DIGITAL COMMUNICATIONS CC442

Final Lab

MAY 21, 2022

FACULTY OF ENGINEERING, COMPUTER AND COMMUNICATIONS DEPARTMENT, UNIVERSITY OF ALEXANDRIA

MATLAB code report

Contributors:

Adham Amr Mohamed - 6713
Mohamed Saeed - 7034
Raphael Guirguis Fawaz - 7252

Contents

Part I

```
close all; clear all; clc;

%%Simulation Parameters
num_bits = 1e5; % Number of bits/SNR
SNR = 0:2:30;   % SNR values
m =input('Enter # of samples representing waveform: ');   %default=20
sampling_instant=input('Enter desired sampling instant: ')%default=20

% Generate continuous waveforms for s1 and s2
s1_amplitude = 1;
s1 = ones(1, m) * s1_amplitude; %s1(t) is rectangular signal with Amp=1
s2_amplitude = 0;
s2 = ones(1, m) * s2_amplitude; %s2(t) is zero signal

%Generate random binary data vector
bits = randi([0, 1], 1, num_bits);
% Generate waveform based on the transmitted bits
waveForm = zeros(1, 1e5 * m);
for i = 0:length(bits)-1
    if bits(i+1) == 1
        val = s1;
    else
        val = s2;
    end
    waveForm((m*i)+1:m*i+length(val)) = val;
end

%BERs for detection methods
match_ber = [];  %Matched Filter
cor_ber = [];    %Correlator
simple_ber = []; %Simple Detector
```

Source Code:

```
for snr = 0:2:30

    % Calculate noise power based on SNR
    noise_power = (s1_amplitude^2) * m / (2 * 10^(snr/10));
    % Generate Gaussian noise
    noise = sqrt(noise_power) * randn(1, length(waveForm));
    % Add noise to the received waveform
    Rx_sequence = waveForm + noise;

    % Matched filter
    diff = s1 - s2;       % Calculate difference
    hmf = diff(end:-1:1); % Impulse response
    %arrays for storing detection results
    out_after_sampling = zeros(1, length(bits));
    conv_output = zeros(1, (2*m-1) * length(bits));
    %Perform matched filtering and sampling
    for i = 0:length(bits)-1
        Rx_sequence_window = Rx_sequence((i*m)+1:(i+1)*m); % Extract
current window
        c = conv(Rx_sequence_window, hmf); % Conv between the window and
MF
        % Store conv o/p in the corresponding range of overall o/p
        conv_output((length(hmf)+length(Rx_sequence_window)-
1)*i+1:(length(hmf)+length(Rx_sequence_window)-1)*i+length(c)) = c;
        % Determine sampling instant within the conv o/p
        m_1 = m + (length(hmf) + length(Rx_sequence_window) - 1) * (i);
        out_after_sampling(i+1) = conv_output(m_1); % value after
sampling
    end
    % Thresholding and bit error calculation for matched filter
    vth = ((s1_amplitude^2) * m / 2);
    out_after_sampling_th = zeros(1, 1e5);
    for j = 1:length(out_after_sampling)
        if out_after_sampling(j) >= vth
            out_after_sampling_th(j) = 1;
        else
            out_after_sampling_th(j) = 0;
        end
    end
    [number1, ratio1] = biterr(bits, out_after_sampling_th, []);
    match_ber = [match_ber ratio1];
```

```
% Correlator
    output_cor = zeros(1, num_bits);     % Correlator output
    output_cor_th = zeros(1, num_bits); % Thresholded output
    for i = 0:length(bits)-1
        Rx_sequence_window = Rx_sequence((i*m)+1:(i+1)*m); % Extract
current window
        cc = sum(Rx_sequence_window .* s1); % correlation between the
window and s1
        output_cor(i+1) = cc; %correlation output
    end
    % Thresholding for correlator
    for j = 1:length(output_cor)
        if output_cor(j) >= vth
            output_cor_th(j) = 1;
        else
            output_cor_th(j) = 0;
        end
    end
    [number2, ratio2] = biterr(bits, output_cor_th, []);
    cor_ber = [cor_ber ratio2];

    % Simple
    out_after_simple = zeros(1, length(bits)); % Output
    for i = 0:length(bits)-1
        Rx_sequence_window = Rx_sequence((i*m)+1:(i+1)*m); % Extract the
current window
        out_after_simple(i+1) = Rx_sequence_window(m/2);    % Select
middle sample of the window
    end
    % Thresholding for simple detection
    vth_simple = 0.5;
    out_after_simple_th = zeros(1, num_bits);
    for j = 1:length(out_after_simple)
        if out_after_simple(j) >= vth_simple
            out_after_simple_th(j) = 1;
        else
            out_after_simple_th(j) = 0;
        end
    end

    [number3, ratio3] = biterr(bits, out_after_simple_th, []);
    simple_ber = [simple_ber ratio3];
end
```

Explanation:

1. Matched Filter:

The matched filter is implemented by convolving the received waveform with the impulse response of the matched filter. The output is sampled at the desired sampling instant. The threshold for decision making is set based on the amplitude of the transmitted signal. The matched filter is known to be optimal in terms of maximizing the signal-to-noise ratio (SNR) at the output, thus providing better detection performance compared to other methods. As the SNR increases, the system's bit error rate (BER) decreases, indicating better accuracy.

2. Correlator:

The correlator computes the correlation between the received waveform and a reference waveform (s1 in this case). The output is thresholded to make a decision. The correlator is a simpler method compared to the matched filter but may have slightly lower performance. It relies on the correlation property between the received waveform and the reference waveform. The BER decreases as the SNR increases, similar to the matched filter.

3. Simple Detection:

The simple detection method selects the middle sample of each received window as the output. It then applies a fixed threshold for decision making. This method is the simplest among the three and may have lower performance compared to the matched filter and correlator. The threshold used for decision making is a fixed value of 0.5. The BER decreases with increasing SNR, indicating improved performance.

Questions:

Calculate the transmitted signal power.

It is important to ensure that the power of the transmitted signal remains the same across all methods for the following reasons:

1. Signal-to-Noise Ratio (SNR) Comparison:

   By keeping the transmitted signal power constant, the SNR can be directly compared across the different detection methods. SNR is a critical factor that determines the quality and reliability of the received signal. It represents the ratio of the signal power to the noise power. By maintaining the same transmitted signal power, any differences in performance between the detection methods can be attributed to the methods themselves rather than variations in signal power.

2. Sensitivity and Thresholding:

   Different detection methods have different sensitivities and thresholding requirements. The decision thresholds used for detecting and decoding the received signal are typically based on the expected power levels of the transmitted signal. If the transmitted signal power varies across the methods being compared, it could lead to different detection thresholds and hence biased comparisons. By maintaining equal transmitted signal power, the detection thresholds can be set consistently, ensuring a fair comparison.

3. Performance Evaluation:

   When comparing the bit error rate (BER) or other performance metrics of different detection methods, having the same transmitted signal power allows for a meaningful and unbiased evaluation. It enables a direct comparison of how each method performs in the presence of the same noise power level, without confounding factors due to variations in the transmitted signal power.

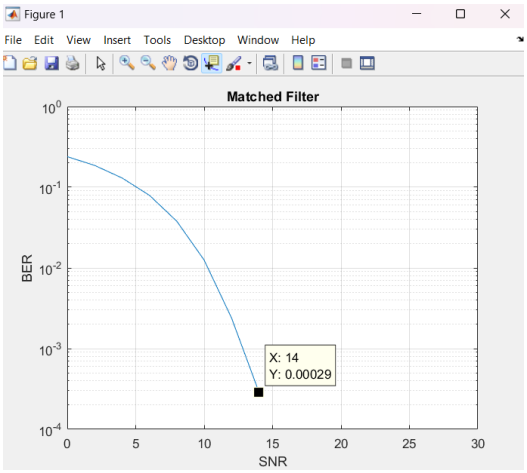
In summary, keeping the power of the transmitted signal constant ensures that the comparison between the matched filter, correlator, and simple detector is fair and unbiased.

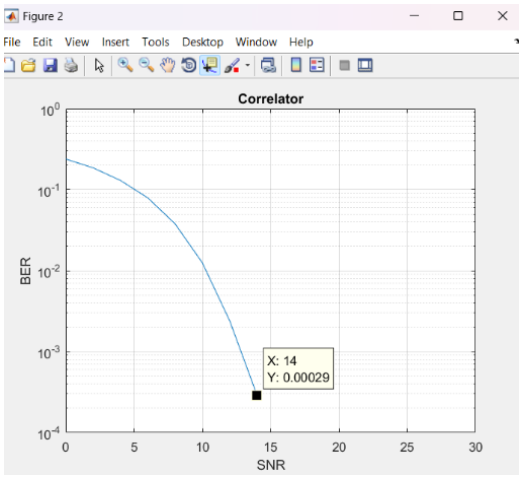
The Rule would be: ***Transmitted signal power = (s1_amplitude^2) * n/2*** ,
where n is the number of bits representing waveform.

n = 10	Power = 5 Watts
n = 20	Power = 10 Watts
n = 100	Power = 50 Watts

At which value of SNR the system is nearly without error (for the given frame)?

1. Number of bits representing waveform = 10





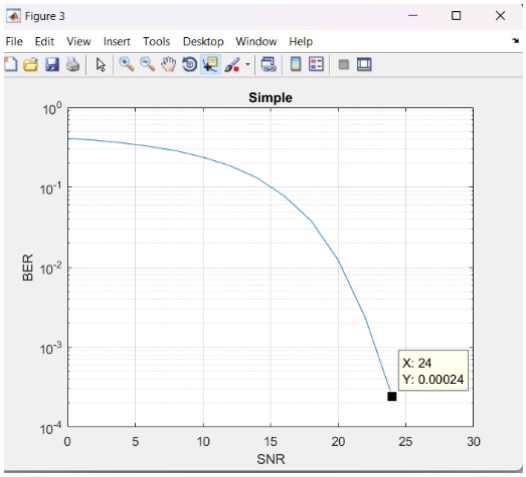
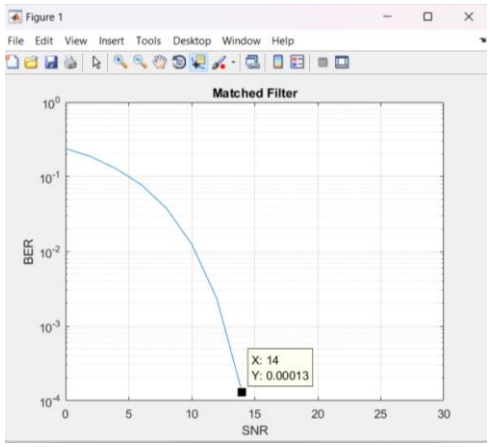
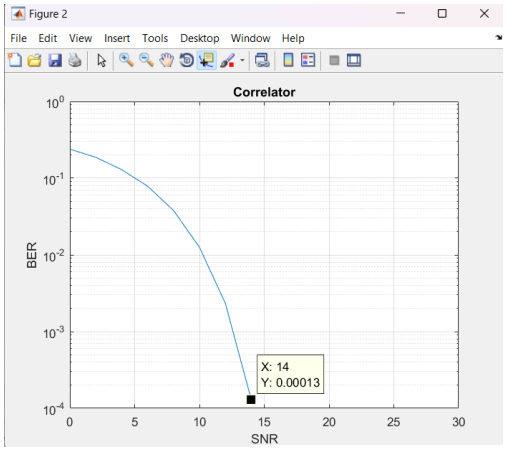


Figure 1

2. Number of bits representing waveform = 20





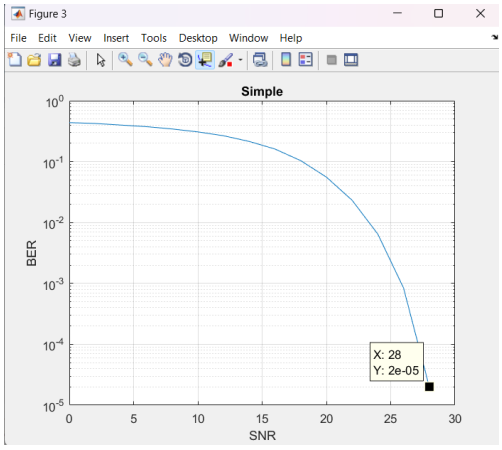
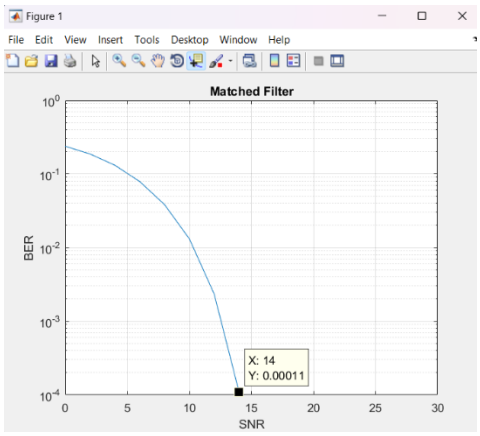
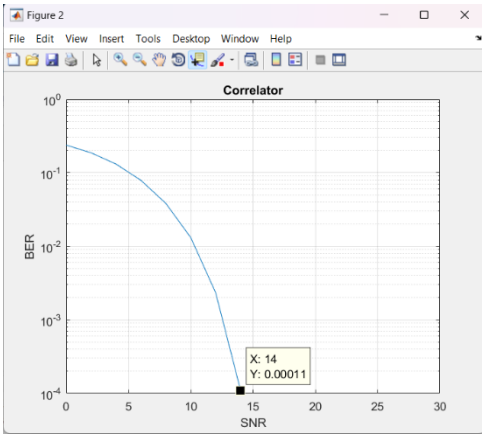


Figure 2

3. Number of bits representing waveform = 100





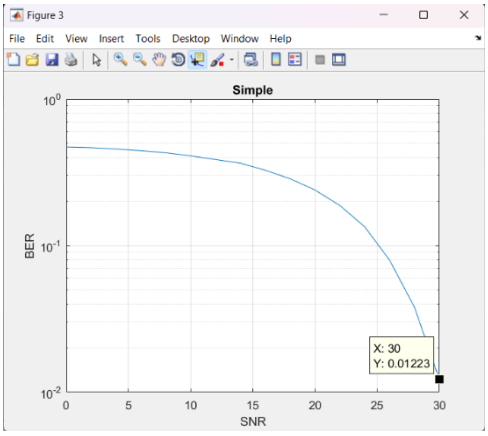


Figure 3

Part II

Source Code :

```
clc; close all; clear all;
%% Variables

% Original uniformly distributed random bits [0 1]
bits = randi([0 1], 1, 1e6);
Power = sum(bits.^2) / length(bits);  % Power of the transmitted
signal
snr = 0:2:30;  % SNR values in dB
BER_PSK = [];  % Bit Error Rate matrix for PSK
BER_ASK = [];  % Bit Error Rate matrix for ASK
BER_FSK = [];  % Bit Error Rate matrix for FSK
BER_OOK = [];  % Bit Error Rate matrix for OOK
BER_PRK = [];  % Bit Error Rate matrix for PRK
BER_16QAM = [];  % Bit Error Rate matrix for 16QAM

%% Modulations

% PSK modulation and demodulation
for i = 1:length(snr)
    modulated_signal = pskmod(bits, 2);
    noisy_signal = awgn(modulated_signal, snr(i), 'measured');
    demodulated_signal = pskdemod(noisy_signal, 2);
    [num_errors, prob] = biterr(bits, demodulated_signal);
    BER_PSK = [BER_PSK prob];
end

% ASK modulation and demodulation
for i = 1:length(snr)
    carrier = sqrt(2*Power);  % Carrier amplitude
    modulated_signal = bits .* carrier;  % ASK modulation
    noisy_signal = awgn(modulated_signal, snr(i), 'measured');
    demodulated_signal = noisy_signal >= carrier/2;  % Threshold
detection
    [num_errors, prob] = biterr(bits, demodulated_signal);
    BER_ASK = [BER_ASK prob];
end
```

```
% FSK modulation and demodulation
fs = 100;  % Sampling frequency
freq_sep = 2; % Frequency separation for FSK
nsamp = 8; % number of samples

for i = 1:length(snr)
    modulated_signal = fskmod(bits, 2, freq_sep, nsamp, fs);
    noisy_signal = awgn(modulated_signal, snr(i), 'measured');
    demodulated_signal = fskdemod(noisy_signal, 2, freq_sep, nsamp,
fs);
    [num_errors, prob] = biterr(bits, demodulated_signal);
    BER_FSK = [BER_FSK prob];
end

% OOK modulation and demodulation
for i = 1:length(snr)
    carrier = sqrt(2 * Power);  % Carrier amplitude
    modulated_signal = bits .* carrier;  % OOK modulation
    noisy_signal = awgn(modulated_signal, snr(i), 'measured');
    demodulated_signal = noisy_signal >= carrier / 2;  % Threshold
detection
    [num_errors, prob] = biterr(bits, demodulated_signal);
    BER_OOK = [BER_OOK prob];
end

% PRK
for i = 1:length(snr)  % SNR values in dB
    % Map the bits to PRK symbols
    prk_symbols = 2 * bits - 1;
    % Calculate the noise power based on the SNR
    noisy_symbols = awgn(prk_symbols, snr(i), 'measured');
    % Threshold detection
    detected_bits = noisy_symbols >= 0;
    % Compare the original bits with the detected bits
    [num_errors, prob] = biterr(bits, detected_bits);
    BER_PRK = [BER_PRK prob];  % Add new probability of error
end
```

```
%% 16QAM
for i = 1:length(snr)
    modulated_signal = qammod(bits, 16);  % 16QAM modulation
    noisy_signal = awgn(modulated_signal, snr(i), 'measured');
    demodulated_signal = qamdemod(noisy_signal, 16);  % 16QAM demodulation
    [num_errors, prob] = biterr(bits, demodulated_signal);
    BER_16QAM = [BER_16QAM prob];
end

%% Plotting BER curves
figure;
semilogy(snr, BER_PSK);
xlim([0 30]);
title('Bit Error Rate (BER) - PSK', 'fontweight', 'bold');
xlabel('SNR (dB)');
ylabel('BER');
grid on;

figure;
semilogy(snr, BER_ASK);
xlim([0 30]);
title('Bit Error Rate (BER) - ASK', 'fontweight', 'bold');
xlabel('SNR (dB)');
ylabel('BER');
grid on;

figure;
semilogy(snr, BER_FSK);
xlim([0 30]);
title('Bit Error Rate (BER) - FSK', 'fontweight', 'bold');
xlabel('SNR (dB)');
ylabel('BER');
grid on;

figure;
semilogy(snr, BER_OOK);
xlim([0 30]);
title('Bit Error Rate (BER) - OOK', 'fontweight', 'bold');
xlabel('SNR (dB)');
ylabel('BER');
grid on;

figure;
semilogy(snr, BER_PRK);
xlim([0 30]);
title('Bit Error Rate (BER) - PRK', 'fontweight', 'bold');
xlabel('SNR (dB)');
ylabel('BER');
grid on;
```

Explantation:

The code performs simulations to analyze the performance of various digital modulation schemes under different Signal-to-Noise Ratio (SNR) conditions. The modulation schemes considered in this analysis include Phase-Shift Keying (PSK), Amplitude-Shift Keying (ASK), Frequency-Shift Keying (FSK), On-Off Keying (OOK), Polar Return-to-Zero (PRK), and Quadrature Amplitude Modulation (16QAM).

For each modulation scheme, the code generates random bits and applies the corresponding modulation technique to encode the bits into a transmitted signal. The transmitted signal is then corrupted by adding noise according to different SNR levels. The demodulation process is performed to recover the transmitted bits from the noisy signal.

The Bit Error Rate (BER) is calculated by comparing the original bits with the demodulated bits. The BER quantifies the accuracy of the transmission, indicating the probability of erroneous bit detection. The code collects the BER values for each modulation scheme and plots them against the SNR levels, using a logarithmic scale for the BER axis.

By examining the BER curves for different modulation schemes, insights can be gained regarding their performance in the presence of noise. The plots provide a visual representation of how each modulation scheme performs under varying SNR conditions, enabling a comparison of their error rates. This analysis helps in understanding the strengths and weaknesses of different modulation techniques and their suitability for specific communication scenarios.

Results:

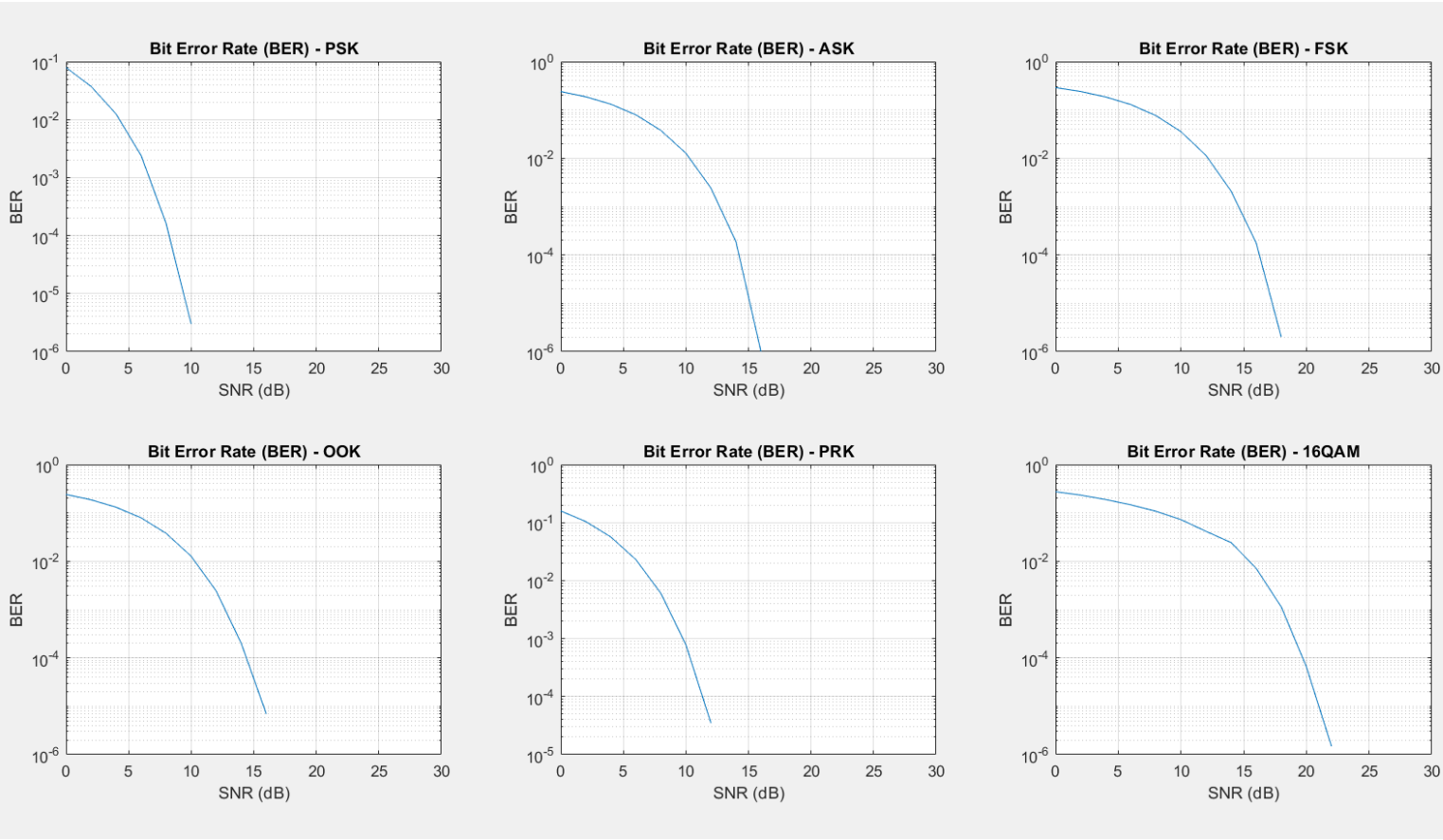


Figure 4 Different modulation results after running the code.

Questions:

Evaluate the same curves using the MATLAB built-in functions:

Code:

```
clc;
close all;
clear all;

% Original uniformly distributed random bits [0 1]
bits = randi([0 1], 1, 1e6);
Power = sum(bits.^2) / length(bits); % Power of the transmitted signal
BER_PSK = []; % Bit Error Rate matrix for PSK
BER_PAM = []; % Bit Error Rate matrix for PAM

snr = 0:2:30; % SNR values in dB

for i = 1:length(snr)
    % PSK modulation
    modulated_signal_psk = pskmod(bits, 2);

    % Add noise to PSK modulated signal
    noisy_signal_psk = awgn(modulated_signal_psk, snr(i), 'measured');

    % PSK demodulation
    demodulated_signal_psk = pskdemod(noisy_signal_psk, 2);

    % Compare the original bits with the detected bits for PSK
    [num_errors_psk, prob_psk] = biterr(bits, demodulated_signal_psk);
    BER_PSK = [BER_PSK prob_psk]; % Add new probability of error for PSK

    % PAM modulation
    modulated_signal_pam = pammod(bits, 2);

    % Add noise to PAM modulated signal
    noisy_signal_pam = awgn(modulated_signal_pam, snr(i), 'measured');

    % PAM demodulation
    demodulated_signal_pam = pamdemod(noisy_signal_pam, 2);

    % Compare the original bits with the detected bits for PAM
    [num_errors_pam, prob_pam] = biterr(bits, demodulated_signal_pam);
    BER_PAM = [BER_PAM prob_pam]; % Add new probability of error for PAM
end

% Plotting BER curves for PSK and PAM
figure;
semilogy(snr, BER_PSK, 'b-o', 'LineWidth', 1.5);
hold on;
semilogy(snr, BER_PAM, 'r-o', 'LineWidth', 1.5);
xlim([0 30]);
title('Bit Error Rate (BER)', 'FontWeight', 'bold');
xlabel('SNR (dB)');
ylabel('BER');
legend('PSK', 'PAM');
grid on;
hold off;
```

Graph:

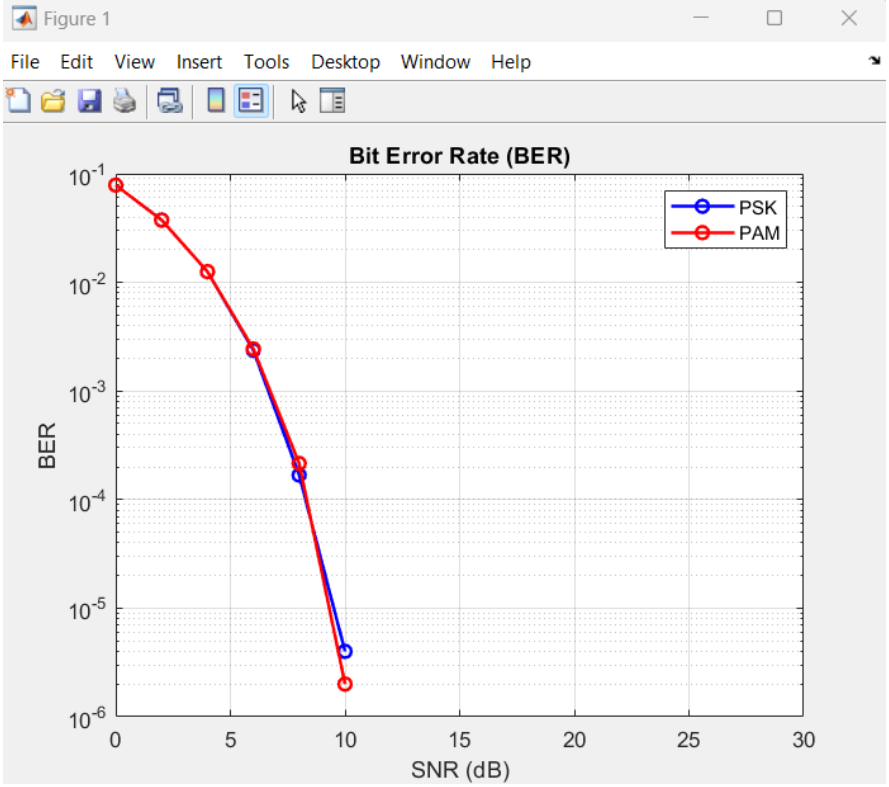


Figure 5 built in function results comparison.

Which type of modulation has the best performance? Why?

Determining which modulation scheme has the best performance depends on the specific requirements and constraints of the communication system. For example:

If the primary concern is bandwidth efficiency, ASK may be preferred due to its narrower bandwidth requirement.

If the communication channel is noisy, FSK may provide better performance due to its frequency-based discrimination, which can better tolerate noise.

If high data rates are required, PSK may be the preferred choice despite requiring a wider bandwidth.

Ultimately, the best modulation scheme choice depends on a thorough analysis of the specific communication system's requirements, channel characteristics, available bandwidth, noise levels, and trade-offs between factors such as data rate, bandwidth efficiency, and noise immunity.

At which value of SNR the system is nearly without error (for each type of modulation)?

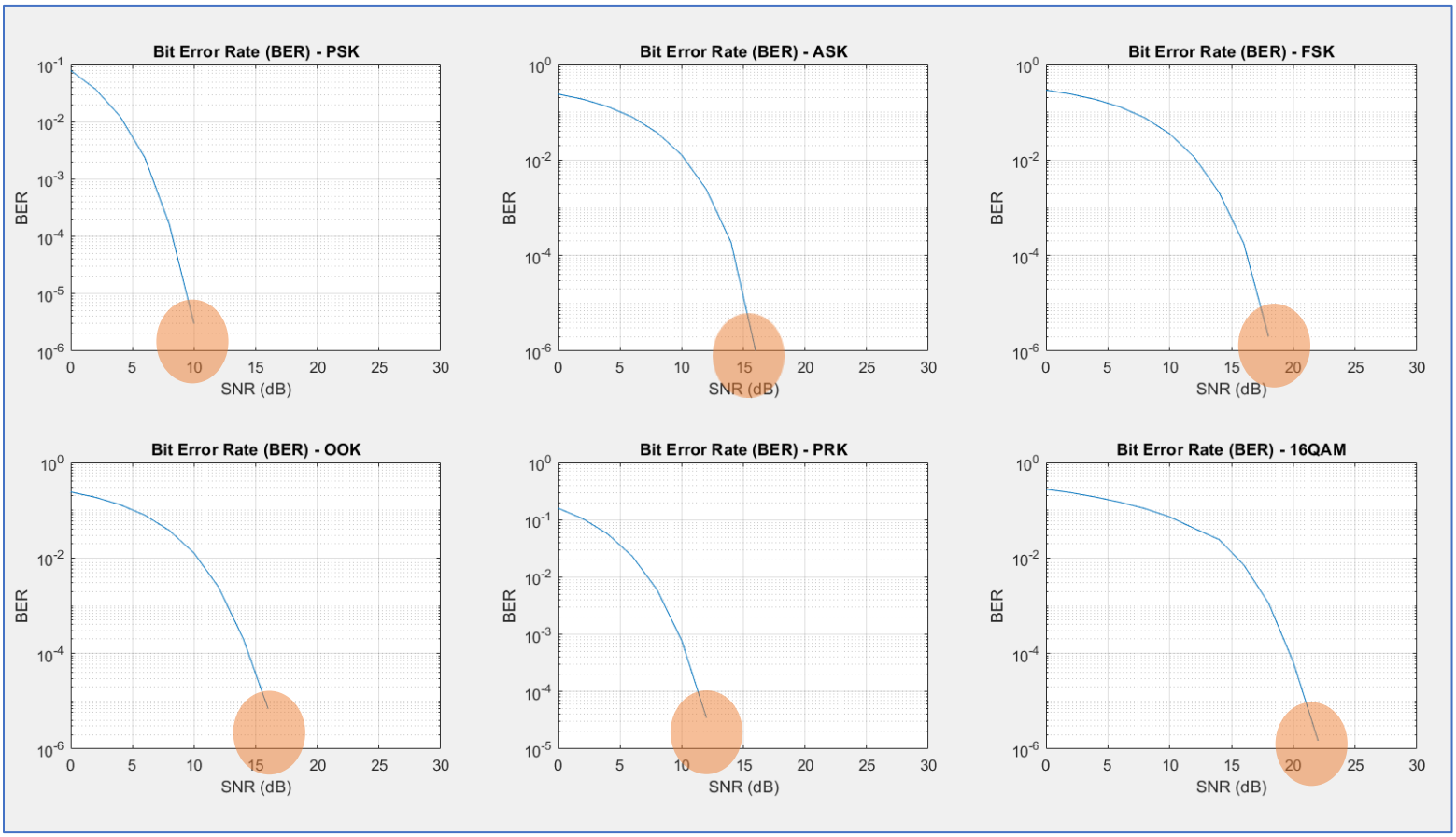


Figure 6 values at which error is lowest.

Comparison:

```
figure;
semilogy(snr, BER_PSK, 'b-', 'LineWidth', 1.5);
hold on;
semilogy(snr, BER_ASK, 'r-', 'LineWidth', 1.5);
semilogy(snr, BER_FSK, 'g-', 'LineWidth', 1.5);
semilogy(snr, BER_OOK, 'm-', 'LineWidth', 1.5);
semilogy(snr, BER_PRK, 'c-', 'LineWidth', 1.5);
semilogy(snr, BER_16QAM, 'k-', 'LineWidth', 1.5);
hold off;
xlim([0 30]);
title('Bit Error Rate (BER) Comparison', 'fontweight', 'bold');
xlabel('SNR (dB)');
ylabel('BER');
grid on;
legend('PSK', 'ASK', 'FSK', 'OOK', 'PRK', '16QAM', 'Location', 'best');
```

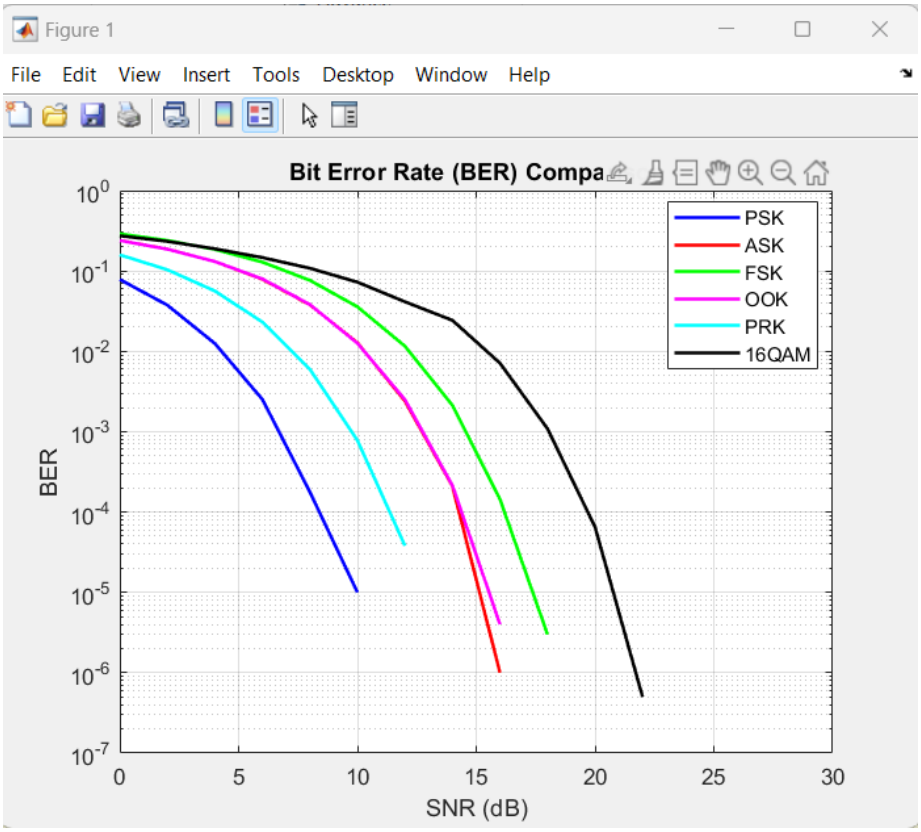


Figure 7