Exp 5

```
% Define the given set of vectors
A = [1 0 0; 1 1 1; 0 0 1]';
% Initialize an empty matrix to store orthogonal vectors
Q = zeros(size(A));
% Perform Gram-Schmidt Orthogonalization
for j = 1:size(A, 2)
    v = A(:, j);
    for i = 1:j-1
        q = Q(:, i);
        v = v - (v' * q) * q;
    end
    Q(:, j) = v / norm(v);
end
disp(Q)
% Plot the orthonormal vectors
figure;
hold on;
quiver3(0, 0, 0, Q(1,1), Q(2,1), Q(3,1), 'r--', 'LineWidth', 2);
quiver3(0, 0, 0, Q(1,2), Q(2,2), Q(3,2), 'g--', 'LineWidth', 2);
quiver3(0, 0, 0, Q(1,3), Q(2,3), Q(3,3), 'b--', 'LineWidth', 2);

quiver3(0, 0, 0, A(1,1), A(2,1), A(3,1), 'r', 'LineWidth', 2);
quiver3(0, 0, 0, A(1,2), A(2,2), A(3,2), 'g', 'LineWidth', 2);
quiver3(0, 0, 0, A(1,3), A(2,3), A(3,3), 'b', 'LineWidth', 2);
xlabel('X-axis');
ylabel('Y-axis');
title('Orthonormal Vectors');
grid on;
```

Exp 6

```
N = 1e4;
SNR_dB = 0:5:20;
pulse_width = 1;
data = randi([0 1], N, 1);
t = 0:0.01:pulse_width; pulse = ones(size(t));
tx_signal = reshape(repmat(data', length(t), 1) .* pulse', [], 1);
BER = zeros(length(SNR_dB), 1);


for k = 1:length(SNR_dB)
               noise = sqrt(1 / (2 * 10^(SNR_dB(k)/10))) *
randn(size(tx_signal));
rx_signal = tx_signal + noise;
filtered_signal = conv(rx_signal, pulse, 'same');
sampled_signal = filtered_signal(1:length(t):end);
BER(k) = mean((sampled_signal > 0.5) ~= data);
end
semilogy(SNR_dB, BER, 'b-o');
grid on;
xlabel('SNR (dB)');
ylabel('BER');
title('BER vs. SNR for Rectangular Pulse Modulated Data');
```

Exp 7

```
clc; clear; close all;

% Input bit sequence
bit_seq = [1 1 0 0 0 0 1 1];
fc = 1; % Carrier frequency
t = 0:0.001:1; % Time vector for each bit (1ms per bit)

% Map bits for QPSK: 0 -> -1
bit_seq(bit_seq == 0) = -1;

% Separate even and odd bits
b_e = bit_seq(2:2:end); % Even bits
b_o = bit_seq(1:2:end); % Odd bits

% Generate base waveforms
cos_wave = cos(2 * pi * fc * t); % Cosine wave for even bits
sin_wave = sin(2 * pi * fc * t); % Sine wave for odd bits

% Generate modulated waveforms
bec = kron(b_e, cos_wave); % Even bits modulated with cosine
bes = kron(b_o, sin_wave); % Odd bits modulated with sine
qpsk_signal = bec + bes; % QPSK modulated signal

% Generate bit waveforms for plotting
bit_wave = repelem(bit_seq, length(t)); % Full input bit waveform
bit_e_plot = repelem(b_e, length(t) ); % Even bits for plotting
bit_o_plot = repelem(b_o, length(t) ); % Odd bits for plotting

% Plot results
figure('Name', 'QPSK Modulation');

% Binary Input Sequence
subplot(5, 1, 1); plot(bit_wave, 'LineWidth', 1.5);
grid on; axis([0 length(bit_wave) -1.5 1.5]);
title('Binary Input Sequence');xlabel('Time'); ylabel('Amplitude');

% Odd Bits (Sine Component)
subplot(5, 1, 2);
plot(bes, 'b', 'LineWidth', 1.5); hold on;
plot(bit_o_plot, 'r--', 'LineWidth', 1.2);
grid on; axis([0 length(bes) -1.5 1.5]);
title('Odd Bits (Sine Component)');
xlabel('Time'); ylabel('Amplitude');

% Even Bits (Cosine Component)
subplot(5, 1, 3);
plot(bec, 'g', 'LineWidth', 1.5);hold on;
plot(bit_e_plot, 'r--', 'LineWidth', 1.2);
grid on; axis([0 length(bec) -1.5 1.5]);
```

```matlab
title('Even Bits (Cosine Component)');
xlabel('Time'); ylabel('Amplitude');

% QPSK Modulated Signal
subplot(5, 1, 4);
plot(qpsk_signal, 'k', 'LineWidth', 1.5);
grid on;axis([0 length(qpsk_signal) -2 2]);
title('QPSK Modulated Signal');
xlabel('Time'); ylabel('Amplitude');

% QPSK Constellation
subplot(5, 1, 5);
constellation = [1 + 1j, -1 + 1j, -1 - 1j, 1 - 1j];
plot(real(constellation), imag(constellation), 'bo', 'MarkerSize', 8,
'LineWidth', 2);
grid on; axis([-2 2 -2 2]);title('QPSK Constellation');
xlabel('In-phase (I)'); ylabel('Quadrature (Q)');
```

Exp 8

```
M = 16;
N = 1000;
bits = randi([0 1], 1, N);
symbols = zeros(1, N/4);
for i = 1:N/4
  symbols(i) = (2*bits(4*i-3)-1) + 1j*(2*bits(4*i-2)-1) +
                  2*(2*bits(4*i-1)-1) + 2j*(2*bits(4*i)-1);
end
scatter(real(symbols), imag(symbols), 'bo');
grid on;
xlabel('In-phase'); ylabel('Quadrature');
title('16-QAM Constellation');
```

Exp 9

```
clc;clear;

% Input probability distribution
e = [1,2,4,2,3];
p = [0.5 0.2 0.15 0.15];
n = length(p);

% Generate Huffman dictionary
symbols = 1:n;
[dict, avglen] = huffmandict(symbols, p);

% Display Huffman dictionary
disp('The Huffman code dictionary:');
for i = 1:n
   fprintf('Symbol %d: %s\n', symbols(i), num2str(dict{i, 2}));
end

% Encode symbols
sym = e;%input(sprintf('Enter the symbols between 1 to %d in []: ', n));
encod = huffmanenco(sym, dict);
disp('The encoded output:');
disp(encod);

% Decode bit stream
bits = encod;
decod = huffmandeco(bits, dict);
disp('The decoded symbols are:');
disp(decod);
```

Exp 10

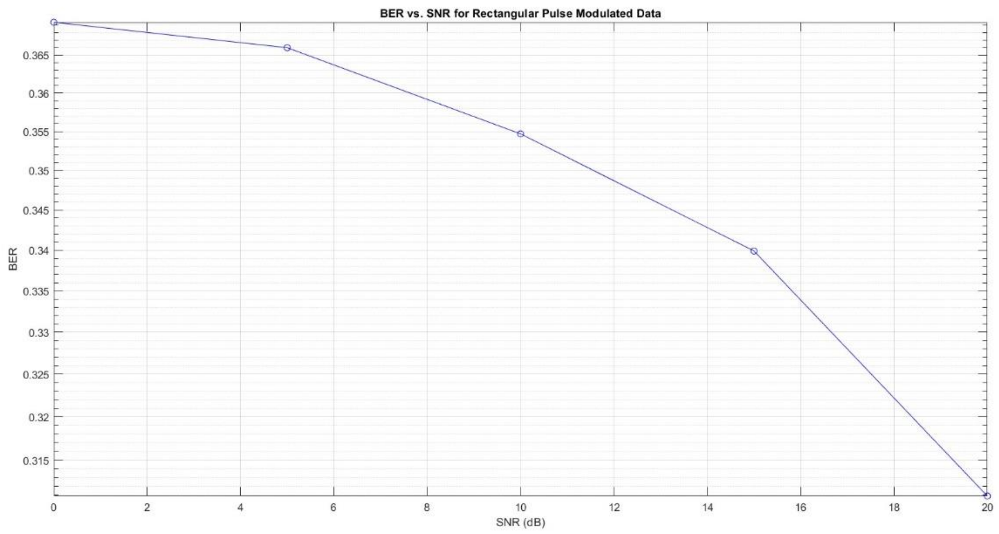
```
% Example 4-bit data
data = [1 0 1 0]  % Example data: 1010


% Encode data
G = [1 0 0 0  1 1 1;
     0 1 0 0  1 0 1;
     0 0 1 0  0 1 1;
     0 0 0 1  1 1 0];
encoded_data = mod(data * G, 2);
disp('Encoded Data:');
disp(encoded_data);


% Simulate bit error (flip 5th bit)
received_data = encoded_data;
received_data(5) = ~received_data(5);
disp('Received Data (with error):');
disp(received_data);


% Decode data
H = [1 1 1 0  1 0 0;
     1 1 0 1  0 1 0;
     1 0 1 1  0 0 1];
syndrome = mod(received_data * H', 2);
error_position = bi2de(syndrome, 'left-msb') + 1;


if any(syndrome)
    disp(['Error detected at position: ', num2str(error_position)]);
    received_data(error_position) = ~received_data(error_position);
% Correct error
else
    disp('No error detected');
end


decoded_data = received_data(1:4);
disp('Decoded Data:');
disp(decoded_data);
```

Exp 12

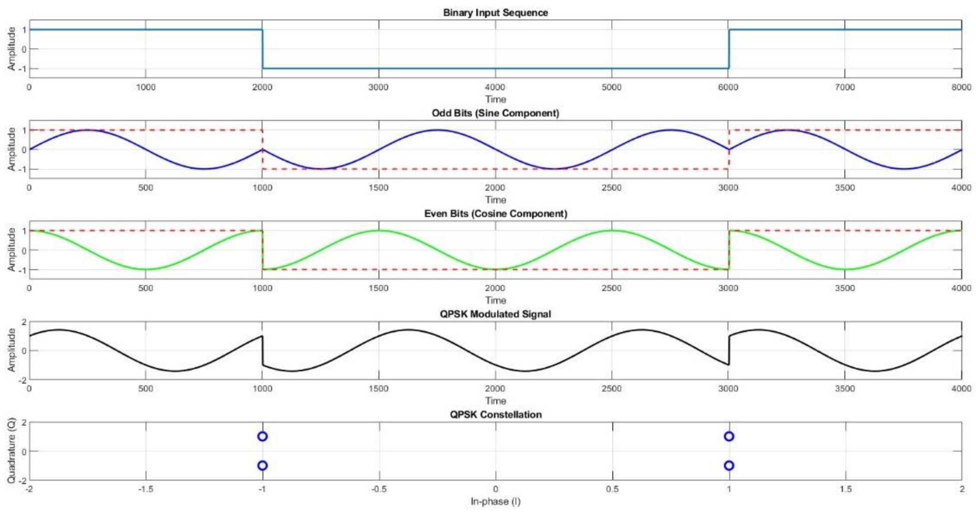
```matlab
clc; clear; close all;

% Input message to be encoded
msg = [1 0 1 1 0 1 0 0];

% Define constraint length and generator polynomial
constraint_length = 3;
generator_polynomials = [7 5];

% Create trellis structure for the convolutional encoder
trellis = poly2trellis(constraint_length, generator_polynomials);

% Encode the message using convolutional encoder
encoded_msg = convenc(msg, trellis);

% Simulate noise by flipping a bit in the encoded message
encoded_msg_noisy = encoded_msg;
encoded_msg_noisy(4) = ~encoded_msg_noisy(4);
% Flip the 4th bit to simulate noise

% Perform Viterbi decoding on the noisy message
traceback_length = 5;
decoded_msg = vitdec(encoded_msg_noisy, trellis, traceback_length, 'trunc',
'hard');

% Display results
disp('Original Message:');
disp(msg);
disp('Encoded Message:');
disp(encoded_msg);
disp('Noisy Encoded Message (with bit flip):');
disp(encoded_msg_noisy);
disp('Decoded Message:');
disp(decoded_msg);
```

OUTPUTS:

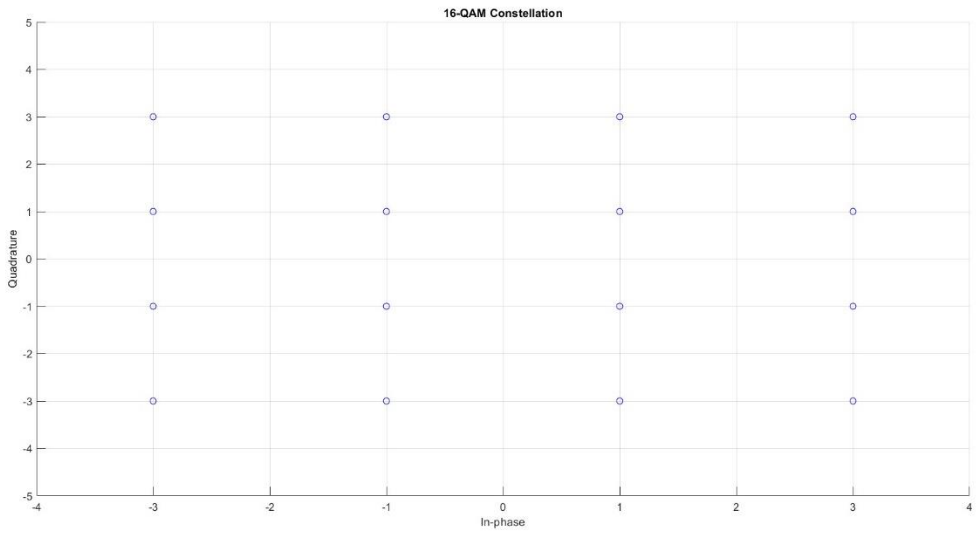
Exp 6



Exp 7



Exp8



Exp 9

```
The Huffman code dictionary:

Symbol 1: 0
Symbol 2: 1  1
Symbol 3: 1  0  1
Symbol 4: 1  0  0

The encoded output:

     0     1     1     1     0     0     1     1     1     0     1

The decoded symbols are:

     1     2     4     2     3
```

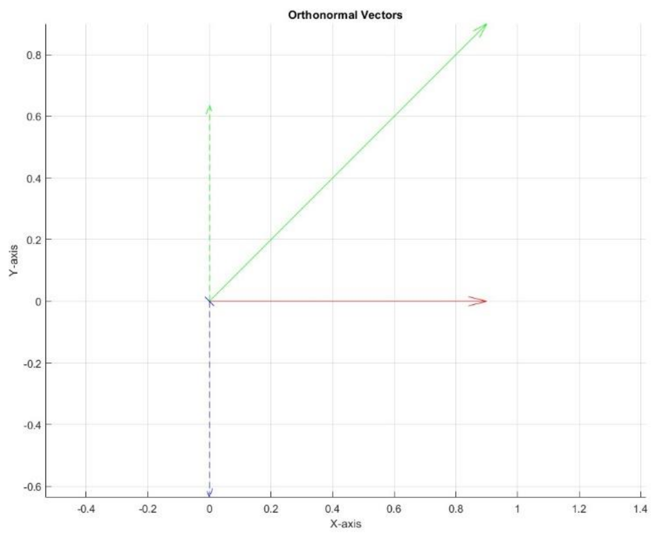
Exp10

```
 data = 1×4
        1     0     1     0


Encoded Data:

     1     0     1     0     1     0     0

Received Data (with error):

     1     0     1     0     0     0     0

Error detected at position: 3

Decoded Data:

     1     0     0     0
```

Exp 12

```
Original Message:

     1     0     1     1     0     1     0     0

Encoded Message:

     1     1     1     0     0     0     0     1     0     1     0     0     1     0     1

Noisy Encoded Message (with bit flip):

     1     1     1     1     0     0     0     1     0     1     0     0     1     0     1

Decoded Message:

     1     0     1     1     0     1     0     0
```

Exp5


Orthonormal Vectors
Y-axis
X-axis