INTRODUCTION TO COMMUNICATION SYSTEMS

(CT216)

Convolution Coding

Simulation Results (Matlab Code)

Lab Group: 3, Project Group : 3

Prof. Yash Vasavada
Mentor TA: Aparna Kumari

Group members :

202201227 - Nishank Kansara
202201228 - Jaimin Prajapati
202201233 - Harsh Baraiya
202201234 - Anshu Dhankecha
202201241 - Divyesh Ramani
202201242 - Dev Davda
202201250 - Parth Prajapati
202201251 - Mehul Vagh
202201256 - Ayush Pandita
202201258 - Nishant Italiya

Honour Code

We declare that :

- The work that we are presenting is our own work.
- We have not copied the work(the code,the results,etc.) that someone else has done.
- Concepts, understanding and insights we will be describing are own.
- We make this pledge truthfully. We know that violation of this solemn pledge can carry grave consequences.

❖ Matlab Code :

- We wrote code for convolution coding for three cases as mention in pdf.
- First of all we make function for encoding , for hard decision decoding and for soft decision decoding.
- These three functions are below the code and then wrote code for three cases and plot graph of BER vs EbNo(dB) .
- We plot three graphs those are for hard decision Viterbi decoding , soft decision Viterbi decoding and for comparison hard and soft decision Viterbi decoding.

- This for case {r = 1/2, Kc = 3}

```
% For {r = 1/2, Kc = 3}
Kc = 3;
R = 1/2;
k=1;
n=2;
m = Kc-1; % SR length

g = [1 0 1; % 5
     1 1 1];% 7
```

```matlab
EbNodB = 0:0.5:10;

practical_error_hard1 = zeros(1,length(EbNodB));
practical_error_soft1 = zeros(1,length(EbNodB));
theoretical_error = zeros(1,length(EbNodB));

idx =1;

% number of simulations
Nsim = 10000;

for j=EbNodB
    EbNo = 10^(j/10);
    sigma = sqrt (1/ (2*R*EbNo));
    BER = 0.5 * erfc(sqrt(EbNo));
    Nerrs_hard = 0;
    Nerrs_soft = 0;

    for i = 1 : Nsim
        msg = randi ([0 1],1,15); %generate random message

        encoded_msg = encoding(msg,g,Kc); % encode the message


        % BPSK bit to symbol conversion modulation
        Modulation = 1 - 2 * encoded_msg;

        % AWGN channel
        received_codeword = Modulation + sigma * randn (1,n*(length(msg)+m));

        % decode the message using soft dicision viterbi algorithm
        decoded_msg = decoding_soft(received_codeword,g,n,Kc);
        decoded_msg = decoded_msg(1:length(msg));

        Nerrs_soft = Nerrs_soft + sum(decoded_msg ~= msg);

        received_codeword = (received_codeword < 0); %threshold at zero

        % decode the message using hard dicision viterbi algorithm
        decoded_msg = decoding_hard(received_codeword,g,n,Kc);
        decoded_msg = decoded_msg(1:length(msg));

        Nerrs_hard = Nerrs_hard + sum(decoded_msg ~= msg); %calculate error
    end
```

```
    practical_error_hard1(1,idx) = (Nerrs_hard/(Nsim*length(msg)));
    practical_error_soft1(1,idx) = (Nerrs_soft/(Nsim*length(msg)));
    theoretical_error(1,idx) = BER;
    idx = idx+1;

end
```

- This for case {r = 1/3, Kc = 4}

```
%For {r = 1/3, Kc = 4}
Kc = 4;
R = 1/3;
k=1;
n=3;
m = Kc-1; % SR length

g = [1 0 1 1; % 13
     1 1 0 1; % 15
     1 1 1 1]; %17

practical_error_hard2 = zeros(1,length(EbNodB));
practical_error_soft2 = zeros(1,length(EbNodB));


idx =1;

for j=EbNodB
    EbNo = 10^(j/10);
    sigma = sqrt (1/ (2*R*EbNo));

    Nerrs_hard = 0;
    Nerrs_soft = 0;

    for i = 1 : Nsim
        msg = randi ([0 1],1,15); %generate random message

        encoded_msg = encoding(msg,g,Kc); % encode the message


        % BPSK bit to symbol conversion modulation
        Modulation = 1 - 2 * encoded_msg;

        % AWGN channel
        received_codeword = Modulation + sigma * randn (1,n*(length(msg)+m));
```

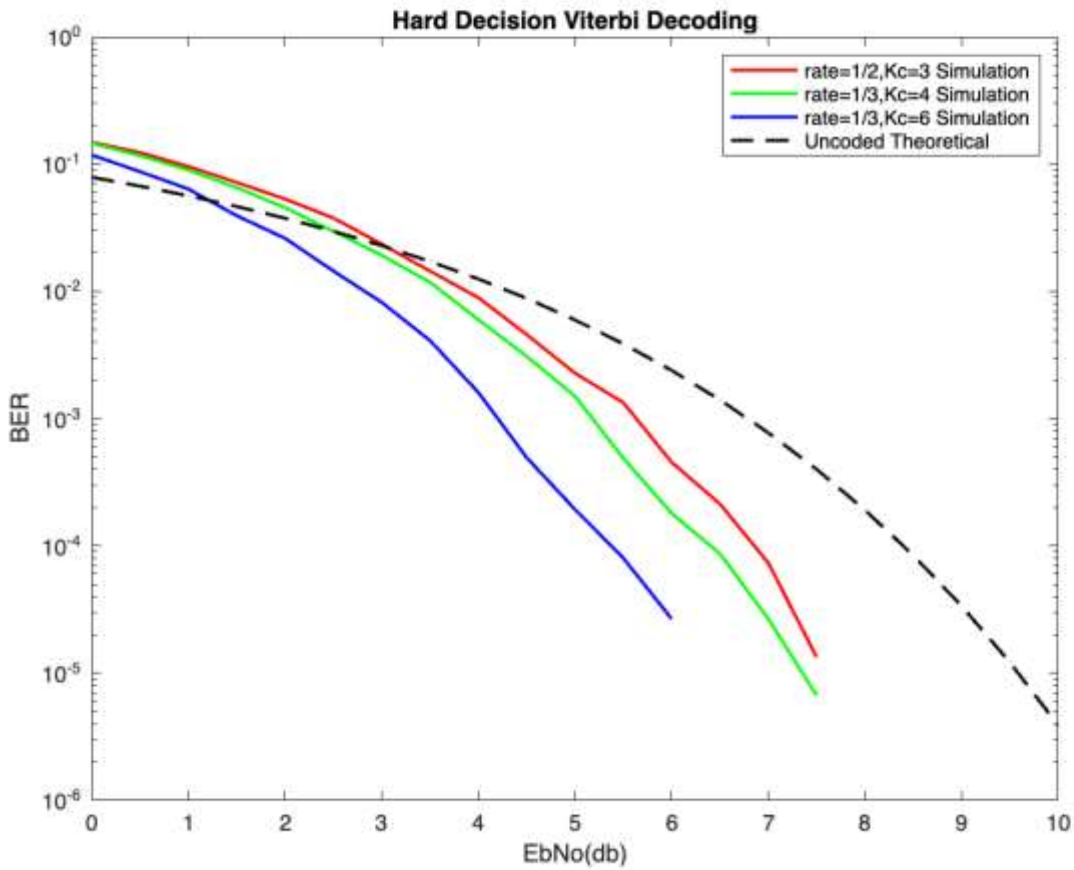
```
        % decode the message using soft dicision viterbi algorithm
        decoded_msg = decoding_soft(received_codeword,g,n,Kc);
        decoded_msg = decoded_msg(1:length(msg));

        Nerrs_soft = Nerrs_soft + sum(decoded_msg ~= msg);

        received_codeword = (received_codeword < 0); %threshold at zero

        % decode the message using hard dicision viterbi algorithm
        decoded_msg = decoding_hard(received_codeword,g,n,Kc);
        decoded_msg = decoded_msg(1:length(msg));

        Nerrs_hard = Nerrs_hard + sum(decoded_msg ~= msg); %calculate error
    end

    practical_error_hard2(1,idx) = (Nerrs_hard/(Nsim*length(msg)));
    practical_error_soft2(1,idx) = (Nerrs_soft/(Nsim*length(msg)));
    idx = idx+1;

end
```

- This for case {r = 1/3, Kc = 6}

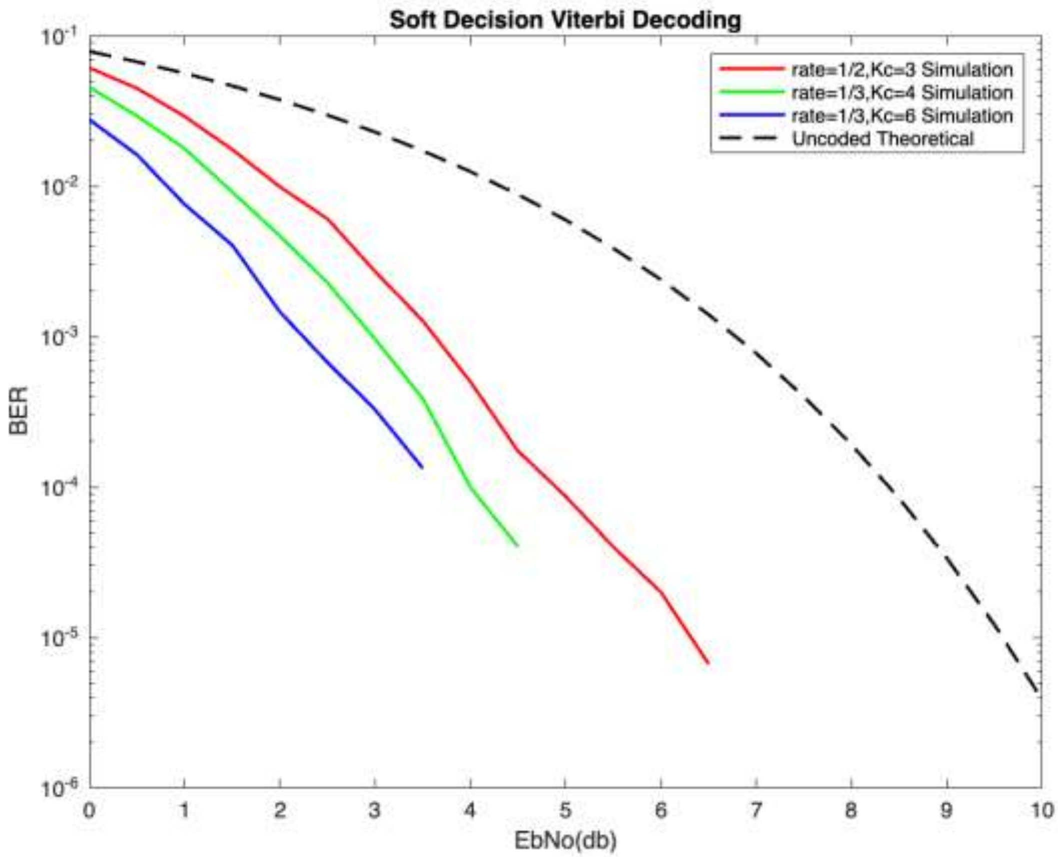
```
% For {r = 1/3, Kc = 6}

Kc = 6;
R = 1/3;
k=1;
n=3;
m = Kc-1; % SR length

g = [1 0 0 1 1 1; % 47
     1 0 1 0 1 1; % 53
     1 1 1 1 0 1]; % 75

practical_error_hard3 = zeros(1,length(EbNodB));
practical_error_soft3 = zeros(1,length(EbNodB));

idx =1;

for j=EbNodB
    EbNo = 10^(j/10);
    sigma = sqrt (1/ (2*R*EbNo));

    Nerrs_hard = 0;
```

```
    Nerrs_soft = 0;

    for i = 1 : Nsim
        msg = randi ([0 1],1,15); %generate random message

        encoded_msg = encoding(msg,g,Kc); % encode the message


        % BPSK bit to symbol conversion modulation
        Modulation = 1 - 2 * encoded_msg;

        % AWGN channel
        received_codeword = Modulation + sigma * randn (1,n*(length(msg)+m));

        % decode the message using soft dicision viterbi algorithm
        decoded_msg = decoding_soft(received_codeword,g,n,Kc);
        decoded_msg = decoded_msg(1:length(msg));

        Nerrs_soft = Nerrs_soft + sum(decoded_msg ~= msg);

        received_codeword = (received_codeword < 0); %threshold at zero

        % decode the message using hard dicision viterbi algorithm
        decoded_msg = decoding_hard(received_codeword,g,n,Kc);
        decoded_msg = decoded_msg(1:length(msg));

        Nerrs_hard = Nerrs_hard + sum(decoded_msg ~= msg); %calculate error
    end

    practical_error_hard3(1,idx) = (Nerrs_hard/(Nsim*length(msg)));
    practical_error_soft3(1,idx) = (Nerrs_soft/(Nsim*length(msg)));

    idx = idx+1;

end
%Hard decision viterbi decoding
semilogy(EbNodB,practical_error_hard1,'LineWidth',1.5,Color="#FF0000");
hold on;
semilogy(EbNodB,practical_error_hard2,'LineWidth',1.5,Color="#00FF00");
semilogy(EbNodB,practical_error_hard3,'LineWidth',1.5,Color="#0000FF");
semilogy(EbNodB,theoretical_error,'--','LineWidth',1.5,Color="#000000");
legend('rate=1/2,Kc=3 Simulation','rate=1/3,Kc=4 Simulation', ...
    'rate=1/3,Kc=6 Simulation','Uncoded Theoretical','Location', 'northeast');
title('Hard Decision Viterbi Decoding');
xlim([0,10]);
```

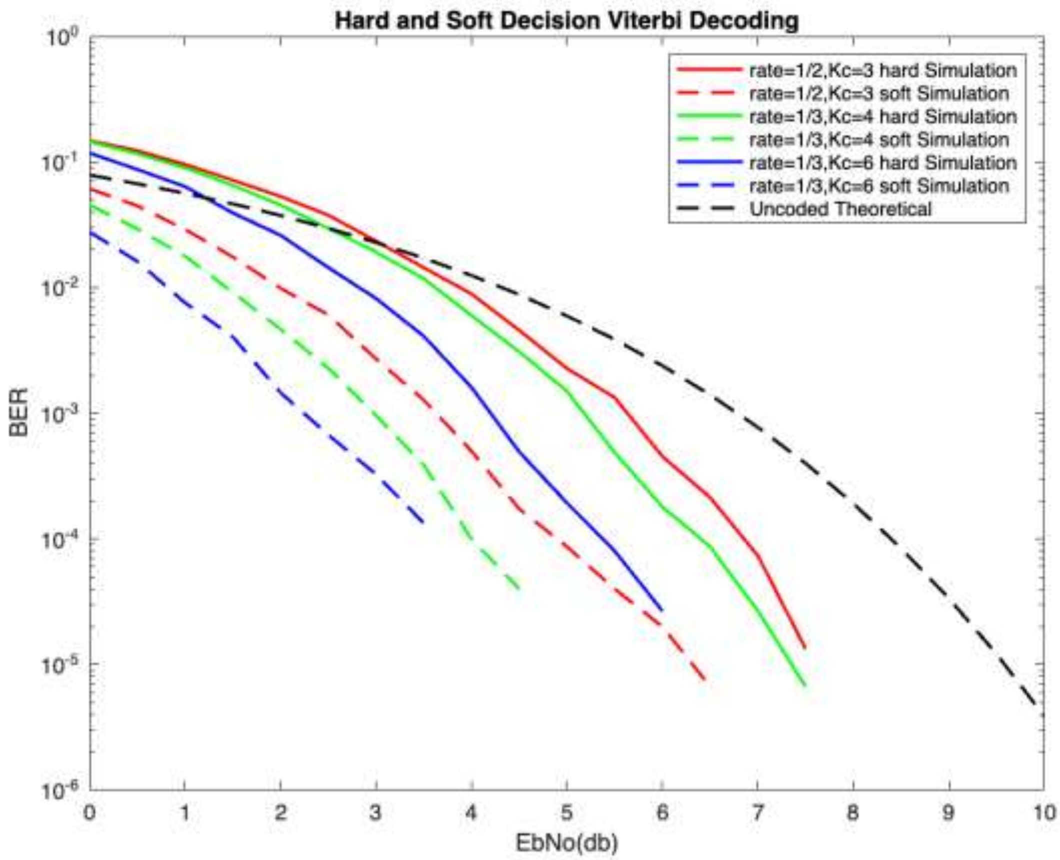
```
xlabel('EbNo(db)');
ylabel('BER');
hold off;
```




```
%Soft decision viterbi decoding
semilogy(EbNodB,practical_error_soft1,'LineWidth',1.5,Color="#FF0000");
hold on;
semilogy(EbNodB,practical_error_soft2,'LineWidth',1.5,Color="#00FF00");
semilogy(EbNodB,practical_error_soft3,'LineWidth',1.5,Color="#0000FF");
semilogy(EbNodB,theoretical_error,'--','LineWidth',1.5,Color="#000000");
title('Soft Decision Viterbi Decoding');
xlim([0,10]);
xlabel('EbNo(db)');
ylabel('BER');
legend('rate=1/2,Kc=3 Simulation','rate=1/3,Kc=4 Simulation', ...
    'rate=1/3,Kc=6 Simulation','Uncoded Theoretical','Location', 'northeast');
hold off;
```




```
%hard -soft decision viterbi decoding
semilogy(EbNodB,practical_error_hard1,'LineWidth',1.5,Color="#FF0000");
hold on;
semilogy(EbNodB,practical_error_soft1,'--','LineWidth',1.5,Color="#FF0000");
semilogy(EbNodB,practical_error_hard2,'LineWidth',1.5,Color="#00FF00");
semilogy(EbNodB,practical_error_soft2,'--','LineWidth',1.5,Color="#00FF00");
semilogy(EbNodB,practical_error_hard3,'LineWidth',1.5,Color="#0000FF");
semilogy(EbNodB,practical_error_soft3,'--','LineWidth',1.5,Color="#0000FF");
semilogy(EbNodB,theoretical_error,'--','LineWidth',1.5,Color="#000000");
title('Hard and Soft Decision Viterbi Decoding');
xlim([0,10]);
xlabel('EbNo(db)');
ylabel('BER');
legend('rate=1/2,Kc=3 hard Simulation','rate=1/2,Kc=3 soft Simulation', ...
    'rate=1/3,Kc=4 hard Simulation','rate=1/3,Kc=4 soft Simulation', ...
    'rate=1/3,Kc=6 hard Simulation','rate=1/3,Kc=6 soft Simulation','Uncoded
Theoretical', ...
    'Location', 'northeast');
hold off;
```




```
function encoded = encoding(msg,g,Kc)

    % temp for flushing SR, so we need extra Kc-1 zeros
    temp = zeros(1, Kc - 1);

    % We initially added extra Kc-1 zeros into msg sequence for flushing SR
    msg = [msg, temp];

    % SR with initially all zeros
    shiftreg = zeros(1, Kc - 1);

    encoded = [];

    for i = msg
        %combine input and shift register
        m = [i,shiftreg];
        m_tran = transpose(m);

        % Multiply with generator matrix
        ans = mod(transpose(g*m_tran),2);
```

```
        encoded = [encoded,ans];

        shiftreg(2:end) = shiftreg(1:end-1);
        shiftreg(1)=i;
    end

end


function decoded_msg = decoding_hard(received,g,n,Kc)
    m = Kc-1; % SR length

    % there will be 2^m states so generate 0 to 2^m-1 decimal numbers
    d = (0:2^m-1);
    % Convert these decimal numbers into binary array
    b = dec2bin(d)-'0';

    % Generate state output table (first n columns represent output when
    % input is 0 and second n columns from n+1 to 2*n is represent output
    % when input is 1)
    state_output = zeros(2^m,2*n);

    for state = 1:2^m

       tmp = b(state,:);

       %for input 0
       %combine input and shift register
       msg = [0,tmp];
       msg_tran = transpose(msg);

       % Multiply with generator matrix
       ans = mod(transpose(g*msg_tran),2);

       state_output(state,1:n)=ans;

       %for input 1
       %combine input and shift register
       msg = [1,tmp];
       msg_tran = transpose(msg);

       % Multiply with generator matrix
       ans = mod(transpose(g*msg_tran),2);
```

```
        state_output(state,n+1:2*n)=ans;

    end

    % generate next state table (first m columns represent next state when
    % input is 0 and second m columns from m+1 to 2*m is represent next
    % state when input is 1)

    NextState = zeros(2^m,2*m);

    for state=0:2^m-1
        b = dec2bin(state,m);

        b(end) = '';
        b = b-'0';
        tmp = [0,b,1,b];
        NextState(state+1,:) = tmp;
    end

    % Now generate path matric with minimum hamming distance

    trellisNodes = length(received)/n+1;

    trellis = repmat(500,2^m,trellisNodes);

    trellis(1,1)=0; % starting from 0

    for i=1:trellisNodes-1
        tmp = received(n*i-(n-1) : i*n); % n bits from received msg
        for state = 1:2^m
            t0 = state_output(state,1:n); % output when input is 0
            t1 = state_output(state,n+1:2*n); % output when input is 1

            % Hamming distance of n bits of received msg with output when input
            % is 0(H0) and 1(H1)
            H0=sum(tmp ~= t0);
            H1=sum(tmp ~= t1);

            next_st0 = NextState(state,1:m); %nextstate whene input is 0
            next_st1 = NextState(state,m+1:2*m); %nextstate whene input is 1

            %convert nextstate into number
            next_st0 = bin2dec(char(next_st0 + '0'));
            next_st1 = bin2dec(char(next_st1 + '0'));
```

```
            % updating trellis node with mininum hamming distance
            trellis(next_st0+1,i+1) = min(trellis(next_st0+1,i+1), 
trellis(state,i)+H0);
            trellis(next_st1+1,i+1) = min(trellis(next_st1+1,i+1), 
trellis(state,i)+H1);
        end
    end
    
    curr_state = zeros(1,m);
    decoded_msg = [];
    
    % for decoding msg traverse trellis from trellisNodes-1
    for i=trellisNodes-1:-1:1
        input = curr_state(1);
        
        % store input
        decoded_msg = [input decoded_msg];
        
        flag = 1;
        
        % Traverse through all state with according to input
        for state = 1:2^m
           if(flag == 1 & NextState(state,m*input+1:m*input+m) == curr_state)
               prev1 = dec2bin(state-1,m) - '0';
               prev1_dec = state;
               flag = 0;
           elseif(NextState(state,m*input+1:m*input+m) == curr_state)
               prev2 = dec2bin(state-1,m) - '0';
               prev2_dec = state;
           end
        end
        
        % output of prev1 and prev2 according to input
        output1 = state_output(prev1_dec,n*input+1:n*input+n);
        output2 = state_output(prev2_dec,n*input+1:n*input+n);
        
        % n bits from received msg
        tmp = received(n*i-(n-1):n*i);
        
        % Hamming distance of outputs and received n bits msg
        H1= sum(output1~=tmp);
        H2 = sum(output2~=tmp);
        
        % find minimun path(backtracing)
        if(trellis(prev1_dec,i)+H1 < trellis(prev2_dec,i)+H2)
```

```
            curr_state = prev1 ;
        elseif(trellis(prev1_dec,i)+H1 > trellis(prev2_dec,i)+H2)
            curr_state = prev2;
        else
            if(H1<H2)
                curr_state = prev1;
            else
                curr_state = prev2;
            end
        end

    end

end

function decoded_msg = decoding_soft(received,g,n,Kc)
    m = Kc-1; % SR length

    %there will be 2^m states

    d = (0:2^m-1);

    b = dec2bin(d)-'0';

    % Generate state output table (first n columns represent output when
    % input is 0 and second n columns from n+1 to 2*n is represent output
    % when input is 1)
    state_output = zeros(2^m,2*n);

    for state = 1:2^m

       tmp = b(state,:);

       %for input 0
       %combine input and shift register
       msg = [0,tmp];
       msg_tran = transpose(msg);

       % Multiply with generator matrix
       ans = mod(transpose(g*msg_tran),2);

       state_output(state,1:n)=ans;

       %for input 1
       %combine input and shift register
```

```
        msg = [1,tmp];
        msg_tran = transpose(msg);

        % Multiply with generator matrix
        ans = mod(transpose(g*msg_tran),2);

        state_output(state,n+1:2*n)=ans;

    end

    % Change state output passing through BPSK
    state_output = 1 - 2.*state_output;

    % generate next state table (first m columns represent next state when
    % input is 0 and second m columns from m+1 to 2*m is represent next
    % state when input is 1)

    NextState = zeros(2^m,2*m);

    for state=0:2^m-1
        b = dec2bin(state,m);

        b(end) = '';
        b = b-'0';
        tmp = [0,b,1,b];
        NextState(state+1,:) = tmp;
    end

    % Now generate path matric with minimum hamming distance

    trellisNodes = length(received)/n+1;

    trellis = repmat(500,2^m,trellisNodes);

    trellis(1,1)=0; % starting from 0

    for i=1:trellisNodes-1
        tmp = received(n*i-(n-1) : i*n); % n bits from received msg
        for state = 1:2^m
            t0 = state_output(state,1:n); % output when input is 0
            t1 = state_output(state,n+1:2*n); % output when input is 1

            % finding Euclidean distance of n bits received msg with output
            % when input is 0(Ud0) and output when input is (Ud1)
            Ud0 = sum((tmp-t0).^2);
```

```
            Ud1 = sum((tmp-t1).^2);

            next_st0 = NextState(state,1:m); %nextstate whene input is 0
            next_st1 = NextState(state,m+1:2*m); %nextstate whene input is 1

            %convert nextstate into number
            next_st0 = bin2dec(char(next_st0 + '0'));
            next_st1 = bin2dec(char(next_st1 + '0'));

            % updating trellis node with mininum hamming distance
            trellis(next_st0+1,i+1) = min(trellis(next_st0+1,i+1),
trellis(state,i)+Ud0);
            trellis(next_st1+1,i+1) = min(trellis(next_st1+1,i+1),
trellis(state,i)+Ud1);
        end
    end

    curr_state = zeros(1,m);
    decoded_msg = [];

    % for decoding msg traverse trellis from trellisNodes-1
    for i=trellisNodes-1:-1:1
        input = curr_state(1);

        % store input
        decoded_msg = [input decoded_msg];

        flag = 1;

        % Traverse through all state with according to input
        for state = 1:2^m
           if(flag == 1 & NextState(state,m*input+1:m*input+m) == curr_state)
               prev1 = dec2bin(state-1,m) - '0';
               prev1_dec = state;
               flag = 0;
           elseif(NextState(state,m*input+1:m*input+m) == curr_state)
               prev2 = dec2bin(state-1,m) - '0';
               prev2_dec = state;
           end
        end

        % output of prev1 and prev2 according to input
        output1 = state_output(prev1_dec,n*input+1:n*input+n);
        output2 = state_output(prev2_dec,n*input+1:n*input+n);
```

```
        % n bits from received msg
        tmp = received(n*i-(n-1):n*i);

        % finding Euclidean distance of n bits received msg with output
        Ud0 = sum((tmp-output1).^2);
        Ud1 = sum((tmp-output2).^2);


        % find minimun path(backtracing)
        if(trellis(prev1_dec,i)+Ud0 < trellis(prev2_dec,i)+Ud1)
            curr_state = prev1 ;
        elseif(trellis(prev1_dec,i)+Ud0 > trellis(prev2_dec,i)+Ud1)
            curr_state = prev2;
        else
            if(Ud0<Ud1)
                curr_state = prev1;
            else
                curr_state = prev2;
            end
        end

    end
end
```