Orkun İbrahim Kök

22003656

EEE431 Assignment #2

Part 1.a)

For a process to be WSS, it needs to satisfy two conditions: Mean is constant, Autocorrelation function $R_x(t+\tau, t)$ does not depend on t.

Because V_n is zero mean, the mean function can be found as follows: $E[W_n] = 0.9*E[W_{n-1}]$. Because the coefficient of this autoregressive process is less than 1 (0.9), the initial condition is assumed to be zero. Therefore the mean of W_n is 0, which is a constant. This checks the first condition.

For the second condition, I need to determine the autocorrelation function of this autoregressive process. Fig. 1 shows the autocorrelation function of a autoregressive process with coefficient less than 1.

$$\mu = \frac{c}{1-\varphi}.$$

In particular, if $c = 0$, then the mean is 0.

The variance is

$$\operatorname{var}(X_t) = \operatorname{E}(X_t^2) - \mu^2 = \frac{\sigma_\varepsilon^2}{1-\varphi^2},$$

where σ_ε is the standard deviation of ε_t. This can be shown by noting that

$$\operatorname{var}(X_t) = \varphi^2 \operatorname{var}(X_{t-1}) + \sigma_\varepsilon^2,$$

and then by noticing that the quantity above is a stable fixed point of this relation.

The autocovariance is given by

$$B_n = \operatorname{E}(X_{t+n} X_t) - \mu^2 = \frac{\sigma_\varepsilon^2}{1-\varphi^2}\ \varphi^{|n|}.$$

Fig. 1: Autocorrelation function for an autoregressive process

From Fig. 1, the autocorrelation function is derived as follows:

$$R_W(t+\tau, t) = \frac{1}{1-0.9^2} * 0.9^{|\tau|}$$

As it can be seen, the autocorrelation function only depends on the time difference, which checks the second condition also. Therefore, it can be concluded that this autoregressive process is WSS.

Part 1.b)

For this part, I generated the realizations and plotted a small portion (100 length) realizations.

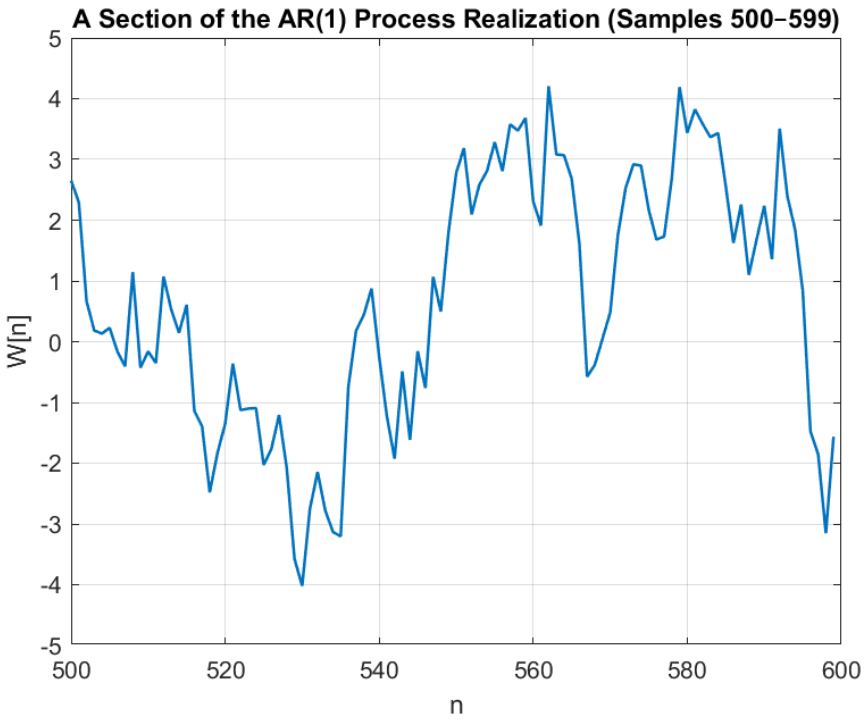

Fig. 2: Generated Realizations from sample 500 to 599

Part 1.c)

In this part, I filtered the process. The below figure shows the realization vs filtered output. Again in this part samples from 500 to 599 are plotted.

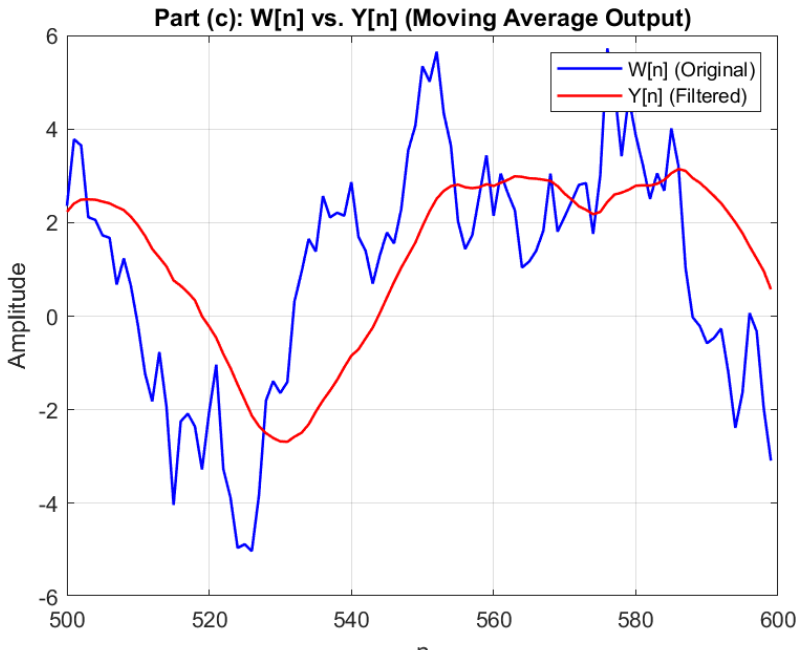

Fig. 3: Realization vs. Filtered output

Part 1.d)

In this part, the PSDs of W_n and Y_n are estimated using pwelch function. For the estimated magnitude response of the system the sqrt ratio of PSDs are used, equation is given below.

$$|H(f)| \sqrt{\frac{PSD_Y}{PSD_W}}$$

The estimated and analytically found magnitude responses started to resemble each other more as frequency increases.

$$Y_n = \sum_{k=0}^{19} h[k] * W_{n-k}$$

From the above equation, h[k] is found as 1/20 for $0 \leq k \leq 19$. Taking Fourier transform of it will give the analytical magnitude response. The corresponding derivations can be found in below equations.

$$H(e^{jw}) = \sum_{k=0}^{19} \frac{1}{20} * e^{-jwk}$$

$$H(e^{jw}) = \frac{1}{20} * \frac{1 - e^{-j20w}}{1 - e^{-jw}}$$

Using Euler's identity, the equation can be simplified further:

$$Euler's\ identity: 1 - e^{-jwN} = e^{-\frac{j(N-1)w}{2}} * 2jsin(\frac{Nw}{2})$$

$$H(e^{jw}) = \frac{1}{20} * e^{-\frac{jw19}{2}} * \frac{\sin(10w)}{\sin\left(\frac{w}{2}\right)}$$

From the above equation, the magnitude is found as follows:

$$\left|H(e^{jw})\right| = \frac{1}{20} * \left|\frac{\sin(10w)}{\sin\left(\frac{w}{2}\right)}\right|$$

This is a sinc shaped function.

The PSDs of W_n and Y_n can be seen in the below figure

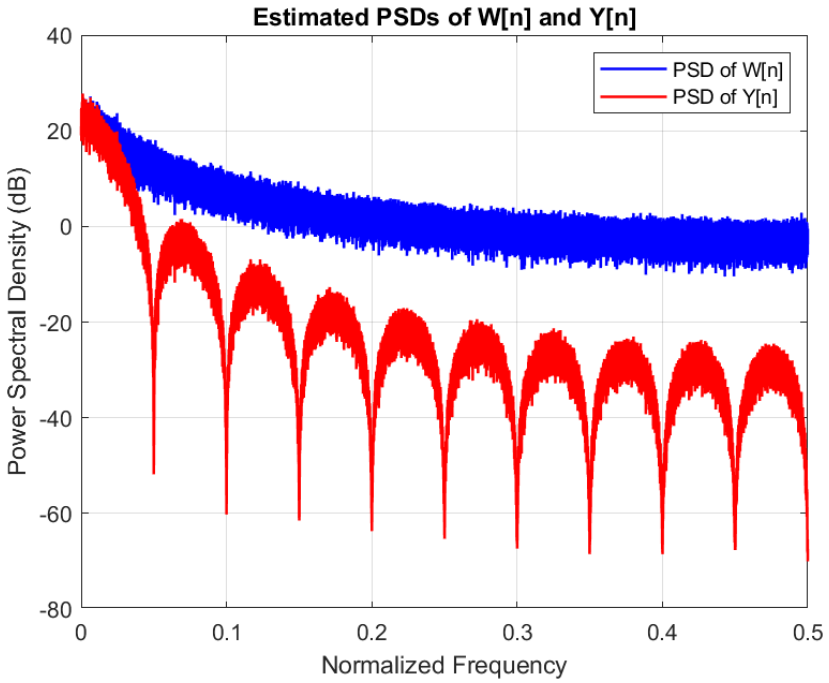

Fig. 4: Estimated PSDs

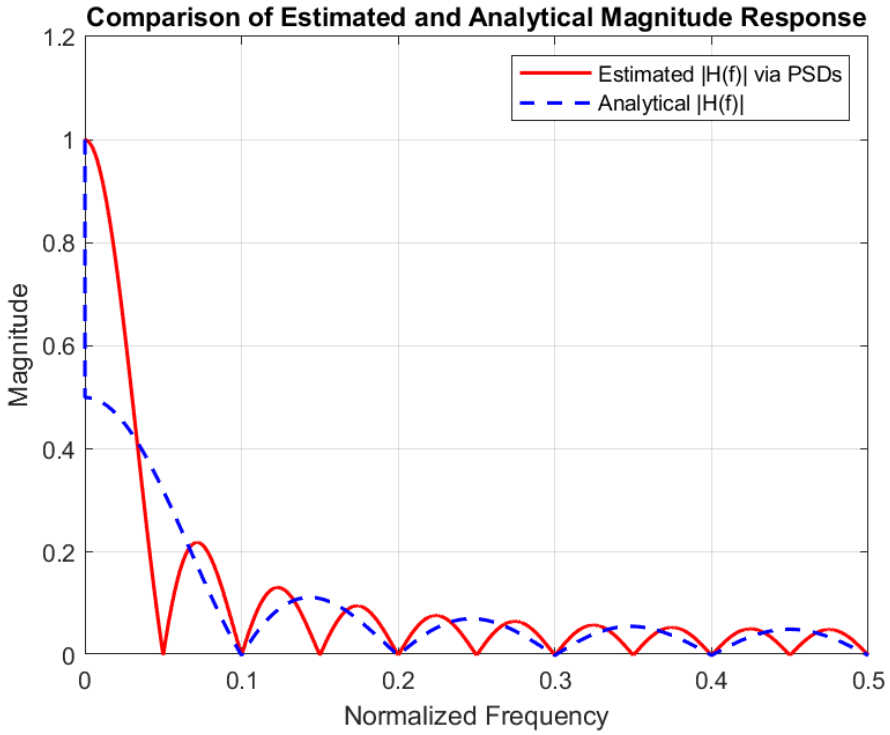

Fig. 5: Comparison of estimated and analytical magnitude responses of the filter

As it can be seen from Fig. 5, the blue dashed lines resemble to magnitude of a sinc function, which is given in Fig. 6.

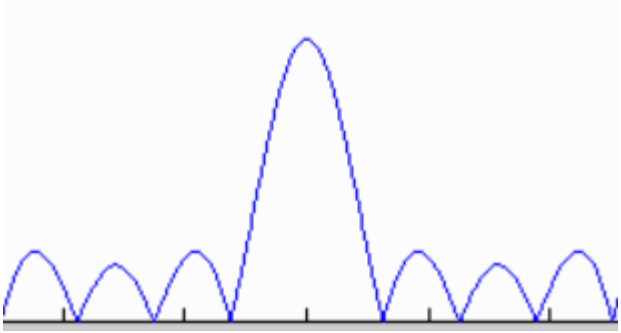

Fig. 6: Plot of magnitude of sinc function

Part 1.e)

In this part, W_n is changed with White Gaussian Noise. This time, White Gaussian Noise is given as input to the system. Fig. 7 shows the realization of White Gaussian Noise (samples from 500 to 599), and the response of the filter.

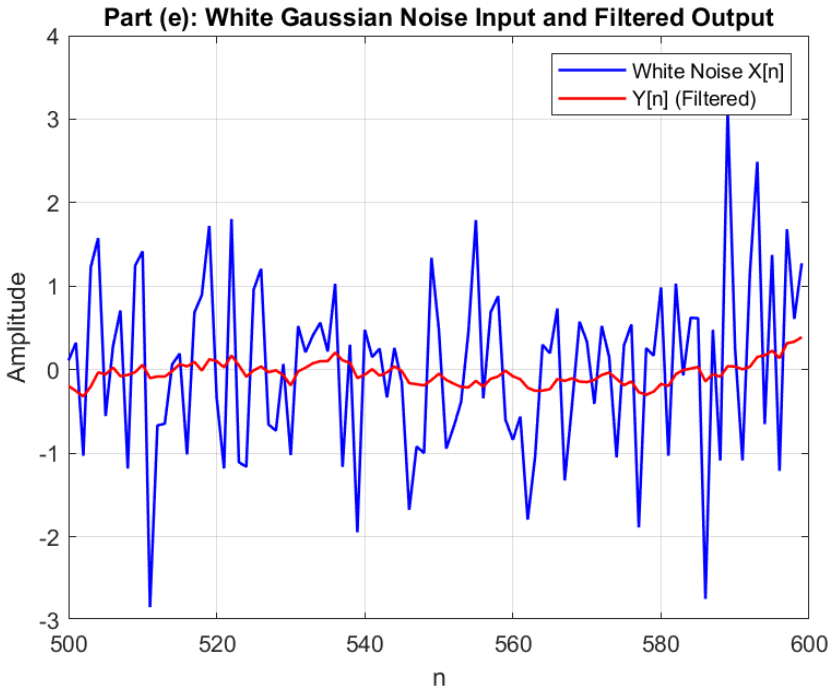

Fig. 7: White Gaussian Noise as input to the system

For the PSDs, same procedure is followed again. Fig. 8 compares the PSDs of W_n and White Gaussian Noise.

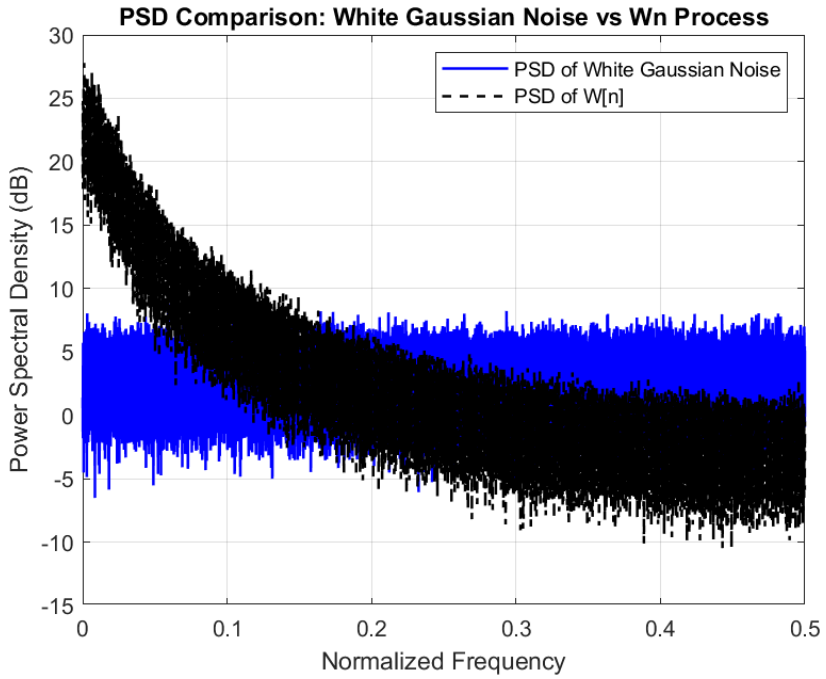

Fig. 8: PSD comparison of W_n vs. White Gaussian Noise

As it can be seen, the PSD of White Gaussian Noise is more shaped like a constant value line, whereas PSD of W_n decreases as frequency increases.

Part 2.a)

In this part, it is asked to modulate m(t) with DSB-SC using a carrier signal c(t).

Below figures shows m(t), c(t), and modulated signal respectively.

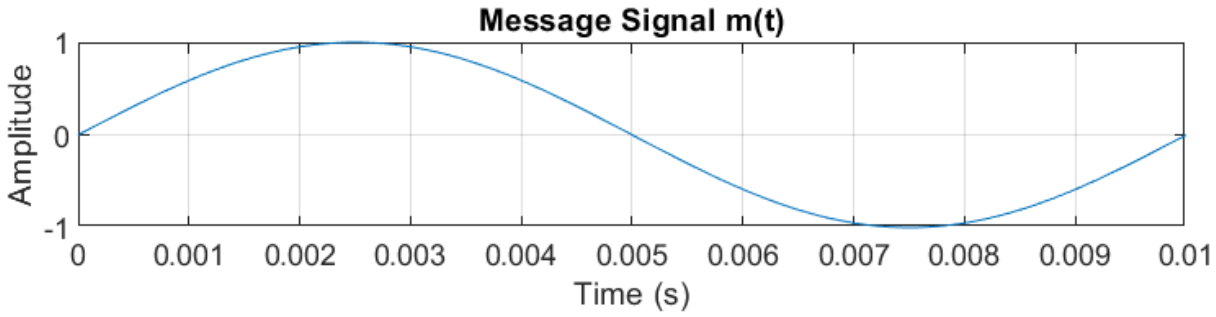

Fig. 9: m(t) signal

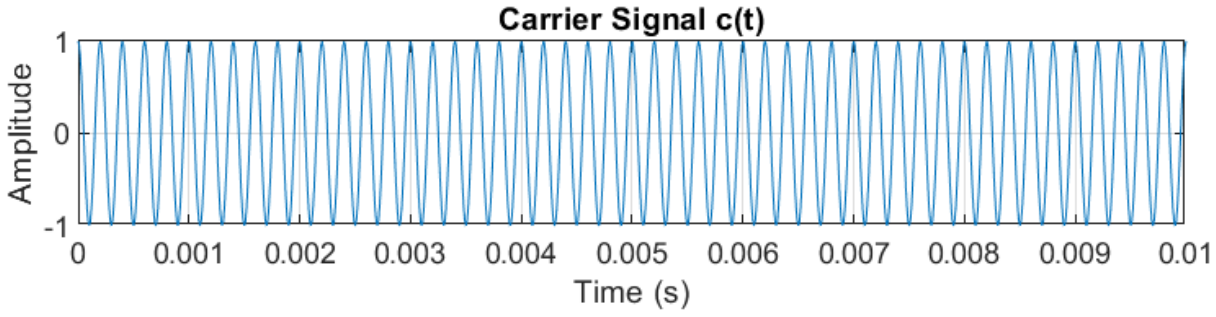

Fig. 10: c(t) signal

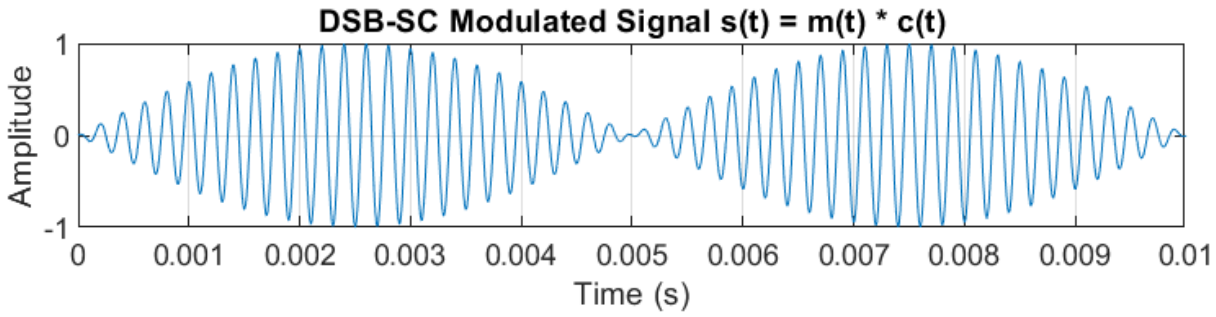

*Fig. 11: Modulated signal: m(t) * c(t)*

Part 2.b)

In this part, a coherent detector is implemented. The coherent detector is in the following form:

$$x(t) = \cos(2\pi(f_c + \Delta)t)$$

The Δ values are {0, 50, 100, 500, 1000} respectively.

Using all these values one by one and plotting the corresponding modulated signals on top, Fig. 12 shows the effect of Δ.

Also, low-pass filtered the output of the coherent detector to get rid of the high frequency term to reduce distortion on the graph even though the question does not asks for any low-pass filtering.

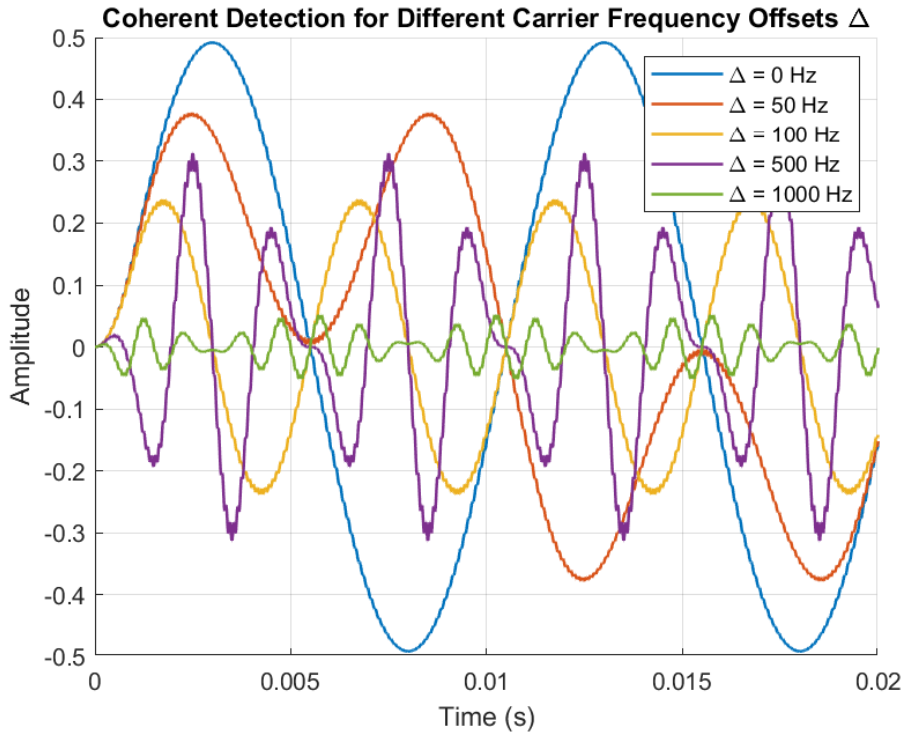

Fig. 102: Coherent Detector output for different Δ values

Looking at Fig. 12, both amplitude and frequency values change with different Δ values. The reason for this can be explained as follows:

New desired signal becomes:

$$r(t) = \frac{1}{2} m(t) * \cos(2\pi\Delta t)$$

Which will make a change in frequency for sure but also an additional scaling in the magnitude of the signal. Therefore, higher the delta, higher the oscillation of the signal, and this will result in reduction in amplitudes.

In other words, as Δ values increase, the signal's frequency increases therefore it oscillates much faster, and this increase in oscillation creates a reduction in the amplitude of the signal.

Part 2.c)

Carrier frequency offset (CFO) is the mismatch between the frequency of the local oscillator and the received signal. The frequency mismathces are shown in Fig. 12. This offset in the carrier signal leads to distortion at the output signal. To eliminate this problem, Phase Lock Loop (PLL) can be used. It basically adjusts the local oscillator to transmitter, which vanishes any possible phase errors. Pilot tones are also used for receivers to estimate the frequency offset and correct it. Thus, any possible phase error will vanish via Pilot tones. Last but not least, noncoherent detection techniques (envelope detector, …) can be used because they do not rely on synchronized carriers and they do not suffer from CFO, however noncoherent detector techniques offer low power performance.

Part 2.d)

In this part, a local oscillator is used to generate a signal with same frequency with c(t) but an additional phase differnce. After the modulation and low-pass filtering the signal again (to get rid of high frequency terms), the resulting signal is in the following form:

$$r(t) = \frac{1}{2} m(t) * \cos(\emptyset)$$

Where Ø takes values from {0, π/6, π/3, π/2} respectively.

As it can be seen, only the amplitude will change because the signal m(t) is scaling by cos(Ø). Fig. 13 shows the resulting signal for all different phase values.

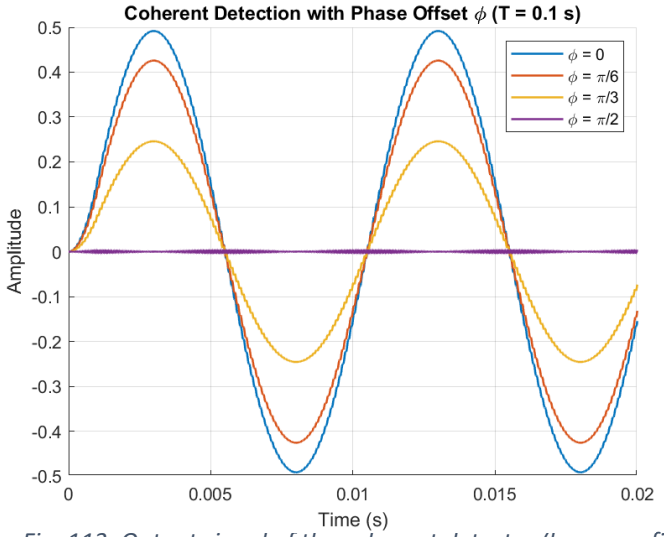

Fig. 113: Output signal of the coherent detector (low-pass filtered)

Appendix

```
rng(55);
%% Part 1.b
N = 1e6;          % # of samples
a = 0.9;          % autoregressive process coefficient which is less than 1
sigma2 = 1;       % Var(noise)

W = zeros(N, 1);  % Output process
V = sqrt(sigma2) * randn(N, 1);  % White Gaussian noise V_n ~ N(0, 1)

% Initial condition
W(1) = 0;

% Generate the AR(1) process
for n = 2:N
    W(n) = a * W(n-1) + V(n);
end

% plot of samples 500-599
section_start = 500;
section_length = 100;

figure;
plot(section_start:section_start+section_length-1,
W(section_start:section_start+section_length-1), 'LineWidth', 1.2);
xlabel('n');
ylabel('W[n]');
title('A Section of the AR(1) Process Realization (Samples 500-599)');
grid on;

%% Part 1.c
% Moving average filter (length 20)
h = ones(20, 1) / 20;

% Apply the system to W[n] to get Y[n]
Y = filter(h, 1, W);

% Plot W[n] and Y[n] over a small range from 500 to 599
section_start = 500;
section_length = 100;
n_range = section_start : section_start + section_length - 1;

figure;
plot(n_range, W(n_range), 'b', 'LineWidth', 1.2); hold on;
plot(n_range, Y(n_range), 'r', 'LineWidth', 1.2);
legend('W[n] (Original)', 'Y[n] (Filtered)');
xlabel('n');
ylabel('Amplitude');
title('Part (c): W[n] vs. Y[n] (Moving Average Output)');
grid on;

%% Part 1.d
[PSD_W, f] = pwelch(W, [], [], [], 1);  % PSD of W_n
[PSD_Y, ~] = pwelch(Y, [], [], [], 1);  % PSD of Y_n

% Estimate magnitude response from PSD ratio
```

```
H_est_mag = sqrt(PSD_Y ./ PSD_W);

% Analytical magnitude response of moving average filter
w = 2 * pi * f;
H_ana_mag = abs(sin(10 * w / 2) ./ sin(w / 2)) / 20;
H_ana_mag(w == 0) = 1;  % Handle singularity at w = 0

figure;
plot(f, 10*log10(PSD_W), 'b', 'LineWidth', 1.2); hold on;
plot(f, 10*log10(PSD_Y), 'r', 'LineWidth', 1.2);
legend('PSD of W[n]', 'PSD of Y[n]');
xlabel('Normalized Frequency');
ylabel('Power Spectral Density (dB)');
title('Estimated PSDs of W[n] and Y[n]');
grid on;

figure;
plot(f, H_est_mag, 'r', 'LineWidth', 1.5); hold on;
plot(f, H_ana_mag, 'b--', 'LineWidth', 1.5);
legend('Estimated |H(f)| via PSDs', 'Analytical |H(f)|');
xlabel('Normalized Frequency');
ylabel('Magnitude');
title('Comparison of Estimated and Analytical Magnitude Response');
grid on;
```

%% Part 1.e

```
N = 1e6;
sigma2 = 1;  % Var(noise)
MA_h = ones(20,1)/20;  % Moving average filter

% Generate white Gaussian noise input
X = sqrt(sigma2) * randn(N, 1);

% Apply the same filter to get the output
Y_white = filter(MA_h, 1, X);

% Plot samples from 500 - 599
section_start = 500;
section_length = 100;
n_range = section_start : section_start + section_length - 1;

figure;
plot(n_range, X(n_range), 'b', 'LineWidth', 1.2); hold on;
plot(n_range, Y_white(n_range), 'r', 'LineWidth', 1.2);
legend('White Noise X[n]', 'Y[n] (Filtered)');
xlabel('n'); ylabel('Amplitude');
title('Part (e): White Gaussian Noise Input and Filtered Output');
grid on;

 [PSD_X, f] = pwelch(X, [], [], [], 1);
[PSD_Ywhite, ~] = pwelch(Y_white, [], [], [], 1);

H_est_mag_white = sqrt(PSD_Ywhite ./ PSD_X);

w = 2 * pi * f;
H_ana_mag = abs(sin(10 * w / 2) ./ sin(w / 2)) / 20;
H_ana_mag(w == 0) = 1;  % fix divide-by-zero at DC
```

```
figure;
[PSD_W, ~] = pwelch(W, [], [], [], 1);
plot(f, 10*log10(PSD_X), 'b', 'LineWidth', 1.2); hold on;
plot(f, 10*log10(PSD_W), 'k--', 'LineWidth', 1.2);
legend('PSD of White Gaussian Noise', 'PSD of W[n]');
xlabel('Normalized Frequency');
ylabel('Power Spectral Density (dB)');
title('PSD Comparison: White Gaussian Noise vs Wn Process');
grid on;
```

%% Part 2.a

```
Fs = 100000;              % Sampling frequency
T = 0.01;                 % T is chosen as 0.01s
t = [0:T*Fs] / Fs;

% Message signal m(t)
fm = 100;                 % Message frequency (Hz)
mt = sin(2 * pi * fm * t);

% Carrier signal
fc = 5000;                % Carrier frequency (Hz)
ct = cos(2 * pi * fc * t);
% DSB-SC modulation: Multiply message with carrier
st = mt .* ct;            % Modulated signal s(t)

figure;
subplot(3,1,1);
plot(t, mt);
title('Message Signal m(t)');
xlabel('Time (s)'); ylabel('Amplitude'); grid on;

subplot(3,1,2);
plot(t, ct);
title('Carrier Signal c(t)');
xlabel('Time (s)'); ylabel('Amplitude'); grid on;

subplot(3,1,3);
plot(t, st);
title('DSB-SC Modulated Signal s(t) = m(t) * c(t)');
xlabel('Time (s)'); ylabel('Amplitude'); grid on;
```

%% Part 2.b

```
Fs = 100000;            % Sampling frequency (Hz)
T = 0.1;                % Total duration is chosen as 0.1s this time
t = [0:T*Fs]/Fs;

fm = 100;               % Message frequency (Hz)
fc = 5000;              % Carrier frequency (Hz)

% Message and DSB-SC modulated signal
mt = sin(2 * pi * fm * t);          % Message signal
ct = cos(2 * pi * fc * t);          % Carrier
st = mt .* ct;                      % DSB-SC modulated signal

delta_set = [0, 50, 100, 500, 1000]; % Carrier offsets (Hz)
colors = lines(length(delta_set));
```

```
figure;
hold on;

% Loop through each delta value
for k = 1:length(delta_set)
    delta = delta_set(k);

    % Local oscillator with frequency offset
    lo = cos(2 * pi * (fc + delta) * t);

    % Coherent detection (multiply with local oscillator)
    rt = st .* lo;

    % Low-pass filter to recover message (simple moving average)
    lpf_order = 100;
    lpf = ones(1, lpf_order) / lpf_order;
    recovered = filter(lpf, 1, rt);

    % Plot 20ms interval
    n_start = 1;
    n_len = round(0.02 * Fs);
    plot(t(n_start:n_start+n_len-1), recovered(n_start:n_start+n_len-1), ...
         'Color', colors(k,:), 'LineWidth', 1.2, ...
         'DisplayName', ['\Delta = ' num2str(delta) ' Hz']);
end

xlabel('Time (s)');
ylabel('Amplitude');
title('Coherent Detection for Different Carrier Frequency Offsets \Delta');
legend('show');
grid on;
```

%% Part 2.d

```
Fs = 100000;            % Sampling frequency (Hz)
T = 0.1;                % Total duration (100 ms)
t = [0:T*Fs] / Fs;

fm = 100;               % Message frequency (Hz)
fc = 5000;              % Carrier frequency (Hz)

% Message and DSB-SC modulated signal
mt = sin(2 * pi * fm * t);         % Message signal
ct = cos(2 * pi * fc * t);         % Carrier
st = mt .* ct;                     % DSB-SC modulated signal

% Phase offsets (radians) and symbolic labels
phi_set = [0, pi/6, pi/3, pi/2];
phi_labels = {'\phi = 0', '\phi = \pi/6', '\phi = \pi/3', '\phi = \pi/2'};
colors = lines(length(phi_set));   % Distinct colors for each signal

figure;
hold on;

% Loop over each phase offset
for k = 1:length(phi_set)
    phi = phi_set(k);

    % Local oscillator with phase offset
```

```
    lo = cos(2 * pi * fc * t + phi);

    % Coherent detection (multiply and LPF)
    rt = st .* lo;

    % Simple low-pass filter (moving average)
    lpf_order = 100;
    lpf = ones(1, lpf_order) / lpf_order;
    recovered = filter(lpf, 1, rt);

    % Plot first 20 ms of recovered signal
    n_start = 1;
    n_len = round(0.02 * Fs);  % 20 ms
    plot(t(n_start:n_start+n_len-1), recovered(n_start:n_start+n_len-1), ...
         'Color', colors(k,:), 'LineWidth', 1.2, ...
         'DisplayName', phi_labels{k});
end

xlabel('Time (s)');
ylabel('Amplitude');
title('Coherent Detection with Phase Offset \phi (T = 0.1 s)');
legend('show');
grid on;
```