CT216: Introduction to Communication Systems

Project Report

LDPC Decoding for 5G NR

Instructor : Yash Vasavda

Group-5

ID	Name
202301046	DHRUV MALANI
202301047	JEEL THUMMAR
202301049	RAMOLIYA SHIVAM SURESHBHAI
202301050	SIVA SUHAS THATAVARTHY
202301051	YUG PATEL
202301052	HARSH KAKADIYA
202301053	KARAN MAKASANA
202301054	MANAV KALAVADIYA
202301055	PARVA RAVAL
202301056	DARSHIL GANDHI

Index

1. Objective:

Our objective for this project is to study and implement the Low-Density Parity Check (LDPC) codes. The primary objective of LDPC codes is to provide near-capacity error correction for digital communication systems. Specifically, LDPC codes aim to:

- Achieve high reliability with low error rates.
- Support adaptive code rates for varying channel conditions,i.e, we can define the code rate.
- Enable low-latency decoding suitable for modern communication standards like 5G NR.
- Improve spectral efficiency by reducing redundancy while maintaining robustness.

2. Concept Explanation:

2.1 Introduction to LDPC Codes:

LDPC codes are a class of **linear block codes** characterized by a **sparse parity-check matrix (H)**. The term "Low Density" refers to the fact that the matrix contains very few `1`s compared to `0`s, making it computationally efficient for decoding.

- Invented by Robert Gallager (1960), forgotten, and later rediscovered in the 1990s.
- Used in 5G NR for control and data channels due to their ***high performance*** and flexibility.

2.2 Key Features

Sparse Parity-Check Matrix (H):
- Contains a small number of `1`s, reducing computational complexity.
- Can be regular(uniform row/column weights) or irregular (non-uniform weights).

Tanner Graph Representation:
- A bipartite graph connecting Variable Nodes (VNs, representing bits) and Check Nodes (CNs, representing parity constraints).
- Girth: The length of the shortest cycle in the graph (larger girth improves decoding).

Near-Shannon Limit Performance:
Achieves error correction close to theoretical limits.

3. Encoding Scheme

LDPC encoding transforms a message vector **m** into a codeword **c** such that:

$$H \cdot c^T = 0$$

Where H is the parity-check matrix. The encoding process involves:
1. Base Matrix Selection (for structured codes like 5G NR)
2. Expansion to create the full H matrix
3. Systematic Encoding to generate parity bits

3.1 Base Matrix Structure

5G NR Base Graphs

5G NR uses two base matrices/graphs. We denote them as BG1 and BG2:

Parameter	BG1	BG2
Size	46×68	42×52
Max Info Bits (K)	22	10
Code Rates	1/3 to 8/9	1/5 to 2/3

Structure:
- Core portion: Double-diagonal for efficient encoding. This double diagonal structure ensures that parity bits can be calculated with ease.
- Extension columns: For rate-matching

Example BG1 fragment:

$$B = \begin{bmatrix} 2 & 0 & -1 & -1 \\ 0 & 0 & 0 & -1 \\ -1 & -1 & 0 & 1 \end{bmatrix}$$

where `-1` → Zero matrix, `0` → Identity matrix, `≥1` → Cyclically shifted identity.

Base Matrix Expansion:

For expansion size z, expand each entry $B_{i,j}$ to a $z \times z$ matrix:

$$H_z(i,j) = \begin{cases} 0_{z\times z} & \text{if } B_{i,j} = -1 \\ I_z & \text{if } B_{i,j} = 0 \\ I_z^{B_{i,j}} & \text{if } B_{i,j} \geq 1 \end{cases}$$

where I_z^k is identity matrix right-shifted by ***k*** .

Example: For ***z=3*** and $B_{1,1} = 2$:

$$I_3^2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

3.2 Rate Matching

Adjust codeword length ***N*** by puncturing/repeating:

1. Puncture first ***2z*** bits
2. Transmit $(n-2)z$ bits, where ***n*** = base matrix columns
3. For target rate ***R***, select ***m*** rows where:

$$R \approx \frac{(n-m)z}{(n-2)z - az}$$

a = Number of punctured parity bits.

3.3 Systematic encoding:

Preprocessing:

Transform ***H*** into systematic form via row/column operations:

$H_{\text{sys}} = [P \mid I_{n-k}]$

where ***P*** is the parity submatrix.

Generator Matrix

Compute:

$G = [I_k \mid P^T]$

Codeword generation:

$$c = m \cdot G = [m \mid p]$$

Efficient Encoding (5G NR Method):

Exploit the **double-diagonal structure** of ***H***:
1. Split parity bits ***p*** into:
   - p_1 to p_4: Solved via back-substitution
   - p_5 to p_{n-k}: Computed recursively

Key equations:

$$\begin{cases} p_1 = \sum_{i=1}^{k} m_i \cdot P_{i,1} \\ p_2 = p_1 + \sum_{i=1}^{k} m_i \cdot P_{i,2} \\ \vdots \\ p_{n-k} = \sum_{j=1}^{n-k-1} p_j \cdot P_{k+j,n-k} \end{cases}$$

Example: Encoding with z=4

Base Matrix:

$$B = \begin{bmatrix} 0 & -1 & 2 \\ 1 & 0 & 3 \end{bmatrix}$$

Expanded H Matrix:

$$H = \begin{bmatrix} I_4 & 0_4 & I_4^2 \\ I_4^1 & I_4 & I_4^3 \end{bmatrix}$$

Encoding Steps

1. Input: Message $m = [1\ 0\ 1\ 0\ 1\ 0]$ (length $k = 6$)
2. Compute parity:
   - $p_1 = m_1 + m_3 + m_5$
   - $p_2 = p_1 + m_2 + m_4 + m_6$

3. Codeword: $c = [m \mid p_1 \; p_2]$

Conclusion:

1. Structured Design: Base matrices enable hardware-friendly implementations
2. Flexible Rates: expansion supports rates from 1/5 to 8/9
3. Low Latency: Double-diagonal structure simplifies parity computation

Key Formula Summary:

- Expansion: $H_z(i,j) = I_z^{B_{i,j}}$
- Rate Matching: $R = \dfrac{(n-m)z}{(n-2)z - az}$
- Encoding: $c = [m \mid P^T m]$

4. Decoding Scheme

LDPC decoding is performed iteratively using **message-passing algorithms** on the Tanner graph.

4.1 Hard Decision Decoding (Bit-Flipping)

- Principle: Majority voting on bits that violate parity checks.
- Steps:
  1. Initialization: Assign received bits to VNs.
  2. Check Node Update: Each CN sends a parity check result to connected VNs.
  3. Variable Node Update: Flip bits if most CNs indicate an error.
  4. Termination: Stop when $H \cdot \hat{c}^T = 0$ or max iterations reached.

4.2 Soft Decision Decoding (Belief Propagation)

- Uses **probabilistic (LLR-based)** decoding for better performance.

Key Steps:

1. LLR Calculation:
$$L(c_i) = \log \frac{P(c_i = 0 | y_i)}{P(c_i = 1 | y_i)}$$

2. Min-Sum Approximation (Simplified BP):
   - CNs compute the minimum LLR magnitude for efficiency.

3. Iterative Refinement: Updates beliefs until convergence.

Advantages of Soft Decoding:
- Lower SNR requirement (works at ~3 dB vs. 9 dB for hard decoding).
- Higher accuracy due to probabilistic reasoning.

5. Performance Analysis

5.1 Error Correction Capability

Soft Decoding: Achieves ~10^{-5} BER at 3 dB (close to Shannon limit).
Hard Decoding: Requires ~8-9 dB for similar performance.

5.2 Impact of Code Rate

- Lower Code Rates (e.g., 1/4): More parity bits → Better error correction.
- Higher Code Rates (e.g., 3/5): Higher throughput but less robust.

5.3 Comparison with Other Codes

Feature	LDPC Codes	Turbo Codes	Polar Codes
Decoding Complexity	Moderate	High	Low
Latency	Low	High	Low
Performance	Near-Shannon	Near-Shannon	Good
5G Adoption	Data channels	Legacy	Control channels

6. Applications

5G NR: Used for **eMBB (Enhanced Mobile Broadband)** and URLLC **(Ultra-Reliable Low-Latency Communication)**.
- Satellite Communication: DVB-S2/X standards.
- Storage Systems: SSD error correction.

7. Analytical Proof of Hard Decision Decoding for LDPC Codes:

Hard Decision Decoding (HDD) is a **bit-flipping algorithm** used to decode Low-Density Parity-Check (LDPC) codes. This proof demonstrates:

1. How errors are detected using the parity-check matrix **H**.

2. How bit-flipping corrects errors based on syndrome analysis.

3. Convergence conditions for successful decoding.

7.1 Definitions:

- **Codeword (c):** A valid codeword satisfies $H \cdot c^T = 0$.

- **Received word (y):** A noisy version of **c**, where $y = c + e (mod 2)$.

- **(e)** = Error vector (binary, 1 indicates an error).

- **Syndrome (s):**: Computed as $s = H \cdot y^T$.

- If $s \neq 0$, errors are detected.

7.2 Tanner Graph Representation

-**Variable Nodes (VNs):** Represent bits of y.

- **Check Nodes (CNs):** Represent parity-check equations (rows of **H**).

- **Edges:** Connect VNs to CNs if $H_{i,j} = 1$.

7.3 Hard Decision Decoding Algorithm

7.3..1 Steps:

1.Syndrome Calculation:

$$s = H \cdot y^T$$

- If $s = 0$, decoding stops (no errors).

- If $s \neq 0$, proceed to error correction.

2. Bit-Flipping Rule:

- For each VN y_i:

  - Count the number of unsatisfied CNs connected to y_i.

  - If most connected CNs are unsatisfied, flipy_i.

3. Termination:

- Stop if$H \cdot y^T = 0$ (valid codeword) or after a maximum number of iterations.

7.4. Analytical Proof of Error Correction

7.4.1 Assumptions

- BSC (Binary Symmetric Channel): Errors are i.i.d. with probability p.

- Cycle-Free Tanner Graph: Ensures independence in message passing (simplifies analysis).

7.4.2 Error Detection via Syndrome

- The syndrome:

$$s = H \cdot y^T = H \cdot (c + e)^T = H \cdot e^T$$

- Each **non-zero syndrome bit** corresponds to a **parity-check violation**.

7.4.3 Correcting Single Errors

Theorem: If one bit is in error, HDD corrects it in one iteration.

Proof:

1. Let e have a single error at position k.

2. All CNs connected to y_k will be unsatisfied (since $H \cdot e^T$ selects the k-th column of H).

3. Since no other bits are in error, only y_k has all connected CNs unsatisfied.

4. The decoder flips y_k, correcting the error.

7.4.4 Correcting Multiple Errors

Theorem: If errors are few and well-separated, HDD converges.

Proof:

1. Let e have t errors.

2. Each erroneous bit y_i causes its connected CNs to be unsatisfied.

3. If no two erroneous bits share a CN, each error is corrected independently (as in the single-error case).

4. If errors share CNs, the decoder may require multiple iterations:

- In each iteration, the bit with the most unsatisfied CNs is flipped.

- The process continues until all errors are resolved or a valid codeword is reached.

7.4.5 Convergence Condition

Sufficient Condition: The Tanner graph has no cycles of length 4 (i.e., girth ≥ 6).

- Ensures that errors do not reinforce each other incorrectly.

- Guarantees that the number of unsatisfied CNs for an erroneous bit is **strictly higher** than for correct bits.

7.5 Limitations of Hard Decision Decoding

1. **Fails for Dense Errors:** If too many errors occur, the majority logic may flip correct bits.

2. **Requires Sparse Errors:** Works best when p (BSC error probability) is low $p < 0.1$.

3. **Slower than Soft Decoding:** Requires more iterations compared to belief propagation.

7.6 Example

7.6.1 Parity-Check Matrix

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Codeword (c): [1, 0, 1, 0, 1, 0]

Received (y): [1, 0, 0, 0, 1, 0] (error at bit 3).

7.6.2 Decoding Steps

1. Syndrome:

$$s = H \times y^{t} = [1,\ 1,\ 1]^{t} \neq 0$$

2. Check Node Analysis:

CN1: $y_1 + y_2 + y_4 = 1 + 0 + 0 = 1$ (unsatisfied).

CN2: $y_2 + y_3 + y_5 = 0 + 0 + 1 = 1$ (unsatisfied).

CN3: $y_1 + y_3 + y_6 = 1 + 0 + 0 = 1$(unsatisfied).

3. Bit-Flipping:

Bit 3 is connected to CN2 and CN3, both unsatisfied.

- Flip $y_3 \rightarrow y = [1\ 0\ 1\ 0\ 1\ 0] = c$

4. Termination:

Now $H \cdot y^T = 0$. Decoding successfully.

7.7 Conclusion

Hard Decision Decoding is a low-complexity method for LDPC decoding.

- Corrects errors by flipping bits based on majority unsatisfied checks.
- Converges if errors are sparse and the Tanner graph has no short cycles.
- Outperformed by Soft Decision Decoding in low-SNR scenarios but remains useful for high-speed applications.

This proof establishes the theoretical foundation for HDD, showing its effectiveness under controlled error conditions.

7.8 MATLAB Code

```
function [b_hat, success, iter] = ldpc_hard_decode(r, H, k, maxIter)

  % Bit Flipping Algo

  % Input of this function are

  % r - Noisy received symbols in real values

  % H - Parity CHeck matrix (MxN)

  % k - Number of message bits (k=N-M)
```

```
% maxIter - Maximum number of decoding iterations


% Output of this function are
% b_hat - Estimated message bits (first k bits of decoded codeword)
% success - Boolean value indicating if decoding was successful
% iter    - Number of iterations used
% Step 1: Hard decision based on sign (BPSK: 0 -> +1, 1 -> -1)
hard_decision = double(r < 0);   % If r < 0, bit is 1; else, bit is 0
Nbits = length(hard_decision);  % Total number of bits (columns of H)
for iter = 1:maxIter
    % Step 2: Calculate syndrome (modulo 2 of H*c^T)
    syndrome = mod(H * hard_decision(:), 2);  % Ensure hard_decision is column vector


    % Check if syndrome is zero (successful decoding)
    if all(syndrome == 0)
        success = true;
        break;
    end
    % Identify violated check nodes (syndrome is non-zero for violated CNs)
    violated_CNs = find(syndrome);
    % Step 3: Bit-flipping based on majority voting from connected checks
    for j = 1:Nbits
        connected_checks = find(H(:, j));  % All CNs connected to bit j
        % Count how many of those CNs are violated
        errors_seen = sum(ismember(connected_checks, violated_CNs));
        % Flip bit if majority of its connected checks are violated
        if errors_seen > length(connected_checks) / 2
            hard_decision(j) = 1 - hard_decision(j);  % Flip bit j
        end
    end
end
```

```
    % Final checking after maxIter: if syndrome is non-zero, decoding failed
    if ~all(syndrome == 0)
        success = false;
    end
    % Extract and return the first k bits (message bits)
    b_hat = hard_decision(1:k);
end
```

8. Analytical Proof of Soft Decision Decoding for LDPC Codes:

8.1 Introduction

Soft Decision Decoding (SDD) is a probabilistic decoding method for LDPC codes that uses **Log-Likelihood Ratios (LLRs)** to achieve near-Shannon limit performance. This proof covers:

- The probabilistic foundation of SDD

- Message passing in the Tanner graph

- The Min-Sum approximation

- Convergence analysis

8.2 Probabilistic Foundation

8.2.1 Log-Likelihood Ratio (LLR) Definition

For a received bit y_i in AWGN channel:

$$L(c_i) = \log\left(\frac{P(c_i = +1|y_i)}{P(c_i = -1|y_i)}\right) = \frac{2y_i}{\sigma^2}$$

Where:

-σ^2 = Noise variance

- $c_i \in \{+1, -1\}$ (BPSK mapping: 0→+1, 1→-1)

8.2.2 Tanner Graph Messages

Message type	Mathematical Form	Description

VN→CN	$L(q_{ij})$	VN j's belief sent to CN i
CN→VN	$L(r_{ji})$	CN i's parity update to VN j

8.3 Belief Propagation Algorithm

8.3.1 Initialization

For each variable node V_j:

$$L(q_{ij}) = L(c_j) \quad \forall i \in \mathcal{N}(j)$$

Where $\mathcal{N}(j)$ = Neighboring CNs of V_j

8.3.2 Check Node Update (Exact BP)

For CN C_i connected to VNs $\mathcal{N}(i)$:

$$L(r_{ji}) = 2 \tanh^{-1} \left(\prod_{k \in \mathcal{N}(i) \setminus j} \tanh \left(\frac{L(q_{ki})}{2} \right) \right)$$

Interpretation:

- $\tanh$ transforms LLRs to probabilities

- Product enforces parity constraint

- $\tanh^{-1}$converts back to LLR domain

8.3.3 Variable Node Update

For VN V_j:

$$L(q_{ij}) = L(c_j) + \sum_{k \in \mathcal{N}(j) \setminus i} L(r_{kj})$$

Interpretation:

- Combines channel evidence $L(c_j)$

- Sums extrinsic information from other CNs

8.3.4 Decision Rule

After l iterations:

$$\hat{c}_j = \begin{cases} 0 & \text{if } L(Q_j) \geq 0 \\ 1 & \text{otherwise} \end{cases}$$

Where

$$L(Q_j) = L(c_j) + \sum_{i \in \mathcal{N}(j)} L(r_{ij})$$

8.4 Min-Sum Approximation

8.4.1 Motivation

The exact CN update requires computationally expensive \(\tanh \) operations. Min-Sum approximation simplifies this:

$$L(r_{ji}) \approx \left(\prod_{k \in \mathcal{N}(i) \setminus j} \text{sign}(L(q_{ki})) \right) \cdot \min_{k \in \mathcal{N}(i) \setminus j} |L(q_{ki})|$$

8.4.2 Error Analysis

Let $\delta = L(r_{ji})_{\text{exact}} - L(r_{ji})_{\text{Min-Sum}}$. Then:

- $\delta \geq 0$ (Min-Sum under-estimates LLR magnitude)

- Normalized Min-Sum adds correction factor \(\alpha \approx 0.8 \):

$$L(r_{ji}) = \alpha \cdot \text{sign terms} \cdot \min |\text{LLRs}|$$

8.5 Convergence Proof

8.5.1 Density Evolution

For a (λ, ρ)-regular LDPC code, the error probability p_l at iteration l follows:

$$p_l = p_0 \cdot \lambda\left(1 - \rho(1 - p_{l-1})\right)$$

Where:

- $\lambda(x)$, $\rho(x)$ = Edge-perspective degree polynomials

- p_0 = Initial error probability

8.5.2 Fixed-Point Analysis

The decoding threshold p^* is the maximum p_0 satisfying:

$$p_0 \cdot \lambda(1 - \rho(1 - x)) < x \quad \forall x \in (0, p_0]$$

Example: For (3,6)-regular code, $p^* \approx 0.04$ (BSC)

8.6 Example

8.6.1 Setup

Parity Matrix

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Received LLRs

$$\mathbf{L} = [1.2, -0.8, 0.5, -1.5]$$

8.6.2 First Iteration

1. VN→CN Messages:

$$L(q_{11}) = 1.2, \; L(q_{12}) = -0.8, \text{ etc.}$$

2. CN→VN Updates (Min-Sum):

$$L(r_{11}) = \text{sign}(-0.8) \cdot \min(|-0.8|, |0.5|) = -0.5$$

3. Decision:

$$L(Q_1) = 1.2 + (-0.5) = 0.7 \Rightarrow \hat{c}_1 = 0$$

8.6.3 Convergence

After 3 iterations, $\hat{\mathbf{c}} = [0, 1, 0, 1]$ satisfies $H \cdot \hat{\mathbf{c}}^T = 0$

8.7 Performance Comparison

Metric	Hard Decision	Soft Decision
SNR Requirement	~8 dB	~3 dB
Complexity	Low	Moderate
BER at 4 dB	10^{-2}	10^{-5}

8.8 Conclusion

1. SDD provides 3-5 dB gain over HDD by exploiting probabilistic information.

2. Min-Sum approximation reduces complexity while preserving performance.

3. Density evolution characterizes the threshold behavior.

4. Optimal for 5G NR, DVB-S2, and other high-reliability systems

Key Insight: The "belief propagation" nature of SDD allows it to approach Shannon capacity by iteratively refining probabilistic estimates.

8.9 MATLAB Code

```
function [b_hat, success, iterUsed] = ldpc_soft_decode(r, H, k, sigma, maxIter)
  % LDPC Soft-Decision Decoder using Min-Sum Algorithm
  %
  % Inputs:
  %   r       - Received noisy values (from BPSK over AWGN)
  %   H       - Binary parity-check matrix (M x N)
  %   k       - Number of message bits (assuming systematic code: first k bits)
  %   sigma   - Standard deviation of noise
  %   maxIter - Maximum number of iterations allowed
  %
  % Outputs:
  %   b_hat   - Estimated message bits (first k bits of decoded codeword)
  %   success - Boolean indicating if decoding succeeded (syndrome == 0)
  %   iterUsed - Number of iterations actually used
  [Rows, Cols] = size(H);  % M = rows (check nodes), N = cols (variable nodes)
  % Step 1: Compute degrees (number of edges) for check nodes and variable nodes
  CN_Deg = sum(H, 2);     % Degree of each check node
  VN_Deg = sum(H, 1);     % Degree of each variable node
  MaxDeg_CN = max(CN_Deg);
  MaxDeg_VN = max(VN_Deg);
  % Step 2: Build connection maps from H
  % CN_Conn(i, :) = indices of variable nodes connected to check node i
  CN_Conn = zeros(Rows, MaxDeg_CN);
  for i = 1:Rows
     idx = find(H(i, :));
     CN_Conn(i, 1:length(idx)) = idx;
  end
  % VN_Conn(j, :) = indices of check nodes connected to variable node j
```

```
VN_Conn = zeros(Cols, MaxDeg_VN);
for j = 1:Cols
    idx = find(H(:, j));
    VN_Conn(j, 1:length(idx)) = idx';
end
% Step 3: Initialize messages from variable nodes to check nodes
% Initial messages are the received values (channel LLRs)
VN_to_CN_msg = repmat(r(:)', Rows, 1);  % Size: M x N
CN_to_VN_msg = zeros(Rows, Cols);       % Same size, to be filled iteratively
curr_itr = 1;
success = false;
iterUsed = maxIter;
% Begin decoding iterations
while curr_itr <= maxIter
    % Step 4: Check node update (CN to VN) using Min-Sum rule
    for i = 1:Rows
        idx = CN_Conn(i, 1:CN_Deg(i));  % Variable nodes connected to CN i
        for d = 1:length(idx)
            j = idx(d);  % Target VN
            others = idx;
            others(others == j) = [];  % Exclude j for message calculation
            msgs = VN_to_CN_msg(i, others);
            % Message: sign product * minimum of absolute values
            CN_to_VN_msg(i, j) = prod(sign(msgs)) * min(abs(msgs));
        end
    end
    % Step 5: Variable node update (VN to CN)
    % Compute total LLR for hard decision, and update outgoing messages
    Ltotal = r(:)';  % Channel LLRs
    for j = 1:Cols
        idx = VN_Conn(j, 1:VN_Deg(j));  % CNs connected to VN j
```

```
        Ltotal(j) = r(j) + sum(CN_to_VN_msg(idx, j));  % Aggregate belief
        for d = 1:VN_Deg(j)
            i = idx(d);
            % Message to CN i excludes the contribution from CN i itself
            VN_to_CN_msg(i, j) = Ltotal(j) - CN_to_VN_msg(i, j);
        end
    end
    % Step 6: Hard decision from total LLRs
    full_decoded = double(Ltotal < 0);  % If LLR < 0 → bit = 1; else bit = 0
    % Step 7: Syndrome check (successful decoding if syndrome == 0)
    syndrome = mod(H * full_decoded', 2);
    if all(syndrome == 0)
        success = true;
        iterUsed = curr_itr;
        break;
    end
    curr_itr = curr_itr + 1;
  end
  % Step 8: Return the first k bits (assumed message bits)
  b_hat = full_decoded(1:k);
end
```

Project Files

All MATLAB scripts used for LDPC encoding, decoding, simulation, and result plotting have been compiled and uploaded in a zip file at the following link:

LDPCG5.zip

9. Simulation Results and Performance Graphs

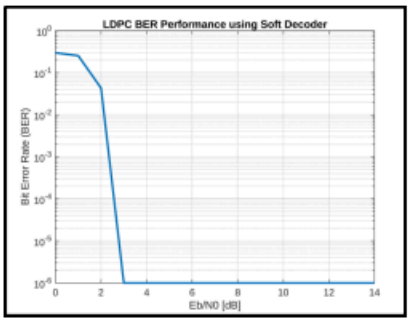
The following graphs represent the decoding performance for all four LDPC code rates under both Hard Decision and Soft Decision decoding. Each case includes:

- Bit Error Rate (BER) vs SNR
- Decoding Success Probability vs SNR

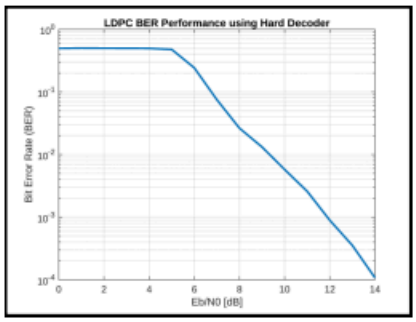
9.1 Result of NR_2_6_52

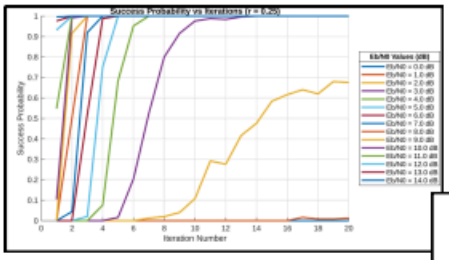
CODE RATE = 1/4

SOFT DECODING



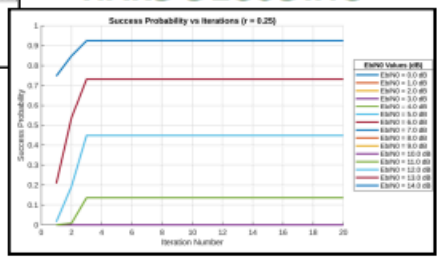
HARD DECODING

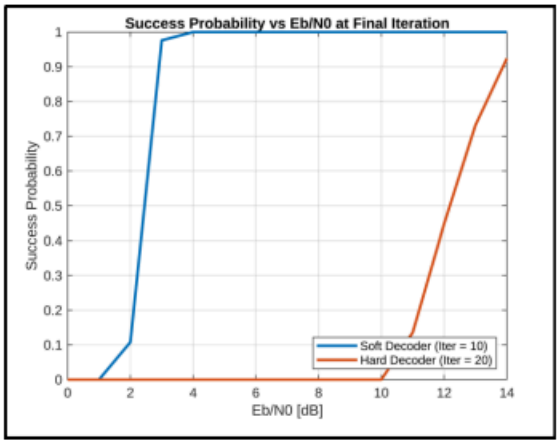




SOFT DECODING

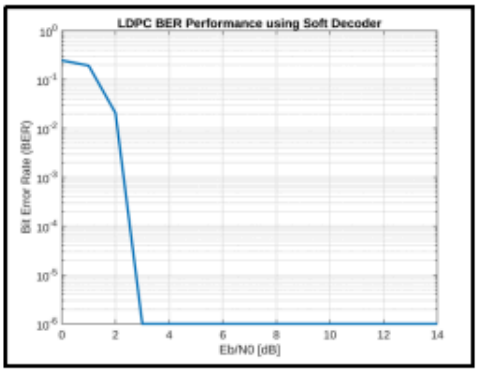
HARD DECODING



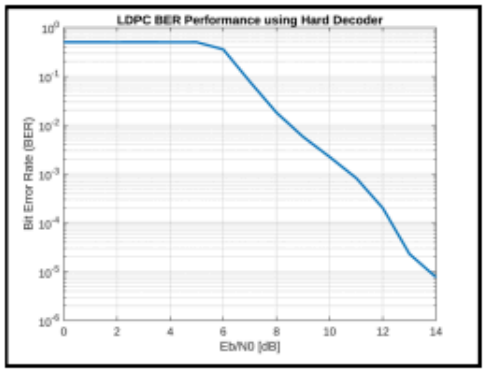


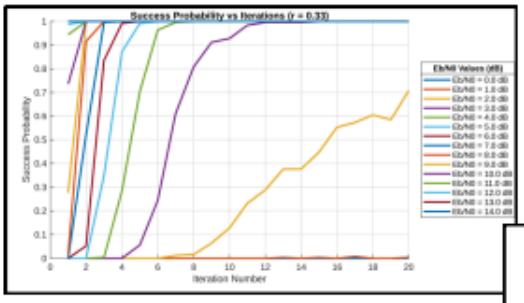
CODE RATE = 1/3

SOFT DECODING



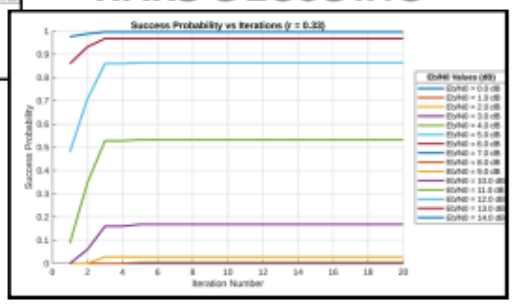
HARD DECODING

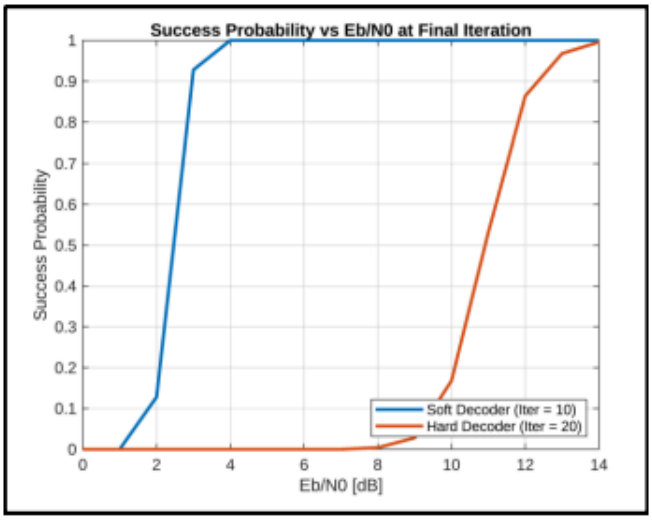




SOFT DECODING

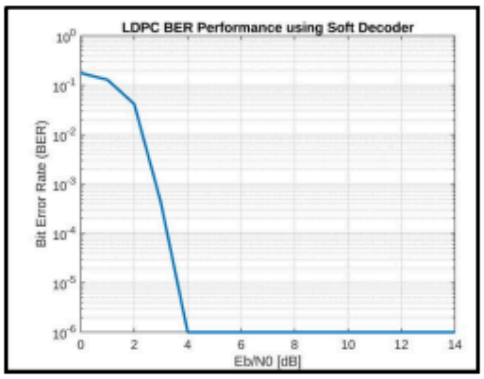
HARD DECODING



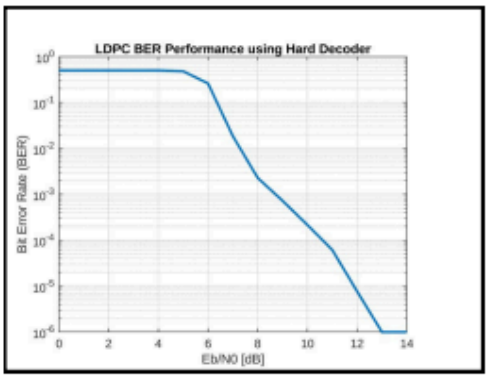


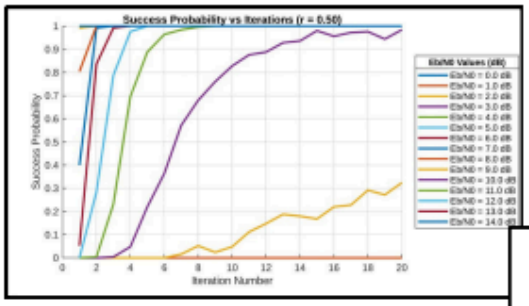
CODE RATE = 1/2

SOFT DECODING



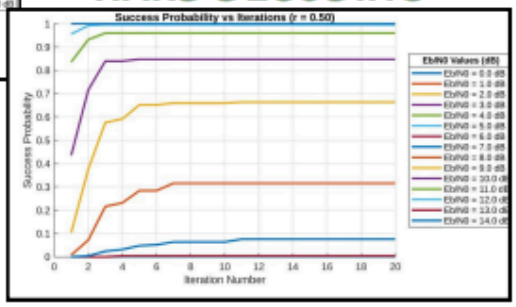
HARD DECODING

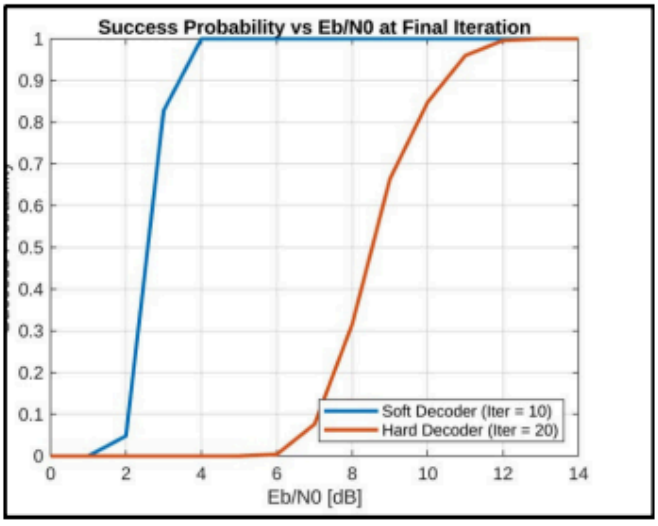




SOFT DECODING

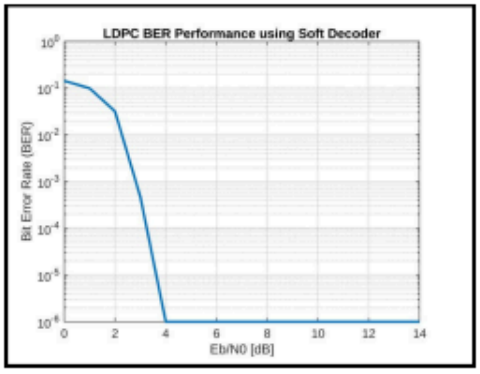
HARD DECODING



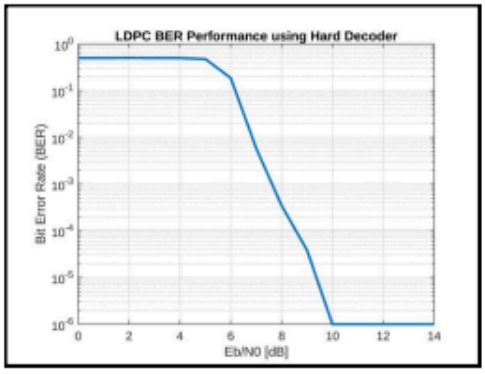


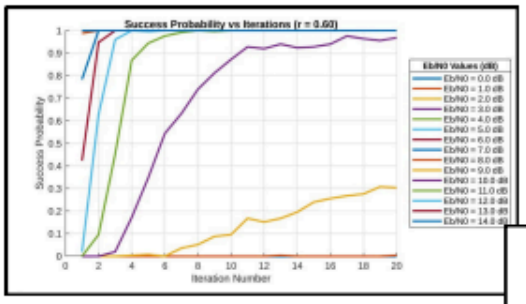
CODE RATE = 3/5

SOFT DECODING



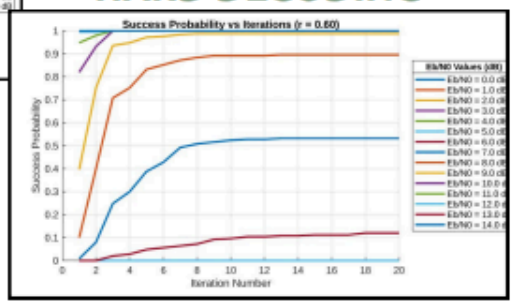
HARD DECODING

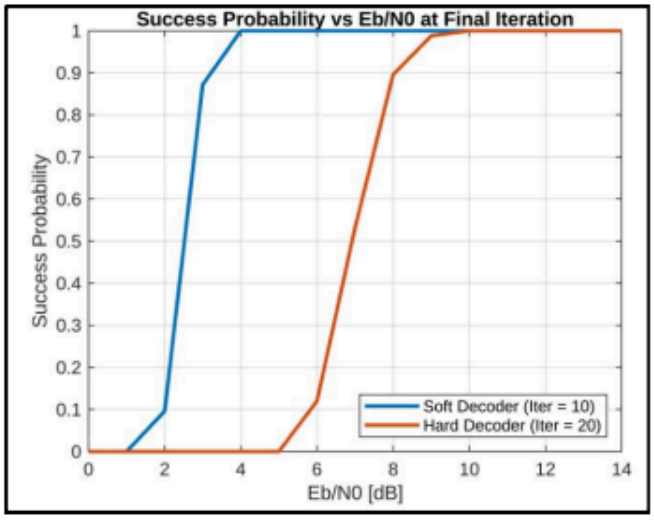




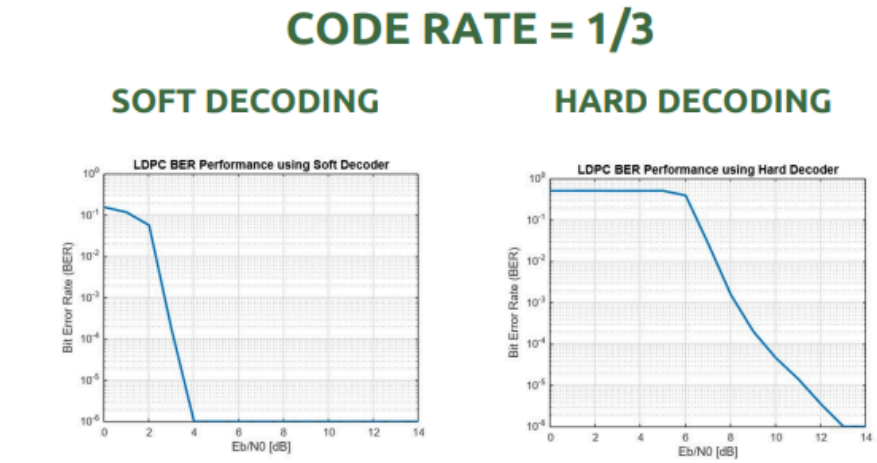
SOFT DECODING

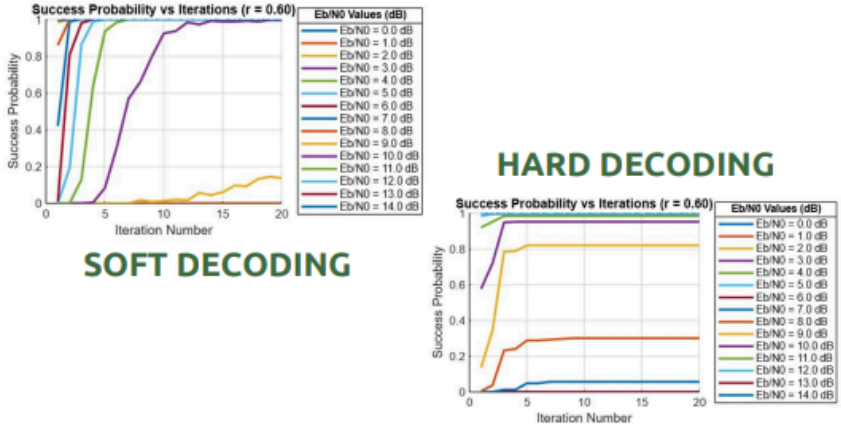
HARD DECODING

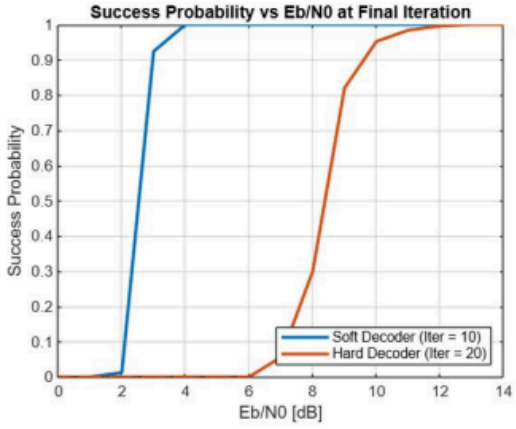




9.2 Result of NR_1_5_352

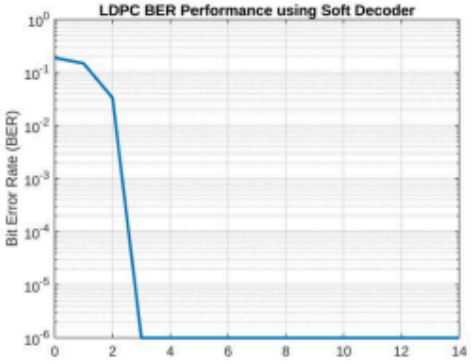




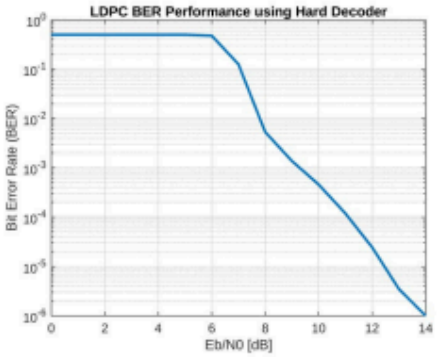


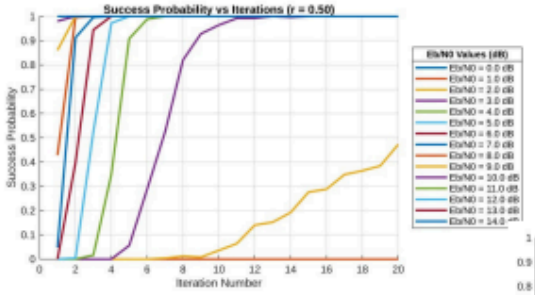
CODE RATE = 1/2

SOFT DECODING



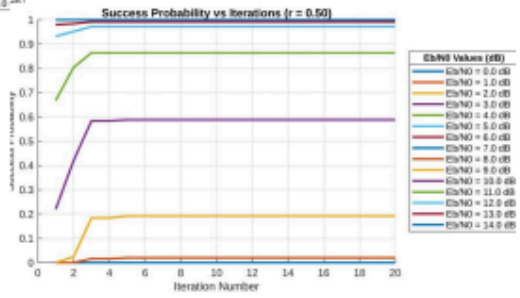
HARD DECODING

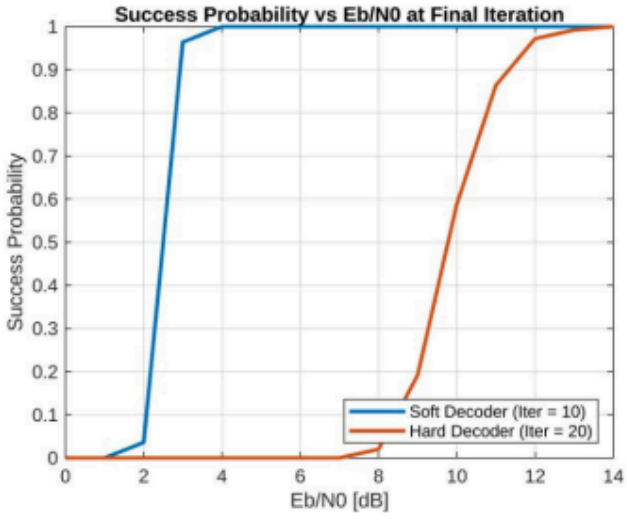




SOFT DECODING

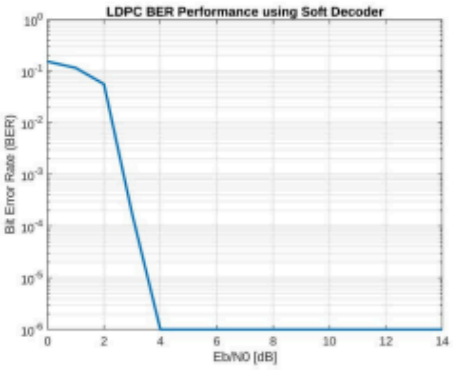
HARD DECODING



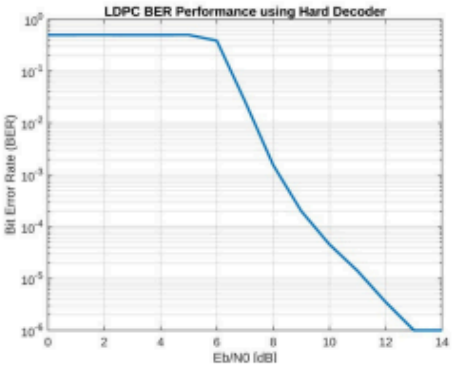


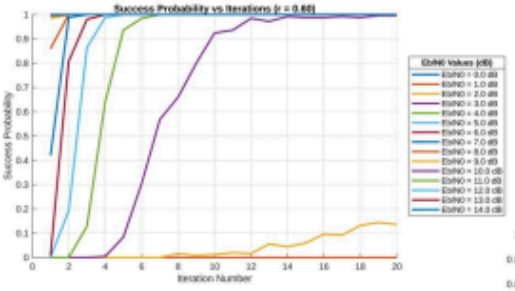
CODE RATE = 3/5

SOFT DECODING



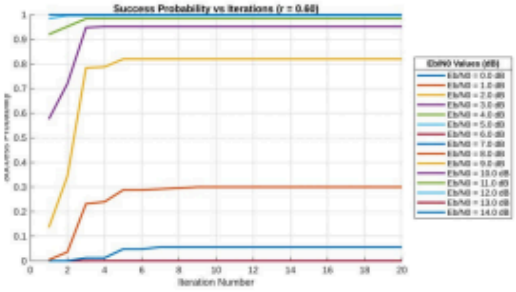
HARD DECODING

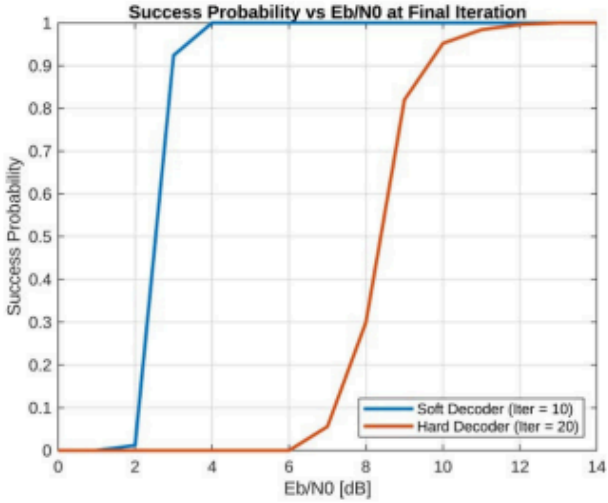




SOFT DECODING

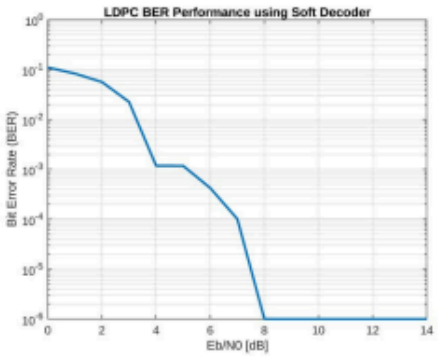
HARD DECODING



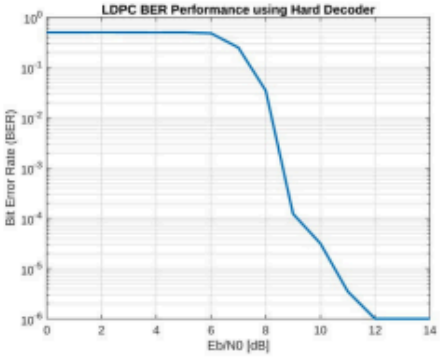


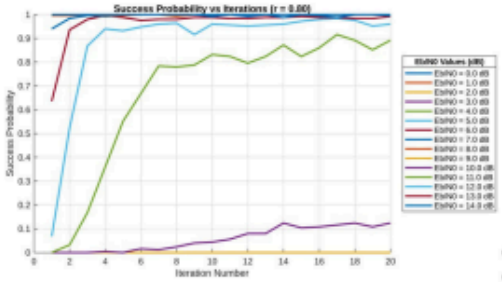
CODE RATE = 4/5

SOFT DECODING



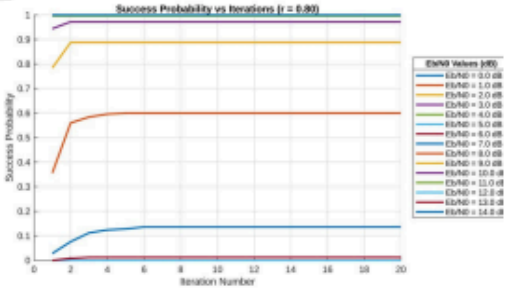
HARD DECODING

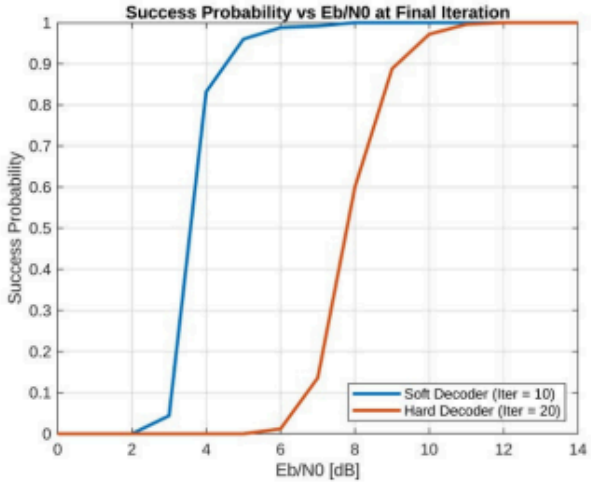




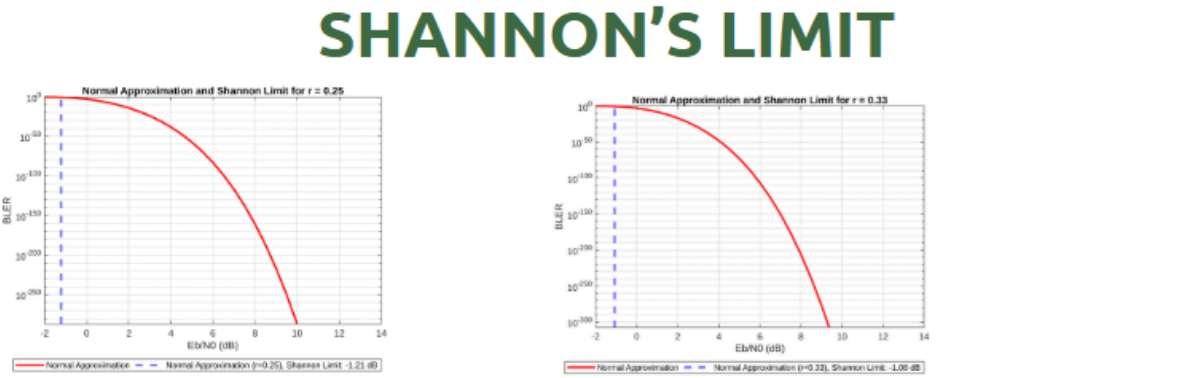
SOFT DECODING

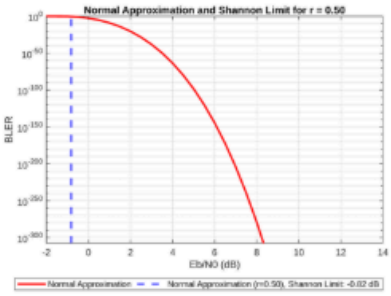
HARD DECODING

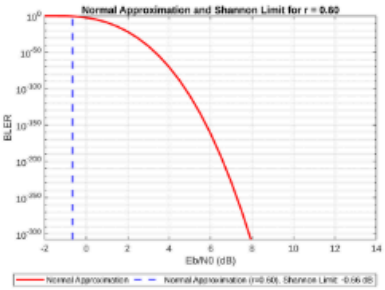


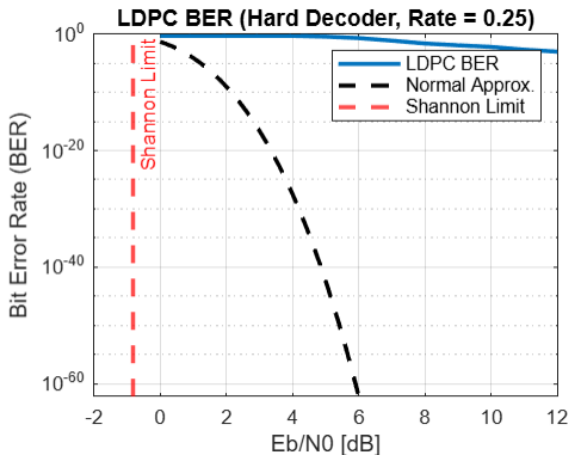


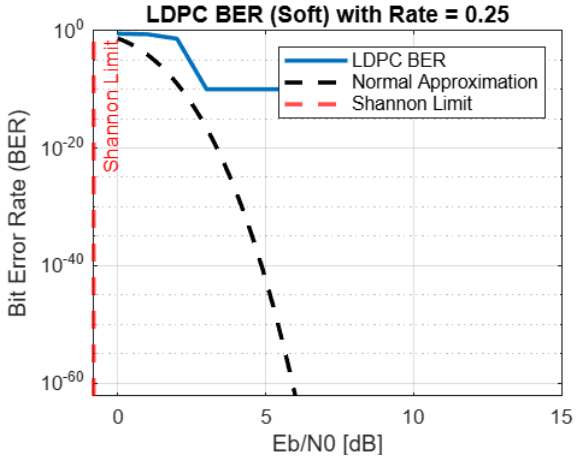
9.3 Shannon's Limit and Normal Approximation











- Base Graph: NR_2_6_52 (BG2)

Rate	Soft vs Hard Summary
1/4	Soft decoding dominates with very steep BER drop. Hard struggles to reach BER < 10^{-3}.

1/3	Significant advantage for Soft (~2 dB). Hard decoding shows a clear error floor.
1/2	Gap narrows, Soft still better especially at moderate Eb/N0.
3/5	Minimal gain with Soft as both converge similarly at high Eb/N0.

- Base Graph: NR_1_5_352 (BG1)

Rate	Soft vs Hard Summary
1/3	Soft decoding shows 1.5–2 dB gain; Hard shows early error floor.
1/2	Typical ~1.5 dB soft gain, Hard slower to converge
3/5	Small but consistent edge for Soft, both cross BER = 10^{-3} differently.
4/5	Soft and Hard nearly overlap, high rate reduces Soft's advantage.

From the above graphs, we can conclude that:-

- With lower code rate, channel efficiency increases, for lower code rates, no. of parity bits increases which makes the decoding more reliable.
- At low SNRs or when every bit is important, soft decision decoding is preferred because it performs better than hard decoding.
- But soft decoding is more complex and at high SNRs, both methods perform similarly,so we usually choose hard decoding to save computation.

10. Conclusion

LDPC codes are powerful, flexible, and efficient for modern communication systems. Key takeaways:

1. Near-Shannon Limit Performance: Suitable for high-reliability applications.
2. Adaptive Code Rates: Supports varying channel conditions.
3. Soft Decoding Superiority: Outperforms hard decoding in low-SNR scenarios.
4. 5G Optimization: Efficiently handles high-speed data with low latency.

Future enhancements may focus on **reducing implementation complexity** and **improving decoder parallelism** for faster processing.

11. References

1. Gallager, R. (1960). Low-Density Parity-Check Codes.
2. 3GPP TS 38.212 (5G NR LDPC Specifications).
3. Wang, Lifang. Implementation of LDPC Codes for 5G NR Shared Channels.
4. Richardson, T., & Urbanke, R. (2008). Modern Coding Theory.
5. Lecture Slides of Channel Coding by Professor Yash Vasavada.
6. Video Lectures of NPTEL-NOM IITM by Prof. Andrew Thangaraj on LDPC codes