DC LAB Project

MLSE using Viterbi Decoding

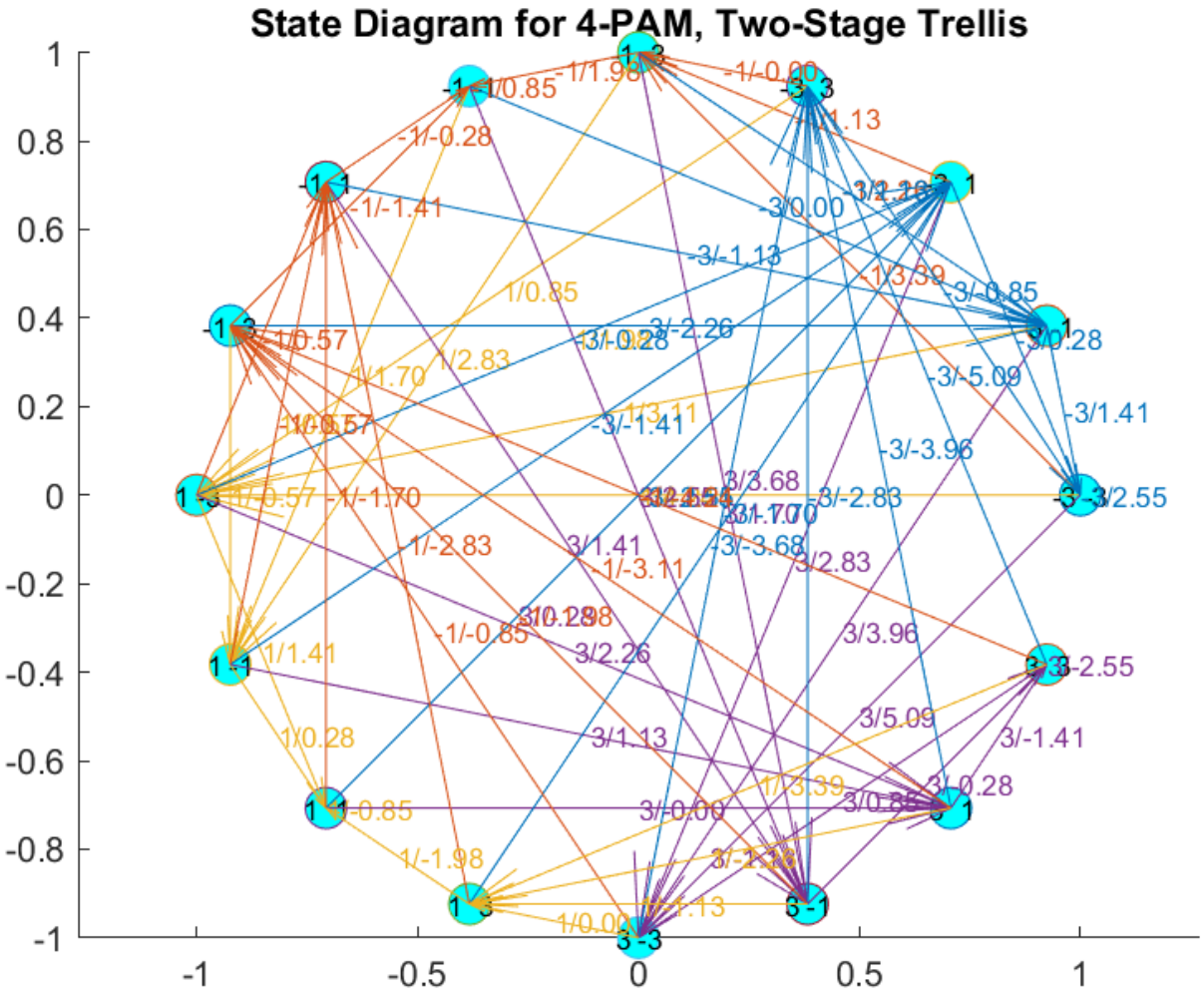
Karan Pattanaik
ROLL-22EC01009

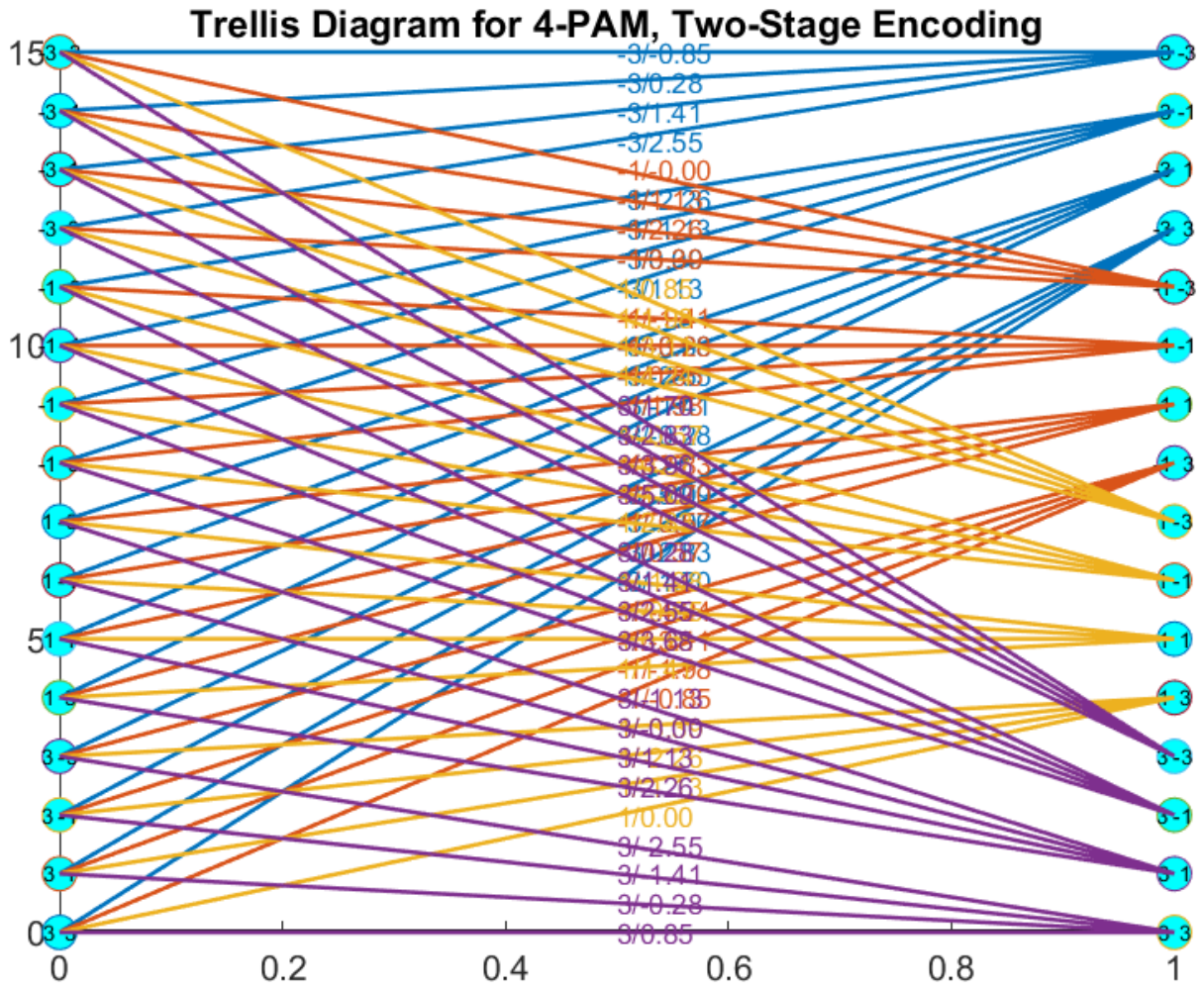
```
%%
% Define parameters for a two-stage trellis diagram with 4-PAM
syms z;
transfer_function = [0.6, -1, 0.8] / sqrt(2); % F(z) coefficients for channel (scaled by sqrt(2))
No_symbols = 1e5;               % Total number of symbols (including termination symbols)
mu = 2;                         % Channel memory

% Parameters for SNR and 4-PAM symbol settings
N = 1 * mu;
symbols = [-3 -1 1 3];          % 4-PAM symbol set
initialstate=[-1 -1];  % symbols for initial state and final termination
M = length(symbols);
Eb = mean(symbols.^2);          % Average energy per symbol for given constellation
SNR_db = 0:2:16;                % SNR values in dB for analysis

% Generate random sequence of 4-PAM symbols
tx_symbols = symbols(randi([1, M], 1, No_symbols));
tx_symbols = [initialstate tx_symbols initialstate];  % Add termination symbols at the start and end

%%
% Define states: two previous symbols form the state
state1 = repmat(symbols, 1, M); % First state (previous symbol)
state2 = repelem(symbols, M); % Second state (current symbol)
statesdiag = [state2; state1]; % State combinations (2 symbols)
states=flip(statesdiag,2);
% Number of states (4 possible symbols at 2 stages)
numStates = M^2;
theta = linspace(0, 2 * pi, numStates + 1);
statePositions = [cos(theta(1:end-1)); sin(theta(1:end-1))]';

% Plot State Diagram
```

```
figure;

hold on;

title('State Diagram for 4-PAM, Two-Stage Trellis');

axis equal;


% Plot nodes (states)

for i = 1:numStates

    plot(statePositions(i, 1), statePositions(i, 2), 'o', 'MarkerSize', 12,
'MarkerFaceColor', 'cyan');

    text(statePositions(i, 1), statePositions(i, 2), num2str([statesdiag(1, i),
statesdiag(2, i)]), 'HorizontalAlignment', 'center', 'FontSize', 8);

end


% Plot transitions

colors = lines(M); % Different colors for each input symbol

for i = 1:numStates

    for j = 1:M

        currentInput = symbols(j);

        nextState = [currentInput; statesdiag(1, i)]; % New state after the input

        nextIndex = find(ismember(statesdiag', nextState', 'rows')); % Find the
new state index



        if ~isempty(nextIndex)

            dx = statePositions(nextIndex, 1) - statePositions(i, 1);

            dy = statePositions(nextIndex, 2) - statePositions(i, 2);

            quiver(statePositions(i, 1), statePositions(i, 2), dx, dy, 0, 'Color',
colors(j, :), 'MaxHeadSize', 0.3);



            % Add transition label (f_value as the result of the transfer
function)

            f_value = transfer_function(1) * symbols(j) + transfer_function(2) *
statesdiag(1, i) + transfer_function(3) * states(2, i);

            label = sprintf('%d/%.2f', currentInput, f_value);

            text(statePositions(i,1) + 0.5*dx, statePositions(i,2) + 0.5*dy,
label, 'FontSize', 8, 'Color', colors(j, :));

        end
```

```
    end
end
hold off;

% Trellis Diagram for a few time steps
numTimeSteps = 2; % Only 2 stages for this trellis diagram
ySpacing = 1;
xSpacing = 1;

% Calculate positions for each state over time (two-stage trellis)
statePositionsTrellis = zeros(numStates, numTimeSteps, 2);
for t = 1:numTimeSteps
    for i = 1:numStates
        statePositionsTrellis(i, t, :) = [(t - 1) * xSpacing, (i - 1) * ySpacing];
    end
end

figure;
hold on;
title('Trellis Diagram for 4-PAM, Two-Stage Encoding');

% Plot each state for each time step
for t = 1:numTimeSteps
    for i = 1:numStates
        plot(statePositionsTrellis(i, t, 1), statePositionsTrellis(i, t, 2), 'o',
'MarkerSize', 10, 'MarkerFaceColor', 'cyan');
        text(statePositionsTrellis(i, t, 1), statePositionsTrellis(i, t, 2),
num2str([statesdiag(1, i), states(2, i)]), 'HorizontalAlignment', 'center',
'FontSize', 6);
    end
end

% Draw transitions between states over time
for t = 1:numTimeSteps-1
    for i = 1:numStates
```

```
        for j = 1:M
            currentState = statesdiag(:, i);
            currentInput = symbols(j);
            nextState = [currentInput; currentState(1)]; % Transition to next
state
            nextIndex = find(ismember(statesdiag', nextState', 'rows')); % Find
the new state index


            if ~isempty(nextIndex)
                plot([statePositionsTrellis(i, t, 1),
statePositionsTrellis(nextIndex, t + 1, 1)], ...
                     [statePositionsTrellis(i, t, 2),
statePositionsTrellis(nextIndex, t + 1, 2)], ...
                     'Color', colors(j, :), 'LineWidth', 1.2);


                % Label transition at midpoint
                mid_x = (statePositionsTrellis(i, t, 1) +
statePositionsTrellis(nextIndex, t + 1, 1)) / 2;
                mid_y = (statePositionsTrellis(i, t, 2) +
statePositionsTrellis(nextIndex, t + 1, 2)) / 2;
                f_value = transfer_function(1) * symbols(j) + transfer_function(2)
* statesdiag(1, i) + transfer_function(3) * states(2, i);
                label = sprintf('%d/%.2f', currentInput, f_value);
                text(mid_x, mid_y, label, 'FontSize', 8, 'Color', colors(j, :));
            end
        end
    end
end
hold off;
```


State Diagram for 4-PAM, Two-Stage Trellis
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.5
0
0.5
1




```
%%

% Pass the symbols through the ISI channel modeled by transfer_function
channel_output = zeros(1, length(tx_symbols) - 2);
for i = 1:length(tx_symbols) - 2
    % Sum of symbols weighted by transfer function coefficients to simulate ISI
    channel_output(i) = sum(tx_symbols(i + 2:-1:i) .* transfer_function);
end

% Alternative way to calculate ISI channel output using convolution
c1 = conv(tx_symbols, transfer_function, "valid");
No_symbols = No_symbols + 2;

%% Define state-space and mapping for Viterbi decoding
```

```
% State-space definition: all combinations of 4-PAM symbols for mu = 2
states = combinations(symbols, symbols).Variables; % Generates M^mu combinations

% Create array `arr` to map each pair of symbols to a unique index
arr = zeros(symbols(end) - symbols(1) + 1, symbols(end) - symbols(1) + 1);
counter = 1;
for stage = symbols
    for j = symbols
        % Map each (stage, j) pair to an index in `arr`
        arr(stage + 1 - symbols(1), j + 1 - symbols(1)) = counter;
        counter = counter + 1;
    end
end

% Matrix to store valid inputs leading from one state to another
input_from_states = zeros(length(states), length(states));
for stage = 1:length(states)
    for j = 1:length(states)
        % Ensure transitions are valid based on symbol alignment
        if (mod(ceil(stage / length(symbols)), length(symbols)) == mod(j,
length(symbols)))
            input_from_states(stage, j) = symbols(ceil(j / length(symbols)));
        end
    end
end

% Calculate output of the ISI channel for each state transition
outputs = zeros(length(states), length(states));
for stage = 1:length(outputs)
    for j = 1:length(outputs)
        l1 = input_from_states(stage, j); % Input symbol for state transition
        if l1 ~= 0
            % Compute the channel output for this state transition
```

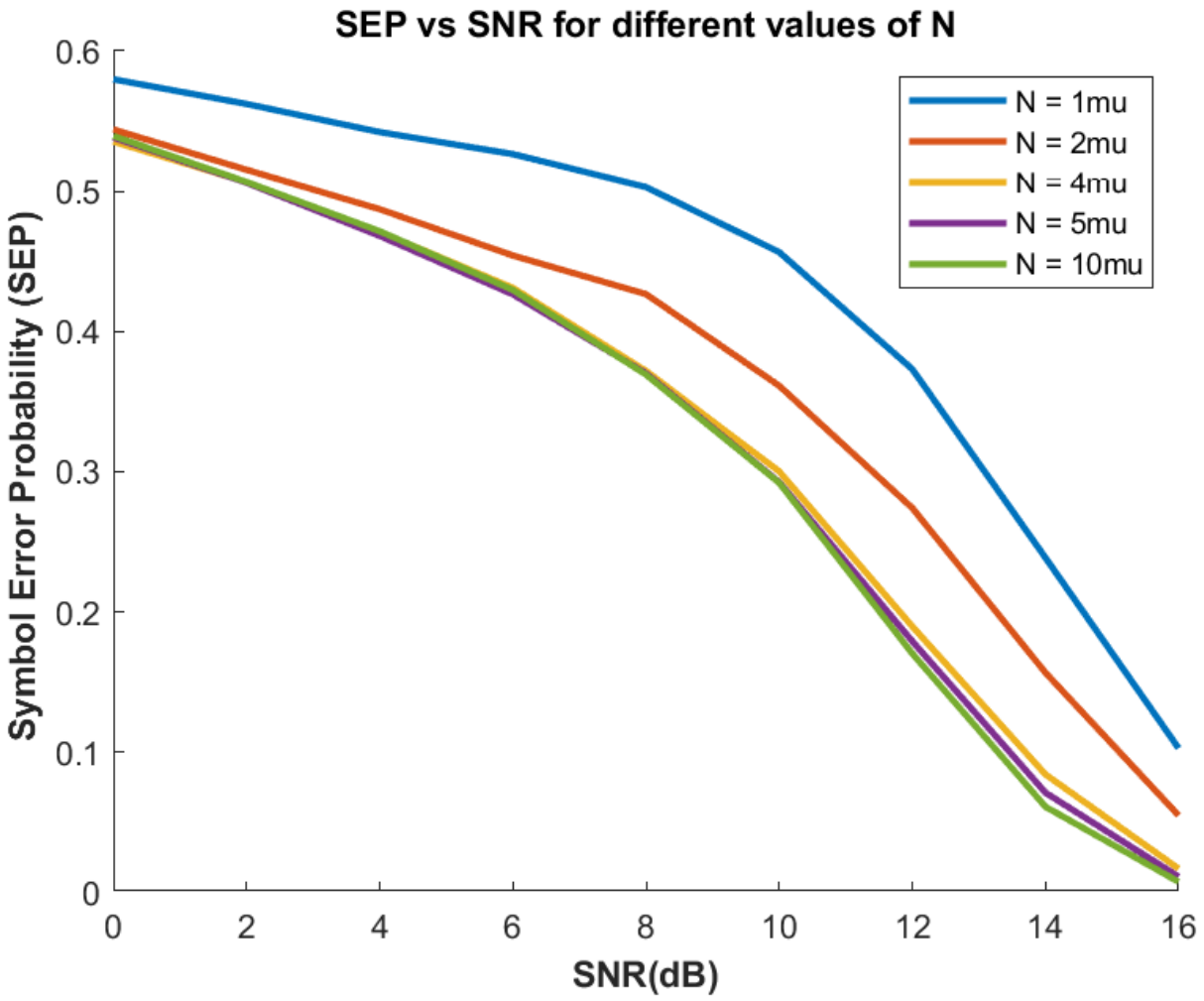
```
            outputs(stage, j) = sum([l1, states(stage, :)] .* transfer_function);
        end
    end
end

%% Viterbi Algorithm
figure
% Matrix for storing minimum weights between states
matrix = inf(length(states), length(states));
weight_of_survivor = zeros(1, length(states));  % Store minimum path weights for each state
survivor_paths = zeros(length(states), No_symbols + 1);  % Store survivor paths for each state

% Loop over different SNR values for Symbol Error Probability (SEP) computation
SNR = 10.^(SNR_db / 10);
N0 = Eb ./ SNR;
SEP = zeros(1, length(N0));
N_array = mu * [1, 2, 4, 5, 10];  % Different backtrack depths for Viterbi

for N = N_array
    for a = 1:length(N0)
        % Generate noise for the received signal
        noise = sqrt(N0(a) / 2) * randn(1, No_symbols);
        rx_signal = channel_output + noise;  % Noisy received signal
        possible_states = [];
        weight_of_survivor = zeros(1, length(states));
        survivor_paths = zeros(length(states), No_symbols + 1);
        matrix = inf(length(states), length(states));

        % Initialize decoded symbols and counter
        decoded_symbols = zeros(1, No_symbols);
        c = 1;

        % Main loop for Viterbi algorithm
```

```
        for stage = 1:No_symbols + 1
            % Determine possible states for each stage
            if stage == 1
                possible_states = 1;
            elseif stage == 2
                possible_states = zeros(size(symbols));
                for p = 1:length(symbols)
                    % Find all states that the initial state maps to with each
input symbol
                    possible_states(p) = next_state_index(1, symbols(p), states,
arr, symbols);
                end
            elseif stage == No_symbols
                possible_states = [];
                for l2 = 1:length(input_from_states(:, 1))
                    if input_from_states(l2, 1) ~= 0
                        possible_states = [l2, possible_states]; % Finding all
states that map to initial state
                    end
                end
            elseif stage == No_symbols + 1
                possible_states = 1;  % Termination to the initial state
            else
                possible_states = 1:length(states);  % All states for middle
stages
            end

            % Initialize weights for this stage
            weight_of_survivor = inf(1, length(states));
            for j = possible_states
                % Find the minimum weight path to the current state
                [val, previous_index] = min(matrix(:, j));
                weight_of_survivor(j) = val;
                survivor_paths(j, stage) = previous_index;
            end
```

```
% Decoding process for the symbol sequence
[~, index] = min(weight_of_survivor);
a1 = [];
if stage >= N + mu && mod(stage, mu) == 0
    % Backtrack for decoded symbols based on its depth
    for iteration = stage:-1:stage - (N + mu) + 2
        index = survivor_paths(index, iteration);
        a1 = [index, a1];
    end
    for n = 2:mu + 1 %Decode mu bits
        decoded_symbols(c) = input_from_states(a1(n - 1), a1(n));
        c = c + 1;
    end
end

% Final backtrack for remaining undecoded symbols
if stage == No_symbols + 1
    index = 1;
    a1 = [index];
    for iteration = stage:-1:c + 1
        index = survivor_paths(index, iteration);
        a1 = [index, a1];
    end
    for n = 2:length(a1)
        decoded_symbols(c) = input_from_states(a1(n - 1), a1(n));
        c = c + 1;
    end
end

 if stage==No_symbols-1
     input_symbol=initialstate(1);
 elseif stage == No_symbols
     input_symbol=initialstate(2);
```

```
             else
                 input_symbol=symbols;
             end



            % Update weights in matrix for the next state transitions
            if stage == 1
                weight_of_survivor(1) = 0;
            end
            if stage ~= No_symbols + 1
                matrix = inf(length(states), length(states));
                for j = possible_states
                    for input = 1:length(input_symbol)
                        k1 = next_state_index(j, input_symbol(input), states, arr,
symbols); % Next state index
                        k2 = outputs(j, k1);  % Output for current to next state
transition
                        matrix(j, k1) = (rx_signal(stage) - k2)^2;
                    end
                end
                matrix = matrix + weight_of_survivor';
            end
        end


        % Calculate Symbol Error Probability
        SEP(a) = sum(tx_symbols(3:end - 2) ~= decoded_symbols(1:end - 2)) /
(No_symbols);
    end
    hold on
    semilogy(SNR_db, SEP, 'LineWidth', 2)
end


legend(('N = ') + string(N_array / mu) + ('mu'))
xlabel("SNR(dB)", "FontWeight", "bold");
ylabel("Symbol Error Probability (SEP) ", "FontWeight", "bold");
```

```
title('SEP vs SNR for different values of N')
```




```
%% Helper Functions


% Calculate the next state index based on current state and input

function cal = next_state_index(current_state_index, input, states, arr, symbols)

    cal = arr(input + 1 - symbols(1), states(current_state_index, 1) + 1 -
symbols(1));

end
```