EXPERIMENT 01

Gram Schmitt ortogonalization

```
clc; clear;

V = [1 0 0; 1 1 1; 0 0 1]';

num_vectors = size(V, 2);

U = zeros(size(V));

E = zeros(size(V));

U(:,1) = V(:,1);

E(:,1) = U(:,1) / norm(U(:,1));

for i = 2:num_vectors

  U(:,i) = V(:,i);

  for j = 1:i-1

    U(:,i) = U(:,i) - (dot(E(:,j), V(:,i))) * E(:,j);

  end

  E(:,i) = U(:,i) / norm(U(:,i));

end

disp('Orthonormal basis vectors (columns):');

disp(E);

figure; hold on; grid on; axis equal;

quiver3(0,0,0, V(1,1), V(2,1), V(3,1), 'r', 'LineWidth', 2);

quiver3(0,0,0, V(1,2), V(2,2), V(3,2), 'g', 'LineWidth', 2);

quiver3(0,0,0, V(1,3), V(2,3), V(3,3), 'b', 'LineWidth', 2);

quiver3(0,0,0, E(1,1), E(2,1), E(3,1), 'r--', 'LineWidth', 2);

quiver3(0,0,0, E(1,2), E(2,2), E(3,2), 'g--', 'LineWidth', 2);

quiver3(0,0,0, E(1,3), E(2,3), E(3,3), 'b--', 'LineWidth', 2);

xlabel('X'); ylabel('Y'); zlabel('Z');

legend('v1','v2','v3','e1','e2','e3');

title('Original and Orthonormal Basis Vectors');
```

EXPERIMENT 02

Qpsk modulation and demodulation

```
clc;
clear all;
close all;
% Input bit sequence
bit_seq = [1 1 0 0 0 0 1 1];
N = length(bit_seq);
fc = 1; % Carrier frequency
t = 0:0.001:2; % Time vector
b = []; % Input bit sequence as a waveform
qpsk1 = []; % QPSK signal
bec = []; % Even bit cosine waveform
bes = []; % Odd bit sine waveform
b_o = []; % Odd bit stream
b_e = []; % Even bit stream
bit_e = []; % Even bit stream (for plotting)
bit_o = []; % Odd bit stream (for plotting)
% Creating input waveform from bit sequence
for i = 1:N
  bx = bit_seq(i) * ones(1, 1000); % Generate pulse for each bit
  b = [b, bx];
end
% Modifying bits for QPSK mapping: 0 -> -1
for i = 1:N
  if bit_seq(i) == 0
    bit_seq(i) = -1;
  end
  if mod(i, 2) == 0
```

```
        e_bit = bit_seq(i);
        b_e = [b_e, e_bit];
    else
        o_bit = bit_seq(i);
        b_o = [b_o, o_bit];
    end
end
% Generate QPSK modulated signal
for i = 1:N/2
    % Even bits modulated on cosine wave
    be_c = (b_e(i) * cos(2*pi*fc*t));
    % Odd bits modulated on sine wave
    bo_s = (b_o(i) * sin(2*pi*fc*t));
    q = be_c + bo_s; % Combine both to form QPSK signal
    % Collect even and odd bit streams for plotting
    even = b_e(i) * ones(1, 2000);
    odd = b_o(i) * ones(1, 2000);
    bit_e = [bit_e, even];
    bit_o = [bit_o, odd];
    qpsk1 = [qpsk1, q];
    bec = [bec, be_c];
    bes = [bes, bo_s];
end
% Plotting the QPSK signals and constellations
figure('name', 'QPSK Modulation');
subplot(5, 1, 1);
plot(b, 'o');
grid on;
axis([0 N*1000 0 1]);
title('Binary Input Sequence');
```

```
subplot(5, 1, 2);
plot(bes);
hold on;
plot(bit_o, 'rs:');
grid on;
axis([0 N*1000 -1 1]);
title('Odd Bits (Sine Component)');
subplot(5, 1, 3);
plot(bec);
hold on;
plot(bit_e, 'rs:');
grid on;
axis([0 N*1000 -1 1]);
title('Even Bits (Cosine Component)');
subplot(5, 1, 4);
plot(qpsk1);
axis([0 N*1000 -1.5 1.5]);
title('QPSK Modulated Signal');
% QPSK Constellation Plot
subplot(5, 1, 5);
% QPSK constellation points
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)');
```

EXPERIMENT 03

BER VS SNR

```
% Simplified Parameters

N = 1e4;

SNR_dB = 0:5:20;

pulse_width = 1;

% Number of bits

% SNR values in dB

% Pulse width for rectangular pulse

% Generate random binary data

data = randi([0 1], N, 1);

% Define the rectangular pulse

t = 0:0.01:pulse_width;

rect_pulse = ones(size(t));

% Initialize BER vector

BER = zeros(length(SNR_dB), 1);

for snr_idx = 1:length(SNR_dB)

% Modulate binary data

tx_signal = [];

for i = 1:N

if data(i) == 1

tx_signal = [tx_signal; rect_pulse'];

else

tx_signal = [tx_signal; zeros(size(rect_pulse'))];

end

end

% Add AWGN
```

```
SNR = 10^(SNR_dB(snr_idx) / 10);

noise_power = 1 / (2 * SNR);

noise = sqrt(noise_power) * randn(length(tx_signal), 1);

rx_signal = tx_signal + noise;

% Matched Filter

matched_filter = rect_pulse;

filtered_signal = conv(rx_signal, matched_filter, 'same');

% Sample the output of the matched filter

sample_interval = round(length(filtered_signal) / N);

sampled_signal = filtered_signal(1:sample_interval:end);

% Decision (Threshold = 0.5)

estimated_bits = sampled_signal > 0.5;

% Compute BER

num_errors = sum(estimated_bits ~= data);

BER(snr_idx) = num_errors / N;

end

% Plot BER vs. SNR

figure;

semilogy(SNR_dB, BER, 'b-o');

grid on;

xlabel('SNR (dB)');

ylabel('Bit Error Rate (BER)');

title('BER vs. SNR for Rectangular Pulse Modulated Binary Data');
```

EXPERIMENT 04

16- qam constellation

```
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');


% Simulate AWGN CHANNEL

snr_db = 20;

rx_signal = awgn(symbols, snr_db, 'measured');

figure;

plot(real(rx_signal), imag(rx_signal), 'bo', 'MarkerSize', 6, 'LineWidth', 2)

xlabel('In-phase');

ylabel('Quadrature');

title('16 QAM constellation with noise')

grid on;

axis equal;

axis([-4 4 -4 4]);
```

EXPERIMENT 05

Huffman coding

```
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);
```

EXPERIMENT 06

Hamming coding

```
% Hamming Encoding

data = [1 0 1 0]

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);

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

disp('Encoded Data:');

disp(encoded_data);

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

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)

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

if error_location ~= 0

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

end

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

disp('Decoded Data:');

disp(decoded_data);
```

EXPERIMENT 07

Convolution code

```
msg = [1 0 1 1 0 1 0 0];

constraint_length = 3;

generator_polynomials = [7 5];

trellis = poly2trellis(constraint_length, generator_polynomials);

encoded_msg = convenc(msg, trellis);

encoded_msg_noisy = encoded_msg;

encoded_msg_noisy(4) = ~encoded_msg_noisy(4);

traceback_length = 5;

decoded_msg = vitdec(encoded_msg_noisy, trellis, traceback_length, 'trunc', 'hard');

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);
```