

```

Experiment No. 5: Gram-Schmidt Orthogonalization
% Given set of vectors as columns
V = [1 1 0; 1 0 1; 0 1 1];
% Number of vectors
num_vectors = size(V, 2);
% Initialize orthogonal and orthonormal matrices
U = zeros(size(V));
E = zeros(size(V));
% Gram-Schmidt Process
U(:,1) = V(:,1); % First orthogonal vector is the first input vector
E(:,1) = U(:,1) / norm(U(:,1)); % First orthonormal vector
for i = 2:num_vectors
    U(:,i) = V(:,i);
    for j = 1:i-1
        U(:,i) = U(:,i) - (dot(V(:,i), E(:,j)) * E(:,j)); % Subtract projection onto previous
    end
    E(:,i) = U(:,i) / norm(U(:,i)); % Normalize to get orthonormal vector
end
% Display the orthonormal vectors
disp('Orthonormal basis vectors:');
disp(E);
% Plot the original and orthonormal vectors
figure;
hold on;
grid on;
axis equal;
% Plot original vectors
quiver(0, 0, 0, V(1,1), V(2,1), V(3,1), 'r', 'LineWidth', 2);
quiver(0, 0, 0, V(1,2), V(2,2), V(3,2), 'g', 'LineWidth', 2);
quiver(0, 0, 0, V(1,3), V(2,3), V(3,3), 'b', 'LineWidth', 2);
% Plot orthonormal vectors
quiver(0, 0, 0, E(1,1), E(2,1), E(3,1), 'r-', 'LineWidth', 2);
quiver(0, 0, 0, E(1,2), E(2,2), E(3,2), 'g-', 'LineWidth', 2);
quiver(0, 0, 0, E(1,3), E(2,3), E(3,3), 'b-', 'LineWidth', 2);
xlabel('X');
ylabel('Y');
zlabel('Z');
legend('Original V1', 'Original V2', 'Original V3', 'Orthonormal E1',
'Orthonormal E2', 'Orthonormal E3');
title('Original and Orthonormal Vectors');
hold off;

```

```

Experiment No. 9: Huffman Coding and Decoding
clc;
p = input('Enter the probabilities: ');
n = length(p);
symbols = [1:n];
[dict, avglen] = huffmandict(symbols, p);
temp = dict;
t = dict(:,2);
for i = 1:length(temp)
    temp{i,2} = num2str(temp{i,2});
end
disp('The Huffman code dict: ');
disp(temp);
symb = input('Enter the symbols between 1 to %d\n', n);
encode = huffmanenco(symb, dict);
disp('The encoded string: ');
disp(encode);
bits = input('Enter the bit stream : ');
decode = huffmanenco(bits, dict);
disp('The symbols are: ');
disp(0);
H = 0;
I = 0;
for k = 1:n
    H = H + (p(k) * log2(1/p(k)));
end
fprintf('Entropy is %f bits\n', H);
et = H/lavglen;
fprintf('Efficiency is %f\n', et);
for k = 1:n
    l(k) = length(0{k});
end
m = max(l);
s = min(l);
v = m - s;
fprintf('The variance is %f\n', v);

```

```

Experiment No. 6: Simulation of Binary Baseband Signal
N = 1e4;
SNR_dB = 0:5:20;
pulse_width = 1;
data = randi([0 1], N, 1);
t = 0:0.01:pulse_width;
rect_pulse = ones(size(t));
BER = zeros(length(SNR_dB), 1);

for snr_idx = 1:length(SNR_dB)
    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
    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 = rect_pulse;
    filtered_signal = conv(rx_signal, matched_filter, 'same');
    sample_interval = round(length(filtered_signal) / N);
    sampled_signal = filtered_signal(1:sample_interval:end);
    estimated_bits = sampled_signal > 0.5;
    num_errors = sum(estimated_bits ~= data);
    BER(snr_idx) = num_errors / N;
end
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 No. 8: 16-QAM Modulation
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 No. 10: Hamming Code Encoding and Decoding
data = [1 0 1 0];
p1 = mod(data(1) + data(2) + data(4), 2);
p2 = mod(data(1) + data(3) + data(4), 2);
p3 = mod(data(2) + data(3) + data(4), 2);
encoded_data = [p1 p2 data(1) p3 data(2) data(3) data(4)];
disp('encoded data:');
disp(encoded_data);

received_data = [1 0 1 1 0 1 0];
disp('received data');
disp(received_data);

s1 = mod(received_data(1) + received_data(3) + received_data(5) +
received_data(7), 2);
s2 = mod(received_data(2) + received_data(3) + received_data(6) +
received_data(7), 2);
s3 = mod(received_data(4) + received_data(5) + received_data(6) +
received_data(7), 2);
error_location = bin2dec([num2str(s3) num2str(s2) num2str(s1)]);
if error_location ~= 0
    encoded_data(error_location) = mod(received_data(error_location) +
1, 2);
end

disp('s1');
disp(s1);
disp('s2');
disp(s2);
disp('s3');
disp(s3);

decoded_data = received_data([3 5 6 7]);
disp('decoded data');
disp(decoded_data);

```

```

Experiment No. 12: Convolutional Code
clc; clear; close all;
msg = [1 0 1 1 0 0];
constraint_length = 3;
generator_polynomial = [4 5 7];
trellis = poly2trellis(constraint_length,
generator_polynomial);
encoded_msg = convenc(msg, trellis);
encoded_msg_noisy = encoded_msg;
encoded_msg_noisy(4) =
~encoded_msg(4);
backtrack = 5;
decoded_msg =
vitdec(encoded_msg_noisy, trellis,
backtrack, 'term');
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);

```