DIGITAL COMMUNICATION LAB

EXPERIMENT 9(PROJECT)

GROUP:

G.HARI KRISHNA SAI – 22EC01011

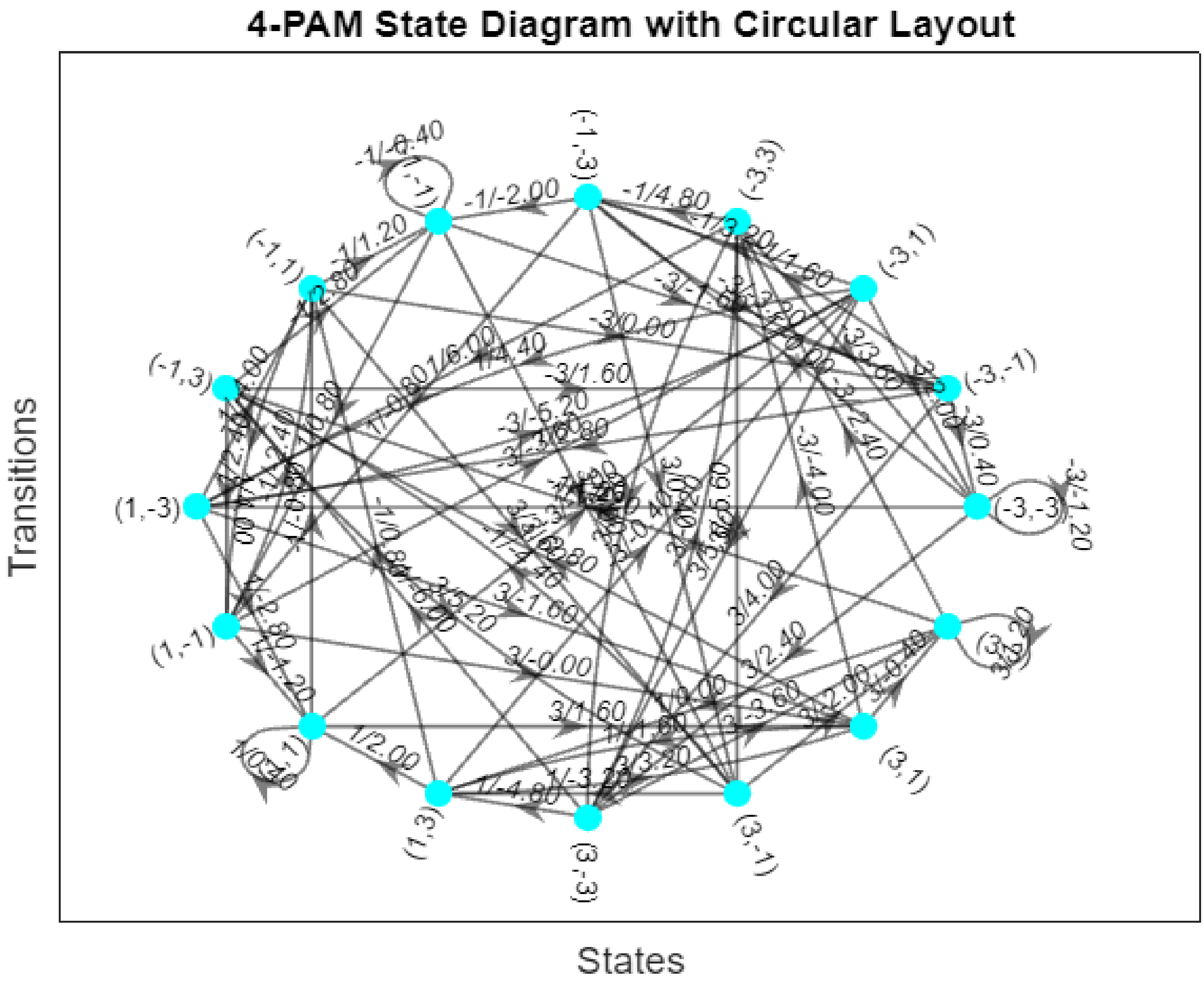
D.KASIF ANSAR – 22EC01035

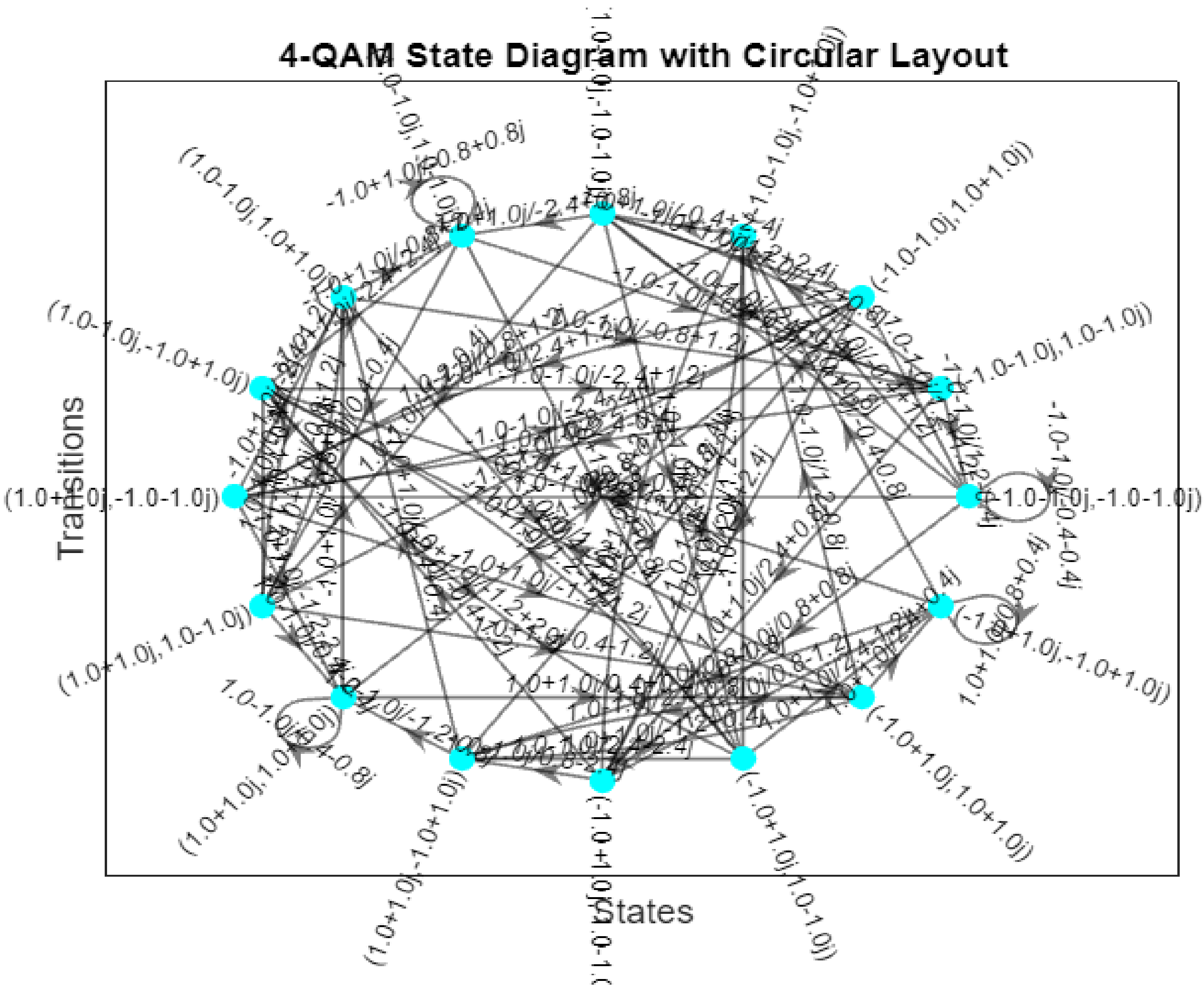
PART-1 : STATE DIAGRAM AND TRELLIS DIAGRAM:

```
% Define constellation and parameters for both 4-PAM and 4-QAM
constellation_type = '4-QAM'; % Change to '4-PAM' or '4-QAM' based on
requirement

switch constellation_type
    case '4-PAM'
        symbols = [-3, -1, 1, 3];
        bit_to_symbol_map = [-3, -1, 1, 3]; % Mapping for 4-PAM (00 -> -3,
01 -> -1, 10 -> +1, 11 -> +3)
        restricted_symbol = -3; % Symbol restriction for the final stages in
4-PAM
        num_bits_per_symbol = 2;
    case '4-QAM'
        symbols = [-1-1j, -1+1j, 1-1j, 1+1j];
        bit_to_symbol_map = [-1-1j, -1+1j, 1-1j, 1+1j]; % Mapping for 4-QAM
        restricted_symbol = -1-1j; % Symbol restriction for the final stages
in 4-QAM
        num_bits_per_symbol = 2;
    otherwise
        error('Unsupported constellation type');
end

num_states = length(symbols)^2;

% Define the channel transfer function coefficients
h = [0.6, -1, 0.8];

% Generate all possible states (pairs of symbols) for 2 memory states
states = combvec(symbols, symbols)'; % (symbol1, symbol2) pairs

% Convert complex numbers to strings in readable format for state names
state_names = arrayfun(@(a, b) sprintf('(%s,%s)', complex_to_str(a),
complex_to_str(b)), states(:, 1), states(:, 2), 'UniformOutput', false);

% Sort the states for consistency
[~, sort_idx] = sortrows(states, [1, 2]);
```

```
states = states(sort_idx, :);
state_names = state_names(sort_idx);

%% State Diagram
% Initialize the graph for the state diagram
G_state = digraph;
inputs_outputs = strings(0);

% Add all states as nodes in the graph
for i = 1:num_states
    G_state = addnode(G_state, state_names{i});
end

% Add transitions with input/output labels
for i = 1:num_states
    current_state = states(i, :);
    for input_symbol = symbols
        % Calculate the next state
        new_state = [input_symbol, current_state(1)];

        % Find the index of the new state
        next_state_idx = find(ismember(states, new_state, 'rows'));

        % Calculate the output using the channel transfer function
        output = input_symbol * h(1) + current_state(1) * h(2) +
current_state(2) * h(3);

        % Format output as readable string
        output_str = complex_to_str(output);

        % Create the input/output label for this transition
        input_output_label = sprintf('%s/%s', complex_to_str(input_symbol),
output_str);
        inputs_outputs(end + 1) = input_output_label;

        % Add edge with the custom label
        G_state = addedge(G_state, state_names{i},
state_names{next_state_idx});
    end
end

% Plot the state diagram with circular layout
figure;
p_state = plot(G_state, 'Layout', 'circle', 'NodeLabel',
G_state.Nodes.Name);
title(sprintf('%s State Diagram with Circular Layout', constellation_type));
xlabel('States');
ylabel('Transitions');
```

```
% Set custom edge labels with input/output
p_state.EdgeLabel = inputs_outputs;
p_state.MarkerSize = 7;
p_state.NodeColor = 'cyan';
p_state.EdgeFontSize = 8;
p_state.NodeFontSize = 8;
p_state.LineWidth = 1;
p_state.EdgeColor = [0, 0, 0];
p_state.ArrowSize = 10;
```



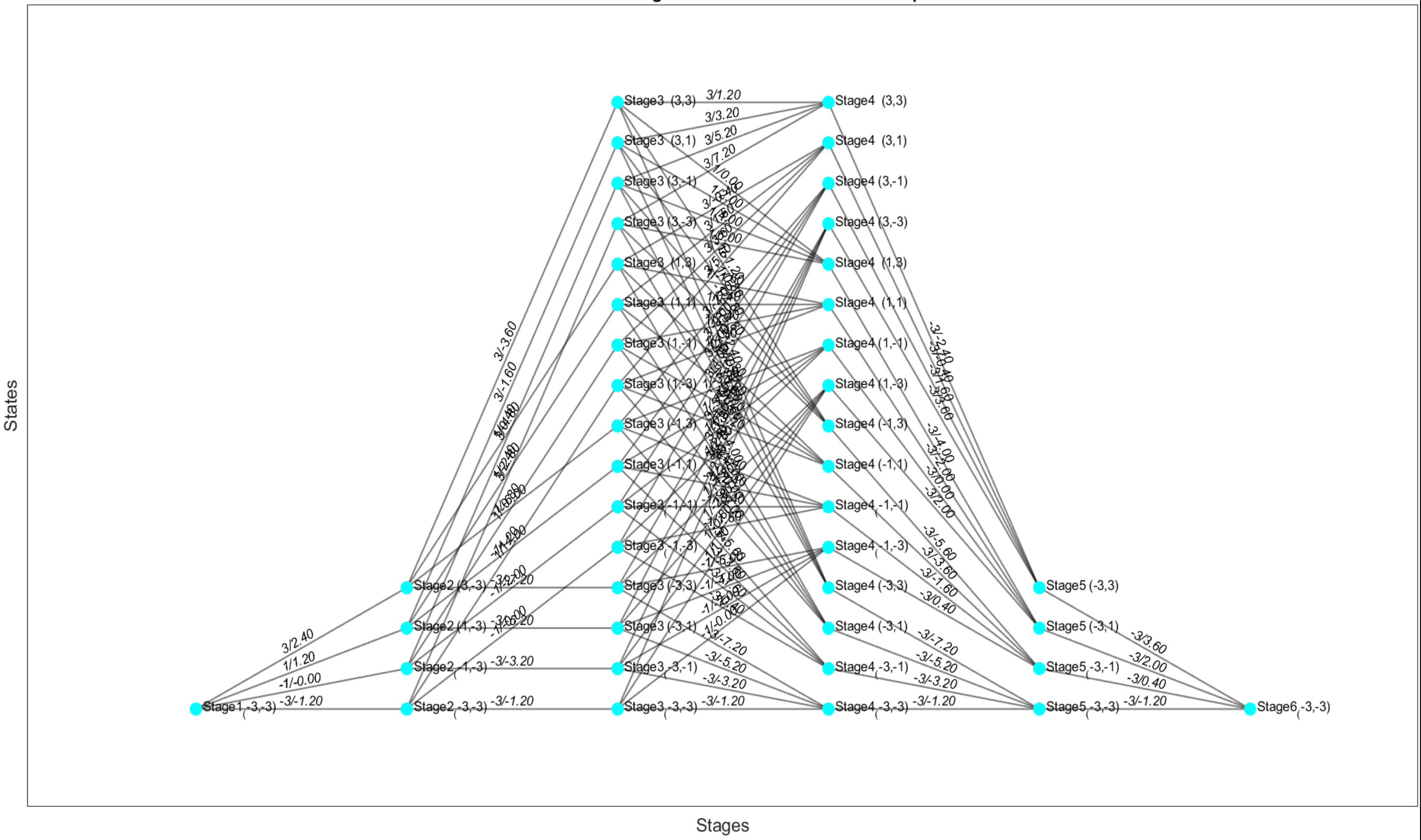



```
%% Trellis Diagram
% Define the number of stages (time steps)
num_stages = 9;

% Generate random input bits for the transmission
input_bits = randi([0, 1], 1, num_bits_per_symbol * num_stages);


% Convert bits to symbols using the bit-to-symbol map
input_symbols = bit_to_symbol_map(bi2de(reshape(input_bits,
num_bits_per_symbol, []).', 'left-msb') + 1);

% Initialize the graph for the trellis diagram
G_trellis = digraph;
initial_state = 1; % Index of the initial state
initial_state_name = sprintf('Stage1_%s', state_names{initial_state});
G_trellis = addnode(G_trellis, initial_state_name);
```

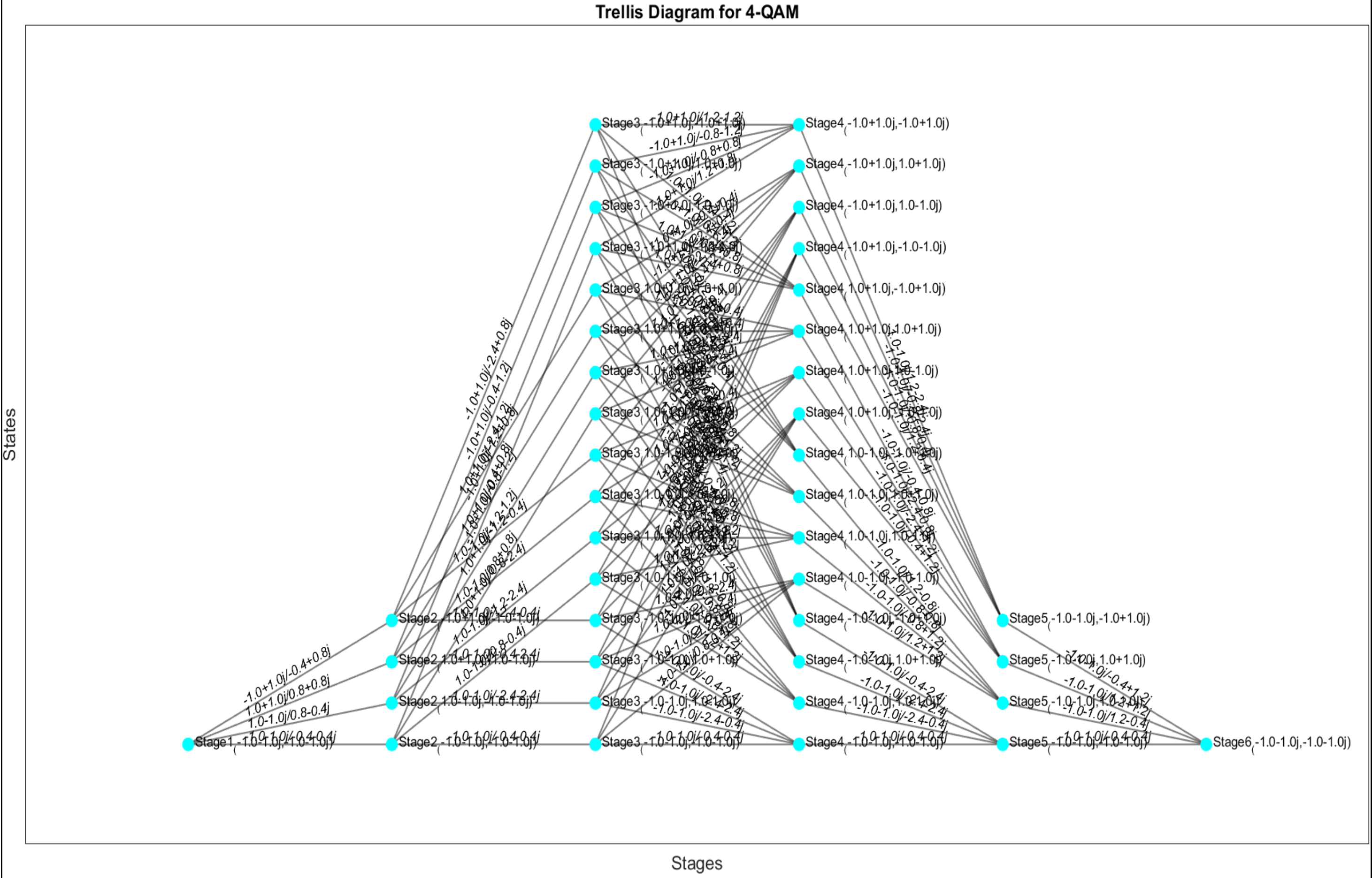
```
inputs_outputs_trellis = strings(0);

% Define the SNR value (in dB)
SNR_dB = 0;
SNR_linear = 10^(SNR_dB / 10);
noise_variance = 1 / SNR_linear;


% Incrementally build the trellis
for stage = 1:num_stages
    start_index = 8 + floor(log10(stage));
    current_stage_nodes = findnode(G_trellis, strcat('Stage',
num2str(stage), '_', state_names));

    for i = current_stage_nodes'
        if i == 0, continue; end

        [~, state_idx] = ismember(G_trellis.Nodes.Name{i}(start_index:end),
state_names);
        current_state = states(state_idx, :);
        a = current_state(1);
        b = current_state(2);

        % Apply restriction in the final stages
        if stage >= num_stages - 1
            current_symbols = restricted_symbol;
        else
            current_symbols = symbols;
        end

        for input_symbol = current_symbols
            new_state = [input_symbol, a];
            next_state_idx = find(ismember(states, new_state, 'rows'));
            if isempty(next_state_idx), continue; end
            next_state_name = sprintf('Stage%d_%s', stage + 1,
state_names{next_state_idx});

            output = input_symbol * h(1) + a * h(2) + b * h(3);

            % Format output as readable string
            output_str = complex_to_str(output);

            input_output_label = sprintf('%s/%s',
complex_to_str(input_symbol), output_str);
            inputs_outputs_trellis(end + 1) = input_output_label;

            if ~ismember(next_state_name, G_trellis.Nodes.Name)
                G_trellis = addnode(G_trellis, next_state_name);
            end
```

```
            source_node = G_trellis.Nodes.Name{i};
            target_node = next_state_name;
            G_trellis = addedge(G_trellis, source_node, target_node);
        end
    end
end


% Plot the trellis
figure;
p_trellis = plot(G_trellis, 'Layout', 'layered', 'NodeLabel',
G_trellis.Nodes.Name);
title(sprintf('Trellis Diagram for %s', constellation_type));
xlabel('Stages');
ylabel('States');
p_trellis.EdgeLabel = inputs_outputs_trellis;

x_offsets = 0:num_stages;
y_values = linspace(-num_states/2, num_states/2, num_states);


for stage = 1:num_stages + 1
    stage_nodes = findnode(G_trellis, strcat('Stage', num2str(stage), '_',
state_names));
    stage_nodes = stage_nodes(stage_nodes > 0);
    for idx = 1:length(stage_nodes)
        p_trellis.XData(stage_nodes(idx)) = x_offsets(stage);
        p_trellis.YData(stage_nodes(idx)) = y_values(idx);
    end
end


p_trellis.MarkerSize = 7;
p_trellis.NodeColor = 'cyan';
p_trellis.EdgeFontSize = 8;
p_trellis.NodeFontSize = 8;
p_trellis.LineWidth = 1;
p_trellis.EdgeColor = [0, 0, 0];
p_trellis.ArrowSize = 0.4;
```

Incremental Trellis Diagram for 4-PAM with Random Input Bits



Trellis Diagram for 4-QAM



```
%% Helper function to format complex numbers as readable strings
function str = complex_to_str(x)
    if imag(x) >= 0
        str = sprintf('%.1f+%.1fj', real(x), imag(x));
    else
        str = sprintf('%.1f%.1fj', real(x), imag(x));
    end
end
```

EXPLANATION:

- The code initializes the modulation scheme (either 4-PAM or 4-QAM) by setting constellation_type. Based on the chosen scheme, it defines the corresponding symbol set, bit-to-symbol mapping, a restricted symbol for specific transitions, and the number of bits per symbol.
- The code specifies the channel transfer function using coefficient values stored in h. These coefficients represent the weights of a 3-tap channel model, which affects the symbol transitions and outputs.
- Using the symbol set, all possible combinations of two consecutive symbols (for 2 memory states) are generated to form the set of states. Each state is represented as a pair, and these are used to create unique state names.
- The state names are formatted for readability and sorted to ensure consistency across state transitions in the graphs.
- An empty directed graph (G_state) is created for the state diagram, and nodes are added to represent each state.
- For each state, transitions to other states are calculated by:
    - Determining the next state based on the current state and input symbol.
    - Computing the output using the channel response.
    - Creating and formatting an input-output label that includes both the input symbol and the calculated output.
    - Adding a directed edge with this label between the current and next state nodes.
- The completed state diagram is displayed in a circular layout with customized visual properties.
- The trellis diagram is configured with a specified number of stages. Random input bits are generated, converted to symbols using the chosen constellation's bit-to-symbol mapping, and arranged for trellis transmission.
- A directed graph (G_trellis) is initialized with nodes representing states at each stage. The initial state is designated and added as the first node in the graph

- complex_to_str is a helper function that converts complex numbers to a readable string format, ensuring clarity in state labels and edge labels.

PART – 2: VITERBI DECODING:

```
% Define the constellation type
constellation_type = '4-PAM'; % Change to '4-PAM' or '4-QAM' based on
requirement

% Define symbols and memory states based on the constellation type
switch constellation_type
    case '4-PAM'
        symbols = [-3, -1, 1, 3];
    case '4-QAM'
        symbols = [-1-1j, -1+1j, 1-1j, 1+1j];
    otherwise
        error('Unsupported constellation type');
end

num_states = length(symbols)^2;

% Define the channel transfer function coefficients
h = [0.6, -1, 0.8];

% Generate all possible states (pairs of symbols) for 2 memory states
states = combvec(symbols, symbols)'; % (symbol1, symbol2) pairs
state_names = arrayfun(@(a, b) sprintf('(%s,%s)', num2str(a), num2str(b)),
states(:, 1), states(:, 2), 'UniformOutput', false);

% Sort the states so that (-3, -3) is first and (3, 3) is last
[~, sort_idx] = sortrows(states, [1, 2]);
states = states(sort_idx, :);
state_names = state_names(sort_idx);

% Define the number of stages (time steps)
num_stages = 10000; % Large number to match Code 2

% Generate random input symbols for the transmission, with the last two set
to a fixed symbol
input_symbols = symbols(randi(length(symbols), 1, num_stages));
input_symbols(num_stages) = symbols(1); % Restrict last two symbols for
error-rate analysis
input_symbols(num_stages - 1) = symbols(1);

% Transmit signal through the channel
tx = input_symbols;
sk = conv(h, tx); % Convolution with channel
sk = sk(1:num_stages); % Truncate to match length

% Define the SNR values for the plot
```

```
 SNR_dB_values = 2:0.25:16;
 SER_values = zeros(size(SNR_dB_values));

 % Loop over different SNR values
 for idx = 1:length(SNR_dB_values)
     SNR_dB = SNR_dB_values(idx);
     SNR_linear = 10^(SNR_dB / 10);
     N0 = 1 / SNR_linear;
     noise_variance = sqrt(N0 / 2);

     % Generate and add noise to the transmitted signal
     noise = noise_variance * (randn(1, num_stages) + 1j * randn(1,
num_stages));
     received_signal = sk + noise; % Received signal with noise

     % Viterbi Decoding
     cost = inf(num_stages + 1, num_states);   % Cost for each state at each
stage
     prev_state = zeros(num_stages + 1, num_states); % Traceback for previous
states
     cost(1, 1) = 0; % Start with zero cost at the initial state

     % Define the cost function for both real and imaginary parts
     calculateCost = @(rx, tx) abs(rx - tx)^2;

     % Perform Viterbi decoding with cost updates
     for t = 1:num_stages
         % Iterate over next states and find minimum-cost path
         for next_state = 1:num_states
             minTransitionCost = inf;
             bestPrevState = 0;

             % Consider all possible previous states
             for prev_state_idx = 1:num_states
                 % Transition calculation
                 current_state = states(prev_state_idx, :);
                 next_input = states(next_state, 1); % Input symbol for next
state
                 predicted_output = h(1) * next_input + h(2) *
current_state(1) + h(3) * current_state(2);

                 % Calculate transition cost
                 transition_cost = calculateCost(received_signal(t),
predicted_output);
                 cumulative_cost = cost(t, prev_state_idx) + transition_cost;

                 % Update minimum cost and path if this path is cheaper
                 if cumulative_cost < minTransitionCost
                     minTransitionCost = cumulative_cost;
```

```
                    bestPrevState = prev_state_idx;
                end
            end

            % Update cost and traceback if valid previous state found
            if bestPrevState > 0
                cost(t + 1, next_state) = minTransitionCost;
                prev_state(t + 1, next_state) = bestPrevState;
            end
        end
    end

    % Backtrack to find the most likely path and decoded symbols
    finalCosts = cost(num_stages + 1, :);
    [~, minEndIndex] = min(finalCosts);
    finalState = minEndIndex;

    decoded_symbols = zeros(1, num_stages);
    for t = num_stages:-1:1
        decoded_symbols(t) = states(finalState, 1);
        finalState = prev_state(t + 1, finalState);
    end

    % Calculate the Symbol Error Rate (SER)
    symbol_errors = sum(decoded_symbols ~= input_symbols);
    SER_values(idx) = (symbol_errors / num_stages);
    disp(SER_values);
end

% Plot SER vs. SNR
figure;
semilogy(SNR_dB_values, SER_values, 'b-o', 'LineWidth', 2,
'MarkerFaceColor', 'b');
xlabel('SNR (dB)');
ylabel('Symbol Error Rate (SER)');
title(['SER vs. SNR for ', constellation_type]);
grid on;
```

SIMULATION:

1. For N stages when N=mu=2, at 16dB SNR Symbol Error Rate is 50%.

When N=2mu=4, at 16dB SNR Symbol Error Rate is 25%.

When N=4mu=8 at 16dB SNR Symbol Error Rate is 20%.

When N=5mu=10 at 16dB SNR Symbol Error Rate is 10%

When N=10mu=20 at 16dB SNR Symbol Error Rate is 20%..

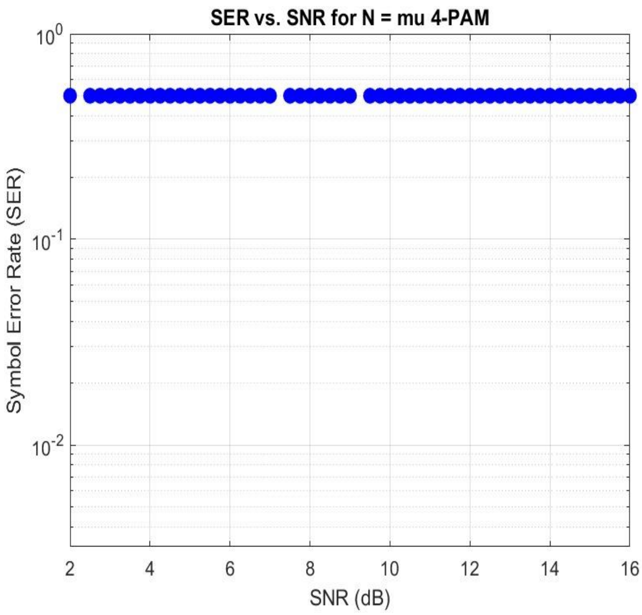
When N=20mu=40 at 16dB SNR Symbol Error Rate is 2.5%.

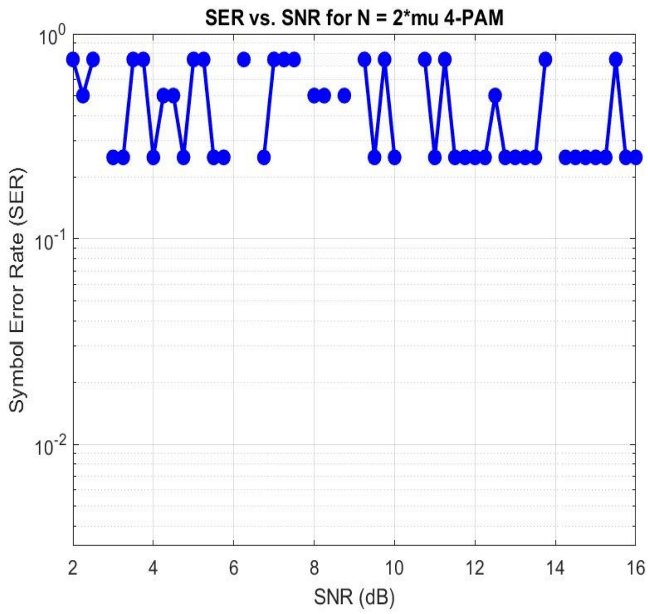
When N=40mu=80 at 16dB SNR Symbol Error Rate is 1.2%.

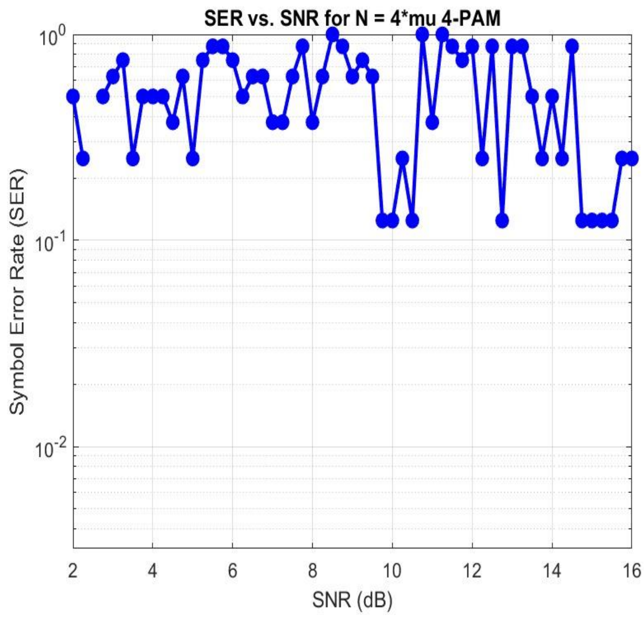
2.

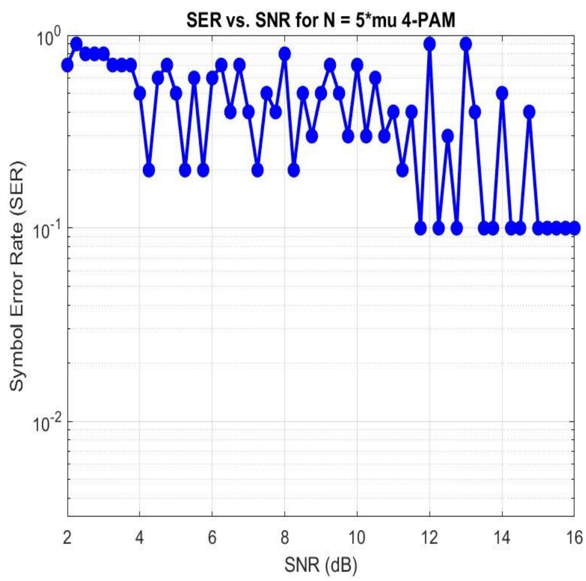
```
Columns 1 through 18

  0.5834    0.5793    0.5708    0.5601    0.5565    0.5670    0.5426    0.5387    0.5285    0.5293    0.5374    0.5274    0.5011    0.4983    0.4975    0.4883    0.4866    0.4611

Columns 19 through 36

  0.4584    0.4447    0.4407    0.4195    0.4069    0.3930    0.3872    0.3845    0.3653    0.3494    0.3356    0.3171    0.2992    0.2810    0.2711    0.2662    0.2558    0.2384

Columns 37 through 54

  0.2070    0.1874    0.1719    0.1691    0.1425    0.1332    0.1220    0.1118    0.1021    0.0819    0.0773    0.0631    0.0582    0.0489    0.0528    0.0320    0.0276    0.0276

Columns 55 through 57

  0.0220    0.0139    0.0127
```

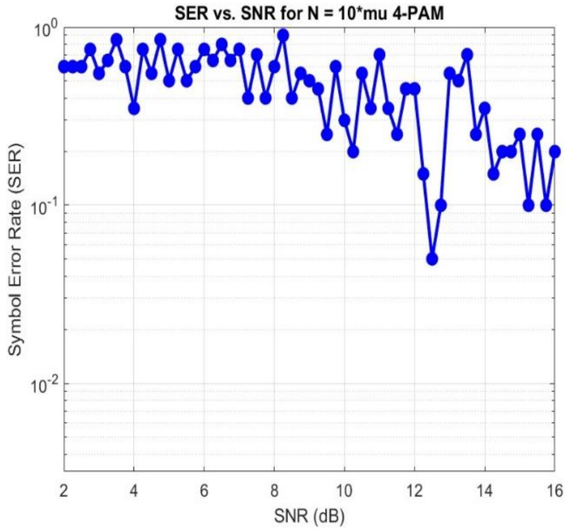
PART-3:

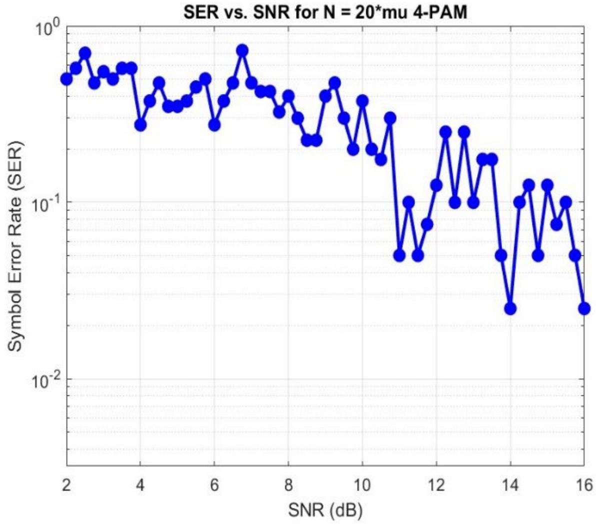


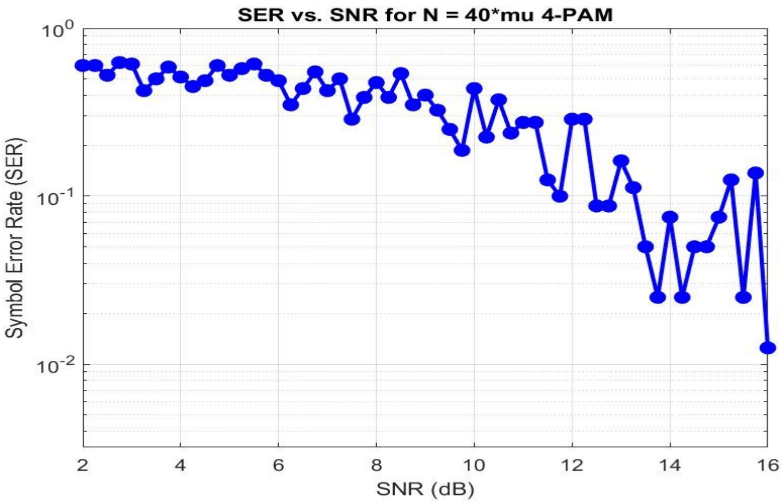




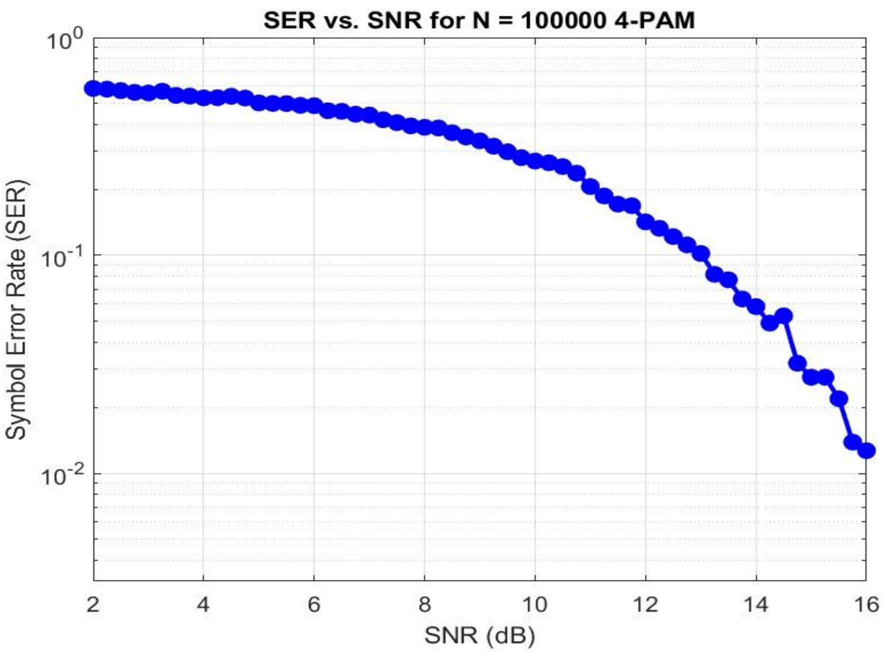



SER vs. SNR for N = 10*mu 4-PAM
Symbol Error Rate (SER)
SNR (dB)


SER vs. SNR for N = 20*mu 4-PAM
Symbol Error Rate (SER)
SNR (dB)


SER vs. SNR for N = 40*mu 4-PAM
Symbol Error Rate (SER)
SNR (dB)

4.


SER vs. SNR for N = 100000 4-PAM
Symbol Error Rate (SER)
SNR (dB)

QAM:

1.

1. For N stages when N=mu=2, at 16dB SNR Symbol Error Rate is 50%.

When N=2mu=4, at 16dB SNR Symbol Error Rate is 25%.

When N=4mu=8 at 16dB SNR Symbol Error Rate is 13%.

When N=5mu=10 at 16dB SNR Symbol Error Rate is 10%

When N=10mu=20 at 16dB SNR Symbol Error Rate is 5%..

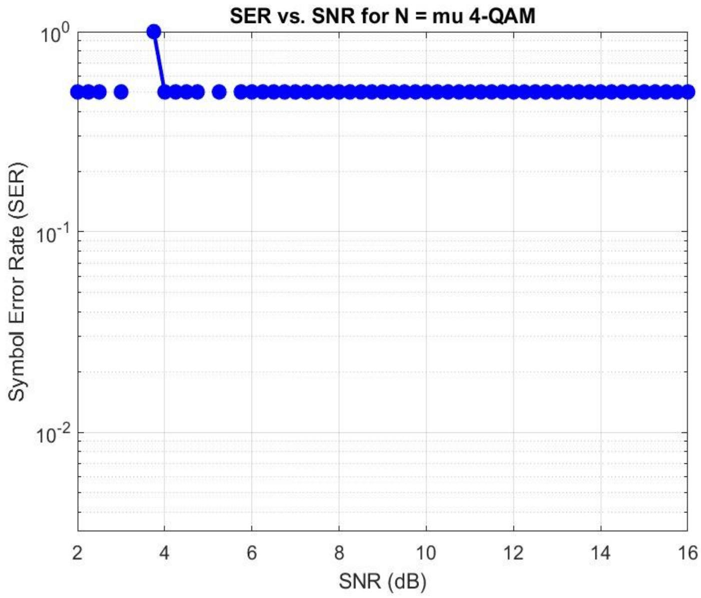
When N=20mu=40 at 16dB SNR Symbol Error Rate is 3%.

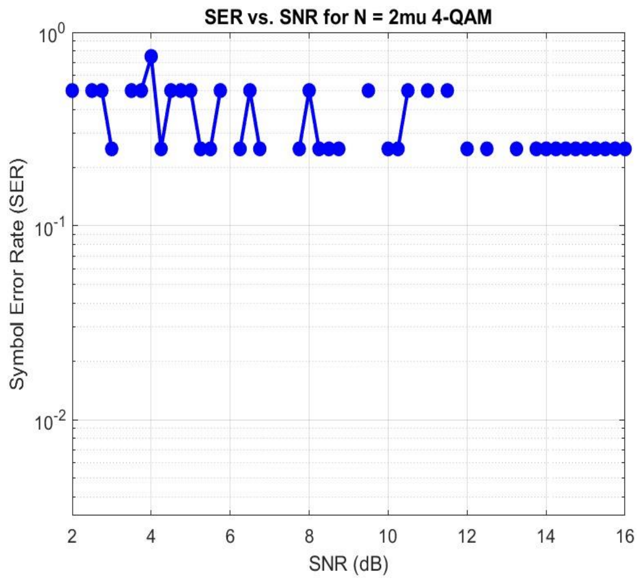
When N=40mu=80 at 16dB SNR Symbol Error Rate is 1.2%.

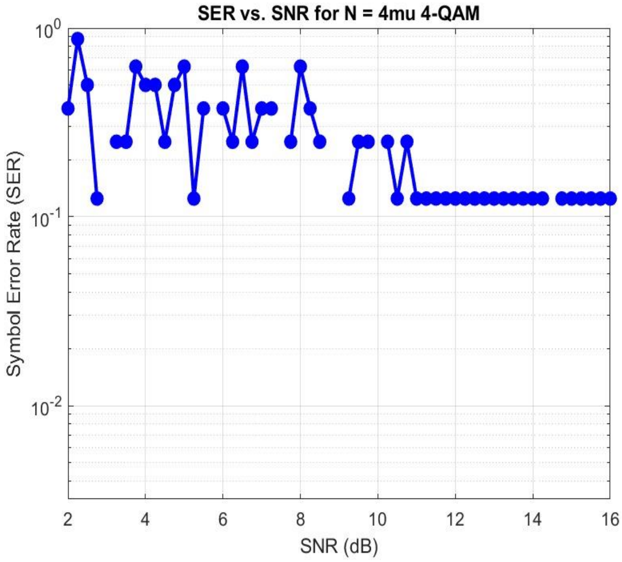
2.

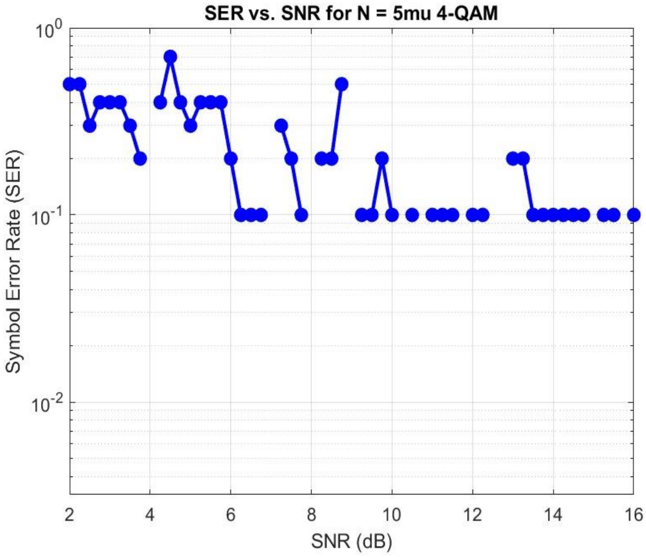
```
Columns 1 through 18

  0.4085    0.4056    0.3940    0.3796    0.3632    0.3629    0.3517    0.3317    0.3226    0.3061    0.2925    0.2780    0.2695    0.2549    0.2392    0.2206    0.2181    0.1971

Columns 19 through 36

  0.1844    0.1804    0.1523    0.1530    0.1339    0.1277    0.1138    0.1195    0.1024    0.0921    0.0896    0.0789    0.0690    0.0716    0.0680    0.0579    0.0510    0.0459

Columns 37 through 54

  0.0422    0.0403    0.0323    0.0333    0.0292    0.0277    0.0232    0.0209    0.0188    0.0182    0.0145    0.0131    0.0112    0.0101    0.0097    0.0079    0.0067    0.0055

Columns 55 through 57

  0.0040    0.0036    0.0031
```

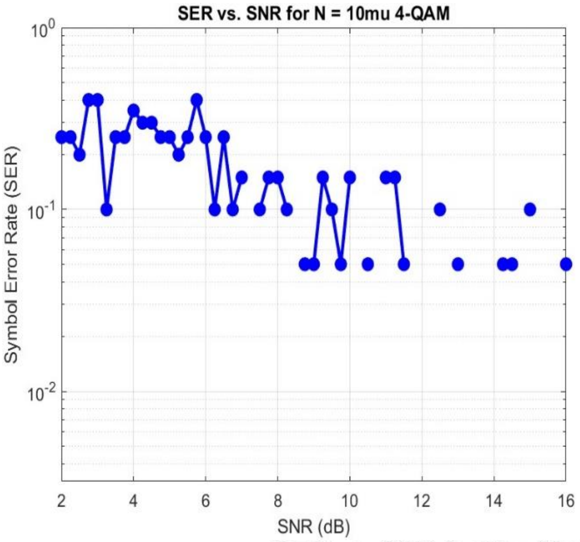
3.

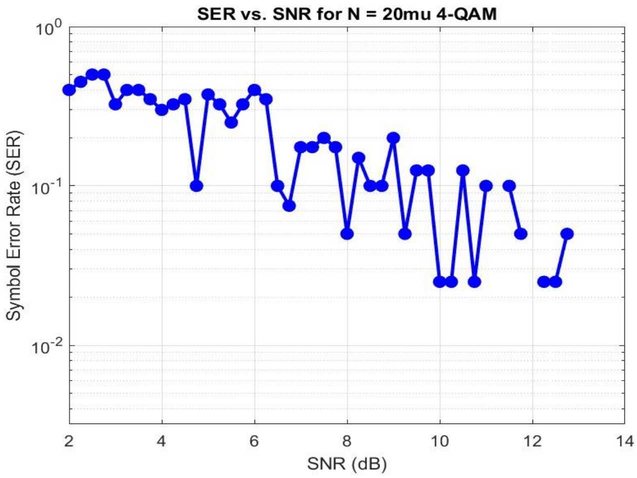


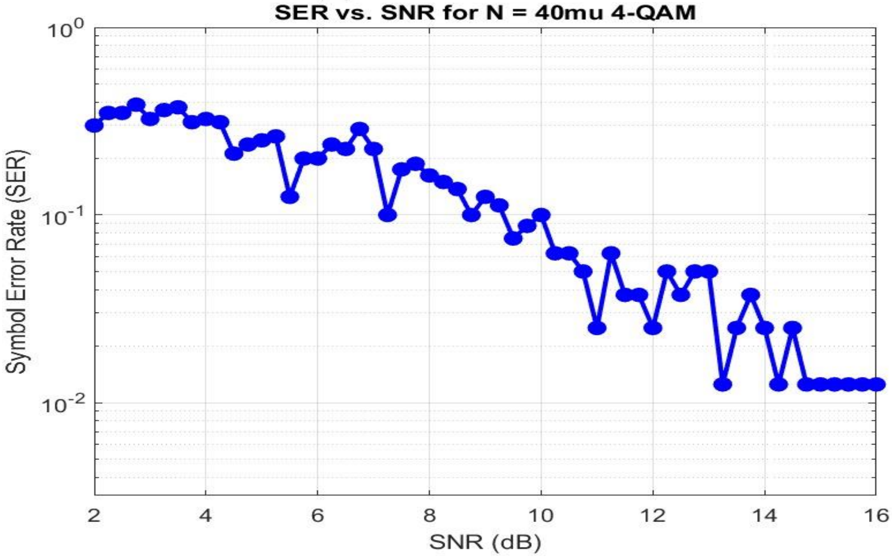



SER vs. SNR for N = 4mu 4-QAM
Symbol Error Rate (SER)
SNR (dB)

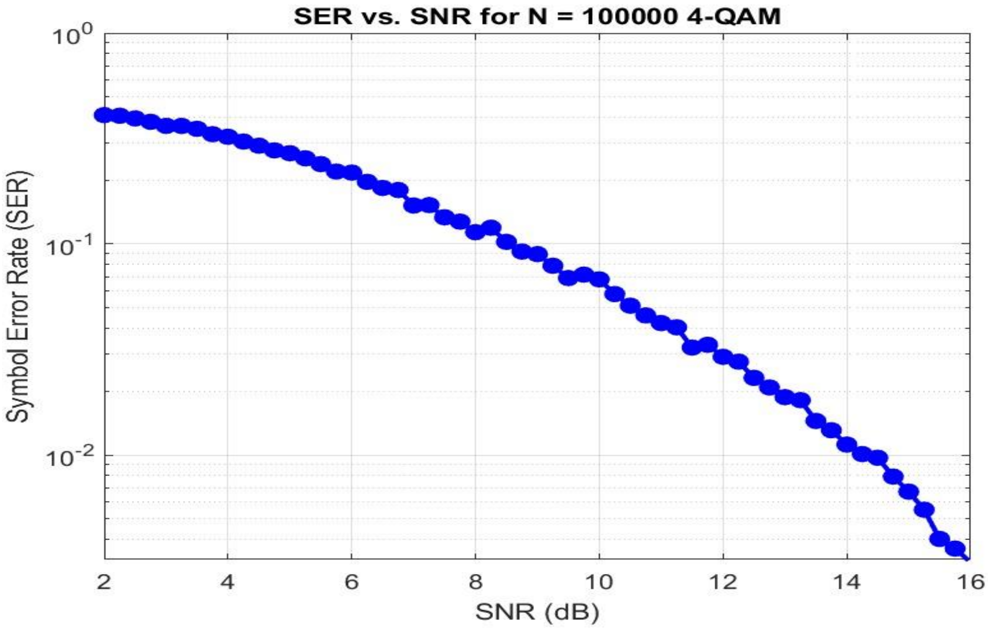

SER vs. SNR for N = 5mu 4-QAM
Symbol Error Rate (SER)
SNR (dB)


SER vs. SNR for N = 10mu 4-QAM
Symbol Error Rate (SER)
SNR (dB)


SER vs. SNR for N = 20mu 4-QAM
Symbol Error Rate (SER)
SNR (dB)


SER vs. SNR for N = 40mu 4-QAM
Symbol Error Rate (SER)
SNR (dB)

4.



CONCLUSION:

- The plot shows a steady decrease in Symbol Error Rate (SER) as the SNR (Signal-to-Noise Ratio) increases. This trend is expected, as higher SNR values mean less noise in the channel, leading to more reliable symbol detection.
- The channel's memory (modeled by convolution with past symbols) introduces intersymbol interference (ISI), making symbol detection challenging, especially at low SNR. The Viterbi decoder attempts to correct for ISI by tracking the most likely sequence, but ISI remains a source of error.
- When noise levels are low (high SNR), there's less chance that noise will distort the transmitted symbols enough to cause errors in detection. This lower noise influence reduces the likelihood of decoding errors, which directly lowers the Symbol Error Rate (SER).
- As SNR increases, the transmitted symbols become more distinct from each other because the noise interference diminishes. This leads to more accurate detection and decoding, with fewer symbols being misinterpreted.
- In essence, as the noise level decreases (with higher SNR), the symbols are received more cleanly, resulting in a lower SER. This is why we see a downward trend in SER with increasing SNR in the plot.
- The SER curve may flatten at high SNR values due to persistent ISI, even with minimal noise. This "error floor" suggests that channel interference limits performance, as ISI introduces errors that noise reduction alone cannot overcome.
- Having more stages allows the decoder to consider a longer sequence for determining the most likely transmitted symbols. This extended observation window help reduce the overall error rate by making the decoding process more robust against individual noise or ISI effects.

- With a larger number of stages, SER estimates become more reliable statistically, as they are based on a larger sample of transmitted symbols.