

%Apply window to signal
xH = x.*ham_w; %Hamming
xR = x.*rec_w; %Rectangular
X_norm_h = abs(fft(xH));
X_norm_r = abs(fft(xR));
X_db=20*log10(X_norm_h);
X_db=20*log10(X_norm_r);

xH = x.*rec_w; %Rectangular
X_fft_r = fft(xR);
X_norm_r = abs(X_fft_r)/N;
X_db=20*log10(X_norm_r);

xH = x.*ham_w; %Hamming
X_fft_h = fft(xH);
X_norm_h = abs(X_fft_h)/N;
X_db=20*log10(X_norm_h);

%Plot Windowed Signals for comparison
figure;
plot(xH,X_db,'b');
hold on;
plot(xR,X_db,'g');
plot(xH,X_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;
fmax = fc + BW/2;
% use an 8kHz bandwidth
```

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 $f_{\text{min}} = 14\text{kHz}$ and $f_{\text{max}} = 22\text{kHz}$ 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.

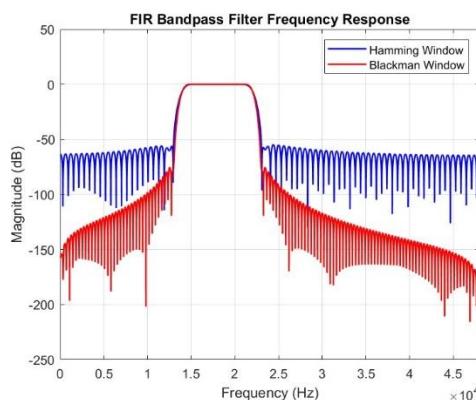


Figure 4: Frequency Response of Hamming and Blackman FIR Bandpass Filter

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

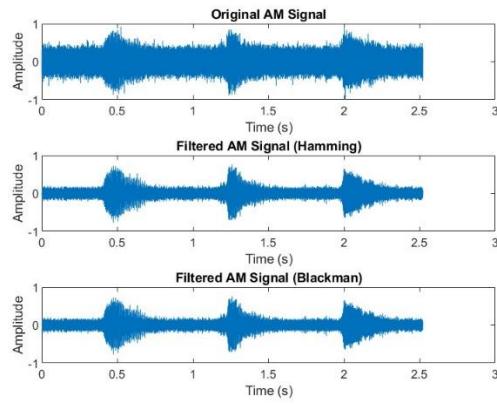


Figure 5: Time-Domain Comparison of Original and FIR-Filtered Signals

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.

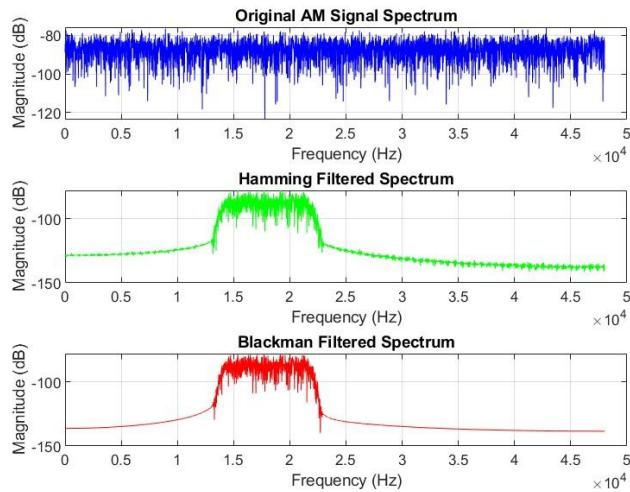


Figure 6: Frequency-Domain Comparison of Original and FIR-Filtered Signals

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

```

% Task 2 - Bandpass Filter
df = 2000; % transition width
% Filter length
Nnum = ceil(3.3*f0/df); if mod(Nnum,2)==0, Nnum=Nnum+1; end
Nb1c = ceil(5.5*f0/df); if mod(Nb1c,2)==0, Nb1c=Nb1c+1; end
% Normalized frequencies
Fc1 = fmin/f0;
Fc2 = fmax/f0;
% Ideal impulse responses
% Hamming
n_h = 0:Nnum-1;
a_h = (Nnum-1)/2;
h_idel_ham = 2^8Fc2*sinc(2^8Fc2*(n_h - a_h)) - 2^8Fc1*sinc(2^8Fc1*(n_h - a_h));
% Blackman
n_b = 0:Nb1c-1;
a_b = (Nb1c-1)/2;
h_idel_blk = 2^8Fc2*sinc(2^8Fc2*(n_b - a_b)) - 2^8Fc1*sinc(2^8Fc1*(n_b - a_b));
% Windows (manual)
w_ham = 0.54 - 0.46*cos(2*pi*n_h/(Nnum-1));
w_blk = 0.42 - 0.5*cos(2*pi*n_b/(Nb1c-1)) + 0.08*cos(4*pi*n_b/(Nb1c-1));
% Apply windows
h_ham = h_idel_ham .* w_ham;
h_blk = h_idel_blk .* w_blk;
% FFT
N_fft = 8192;
H_ham = fft(h_ham, N_fft);
H_blk = fft(h_blk, N_fft);
% Frequency response
f_resp = (0:N_fft/2) * f0 / N_fft;
H_ham_mag = 20*log10(abs(H_ham(1:N_fft/2+1))) + eps;
H_blk_mag = 20*log10(abs(H_blk(1:N_fft/2+1))) + eps;
% Plot
figure;
plot(f_resp, H_ham_mag, 'b', 'LineWidth', 1.2); hold on;
plot(f_resp, H_blk_mag, 'r', 'LineWidth', 1.2);
title('IR Bandpass Filter Frequency Response');
legend('Hamming window', 'Blackman window');
grid on; xlim([0 f0/2]);
% Manual FIR convolution
y_ham = zeros(1, N);
y_blk = zeros(1, N);
% Hamming filter
for n = 1:N
    acc = 0;
    for k = 1:Nnum
        if (n - k + 1) > 0
            acc = acc + h_idel_ham(k)*x(n - k + 1);
        end
    end
    y_ham(n) = acc;
end
% Blackman filter
for n = 1:N
    acc = 0;
    for k = 1:Nb1c
        if (n - k + 1) > 0
            acc = acc + h_idel_blk(k)*x(n - k + 1);
        end
    end
    y_blk(n) = acc;
end
% Time-domain comparison
% Plot
figure;
subplot(3,1,1); plot(t,x); title('Original AM Signal'); xlabel('Time (s)'); ylabel('Amplitude');
subplot(3,1,2); plot(t,y_ham); title('Filtered AM Signal (Hamming)'); xlabel('Time (s)'); ylabel('Amplitude');
subplot(3,1,3); plot(t,y_blk); title('Filtered AM Signal (Blackman)'); xlabel('Time (s)'); ylabel('Amplitude');
% Frequency-domain comparison
% Compute FFT and one-sided magnitude in dB
X = 20*log10(abs(fft(x, N_fft))/N) + eps; % Original signal
Yh = 20*log10(abs(fft(y_ham, N_fft)/N) + eps); % Hamming filtered
Yb = 20*log10(abs(fft(y_blk, N_fft)/N) + eps); % Blackman filtered
% Keep only positive frequencies
X = X(1:N_fft/2+1);
Yh = Yh(1:N_fft/2+1);
Yb = Yb(1:N_fft/2+1);
% Plot
figure;
subplot(3,1,1); plot(f_resp, X, 'k'); title('Original AM Signal Spectrum'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); grid on;
subplot(3,1,2); plot(f_resp, Yh, 'g'); title('Hamming Filtered Spectrum'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)'); grid on;
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.

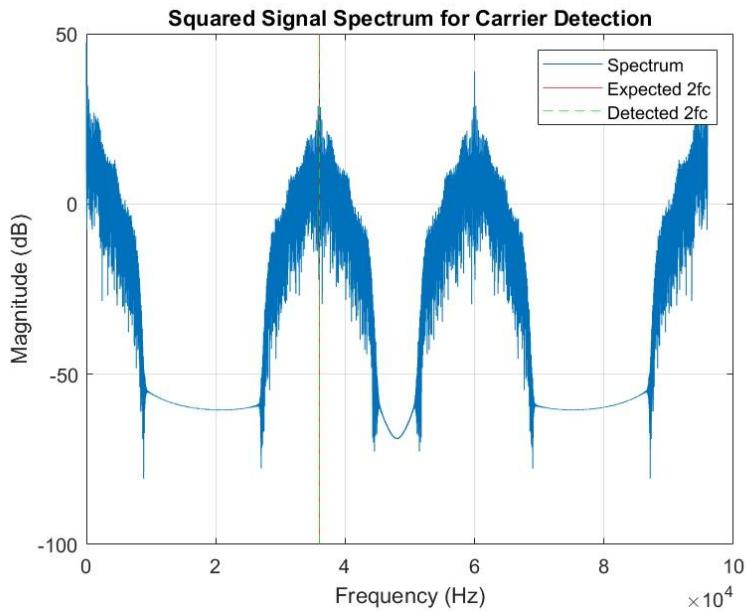


Figure 7: Squared Signal Spectrum for Carrier Detection

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.

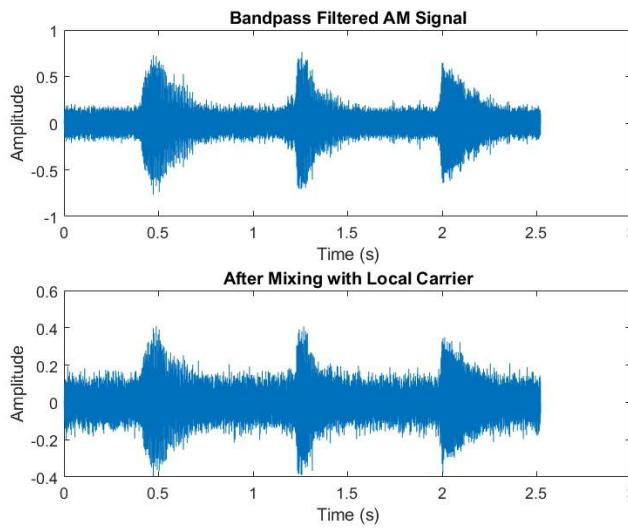


Figure 8: Time-Domain Signals Before and After Mixing

Picture 3 shows the complete code for task 3.

```

173 % Task 3 - Carrier Recovery & Mixing
174
175 x_bp = y_blk;
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 DfC
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 index 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_2fc = f_2fc*(fs/N_fft);
203
204 % Compute estimated carrier frequency
205 fc_est = f_2fc / 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(fft(x_mix, N_fft)));
223 f_fft = linspace(0, fs, N_fft);
224 f_sq = f_fft;
225 plot(f_sq, 20*log10(abs(Xsq)*eps)); hold on;
226 xline(2*expected_fc, 'r');
227 xline(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 4th 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.

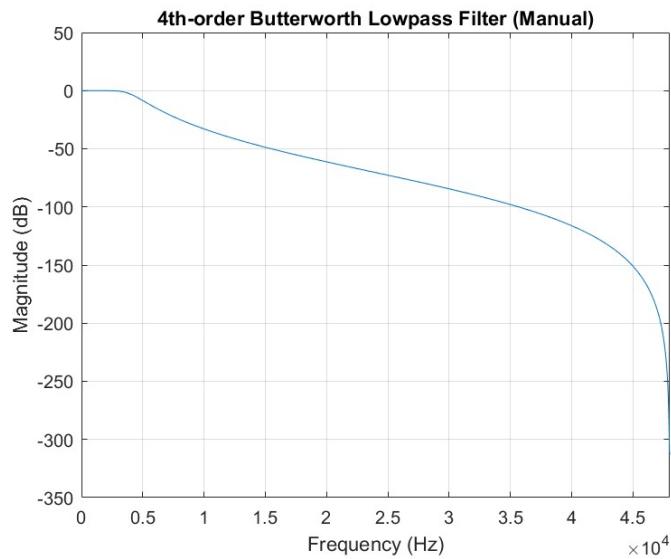


Figure 9: Frequency Response of 4th-order Butterworth Lowpass Filter

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.

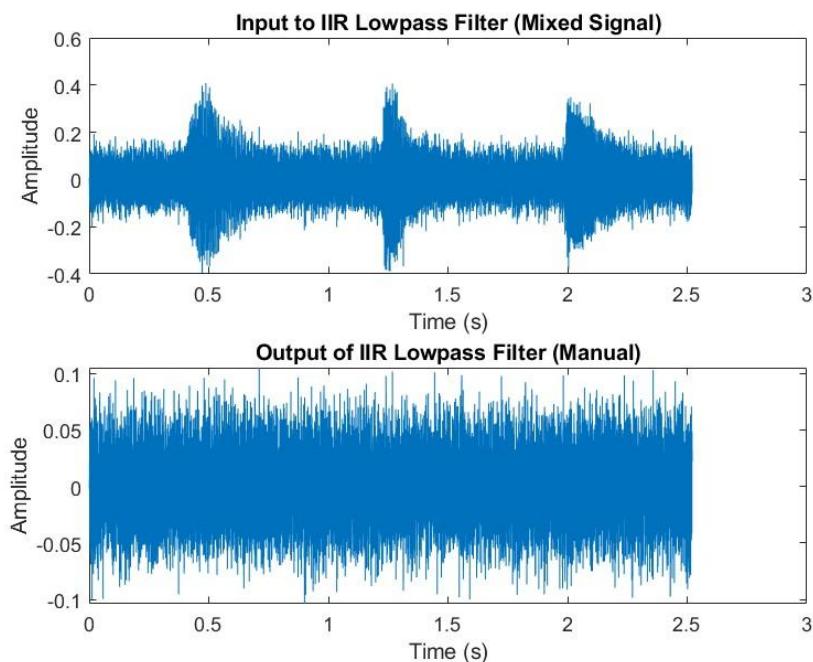


Figure 10: Time Domain Signal Before and After IIR Filtering

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.

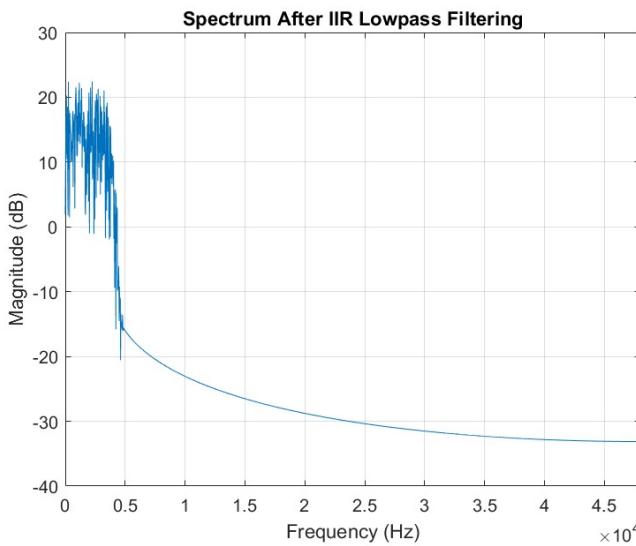


Figure 11: Spectrum of Signal After IIR Lowpass Filtering

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.

```

225 % Task 4 - IIR Lowpass Filter (Butterworth)
226
227 signal = x_mix;
228
229 % Design Butterworth filter
230 order = 4;
231 fc_lp = 4000;
232 wn = fc_lp/(fs/2);
233
234 [b, a] = butter(order, wn, 'low');
235
236 % Verify Frequency response manually
237 N_fft = 4096;
238 H = fft(b, N_fft) ./ fft(a, N_fft);
239
240 f_resp = (0:N_fft/2) * fs / N_fft;
241 H_mag = 20*log10(abs(H(1:N_fft/2+1))) + eps;
242
243 figure;
244 plot(f_resp, H_mag);
245 xlabel('Frequency (Hz)');
246 ylabel('Magnitude (dB)');
247 title('4th-order Butterworth Lowpass Filter (Manual)');
248 grid on;
249 xlim([0 fs/2]);
250
251 % Manual filtering using difference equation
252 y_ilr = zeros(1, N);
253
254 for n = 1:N
255 acc_b = 0;
256 for k = 1:length(b)
257 if n - k + 1 > 0
258 acc_b = acc_b + b(k) * signal(n - k + 1);
259 end
260 end
261
262 acc_a = 0;
263 for k = 2:length(a)
264 if n - k + 1 > 0
265 acc_a = acc_a + a(k) * y_ilr(n - k + 1);
266 end
267 end
268
269 y_ilr(n) = acc_b - acc_a;
270 end
271
272 % Time-domain plots
273 figure;
274 subplot(2,1,1);
275 plot(t, signal);
276 title('Input to IIR Lowpass Filter (Mixed Signal)');
277 xlabel('Time (s)'); ylabel('Amplitude');
278
279 subplot(2,1,2);
280 plot(t, y_ilr);
281 title('Output of IIR Lowpass Filter (Manual)');
282 xlabel('Time (s)'); ylabel('Amplitude');
283
284 % Frequency-domain plot
285 y_ilr = fft(y_ilr, N_fft);
286 y_ilr_mag = 20*log10(abs(y_ilr(1:N_fft/2+1)) + eps);
287 f_ilr = (0:N_fft/2) * fs / N_fft;
288
289 figure;
290 plot(f_ilr, y_ilr_mag);
291 title('Spectrum After IIR Lowpass Filtering'); xlabel('Frequency (Hz)');
292 ylabel('Magnitude (dB)');
293 grid on;
294 xlim([0 fs/2]);
295
296

```

Picture 4: code for task 4

Task 5– Audio signal output

The final stage of the demodulation process involves selecting the optimal carrier phase (ϕ) 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 π 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.

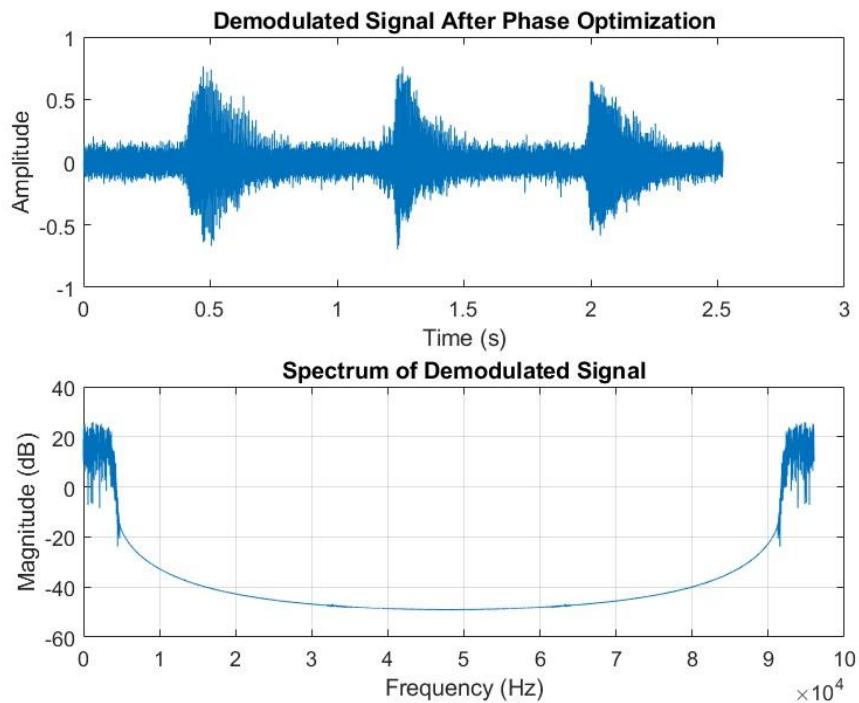


Figure 12: Time and Frequency Domains of Demodulated Signal

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.

```

306 % Task 5 – Improved Phase Adjustment and Playback
307
308 signal_lp = y_llir; % Output of manual IIR lowpass (Task 4)
309 y_final = zeros(size(signal_lp));
310
311 % Stage 1: Course Phase Search
312 phi_coarse = linspace(0, pi, 58);
313 rms_coarse = zeros(size(phi_coarse));
314
315 for p = 1:length(phi_coarse)
316     phi = phi_coarse(p);
317     local_carrier = cos(2*pi*f_c_est*t_bp + phi);
318     mixed_signal = y_lllk .* local_carrier;
319
320     % Apply same IIR Filter
321     % Manual IIR filtering (inline)
322     y_temp = zeros(1,N);
323
324     for n = 1:N
325         acc_b = 0;
326         for k = 1:length(b)
327             if (n - k + 1) > 0
328                 acc_b = acc_b + b(k)*mixed_signal(n - k + 1);
329             end
330         end
331
332         acc_a = 0;
333         for k = 2:length(a)
334             if (n - k + 1) > 0
335                 acc_a = acc_a + a(k)*y_temp(n - k + 1);
336             end
337         end
338
339         y_temp(n) = acc_b - acc_a;
340     end
341
342     rms_coarse(p) = sqrt(mean(y_temp.^2));
343 end
344
345 [~, idx_best_coarse] = max(rms_coarse);
346 phi_best_coarse = phi_coarse(idx_best_coarse);
347
348 % Stage 2: Fine Phase Search Around Coarse Optimum
349 phi_fine = linspace(max(0, phi_best_coarse*pi/20), min(pi, phi_best_coarse+pi/20), 41); % +/- 9° around coarse
350 rms_fine = zeros(size(phi_fine));
351
352 for p = 1:length(phi_fine)
353     phi = phi_fine(p);
354     local_carrier = cos(2*pi*f_c_est*t_bp + phi);
355     mixed_signal = y_lllk .* local_carrier;
356
357     % Manual IIR filtering (inline)
358     y_temp = zeros(1,N);
359
360     for n = 1:N
361         acc_b = 0;
362         for k = 1:length(b)
363             if (n - k + 1) > 0
364                 acc_b = acc_b + b(k)*mixed_signal(n - k + 1);
365             end
366         end
367     end
368
369     rms_fine(p) = sqrt(mean(y_temp.^2));
370 end
371
372 % coarse 58-point grid
373 % y_lllk = filtered bandpass from Task 2

```

Picture 5: code for task 5 – part 1

```

368 end
369 acc_a = 0;
370 for k = 1:length(a)
371 if (n - k + 1) > 0
372 acc_a = acc_a + a(k)*y_temp(n - k + 1);
373 end
374
375
376 y_temp(n) = acc_b - acc_a;
377
378 rms_fine(p) = sqrt(mean(y_temp.^2));
379
380 [~, idx_best_fine] = max(rms_fine);
381 phi_best = p_idx(idx_best_fine);
382
383 % Apply best phase and filter
384 local_carrier_best = cos(2*pi*f_c_est*t_hp + phi_best);
385 mixed_signal_best = y_llk.*local_carrier_best;
386
387 % Apply best phase and filter
388
389 y_final = zeros(1,N);
390
391 for n = 1:N
392 acc_b = 0;
393 for k = 1:length(b)
394 if (n - k + 1) > 0
395 acc_b = acc_b + b(k)*mixed_signal_best(n - k + 1);
396 end
397
398 acc_a = 0;
399 for k = 1:length(a)
400 if (n - k + 1) > 0
401 acc_a = acc_a + a(k)*y_final(n - k + 1);
402 end
403
404 end
405
406 y_final(n) = acc_b - acc_a;
407
408 end
409
410 fprintf('Optimal phase selected: %.4f rad (%.2f degrees)\n', phi_best, phi_best*180/pi);
411
412 % SNR check
413 signal_band = 300:3400;
414 f_FFT = abs(fft(y_final));
415 NFFT = length(f_FFT);
416 Freq_axis = (0:NFFT-1)/NFFT*f_s;
417 speech_ids = freq_axis <= signal_band(1) & freq_axis > signal_band(2);
418 noise_ids = freq_axis > 4800 & freq_axis <= fs/2; % above speech band
419 SNRest = 10*log10(sum(fft(y_speech_ids).^2)/sum(fft(y_noises_ids).^2));
420 fprintf('Estimated SNR: %.2f dB\n', SNRest);
421
422 % Split time and frequency domain of demodulated signal
423 figure;
424 subplot(2,1,1); plot(t_hp, mixed_signal_best);
425 title('Demodulated Signal After Phase Optimization'); xlabel('Time (s)') ylabel('Amplitude');
426
427 subplot(2,1,2);
428 y_final_idc = fft(y_final, R192);
429 f_fft = linspace(0, fs/2, R192);
430 plot(f_fft, 20*log10(abs(fft(y_final_idc)))); % Magnitude of final FFT(signed);
431 title('Spectrum of Demodulated Signal'); xlabel('Frequency (Hz)'); ylabel('Magnitude (dB)');
432 grid on;
433
434 % Normalise and Playback
435 x_playback = y_final / max(abs(y_final)) * 0.9;
436 sound(x_playback, fs);
437 pause(length(x_playback)/fs + 0.5);
438 fprintf('Audio playback complete.\n');
439
440 % Save audio file
441 audiowrite('demodulated_message_Vasiliki_Sava.wav', x_playback, fs);
442 fprintf('Audio saved as demodulated_message_Vasiliki_Sava.wav\n');

```

Picture 6: code for task 5- part 2

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

