CT 216
INTRODUCTION TO COMMUNICATION SYSTEM

PROJECT

LDPC CODES

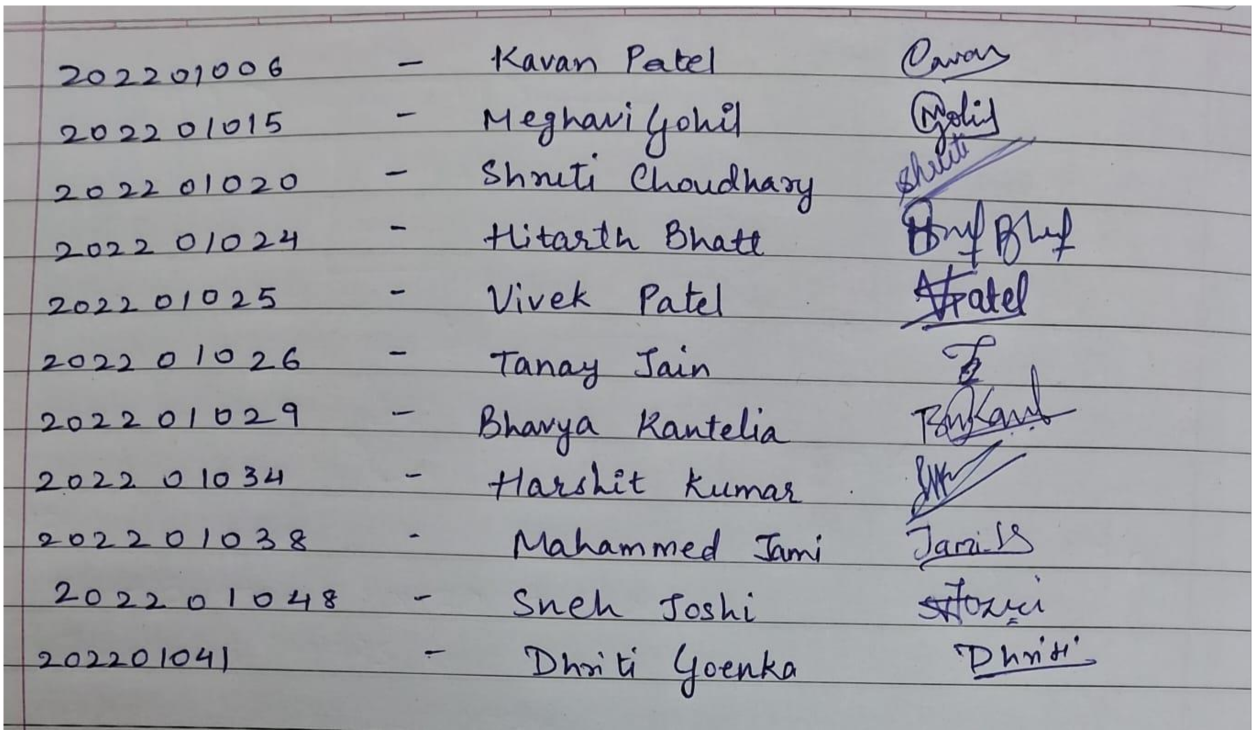
Lab Group 1 – Sub Group 3

Under the Guidance of Prof. Yash Vasavda

Mentor TA: Mr Harshaneel

HONOR 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 our own.
  - We make this pledge truthfully. We know that violation of this solemn pledge can carry grave consequences.



202201006	–	Kavan Patel
202201015	–	Meghavi Gohil
202201020	–	Shruti Choudhary
202201024	–	Hitarth Bhatt
202201025	–	Vivek Patel
202201026	–	Tanay Jain
202201029	–	Bhavya Kantelia
202201034	–	Harshit Kumar
202201038	–	Mahammed Jami
202201048	–	Sneh Joshi
202201041	–	Dhriti Goenka

CONTENTS:

1. Hard Decision Decoding Code
2. Soft Decision Decoding Code
3. Results
    i. Matrix NR_2_6_52
    ii. Matrix NR_1_5_352
4. Analysis with Shannon's Bound
5. Appendix A

HARD DECISION DECODING:

```
colors = [ 0.0, 0.7, 0.8;
           0.12, 0.34, 0.57;
           0.91, 0.15, 0.76;
           0.31, 0.12, 0.77;
           0.93, 0.13, 0.65;
           0.55, 0.51, 0.87;
           0.61, 0.78, 0.79;
           0.01, 0.31, 0.39;
           0.71, 0.25, 0.81;
           0.83, 0.69, 0.44;
           0.06, 0.40, 0.74;
           0.18, 0.18, 0.53;
           0.34, 0.72, 0.53;
           0.94, 0.38, 0.64;
           0.70, 0.15, 0.88;
           0.60, 0.67, 0.09;
           0.91, 0.29, 0.31;
           0.80, 0.86, 0.31;
           0.19, 0.93, 0.42;
           0.95, 0.79, 0.21;
           0.14, 0.41, 0.05
         ];
%for matrix NR_2_6_52
baseGraph5GNR = 'NR_2_6_52'; % load 5G NR LDPC base H matrix
coderate = [1/4 1/3 1/2 3/5];
eb_no_dbvec = 0:0.5:10;
[B,Hfull,z] = nrldpc_Hmatrix(baseGraph5GNR,52); % Convert the base H matrix to binary H matrix
nsim = 1000;
max_it = 20;
iterations = 1:1:max_it;

for cr = coderate

    %performing rate matching
    [mb,nb] = size(B); kb = nb - mb; % 5G NR specific details
    kNumInfoBits = kb * z; % Number of information bits
    k_pc = kb-2; nbRM = ceil(k_pc/cr)+2; % Some 5G NR specific details
    nBlockLength = nbRM * z; % Number of encoded bits
    H = Hfull(:,1:nBlockLength);
    nChecksNotPunctured = mb*z - nb*z + nBlockLength;
    H = H(1:nChecksNotPunctured,:); % this is the binary H matrix

    [row,col] = size(H);
    L = zeros(size(H)); %initialising L
    k = col - row;
    cn_to_vn_map = cn_vn(H); %shows ith cn connected to which all vns
    vn_to_cn_map = vn_cn(H); %shows ith vn connected to which all cns

    d_iter = 1;
    decoding_error = zeros(1,length(eb_no_dbvec));
    bit_error = zeros(1,length(eb_no_dbvec));
    for eb_no_db = eb_no_dbvec
```

```
        eb_no_db
        eb_no = 10^(eb_no_db/10);
        sigma = sqrt(1/(2*cr*eb_no)); %noise variance
        success = 0;
        error1 = 0;
        itr_success = nsim.*ones(1,max_it);
        vn_sum_vec = zeros(1,col);

        for sim=1:nsim

            org_msg = randi([0 1],[k 1]); % Generate information (or message) bit vector
            encoded_msg = nrldpc_encode(B,z,org_msg'); % Encode using 5G NR LDPC base matrix
            encoded_msg = encoded_msg(1:nBlockLength);

            n = length(encoded_msg);
            %performing bpsk modulation
            bpsk_msg = 1 - 2.*encoded_msg;
            %generating noise
            noise = sigma * randn(1,n);

            received_bpsk = bpsk_msg + noise;
            %changing message back to bits
            received_bits = (received_bpsk<0);
            prev_msg1 = received_bits;
            c_hat = zeros(1,col);

            %performing hard decision decoding - uses recived bits to decode
            for it = 1:1:max_it

                %message from VN to CN

                %for 1st iteration, load all received bits into respective VNs and send them
directly to CN
                if(it==1)
                    for i=1:col
                        for j=vn_to_cn_map{i,1}
                            L(j,i) = received_bits(1,i);
                        end
                    end

                %for all other iterations, perform majority voting of the bits received by the
VN
                else
                    for i = 1:col
                        for j=vn_to_cn_map{i,1}
                            ele = vn_sum_vec(1,i) - L(j,i);
                            L(j,i) = ele>(length(vn_to_cn_map{i,1})/2);
                        end
                    end
                end

                %message passing from CN to VN using XOR
                for i=1:row
                    xor_val = 0;
                    %computing xor of all the values received by CN
```

```
                for j=cn_to_vn_map{i,1}
                    xor_val = mod((xor_val+L(i,j)),2);
                end

                %sending the message to particular VNs connected
                for j=cn_to_vn_map{i,1}
                    L(i,j) = mod((xor_val+L(i,j)),2);
                end
            end

            %finding the sum of values received by each VN and performing
            %majority voting with originally received bit to estimate c_hat
            for i = 1:col
                sum1 = received_bits(1,i);
                temp = L(:,i);
                sum1 = sum1 + sum(temp);
                vn_sum_vec(1,i)=sum1;
                c_hat(1,i) = sum1>((length(vn_to_cn_map{i,1})+1)/2);
            end

             %if c_hat is equal to the encoded message, decoding is successful, so break
            if(sum(xor(c_hat(1:k),org_msg'))==0)
                success = success+1;
                break;
            else
                itr_success(1,it)=itr_success(1,it)-1;
            end

            %{
             %calculating BER
             for i=1:col
                 if c_hat(1,i)~=encoded_msg(1,i)
                     error1=error1+1;
                 end
             end
            %}

            %if c_hat equal to previously computed c_hat, then also break
            if(sum(xor(prev_msg1,c_hat))==0)
                for tmp_itr=it+1:max_it
                    itr_success(1,tmp_itr)=itr_success(1,tmp_itr)-1;
                end
                break;
            end
            prev_msg1 = c_hat;
        end
    end
    plot(iterations,itr_success./nsim,'Color',colors(d_iter,:));
    hold on;
    decoding_error(1,d_iter) = (nsim-success)/nsim;
    bit_error(1,d_iter) = error1/(nsim*col);
    d_iter = d_iter+1;
end
hold off;
xlabel("Iteration number");
```

```
    ylabel('Success Probability at each iteration');
    title('Success Probability v/s iteration for Hard Decoding');
    grid on;

legend('0.0','0.5','1.0','1.5','2.0','2.5','3.0','3.5','4.0','4.5','5.0','5.5','6.0','6.5','7.
0','7.5','8.0','8.5','9.0','9.5','10.0');
    plot(eb_no_dbvec,decoding_error,'LineWidth',2);
    %plot(eb_no_dbvec,bit_error,'Linewidth',2);
    hold on;
end
xlabel("Eb/No (dB)");
ylabel("Decoding error probability");
title("Hard Decision Decoding error probability");
legend('Coderate = 1/4', 'Coderate = 1/3', 'Coderate = 1/2', 'Coderate = 3/5');
hold off;

%{
xlabel("Eb/No (dB)");
ylabel("BER");
title("Hard decision Bit error rate probability");
legend('Coderate = 1/4', 'Coderate = 1/3', 'Coderate = 1/2', 'Coderate = 3/5');
hold off;
%}

% Add a section break

function [B,H,z] = nrldpc_Hmatrix(BG,z)
    load(sprintf('%s.txt',BG),BG);
    B = NR_2_6_52;
    [mb,nb] = size(B);
    H = zeros(mb*z,nb*z);
    Iz = eye(z); I0 = zeros(z);
    for kk = 1:mb
        tmpvecR = (kk-1)*z+(1:z);
        for kk1 = 1:nb
            tmpvecC = (kk1-1)*z+(1:z);
            if B(kk,kk1) == -1
                H(tmpvecR,tmpvecC) = I0;
            else
                H(tmpvecR,tmpvecC) = circshift(Iz,-B(kk,kk1));
            end
        end
    end

    [U,N]=size(H); K = N-U; % n = length of codeword, u = number of CNs or parities, k =
length of original message
    P = H(:,1:K);
    G = [eye(K); P];
    Z = H*G;
end

function out=cn_vn(H)
    [row, col]=size(H);
    out=cell(row,1);
    for i = 1:row
```

```
        out{i,1} = [];
    end
    for i=1:row
        for j=1:col
            if(H(i,j)==1)
                out{i,1} = [out{i,1} j];
            end
        end
    end
end


function out=vn_cn(H)
    [row, col]=size(H);
    out=cell(col,1);
    for i = 1:col
        out{i,1} = [];
    end
    for i=1:col
        for j=1:row
            if(H(j,i)==1)
                out{i,1} = [out{i,1} j];
            end
        end
    end
end


function cword = nrldpc_encode(B,z,msg)
    %B: base matrix
    %z: expansion factor
    %msg: message vector, length = (#cols(B)-#rows(B))*z
    %cword: codeword vector, length = #cols(B)*z

    [m,n] = size(B);

    cword = zeros(1,n*z);
    cword(1:(n-m)*z) = msg;

    %double-diagonal encoding
    temp = zeros(1,z);
    for i = 1:4 %row 1 to 4
        for j = 1:n-m %message columns
            temp = mod(temp + mul_sh(msg((j-1)*z+1:j*z),B(i,j)),2);
        end
    end
    if B(2,n-m+1) == -1
        p1_sh = B(3,n-m+1);
    else
        p1_sh = B(2,n-m+1);
    end
    cword((n-m)*z+1:(n-m+1)*z) = mul_sh(temp,z-p1_sh); %p1
    %Find p2, p3, p4
    for i = 1:3
        temp = zeros(1,z);
```

```
        for j = 1:n-m+i
            temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z),B(i,j)),2);
        end
        cword((n-m+i)*z+1:(n-m+i+1)*z) = temp;
    end
    %Remaining parities
    for i = 5:m
        temp = zeros(1,z);
        for j = 1:n-m+4
            temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z),B(i,j)),2);
        end
        cword((n-m+i-1)*z+1:(n-m+i)*z) = temp;
    end
end

function y = mul_sh(x,k)
    if(k==-1)
        y = zeros(1,length(x));
    else
        y = [x(k+1:end) x(1:k)];
    end
end
```

SOFT DECISION DECODING

```
colors = [ 0.0, 0.7, 0.8;
           0.12, 0.34, 0.57;
           0.91, 0.15, 0.76;
           0.31, 0.12, 0.77;
           0.93, 0.13, 0.65;
           0.55, 0.51, 0.87;
           0.61, 0.78, 0.79;
           0.01, 0.31, 0.39;
           0.71, 0.25, 0.81;
           0.83, 0.69, 0.44;
           0.06, 0.40, 0.74;
           0.18, 0.18, 0.53;
           0.34, 0.72, 0.53;
           0.94, 0.38, 0.64;
           0.70, 0.15, 0.88;
           0.60, 0.67, 0.09;
           0.91, 0.29, 0.31;
           0.80, 0.86, 0.31;
           0.19, 0.93, 0.42;
           0.95, 0.79, 0.21;
           0.14, 0.41, 0.05
         ];
%for matrix NR_2_6_52
baseGraph5GNR = 'NR_2_6_52'; % load 5G NR LDPC base H matrix
coderate = [1/4 1/3 1/2 3/5];
eb_no_dbvec = 0:0.5:10;
[B,Hfull,z] = nrldpc_Hmatrix(baseGraph5GNR,52); % Convert the base H matrix to binary H matrix
nsim = 1000;
max_it = 20;
```

```
iterations = 1:1:max_it;
for cr = coderate
    cr
    %performing rate matching
    [mb,nb] = size(B); kb = nb - mb; % 5G NR specific details
    kNumInfoBits = kb * z; % Number of information bits

    k_pc = kb-2; nbRM = ceil(k_pc/cr)+2; % Some 5G NR specific details

    nBlockLength = nbRM * z; % Number of encoded bits
    H = Hfull(:,1:nBlockLength);
    nChecksNotPunctured = mb*z - nb*z + nBlockLength;
    H = H(1:nChecksNotPunctured,:); % this is the binary H matrix

    %rate matching done
    [row,col] = size(H);
    L = zeros(size(H));
    k = col - row;
    cn_to_vn_map = cn_vn(H); %shows ith cn connected to which all vns
    vn_to_cn_map = vn_cn(H); %shows ith vn connected to which all cns


    %performing soft decoding
    %estimates on the basis of original vector received without
    %changing it to bits
    d_iter = 1;
    decoding_error = zeros(1,length(eb_no_dbvec));
    bit_error = zeros(1,length(eb_no_dbvec));

    for eb_no_db = eb_no_dbvec
        eb_no = 10^(eb_no_db/10);
        sigma = sqrt(1/(2*cr*eb_no));
        success = 0;
        error1 = 0;
        itr_success = nsim.*ones(1,max_it);
        vn_sum_vec = zeros(1,col);

        for sim=1:nsim

            org_msg = randi([0 1],[k 1]); % Generate information (or message) bit vector
            encoded_msg = nrldpc_encode(B,z,org_msg'); % Encode using 5G NR LDPC base matrix
            encoded_msg = encoded_msg(1:nBlockLength);

            n = length(encoded_msg);
            %performing bpsk modulation
            bpsk_msg = 1 - 2.*encoded_msg;
            %generating noise
            noise = sigma * randn(1,n);

            received_bpsk = bpsk_msg + noise;
            %changing message back to bits
            received_bits = (received_bpsk<0);
            prev_msg = received_bits;
            c_hat = zeros(1,col);
```

```
            for it =1 :1:max_it

                %message from VN to CN

                %for 1st iteration, load all received values into VN and
                %send them directly to CN
                if(it==1)
                    for i=1:col
                        for j=vn_to_cn_map{i,1}
                            L(j,i) = received_bpsk(1,i);
                        end
                    end

                %otherwise subtract the current value from the total sum vec.
                else
                    for i = 1:col
                        for j=vn_to_cn_map{i,1}
                            L(j,i) = vn_sum_vec(1,i) - L(j,i);
                        end
                    end
                end


                %message from CN to VN using minsum approximation
                for i=1:row
                    min1=1e9;               %first minimum
                    min2=1e9;               %second minimum
                    pos=-1;                 %VN number which has minimum1 value
                    total_sign=1;           % the sign obtained by multiplying all the non-
zero elemnts in the row

                    for j=cn_to_vn_map{i,1}
                        ele = abs(L(i,j));

                        %computing the minimums
                        if(ele<=min1)
                            min2=min1;
                            min1=ele;
                            pos = j;
                        elseif(ele<=min2 && ele>min1)
                            min2=ele;
                        end
                        %computing overall sign
                        if(L(i,j)~=0)
                            total_sign = total_sign*(sign(L(i,j)));
                        end
                    end

                    %sending the message
                    for j=cn_to_vn_map{i,1}
                        if(j~=pos)
                            L(i,j) = total_sign * sign(L(i,j)) * min1;
                        else
                            L(i,j) = total_sign * sign(L(i,j)) * min2;
                        end
```

```
                end
            end

            %finding sum of values received by each vn
            for i = 1:col
                sum1 = received_bpsk(1,i);
                temp = L(:,i);
                sum1 = sum1 + sum(temp);
                vn_sum_vec(1,i)=sum1;
            end

            c_hat = (vn_sum_vec<0);

            if(sum(xor(c_hat(1:k),org_msg'))==0)
                success = success+1;
                break;
            else
                itr_success(1,it)=itr_success(1,it)-1;
            end

             %{
             %calculating BER
             for i=1:col
                 if c_hat(1,i)~=encoded_msg(1,i)
                     error1=error1+1;
                 end
             end
            %}

            if(sum(xor(prev_msg,c_hat))==0)
                for tmp_itr=it+1:max_it
                    itr_success(1,tmp_itr)=itr_success(1,tmp_itr)-1;
                end
                break;
            end
            prev_received = c_hat;

        end

    end
    plot(iterations,itr_success./nsim,'Color',colors(d_iter,:));

    grid on;
    hold on;
    decoding_error(1,d_iter) = (nsim-success)/nsim;
    bit_error(1,d_iter) = error1/(nsim*col);
    d_iter = d_iter+1;
end
hold off;
xlabel("Iteration number");
ylabel('Success Probability at each iteration');
title('Success Probability v/s iteration for Soft Decoding');
grid on;
```

```
legend('0.0','0.5','1.0','1.5','2.0','2.5','3.0','3.5','4.0','4.5','5.0','5.5','6.0','6.5','7.
0','7.5','8.0','8.5','9.0','9.5','10.0');
    plot(eb_no_dbvec,decoding_error,'LineWidth',2);
    %plot(eb_no_dbvec,bit_error,'Linewidth',2);
    hold on;

end
xlabel("Eb/No (dB)");
ylabel("Decoding error probability");
title("Soft Decision Decoding error probability");
legend('Coderate = 1/4', 'Coderate = 1/3', 'Coderate = 1/2', 'Coderate = 3/5');
hold off;

%{
%for matrix NR_1_5_352
baseGraph5GNR = 'NR_1_5_352'; % load 5G NR LDPC base H matrix
coderate = [1/3 1/2 3/5 4/5];
[B,Hfull,z] = nrldpc_Hmatrix(baseGraph5GNR,352); % Convert the base H matrix to binary H
matrix
%}

%{
xlabel("Eb/No (dB)");
ylabel("BER");
title("Soft decision Bit error rate probability");
legend('Coderate = 1/4', 'Coderate = 1/3', 'Coderate = 1/2', 'Coderate = 3/5');
hold off;
%}

% Add a section break

function [B,H,z] = nrldpc_Hmatrix(BG,z)
    load(sprintf('%s.txt',BG),BG);
    B = NR_2_6_52;
    [mb,nb] = size(B);
    H = zeros(mb*z,nb*z);
    Iz = eye(z); I0 = zeros(z);
    for kk = 1:mb
        tmpvecR = (kk-1)*z+(1:z);
        for kk1 = 1:nb
            tmpvecC = (kk1-1)*z+(1:z);
            if B(kk,kk1) == -1
                H(tmpvecR,tmpvecC) = I0;
            else
                H(tmpvecR,tmpvecC) = circshift(Iz,-B(kk,kk1));
            end
        end
    end

    [U,N]=size(H); K = N-U; % n = length of codeword, u = number of CNs or parities, k =
length of original message
    P = H(:,1:K);
    G = [eye(K); P];
    Z = H*G;
```

```
end

function out=cn_vn(H)
    [row, col]=size(H);
    out=cell(row,1);
    for i = 1:row
        out{i,1} = [];
    end
    for i=1:row
        for j=1:col
            if(H(i,j)==1)
                out{i,1} = [out{i,1} j];
            end
        end
    end
end


function out=vn_cn(H)
    [row, col]=size(H);
    out=cell(col,1);
    for i = 1:col
        out{i,1} = [];
    end
    for i=1:col
        for j=1:row
            if(H(j,i)==1)
                out{i,1} = [out{i,1} j];
            end
        end
    end
end


function cword = nrldpc_encode(B,z,msg)
    %B: base matrix
    %z: expansion factor
    %msg: message vector, length = (#cols(B)-#rows(B))*z
    %cword: codeword vector, length = #cols(B)*z

    [m,n] = size(B);

    cword = zeros(1,n*z);
    cword(1:(n-m)*z) = msg;

    %double-diagonal encoding
    temp = zeros(1,z);
    for i = 1:4 %row 1 to 4
        for j = 1:n-m %message columns
            temp = mod(temp + mul_sh(msg((j-1)*z+1:j*z),B(i,j)),2);
        end
    end
    if B(2,n-m+1) == -1
        p1_sh = B(3,n-m+1);
    else
```

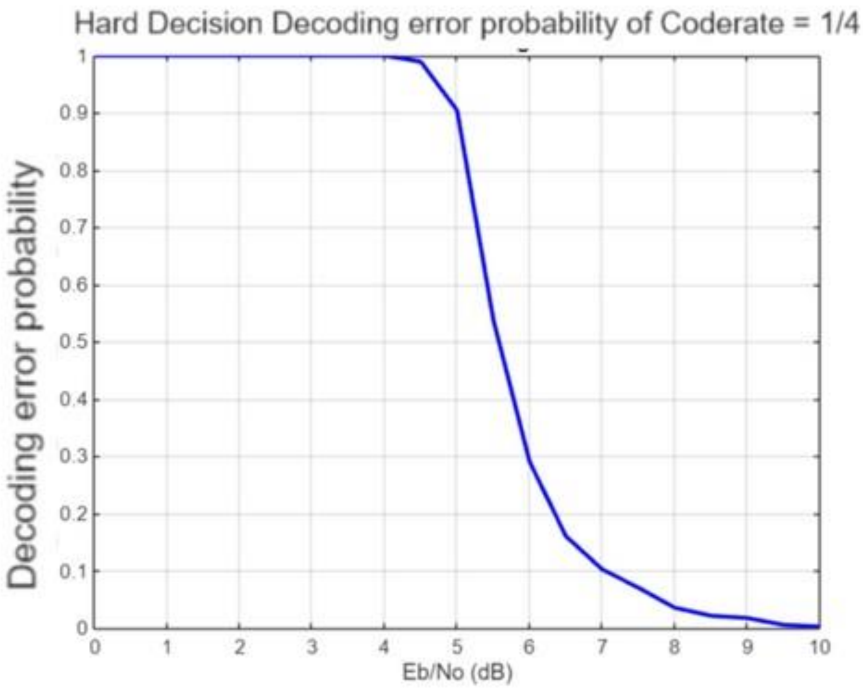
```
        p1_sh = B(2,n-m+1);
    end
    cword((n-m)*z+1:(n-m+1)*z) = mul_sh(temp,z-p1_sh); %p1
    %Find p2, p3, p4
    for i = 1:3
        temp = zeros(1,z);
        for j = 1:n-m+i
            temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z),B(i,j)),2);
        end
        cword((n-m+i)*z+1:(n-m+i+1)*z) = temp;
    end
    %Remaining parities
    for i = 5:m
        temp = zeros(1,z);
        for j = 1:n-m+4
            temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z),B(i,j)),2);
        end
        cword((n-m+i-1)*z+1:(n-m+i)*z) = temp;
    end
end

function y = mul_sh(x,k)
    if(k==-1)
        y = zeros(1,length(x));
    else
        y = [x(k+1:end) x(1:k)];
    end
end
```

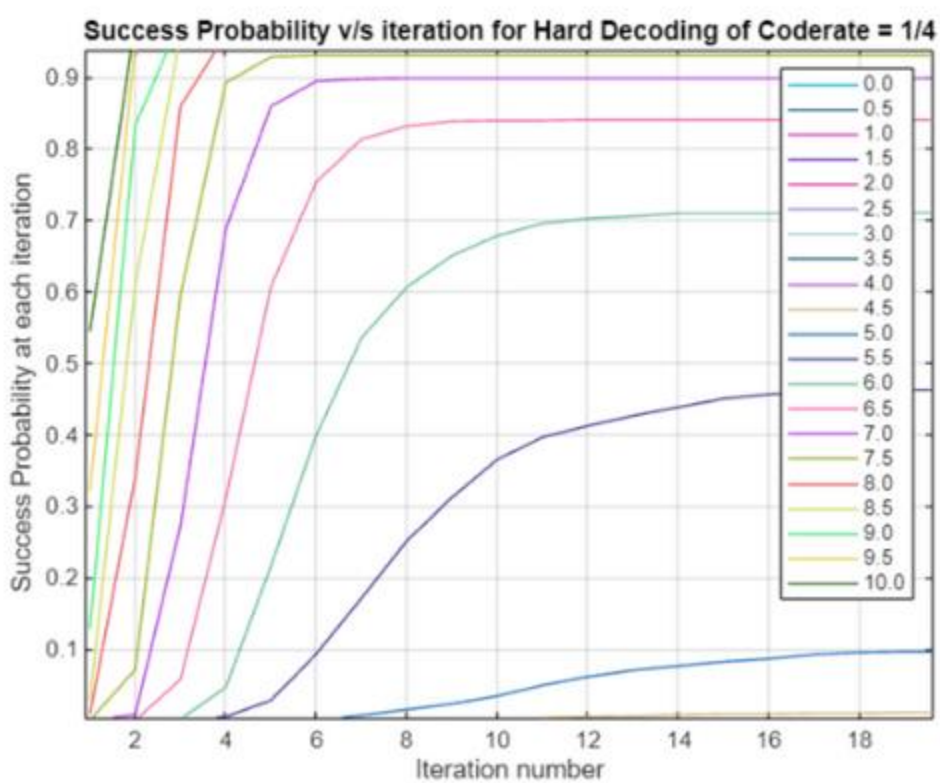
RESULTS:

1. **For Matrix NR_2_6_52:**

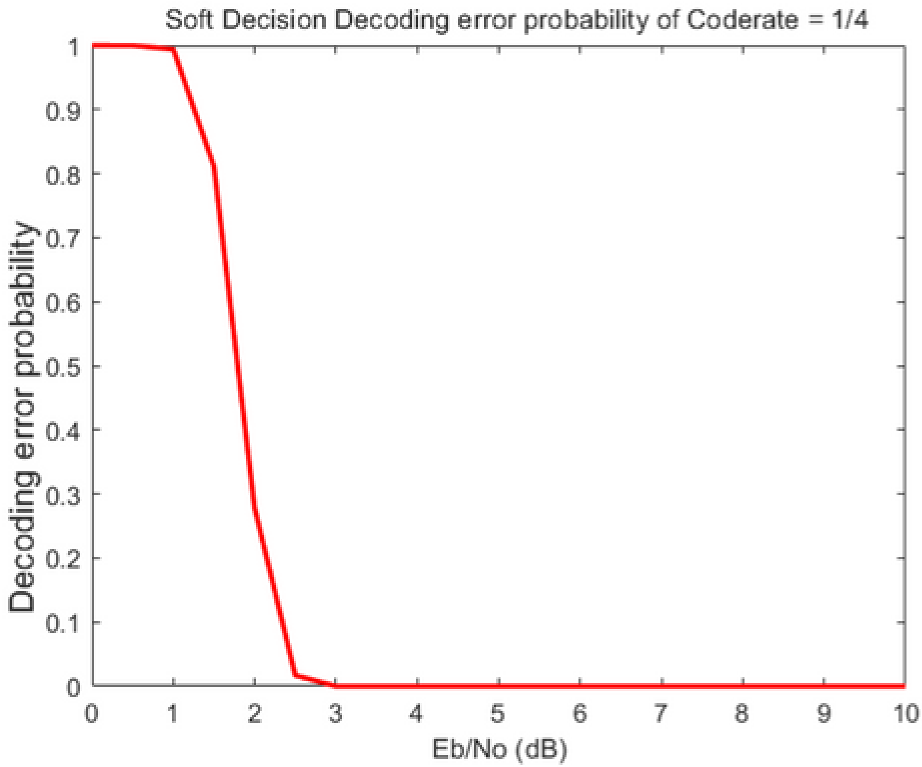
   a. Code rate = 1/4

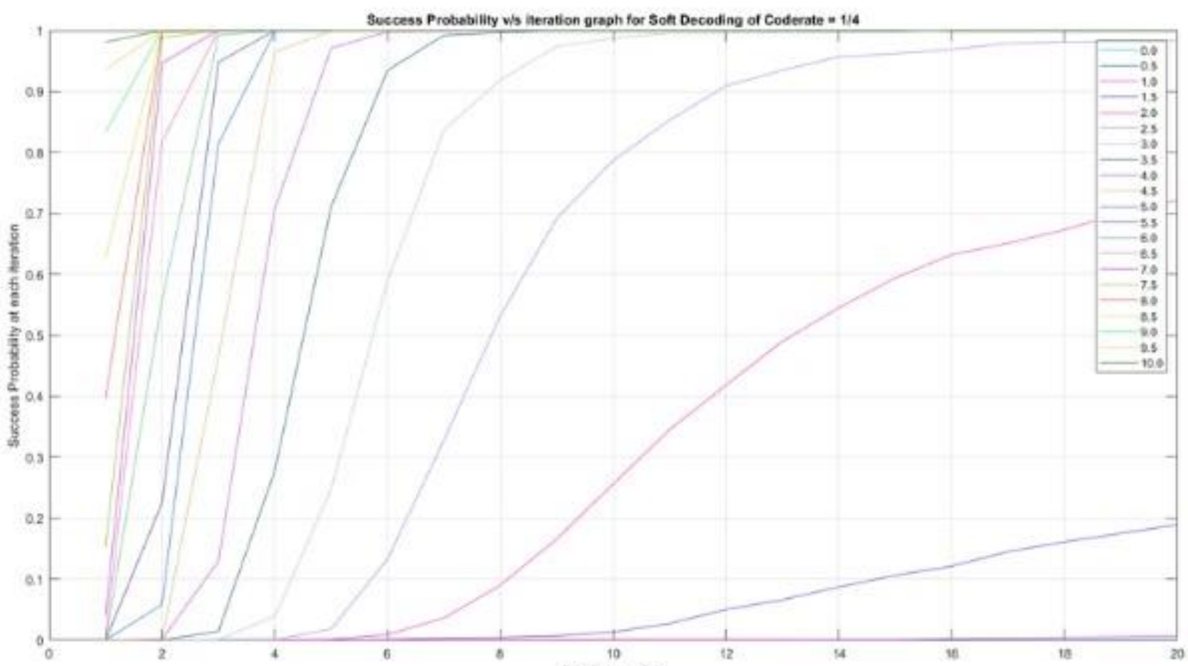
      Hard Decision Decoding:





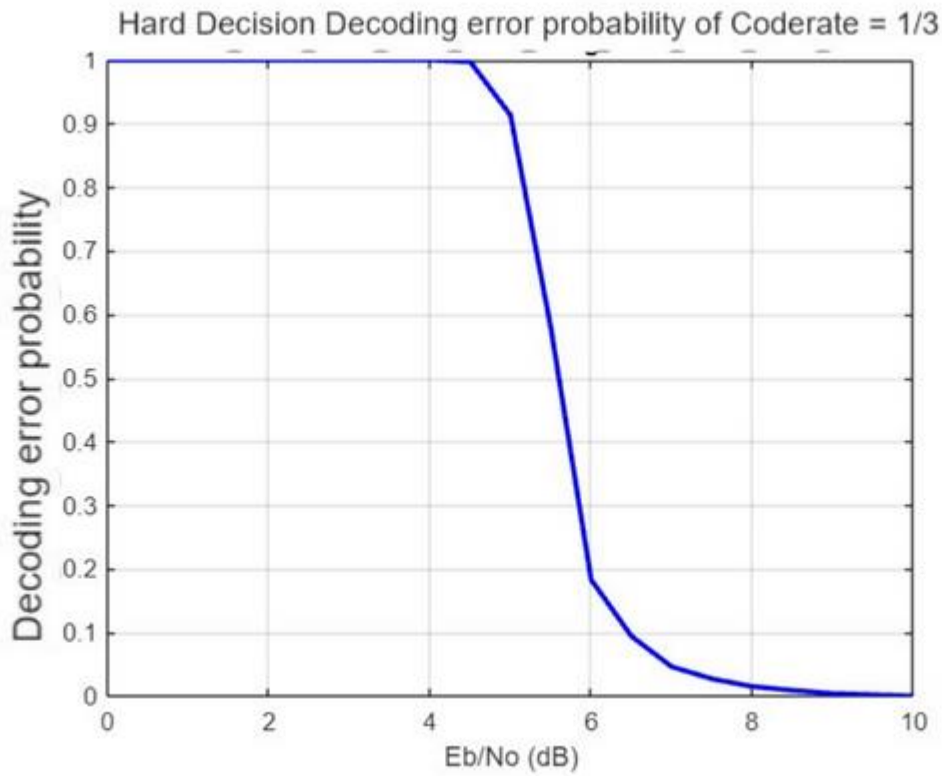
Soft Decision Decoding:

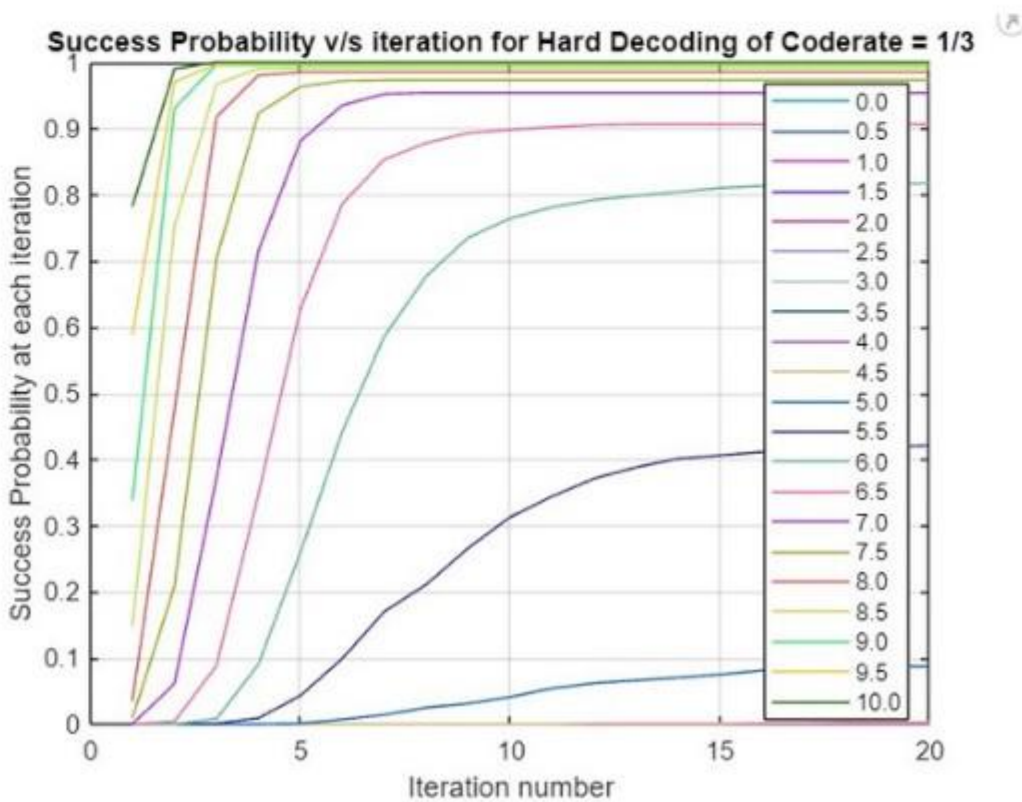




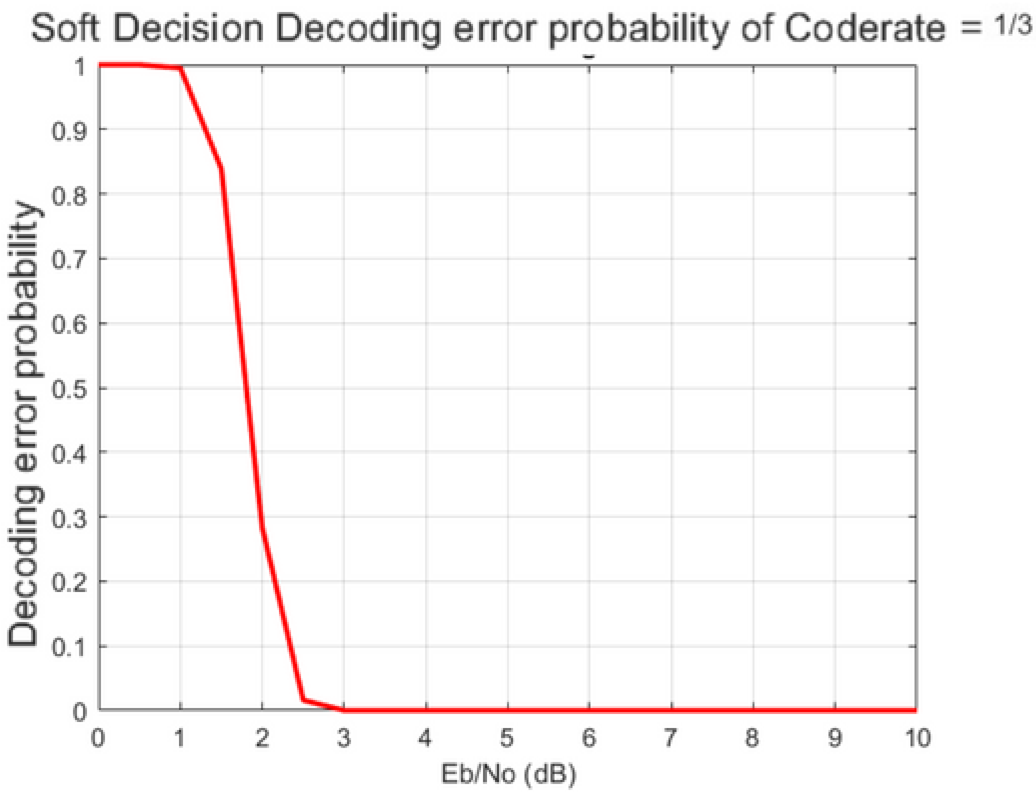
b. Code rate = 1/3

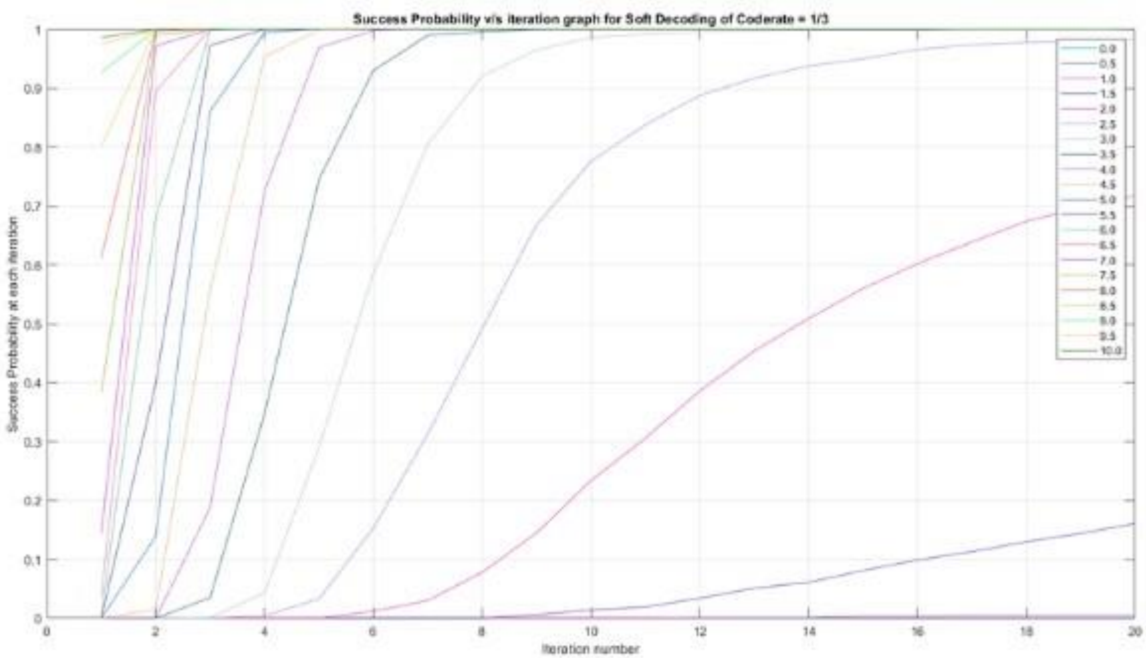
Hard Decision Decoding:





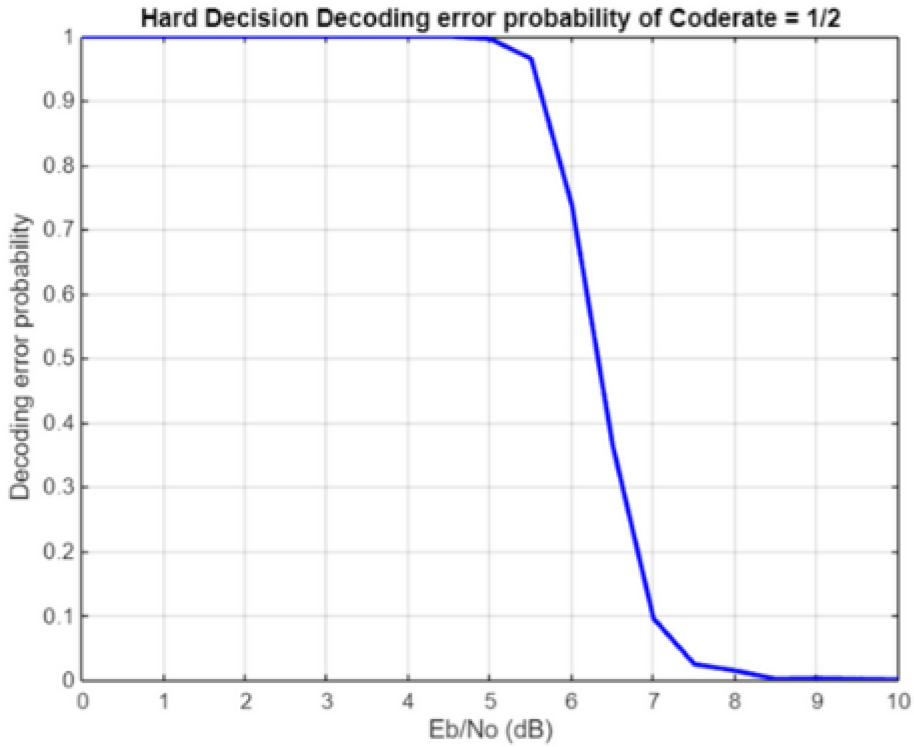
Soft Decision Decoding:

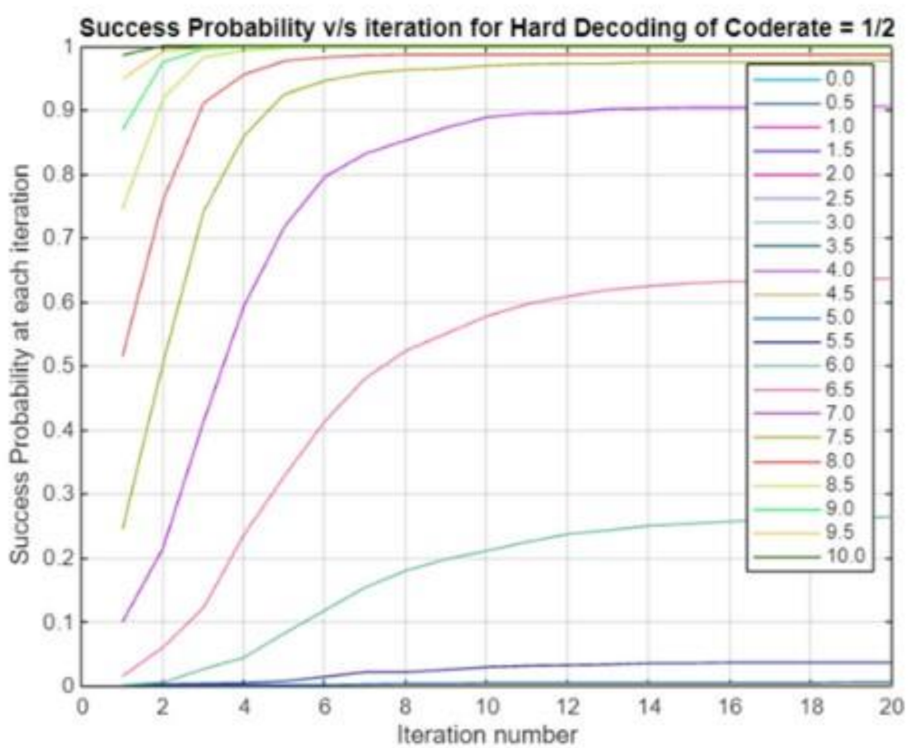




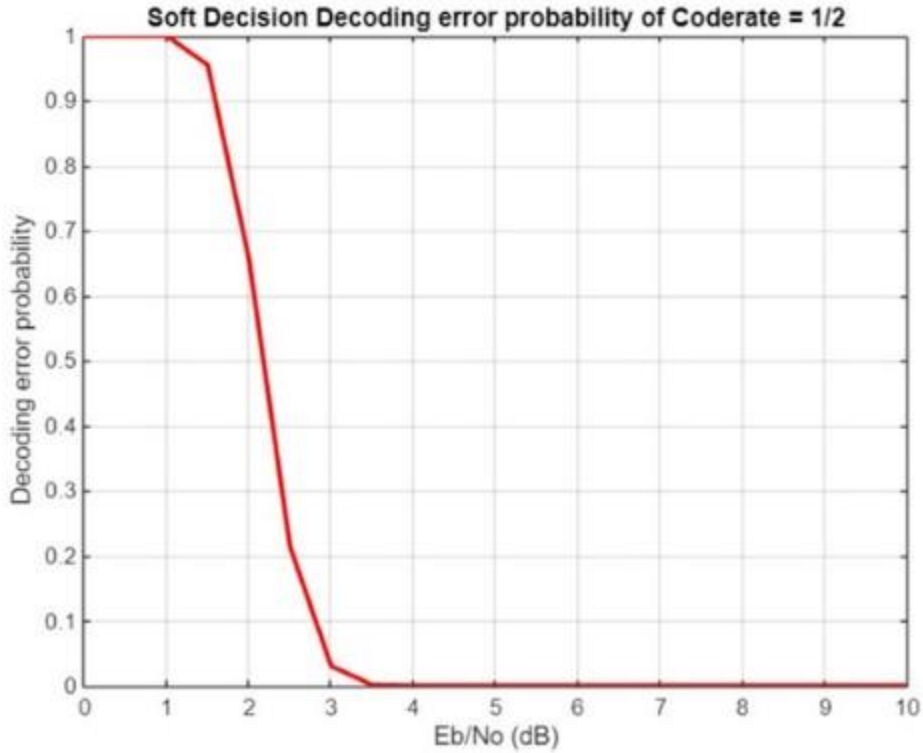
c. Code rate = 1/2

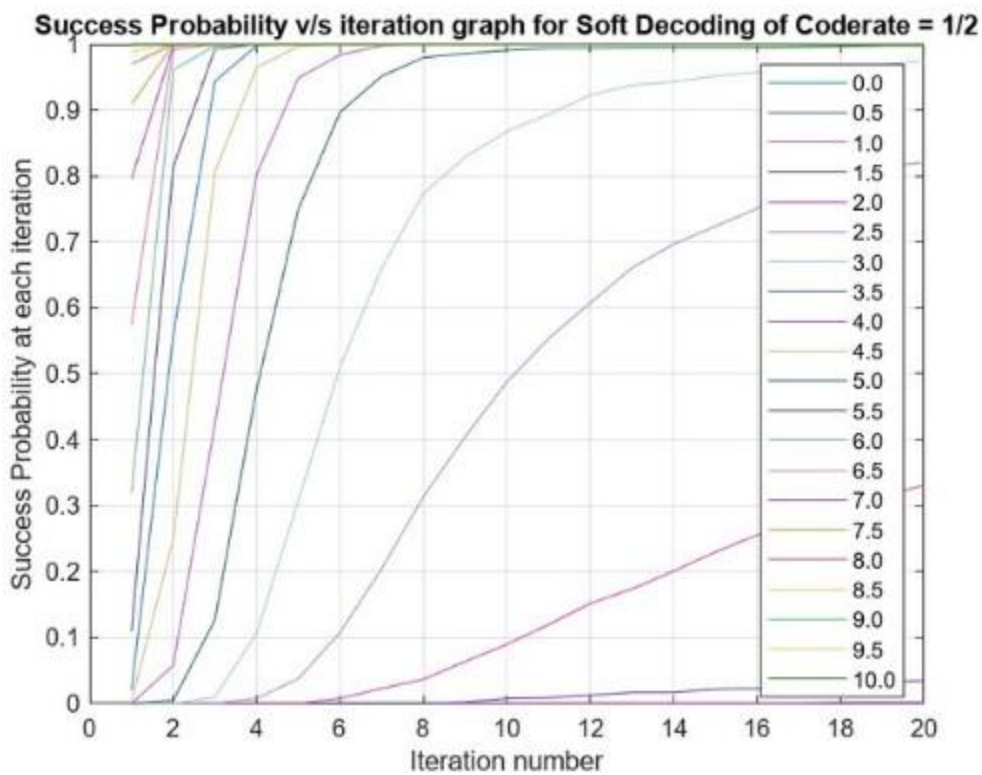
Hard Decision Decoding:





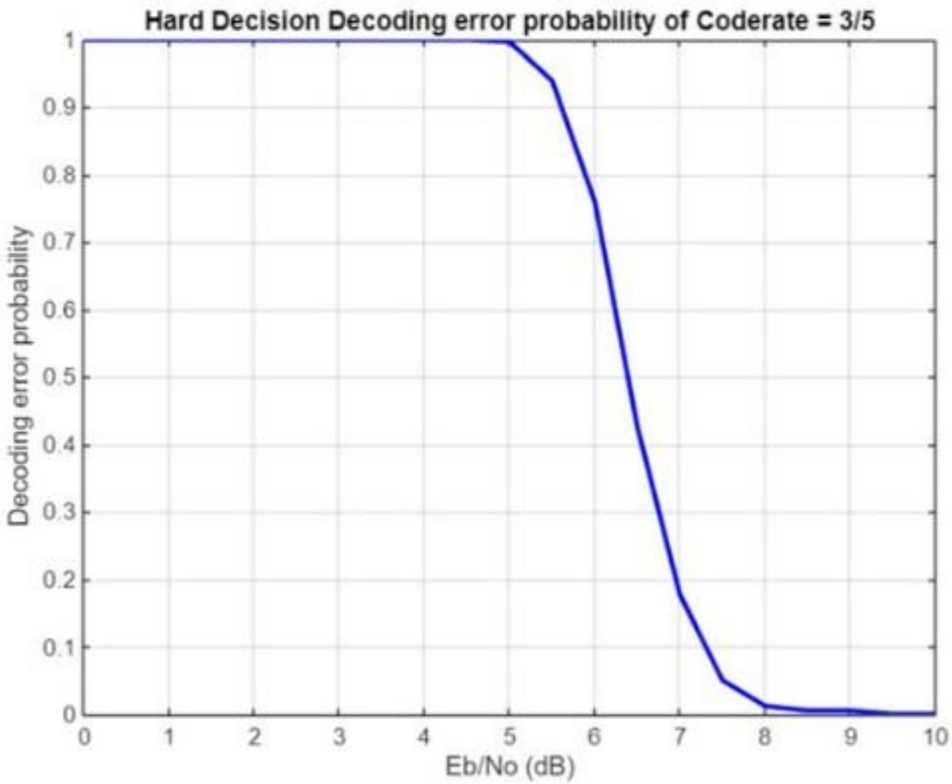
Soft Decision Decoding:

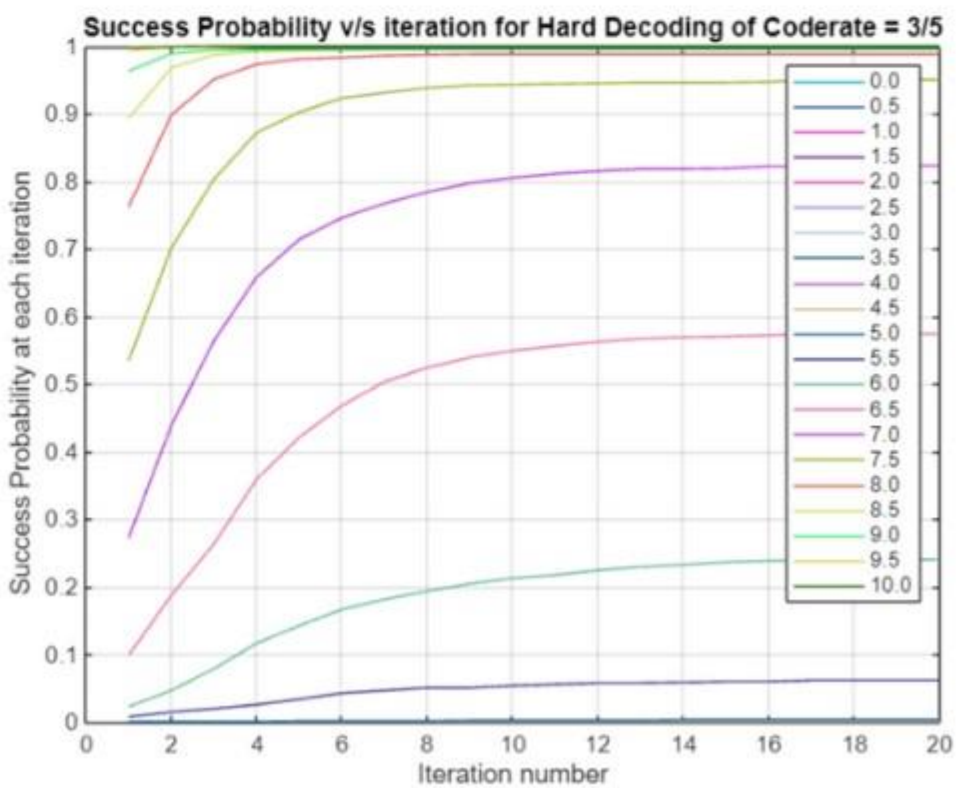




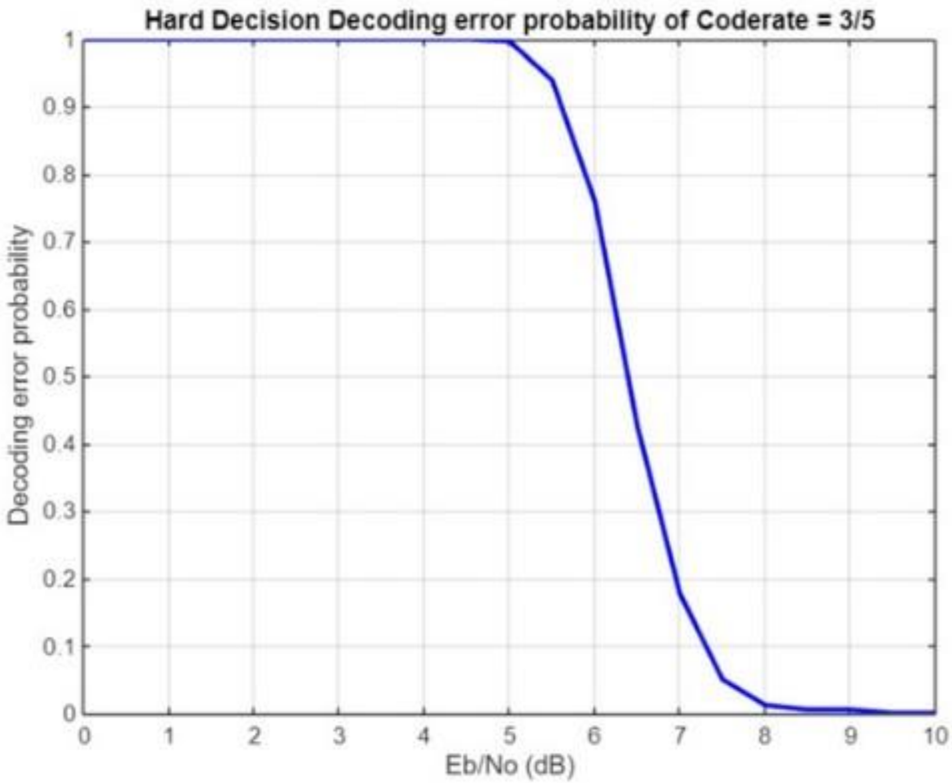
d. Code rate = 3/5

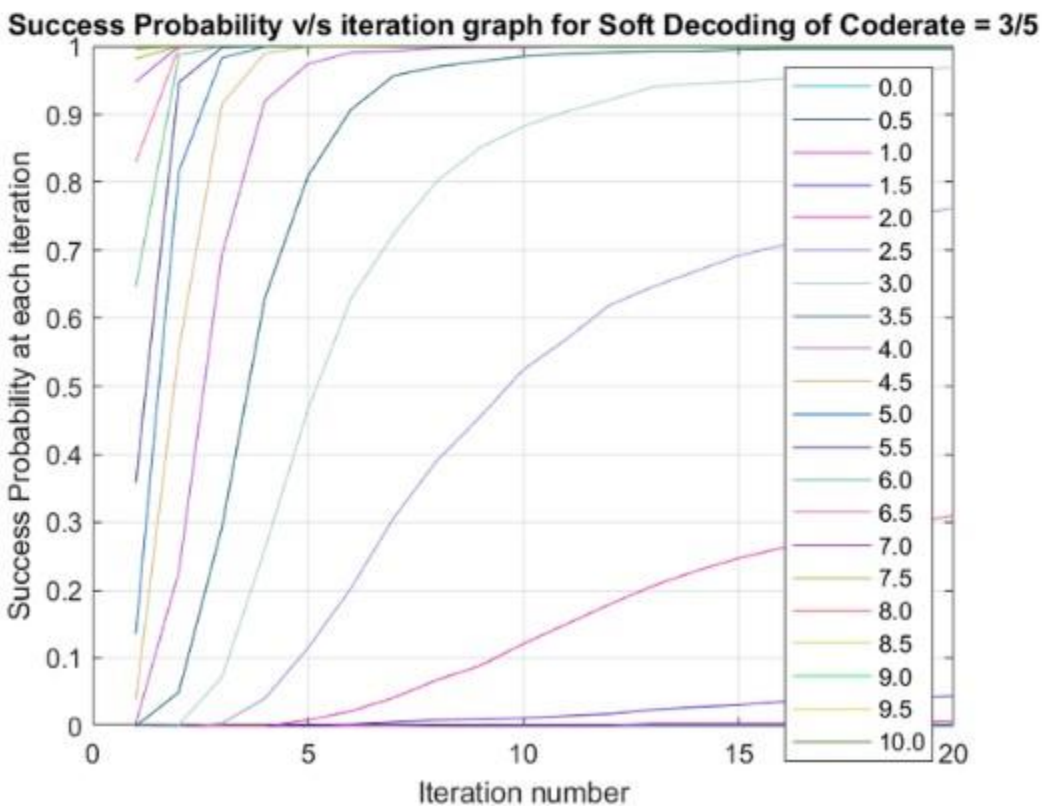
Hard Decision Decoding:



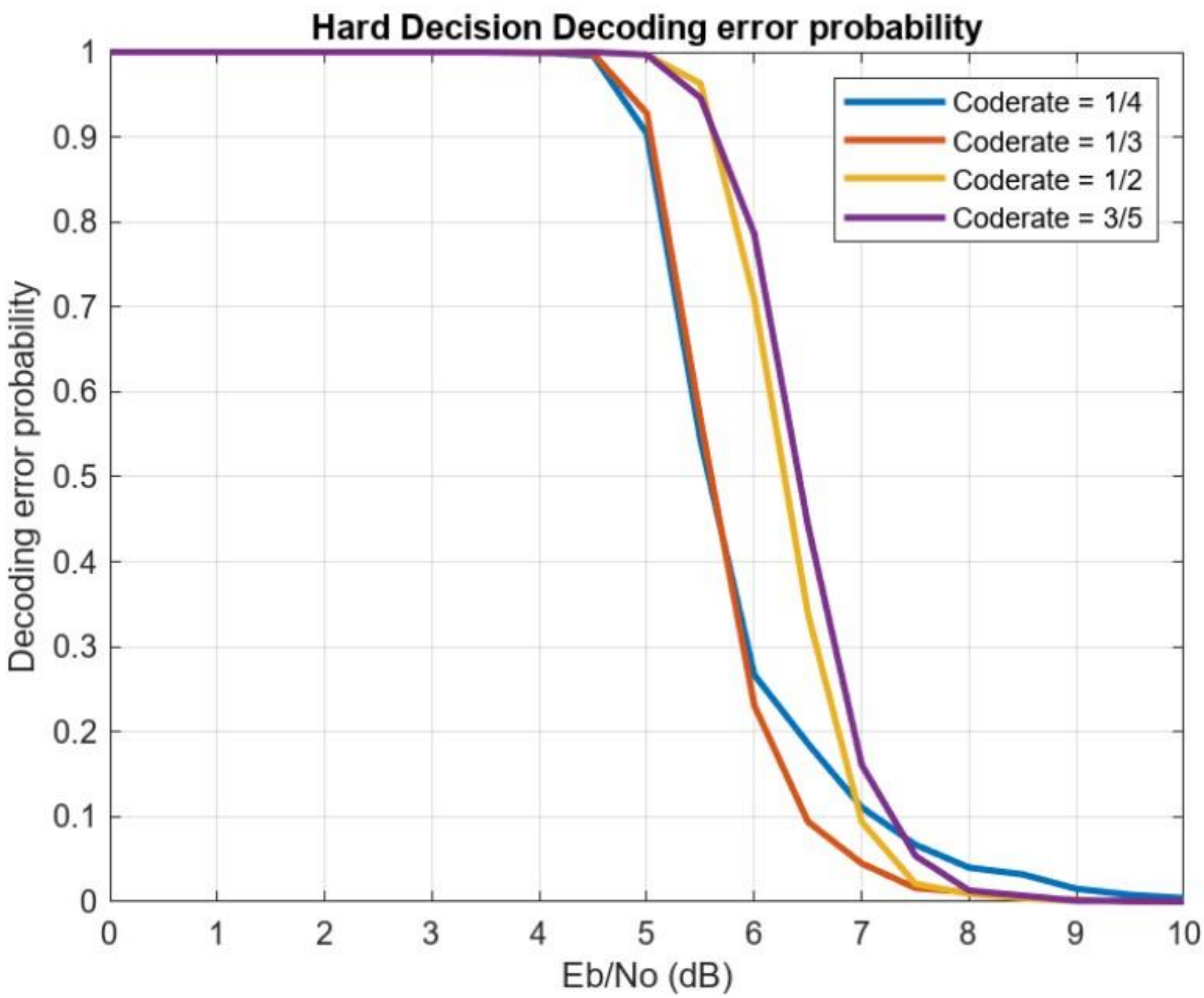


Soft Decision Decoding:

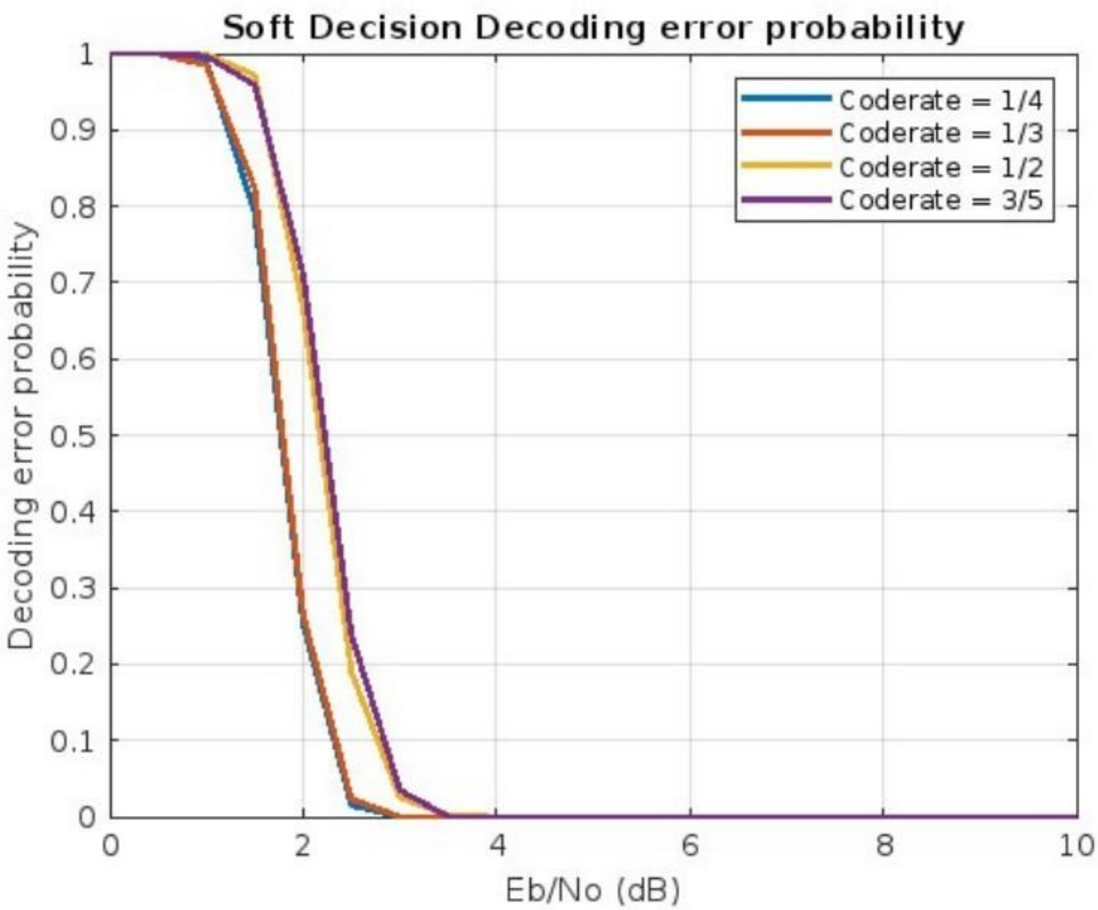




Comparison of all graphs obtained by Hard Decision decoding:

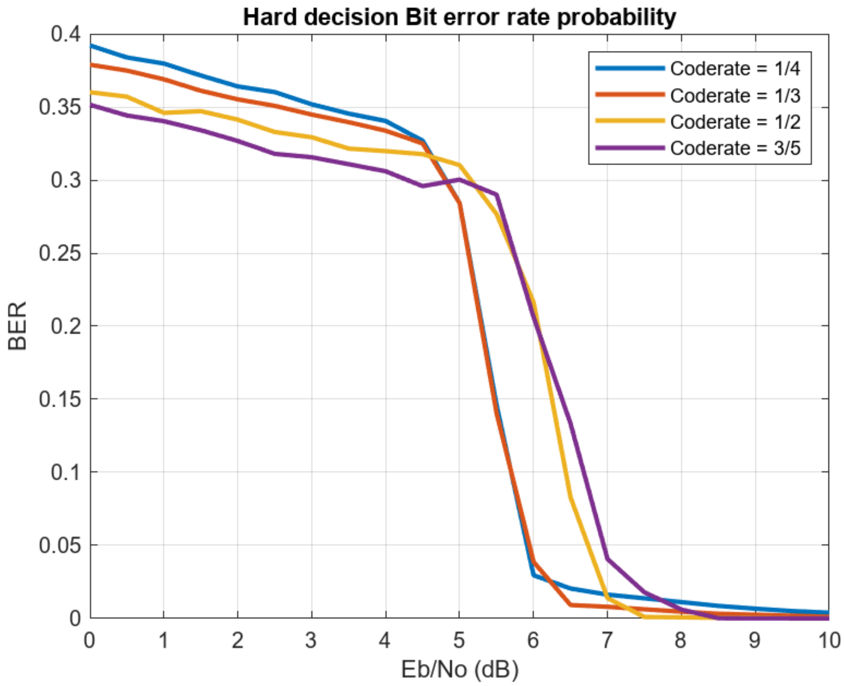


Comparison of all graphs obtained by Soft Decision decoding:

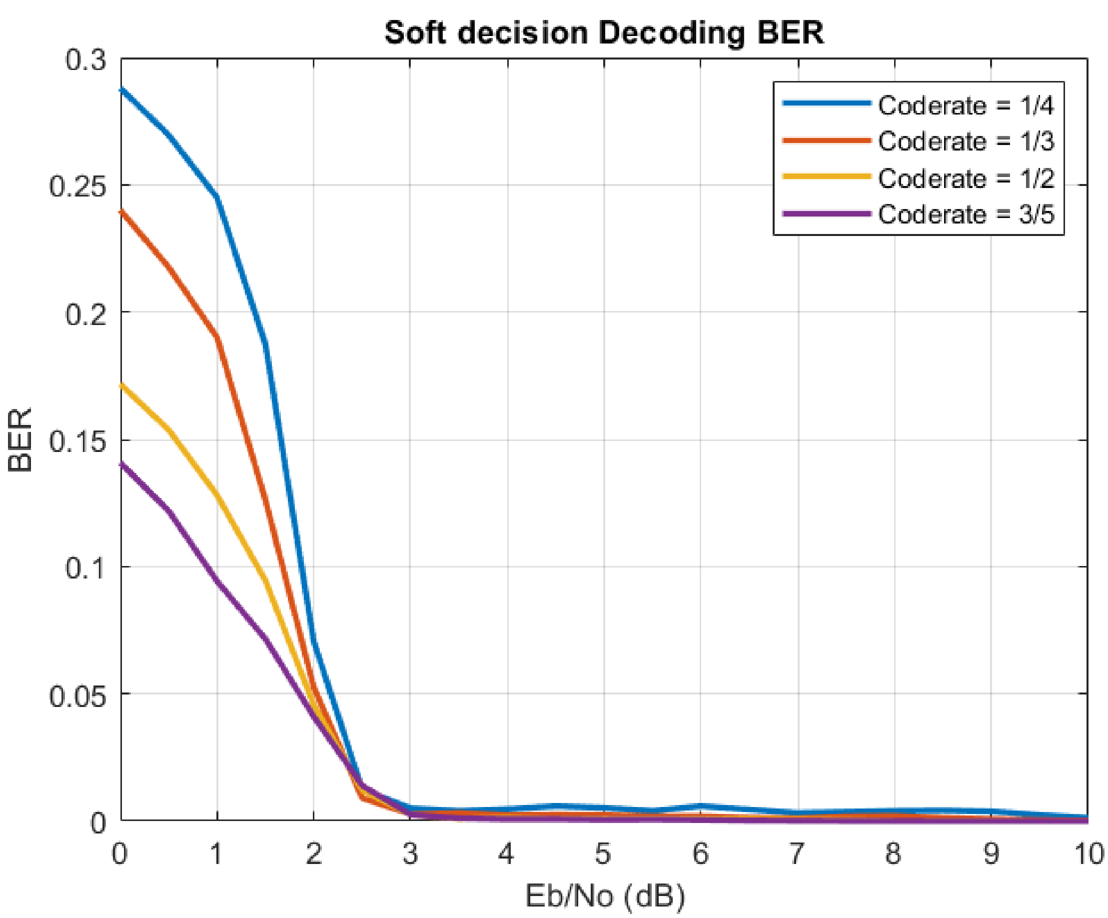


Bit Error Rate:

Hard Decision Decoding:



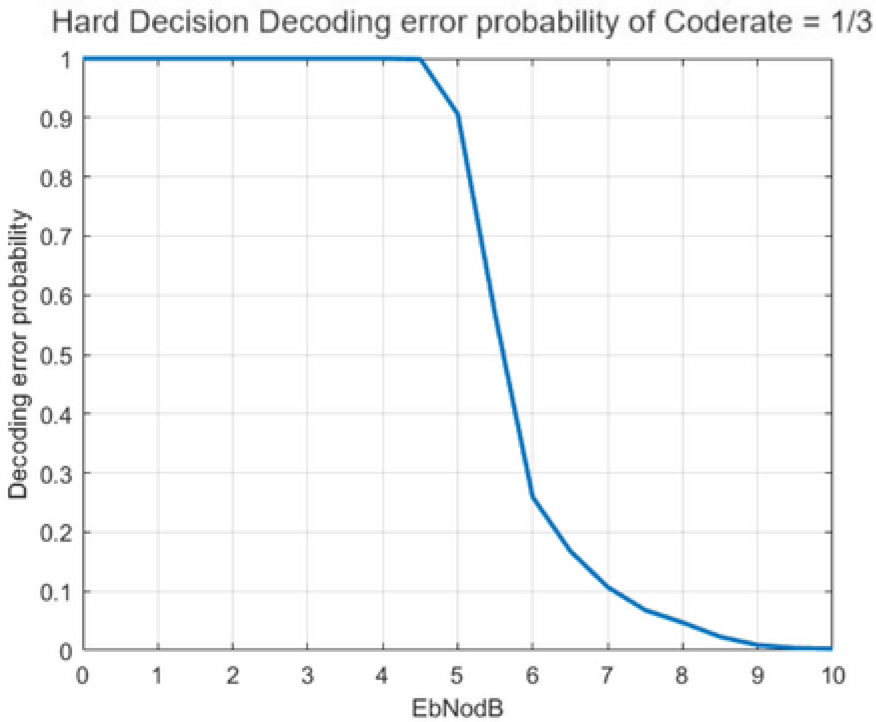
Soft Decision Decoding:

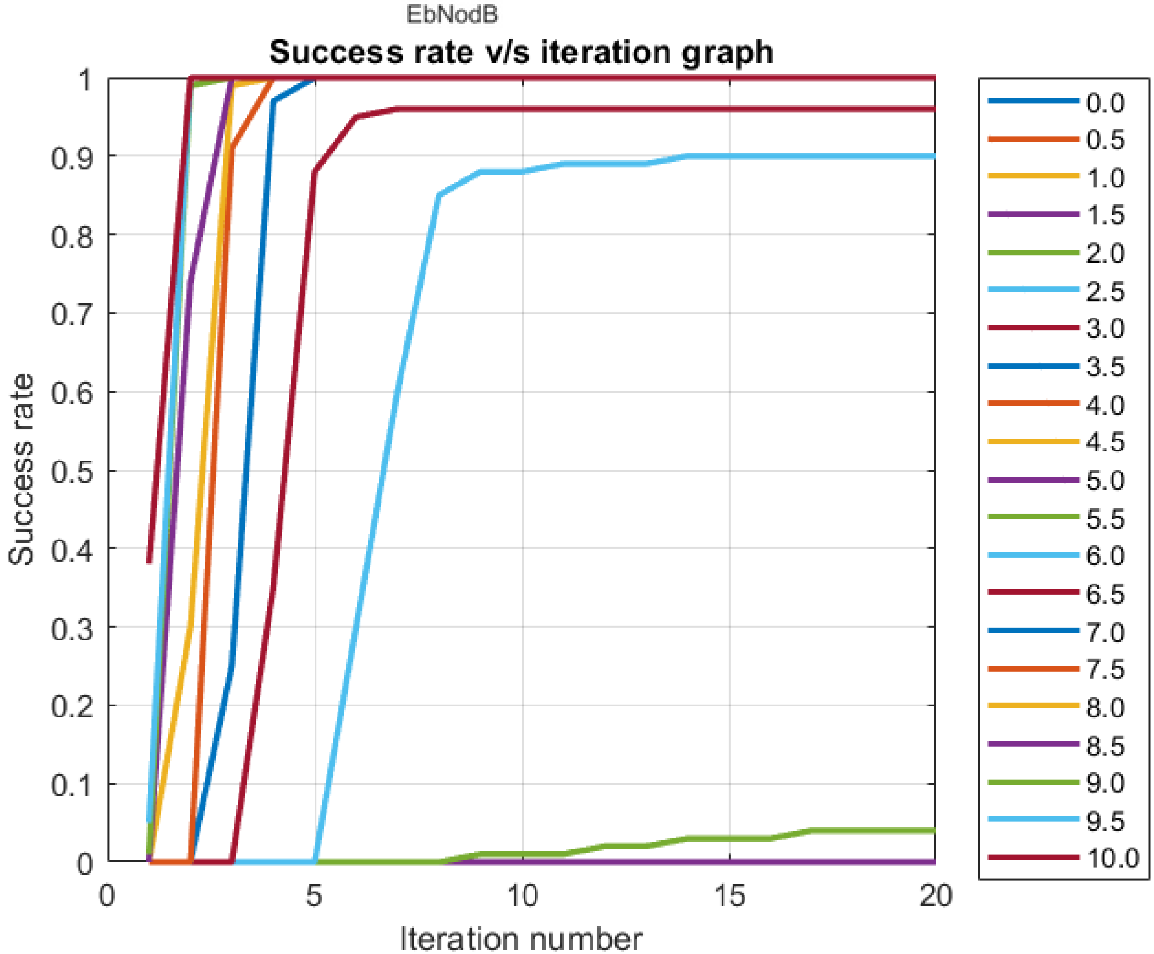


2. **For Matrix NR_1_5_352:**

a. Code rate = 1/3

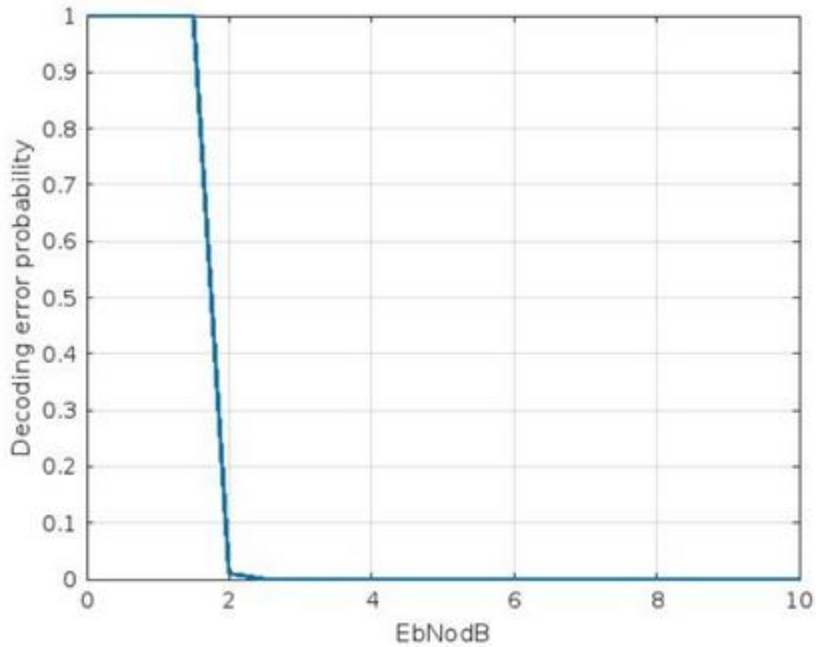
Hard Decision Decoding:

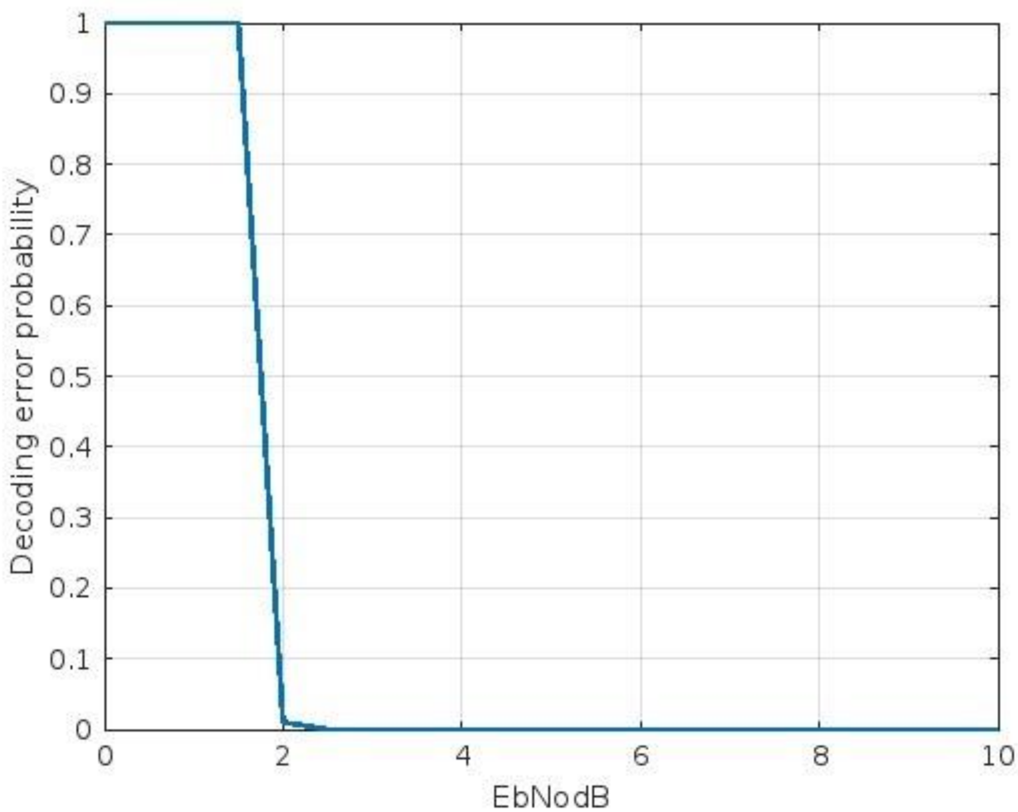




Soft Decision Decoding:

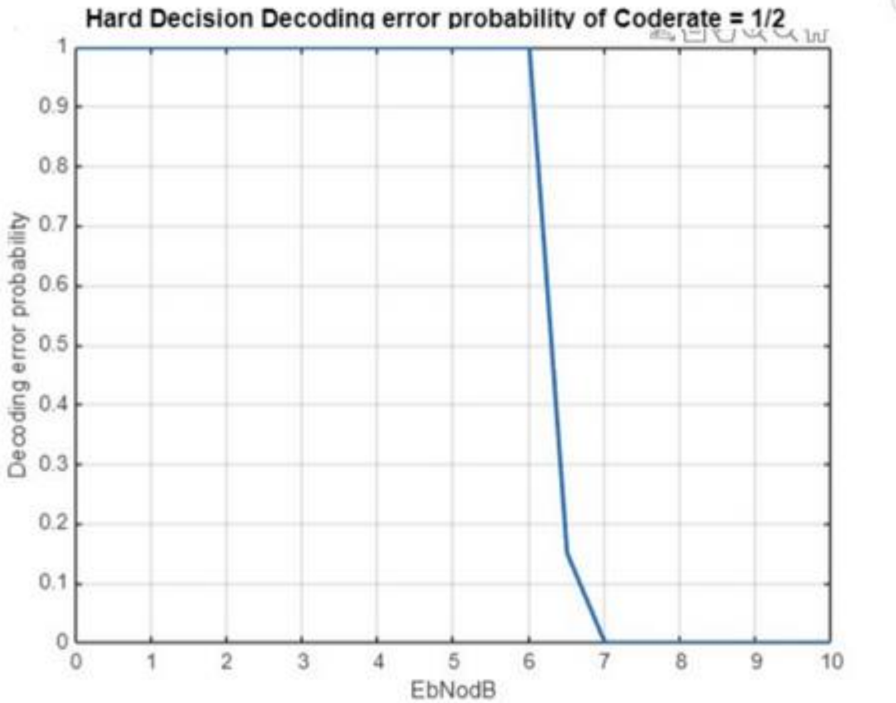
Soft Decision Decoding error probability of Coderate = 1/3

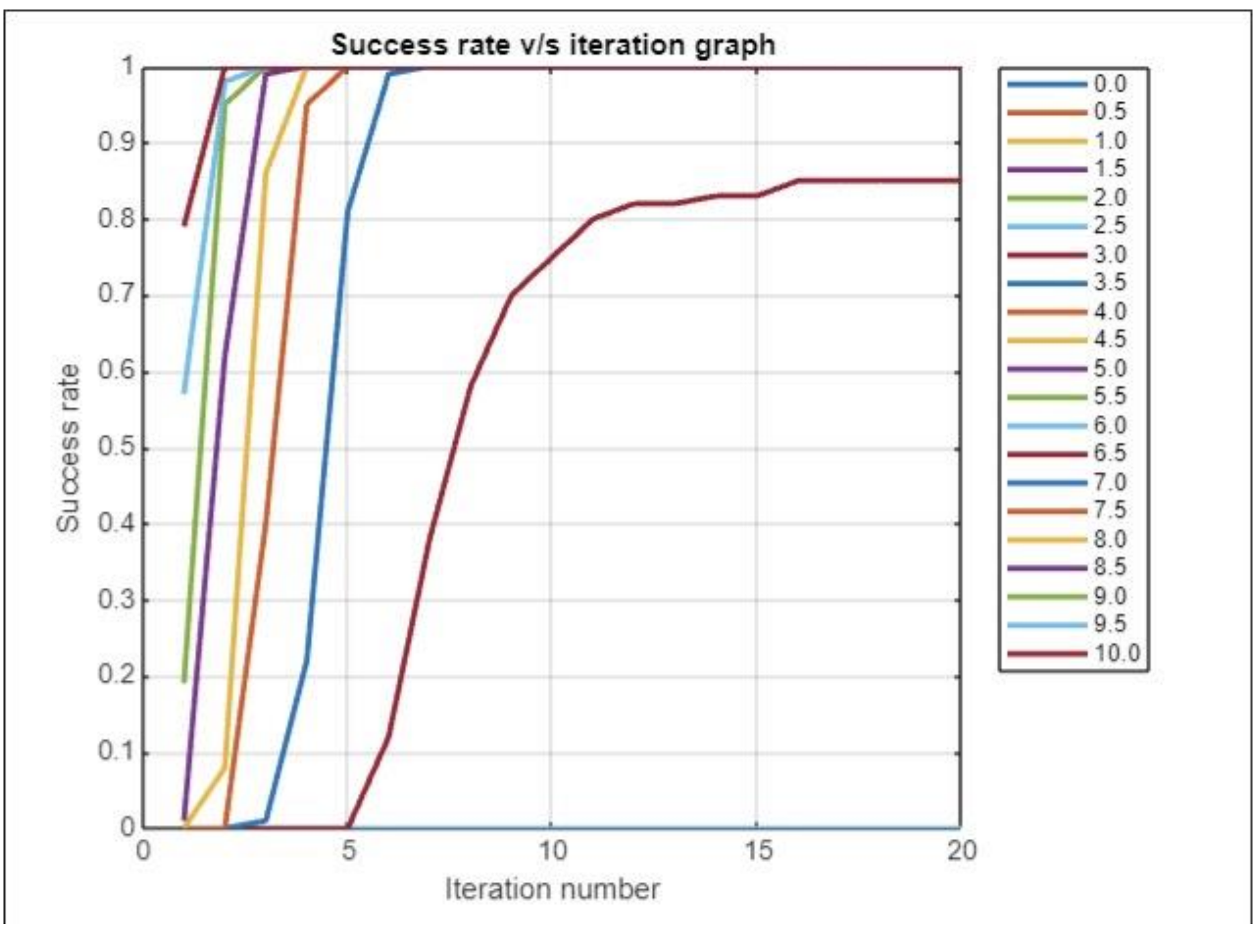




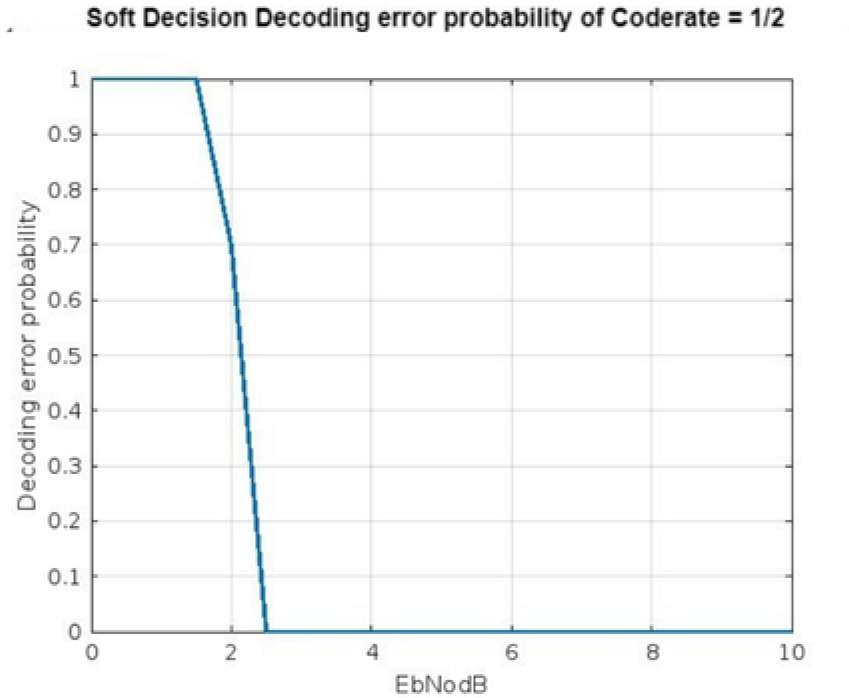
b. Code rate = 1/2

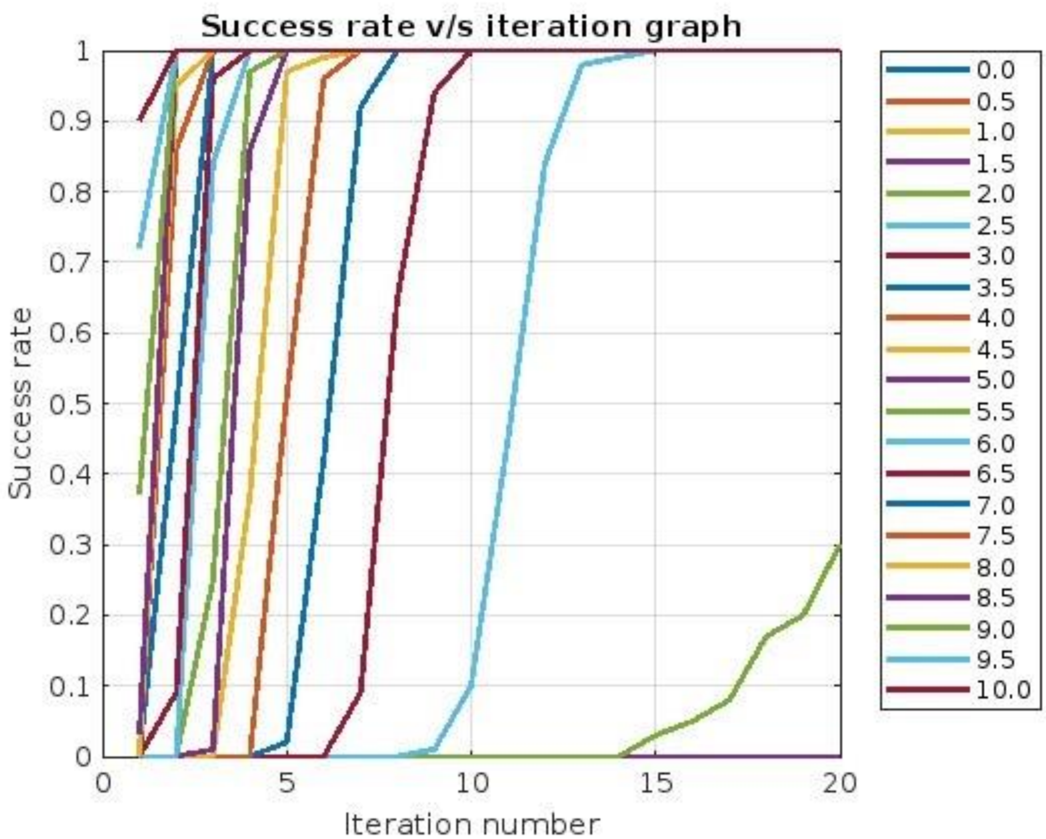
Hard Decision Decoding:





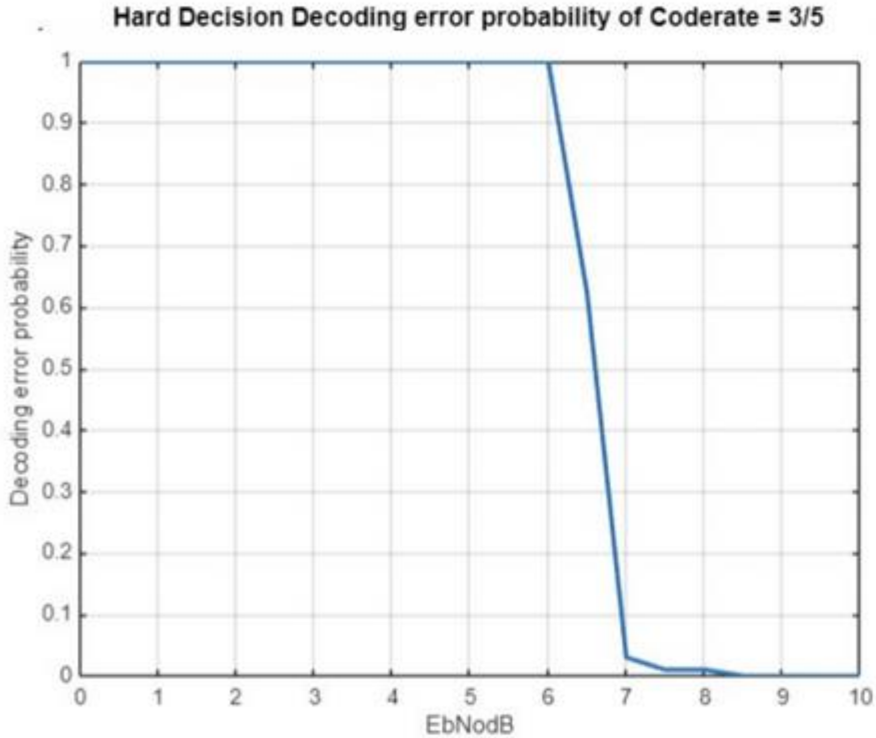
Soft Decision Decoding:

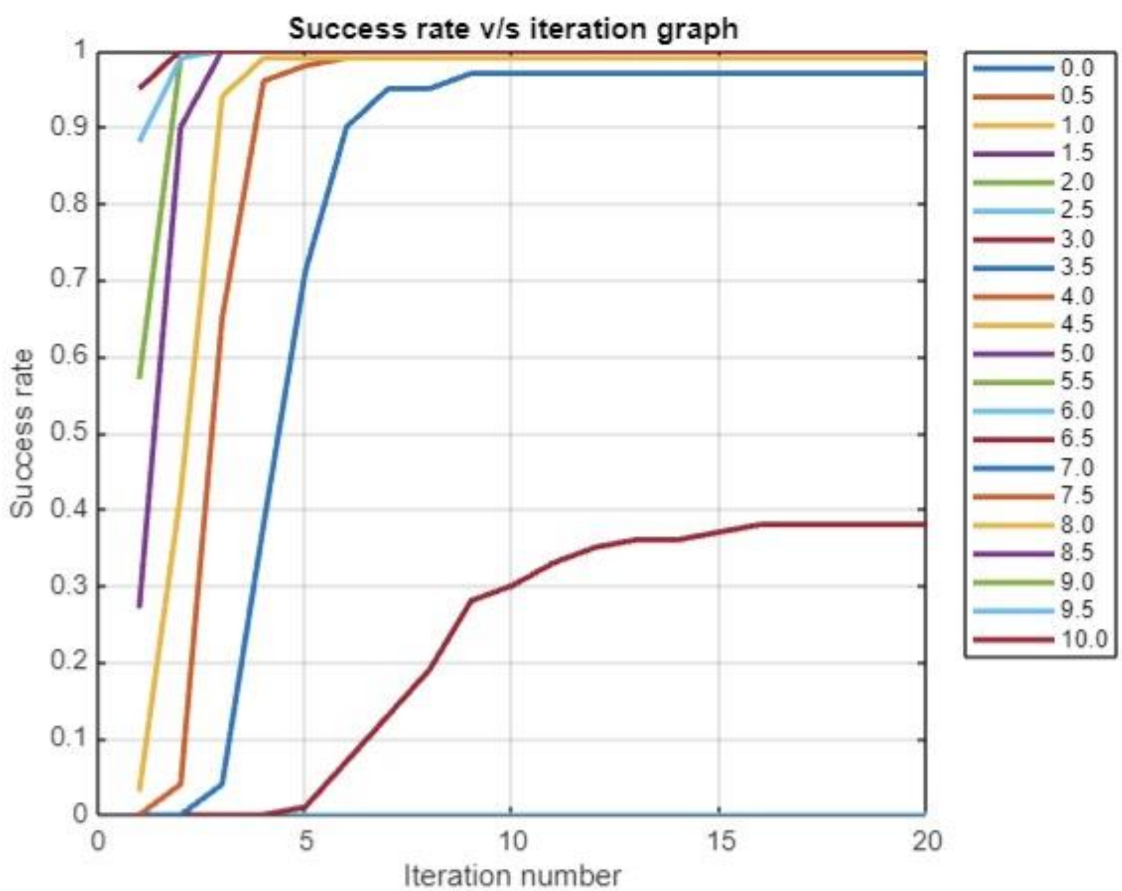




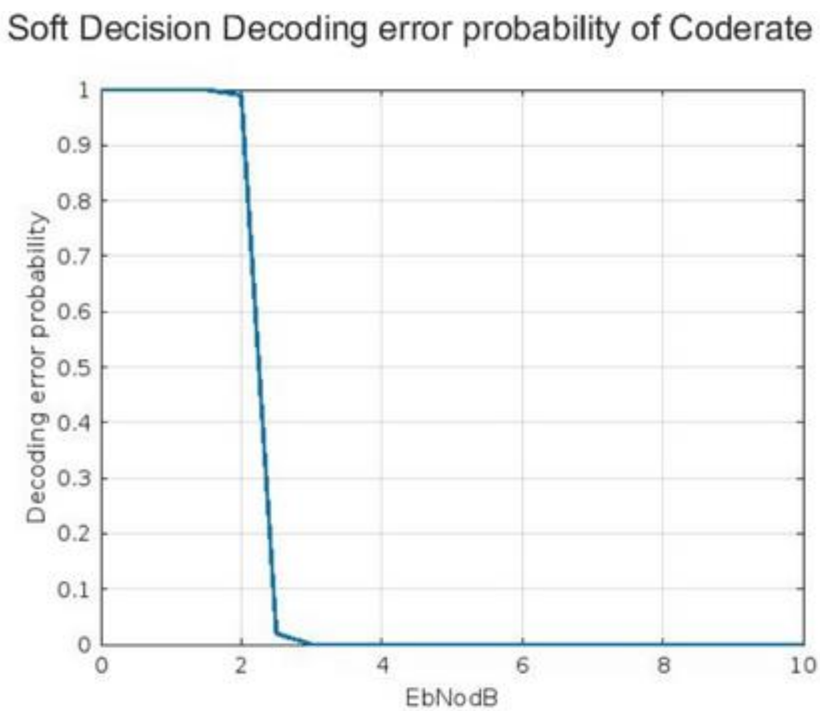
c. Code rate = 3/5

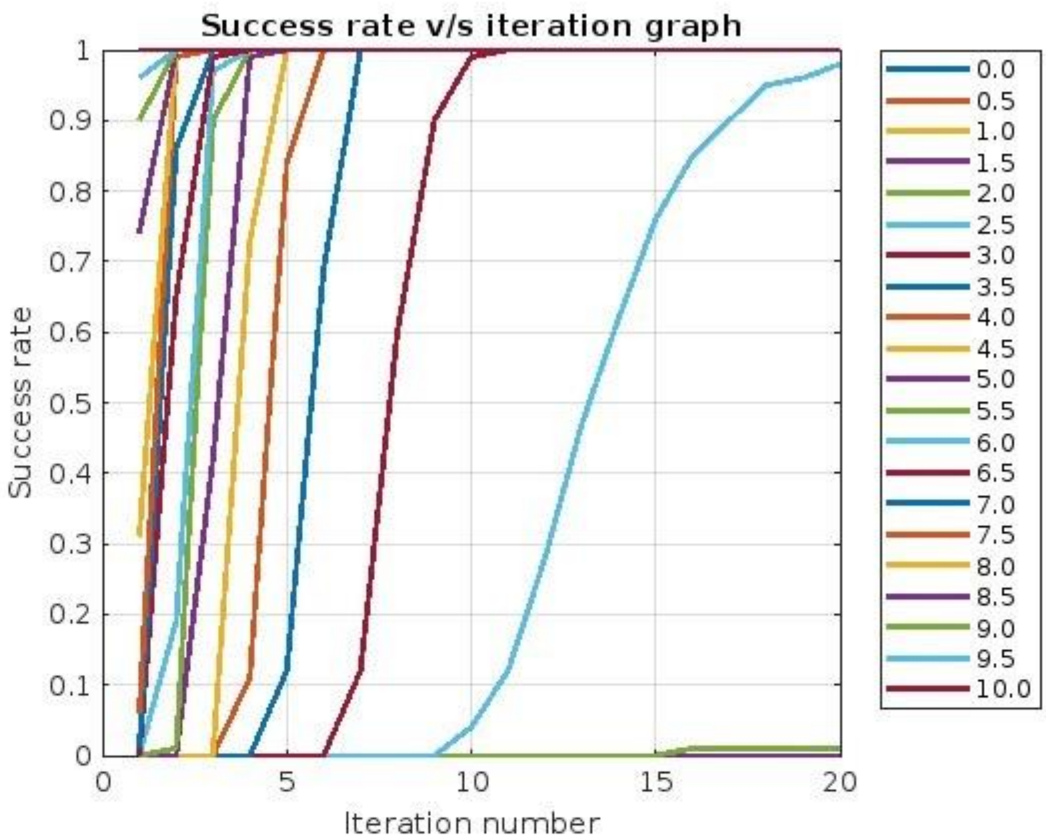
Hard Decision Decoding:





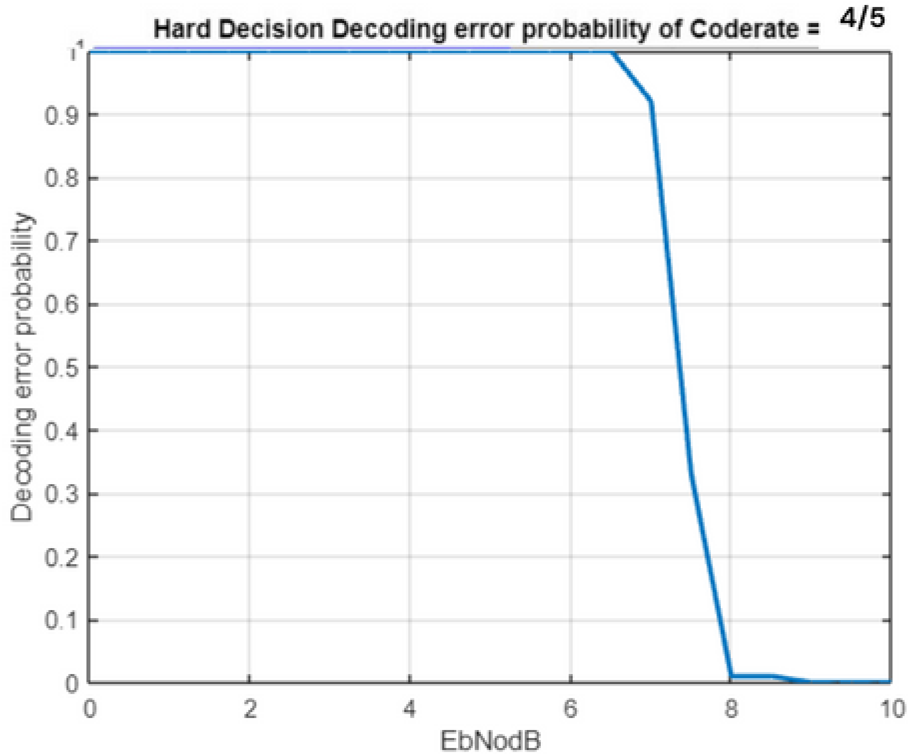
Soft Decision Decoding:

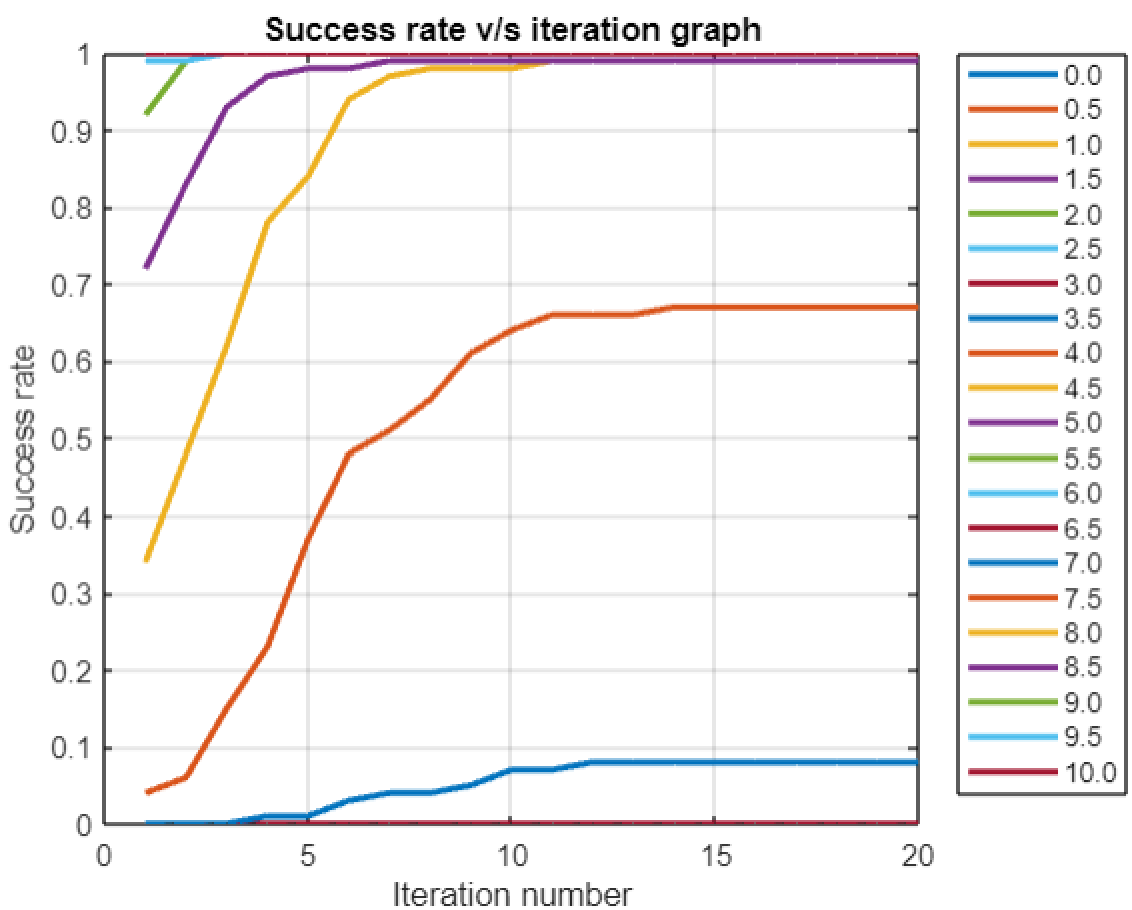




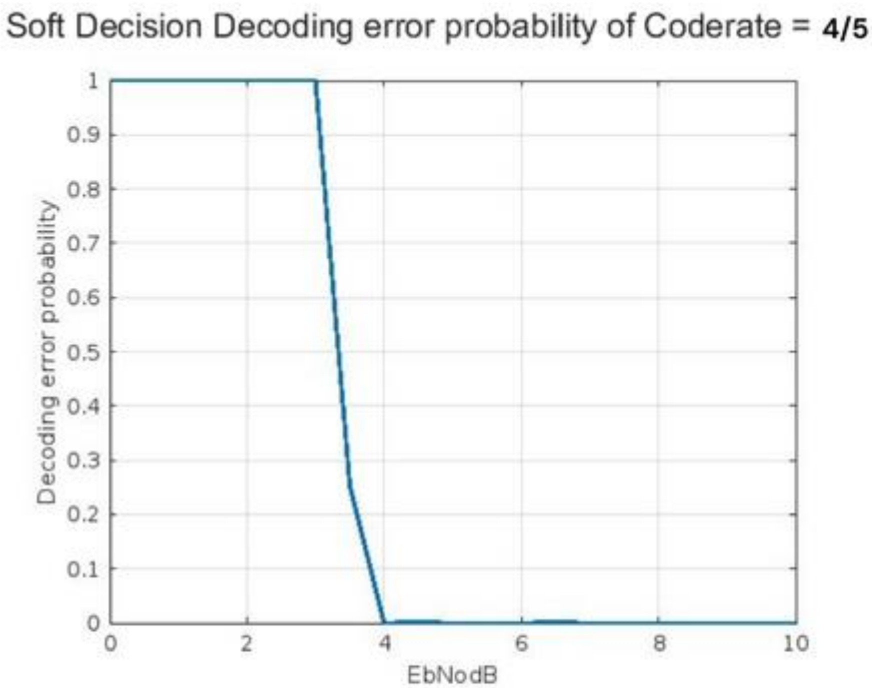
d. Code rate = 4/5

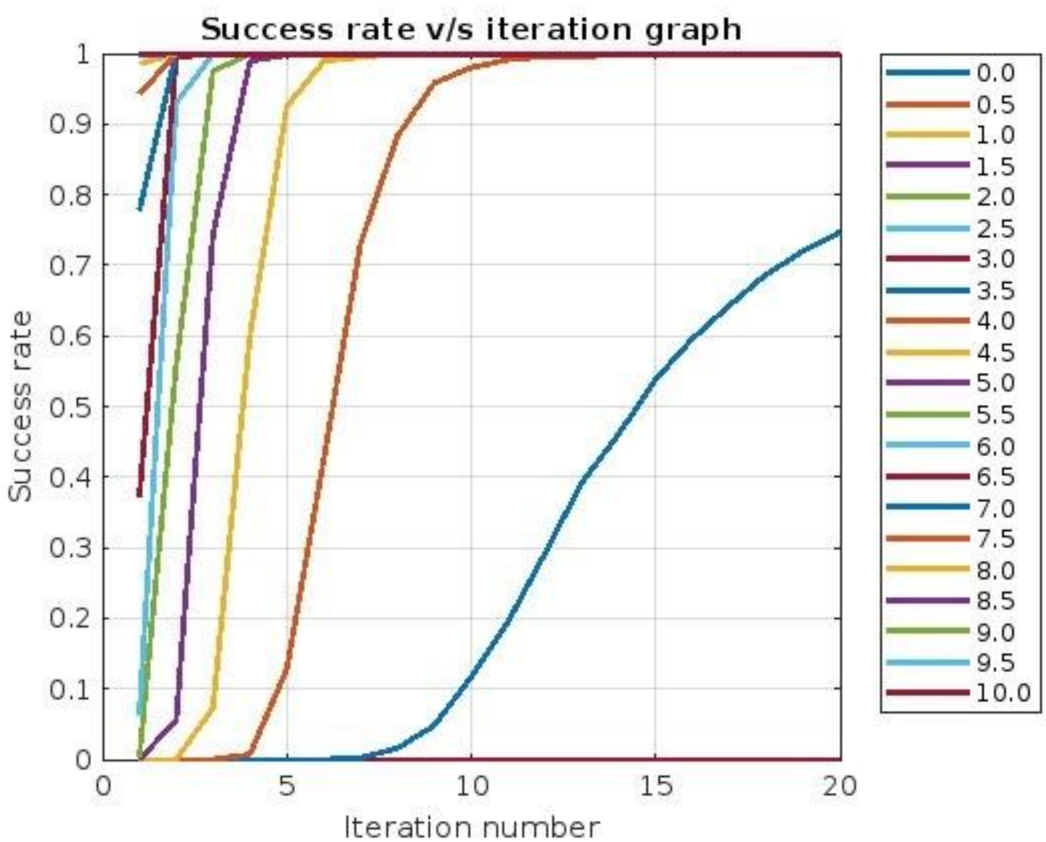
Hard Decision Decoding:





Soft Decision Decoding:



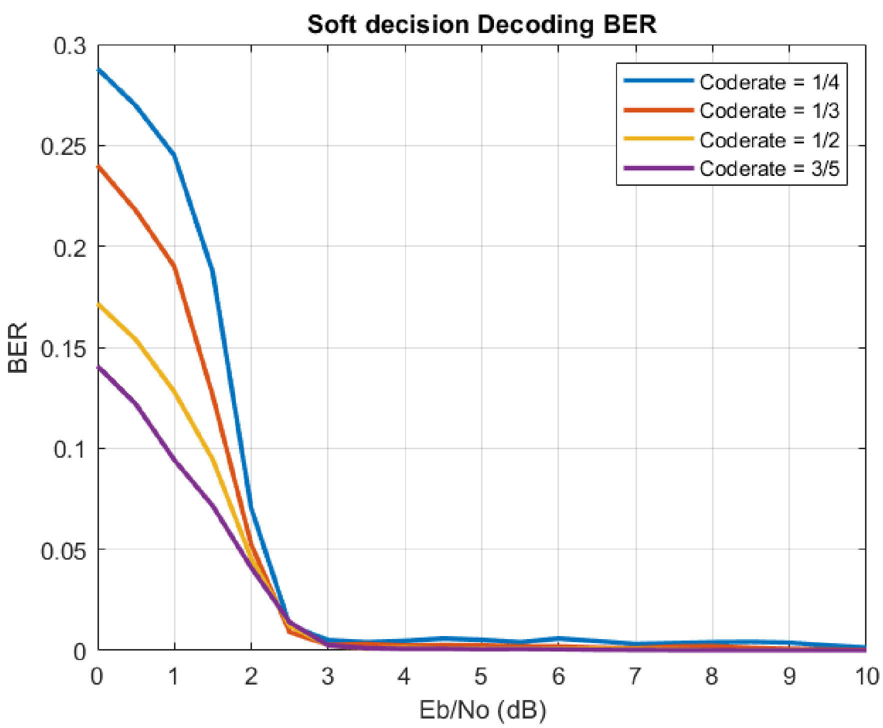


ANALYSIS:

Compare the simulation results with the Shannon channel capacity bound.

Bit Error Rate (BER) is the number of bits received in error divided by the total number of bits transferred.
According to Shannon's Bound, for Eb/No greater than equal to 2dB, the BER is approximately 1e^-5. If we perform BPSK modulation, for an error rate of 1e^-5, we need Eb/No as high as 9.5dB which is not efficient.

From the graphs we obtained, we can see that there is a significant drop in BER after 2.8 dB.



The BER becomes closer to 0 as we move ahead. We can say that the channel turns on after 2.8 dB since it efficiently decodes and the BER is also less. Thus, LDPC codes are quite close to Shannon's capacity bound.

APPENDIX A:

Derivation for results of Soft Decision Decoding:

1) Calculation of the intrinsic LLR.

$$\frac{P(c_1=0\mid r_1)}{P(c_1=1\mid r_1)} = \frac{f(r_1\mid c_1=0)}{f(r_1\mid c_1=1)}$$

$$\Rightarrow \frac{\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-(r_1-1)^2/2\sigma^2}}{\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-(r_1+1)^2/2\sigma^2}}$$

($r_i \Rightarrow$ normal distribution)

Likelihood ratio $\Rightarrow e^{2r_1/\sigma^2}$

(taking $\log_e$ on both sides)

$$\boxed{LLR = \frac{2r_1}{\sigma^2}}$$

($\frac{2}{\sigma^2}$ is a +ve factor which we have ignored in our implementation)

2) Calculation of Output LLR for repitetion codes ($n=3$)

$$L_i = \log \frac{P(c_i=0\mid r_1,r_2,r_3)}{P(c_i=1\mid r_1,r_2,r_3)}$$

$$L_1 = \log \frac{f(r_1,r_2,r_3\mid c_1=0)}{f(r_1,r_2,r_3\mid c_1=1)}$$

here $r_1 = 1+N_1(0,\sigma^2)$
$r_2 = 1+N_2(0,\sigma^2)$
$r_3 = 1+N_3(0,\sigma^2)$
where N_1, N_2 & N_3 are independent.

$$= \log\left(\frac{e^{-(r_1-1)^2/2\sigma^2}\, e^{-(r_2-1)^2/2\sigma^2}\, e^{-(r_3-1)^2/2\sigma^2}}{e^{-(r_1+1)^2/2\sigma^2}\, e^{-(r_2+1)^2/2\sigma^2}\, e^{-(r_3+1)^2/2\sigma^2}}\right)$$

$$= \log\left(e^{\frac{2}{\sigma^2}(r_1+r_2+r_3)}\right)$$

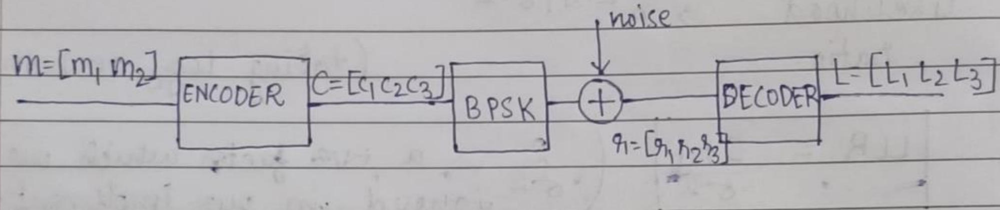
$$L_1 = \frac{2}{\sigma^2}(r_1+r_2+r_3)$$

here L_1, L_2 & L_3 will be equal, and thus
$L_i = (r_1+r_2+r_3)$ (we ignore the factor of $\frac{2}{\sigma^2}$)

3) Calculation of output LLR for Single Parity check codes.

$L_i \Rightarrow$ output LLR for bit i
$l_i \Rightarrow$ input LLR for bit i (channel LLR)
$l_{ext,i} \Rightarrow$ extrinsic LLR for bit i
$L_i = l_i + l_{ext,i}$

let us take an example of (3,2) SPC code.

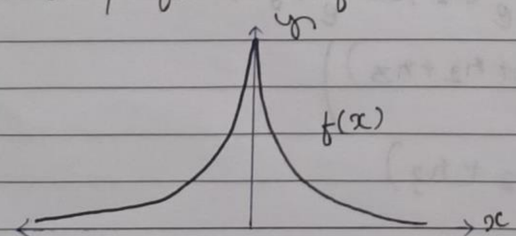


here, $L_1 = l_1 + l_{ext,1}$
$L_2 = l_2 + l_{ext,2}$
$L_3 = l_3 + l_{ext,3}$
where $l_i = \frac{2}{\sigma^2} r_i$ (assumed $\frac{2}{\sigma^2} = 1$ for implementation).

& $|l_{ext,1}| = f\big(f(l_2) + f(l_3)\big)$ $\quad sgn(l_{ext,1}) = sgn(l_2)\,sgn(l_3)$.

where $f(x) = \left| \log \tanh\left(\frac{|x|}{2}\right)\right|$

but $f(x)$ has a distinct graph which can be exploited for simplification of the decoder.



the value of $f(x)$ decreases substantially for larger values of $|x|$, thus $f(x)$ is dominated by the smallest value of $|x|$.

So we can rewrite the above formulas as:

$$|l_{ext,1}| = f(f(l_2) + f(l_3))$$

$$f(l_2) + f(l_3) \approx f(\min(|l_2|, |l_3|))$$

$$|l_{ext,1}| = f(f(\min(|l_2|, |l_3|)))$$

and $f = f^{-1}$

$$\therefore \boxed{|l_{ext,1}| = \min(|l_2|, |l_3|)}$$

this is referred as the min-sum approximation.