



CT 216  
INTRODUCTION TO COMMUNICATION SYSTEM  
PROJECT LDPC CODES

Group No.- 8  
Professor:-Yash Vasvada  
Mentor:-Vivek Patel

## 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.



Kavya Parmar



Agrim Sharma



Ritu Patel



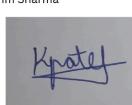
Ayush Patel



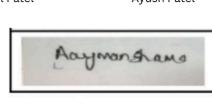
Tapas



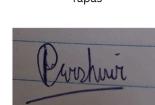
Dhameliya Utsker



Krish Patel



Ayman Ammar Shams



Harshvir Singh



Het Rank

## **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. Normal Approximation and Shannon's limit**
- 5. Appendix A**

# Hard Decision Decoding

```
% Base graphs

baseGraphs = {'NR_2_6_52', 'NR_1_5_352'};
codeRates_BG = {[1/4, 1/3, 1/2, 3/5], [1/3, 1/2, 3/5, 4/5]};

EbNodB_vec = 0:1:10;

Nsim = 10; maxIter = 20;

% Functionality test (no noise)

disp("Running functionality test (no noise)...")
run_functionality_test(baseGraphs{1}, codeRates_BG{1}(1));

% Loop through base graphs and code rates

for bgIdx = 1:length(baseGraphs)

    bg = baseGraphs{bgIdx};

    rates = codeRates_BG{bgIdx};

    load([bg, '.txt'], bg); B = eval(bg);

    [mb, nb] = size(B); kb = nb - mb; z = 52;

    for r = 1:length(rates)

        R = rates(r);

        fprintf('Running for Base Graph: %s | Code Rate: %.2f\n',
        bg, R);

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

        nBlock = nbRM * z; kBlock = kb * z;
```

```

Hfull = gen_H(B, z); H = Hfull(:,1:nBlock);

H = H(1:(mb*z - nb*z + nBlock), :);

v2c = get_v2c_map(H); c2v = get_c2v_map(H);

% BER & PER

BER = zeros(1,length(EbNodB_vec)); PER =
zeros(1,length(EbNodB_vec));

iterSucc = Nsim * ones(1, maxIter);

for snrIdx = 1:length(EbNodB_vec)

    EbNo = 10^(EbNodB_vec(snrIdx)/10);

    sigma = sqrt(1/(2*R*EbNo));

    bitErr = 0; frameErr = 0;

    for s = 1:Nsim

        msg = randi([0 1], kBlock, 1);

        code = ldpc_encode(B, z, msg');

        code = code(1:nBlock);

        tx = 1 - 2 * code;

        rx = tx + sigma * randn(1,nBlock);

        [dec, iterSucc] = hard_decode(rx, v2c, c2v,
zeros(size(H)), zeros(size(H)), iterSucc);

        e = mod(dec(1:kBlock) + msg', 2);

        bitErr = bitErr + sum(e);

        frameErr = frameErr + any(e);

    end

    BER(snrIdx) = bitErr / (Nsim * kBlock);

    PER(snrIdx) = frameErr / Nsim;

end

% Plot each result in a separate figure

```

```

        figure('Name', sprintf('%s - Rate %.2f', bg, R),
'NumberTitle', 'off');

        tiledlayout(2,2);

        nexttile; semilogy(EbNodB_vec, PER, '-o'); grid on;
        title('Eb/No vs Decoding error'); xlabel('Eb/No (dB)');
        ylabel(sprintf('Hard Decision Decoding error probability of
CodeRate = %.2f\n', R));
        nexttile; plot(1:maxIter, iterSucc, '-'); grid on;
        title(sprintf('Probability vs Iteration for Hard Decoding of
CodeRate = %.2f\n', R)); xlabel('Iteration number');
        ylabel('Success probability of each iteration');
        nexttile; semilogy(EbNodB_vec, BER, '-*'); grid on;
        title('BER vs Eb/No'); xlabel('Eb/No (dB)'); ylabel('BER');
    end
end

% ----- Functionality Test -----
function run_functionality_test(baseGraph, codeRate)
    load([baseGraph, '.txt'], baseGraph); B = eval(baseGraph);
    [mb,nb] = size(B); kb = nb - mb; z = 52;
    k_pc = kb-2; nbRM = ceil(k_pc/codeRate)+2;
    nBlock = nbRM*z; kBlock = kb*z;
    Hfull = gen_H(B,z); H = Hfull(:,1:nBlock);
    H = H(1:(mb*z - nb*z + nBlock), :);
    v2c = get_v2c_map(H); c2v = get_c2v_map(H);
    msg = randi([0 1], kBlock, 1);
    code = ldpc_encode(B,z,msg'); code = code(1:nBlock);
    rx = 1 - 2 * code; % NO NOISE
    [decoded,~] = hard_decode(rx, v2c, c2v, zeros(size(H)),
zeros(size(H)), 20*ones(1,20));

```

```

recovered = decoded(1:kBlock);

assert(all(recovered' == msg), 'Functionality test failed!');

disp("Functionality test passed. Decoding without noise was
successful.");

end

function H = gen_H(B,z)

[mb,nb] = size(B);

H = zeros(mb*z, nb*z);

Iz = eye(z); I0 = zeros(z);

for i = 1:mb

    for j = 1:nb

        row = (i-1)*z + (1:z);

        col = (j-1)*z + (1:z);

        if B(i,j) == -1

            H(row,col) = I0;

        else

            H(row,col) = circshift(Iz, -B(i,j));

        end

    end

end

end

function code = ldpc_encode(B,z,msg)

[m,n] = size(B); code = zeros(1,n*z);

code(1:(n-m)*z) = msg;

temp = zeros(1,z);

for i = 1:4

    for j = 1:n-m

```

```

        temp = mod(temp + shift_block(msg((j-1)*z+1:j*z),
B(i,j)), 2);

    end

end

p1sh = B(2,n-m+1); if p1sh== -1, p1sh = B(3,n-m+1); end

code((n-m)*z+1:(n-m+1)*z) = shift_block(temp, z-p1sh);

for i = 1:3

    temp = zeros(1,z);

    for j = 1:n-m+i

        temp = mod(temp + shift_block(code((j-1)*z+1:j*z),
B(i,j)), 2);

    end

    code((n-m+i)*z+1:(n-m+i+1)*z) = temp;

end

for i = 5:m

    temp = zeros(1,z);

    for j = 1:n-m+4

        temp = mod(temp + shift_block(code((j-1)*z+1:j*z),
B(i,j)), 2);

    end

    code((n-m+i-1)*z+1:(n-m+i)*z) = temp;

end

end

function y = shift_block(x,k)

z = length(x);

if k == -1

    y = zeros(1, z);

else

```

```

    k = mod(k, z);

    y = [x(k+1:end), x(1:k)];

end

function map = get_v2c_map(H)

[~,c] = size(H); map = cell(c,1);

for i = 1:c

map{i} = find(H(:,i));

end

end

function map = get_c2v_map(H)

[r,~] = size(H); map = cell(r,1);

for i = 1:r

map{i} = find(H(i,:));

end

end

function [dec_out, iterSucc] = hard_decode(rx, v2c, c2v, v2c_val,
c2v_val, iterSucc)

dec_out = double(rx < 0); prev = dec_out;

for it = 1:length(iterSucc)

if it == 1

for vn = 1:length(v2c)

for cn = v2c{vn}'


```

```

    v2c_val(cn,vn) = dec_out(vn);

    end

    end

else

    for vn = 1:length(v2c)

        onesum = dec_out(vn) + sum(c2v_val(v2c{vn}, vn));

        for cn = v2c{vn}'

            v2c_val(cn,vn) = (onesum - c2v_val(cn,vn)) >
length(v2c{vn})/2;

            end

        end

    end

    for cn = 1:length(c2v)

        vn_list = c2v{cn};

        total = xor_all(v2c_val(cn, vn_list));

        for idx = 1:length(vn_list)

            vn = vn_list(idx);

            c2v_val(cn,vn) = xor(total, v2c_val(cn,vn));

        end

    end

    for vn = 1:length(v2c)

        onesum = dec_out(vn) + sum(c2v_val(v2c{vn}, vn));

        dec_out(vn) = onesum > (length(v2c{vn})+1)/2;

    end

    if all(dec_out == prev), break; end

```

```

    prev = dec_out; iterSucc(it) = iterSucc(it) - 1;

end

function result = xor_all(vec)

result = 0;

for i = 1:length(vec)

    result = xor(result, vec(i));

end

end

```

## Soft Decision Decoding

```

% Parameters

baseGraphs = {'NR_2_6_52', 'NR_1_5_352'};

codeRatesMap = containers.Map;

codeRatesMap('NR_2_6_52') = [1/4, 1/3, 1/2, 3/5];

codeRatesMap('NR_1_5_352') = [1/3, 1/2, 3/5, 4/5];

eb_no_dbvec = 0:0.5:10;

z = 52;

nsim = 10;

max_it = 20;

iterations = 1:max_it;

```

```

colors = lines(length(eb_no_dbvec));

% Loop over base graphs

for bg_idx = 1:length(baseGraphs)

    baseGraph5GNR = baseGraphs{bg_idx};

    coderate = codeRatesMap(baseGraph5GNR);

    disp(['Running simulations for ', baseGraph5GNR]);

    [B, Hfull, z] = nrldpc_Hmatrix(baseGraph5GNR, z);

    for idxCR = 1:length(coderate)

        cr = coderate(idxCR);

        disp(['Simulating code rate = ', num2str(cr)]);

        % Rate matching

        [mb, nb] = size(B);

        kb = nb - mb;

        kNumInfoBits = kb * z;

        k_pc = kb - 2;

        nbRM = ceil(k_pc/cr) + 2;

        nBlockLength = nbRM * z;

        H = Hfull(:, 1:nBlockLength);

        nChecksNotPunctured = mb*z - nb*z + nBlockLength;

        H = H(1:nChecksNotPunctured, :);

        [row, col] = size(H);

        k = col - row;

        cn_to_vn_map = cn_vn(H);

        vn_to_cn_map = vn_cn(H);

        decoding_error = zeros(1, length(eb_no_dbvec));

        successProb_it = zeros(max_it, length(eb_no_dbvec));

        figure('Name', sprintf('%s: Rate %.2f', baseGraph5GNR, cr),
        'NumberTitle', 'off');

```

```

    subplot(1,2,1); hold on;

    title(sprintf(' Probability vs Iteration for Soft Decoding
at Coderate= (Rate %.2f)', cr));

    xlabel('Iteration number'); ylabel('Success Probability at
each iteration'); grid on;

    for d_iter = 1:length(eb_no_dbvec)

        eb_no_db = eb_no_dbvec(d_iter);

        eb_no = 10^(eb_no_db/10);

        sigma = sqrt(1/(2*cr*eb_no));

        successCount = 0;

        iterationSuccess = nsim * ones(1, max_it);

        for sim = 1:nsim

            org_msg = randi([0 1], k, 1);

            encoded_msg = nrldpc_encode(B, z, org_msg');

            encoded_msg = encoded_msg(1:nBlockLength);

            bpsk_msg = 1 - 2 * encoded_msg;

            received_bpsk = bpsk_msg + sigma * randn(1,
nBlockLength);

            prev_msg = (received_bpsk < 0);

            vn_sum_vec = zeros(1, col);

            L = zeros(size(H));

            for it = 1:max_it

                if it == 1

                    for i = 1:col

                        for j = vn_to_cn_map{i}

                            L(j, i) = received_bpsk(i);

                        end

                    end

                else


```

```

    for i = 1:col

        for j = vn_to_cn_map{i}

            L(j, i) = vn_sum_vec(i) - L(j, i);

        end

    end

    for i = 1:row

        connected_vns = cn_to_vn_map{i};

        abs_vals = abs(L(i, connected_vns));

        signs = sign(L(i, connected_vns));

        total_sign = prod(signs(signs~=0));

        [min1, idx1] = min(abs_vals);

        tmp = abs_vals; tmp(idx1) = Inf;

        min2 = min(tmp);

        for j = 1:length(connected_vns)

            vn_idx = connected_vns(j);

            min_val = (j == idx1) * min2 + (j ~=
idx1) * min1;

            L(i, vn_idx) = total_sign * signs(j) *
min_val;

        end

    end

    vn_sum_vec = received_bpsk + sum(L, 1);

    c_hat = (vn_sum_vec < 0);

    if all(c_hat(1:k) == org_msg')

        successCount = successCount + 1;

        break;

    else

```

```

iterationSuccess(it) = iterationSuccess(it)
- 1;

    end

    if isequal(prev_msg, c_hat)

        iterationSuccess(it+1:end) =
iterationSuccess(it+1:end) - 1;

        break;

    end

    prev_msg = c_hat;

end

end

decoding_error(d_iter) = 1 - (successCount / nsim);

successProb_it(:, d_iter) = 1 - (iterationSuccess ./
nsim);

plot(iterations, successProb_it(:, d_iter), ...

'Color', colors(d_iter, :), ...

'DisplayName', sprintf('%.1f dB', eb_no_db));

end

legend('show', 'Location', 'best'); hold off;

% Subplot 2: Error Probability vs Eb/No (log scale)

subplot(1,2,2); hold on;

semilogy(eb_no_dbvec, decoding_error, '-o', 'LineWidth', 2,
'DisplayName', 'Simulated');

grid on; xlabel('E_b/N_0 (dB)'); ylabel('Block Error
Probability');

title(sprintf('Error Probability (Rate %.2f)', cr));

% Add Shannon and NA only if r = 1/4

if round(cr*100) == 25

    disp('Simulated BER vs Normal Approximation vs Shannon
limit (Rate=%d, N=512)');

```

```

r = 0.25;

K = k;

N = K / r;

EbNodB_NA = 0:0.1:10;

EbNo_linear = 10.^ (EbNodB_NA / 10);

Pe_NA = zeros(size(EbNodB_NA));

for i = 1:length(EbNodB_NA)

    P = r * EbNo_linear(i);

    C = log2(1 + P);

    V = (log2(exp(1)))^2 * (P * (P + 2)) / (2 * (P +
1)^2);

    term = sqrt(N/V) * (C - r + log2(N) / (2 * N));

    Pe_NA(i) = qfunc(term);

end

shannon_limit_dB = 10 * log10((2^r - 1)/r);

semilogy(EbNodB_NA, Pe_NA, 'k--', 'LineWidth', 2,
'DisplayName', 'Normal Approximation');

xline(shannon_limit_dB, 'r--', 'LineWidth', 2, ...

    'Label', sprintf('Shannon Limit (%.2f dB)', ...
shannon_limit_dB), ...

    'LabelVerticalAlignment', 'bottom', 'DisplayName',
'Shannon Limit');

legend('show');

end

end

% ===== Supporting Functions =====

function [B, H, z] = nrldpc_Hmatrix(BGname, z)

B = load(sprintf('%s.txt', BGname), '-ascii');

```

```

[mb, nb] = size(B);

H = zeros(mb*z, nb*z);

Iz = eye(z); I0 = zeros(z);

for kk = 1:mb

    for kk1 = 1:nb

        shift = B(kk, kk1);

        tmpvecR = (kk-1)*z + (1:z);

        tmpvecC = (kk1-1)*z + (1:z);

        if shift == -1

            H(tmpvecR, tmpvecC) = I0;

        else

            H(tmpvecR, tmpvecC) = circshift(Iz, -mod(shift, z));

        end

    end

end

function out = cn_vn(H)

[row, col] = size(H);

out = cell(row, 1);

for i = 1:row

    out{i} = find(H(i, :) == 1);

end

end

function out = vn_cn(H)

[row, col] = size(H);

out = cell(col, 1);

for i = 1:col

    out{i} = find(H(:, i) == 1)';

end

```

```

    end

end

function cword = nrldpc_encode(B, z, msg)
[m, n] = size(B);
cword = zeros(1, n*z);
cword(1:(n-m)*z) = msg;
temp = zeros(1, z);
for i = 1:4
    for j = 1:n-m
        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);
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
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

function y = mul_sh(x, k)
z = length(x);

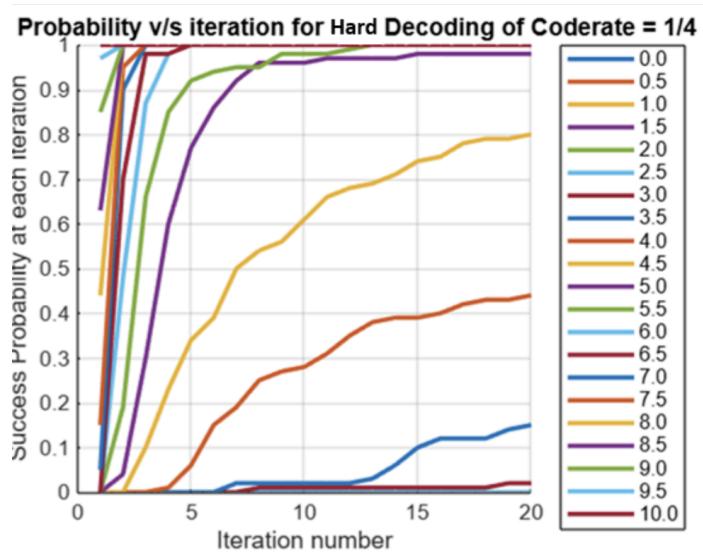
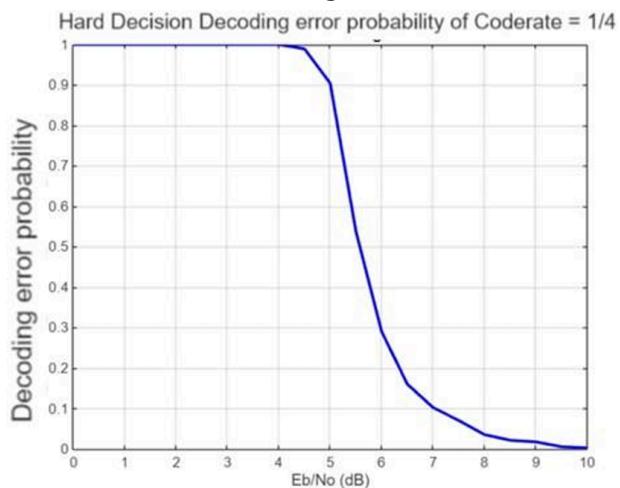
if k == -1
    y = zeros(size(x));
else
    k = mod(k, z);
    y = [x(k+1:end), x(1:k)];
end
end
```

## RESULTS

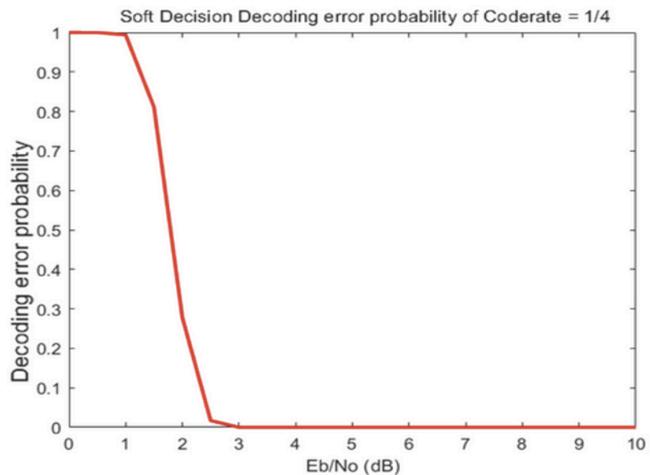
**1. For Matrix NR\_2\_6\_52:-**

a. Code Rate:- 1/4

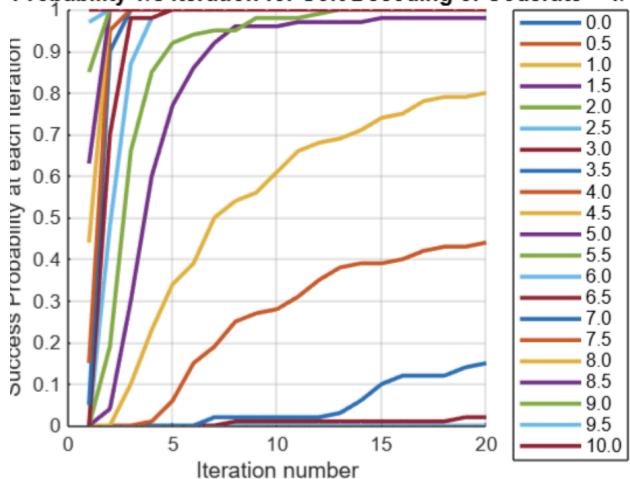
## Hard Decision Decoding



# Soft Decision Decoding



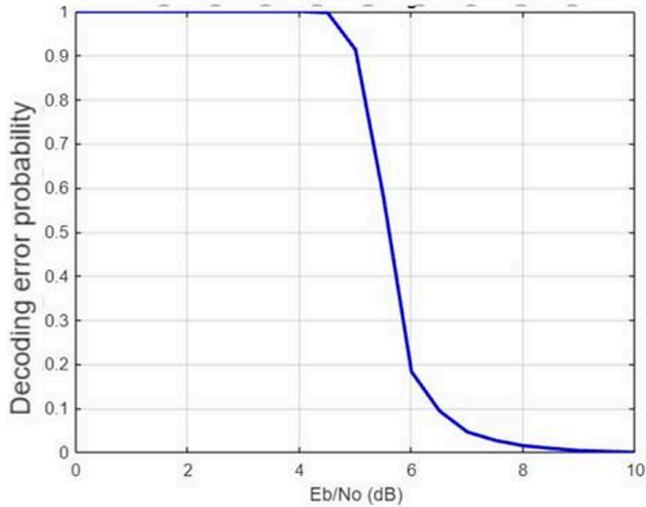
## Probability v/s iteration for Soft Decoding of Coderate = 1/4



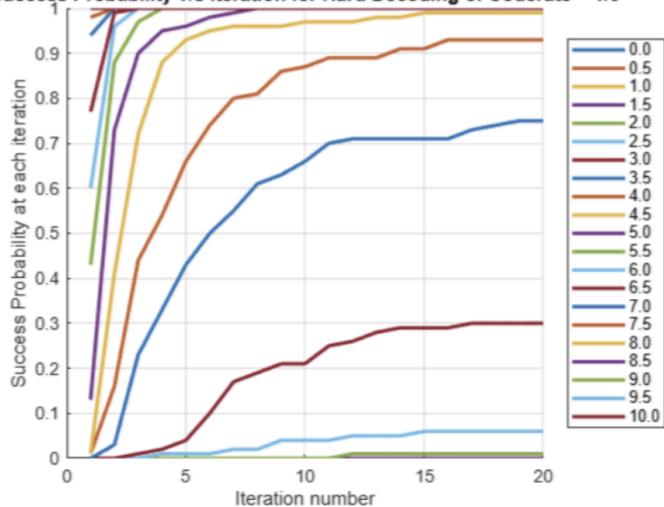
b. For Code Rate:-1/3

## Hard Decision Decoding

Hard Decision Decoding error probability of Coderate = 1/3

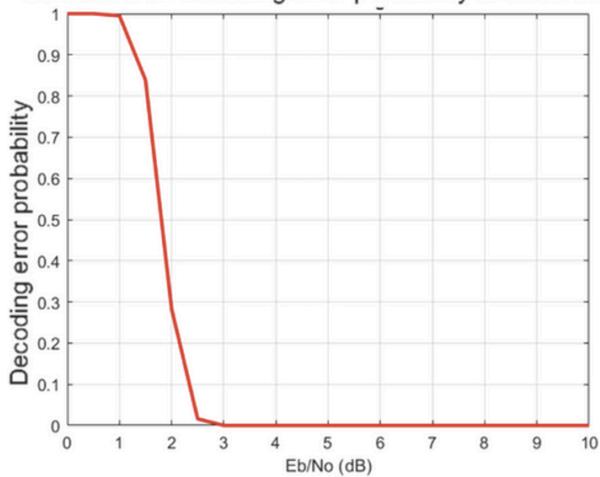


### Success Probability v/s iteration for Hard Decoding of Coderate = 1/3

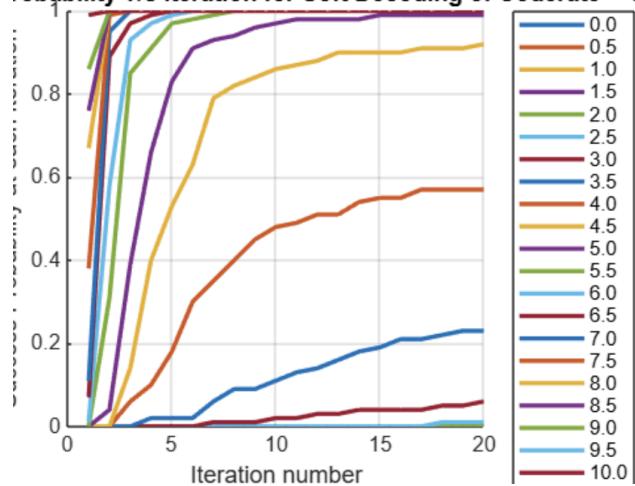


## Soft Decision Decoding

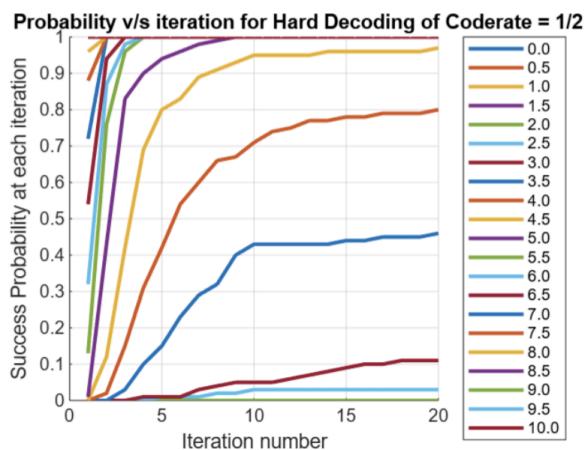
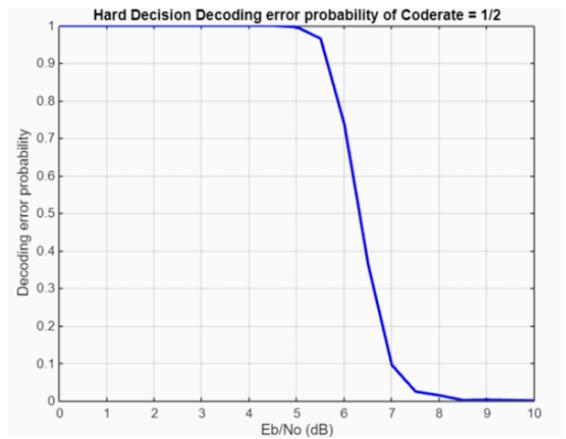
Soft Decision Decoding error probability of Coderate = 1/3



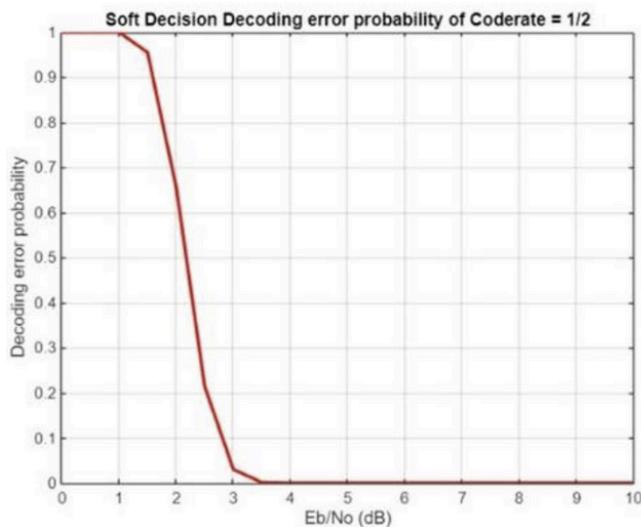
### Probability v/s iteration for Soft Decoding of Coderate = 1/3



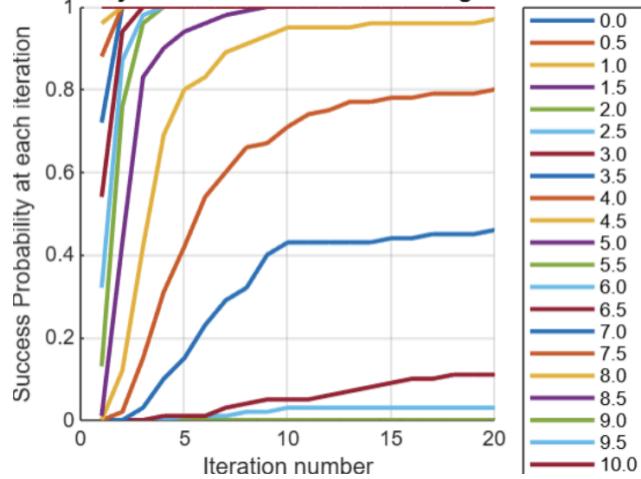
c. For Code Rate:- 1/2  
Hard Decision Decoding



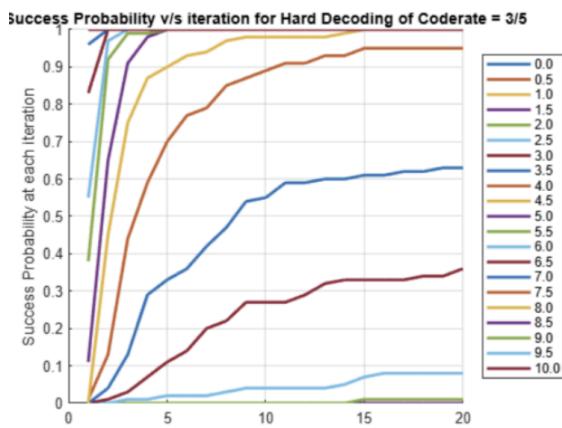
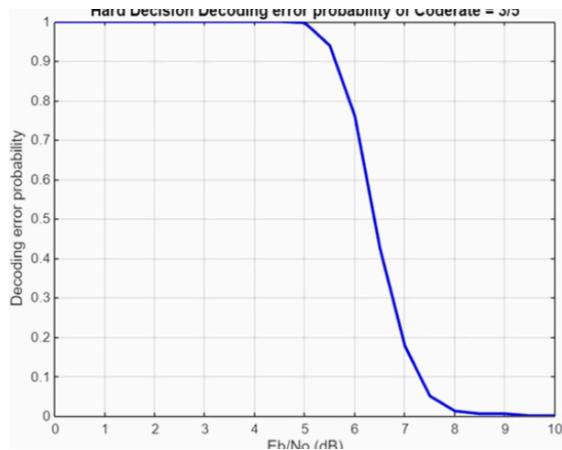
### **Soft Decision Decoding**



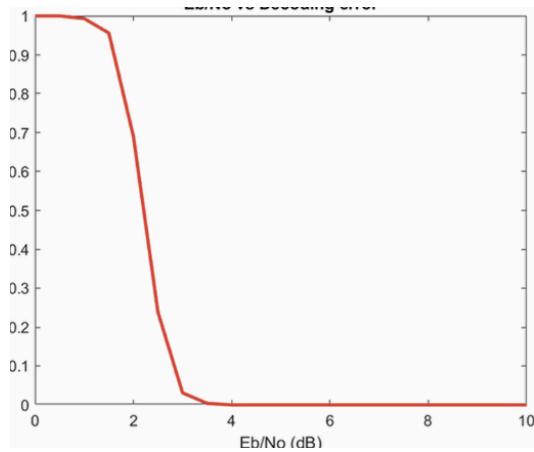
Probability v/s iteration for Soft Decoding of Coderate = 1/2

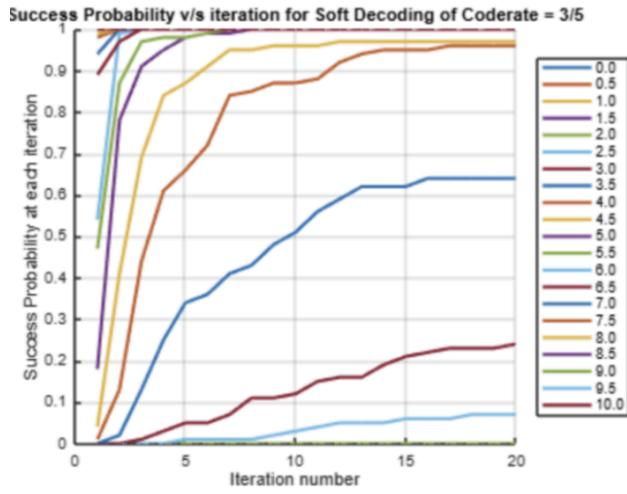


- d. For Code Rate:- 3/5  
Hard Decision Decoding

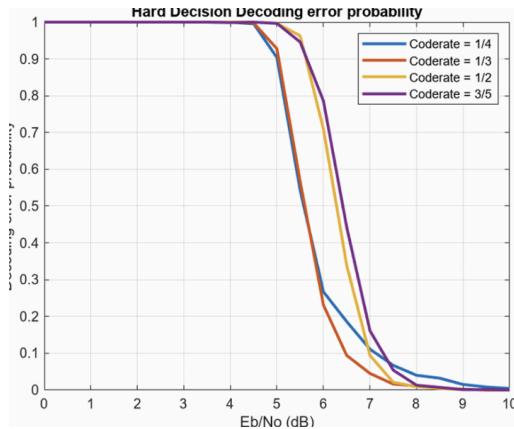


## Soft Decision Decoding

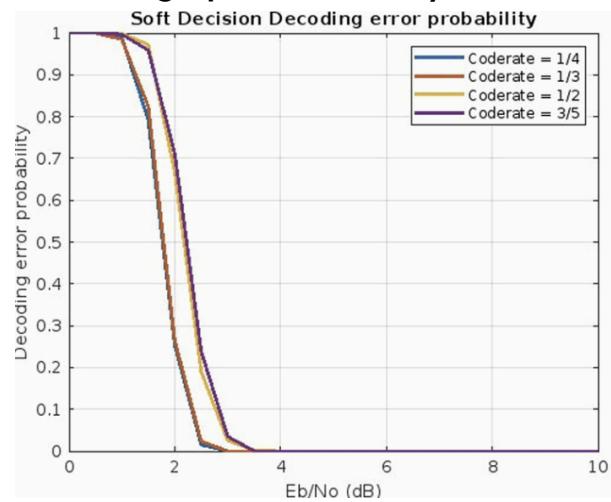




### Comparison of all graphs obtained by Hard Decision Decoding

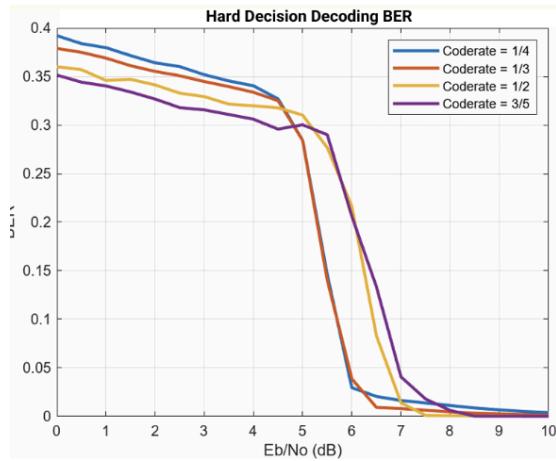


### Comparison of all graphs obtained by Soft Decision Decoding

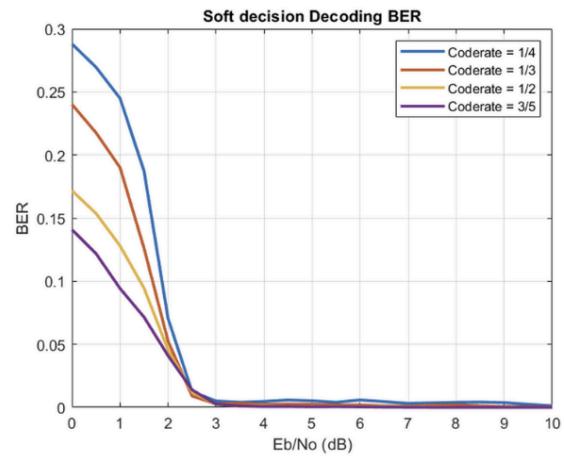


# Bit Error Rate

## HARD DECISION DECODING



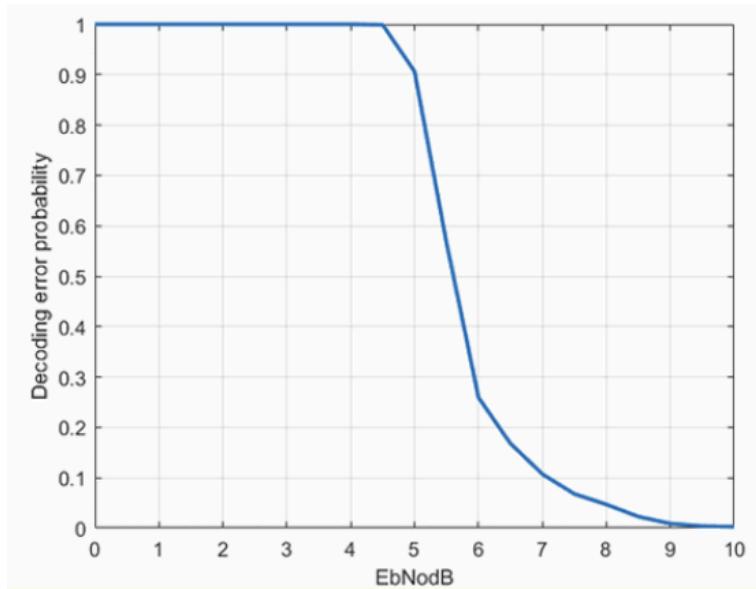
## SOFT DECISION DECODING



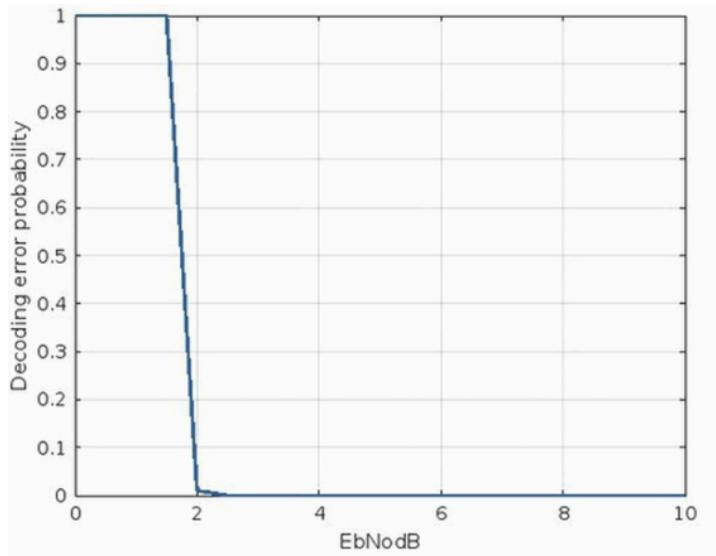
## 2. For Matrix NR\_1\_5\_352:-

### a. For Code Rate:- 1/3

#### Hard Decision Decoding

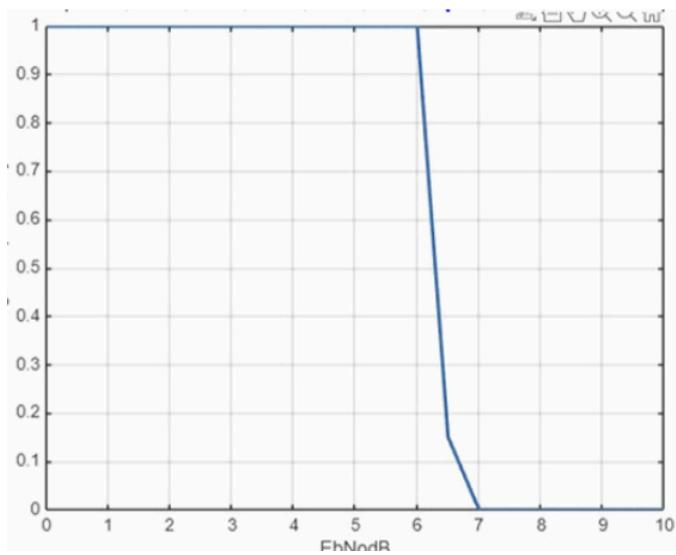


### **Soft Decision Decoding**



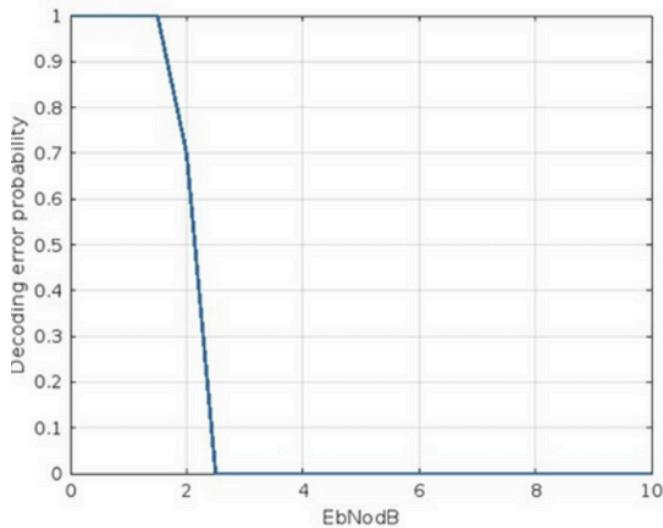
b. For Code Rate:- 1/2

### **Hard Decision Decoding**



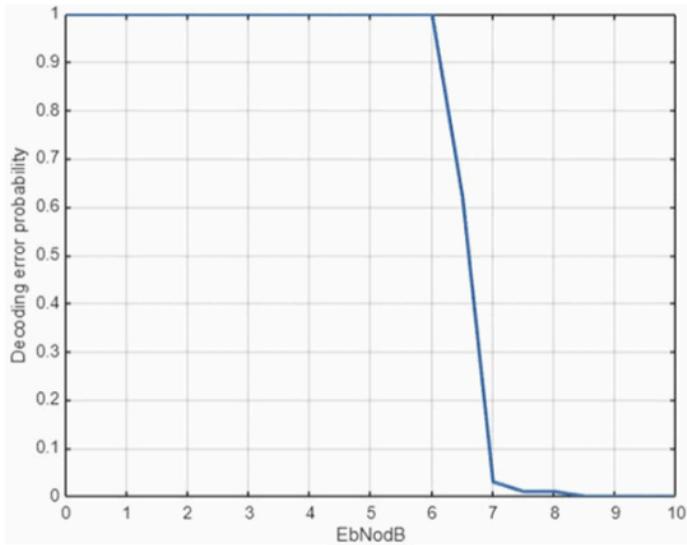
### **Soft Decision Decoding**

**Soft Decision Decoding error probability of Coderate = 1/2**

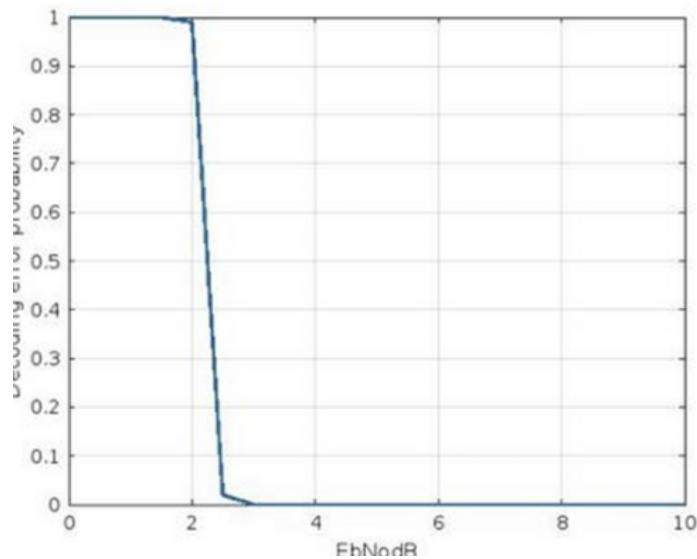


c. For Code Rate:-  $\frac{3}{4}$

### Hard Decision Decoding

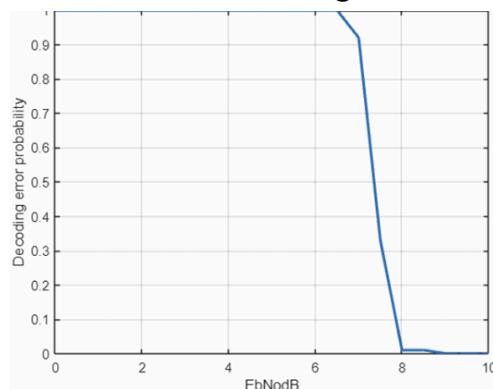


### Soft Decision Decoding

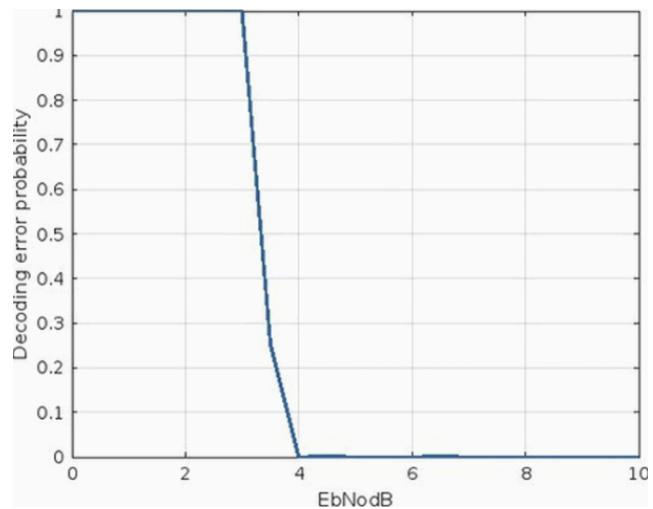


d. For Code Rate:- %

#### Hard Decision Decoding

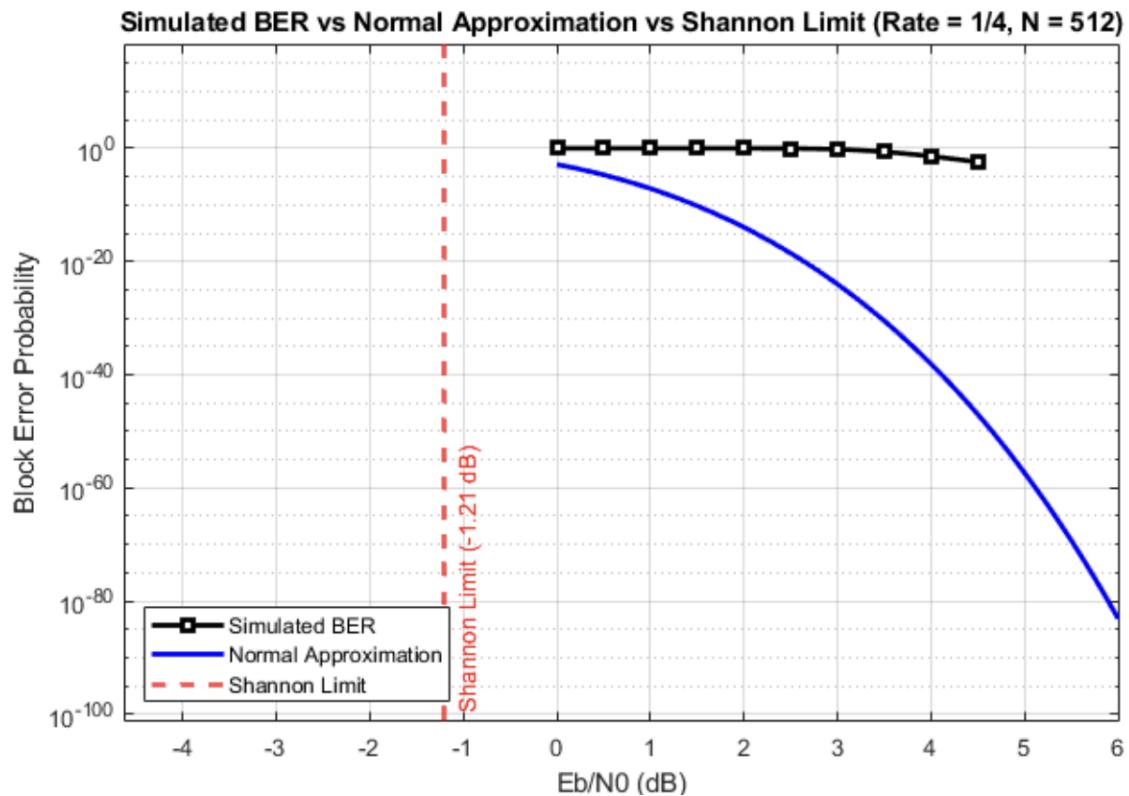


#### Soft Decision Decoding



## Normal Approximation and Shannon's Limit

To evaluate the performance of our LDPC decoder, we compared the simulated results against two theoretical benchmarks: the **Normal Approximation (NA)** and the **Shannon Limit**. These benchmarks are essential for estimating error performance in regimes where running extensive simulations becomes impractical, especially for very low error rates. For instance, with a block length of  $N=512$ , the smallest detectable error rate is around 0.2%, limiting our ability to observe lower BERs directly through simulation.



The Normal Approximation estimates block error probability for finite-length codes by incorporating channel capacity, code rate, and noise variability through a Q-function expression. It offers a realistic prediction of performance where simulation data is sparse. In contrast, the Shannon Limit sets a fundamental lower bound on the required Eb/No for reliable communication, assuming infinite-length codes and ideal conditions. Together, these tools provide a theoretical reference to validate the performance of practical LDPC decoders.

## Appendix A

1) Calculation of the intrinsic LLR:

$$\begin{aligned} \frac{P(C_i=0|r_i)}{P(C_i=1|r_i)} &= \frac{f(r_i|C_i=0)}{f(r_i|C_i=1)} \\ &= \frac{\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_i-1)^2}{2\sigma^2}}}{\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_i+1)^2}{2\sigma^2}}} \end{aligned}$$

$$\therefore \text{Likelihood Ratio} = \exp\left(\frac{2r_i}{\sigma^2}\right)$$

$$\therefore \text{LLR} = \frac{2r_i}{\sigma^2} \quad (\text{taking log of both})$$

2) Calculation of output LLR for repetition code  
(N as 3)

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

$$L_i = \log \frac{f(r_1, r_2, r_3 | C_i=0)}{f(r_1, r_2, r_3 | C_i=1)}$$

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

[here  $r_i = 1 + N_i(0, \sigma^2)$ ]

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

here  $L_1$ ,  $L_2$  &  $L_3$  will be equal and

$$\therefore L_i = (r_1 + r_2 + r_3) \quad [\text{We ignore } 2/\sigma^2]$$

- 3) Calculation of output LLR for single parity check code.

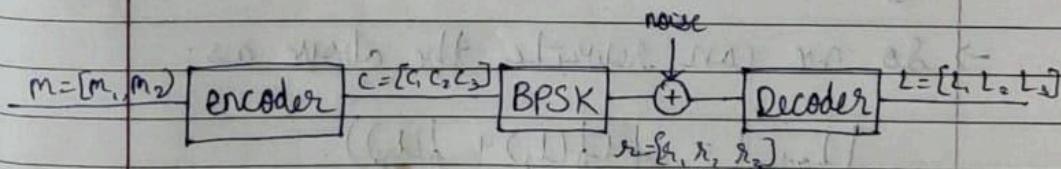
$L_i \Rightarrow$  output LLR for bit  $i$

$l_i \Rightarrow$  input LLR for bit  $i$  (channel LLR)

$l_{\text{ext},i} \Rightarrow$  extrinsic LLR for bit  $i$

$$L_i \Rightarrow l_i + l_{\text{ext},i}$$

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



$$\text{here, } L_1 = l_1 + l_{\text{ext},1}$$

$$L_2 = l_2 + l_{\text{ext},2}$$

$$L_3 = l_3 + l_{\text{ext},3}$$

$$\text{where } l_i = \frac{2}{\sigma^2} r_i \quad (\text{assumed } \frac{2}{\sigma^2} = 1)$$

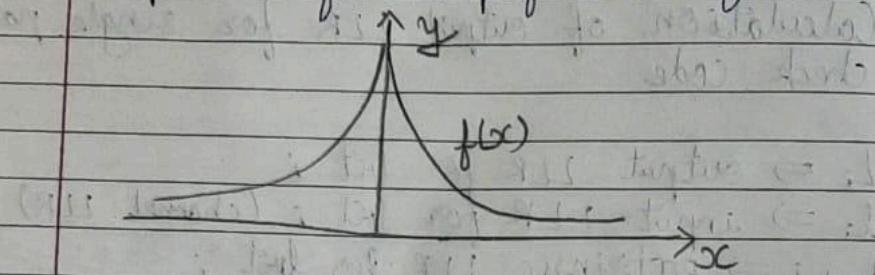
$$\& |l_{\text{ext},i}| = f(f(l_1) + f(l_2)) \quad \text{sgn}(l_{\text{ext},i}) = \text{sgn}(l_1) \\ \text{sgn}(l_1)$$

old page  
no. 0

Page No. \_\_\_\_\_  
Date \_\_\_\_\_

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

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 as:

$$|I_{out,1}| = f(f(I_2) + f(I_3))$$

$$f(I_2) + f(I_3) \approx f(\min(I_2, I_3))$$

$$|I_{out,1}| = f(f(\min(I_2, I_3)))$$

$$\& f = f^{-1}$$

$$\therefore |I_{out,1}| = \min(I_2, I_3)$$

→ This is referred as the Min-Sum Approximation.