INTRODUCTION TO COMMUNICATION SYSTEM

CT 216 PROJECT

LDPC Decoding for 5G NR

Group - 3

Project Group – 3

Assigned By – Prof. Yash Vasavada

Mentor TA – Vansh Joshi

Honor code

- The work we are presenting is our own work and we have not copied the work (Matlab code, results, etc) that someone else has done.
- Concepts, understanding and insights we will be describing are our own.
- Wherever we have relied on an existing work that is not our own, we have provided a proper reference citation. We make this pledge truthfully, knowing that violation of this solemn pledge can carry grave consequences.

Team Members

Rudra Patel	- 202201193	
Devansh Modi	- 202201198	D.J.MODI
Zeel Boghara	- 202201201	Zeel
Meet Mahaliya	- 202201204	Mahaliya MP
Swayam Hingu	- 202201207	Swayam
Mihir Prajapati	- 202201210	
Kartavya Akabari	- 202201213	
Jeet Patel	- 202201216	J.K.Patel
Meet Sarvan	- 202201218	
Chirag Chaudhari	- 202201219	Chaudhari
Himanshu pandar	- 202201222	H.L.P.

Hard Decision Decoding

```
% load 5G NR LDPC base H matrix, use both NR_2_6_52 and NR_1_5_352
baseGraph5GNR = 'NR_2_6_52';
% baseGraph5GNR = 'NR_1_5_352';

code_rate = [1/4 1/3 1/2 3/5];
% code_rate = [1/3 1/2 3/5 4/5];

% Varing the coderate as a variable named c_r
for c_r=code_rate

    % Expand Base matrix in Binary matrix H
    [B,Hfull,z] = nrldpc_Hmatrix(baseGraph5GNR);

    [mb,nb] = size(B);

    kb = nb - mb;

    % Number of information bits or Uncoded msg length
    kNumInfoBits = kb * z ;

    k_pc = kb-2; nbRM = ceil(k_pc/c_r)+2;

    % Length of the codeword after puncturing
    nBlockLength = nbRM * z; % Number of encoded bits

    % Next three lines are some 5G NR-specific details
    H = Hfull(:,1:nBlockLength);
    nChecksNotPunctured = mb*z - nb*z + nBlockLength;

    % Binary Matrix H
    H = H(1:nChecksNotPunctured,:);

    Nchecks = size(H,1); % Number of CNs (we have denoted this as U = N - K
in the class)

    % Adjacency List of tanner graph
    VN_to_CN_map = get_VN_CN_map(H);
    CN_to_VN_map = get_CN_VN_map(H);

    [ROW , COL] = size(H);

    % Matrix for VN to CN and CN to VN values transfer
```

```
VN_2_CN_values = zeros(ROW ,COL);

CN_2_VN_values = zeros(ROW ,COL);
```

Monte-Carlo Simulation

```
EbNodB_vec = 0:0.5:10;

% Bit Error Rate For different Eb/No
BER = zeros(1,length(EbNodB_vec));

% Decoding Error Probability For different Eb/No
decoded_error = zeros(1,length(EbNodB_vec));
p=1;

% Varing the Eb/No in dB as variable named EbNodB
for EbNodB = EbNodB_vec

    EbNo = 10^(EbNodB/10);

    % Calculating Sigma which is the function of Eb/No
    sigma=sqrt(1/(2*c_r*EbNo));

    error=0;
    d_error=0;

    Nsim=1000;
    max_iteration=20;
    iteration_success = Nsim.*ones(1,max_iteration);

    for i=1:Nsim

        % Generate information (or message) bit vector
        msg = randi([0 1],[kNumInfoBits 1]);

        % Encode using 5G NR LDPC base matrix
        cward = nrldpc_encode(B,z,msg');

        cward = cward(1:nBlockLength);

        % BPSK modulation
        cward_BPSK = (1 - 2*cward);
```

```
            % Add AWGN
            noise = sigma * randn(1,nBlockLength);

            % received Message to receiver side
            received_BPSK = cward_BPSK + noise ;


            % Hard_Decision_decoding of received message
            [decoded_codeword,iteration_success]=
Hard_decoding(received_BPSK, VN_to_CN_map,
CN_to_VN_map,VN_2_CN_values,CN_2_VN_values,iteration_success);

            % check decoded codeword is same as transmitted or not
            % if not then calculate how many different bits are their

            decoded_msg=decoded_codeword(1:kNumInfoBits);
            e = mod(decoded_msg+msg',2);
            x=sum(e);


            if(x>0)
                error=error+x;
                d_error = d_error +1;
            end
        end

        EbNodB

        BER(p) = (error/(Nsim*kNumInfoBits));
        decoded_error(p) = (d_error/(Nsim));
        p=p+1;

        plot(1:max_iteration,iteration_success,'LineWidth',2);
        xlabel('Iteration number');
        ylabel('Success rate');
        title('Success rate vs Iteration number');
        grid on;
        hold on;

    end

    % Plot Decoding Error Probability vs Eb/No graph for every code rate
that we use
    hold off;
    plot(EbNodB_vec,decoded_error,'LineWidth',2);
    xlabel('Eb/No (dB)');
```

```
    ylabel('Decoding Error Probability');
    title('Decoding Error Probability vs Eb/No');
    legend('r = 1/4', 'r = 1/3' , 'r = 1/2' , 'r =
3/5','Location','bestoutside');

    disp(c_r);
    hold on;
end
```

Function that we use in our Program

Functions For Hard Decision Decoding

Hard_decoding: For Hard Decision Decoding

```
function [prev_received,iteration_success] =
Hard_decoding(received_BPSK,VN_to_CN_map,
CN_to_VN_map,VN_2_CN_values,CN_2_VN_values,iteration_success)

    received_msg = (received_BPSK<0);
    prev_received=received_msg;

    max_iteration = 20;

    for i=1:max_iteration

        % Each VN send appropriate values to their connected CNs
        if(i==1)

VN_2_CN_values=VN_to_CN_first(VN_to_CN_map,VN_2_CN_values,received_msg);
        else

VN_2_CN_values=VN_to_CN_transfer(VN_to_CN_map,VN_2_CN_values,CN_2_VN_values,
received_msg);
        end

        % Each CN send appropriate values to their connected VNs

CN_2_VN_values=CN_to_VN_transfer(CN_to_VN_map,VN_2_CN_values,CN_2_VN_values)
;

        % Estimate the codeword
        c_cap = c_hat(VN_to_CN_map,CN_2_VN_values, received_msg);
```

```
        % Break if estimated codeword is same as previous estimated
        if(sum(xor(c_cap,prev_received))==0)
            break;
        end

        iteration_success(i)=iteration_success(i)-1;
        prev_received = c_cap;

    end
end
```

VN_to_CN_first: Each VN send appropriate (repetition code) values to their connected CNs in first iteration

```
function VN_2_CN=VN_to_CN_first(VN_to_CN_map,VN_2_CN_values,received_msg)

    number_of_vn=size(VN_to_CN_map,1);

    % traverse each vn
    for vn=1:number_of_vn
        ith_CN_list=VN_to_CN_map{vn};

        % transfer own value to connected cns
        for cn=ith_CN_list
            VN_2_CN_values(cn,vn)=received_msg(vn);
        end
    end
    VN_2_CN=VN_2_CN_values;
end
```

VN_to_CN_transfer: Each VN send appropriate (repetition code) values to their connected CNs

```
function
VN_2_CN=VN_to_CN_transfer(VN_to_CN_map,VN_2_CN_values,CN_2_VN_values,
received_msg)

    number_of_vn = size(VN_to_CN_map,1);

    % traverse each vn
    for vn = 1:number_of_vn
        ith_CN_list = VN_to_CN_map{vn};

        dv = length(ith_CN_list);
```

```
        % first take sum of all values that connected cns gives
        n_of_one =received_msg(vn);

        for cn=ith_CN_list
            if(CN_2_VN_values(cn,vn)==1)
                n_of_one = n_of_one + 1;
            end
        end

        % transfer appropriate value to connected cns
        for cn = ith_CN_list

              % Give value to ith CN after subtract value of ith CN from
              % sum of all CNs
              VN_2_CN_values(cn,vn) = (n_of_one - CN_2_VN_values(cn,vn)) >
(dv/2);
        end

    end
    VN_2_CN=VN_2_CN_values;

 end
```

CN_to_VN_transfer: Each CN send appropriate (SPC code) values to their connected VNs

```
 function CN_2_VN =
CN_to_VN_transfer(CN_to_VN_map,VN_2_CN_values,CN_2_VN_values)

    number_of_cn = size(CN_to_VN_map,1);

    % traverse each CN
    for cn = 1:number_of_cn
        ith_VN_list = CN_to_VN_map{cn};

        % First We take Xor Of all connected VNs
        xor_all=0;

        for vn=ith_VN_list
            xor_all=xor(xor_all,VN_2_CN_values(cn,vn));
        end

        % transfer appropriate value to connected cns
        for vn = ith_VN_list
```

```
            % By using xor property : c=xor(a,b) then a=xor(b,c)
            CN_2_VN_values(cn,vn) =xor(xor_all , VN_2_CN_values(cn,vn));
        end

    end
    CN_2_VN=CN_2_VN_values;

end
```

c_hat: Estimate codeword after each iteration

```
function Estimated_codeword = c_hat(VN_to_CN_map,CN_2_VN_values,
received_msg)

    number_of_vn = size(VN_to_CN_map,1);
    Estimated_codeword = zeros(1,number_of_vn);

    itr=1;
    for vn = 1:number_of_vn

        s = received_msg(vn);

        dv = length(VN_to_CN_map{vn});

        for cn = VN_to_CN_map{vn}
            if(CN_2_VN_values(cn,vn)==1)
                s = s + 1;
            end
        end

        Estimated_codeword(itr)= (s>((1+dv)/2));
        itr=itr+1;
    end

end
```

Other functions that we use in our program

get_CN_VN_map: Generate CN to VN adjacency List

```
function output = get_CN_VN_map(H)
    [row,col]=size(H);
    output = cell(row, 1);
```

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

get_VN_CN_map: Generate VN to CN adjacency List

```
function output = get_VN_CN_map(H)
    [row , col]=size(H);
    output = cell(col, 1);

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

nrldpc_encode: Get Encoded Codeword form msg

```
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
```

mul_sh: Multiply codeword with Zero , identity or Shifted Identity Matrix

```
function y=mul_sh(x,k)
%x=inout blocks
%k= -1 or shifts
%y = output

    if(k==-1)
```

```
        y = zeros(1,length(x));
    else
        y = [x(k+1:end) x(1:k)];  %multiplication by shift identity
    end

end
```

nrldpc_Hmatrix: Base matrix Expantion

```
function [B,H,z] = nrldpc_Hmatrix(BG)

load(sprintf('%s.txt',BG),BG);
B = NR_2_6_52;
[mb,nb] = size(B);
z = 52;
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;
P = H(:,1:K);
G = [eye(K); P];
Z = H*G;
end
```

Derivation of the result of soft-decision decoding (min-sum approximation algo)

❖ Log Likelihood Ratio:

$$P(C_i = 1|r_i) = \frac{P(r_i|\, C_i = 1)P(C_i = 1)}{P(r_i)}$$

$$P(C_i = 0|r_i) = \frac{P(r_i|\, C_i = 0)P(C_i = 0)}{P(r_i)}$$

$$\lambda_i = \frac{P(C_i = 1|r_i)}{P(C_i = 0|r_i)} = \frac{P(r_i|\, C_i = 1)}{P(r_i|\, C_i = 0)}$$

$$\lambda_i = \frac{\left(\frac{1}{\sqrt{2\pi\sigma}}\right)\left(e^{-\frac{(r_i+1)^2}{2\,\sigma^2}}\right)}{\left(\frac{1}{\sqrt{2\pi\sigma}}\right)\left(e^{-\frac{(r_i-1)^2}{2\,\sigma^2}}\right)}$$

$$\lambda_i = e^{\frac{2r_i}{\sigma^2}}$$

$$\iota_i = \ln(\lambda_i) = \frac{2r_i}{\sigma^2}$$

This is, intrinsic LLR

For, simplicity in our implementation we will ignore $\frac{2}{\sigma^2}$ because it is just a positive factor.

❖ For repetition:

$$l_i = \frac{P(C_i = 1\,|\, r_1, r_2, \ldots\ldots, r_n)}{P(C_i = 0\,|\, r_1, r_2, \ldots\ldots, r_n)} = \frac{P(\, r_1, r_2, \ldots\ldots, r_n\,|\, C_i = 1)}{P(\, r_1, r_2, \ldots\ldots, r_n|\, C_i = 0)}$$

$$l_i = \frac{P(r_1|\, C_1 = 1)\, P(r_2|\, C_2 = 1) \ldots P(r_n|\, C_n = 1)}{P(r_1|\, C_1 = 0)\, P(r_2|\, C_2 = 0) \ldots P(r_n|\, C_n = 0)}$$

(Because all r 1 … independent from each other)

$$= \frac{\left(e^{-\frac{(r_1+1)^2}{2\,\sigma^2}}\right)}{\left(e^{-\frac{(r_1-1)^2}{2\,\sigma^2}}\right)} \frac{\left(e^{-\frac{(r_2+1)^2}{2\,\sigma^2}}\right)}{\left(e^{-\frac{(r_2-1)^2}{2\,\sigma^2}}\right)} \frac{\left(e^{-\frac{(r_3+1)^2}{2\,\sigma^2}}\right)}{\left(e^{-\frac{(r_3-1)^2}{2\,\sigma^2}}\right)} \qquad \left(\because \quad \lambda = \frac{\left(\frac{1}{\sqrt{2\pi\sigma}}\right)\left(e^{-\frac{(r_i-1)^2}{2\,\sigma^2}}\right)}{\left(\frac{1}{\sqrt{2\pi\sigma}}\right)\left(e^{-\frac{(r_i+1)^2}{2\,\sigma^2}}\right)} \right)$$

$$= e^{\frac{2\,r_1}{\sigma^2}}\, e^{\frac{2\,r_2}{\sigma^2}}\, e^{\frac{2\,r_3}{\sigma^2}}$$

$L_i = \ r_1 + r_2 + \ldots\ldots + r_n$ we will ignore $2/\sigma^2 \ factor$

- **SPC (n, n-1) code:**

Example of (3,2) SPC code:

C1	C2	C3
0	0	0
0	1	1
1	0	1
1	1	0

$c1 = \ c2 \ \oplus \ c3$

Where Pi = (Ci=1)

$$P1 \ = \ P2(1 - P3) \ + \ P3(1 - P2) \ \ldots\ldots. (1)$$

$$(1 - P1) = \ P2P3 \ + \ (1 - P2)(1 - P3) \ \ldots\ldots. (2)$$

Now Subtracting Equations (1-2)

$$P1 - (1 - P1) \ = \ P2(P3(1 - p3)) - (1 - \ P2) \ (P3(1 - p3))$$

$$P1 - (1 - P1) \ = \ (P2 - (1 - \ P2)) \ (P3 - (1 - p3))$$

$$\frac{P1 - (1 - P1)}{P1 \ + \ (1 - P1)} = \frac{(P2 - (1 - \ P2))}{P2 + (1 - P2)} \frac{(P3 - (1 - p3))}{P3 + (1 - P3)}$$

$$\frac{1 - \frac{(1 - P1)}{P1}}{1 \ + \ \frac{(1 - P1)}{P1}} = \frac{1 - \frac{(1 - P2)}{P2}}{1 \ + \ \frac{(1 - P2)}{P2}} \frac{1 - \frac{(1 - P3)}{P3}}{1 \ + \ \frac{(1 - P3)}{P3}}$$

$$\frac{1 - e^{-l_{ext,1}}}{1 + e^{-l_{ext,1}}} \ = \ \frac{1 - e^{-l_2}}{1 + e^{-l_2}} \frac{1 - e^{-l_3}}{1 + e^{-l_3}}$$

$$\tanh(x) \ = \ \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

$$\tanh\left(\frac{l_{ext,1}}{2}\right) \ = \ \tanh\left(\frac{l_2}{2}\right) \ \tanh\left(\frac{l_3}{2}\right)$$

$l_{ext,1}$ can be written in two parts Magnitude and sign,

$$sgn(l_{ext,1}) = sgn(l_2)\, sgn(l_3)$$

$$\tanh\left(\frac{|l_{ext,1}|}{2}\right) = \tanh\left(\frac{|l_2|}{2}\right) \tanh\left(\frac{|l_3|}{2}\right)$$

$$\log\left(\tanh\left(\frac{|l_{ext,1}|}{2}\right)\right) = \log\left(\tanh\left(\frac{|l_2|}{2}\right)\right) + \log\left(\tanh\left(\frac{|l_3|}{2}\right)\right)$$

Now, $f(x)$ is

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

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

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

$f(|l_2|) + f(|l_3|) \approx f(min(|l_2|, |l_3|))$ because of the characteristics of the f function.

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

Now, f is the inverse of its own

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

$$l_{ext,1} = sgn(l_{ext,1}) * |l_{ext,1}|$$

Similarly for n

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

$$sgn(l_{ext,1}) = sgn(l_2)\, sgn(l_3) \ldots\ldots sgn(l_n)$$

$$l_{ext,1} = sgn(l_{ext,1}) * |l_{ext,1}|$$

This is the return value after the min operation.

Soft Decision Decoding

```
% load 5G NR LDPC base H matrix, use both NR_2_6_52 and NR_1_5_352
baseGraph5GNR = 'NR_2_6_52';
% baseGraph5GNR = 'NR_1_5_352';

code_rate = [1/4 1/3 1/2 3/5];
% code_rate = [1/3 1/2 3/5 4/5];

EbNodB_vec = 0:0.5:10;

% Varing the coderate as a variable named c_r
for c_r = code_rate

    % Expand Base matrix in Binary matrix H
    [B,Hfull,z] = nrldpc_Hmatrix(baseGraph5GNR);

    [mb,nb] = size(B);

    kb = nb - mb;

    % Number of information bits or Uncoded msg length
    kNumInfoBits = kb * z

    k_pc = kb-2; nbRM = ceil(k_pc/c_r)+2;

    % Length of the codeword after puncturing
    nBlockLength = nbRM * z; % Number of encoded bits

    % Next three lines are some 5G NR specific details
    H = Hfull(:,1:nBlockLength);
    nChecksNotPunctured = mb*z - nb*z + nBlockLength;

    % Binary Matrix H
    H = H(1:nChecksNotPunctured,:);

    Nchecks = size(H,1); % Number of CNs (we have denoted this as U = N - K
in the class)

    % Adjacency List of tanner graph
    VN_to_CN_map = get_VN_CN_map(H);
    CN_to_VN_map = get_CN_VN_map(H);

    [ROW , COL] = size(H);
```

```
% Matrix for VN to CN and CN to Vn values transfer
VN_2_CN_values = zeros(ROW ,COL);
CN_2_VN_values = zeros(ROW ,COL);
```

Monte-Carlo Simulation

```
EbNodB_vec = 0:0.5:10;

% Bit Error Rate For different Eb/No
prac_bit_error = zeros(1,length(EbNodB_vec));

% Decoding Error Probability For different Eb/No
prac_decoded_error = zeros(1,length(EbNodB_vec));
p=1;

% Varing the Eb/No in dB as a variable named EbNodB
for EbNodB = EbNodB_vec

    EbNo = 10^(EbNodB/10);

    % Calculating Sigma which is the function of Eb/No
    sigma=sqrt(1/(2*c_r*EbNo));

    bit_error=0;
    decoded_error=0;

    Nsim=100;
    max_iteration=20;
    iteration_success = Nsim.*ones(1,max_iteration);

    for i=1:Nsim

        % Generate information (or message) bit vector
        msg = randi([0 1],[kNumInfoBits 1]);

        % Encode using 5G NR LDPC base matrix
        cward = nrldpc_encode(B,z,msg');

        cward = cward(1:nBlockLength);

        % BPSK modulation
        cward_BPSK = (1 - 2*cward);

        % Add AWGN
        noise = sigma * randn(1,nBlockLength);
```

```
            % received Message to receiver side
            received_BPSK = cward_BPSK + noise ;

            % Soft_Decision_decoding of received message
            [received,iteration_success] =
soft_decoding(received_BPSK,sigma, VN_to_CN_map,
CN_to_VN_map,VN_2_CN_values,CN_2_VN_values,iteration_success);

            % check decoded codeword is same as transmitted or not
            % if not then calculate how many different bits are their
            received=received<0;

            decoded_msg=received(1:kNumInfoBits);
            e = mod(decoded_msg+msg',2);
            x=sum(e);
            if(x>0)
                decoded_error=decoded_error+1;
                bit_error=bit_error+x;
            end
        end

        prac_bit_error(p) = (bit_error/(Nsim*kNumInfoBits));
        prac_decoded_error(p) = (decoded_error/Nsim);
        p=p+1;

        plot(1:max_iteration,iteration_success,'LineWidth',2);
        xlabel('Iteration number');
        ylabel('Success rate');
        title('Success rate vs Iteration number');
        grid on;
        hold on;

    end

    % Plot Decoding Error Probability vs Eb/No graph for every code rate
that we use

    plot(EbNodB_vec,prac_decoded_error,'LineWidth',2);
    xlabel('Eb/No (dB)');
    ylabel('Decoding Error Probability');
    title('Decoding Error Probability vs Eb/No');
    legend('r = 1/4', 'r = 1/3' , 'r = 1/2' , 'r =
3/5','Location','bestoutside');

    % disp(c_r);
```

```
    hold on;

end
```

Function That we used in our Programme

Functions for Soft Decision Decoding

Soft_decision_decoding: For Soft Decision Decoding using the MIN-SUM algorithm

```
function [prev_L,iteration_success] =
soft_decoding(received_BPSK,sigma,VN_to_CN_map,
CN_to_VN_map,VN_2_CN_values,CN_2_VN_values,iteration_success)

    received_01 = received_BPSK < 0;

    if(sigma==0)
        L=received_BPSK;
    else
    L=((2*received_BPSK)/(sigma^2));
    end

    prev_L = L;
    max_iteration = 20;

    for i=1:max_iteration

        % Each VN send appropriate values to their connected CNs
        if(i==1)
            VN_2_CN_values=VN_to_CN_first(VN_to_CN_map,VN_2_CN_values,L);
        else

VN_2_CN_values=VN_to_CN_transfer(VN_to_CN_map,VN_2_CN_values,CN_2_VN_values,
L);
        end

        % Each CN send appropriate values to their connected VNs

CN_2_VN_values=CN_to_VN_transfer(CN_to_VN_map,VN_2_CN_values,CN_2_VN_values)
;

        % Estimate the codeword
        c_cap = c_hat(VN_to_CN_map,CN_2_VN_values, L);
```

```
        % Break if estimated codeword same as previous estimated

        c_cap_01 = c_cap < 0;
        prev_L_01 = prev_L < 0;
        if(sum(xor(c_cap_01,prev_L_01))==0)
            break;
        end
        iteration_success(i)=iteration_success(i)-1;
        prev_L = c_cap;
    end
end
```

VN_to_CN_first: Each VN send appropriate (repetition code) values to their connected CNs in first iteration

```
function VN_2_CN=VN_to_CN_first(VN_to_CN_map,VN_2_CN_values,received_L)

    number_of_vn=size(VN_to_CN_map,1);

    % traverse each vn
    for vn=1:number_of_vn
        ith_CN_list=VN_to_CN_map{vn};

        % transfer own value to connected cns
        for cn=ith_CN_list
            VN_2_CN_values(cn,vn)=received_L(vn);
        end
    end

    VN_2_CN=VN_2_CN_values;
end
```

VN_to_CN_transfer: Each VN send appropriate values to their connected CNs

```
function
VN_2_CN=VN_to_CN_transfer(VN_to_CN_map,VN_2_CN_values,CN_2_VN_values,
received_L)

    number_of_vn = size(VN_to_CN_map,1);

    % traverse each vn
    for vn = 1:number_of_vn
        ith_CN_list = VN_to_CN_map{vn};
```

```
        % dv = length(ith_CN_list);

        % first take sum of all values that connected cns gives
        sum_of_L = received_L(vn);

        for cn=ith_CN_list
                sum_of_L = sum_of_L + CN_2_VN_values(cn,vn);
        end

        % transfer appropriate value to connected cns
        for cn = ith_CN_list

              % Give value to ith CN after subtract value of ith CN from
              % sum of all CNs
              VN_2_CN_values(cn,vn) = (sum_of_L - CN_2_VN_values(cn,vn));
        end

    end
    VN_2_CN=VN_2_CN_values;

end
```

CN_to_VN_transfer: Each CN send appropriate (do SPC code) values to their connected VNs

```
 function
CN_2_VN=CN_to_VN_transfer(CN_to_VN_map,VN_2_CN_values,CN_2_VN_values)

    number_of_cn = size(CN_to_VN_map,1);

    % traverse each CN
    for cn = 1:number_of_cn
        ith_VN_list = CN_to_VN_map{cn};

        % First We find abs minimum
        mini1 = 1e10;
        mini2 = 1e10;

        sgn = 1;
        for vn=ith_VN_list

            sgn = sgn*sign(VN_2_CN_values(cn,vn));
            value =  abs(VN_2_CN_values(cn,vn));

            if (value<mini1)
```

```
                mini2=mini1;
                mini1=value;
            elseif(value==mini1 )
                mini2=mini1;
            elseif(mini1<value && value<mini2)
                mini2=value;
            end
        end

        % transfer appropriate value to connected cns
        for vn = ith_VN_list

            if(abs(VN_2_CN_values(cn,vn))==mini1)
                m=mini2;
            else
                m=mini1;
            end

            % L_ext = sign(L_ext)*|L_ext|
            CN_2_VN_values(cn,vn)=sign(VN_2_CN_values(cn,vn))*sgn*m ;
        end

    end
    CN_2_VN=CN_2_VN_values;

end
```

c_hat: Estimate codeword after each iteration

```
 function Estimated_codeword = c_hat(VN_to_CN_map,CN_2_VN_values,
received_L)

    number_of_vn = size(VN_to_CN_map,1);
    Estimated_codeword = zeros(1,number_of_vn);

    itr=1;
    for vn = 1:number_of_vn
        sum_of_L = received_L(vn);

        for cn = VN_to_CN_map{vn}
            sum_of_L = sum_of_L + CN_2_VN_values(cn,vn);
        end

        Estimated_codeword(itr)= sum_of_L;
        itr=itr+1;
```

```
    end

end
```

Other used function

get_CN_VN_map: Get CN to VN adj List

```
function output = get_CN_VN_map(H)
    [row,col]=size(H);
    output = cell(row, 1);

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

get_VN_CN_map: Get VN to CN adj List

```
function output = get_VN_CN_map(H)
    [row , col]=size(H);
    output = cell(col, 1);

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

nrldpc_encode: Get Encoded Codeword form msg

```
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
```

mul_sh : Multiply codeword with Zero , identity or Shifted Identity Matrix

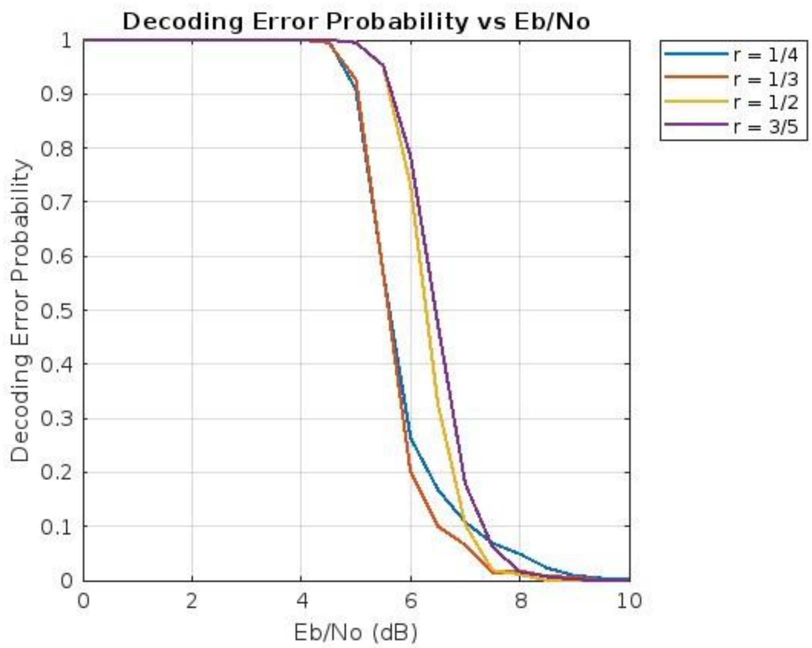
```
function y=mul_sh(x,k)
%x=inout blocks
%k= -1 or shifts
%y = output

    if(k==-1)
        y = zeros(1,length(x));
    else
        y = [x(k+1:end) x(1:k)];  %multiplication by shift identity
    end

end
```

nrldpc_Hmatrix : Base matrix Expantion

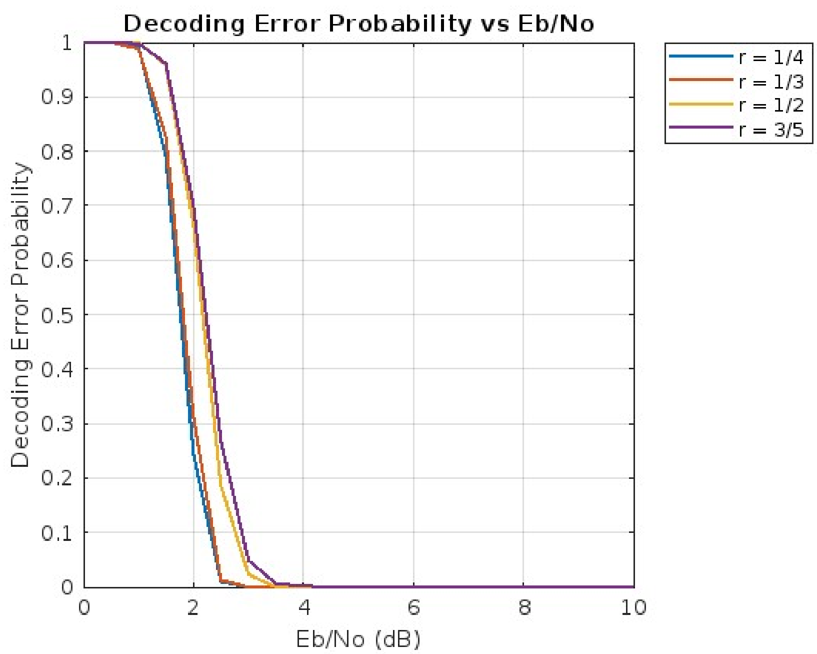
```
function [B,H,z] = nrldpc_Hmatrix(BG)

load(sprintf('%s.txt',BG),BG);
B = NR_2_6_52;
[mb,nb] = size(B);
z = 52;
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;
P = H(:,1:K);
G = [eye(K); P];
Z = H*G;
end
```

Final results of
Matrix1 (NR_2_6_52)

Combine graphs



Hard Decision Decoding (HDD)

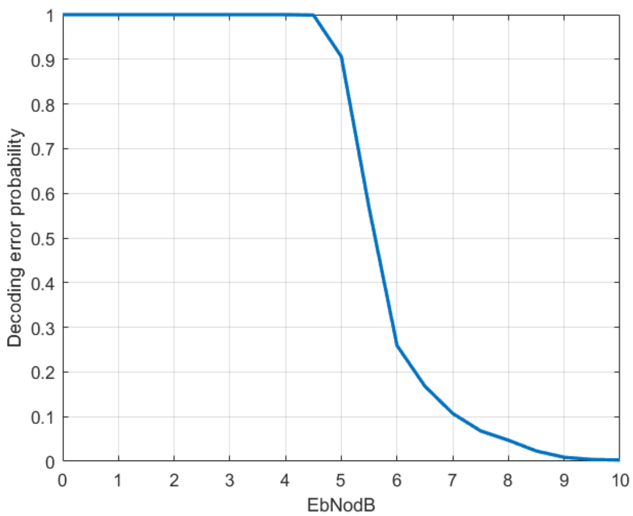


Soft Decision Decoding (SDD)

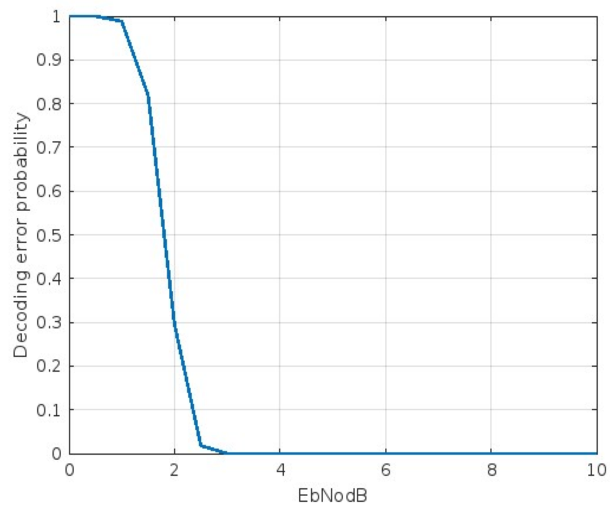
Code Rate = 1/4:-

HDD | **SDD**

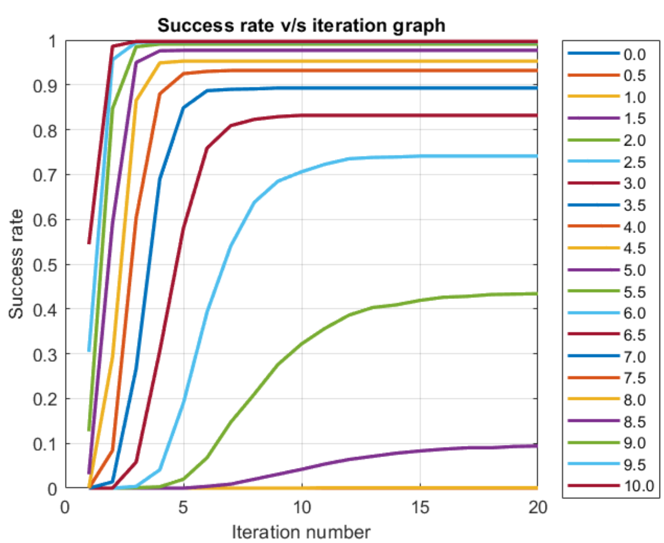
Decoding error probability



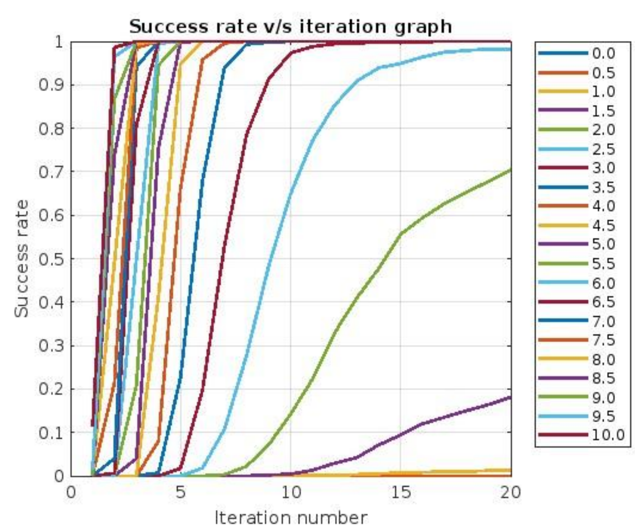
Decoding error probability



Success rate in each iteration



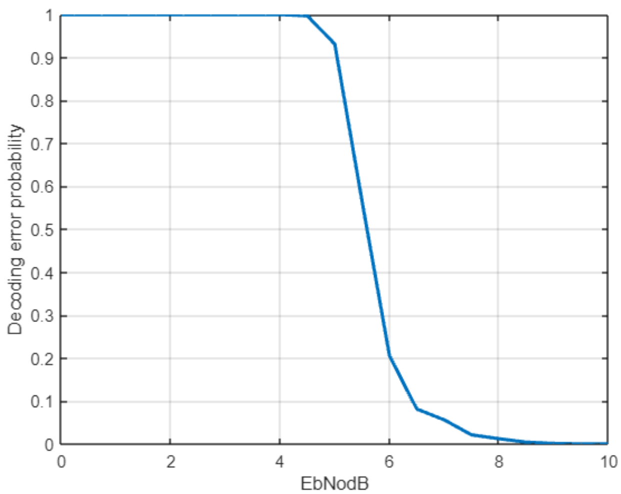
Success rate in each iteration



Code rate = 1/3 :-

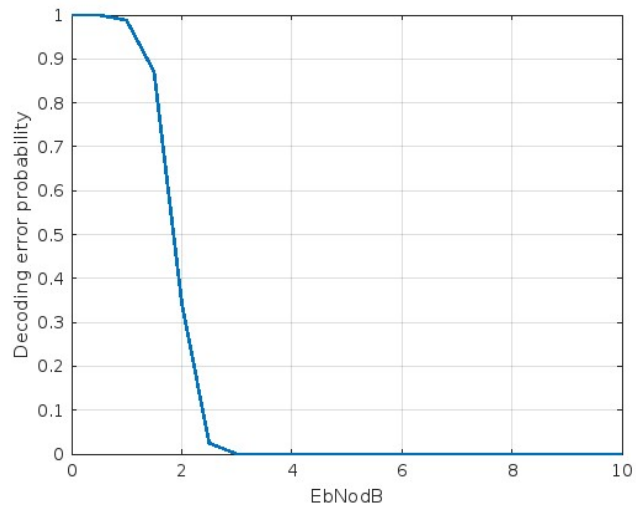
HDD

Decoding error probability

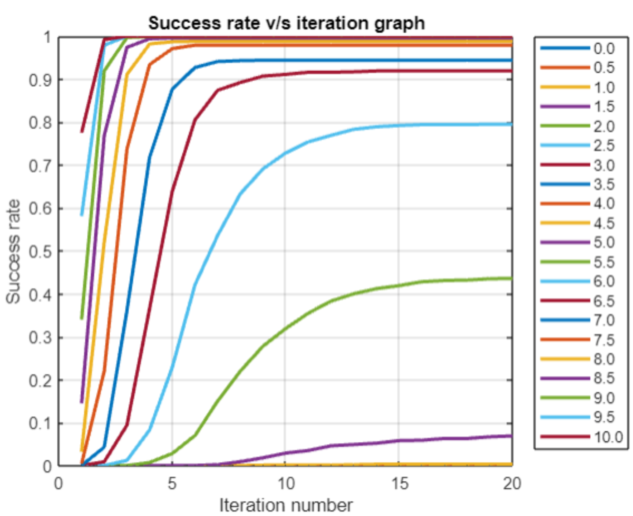


SDD

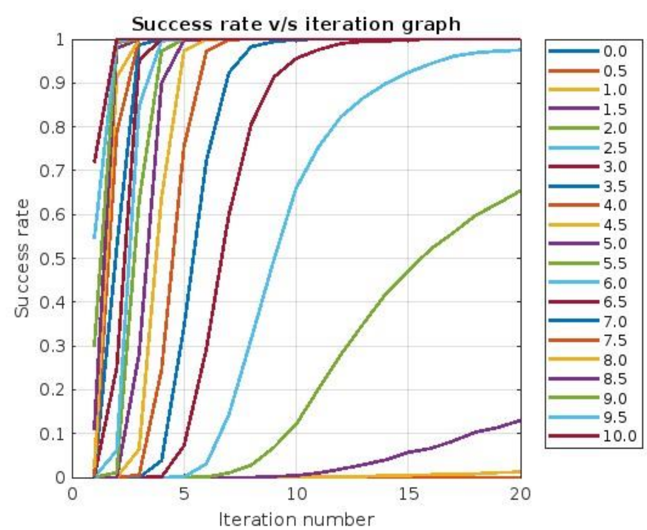
Decoding error probability



Success rate in each iteration



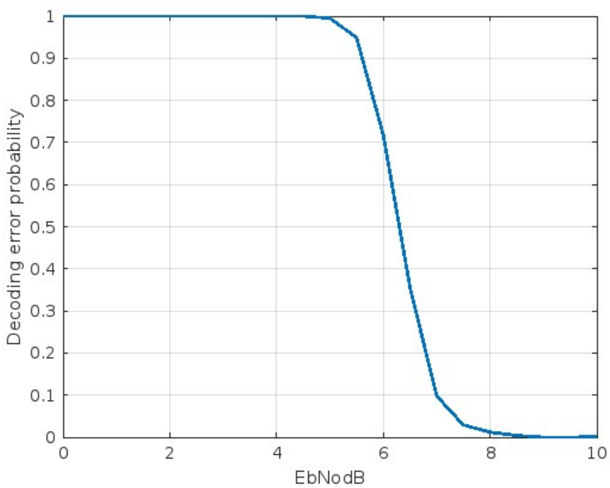
Success rate in each iteration



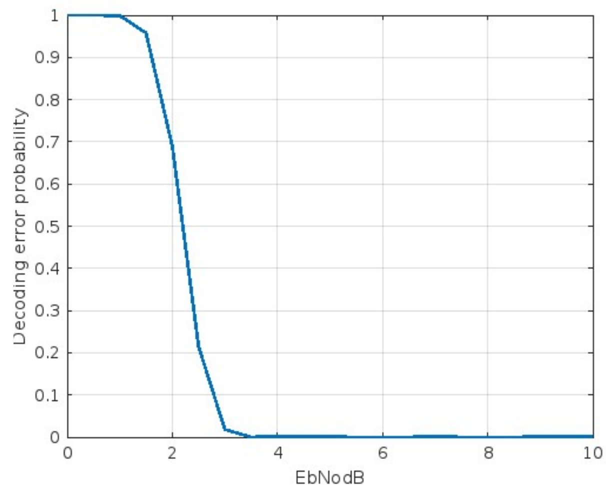
Code rate = 1/2:-

HDD **SDD**

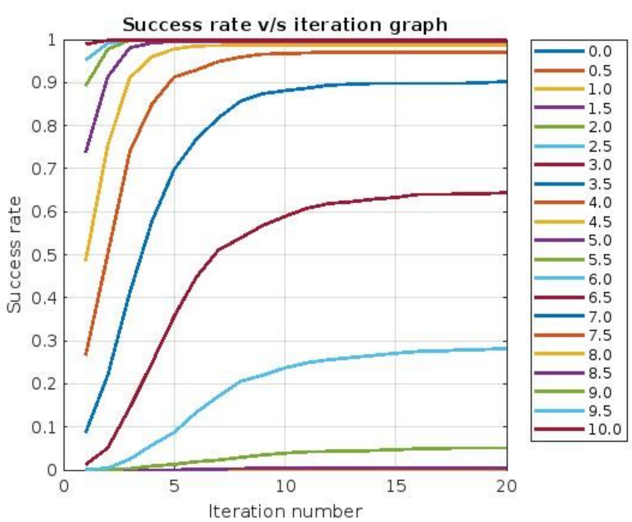
Decoding error probability



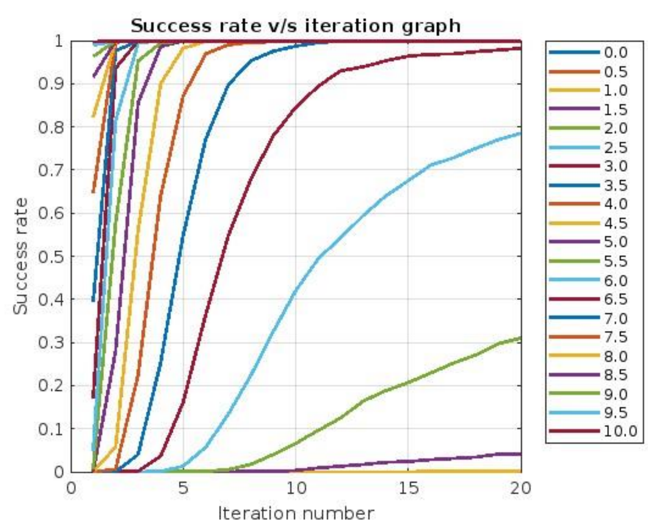
Decoding error probability



Success rate in each iteration



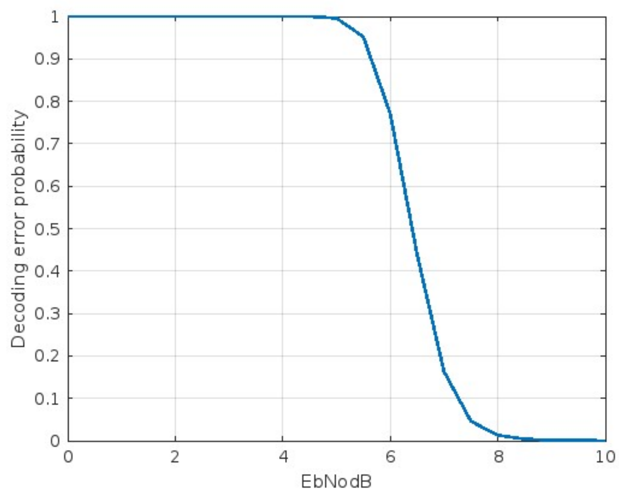
Success rate in each iteration



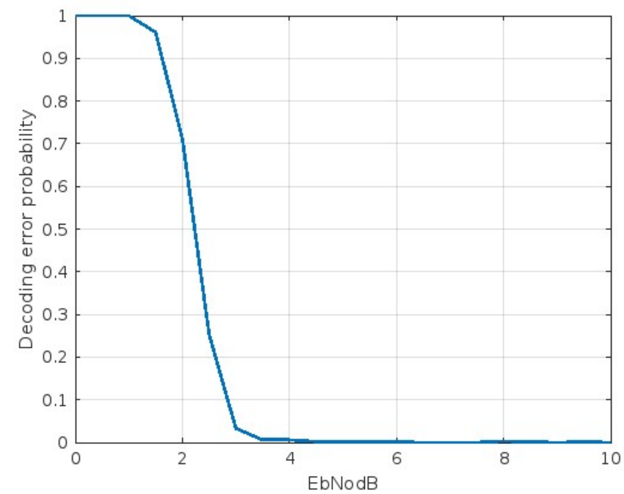
Code rate = 3/5

HDD | **SDD**

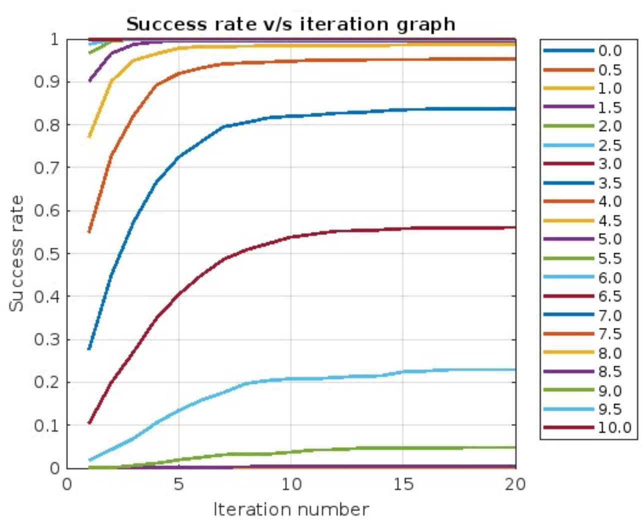
Decoding error probability



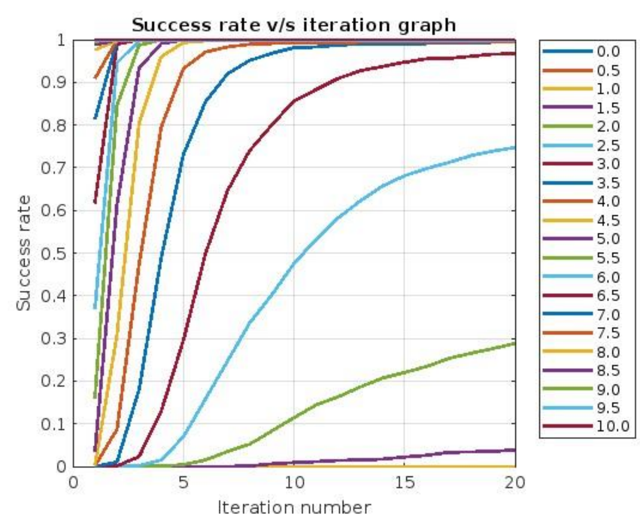
Decoding error probability



Success rate in each iteration

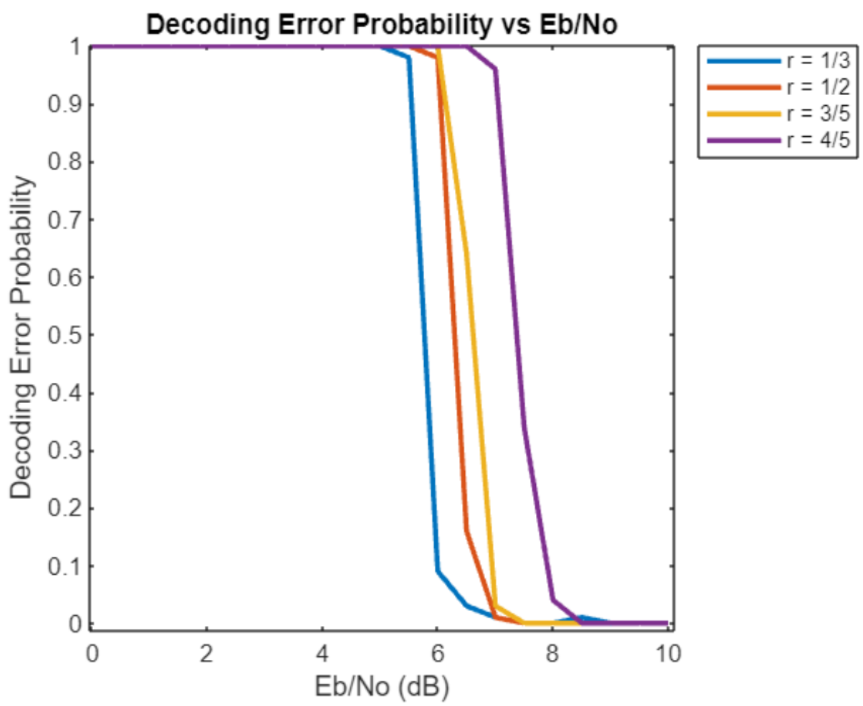


Success rate in each iteration

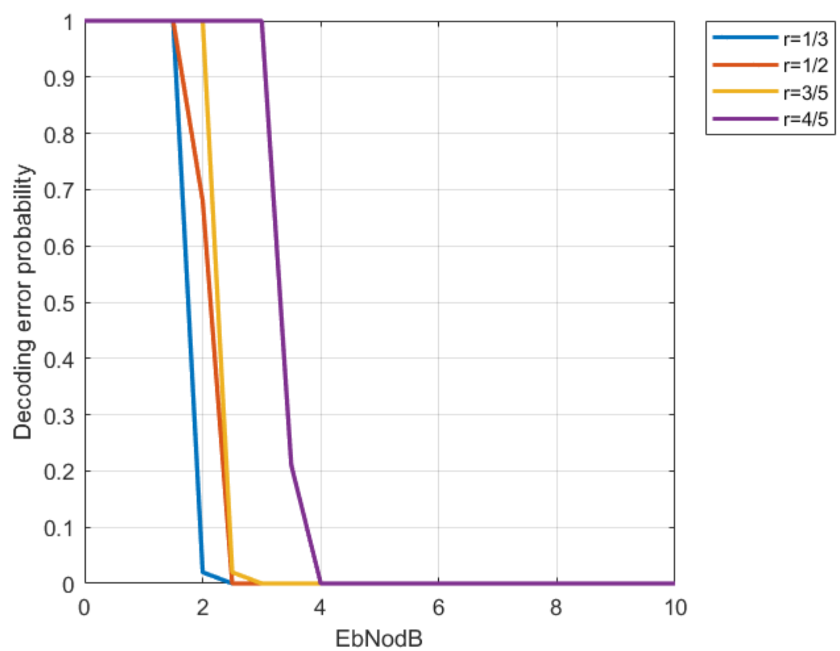


Matrix2 (NR_1_5_352)

Combine graph



Hard Decision Decoding (HDD)

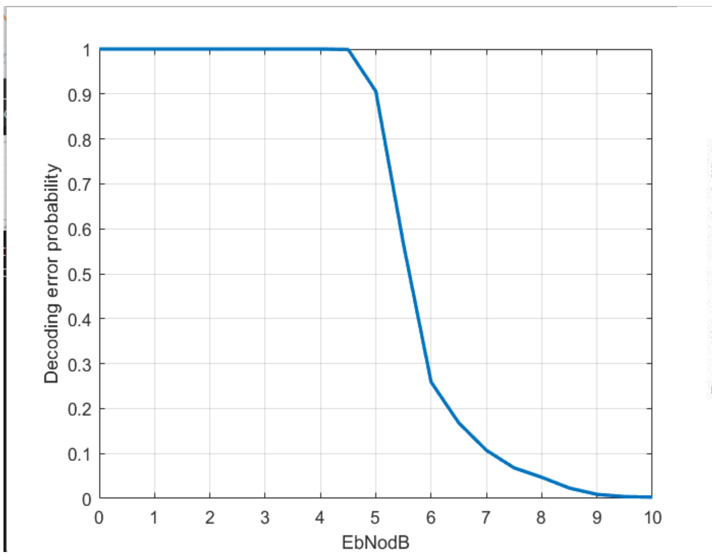


Soft Decision Decoding (SDD)

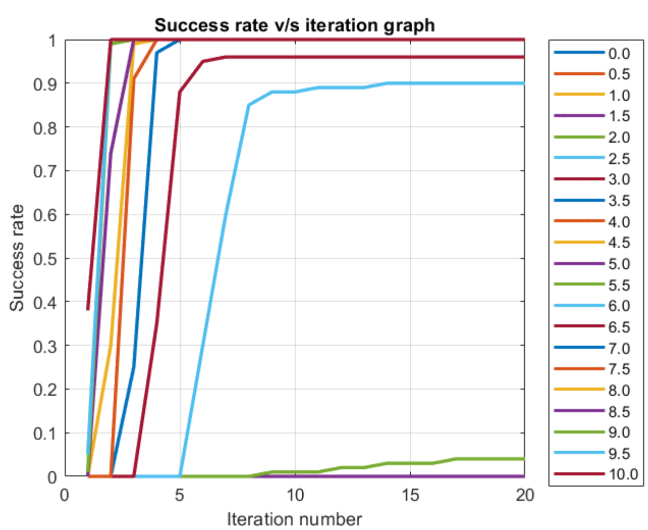
Code rate = 1/3 :-

HDD

Decoding error probability

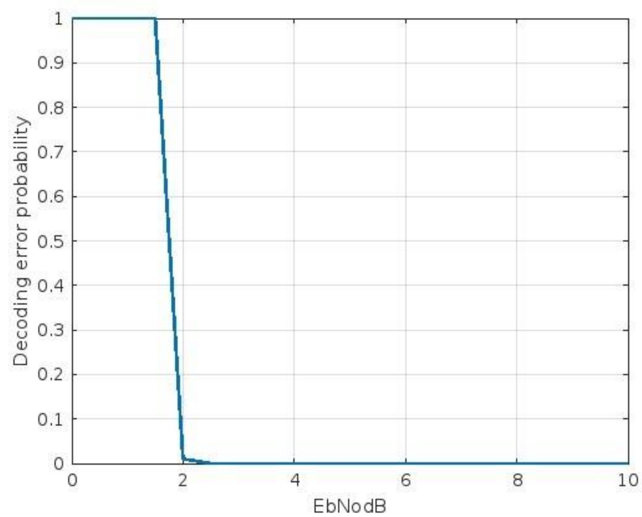


Success rate in each iteration

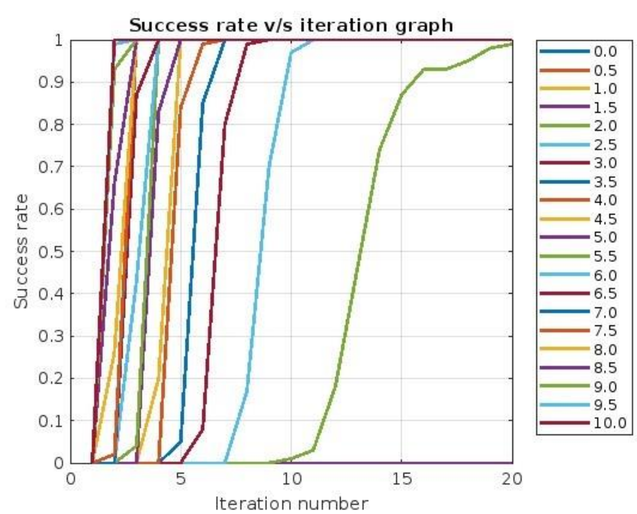


SDD

Decoding error probability



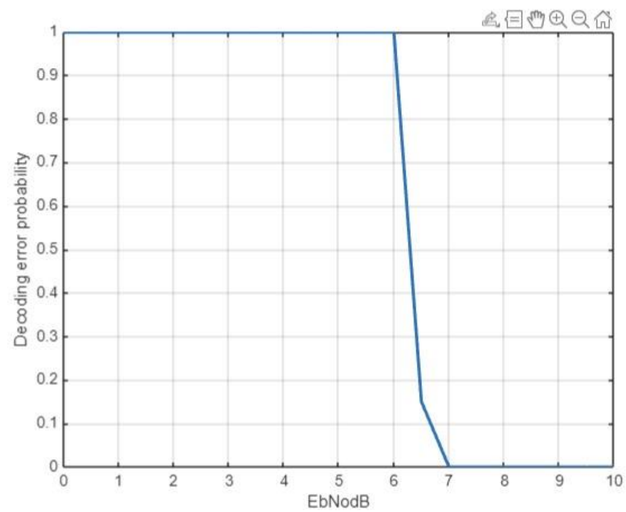
Success rate in each iteration



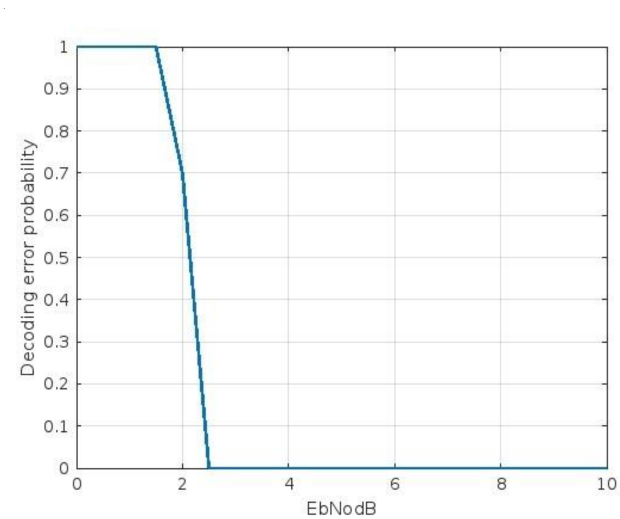
Code rate = 1/2:-

HDD | **SDD**

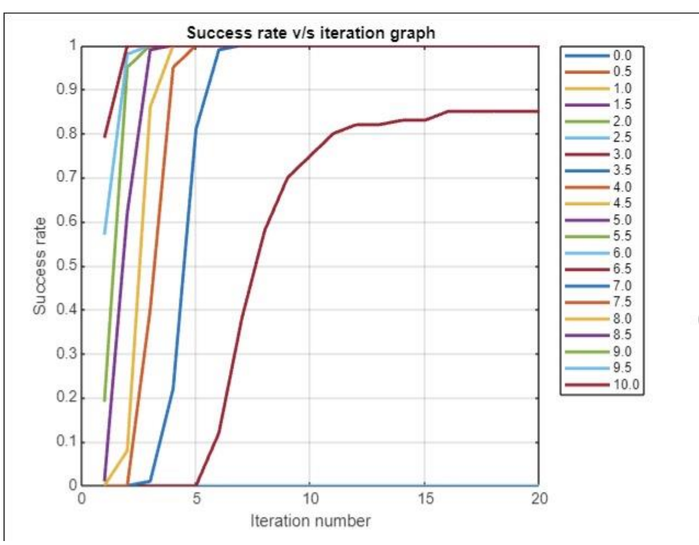
Decoding error probability



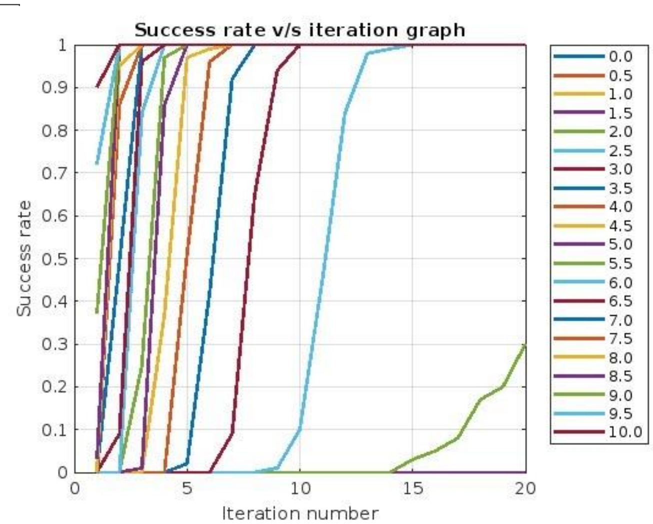
Decoding error probability



Success rate in each iteration



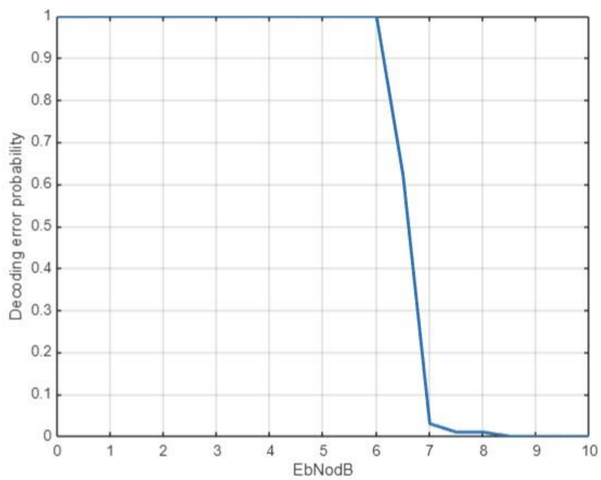
Success rate in each iteration



Code rate = 3/5:-

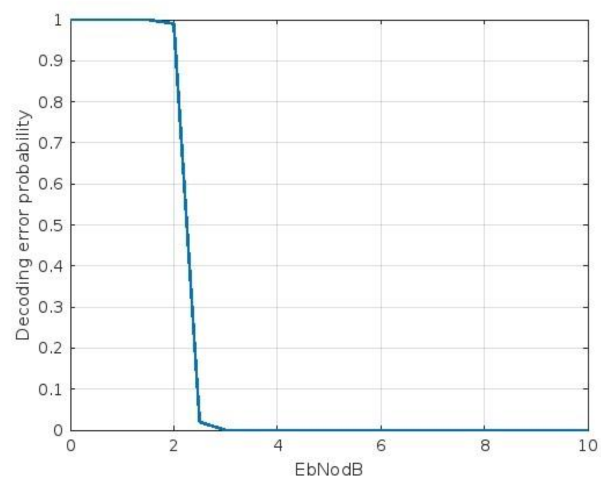
HDD

Decoding error probability

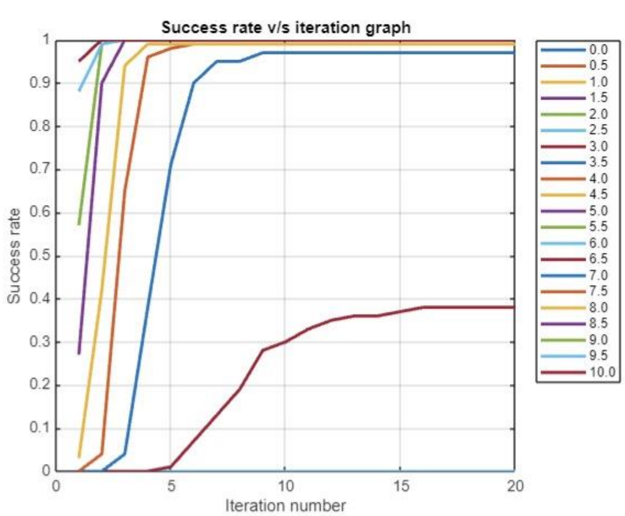


SDD

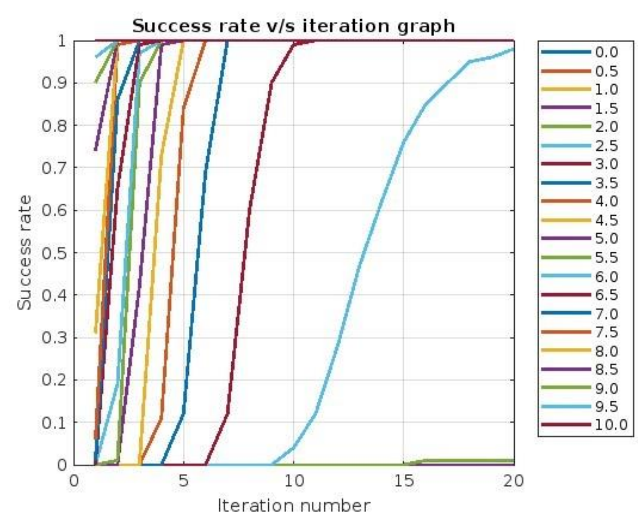
Decoding error probability



Success rate in each iteration



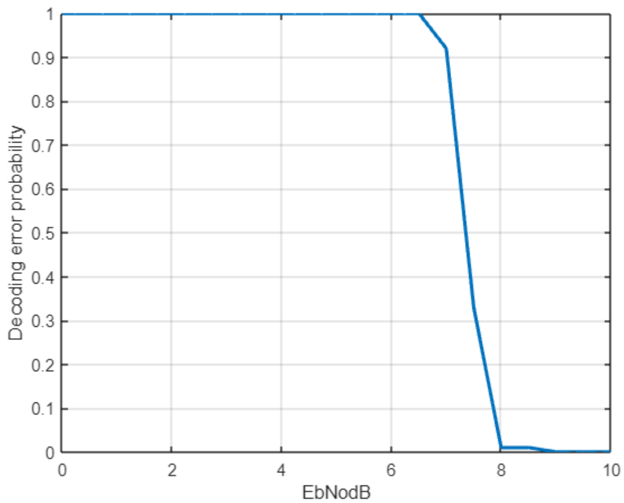
Success rate in each iteration



Code rate = 4/5:-

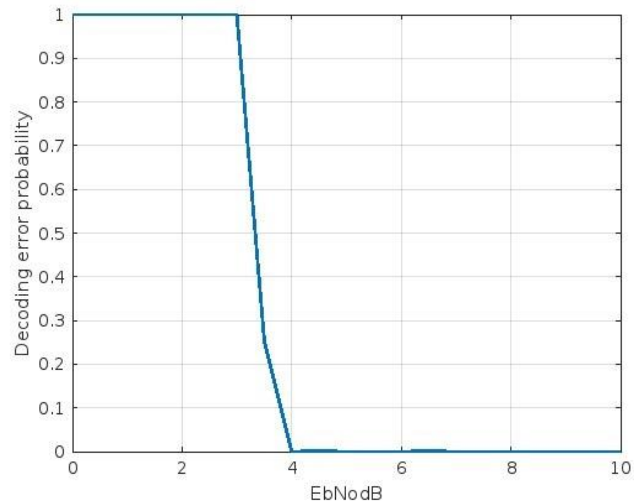
HDD

Decoding error probability

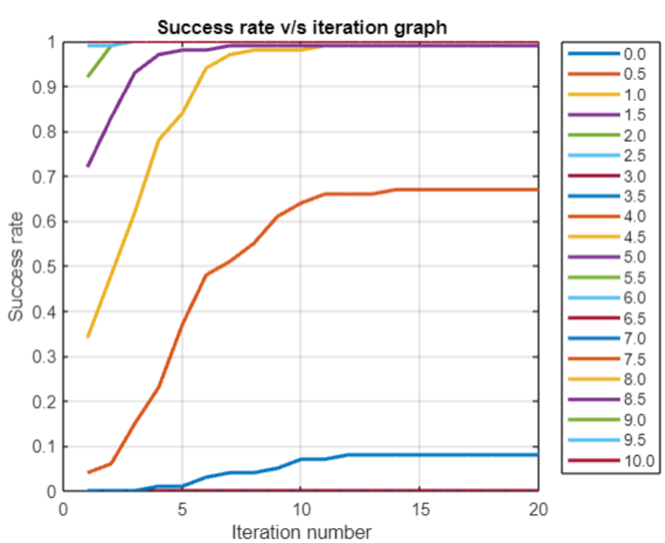


SDD

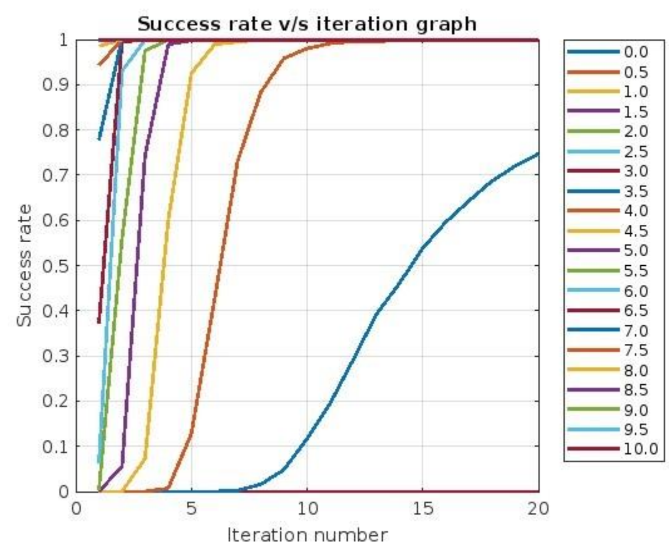
Decoding error probability



Success rate in each iteration



Success rate in each iteration



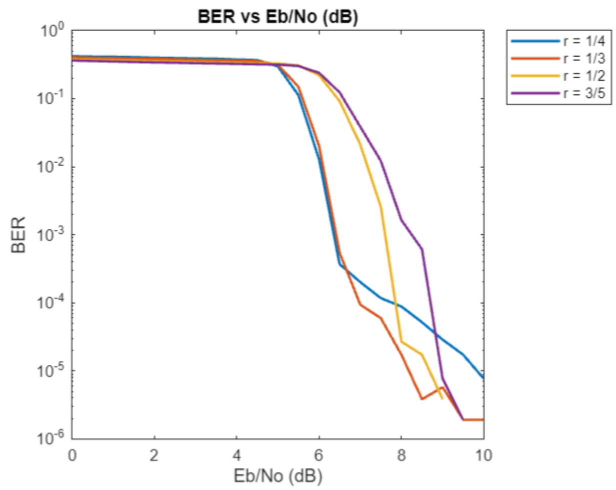
Observation:

- As the signal-to-noise ratio (SNR) increases, the corresponding variance of noise decreases. Consequently, the decoding error probability decreases, leading to an increase in the success rate of each iteration for both decoding schemes.
- Additionally, the soft decision decoder outperforms the hard decision decoder for any code rate and SNR.

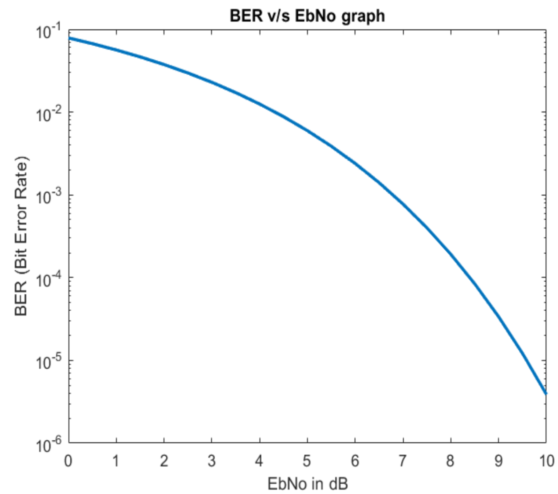
Comparison of the simulation with Shannon channel capacity bound

- BER is the number of bits that are received in error divided by the total number of bits.
  - BER v/s Eb/No graph of Hard decision decoding:

Coded

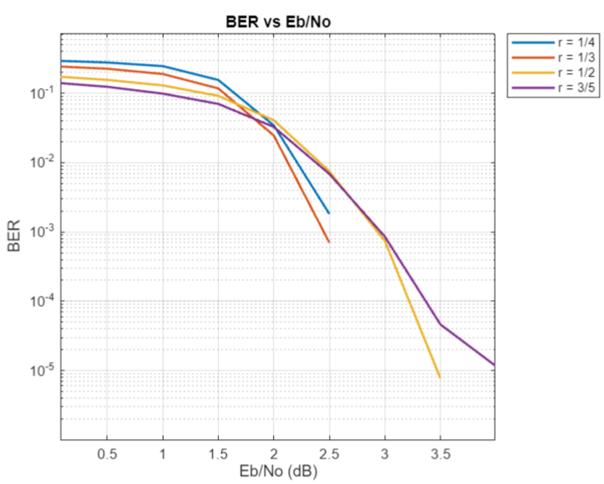


Uncoded

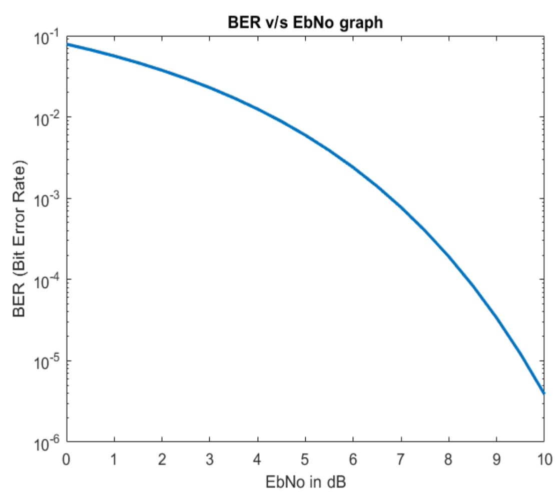


  - BER v/s Eb/No graph of Soft decision decoding:

Coded



Uncoded



    - In the above graphs, we observe that the BER vs. Eb/No curve for Soft Decision Decoding achieves the same BER for a lower Eb/No value.
    - This suggests that Soft decision decoding approaches the theoretical limit set by Shannon's capacity theorem more closely than Hard decision decoding does. This indicates that Soft decision decoding provides better performance in terms of error correction capability.

References:

[1] Andrew Thangaraj. Ldpc and polar codes in the 5g standard, 2019. Accessed on March 24,2024.

[2] Implementation of Low-Density Parity-Check codes for 5G NR shared channels: LIFANG WANG.

[3] CT216. Introduction to Communication Systems: Lecture 3 Channel Coding. DA-IICT, Winter 2024: Prof. Yash Vasavada.

[4] LDPC Codes - a brief Tutorial: Bernhard M.J. Leiner