

EXP 05

```
% 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 06
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 07

```
clc; clear; close all;

bit_seq = [1 1 0 0 0 0 1 1];
fc = 1; t = 0:0.001:1;
bit_seq(bit_seq == 0) = -1;
b_e = bit_seq(2:2:end); b_o = bit_seq(1:2:end);
qpsk_signal = kron(b_e, cos(2*pi*fc*t)) + kron(b_o, sin(2*pi*fc*t));
figure;
subplot(5,1,1);
plot(repelem(bit_seq, length(t)), 'LineWidth', 1.5); grid on;
subplot(5,1,2);
plot(kron(b_o, sin(2*pi*fc*t)), 'b'); hold on;
plot(repelem(b_o, 2*length(t)), 'r--'); grid on;
subplot(5,1,3); plot(kron(b_e, cos(2*pi*fc*t)), 'g'); hold on;
plot(repelem(b_e, 2*length(t)), 'r--'); grid on;
subplot(5,1,4);
plot(qpsk_signal, 'k', 'LineWidth', 1.5); grid on;
subplot(5,1,5);
plot([-1 1 1 -1], [1 1 -1 -1], 'bo'); grid on;
```

EXP 08

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');

```

EP 9
clc;

clear;

% Input probability distribution
p = input('Enter the probabilities: ');
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 = 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 = input('Enter the bit stream in []: ');
decod = huffmandeco(bits, dict);
disp('The decoded symbols are:');
disp(decod);

```

EXP 10

```
% Hamming Encoding

% Define the data to encode

data = [1 0 1 0];

% Calculate the parity bits

p1 = mod(data(1) + data(3) + data(4), 2);

p2 = mod(data(1) + data(2) + data(4), 2);

p3 = mod(data(1) + data(2) + data(3), 2);

% Create the encoded data

encoded_data = [p1 p2 data(1) p3 data(2) data(3) data(4)];

disp('Encoded Data:');

disp(encoded_data);

% Hamming Decoding

% Define the encoded data with error

encoded_data = [1 0 1 0 1 0 1];

% Calculate the syndrome

s1 = mod(encoded_data(1) + encoded_data(3) + encoded_data(5) + encoded_data(7), 2);

s2 = mod(encoded_data(2) + encoded_data(3) + encoded_data(6) + encoded_data(7), 2);

s3 = mod(encoded_data(4) + encoded_data(5) + encoded_data(6) + encoded_data(7), 2);

% Determine the error location

error_location = bin2dec([num2str(s1) num2str(s2) num2str(s3)]);

% Correct the error

if error_location ~= 0

    encoded_data(error_location) = mod(encoded_data(error_location) + 1, 2);

end

% Extract the decoded data

decoded_data = encoded_data([3 5 6 7]);

disp('Decoded Data:');

disp(decoded_data);
```

EXP 12

```
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);
```