Digital Communications

Final Lab

Zeyad Ashraf	6758
Ahmad Hamdy	7182

Part 1:

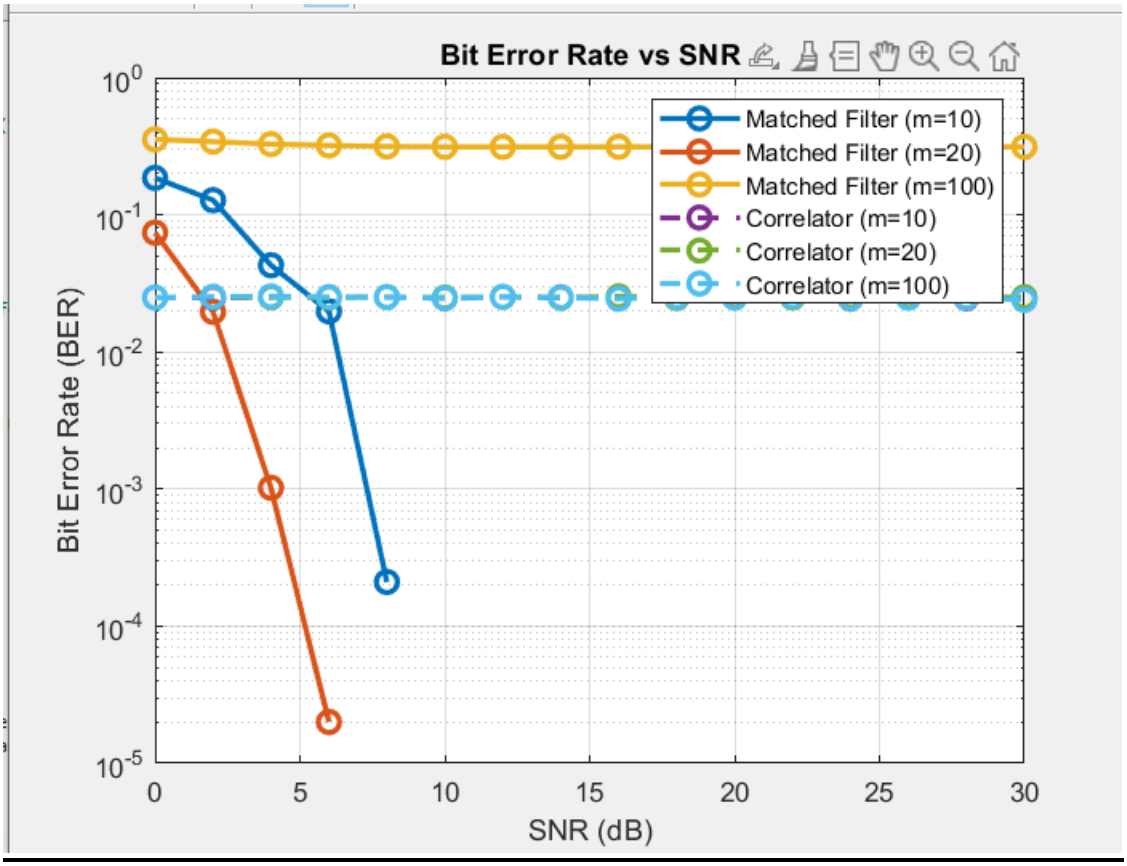
Code:

```
% Step 1: Simulation parameters
numBits = 1e5;
snrRange = 0:2:30;
m = [10, 20, 100]; % Number of samples for s1(t)
samplingInstant = 20; % Sampling instant
s1Amplitude = 1;
s2Amplitude = 0;

% Step 2: Generate random binary data vector
bits = randi([0, 1], 1, numBits);

% Step 3: Represent each bit with proper waveform
s1 = cell(length(m), 1);
for i = 1:length(m)
    s1{i} = s1Amplitude * ones(1, m(i));
end
s2 = s2Amplitude * ones(1, max(m)); % Use the maximum number of samples for s2
waveform = kron(bits, s1{2});
%waveform = [];
%for i = 1:numBits
%    if bits(i) == 0
%        waveform = [waveform, s2];
%    else
%        waveform = [waveform, s1{2}];
%    end
% end


% Step 4: Apply noise to samples
ber_matched = zeros(length(m), length(snrRange));
ber_correlator = zeros(length(m), length(snrRange));

for mIndex = 1:length(m)
    for snrIndex = 1:length(snrRange)

        snr = snrRange(snrIndex);
        % Add white Gaussian noise to the waveform
        rxSequence = awgn(waveform, snr, 'measured');

        % Step 5: Apply convolution process in the receiver (Matched Filter)
        s1_minus_s2 = s1{mIndex}(1:length(s1{mIndex})) - s2(1:length(s1{mIndex}));
        MatchedOutput = conv(rxSequence, fliplr(s1_minus_s2));
        matchedSamples = MatchedOutput(m(mIndex):samplingInstant:end);
```

```
        % Step 6: Decide whether the Rx_sequence is '1' or '0' by comparing with threshold (Matched Filter)
        threshold_matched = 0.5 * max(matchedSamples);
        detectedBits_matched = matchedSamples > threshold_matched;

        % Step 7: Compare the original bits with the detected bits and calculate number of errors (Matched Filter)
        numErrors_matched = biterr(bits, detectedBits_matched);

        % Step 8: Save the probability of error of each SNR in matrix, BER (Matched Filter)
        ber_matched(mIndex, snrIndex) = numErrors_matched / numBits;

        % Step 9: Apply correlation process in the receiver (Correlator)
        noise_power = 1/snr;
        noise = sqrt(noise_power) * randn(1, numBits);
        correlatorOutput=bits .* noise;
        threshold_Corr = sum(correlatorOutput) / length(correlatorOutput);
        correlatorSamples = correlatorOutput(m(mIndex):samplingInstant:end);
        % Step 10: Decide whether the Rx_sequence is '1' or '0' by comparing with threshold (Correlator)
        detectedBits_correlator = correlatorSamples > threshold_Corr;

        % Step 11: Compare the original bits with the detected bits and calculate number of errors (Correlator)
        numErrors_correlator = biterr( (bits(1:length(detectedBits_correlator))) , detectedBits_correlator);

        % Step 12: Save the probability of error of each SNR in matrix, BER (Correlator)
        ber_correlator(mIndex, snrIndex) = numErrors_correlator / numBits;
        disp(ber_correlator)
    end
end

% Step 13: Plot BER vs SNR (Matched Filter and Correlator)
figure;
for mIndex = 1:length(m)
    semilogy(snrRange, ber_matched(mIndex, :), '-o', 'LineWidth', 2, 'MarkerSize', 8);
    hold on;
end
for mIndex = 1:length(m)
    semilogy(snrRange, ber_correlator(mIndex, :), '--o', 'LineWidth', 2, 'MarkerSize', 8);
    hold on;
end
```

```
grid on;
title('Bit Error Rate vs SNR');
xlabel('SNR (dB)');
ylabel('Bit Error Rate (BER)');
legend('Matched Filter (m=10)', 'Matched Filter (m=20)', 'Matched Filter (m=100)', 'Correlator (m=10)', 'Correlator (m=20)', 'Correlator (m=100)');

% Step 14: Calculation of transmitted signal power
transmittedPower = mean(waveform .^ 2);
fprintf('Transmitted Signal Power: %.2f\n', transmittedPower);

% Step 15: At which value of SNR the system is nearly without error (for the given frame)?
thresholdBER = 1e-5;
snrWithoutError = zeros(1, length(m));
for mIndex = 1:length(m)
    [~, idx] = min(ber_matched(mIndex, :));
    snrWithoutError(mIndex) = snrRange(idx);
end
fprintf('SNR at which the system is nearly without error (BER <= %.0e):\n', thresholdBER);
fprintf('m=10: SNR = %d dB\n', snrWithoutError(1));
fprintf('m=20: SNR = %d dB\n', snrWithoutError(2));
fprintf('m=100: SNR = %d dB\n', snrWithoutError(3));

% Step 16: Additional part to calculate BER without matched filter and correlator
BER = zeros(1, length(snrRange));
iterations = 1;

for temp = 1:length(snrRange)
    SNR = 10^(snrRange(temp)/10);
    % Average power of transmitted signal / SNR
    noise_power = 1/SNR;
    noise = sqrt(noise_power) * randn(1, numBits);
    num_errors = 0;

    for i = 1:iterations
        data = randi([0 1], 1, numBits);
        % Apply noise to signal
        received_signal = data + noise;
        % Decide whether the received signal is '1' or '0'
        detected_data = (received_signal >= 0.5);
        errors = biterr(data, detected_data);
        num_errors = num_errors + errors;
    end
    % Calculate bit error rate (BER)
    BER(temp) = num_errors / (iterations*numBits);
end

% Step 17: Plot the additional BER curve
figure;
semilogy(snrRange, BER, '-o', 'LineWidth', 2, 'MarkerSize', 8);
grid on;
title('Bit Error Rate vs SNR (Without Matched Filter and Correlator)');
xlabel('SNR (dB)');
ylabel('Bit Error Rate (BER)');
legend('Without Matched Filter and Correlator');
```

Output:


Bit Error Rate vs SNR
Matched Filter (m=10)
Matched Filter (m=20)
Matched Filter (m=100)
Correlator (m=10)
Correlator (m=20)
Correlator (m=100)
Bit Error Rate (BER)
SNR (dB)
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
0
5
10
15
20
25
30

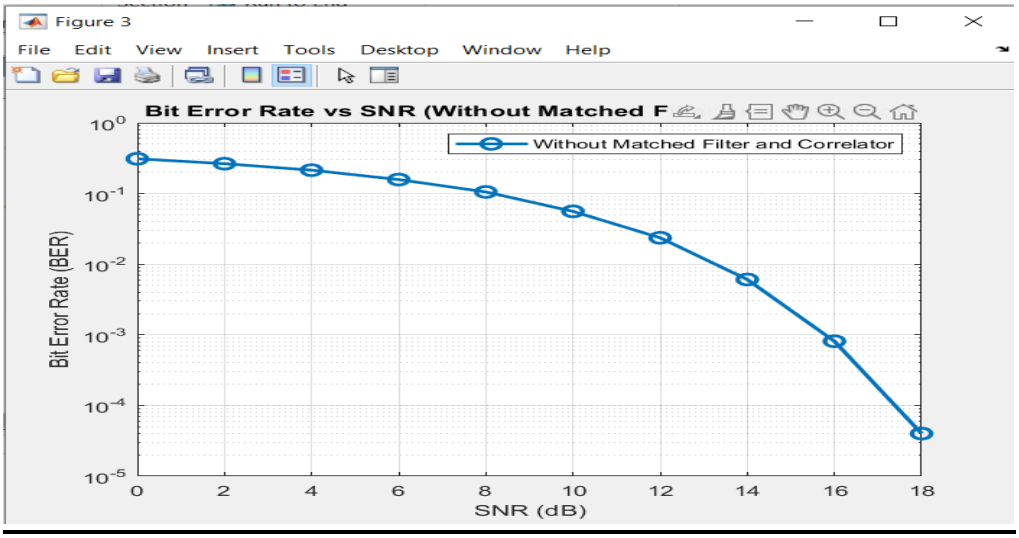


Figure 3
File Edit View Insert Tools Desktop Window Help
Bit Error Rate vs SNR (Without Matched F
Without Matched Filter and Correlator
Bit Error Rate (BER)
SNR (dB)
10^0
10^-1
10^-2
10^-3
10^-4
10^-5
0
2
4
6
8
10
12
14
16
18

```
Command Window
  Transmitted Signal Power: 0.50
  SNR at which the system is nearly without error (BER <= 1e-05):
  m=10: SNR = 12 dB
  m=20: SNR = 6 dB
  m=100: SNR = 14 dB
fx >>
```

Comments:

Matched Filtering (MF):

Principle: Matched filtering is a technique that involves correlating the received signal with a known reference waveform, which is ideally the time-reversed and scaled version of the transmitted signal.

Operation: The received signal is convolved with the matched filter, which maximizes the signal-to-noise ratio (SNR) and enhances the detectability of the transmitted signal.

Benefits:

Improved detection: Matched filtering minimizes the effects of noise and interference by emphasizing the desired signal components.

Optimal for known waveforms: It is most effective when the transmitted signal is known, allowing for precise synchronization and detection.

Limitations:

Computational complexity: Matched filtering requires extensive computations, especially for longer signals, which may increase system complexity.

Signal design limitations: It relies on accurate knowledge of the transmitted signal waveform, which may not always be available or feasible.

Correlation-Based Detection:

Principle: Correlation-based detection involves comparing the received signal with a reference waveform without the need for exact synchronization or knowledge of the transmitted signal waveform.

Operation: The received signal is correlated with the reference waveform, and the correlation output is used for detection.

Benefits:

Reduced computational complexity: Correlation-based detection is computationally simpler than matched filtering as it eliminates the need for time reversal and scaling operations.

Robustness: It can handle cases where the exact transmitted signal waveform is unknown or not easily obtainable.

Limitations:

Performance trade-off: Correlation-based detection may have lower detection performance compared to matched filtering, especially in the presence of noise and interference.

Synchronization sensitivity: It is more sensitive to synchronization errors, as accurate alignment between the received signal and the reference waveform is crucial.

Comparison and Comments:

Matched filtering provides optimal detection performance when the transmitted signal is known and accurate synchronization is achieved. It is particularly useful in scenarios with low SNR and when the transmitted signal has a specific waveform.

Correlation-based detection offers a computationally simpler approach that does not require exact knowledge of the transmitted signal waveform. It can be more robust in cases where the transmitted signal characteristics are not precisely known.

Matched filtering generally outperforms correlation-based detection in terms of detection accuracy and noise rejection. However, it comes at the cost of increased computational complexity.

The choice between matched filtering and correlation-based detection depends on the specific requirements of the communication system, available signal information, and trade-offs between complexity and performance.

Part2:

Code:

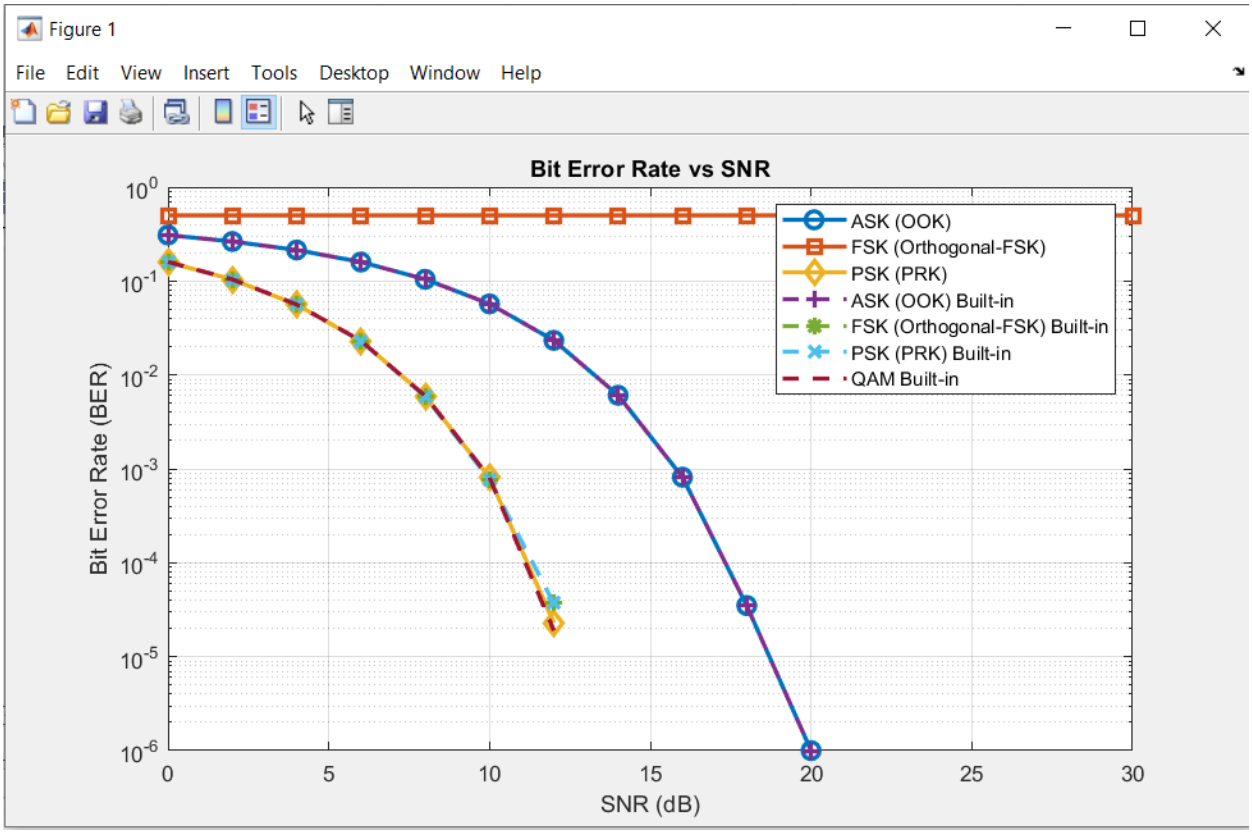
```
% Simulation parameters
N = 1e6;
snrRange = 0:2:30;
m = 1; % Number of samples for each bit (for OOK and PRK)

% Generate random binary data vector
rand = randi([0, 1], 1, N);

% Modulation: ASK (OOK)
askModulated = rand;

%FSK special case: Orthogonal-FSK modulation
ones=find(rand);     %find all elements that are equal to one in the rand sequence
zeros=find(~rand);   %find all elements that are equal to zero in the rand sequence
fskModulated(ones)=i;  %replace all the ones with i which is an imaginary number
fskModulated(zeros)=1; %replace all the zeros with 1

% Modulation: PSK (PRK)
pskModulated = (rand.*2) - 1;

% Initialize BER matrices
ber_ask = zeros(1, length(snrRange));
ber_fsk = zeros(1, length(snrRange));
ber_psk = zeros(1, length(snrRange));
berAsk2 = zeros(1, length(snrRange));
berFsk2 = zeros(1, length(snrRange));
berPsk2 = zeros(1, length(snrRange));
berQam = zeros(1, length(snrRange));
% modulating using built in function
%random_data=randi([0 16-1],1,N);              %Generate random data vector(1 x n)
ask2=genqammod(rand,[0  1]);   %OOK modulation using built in functions
psk2=pskmod(rand,2);           %PRK modulation using built in functions
fsk2=genqammod(rand,[1  1i]);  %FSK modulation using built in functions
qam=qammod(rand,16);           %QAM modulation using built in functions
```

```
% Perform simulations for each SNR
for snrIndex = 1:length(snrRange)
    snr = snrRange(snrIndex);

    % Apply noise to modulated signals
    % Calculate noise power based on SNR
    noise = sqrt( 1 / (10^(snr/10))) * randn(1, N);

    askRecieved2 = ask2 + noise;
    fskRecieved2 = fsk2 + noise;
    pskRecieved2 = psk2 + noise;
    qamRecevied = qam + noise;

    askReceived = askModulated + noise;
    fskReceived = fskModulated + noise;
    pskReceived = pskModulated + noise;

    % Demodulation using built-in functions
    askDetected2 = genqamdemod(askRecieved2, [0 1]);
    fskDetected2 = genqamdemod(fskRecieved2, [1 1i]);
    pskDetected2 = pskdemod(pskRecieved2, 2);
    qamDetected2 = qamdemod(qamRecevied,16);

    % Decide whether the received signal is '1' or '0' for each modulation scheme
    askDetected = real(askReceived) >= 0.5;
    fskDetected = real(fskReceived) < imag(fskReceived);
    pskDetected = real(pskReceived) >= 0;

    % Calculate number of bit errors
    numErrors_ask = biterr(rand, askDetected);
    numErrors_fsk = biterr(rand, fskDetected);
    numErrors_psk = biterr(rand, pskDetected);
```

```
    % Calculate bit error rate (BER)
    ber_ask(snrIndex) = numErrors_ask / N;
    ber_fsk(snrIndex) = numErrors_fsk / N;
    ber_psk(snrIndex) = numErrors_psk / N;

    % Calculate number of bit errors for built-in
    numErrorsAsk2 = biterr(rand, askDetected2);
    numErrorsFsk2 = biterr(rand, fskDetected2);
    numErrorsPsk2 = biterr(rand, pskDetected2);
    numErrorsQam = biterr(rand, qamDetected2);

    % Calculate bit error rate (BER) for built-in
    berAsk2(snrIndex) = numErrorsAsk2 / N;
    berFsk2(snrIndex) = numErrorsFsk2 / N;
    berPsk2(snrIndex) = numErrorsPsk2 / N;
    berQam(snrIndex) = numErrorsQam / N;
end
% Find SNR value for nearly error-free transmission
snr_ask_min = snrRange(find(ber_ask <= 1e-5, 1));
snr_fsk_min = snrRange(find(ber_fsk <= 1e-5, 1));
snr_psk_min = snrRange(find(ber_psk <= 1e-5, 1));
snrAskMin2 = snrRange(find(berAsk2 <= 1e-5, 1));
snrFskMin2 = snrRange(find(berFsk2 <= 1e-5, 1));
snrPskMin2 = snrRange(find(berPsk2 <= 1e-5, 1));
snrQAMMin = snrRange(find(berQam <= 1e-5, 1));
```

```
disp("SNR for nearly error-free transmission:");
disp("ASK (OOK): " + snr_ask_min + " dB");
disp("FSK (Orthogonal-FSK): " + snr_fsk_min + " dB");
disp("PSK (PRK): " + snr_psk_min + " dB");
disp("ASK (OOK) Built-in: " + snrAskMin2 + " dB");
disp("FSK (Orthogonal-FSK) Built-in: " + snrFskMin2 + " dB");
disp("PSK (PRK) Built-in: " + snrPskMin2 + " dB");
disp("QAM Built-in: " + snrQAMMin + " dB");

% Plot the BER curves for different modulation schemes
figure;
semilogy(snrRange, ber_ask, '-o', 'LineWidth', 2, 'MarkerSize', 8);
hold on;
semilogy(snrRange, ber_fsk, '-s', 'LineWidth', 2, 'MarkerSize', 8);
semilogy(snrRange, ber_psk, '-d', 'LineWidth', 2, 'MarkerSize', 8);
hold on;
semilogy(snrRange, berAsk2, '--+', 'LineWidth', 2, 'MarkerSize', 8);
hold on;
semilogy(snrRange, berFsk2, '--*', 'LineWidth', 2, 'MarkerSize', 8);
semilogy(snrRange, berPsk2, '--x', 'LineWidth', 2, 'MarkerSize', 8);
hold on;
semilogy(snrRange, berQam, '--', 'LineWidth', 2, 'MarkerSize', 8);
grid on;
title('Bit Error Rate vs SNR');
xlabel('SNR (dB)');
ylabel('Bit Error Rate (BER)');
legend('ASK (OOK)', 'FSK (Orthogonal-FSK)', 'PSK (PRK)','ASK (OOK) Built-in', ...
    'FSK (Orthogonal-FSK) Built-in', 'PSK (PRK) Built-in','QAM Built-in');
```

Output:




```
Command Window
SNR for nearly error-free transmission:
ASK (OOK): 20 dB
FSK (Orthogonal-FSK): 14 dB
PSK (PRK): 14 dB
ASK (OOK) Built-in: 20 dB
FSK (Orthogonal-FSK) Built-in: 14 dB
PSK (PRK) Built-in: 14 dB
QAM Built-in: 14 dB
```