# LDPC_CODES
# GRP:17

**Group Members:**

| Student ID | **Name** |
|---|---|
| 202301179 | VAISHNAV POORVA MUKUL |
| 202301180 | VALA SIDDHARTHSINH HARIBHAI |
| 202301181 | PATEL PREXA JAYANTKUMAR |
| 202301182 | SORATHIYA BRINDABEN DILIPBHAI |
| 202301183 | MAKAVANA RAJKUMAR BHAMABHAI |
| 202301186 | DONDA VED VIPULBHAI |
| 202301187 | BHATIA NISARG MAHESHKUMAR |
| 202301189 | DHOLA KRISH HARESHBHAI |
| 202301190 | VALA HARDIK VINUBHAI |

## Project objective:

This project focuses on implementing LDPC codes, which provide near-capacity error correction for digital communication. They offer low error rates, adaptable code rates, fast decoding for standards like 5G NR, and improved spectral efficiency through reduced redundancy

# LDPC Codes (Low-Density Parity-Check Codes)

## Introduction to LDPC:

LDPC codes are linear block error-correcting codes defined by a sparse parity-check matrix. In practice, this means the number of 1's in the parity matrix grows only linearly with block size. They were introduced by Gallager (1960) and use iterative message-passing (belief-propagation) decoding.

## Key properties:

LDPC codes are capacity-approaching for large block lengths – their performance can come within a fraction of a dB of the Shannon limit. Their sparse (low-density contains a small number of `1`s.)This makes decoding algorithms highly parallelizable and efficient. For example, LDPC decoders break the overall decoding into many small, independent computations, enabling very high throughput in hardware.

## Usage:

LDPC codes have been widely adopted in modern systems  (Wi-Fi 802.11, DVB-S2/T2, etc.) and are now used in 5G NR for data channels. They offer excellent error correction (low error rates), especially at high code rates and long block lengths, with very low error floors (undetectable error  rates around BLER≈10^–5 or below).

## Why LDPC Codes in 5G NR

- **Data-channel coding scheme:** 5G NR uses LDPC codes for all data (PDSCH/PUSCH) channels, while polar codes are used for control channels. The 3GPP specification defines two quasi-cyclic LDPC base graphs that cover a wide range of block lengths and rates. This structured design supports rate compatibility and incremental-redundancy HARQ: large transport blocks can be segmented into smaller LDPC code blocks, and parity bits can be punctured or extended as needed for any code rate

- **High-throughput, low-latency decoding:** 5G targets very high data rates (e.g. up to ~20 Gb/s DL). LDPC decoders are inherently parallelizable: the sparse Tanner-graph structure lets many check-node operations run concurrently. In hardware this yields extremely high area-throughput efficiency (Gb/s per chip area) and very high peak throughput. As a result, LDPC decoding can meet 5G's low-latency, high-rate requirements (e.g. >10–20 Gb/s) more easily than older codes.

- **Excellent error-rate performance:** The 5G LDPC codes (with optimized base graphs) provide strong coding gain, especially at higher code rates. They achieve very low frame/packet error rates with no prominent error floor down to BLER≈10^–5, which is crucial for ultra-reliable scenarios. (In contrast, turbo codes tend to exhibit higher error floors at high SNR.) LDPC's near-capacity performance ensures robust link reliability for eMBB and URLLC use cases.

## 4G LTE Limitations Addressed by LDPC

- **Turbo-code throughput and complexity:** 4G LTE relies on turbo codes for data channels, which achieve good performance but require serial (SISO) iterative decoding. Turbo decoders are harder to parallelize, so at very high throughput their latency and complexity grow quickly. In contrast, LDPC decoders decompose into many small, independent operations, enabling much greater parallelism. This yields substantially higher area-throughput and lower latency at high code rates.

- **Flexible block sizes and rates:** LTE turbo codes had fixed block sizes (max ~6144 bits) and supported a limited set of code rates. 5G eMBB requires supporting much larger transport blocks and a wide range of rates (for IR-HARQ). The quasi-cyclic LDPC design (two base graphs) lets 5G handle arbitrarily large blocks (via segmentation) and easily change rates by puncturing/extension. This flexibility was not possible with LTE's codes without complex concatenation or overhead.

- **High-rate performance and error floor:** LDPC codes outperform turbo codes at high coding rates. In LTE the highest-rate turbo modes showed diminishing returns (higher error floors). By contrast, 5G LDPC codes maintain low error floors (≈10^–5) even at high rates. This ensures that very high data-rate links can still meet reliability targets (important for eMBB).

- **HARQ and parallel decoding:** The 5G LDPC design natively supports incremental-redundancy HARQ with low re-encoding/decoding overhead. The 4G turbo required more complex rate-matching and was less efficient at parallel re-transmissions. LDPC's structured parity-check matrices make both encoding and decoding hardware-friendly, reducing the energy/operation count needed per bit to achieve a given BLER.

# Encoding in LDPC Codes:

To encode the message vector for transmission, redundant bits (or parity bits) are added, and the full codeword is transmitted. For the 5G LDPC codes, two base graphs, BG1 and BG2, are defined, and a parity check matrix, H, can be generated from each one of them by replacing each element of the base matrix with a z x z matrix, where z is the expansion factor of the base graph. The resultant parity check matrix is used for calculating the parity bits. For an m x n matrix with an expansion factor z, each element is -1 ≤k ≤ z-1. Each -1 is replaced by a $zero\ matrix$ ($\emptyset$), 0 is replaced by an identity matrix I, and each 1kz-1 is replaced by a matrix Ik, which denotes the identity matrix circularly shifted to the right by k. The resultant parity check matrix H, of size (m × z) × (n × z), is generated. The codeword matrix, containing the message bits and the parity bits, on multiplying with the H matrix, should result in a null matrix.

$$\text{i.e., } H\,C^{T} = \emptyset\text{(zero matrix)}$$

The structure of base matrices is such that the resultant parity check matrix has a double diagonal structure, which makes it easier to solve for the parity blocks of the codeword to be transmitted. This is explained with the help of a smaller base graph on page 13 of the PPT.

## Starter Code:

In the starter code, the base matrix is used in a similar way to encode the message bits for transmission. Instead of explicitly calculating the parity matrix from the base graph and using it to generate the codeword, the base matrix is used to calculate the vector C directly.

As shown here, every $c_i$ & $p_j$ represents a block of z bits. In order to find each of the parity bits, we can iterate through the rows of the base matrix B, and in each iteration, we iterate through the required columns of B. For every B(i,j), we take the j-th block of the z message bits and right-shift the bits of the block according to the value of B(i,j). This result is computed for each of the four rows and then XORed and stored in the vector 'temp.'

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-m} \\ p_1 \\ \vdots \\ p_m \end{bmatrix}$$

A part of the starter code is shown below, where temp, a $z \times 1$ vector, is used to store the following result:

$$\sum_{i=1}^{4} \sum_{j=1}^{n-m} (I_{B(i,j)} \cdot (msg\_j))$$

Here msg_j is the jth block of z bits, and [m, n] denotes the dimension of the base matrix. $I_{B(i,j)}$ denotes an identity matrix circularly shifted right by $B(i,j)$.

```
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
```

Then, the shifting value for the parity block p1, which is B(2, n-m+1), is used, and the 1st parity block is calculated as follows:

```
cword((n-m)*z+1:(n-m+1)*z) = mul_sh(temp, z - p1_sh);
```

Here, mul_sh denotes a function that shifts a vector temp circularly right by z-p1_sh.

The presence of a double diagonal structure makes it easier to compute $p_1$ , using the 2nd row and n-m+1 columns of the base matrix. Similarly $p_2$, $p_3$ and $p_4$ can also be found. The remaining parity bits from $p_5$ to $p_m$ can be found by iterating from the 5th row of the base matrix. In this way, a codeword of (n × z) bits are generated.

## Hard Decoding :

### What is it :

Hard-decision decoding is the simplest form of error-correction decoding, in which each received symbol is first forced to a binary value (0 or 1) by thresholding (or by definition of the channel) and then decoded using only those binary values. In the context of LDPC codes, a hard-decision decoder passes only hard bits along the Tanner graph; for example, a bit-flipping decoder sends 0 or 1 messages based on parity checks, rather than log-likelihood values. It leverages the sparse parity-check matrix **H** to iteratively detect and correct errors via localized parity-check equations.

Mathematically, given a received vector **r**, HDD first computes:

$$\hat{c}_i = \begin{cases} 1 & \text{if } r_i < \tau \quad (\text{e.g., } \tau = 0 \text{ for BPSK}) \\ 0 & \text{otherwise} \end{cases}$$

where τ is a decision threshold. The decoder then iteratively refines ˆc until all parity checks H ˆcT = 0 are satisfied.

### Why it is used :

- **Complexity-Efficiency Trade-off:** HDD avoids floating-point operations, reducing computational complexity to O(E) per iteration(E = edges in H), enabling FPGA/ASIC implementations with 50–60% lower power than soft-decision decoders.
- **High-Throughput Applications:** Parallel processing of sparse H allows simultaneous updates of variable nodes (VNs) and check nodes (CNs), achieving throughputs > 200 Mbps in 5G NR.
- **Robustness to Quantization Noise:** Suitable for low-resolution ADCs in mmWave systems, where soft reliability metrics are unreliable.

### Method: Gallager's Bit-Flipping Algorithm

The canonical HDD method for LDPC codes involves three steps:

### Step 1: Graph Initialization

Variable Node (VN) to Check Node (CN) Messages: Each VN $v_i$ sends its hard-decoded bit to connected CNs.
Check Node (CN) to Variable Node (VN) Messages: Each CN $c_j$ computes the parity of its connected VNs except the VN for which it is calculating.

$$s_j = \bigoplus_{k \in N(j)} \hat{c}_k$$

where N( j) are VNs connected to $c_j$. If $s_j = 1$, the CN flags an error.

### Step 2: Iterative Bit Flipping

VN Update Rule: Each VN $v_i$ counts the number of connected CNs M(i) reporting errors:

$$\Delta_i = \sum_{j \in M(i)} s_j$$

where M(i) are CNs connected to $v_i$.
If $\Delta_i > \frac{M(i)}{2}$ , $v_i$ flips its bit.

### Step 3: Stopping Criteria

At the end for each VN majority voting is performed where the value which have major vote among all CNs connected to the VN and VN itself, are assigned to VN now.
**Syndrome Check:** Terminate if $H\hat{c}^T = 0$.
**Max Iterations:** Cap iterations to prevent infinite loops (e.g., 20–50 in 5G NR)

## Rationale Behind Hard-Decision Steps in LDPC Decoding

The specific steps of a hard-decision (bit-flipping) LDPC decoder are motivated by simple reasoning about errors and parity constraints:

- **Binary quantization (hard decision):** We start by converting analog or multi-level channel outputs into bits. This is done either because the channel is naturally binary, or because it simplifies the decoder. By ignoring soft information we vastly reduce complexity; a hard-decoder only needs to know whether each bit is more likely a 0 or 1. The downside is that we lose information (the decoder cannot tell if a bit was only barely above threshold or very far above). Nonetheless, for many block codes this simplification is acceptable. Indeed, classical block codes often assume hard inputs. In LDPC decoding, using hard bits effectively models a binary symmetric channel.
- **Parity-check message-passing:** LDPC codes are defined by sparse parity-check equations. The bit-flipping decoder enforces these constraints iteratively. Each parity-check (check node) essentially “votes” on what each bit should be based on the other bits in that check. Mathematically, a check node computes the XOR of its other bits; if that XOR is 0, it would

prefer the remaining bit to be 0, if it is 1, it wants the remaining bit to be 1. This is why each check sends the modulo-2 sum of its other neighbors. From a probabilistic standpoint, this is an extrinsic message: it reflects what value would satisfy that check.

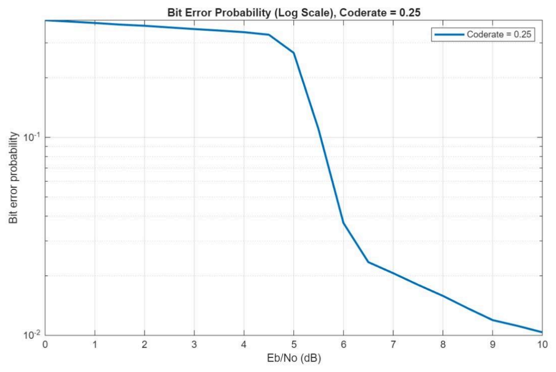
- **Majority logic (flip rule):** At each bit node, we gather the "opinions" of all adjacent checks. If most checks agree that the bit should be flipped, it is likely that the current bit is in error. Flipping it will then satisfy many checks. For example, if a bit participates in many unsatisfied checks (each unsatisfied check "votes" for the opposite bit), flipping reduces the total number of violations. The simplest rule is to flip whenever the number of unsatisfied checks exceeds some threshold (often half the degree, i.e. a majority). In Gallager's Algorithm A, the threshold is unanimity: the bit only changes if *all* other checks agree on a new value. Gallager's Algorithm B relaxes this to a tunable threshold. In effect, the algorithm is trying to minimize the Hamming distance to a valid codeword by greedily reducing parity errors.

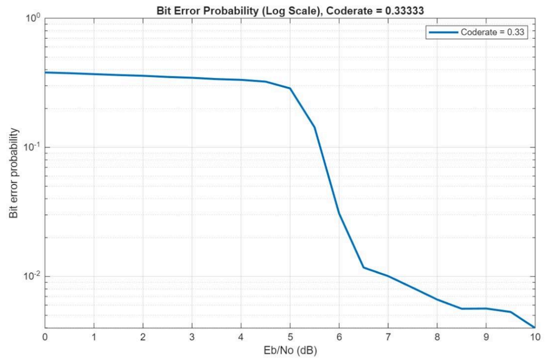
- **Iterations and stopping:** We repeat these message exchanges and flips because one bit error can cause multiple parity failures, and fixing one bit may reveal other errors. Each iteration propagates information one hop further in the graph. The decoder stops when it reaches a consistent solution (all parity-checks satisfied). At that point the bit estimates form a valid codeword; further iterations cannot improve it. If convergence does not occur quickly, the algorithm stops anyway (after a maximum number of rounds) to limit latency.

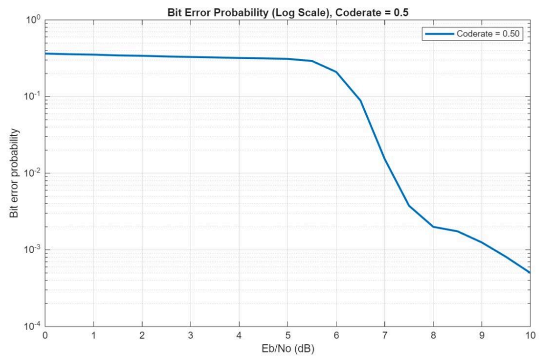
The overall rationale is to use local parity constraints to guide correction. Hard-decision bit-flipping is essentially a heuristic for maximum-likelihood decoding: in a BSC, ML decoding would find the codeword nearest (in Hamming distance) to the received word.

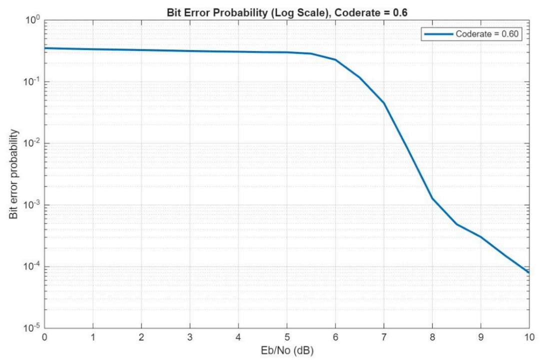
## Observations from HDD's output graphs

## 1)BER GRAPHS







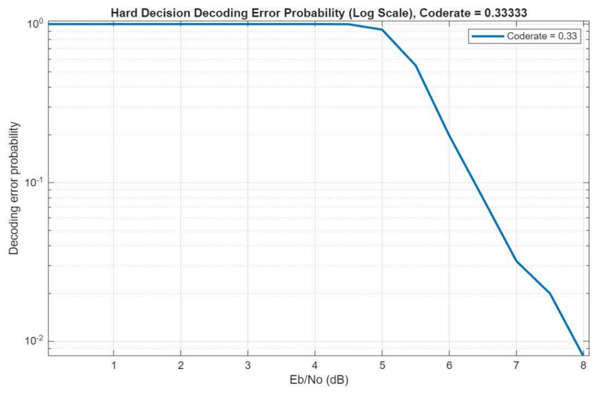


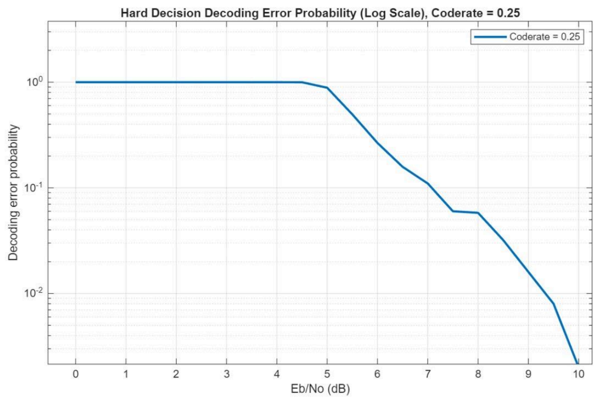
| **Coderate** | **Waterfall Start (Eb/No, dB)** | **Error Floor (Lowest BER)** | **Curve Slope** | **Redundancy** | **Summary of Performance** |
|---|---|---|---|---|---|
| 0.25 | ~5 | ~1e-3 | Steep | Highest | Best error correction, needs least SNR |
| 0.33 | ~6 | ~1e-3 | Steep | High | Very good error correction, low SNR needed |
| 0.5 | ~7 | ~1e-4 | Steep | Moderate | Good correction, moderate SNR requirement |

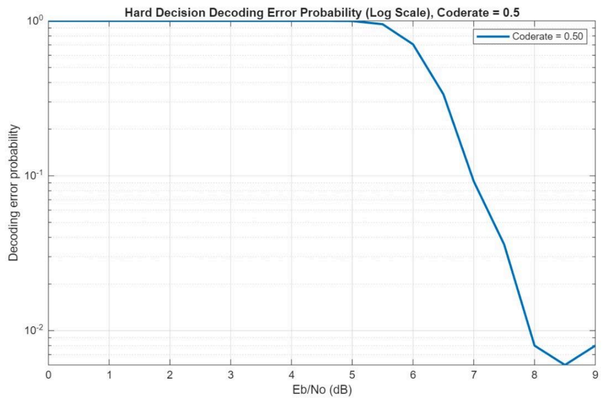
| | | | | | |
|---|---|---|---|---|---|
| 0.6 | ~8 | ~1e-4 | Steep but shifts right | Lowest | Needs highest SNR, least error protection because least parity |

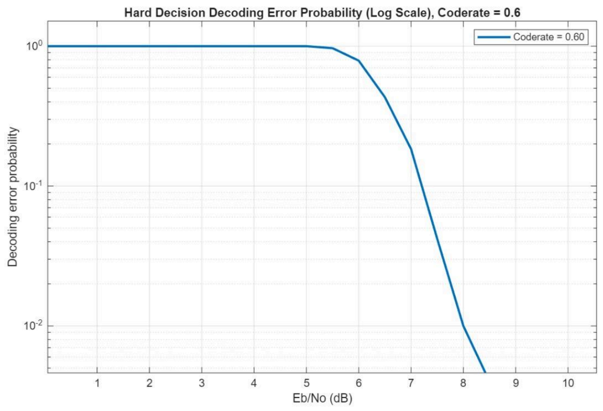
The error probability curves shift to the right as the coderate increases from 0.25 to 0.6. This means higher coderates require a higher Eb/No to achieve the same error performance, because they have less redundancy for error correction. Lower coderates, with more parity bits, can achieve low error rates at a lower Eb/No, demonstrating stronger error correction capability. This illustrates the classic trade-off between data rate and error protection in coding theory.
**Lower coderate = better error correction at lower Eb/No (more robust, less efficient).**
**Higher coderate = needs higher Eb/No for same performance (less robust, more efficient).**

## 2)DER GRAPHS









**All curves start with DER ≈ 1 at low Eb/No, meaning almost every codeword is decoded incorrectly**

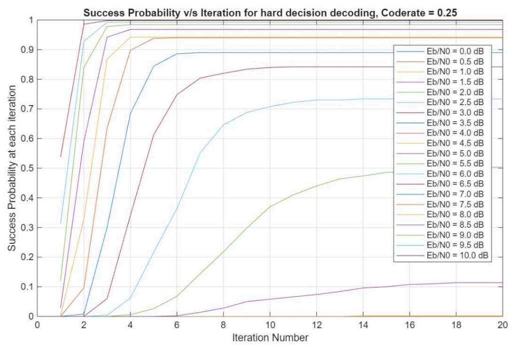
| **Coderate** | **Waterfall Start (Eb/No, dB)** | **Error Floor (Lowest DER)** | **Curve Slope** | **Redundancy** | **Summary of Performance** |
|---|---|---|---|---|---|
| 0.25 | ~5 | ~1e-2 | Steep | Highest | Best block error correction, lowest SNR needed, highest redundancy |
| 0.33 | ~6 | ~1e-2 | Steep | High | Very good block error correction, low SNR needed |
| 0.5 | ~7 | ~2e-2 | Steep | Moderate | Good block error correction, moderate SNR needed |
| 0.6 | ~8 | ~2e-2 | Steep but shifts right | Lowest | Needs highest SNR, weakest block error correction, least redundancy |

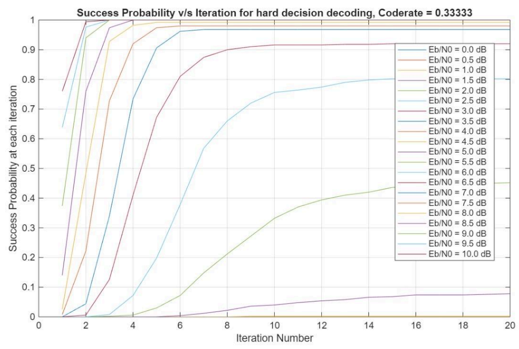
- DER (Decoding Error Rate )
  Measures the fraction of entire codewords that are decoded incorrectly (i.e., at least one bit is wrong in the block). Interpretation: Probability that a whole codeword is bad, even if only one bit is in error.

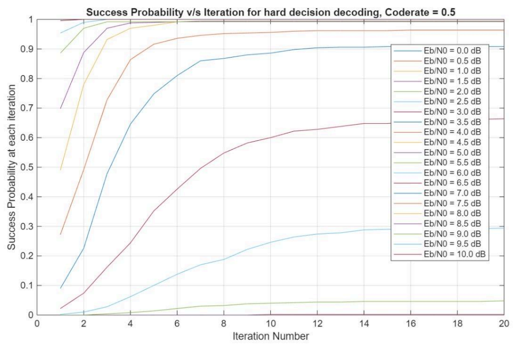
**Why Do DER Curves Look Different from BER?**

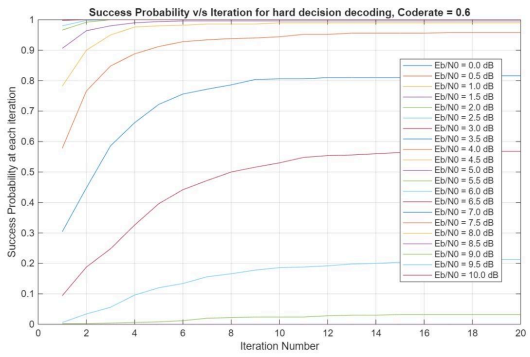
DER is always higher than BER for the same Eb/No, especially for longer block lengths. In practical systems, DER is often more important because a single bit error can corrupt an entire packet or frame.

## 3) Success Probability vs. Iteration Number (for various Eb/No values)









| **Coderate** | **Convergence Speed (Iterations for High Success)** | **Max Success Probability at High SNR** | **Effect of Eb/No** | **Summary of Performance** |
|---|---|---|---|---|
| 0.25 | Fast (≤5 iterations at moderate/high Eb/No) | ≈1.0 | Steep improvement with Eb/No | Rapid convergence, high success even at lower Eb/No. |

| | | | | |
|---|---|---|---|---|
| 0.33 | Fast (≤5 iterations at moderate/high Eb/No) | ≈0.95–1.0 | Steep improvement with Eb/No | Quick convergence, robust at moderate SNR. |
| 0.5 | Moderate (5–10 iterations at moderate Eb/No) | ≈0.9–1.0 | Gradual improvement with Eb/No | Slower convergence, needs higher SNR for high success. |
| 0.6 | Slower (≥10 iterations at moderate Eb/No) | ≈0.8–1.0 | Gradual improvement with Eb/No | Slowest convergence, lowest success at low SNR. |

The success probability graphs show that LDPC codes with lower coderates converge to correct decoding much faster and more reliably, even at lower SNRs. As the coderate increases, the decoder requires more iterations and higher SNR to achieve a similar level of success.

## Theoretical Justification for HDD's Structure

- Sparsity-Driven Efficiency: The sparse H ensures each CN connects to few VNs (≈ 3–6), limiting per-iteration computations.This sparsity enables parallel layered decoding, a key 5G innovation.

- Majority Logic and Error Localization: The bit-flipping rule is a majority-logic decoder that corrects bits involved in the most failed parity checks. For a VN with degree dv, the probability of mis-correction is bounded by:

$$P_e \leq \sum_{k=\lceil d_v/2 \rceil}^{d_v} \binom{d_v}{k} p^k (1-p)^{d_v-k}$$

  where p is the raw bit error rate.

- Convergence Guarantees: For a code with girth g, HDD converges if the Tanner graph is free of trapping sets—subgraphs causing oscillatory behavior.

## Conclusion

HDD in LDPC codes balances between computational efficiency and error correction,
making it indispensable for 5G's high-speed, low-power demands. Its reliance on majority logic
and sparse matrix operations ensures scalability to block lengths > 10, 000 bits while maintaining
linear complexity.

## Soft Decoding:

Just as we use the bits received at the receiver directly in hard decision decoding, in soft decoding we use probability-specific LLR (log-likelihood ratio) to decode the message.

In soft decision decoding, Low-Density Parity-Check (LDPC) codes can be represented using Tanner graphs. These graphs consist of variable nodes (VNs) (which represent code bits) and check nodes (CNs) (which represent parity-check equations).

Instead of passing bits in the message-passing algorithm, we pass Log-Likelihood Ratios (LLRs) from variable nodes to check nodes and vice versa.

There are two types of LLRs used in the soft decision decoding algorithm:

1. Intrinsic LLR
2. Extrinsic LLR

For instance, consider a variable node VN1, that is connected to three check nodes: CN1, CN2, and CN3. When we decode VN1, the information that CN1 provides about VN1 is referred to as the intrinsic LLR. while the information that CN2 and CN3 provide about VN1 is referred to as the extrinsic LLR.

### Min-Sum Algorithm

**Below is the proof of the min-sum algorithm using intrinsic as well as extrinsic LLR in SPC decoding.**

Page No. Date

(1) calculation of the intrinsic LLR

→ we Assume that the BPSK transmitter sends x = 0 and x = 1 with equal probability, i.e., the prior on x = 1 is q = 0.5. with this, the optimal bayesian detector is the maximum likelihood detector, which evalutes the ratio,

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

$$= \frac{\dfrac{P\left(\dfrac{r_1}{c_1=1}\right) \cdot \cancel{P(c_1=1)}}{\cancel{P(r_1)}}}{\dfrac{P\left(\dfrac{r_1}{c_1=0}\right) \cdot \cancel{P(c_0=0)}}{\cancel{P(r_1)}}} \qquad [\because P(c_0 = 0) = P(c_1 = 1) = 0.5]$$

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

$$= \frac{\dfrac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[\dfrac{-1}{2}\left(\dfrac{r_1-1}{\sigma}\right)^2\right]}{\dfrac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\dfrac{-1}{2}\left(\dfrac{r_1+1}{\sigma}\right)^2\right]}$$

$$= \exp\left[\frac{-1}{2}\frac{(r_1-1)^2}{\sigma^2} - \left(\frac{-1}{2}\frac{(r_1+1)^2}{\sigma^2}\right)\right]$$

$$= \exp\left[\frac{1}{2\sigma^2}\left[(r_1+1)^2 - (r_1-1)^2\right]\right]$$

$$= \exp\left(\frac{1}{2\sigma^2} \times 4r_1\right)$$

$$= \exp\left(\frac{2r_1}{\sigma^2}\right)$$

→ log likelihood Ratio,

$$\text{LLR for } c_1 = \log_e\left(\exp\left(\frac{2r_1}{\sigma^2}\right)\right)$$

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

→ General LLR for $c_j$,

$$= \frac{2r_j}{\sigma^2}$$

(2) calculation of output LLR for repetation codes (n=3

$$L_j = \log\left[\frac{P(C_j = 1 \mid r_1, r_2, r_3)}{P(C_j = 0 \mid r_1, r_2, r_3)}\right]$$

$$= \log\left[\frac{P(r_1, r_2, r_3 \mid C_j = 1)}{P(r_1, r_2, r_3 \mid C_j = 0)}\right] \quad [\because \text{prior} = 0.5]$$

→ here,

$r_1 = 1 + N(0, \sigma^2)$

$r_2 = 1 + N(0, \sigma^2)$

$r_3 = 1 + N(0, \sigma^2)$

$$= \log \left[ \frac{\frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2}\left(\frac{r_1-1}{\sigma}\right)^2} \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2}\left(\frac{r_2-1}{\sigma}\right)^2} \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2}\left(\frac{r_3-1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2}\left(\frac{r_1+1}{\sigma}\right)^2} \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2}\left(\frac{r_2+1}{\sigma}\right)^2} \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{1}{2}\left(\frac{r_3+1}{\sigma}\right)^2}} \right]$$

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

$$= \frac{-1}{2\sigma^2}\left[-4r_1-4r_2-4r_3\right]$$

$$= \frac{2}{\sigma^2}\left[r_1+r_2+r_3\right]$$

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

$$L_j = \frac{2}{\sigma^2} \cdot \left[r_1+r_2+r_3\right]$$

(9) calculation of output LLR for single parity check codes.

Notation :-

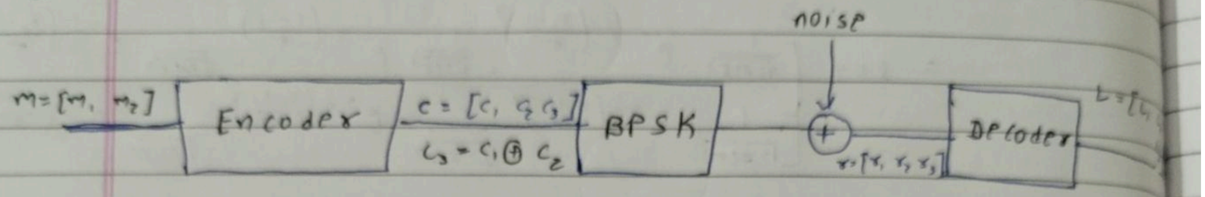
$L_j$ = output LLR for bit $j$

$l_j$ = input LLR for bit $j$ (or channel LLR)

$l_{ext,j}$ = extrinsic LLR for bit $j$.

$L_j = l_j + l_{ext,j}$

**Ex** :- (3, 2) SPC code



→ Here,

$$L_1 = l_1 + l_{ext,1}$$
$$L_2 = l_2 + l_{ext,2}$$
$$L_3 = l_3 + l_{ext,3}$$

where,

$$l_i = \frac{2}{\sigma^2} \cdot r_i \quad \text{[intrinsic LLR]}$$

→ using this instrinsic LLR, find the extrinsic LL

extrinsic : what do $r_2$ and $r_3$ say about $c_1$?

→ Now,

$$c_1 = c_2 \oplus c_3 .$$

→ Let's

$$P_2 = P[c_2 = 0 | r_2] \implies 1 - P_2 = P[c_2 = 1 | r_2]$$

$$P_3 = P[c_3 = 0 | r_3] \implies 1 - P_3 = P[c_3 = 1 | r_3]$$

→ so, instrinsic LLR,

$$l_2 = \log\left(\frac{P_2}{1 - P_2}\right), \quad l_3 = \log\left(\frac{P_3}{1 - P_3}\right)$$

Q:- Given $P_2$ and $P_3$, what is $P_1 = P[c_1 = 0 | r_2, r_3)$? and extrinsic LLR $\log\left(\frac{P_1}{1-P_1}\right) = ?$

| $c_2$ | $c_3$ | $c_1$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

→ $c_1 = 1$ when $(c_2 = 0$ and $c_3 = 1)$ or $(c_2 = 1$ and $c_3 = 0)$

→ $c_1 = 0$ when $(c_2 = 0$ and $c_3 = 0)$ or $(c_2 = 1$ and $c_3 = 1)$;

$$\therefore P_1 = P_2 P_3 + (1-P_2).(1-P_3)$$

and

$$1 - P_1 = P_2(1-P_3) + (1-P_2).P_3$$

→ Now,

$$\therefore P_1 - (1-P_1) = P_2[P_3 - (1-P_3)] + (1-P_2)[(1-P_3) - P_3]$$

$$\therefore P_1 - (1-P_1) = P_2[P_3 - (1-P_3)] - (1-P_2)[P_3 - (1-P_3)]$$

$$\therefore P_1 - (1-P_1) = [P_2 - (1-P_2)].[P_3 - (1-P_3)]$$

$$\therefore \frac{P_1 - (1-P_1)}{1} = \frac{[P_2 - (1-P_2)]}{1} . \frac{[P_3 - (1-P_3)]}{1}$$

$$\therefore \frac{P_1 - (1-P_1)}{P_1 + (1-P_1)} = \frac{[P_2 - (1-P_2)]}{P_2 + (1-P_2)} . \frac{[P_3 - (1-P_3)]}{P_3 + (1-P_3)}$$

$$\therefore \frac{1 - \left(\frac{1-P_1}{P_1}\right)}{1 + \left(\frac{1-P_1}{P_1}\right)} = \frac{1 - \left(\frac{1-P_2}{P_2}\right)}{1 + \left(\frac{1-P_2}{P_2}\right)} . \frac{1 - \left(\frac{1-P_3}{P_3}\right)}{1 + \left(\frac{1-P_3}{P_3}\right)} \quad \text{— (i)}$$

→ Now,

$$l_{ext,1} = \log\left(\frac{P_1}{1-P_1}\right)$$

$$\therefore\ l_{ext,1} = -\log\left(\frac{1-P_1}{P_1}\right)$$

$$\therefore\ \frac{1-P_1}{P_1} = e^{-l_{ext,1}}$$

→ similarly,

$$\therefore\ \frac{1-P_2}{P_2} = e^{-l_2} \quad \text{and} \quad \frac{1-P_3}{P_3} = e^{-l_3}$$

→ Put this value in equation I,

$$\therefore\ \frac{1-e^{-l_{ext,1}}}{1+e^{-l_{ext,1}}} = \frac{1-e^{-l_2}}{1+e^{-l_2}} \cdot \frac{1-e^{-l_3}}{1+e^{-l_3}} \quad \text{— (ii)}$$

→ Now, we know,

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

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

$$\therefore\ \text{so,}\quad \tanh\left(\frac{x}{2}\right) = \frac{1-e^{-x}}{1+e^{x}}$$

→ So, Now

$$\frac{1-e^{-l_{ext,1}}}{1+e^{-l_{ext,1}}} = \tanh\left(\frac{l_{ext,1}}{2}\right)$$

→ similarly,

$$\frac{1-e^{-l_2}}{1+e^{-l_2}} = \tanh\left(\frac{l_2}{2}\right) \quad \text{and} \quad \frac{1-e^{-l_3}}{1+e^{-l_3}} = \tanh\left(\frac{l_3}{2}\right)$$

→ put this value in equation - (ii),

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

→ Now, consider absolute value only, sign consider separately,

signs :- $\operatorname{sign}(l_{ext,1}) = \operatorname{sign}(l_2) \cdot \operatorname{sign}(l_3)$

Absolute value :-

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

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

→ Now, Define function,

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

we check that, $f^{-1}(x) = f(x)$.

→ Now, our equation is,

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

→ Take $f^{-1}$ both side,

$\therefore f^{-1}(f|l_{ext,1}|) = f^{-1}[f(|l_2|) + f(|l_3|)]$

$\therefore |l_{ext,1}| = f[f(|l_2|) + f(|l_3|)]$

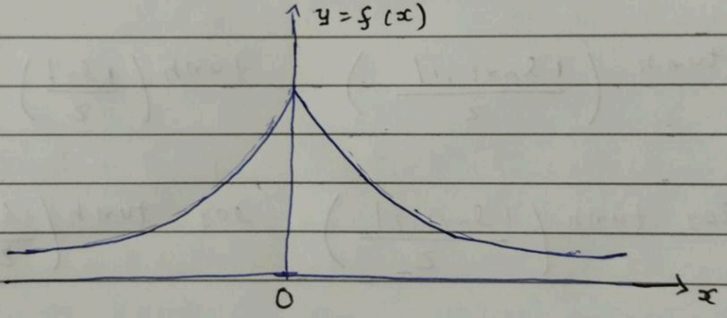
$[\because f^{-1}(x) = f(x)]$

and,

$\text{sign}(l_{ext,1}) = \text{sign}(l_2) \cdot \text{sign}(l_3)$

→ But,

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



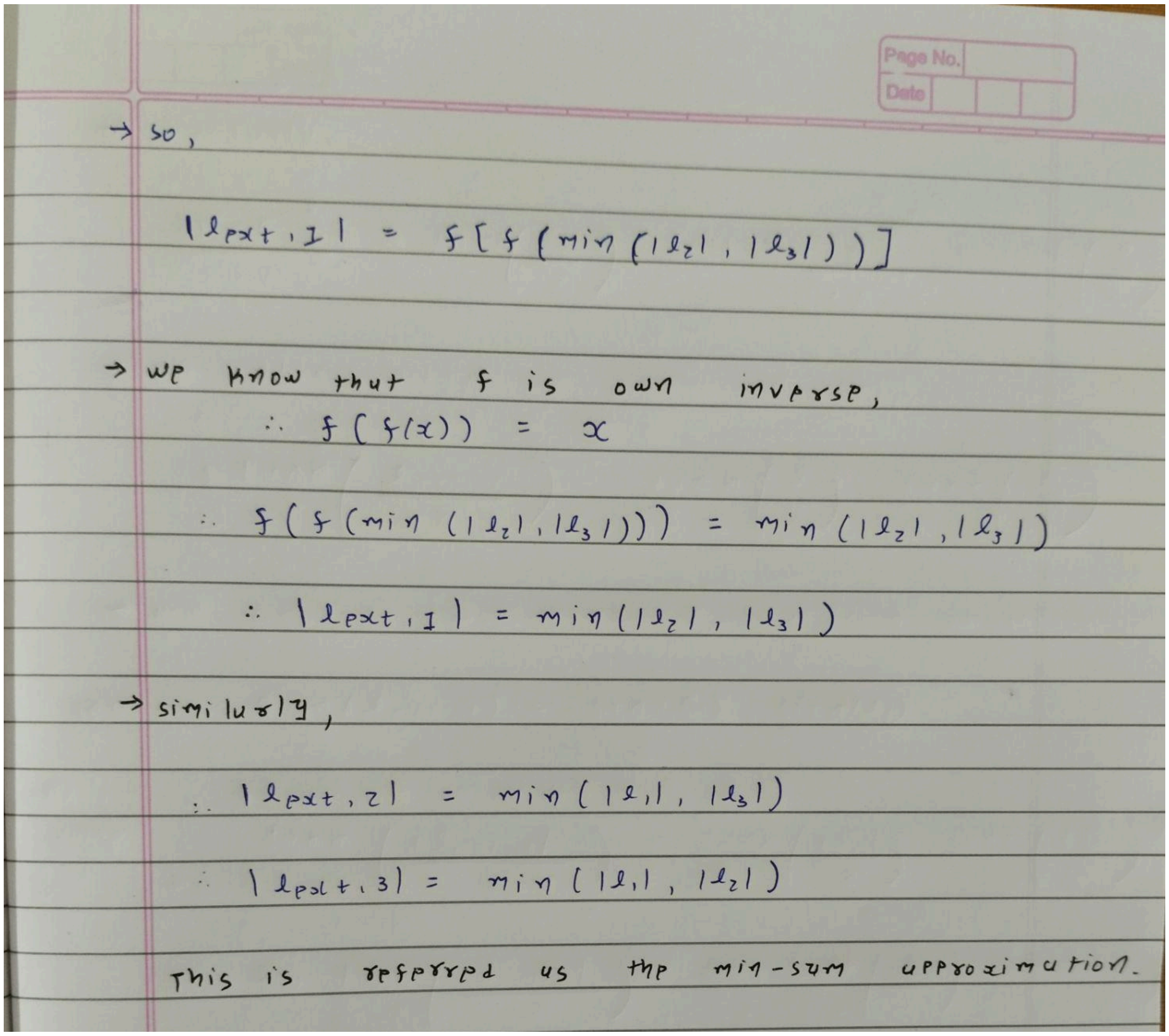
→ The value of $f(x)$ decrease substantially for larger value of $|x|$, thus $f(x)$ is deminated by the smallest value of $|x|$.

→ Now,

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

→ we can write,

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



→ so,

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

→ we know that $f$ is own inverse,

$$\therefore f(f(x)) = x$$

$$\therefore f(f(\min(|l_2|, |l_3|))) = \min(|l_2|, |l_3|)$$

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

→ similarly,

$$\therefore |l_{ext,2}| = \min(|l_1|, |l_3|)$$

$$\therefore |l_{ext,3}| = \min(|l_1|, |l_2|)$$

This is referred as the min-sum approximation.

**General Min-Sum Algorithm for Soft-Decision Decoding**

1. At first, all received values are stored in Variable Nodes (VNs). In the first iteration, each VN sends the value it received from the channel to the Check Nodes (CNs).

2. Next, each CN applies the Min-Sum algorithm to the received values. This process involves calculating the minimum value among all received inputs, and determining the overall sign by multiplying signs of all the received values together.

3. To avoid positive feedback, we do not use the data that has already been sent by the VN itself in the first place. If the VN was to send the minimum value among all incoming messages to a particular CN, it instead sends the second smallest value to that CN, while the minimum value is

sent to the other connected CNs. The sign of each message is determined by multiplying the overall sign of the inputs with the original sign of the VN.

4. The Variable Node applies the repetition Log-Likelihood Ratio (LLR) to the values it has received which is simply the sum of the received LLR values. Once again, to avoid positive feedback, we again do not use any values sent from a particular CN. Instead, we sum the values from all other CNs and send that result.

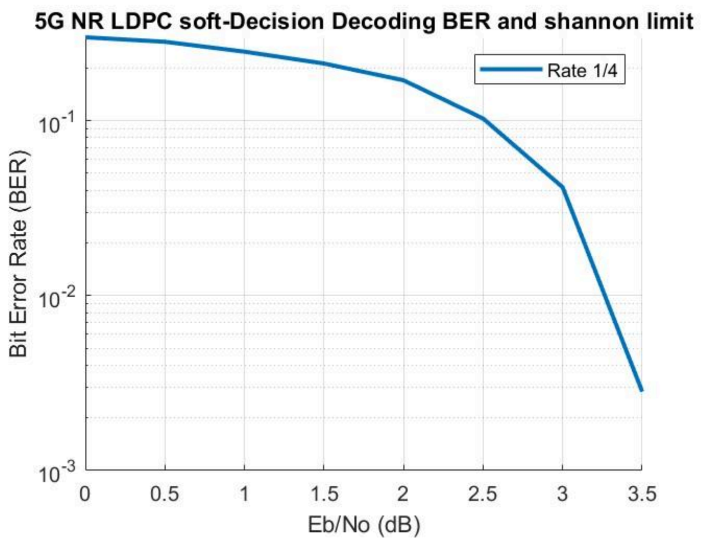
5. The updated belief is calculated as the sum of the received LLR values.

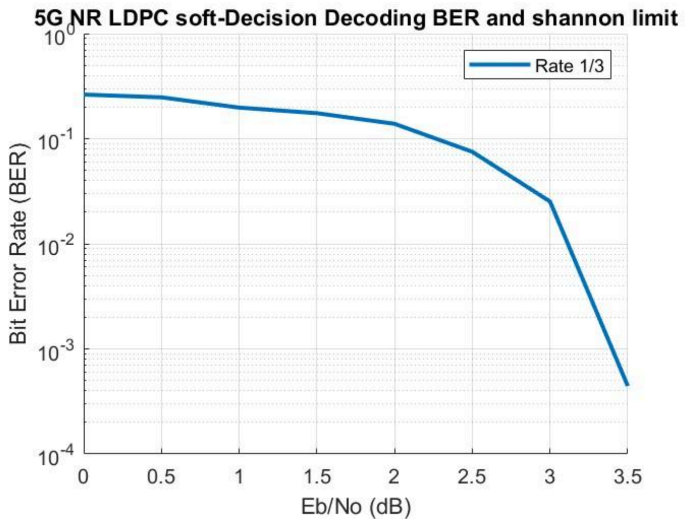
After completing all iterations, the message is decoded using the following equation:

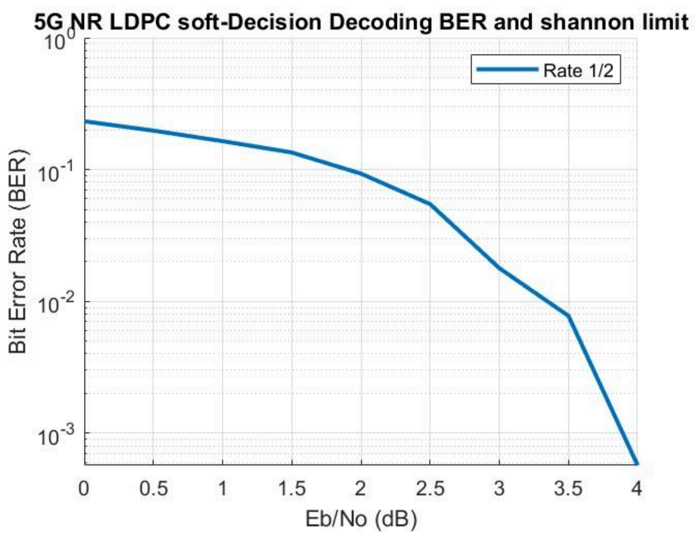
$$\hat{c}_i = \begin{cases} 1 & \text{if } r_i < \tau \quad (\text{e.g., } \tau = 0 \text{ for BPSK}) \\ 0 & \text{otherwise} \end{cases}$$

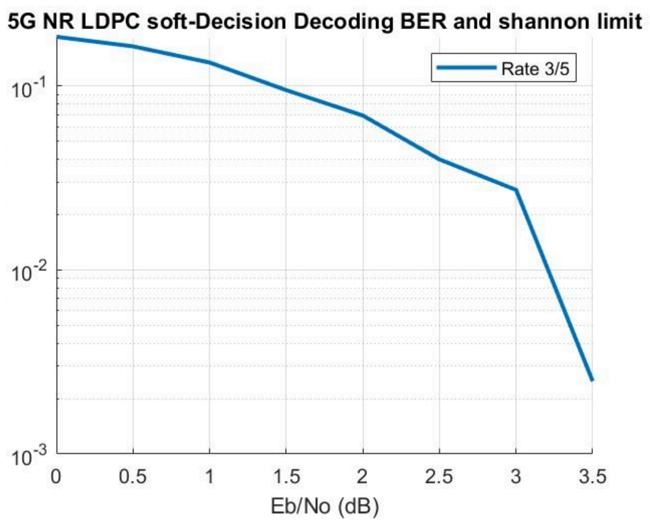
## Observation from Soft Decoding output graphs

### 1) BER GRAPHS







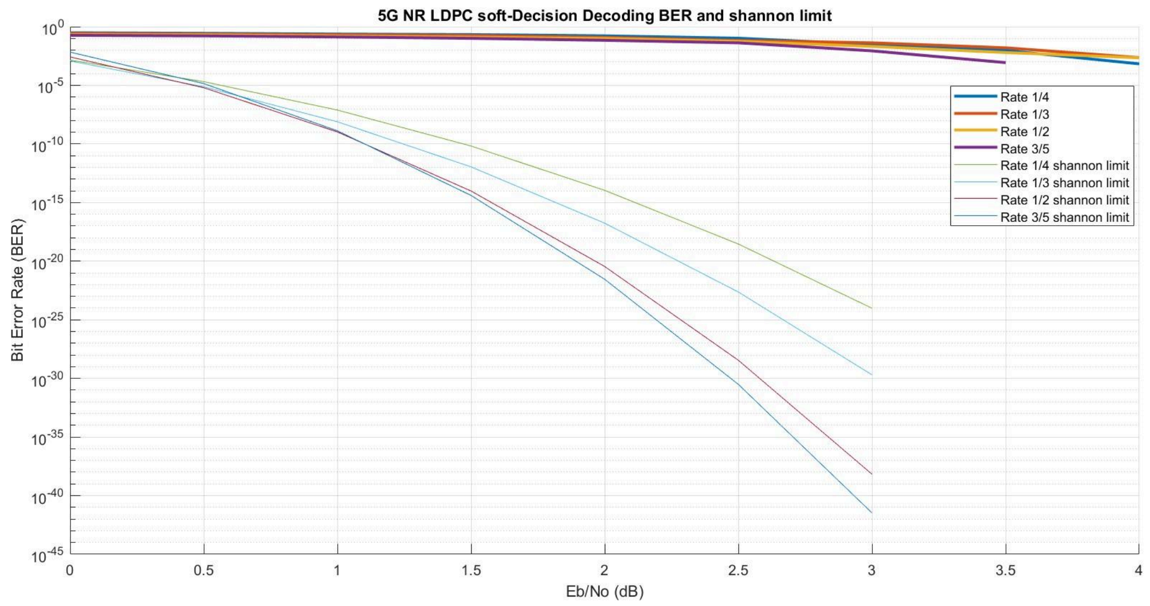


| Coderate | Waterfall Start (Eb/No, dB) | Error Floor (Lowest BER) | Curve Slope | Redundancy | Summary of Performance |
|---|---|---|---|---|---|
| 0.25 | ~3 | ~1e-3 | Gradual decrease until 3 dB, then steep waterfall | Highest | Achieves BER of $10^{-3}$ at 3.5 dB (approx.) |
| 0.33 | ~3 | ~1e-4 | More gradual initial decrease until 3 dB, then steep slope | High | Very good error correction, low SNR needed |
| 0.5 | ~3.5 | ~1e-3 | Smooth decline until 3.5 dB, then steeper drop | Moderate | Needs highest SNR to achieve BER close to 1e-3 |

| | | | | | |
|---|---|---|---|---|---|
| 0.6 | ~3 | ~1e-3 | Similar minimum BER as ¼, but with less steep slope | Lowest | Similar to that with ¼ code rate but steeper graph |

## Comparison with Shannon Limit:

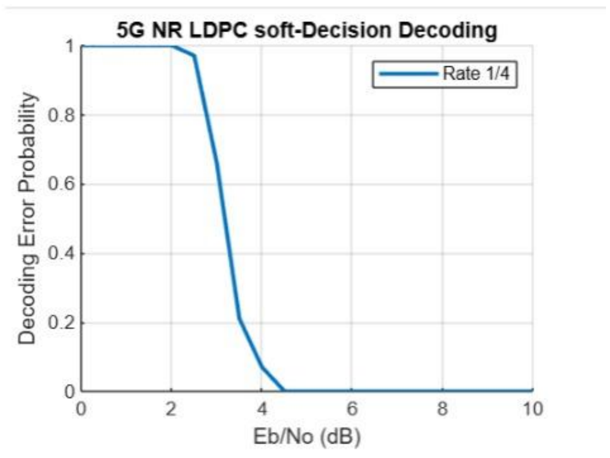
As shown below, the graph shows the comparison of the implementation of soft decision decoding against the Shannon limit, which is evaluated as $10\, log_{10}\left(\frac{2^r - 1}{r}\right)$ dB.

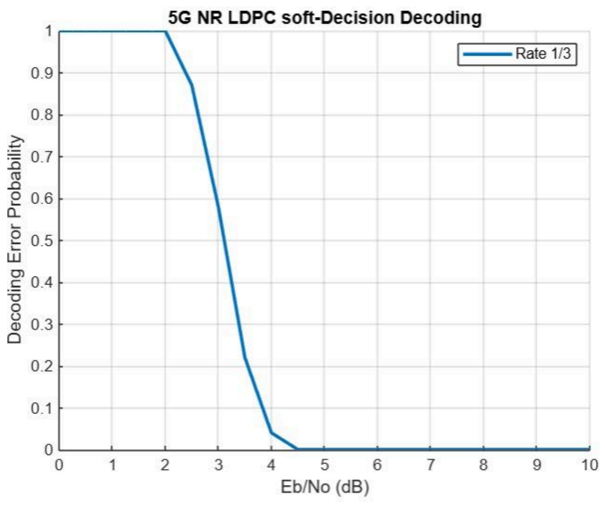


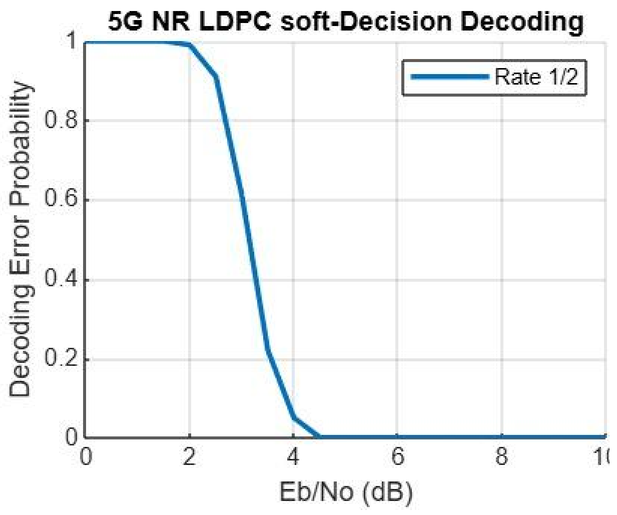
From the graph, we find that as the SNR increases, the BER starts reducing. This is because a stronger signal is easier to decode correctly. It's observed that lower code rates achieve lower BER at lower Eb/No compared to higher code rates. This is because they have more redundant bits for error correction.

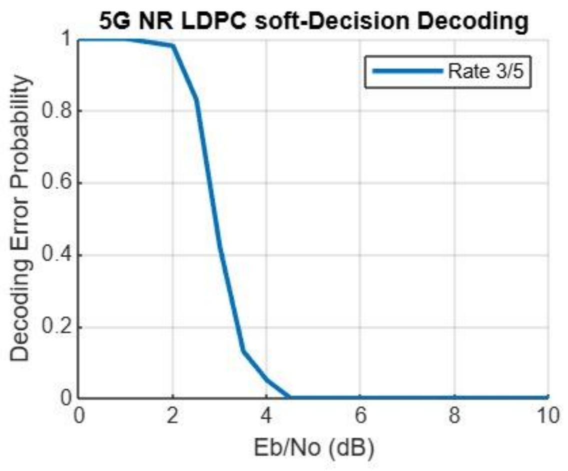
The Shannon Limit shows the theoretically best-case scenario for each of the code rates.

## 2) Decoding Error Probability Graphs







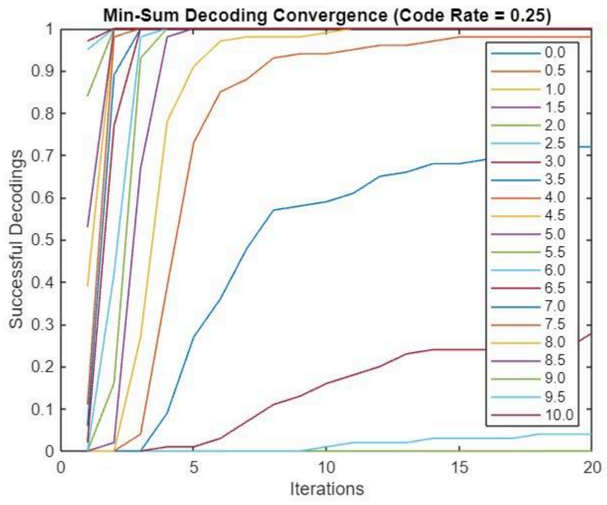


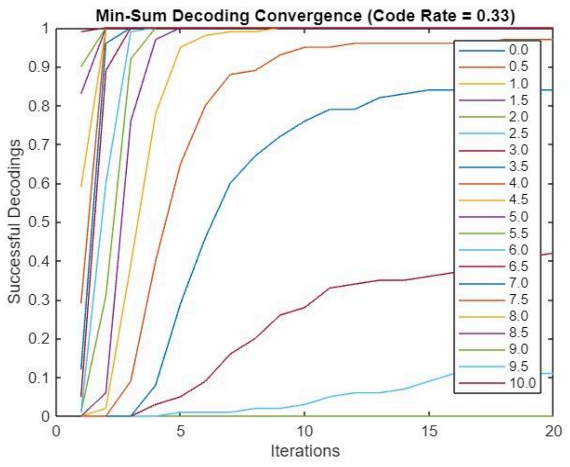
| **Coderate** | **Waterfall Start (Eb/No, dB)** | **Error Floor (Lowest)** | **Curve Slope** | **Redundancy** | **Summary of Performance** |
|---|---|---|---|---|---|
| 0.25 | ~2.5 | Approaches 0 at Eb/No ≈ 4.5 | Steep | Highest | Achieves very low decoding error probability at around 4.5 to 5 dB. |

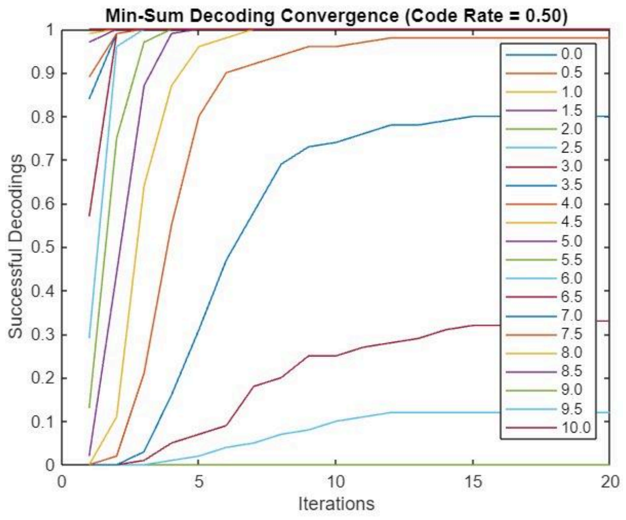
| | | | | | |
|---|---|---|---|---|---|
| 0.33 | ~2.5 | Approaches 0 at Eb/No $\approx 4.5$ | Steep | High | Requires slightly higher signal power than for rate ¼ for very low decoding error probability. |
| 0.5 | ~2.5 | Approaches 0 at Eb/No $\approx 4.5$ | Steep | Moderate | Achieves very low decoding error probability at slightly higher Eb/No |
| 0.6 | ~2.5 | Approaches 0 at Eb/No $\approx 4.5$ | steep | Lowest | Requires highest SNR for decoding error probability to approach zero. |

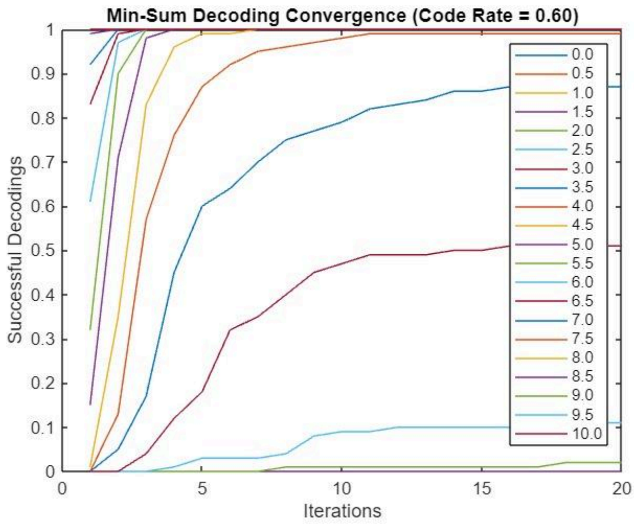
Lower code rates have more redundancies, which provide strong error correction capabilities, allowing the code to perform well even at lower SNR values. On the other hand, higher code rates have lesser redundancy which is why they require higher SNR to achieve strong error correction capabilities.

## 3) Success Probability vs. Iteration Number (for various Eb/No values):









| **Coderate** | **Convergence Speed (Iterations for High Success)** | **Max Success Probability at High SNR** | **Effect of Eb/No** | **Summary of Performance** |
|---|---|---|---|---|
| 0.25 | Fast (≤5 iterations at moderate/high Eb/No) | ≈1.0 | Very Steep improvement with increasing Eb/No | Rapid convergence, high success even at lower Eb/No. |

| | | | | |
|---|---|---|---|---|
| 0.33 | Fast (≤5 iterations at moderate/high Eb/No) | ≈1.0 | Steep improvement with Eb/No | Rapid convergence, high success even at lower Eb/No. |
| 0.5 | Moderate (5–10 iterations at high Eb/No) | ≈0.95–1.0 | Gradual improvement as Eb/No increases | Moderate convergence, requires higher Eb/No |
| 0.6 | Slow (10+ iterations at high Eb/No) | ≈0.95–1.0 | Gradual improvement with Eb/No | Slowest convergence, most sensitive to SNR conditions. |

From the above graphs, we find that the soft decoding approach provides better performance as compared to hard decision decoding in terms of successful decoding across all code rates.

## Conclusion:

Soft decoding is a fundamental technique that significantly enhances the performance of LDPC codes by utilizing probabilistic information from the received signal, rather than relying solely on binary decisions unlike hard decoding. Soft decoding increases confidence levels by using log-likelihood ratios (LLRs) to make more accurate and robust decisions. This probabilistic approach results in notably lower bit error rates, especially in noisy or low-SNR conditions. By enabling iterative refinement through message passing between variable and check nodes, soft decoding brings LDPC decoding performance closer to the theoretical limits set by Shannon.

## Sources:

1. https://www.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld/www/slidesS14/ldpc-amin.pdf
2. https://www.digitalxplore.org/up_proc/pdf/93-1406552433112-117.pdf
3. https://www.diva-portal.org/smash/get/diva2:1611415/FULLTEXT01.pdf
4. https://youtube.com/playlist?list=PLyqSpQzTE6M81HJ26ZaNv0V3ROBrcv-Kc&si=f0sFGVXhsoSv14VW