LDPC DECODING FOR 5G NR

Lab Group - 6, Sub Group - 32

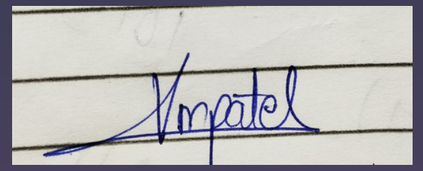
Under guidance of Prof. Yash Vasavda TA : Kartavya A.

HONOR CODE

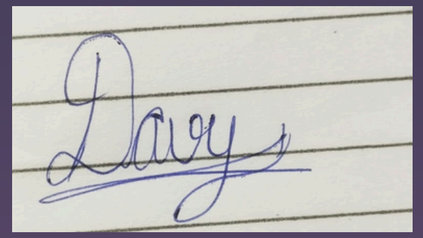
We declare that:

- 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 solemnpledge can carry grave consequences.

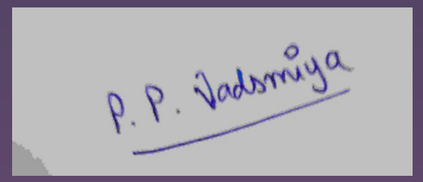
Vansh Patel



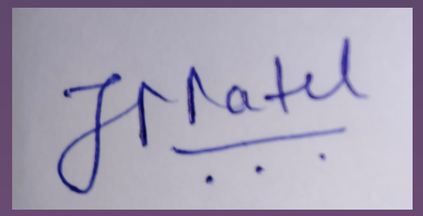
Vagh Divyesh



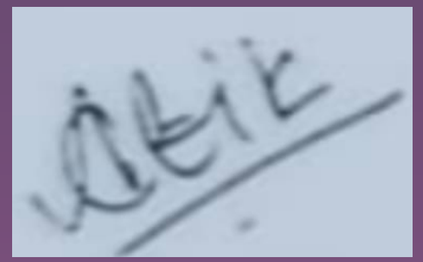
Pransu Vadsmiya



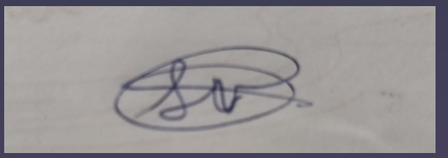
Jainesh Patel



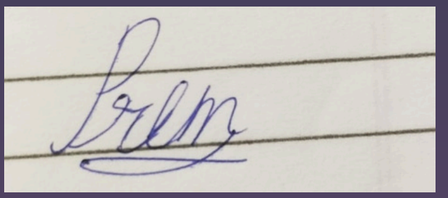
Atik Vohra



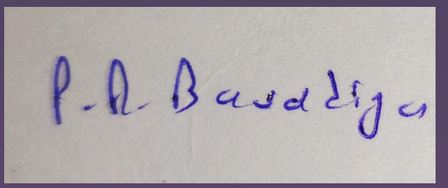
Shubham Varmora



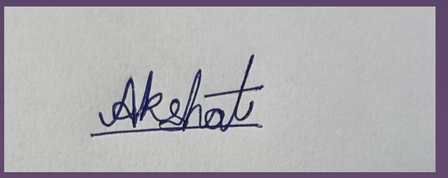
Prem Kukadiya



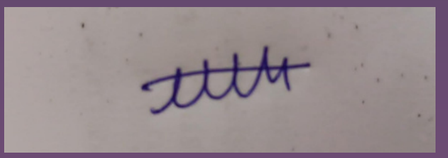
Pranav Bavadiya



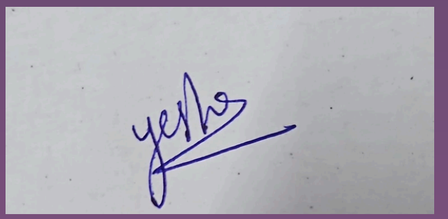
Akshat Bhatt



Ayush Chovatiya

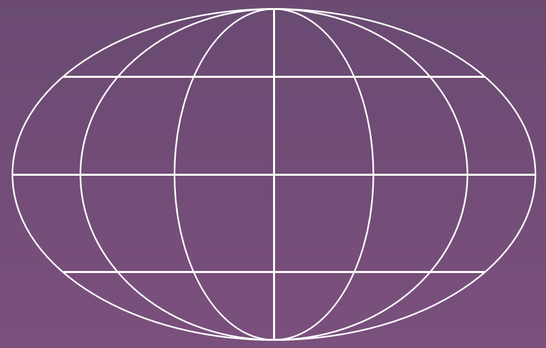


Yesha Joshi



Contents

About LDPC Codes



1. Introduction to LDPC Codes

LDPC codes, made by Gallager, use sparse matrices. They're fast, simple, and reliable. They are widely used in error correction for communication systems.

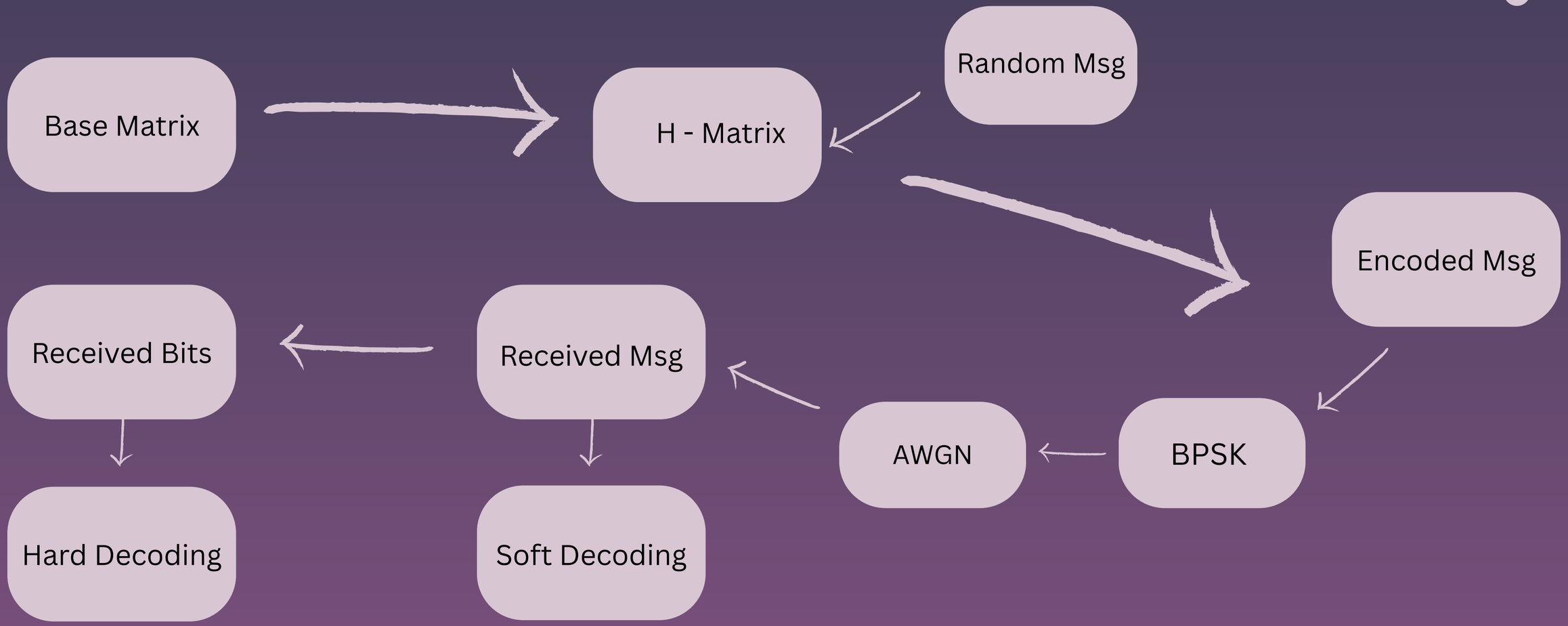
2. Advantages in 5G NR

Ideal for 5G: low delay, easy decoding, and strong in noise.
They ensure smooth data flow in high-speed networks.
LDPC codes approach the Shannon limit, offering near-optimal performance.

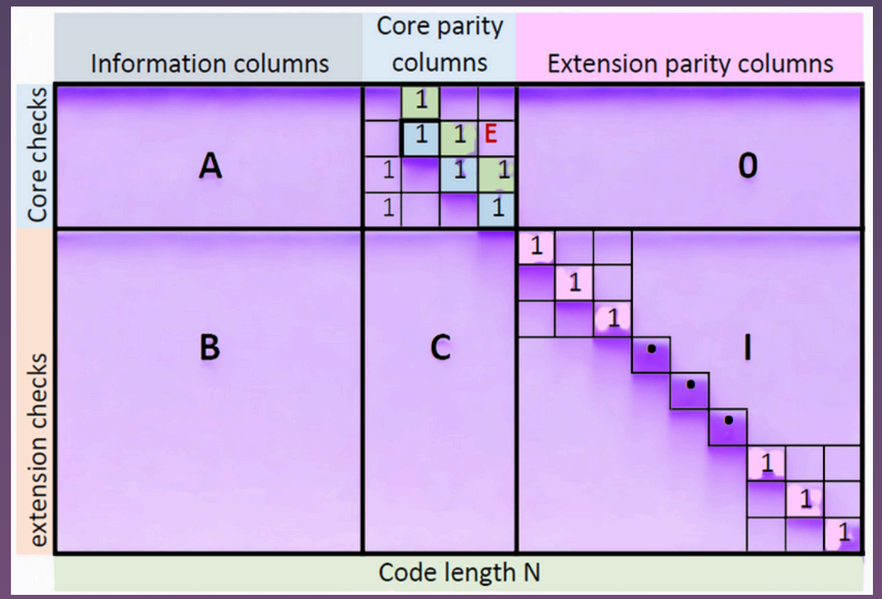
3. Flexibility and Adaptability

Support all sizes and rates. Fit well in modern networks.
Their design allows easy scaling for various applications.

Flow Chart


Base Matrix
H - Matrix
Random Msg
Encoded Msg
BPSK
AWGN
Received Msg
Received Bits
Soft Decoding
Hard Decoding

Base Graph Matrix



- Base Graphs in 5G NR LDPC:

5G NR employs two types of base graphs, BG1 and BG2, selected depending on the coding rate and input data length.

- Dimensions of Base Graphs:

BG1 has dimensions of 46 rows by 68 columns, while BG2 measures 42 rows by 52 columns.

- Calculation of Message Bits:

The number of information bits (k) is calculated as the difference between the number of columns and rows (k = m - n). This results in a maximum of 22 bits for BG1 and 10 bits for BG2.

- Block-Based Design:

Each base graph features a specific block-style arrangement that is beneficial during the encoding phase.

- Adaptability of LDPC Codes:

Using two base graphs enables 5G NR LDPC codes to efficiently adjust to varying data lengths, supporting dependable transmission in multiple scenarios.

Parity Check Matrix Generation

To generate the parity-check matrix H from the base graph B, a parameter called z—also known as the lifting size or expansion factor—is used. This value is defined for each base graph and applied according to these guidelines:

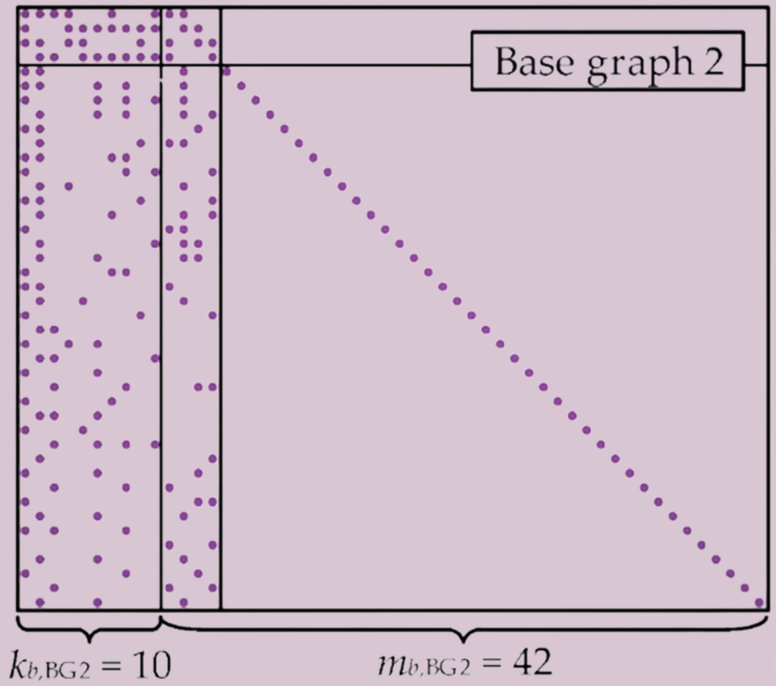
- If an entry in B is -1, it is substituted with a z × z matrix filled entirely with zeros.
- If an entry in B is 0, it is replaced by a z × z identity matrix.
- If an entry in B ranges from 1 to z - 1, it is replaced by a z × z identity matrix that has been circularly shifted to the right by B_{ij} positions, where B_{ij} represents the value of that particular element in B.

Base Matrix to H Matrix

$$B = \begin{bmatrix} 1 & -1 & 3 & 1 & 0 & -1 \\ 2 & 0 & -1 & 0 & 0 & 0 \\ -1 & 4 & 2 & 1 & -1 & 0 \end{bmatrix}$$

Expansion factor: 5

0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1



Base Matrix
Size Of 42x52

Coderate Matching

$$\frac{(n-m)\cdot z}{(n-2)\cdot z - a\cdot z} = \frac{x}{y}$$

Rate matching is a method applied in Low-Density Parity-Check (LDPC) coding to modify the code rate so it aligns with the target data rate of the communication system.

Assume we begin with a base matrix sized m × n and an expansion factor z. This means the total possible length of the LDPC codeword becomes n × z. Out of these, the initial 2z bits are always removed through puncturing, leaving (n - 2) × z bits.

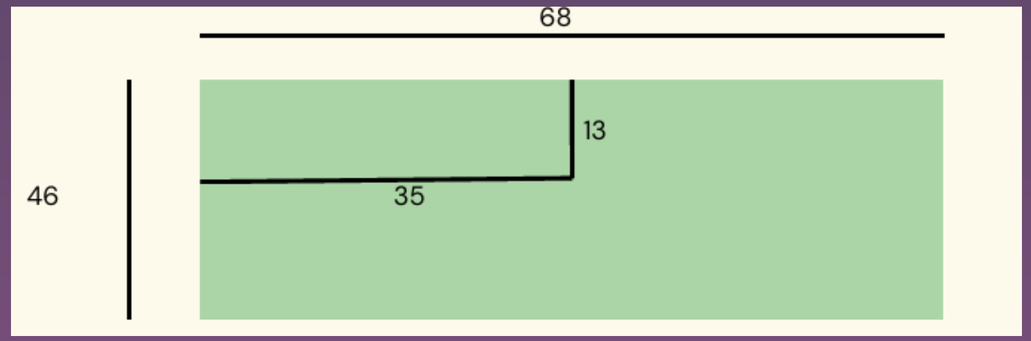
Now, if the desired code rate is x/y, and you choose to puncture a × z bits from the available (n - 2) × z, then the number of message bits becomes (n - m) × z. Using this relationship, you can calculate how many parity bits are necessary.
Accordingly, the dimensions of the H matrix are determined as:

- Number of rows = number of required parity bits (from the rate equation).
- Number of columns = total columns from the equation plus 2.

To make this clearer, we'll go through an example next.

Coderate Matching Example

- Suppose we have a 46 X 68 Base matrix and we want a code rate of 2/3.
- So, n = 68, m= 46, x = 2, y = 3. Putting these values in the previous equation and solving for a, we get a = 33.
- So, now we have to transmit 66z - 33z = 33z bits in all.
- Out of these 33z bits, 20z are message bits. Thus, we send 13z parity bits.
- Number of rows of B matrix used = 13 rows
- Number of columns of B matrix used = 33 + 2 = 35
- So, for a code rate of 2/3 we use 13 X 35 entries of the B matrix.



Encoding Using Base Matrix

- The structure of the base matrix greatly simplifies the encoding process.
- During decoding, the parity bits (starting from a certain position) can be derived by solving straightforward linear equations due to the predictable pattern in the matrix.
- For the initial set of message bits, a few equations need to be solved together. Thanks to the double-diagonal form, this step is computationally efficient.
- Let's look at a different example to clarify how encoding works:
- Suppose our message vector is m = [m_1 m_2 m_3 m_4]
- We append 4 parity bits to form the complete codeword: [p_1 p_2 p_3 p_4]
- Final encoded vector (codeword) = [v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8]
- The base matrix used in this case is shown below:

$$H = \begin{bmatrix} I_1 & 0 & I_3 & I_1 & I_2 & I & 0 & 0 \\ I_2 & I & 0 & I_3 & 0 & I & I & 0 \\ 0 & I_4 & I_2 & I & I_1 & 0 & I & I \\ I_4 & I_1 & I & 0 & I_2 & 0 & 0 & I \end{bmatrix}$$

Expansion: 5

Encoding Using Base Matrix

$$\begin{bmatrix} I_1 & 0 & I_3 & I_1 & I_2 & I & 0 & 0 \\ I_2 & I & 0 & I_3 & 0 & I & I & 0 \\ 0 & I_4 & I_2 & I & I_1 & 0 & I & I \\ I_4 & I_1 & I & 0 & I_2 & 0 & 0 & I \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = 0$$

Message m = [m1 m2 m3 m4]
Codeword c = [v1 v2 v3 v4 v5 v6 v7 v8]

From the matrix structure, we get the following equations:

$I_1v_1 + I_3v_3 + I_1v_4 + I_2v_5 + Iv_6 = 0$ **-Equation 1**
$I_2v_1 + Iv_2 + I_3v_4 + Iv_6 + Iv_7 = 0$ **-Equation2**
$I_4v_2 + I_2v_3 + Iv_4 + I_1v_5 + Iv_7 + Iv_8 = 0$ **-Equation 3**
$I_4v_1 + I_1v_2 + Iv_3 + I_2v_5 + Iv_8 = 0$ **-Equation 4**

Adding all four equations:

$I_1v_5 = I_1v_1 + I_2v_1 + I_4v_1 + Iv_2 + I_4v_2 + I_1v_2 + I_3v_3 + I_2v_3 + Iv_3 + I_1v_4 + I_3v_4 + Iv_4$

We know that $[v_1\ v_2\ v_3\ v_4\ v_5\ v_6\ v_7\ v_8] = [m_1\ m_2\ m_3\ m_4\ p_1\ p_2\ p_3\ p_4]$

So, rewriting the equation, we get:

$I_1p_1 = I_1m_1 + I_2m_1 + I_4m1 + Im_2 + I_4m_2 + I_1m_2 + I_3m_3 + I_2m_3 + Im_3 + I_1m_4 + I_3m_4 + Im_4$

From here we can get p_1.Substitute it in equation1 to get p_2 and solve so on to get all the 4 parity bits..

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

BPSK Modulation and AWGN Noise

The encoded bits are sent through a BPSK modulator, which maps:

- 1 → −1
- 0 → +1

These symbols are then transmitted over an AWGN channel, which adds noise to simulate real-world conditions.
The noise standard deviation is given by:

$$\sigma = \frac{1}{\sqrt{2 \cdot E_b/N_0 \cdot r}}$$

where Eb/No is the signal-to-noise ratio per bit, and r is the code rate.

Demodulation

- We can demodulate the message by using the following scheme: the threshold here is zero, the value of demodulated bit depends on whether the value is greater than or less than zero.

- If the voltage value is less than zero or negative, the demodulation of thebit gives us one bit, otherwise, if it is greater than or equal to zero, the demodulation gives us zero bit.

- [0.54, -0.12, -1.31, 0.77, 1.21] ⟶ [0, 1, 1, 0, 0]

Hard vs Soft decoding

HARD Decision

- The demodulator makes a firm or hard decision whether one or zero is transmitted and provides no other information regarding how reliable the decision is.
- Hence, its output is only zero or one (the out put is quantized only to two level) which are called “hard bits”

SOFT Decision

- The demodulator provides the decoder with some side information(belief) together with the decision.
- The side information provides the decoder with a measure of confidence for the decision.
- The demodulator outputs which are called soft-bits, are quantized to two levels.

Tanner Graph

- These are a type of graphical representation used in coding theory and information theory.
- A Tanner graph is a bipartite graph used to represent linear error-correcting codes. It consists of two sets of vertices: variable nodes (representing bits) and check nodes (representing constraints or parity checks).
- In a Tanner graph, variable nodes are connected to check nodes, and vice versa, through edges. Each edge represents the relationship between a variable node and a check node in the code's parity-check matrix.

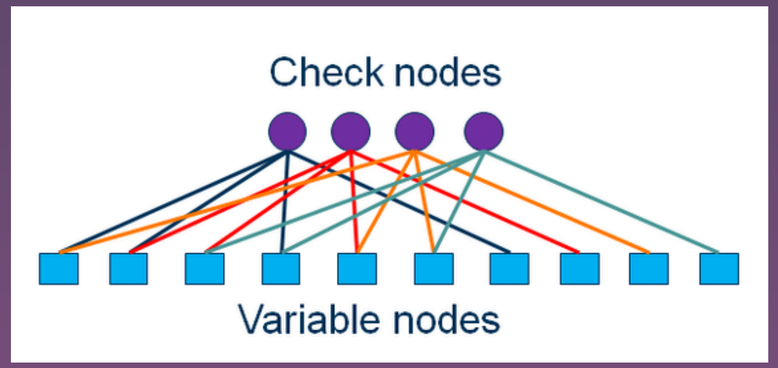


Image source: https://en.wikipedia.org/wiki/File:Tanner_graph%281%29.png

Tanner Graph

- In our code we are implementing tanner graph using the matrices.
- We initialize two matrices:
- One for variable nodes to check nodes message passing (VN to CN).
- While the other for check nodes to variable nodes message passing (CN to VN).

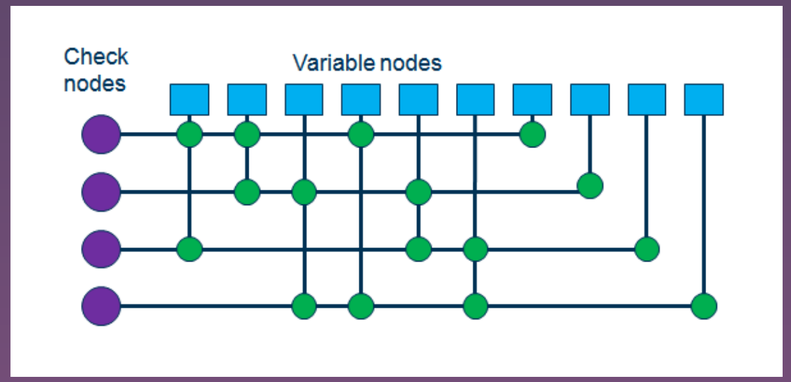
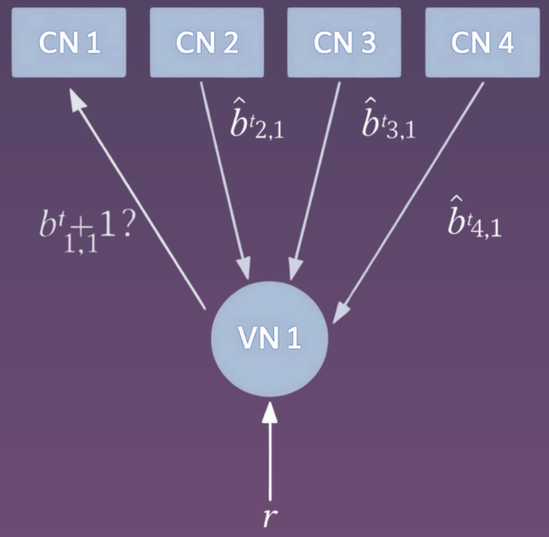


Image source: https://en.wikipedia.org/wiki/File:Tanner_graph%281%29.png

Hard Decision Decoding



VN to CN Message Passing

- First we load all the received values into the VN.
- Then, for first iteration: We directly send the received value to all connected CNs of respective VNs.

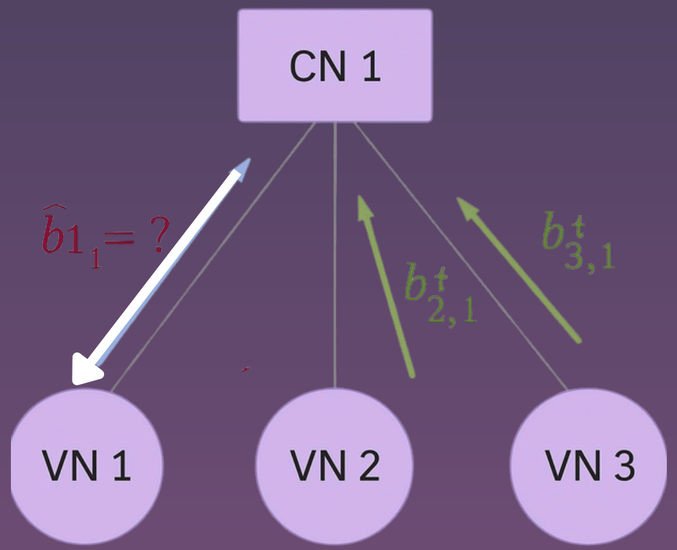
- For all other iterations:
- Each Variable Node (VN) sends messages to connected Check Nodes (CNs) in every iteration.
- To send a message to CN i in iteration t+1:
- VN takes the messages received from all other CNs (≠ i) in iteration t.
- Includes its original received bit from the channel.
- Performs majority voting on these values.
- Sends the result as the new message to CN i.

Example (as in the diagram):

- VN1 sends $b^{t+1}_{1,1}$ to CN1 using:

- $b^t_{2,1}$, $b^t_{3,1}$, $b^t_{4,1}$, and received bit r.

Hard Decision Decoding



CN to VN Message Passing

- Each Check Node (CN) applies XOR logic (Single Parity Check code).

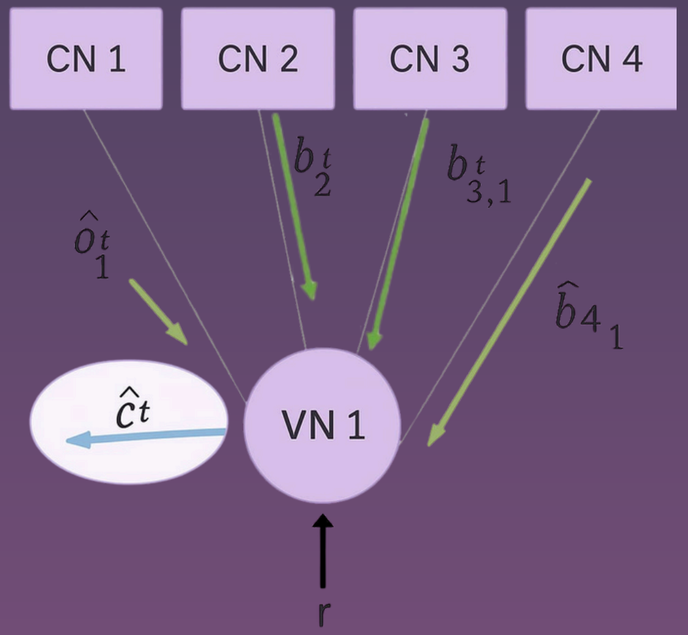
To send a message to VN j:

- CN takes all bits received from other VNs (≠ j).
- Performs XOR on them.
- Sends the result to VN j.

Example (from diagram):

- To send $\hat{b}^t_{1,1}$ to VN1, CN1 computes:
- $b^t_{2,1} \oplus b^t_{3,1}$

Hard Decision Decoding



Computing ĉ

- After each iteration, estimate ĉ using majority voting.

For each VN:

- Combine the original received bit and all messages from connected CNs.
- Use majority rule to decide the final bit value.

Repeat iterations until:

- Reached maximum iterations, or
- ĉ matches the encoded codeword, or
- $\hat{c}^{(t)} = \hat{c}^{(t-1)}$ (no change).

Soft Decision Decoding

Algorithm

- The decoder receives real-valued inputs from the channel, which are interpreted as Log-Likelihood Ratios (LLRs).
- Initially, these LLRs are assigned to the Variable Nodes (VNs). Then, messages are sent from each VN to its connected Check Nodes (CNs), computed using the formula below:

$$L(r_{ji}) = \log \frac{r_{ji}(0)}{r_{ji}(1)} = 2 \tanh^{-1} \left(\prod_{r \in V_j \setminus i} \tanh \left(\frac{1}{2} L(q_{rj}) \right) \right)$$

Alternatively, using the φ-transform:

$$= \left(\prod_{i' \in V_j \setminus i} \alpha_{ij} \right) \cdot \varphi \left(\sum_{i' \in V_j \setminus i} \varphi(\beta_{rj}) \right)$$

Where:

$$\alpha_{ij} \equiv \text{sign}(L(q_{rj})) \quad \text{and} \quad \beta_{rj} \equiv |L(q_{rj})|$$

$$\varphi(x) \equiv \log \left(\frac{e^x + 1}{e^x - 1} \right)$$

Computing this for each CN at every iteration is computationally intensive. To simplify the process, the Min-Sum approximation is used for VN-to-CN message updates.

cont.

MIN-SUM Approximation

- Because of the behavior of the function φ(x), the total sum is largely influenced by the smallest value of x. Hence, we approximate:

$$\sum_{i' \in V_j \setminus i} \varphi(\beta_{i'j}) \approx \varphi\left(\min_{i' \in V_j \setminus i} \beta_{i'j}\right)$$

- Using this Min-Sum approximation, the message passed from CN to VN simplifies to:

$$L(r_{ji}) = \prod_{i' \in V_j \setminus i} \alpha_{ij} \cdot \min_{i' \in V_j \setminus i} \beta_{i'j}$$

- Once CNs send messages to VNs, each VN updates and sends its next message to all connected CNs using:

$$L(q_{ij}) = L(c_i) + \sum_{j' \in C_i \setminus j} L(r_{j'i})$$

cont.

Final Decision Making

- After each iteration, we compute C, the decoded message. This is done by evaluating:

$$L(Q_i) = L(c_i) + \sum_{j' \in C_i} L(r_{j'i})$$

- Using this value, each decoded bit $\hat{c}_i$ is determined as follows:

$$\hat{c}_i = \begin{cases} 1, & \text{if } L < 0 \\ 0, & \text{otherwise} \end{cases}$$

- The final decoded message $\hat{C}$ is accepted either when:
  - The maximum number of iterations is reached, or
  - The syndrome condition $H \cdot \hat{C} = 0$ is satisfied.

SISO Decoder SPC (3,2)

$$C_1 = C_2 \oplus C_3$$

$$l_2 = \log \frac{\Pr(C_2 = 0 \mid Y_2)}{\Pr(C_2 = 1 \mid Y_2)}, \quad l_3 = \log \frac{\Pr(C_3 = 0 \mid Y_3)}{\Pr(C_3 = 1 \mid Y_3)}$$

Given p_2 and p_3, what is $\Pr(C_1 = 0 \mid p_2, p_3)$?

$$l_2 = \log \frac{p_2}{1 - p_2}, \quad l_3 = \log \frac{p_3}{1 - p_3}$$

C_1	C_2	C_3
0	0	0
0	1	1
1	0	1
1	1	0

$$p_1 = \Pr(C_1 = 0) = p_2 p_3 + (1 - p_2)(1 - p_3)$$

$$1 - p_1 = \Pr(C_1 = 1) = p_2(1 - p_3) + p_3(1 - p_2)$$

$$p_1 - (1 - p_1) = p_2 p_3 + (1 - p_2)(1 - p_3) - p_2(1 - p_3) - p_3(1 - p_2)$$

$$p_1 - (1 - p_1) = (p_2 - (1 - p_2))(p_3 - (1 - p_3))$$

$$\frac{p_1 - (1 - p_1)}{p_1 + (1 - p_1)} = \frac{p_2 - (1 - p_2)}{p_2 + (1 - p_2)} \cdot \frac{p_3 - (1 - p_3)}{p_3 + (1 - p_3)}$$

$$\frac{1 - \frac{1-p_1}{p_1}}{1 + \frac{1-p_1}{p_1}} = \frac{1 - \frac{1-p_2}{p_2}}{1 + \frac{1-p_2}{p_2}} \cdot \frac{1 - \frac{1-p_3}{p_3}}{1 + \frac{1-p_3}{p_3}}$$

$$l_{\text{ext},1} = \log \frac{p_1}{1 - p_1}$$

$$\Rightarrow \frac{1 - e^{-l_{\text{ext},1}}}{1 + e^{-l_{\text{ext},1}}} = \left(\frac{1 - e^{-l_2}}{1 + e^{-l_2}}\right)\left(\frac{1 - e^{-l_3}}{1 + e^{-l_3}}\right)$$

Check Node Update using Tanh Rule

The check node update in the log-likelihood ratio (LLR) domain uses the following identity:

$$\tanh\left(\frac{L_{\text{ext},1}}{2}\right) = \tanh\left(\frac{\ell_2}{2}\right) \cdot \tanh\left(\frac{\ell_3}{2}\right)$$

Sign Rule

$$\text{sign}(L_{\text{ext},1}) = \text{sign}(\ell_2) \cdot \text{sign}(\ell_3)$$

Absolute Value Rule

$$\tanh\left(\frac{|L_{\text{ext},1}|}{2}\right) = \tanh\left(\frac{|\ell_2|}{2}\right) \cdot \tanh\left(\frac{|\ell_3|}{2}\right)$$

Log Transformation

Define the function:

$$f(x) = \log\left(\tanh\left(\frac{x}{2}\right)\right), \quad x > 0$$

This leads to:

$$f(|L_{\text{ext},1}|) = f(|\ell_2|) + f(|\ell_3|)$$

Inverse Transformation

Apply the inverse function:

$$|L_{\text{ext},1}| = f^{-1}\left(f(|\ell_2|) + f(|\ell_3|)\right)$$

Final Check Node Update Equation

$$L_{\text{ext},1} = \text{sign}(\ell_2 \cdot \ell_3) \cdot f^{-1}\left(f(|\ell_2|) + f(|\ell_3|)\right)$$

SISO Decoder Repetition (3,1)

Repetition Code SISO Decoder (3,1) Repetition Code

- *Message bits:* C_1
- *Received bits:* r_1, r_2, r_3

$$\Pr(C_1 = 0 \mid r_1) = \frac{f(r_1 \mid C_1 = 0)\Pr(C_1 = 0)}{f(r_1)}$$

$$\Pr(C_1 = 1 \mid r_1) = \frac{f(r_1 \mid C_1 = 1)\Pr(C_1 = 1)}{f(r_1)}$$

Posterior Ratio

$$\frac{\Pr(C_1 = 0 \mid r_1)}{\Pr(C_1 = 1 \mid r_1)} = \frac{f(r_1 \mid C_1 = 0)}{f(r_1 \mid C_1 = 1)}$$

$$f(r_1 \mid C_1 = 0) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(r_1-1)^2/(2\sigma^2)}, \quad f(r_1 \mid C_1 = 1) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(r_1+1)^2/(2\sigma^2)}$$

$$\text{LLR} = \log\left(\frac{f(r_1 \mid C_1 = 0)}{f(r_1 \mid C_1 = 1)}\right) = \frac{2}{\sigma^2} \cdot r_1 \quad \text{(intrinsic)}$$

Output LLR

$$LLR = \log\left(\frac{\Pr(C_1 = 0 \mid r_1, r_2, r_3)}{\Pr(C_1 = 1 \mid r_1, r_2, r_3)}\right)$$

LLR Derivation for Repetition Code under AWGN

We want to compute the LLR for a repetition code of length 3 given received values r_1, r_2, r_3 and assuming AWGN with noise variance σ^2.

$$\text{LLR} = \log\left(\frac{f(r_1, r_2, r_3 \mid c = 0)}{f(r_1, r_2, r_3 \mid c = 1)}\right)$$

Assume BPSK: bit $0 \to +1$, bit $1 \to -1$. So for $c = 0$, all transmitted symbols are $+1$, and for $c = 1$, all are -1.

$$= \log\left(\frac{\prod_{i=1}^{3} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(r_i-1)^2}{2\sigma^2}\right)}{\prod_{i=1}^{3} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(r_i+1)^2}{2\sigma^2}\right)}\right)$$

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

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

General Form for Repetition Code of Length n:

$$L = \frac{2}{\sigma^2}\left(\sum_{i=1}^{n} r_i\right)$$

SPA Code Example

- **Encoded:** c_1, c_2, c_3
- **Received:** r_1, r_2, r_3
- **Decoded:**

$$L_1 = l_1 + L_{\text{ext1}}$$
$$L_2 = l_2 + L_{\text{ext2}}$$
$$L_3 = l_3 + L_{\text{ext3}}$$

Derivation :

Min-Sum Approximation in Soft Decoding

$$L_3 = l + \text{Lext}_3$$
$$\text{sgn}(\text{Lext}_1) = \text{sgn}(l_2) \cdot \text{sgn}(l_3)$$

Define function:

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

Then:

$$|\text{Lext}_1| = f\left(f(|l_2|) + f(|l_3|)\right)$$

Now, for large values of l_2 and l_3, the function $f(x)$ is small. So, we approximate for small inputs:

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

Therefore:

$$|\text{Lext}_1| = f\left(f(\min(|l_2|, |l_3|))\right)$$

Since $f^{-1}(x) = f(x)$, we get:

$$|\text{Lext}_1| = \min(|l_2|, |l_3|)$$

Conclusion: Hence, the *min-sum algorithm* is a simplified approximation for soft decoding:

- SPA for CN $\rightarrow$ VN uses log-tanh - Min-sum for CN $\rightarrow$ VN uses only minimum operation

Algorithm Implementation

- To implement the soft-decision decoder in MATLAB, we utilize the parity-check matrix H, where each row corresponds to a Check Node (CN) and each column to a Variable Node (VN). A value of 1 at position H(i,j) indicates a connection (edge) between CN i and VN j in the Tanner graph.
- Based on this Tanner graph representation, we create a copy of the matrix H, denoted as L, and initialize each 1 in L with the received Log-Likelihood Ratio (LLR), effectively assigning the received LLRs to the corresponding VNs.
- We then apply the Min-Sum approximation algorithm to perform message passing from CNs to VNs. This involves, for each row, replacing each non-zero entry with the minimum of all other non-zero values in that row, multiplied by the appropriate parity sign.
- Following this, message passing is performed from VNs to CNs by summing all values in each column of L. The resulting values represent the updated LLRs, which are used to determine the decoded bits C^. We then check whether additional iterations are needed, based on convergence or a stopping condition.

Example for Better Understanding

Given Parity Check Matrix `H`:

Let:

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

This implies:

- 3 Check Nodes (CN)
- 7 Variable Nodes (VN)

Received LLR Vector `r`:

Let the received LLRs be:

$$r = [0.4, -0.8, 1.1, -0.6, 0.3, 0.5, -1.0]$$

Copy the structure of `H`, and replace each `1` with its corresponding value from `r`.

$$L = \begin{bmatrix} 0.4 & 0 & 1.1 & -0.6 & 0 & 0 & -1.0 \\ 0 & -0.8 & 1.1 & 0 & 0.3 & 0.5 & 0 \\ 0.4 & -0.8 & 0 & -0.6 & 0 & 0.5 & 0 \end{bmatrix}$$

Step 1: CN to VN Update (Row-wise Min-Sum Update)

Let's do for **Row 1 (CN1)**:

Values: `[0.4, 1.1, -0.6, -1.0]`

- Absolute values: `[0.4, 1.1, 0.6, 1.0]`
- `min1 = 0.4`, `min2 = 0.6`
- Overall sign product = sign(0.4 × 1.1 × -0.6 × -1.0) = positive

→ Updated row:

$$[0.6,\ 0,\ 0.4,\ -0.4,\ 0,\ 0,\ -0.4]$$

🔸 Row 2 (CN2):

Non-zero entries: `[-0.8, 1.1, 0.3, 0.5]`

- Absolute values: `[0.8, 1.1, 0.3, 0.5]`
- `min1 = 0.3`, `min2 = 0.5`
- Sign product: `sign(-0.8 * 1.1 * 0.3 * 0.5) = -`

→ **Updated row 2**: `[0, -0.3, -0.3, 0, -0.5, -0.3, 0]`

Updated values:

Position	Value	Updated
2 (-0.8)	min1	-0.3
3 (1.1)	min1	-0.3
5 (0.3)	min2	-0.5
6 (0.5)	min1	-0.3

🔸 Row 3 (CN3):

Non-zero entries: `[0.4, -0.8, -0.6, 0.5]`

- Absolute values: `[0.4, 0.8, 0.6, 0.5]`
- `min1 = 0.4`, `min2 = 0.5`
- Sign product: `sign(0.4 * -0.8 * -0.6 * 0.5) = +`

→ **Updated row 3**: `[0.5, -0.4, 0, -0.4, 0, 0.4, 0]`

Updated values:

Position	Value	Updated
1 (0.4)	min2	0.5
2 (-0.8)	min1	-0.4
4 (-0.6)	min1	-0.4
6 (0.5)	min1	0.4

☑ Final CN to VN update matrix:

$$L_{\text{CN}\to\text{VN}} = \begin{bmatrix} 0.6 & 0 & 0.4 & -0.4 & 0 & 0 & -0.4 \\ 0 & -0.3 & -0.3 & 0 & -0.5 & -0.3 & 0 \\ 0.5 & -0.4 & 0 & -0.4 & 0 & 0.4 & 0 \end{bmatrix}$$

Step 2: VN to CN Update (Column-wise Sum Update)

The rule is:

For each column:

1. **Sum all values in the column** + original `r[i]` (received LLR).
2. For each cell, compute:

$$\text{New value} = \text{Total Sum} - \text{Current cell value}$$

We sum the non-zero values of each column and add the corresponding $r[i]$:

Column Index	Non-zero values in column	Sum (L values)	$r[i]$	Total
0	0.6, 0.5	1.1	0.4	1.5
1	-0.3, -0.4	-0.7	-0.8	-1.5
2	0.4, -0.3	0.1	1.1	1.2
3	-0.4, -0.4	-0.8	-0.6	-1.4
4	-0.5	-0.5	0.3	-0.2
5	-0.3, 0.4	0.1	0.5	0.6
6	-0.4	-0.4	-1.0	-1.4

Row 1:

- Column 0: $1.5 - 0.6 = 0.9$
- Column 2: $1.2 - 0.4 = 0.8$
- Column 3: $-1.4 - (-0.4) = -1.0$
- Column 6: $-1.4 - (-0.4) = -1.0$

$$\Rightarrow [0.9, 0, 0.8, -1.0, 0, 0, -1.0]$$

Row 2:

- Column 1: $-1.5 - (-0.3) = -1.2$
- Column 2: $1.2 - (-0.3) = 1.5$
- Column 4: $-0.2 - (-0.5) = 0.3$
- Column 5: $0.6 - (\cdot\text{-}0.3) = 0.9$

$$\Rightarrow [0, -1.2, 1.5, 0, 0.3, 0.9, 0]$$

Row 3:

- Column 0: $1.5 - 0.5 = 1.0$
- Column 1: $-1.5 - (-0.4) = -1.1$
- Column 3: $-1.4 - (-0.4) = -1.0$
- Column 5: $0.6 - 0.4 = 0.2$

$$\Rightarrow [1.0, -1.1, 0, -1.0, 0, 0.2, 0]$$

Final Updated Matrix

$$L' = \begin{bmatrix} 0.9 & 0 & 0.8 & -1.0 & 0 & 0 & -1.0 \\ 0 & -1.2 & 1.5 & 0 & 0.3 & 0.9 & 0 \\ 1.0 & -1.1 & 0 & -1.0 & 0 & 0.2 & 0 \end{bmatrix}$$

Repeat Until Convergence

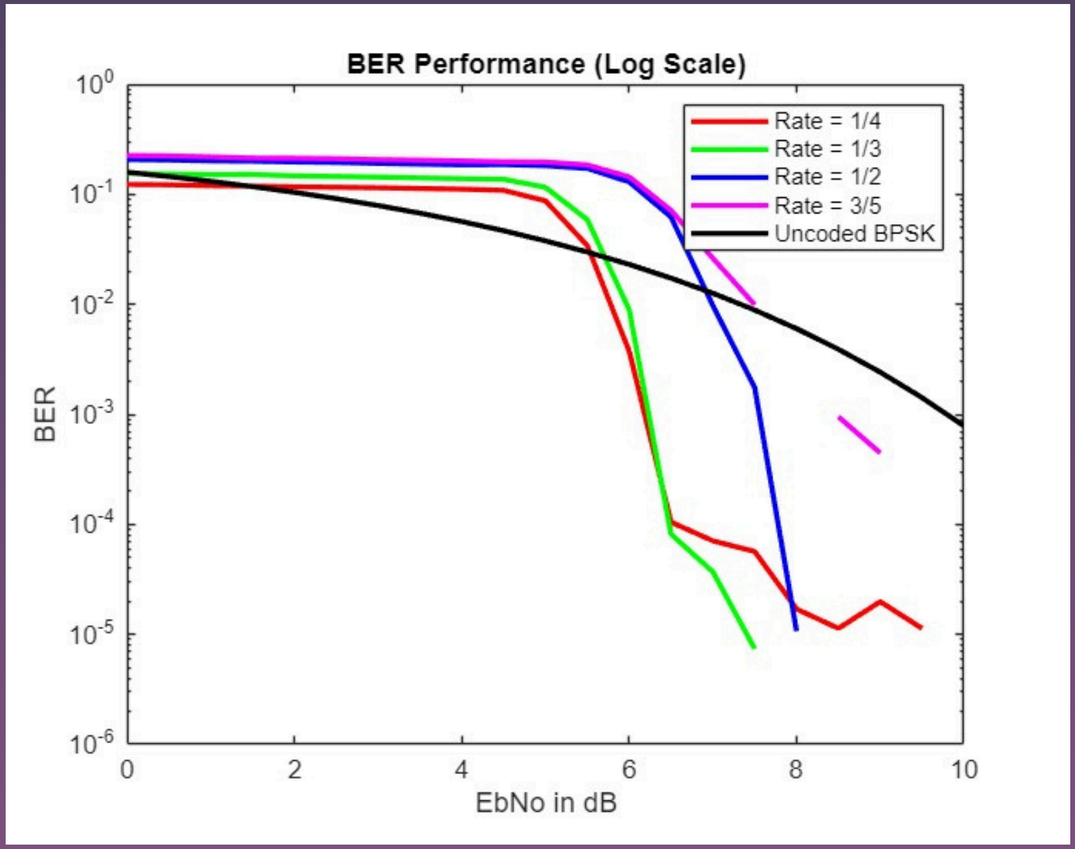
After a few iterations of alternating row and column updates:

- Check if parity check `H x Ĉ' mod 2 = 0`
- If so, decoding is successful

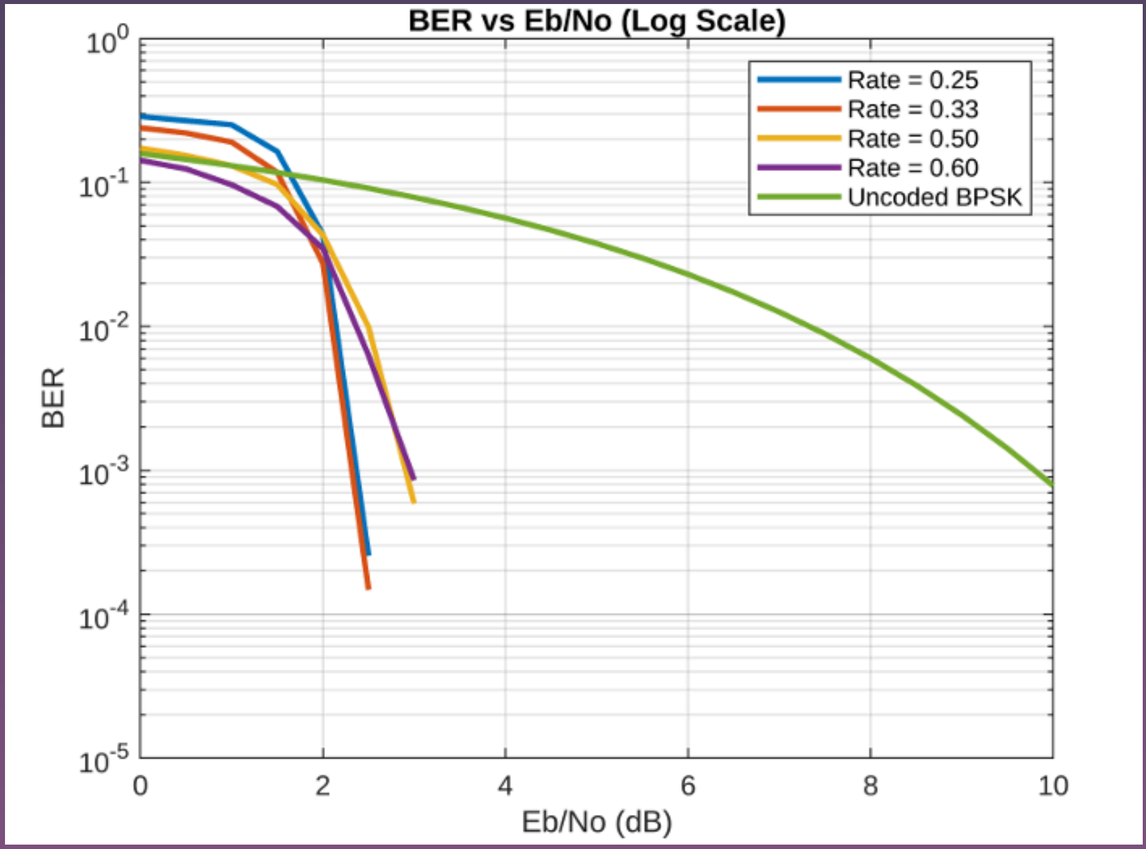
Results of Matrix NR_2_6_52

BER Vs Eb/No (Log Scale)

Hard Decision Decoding

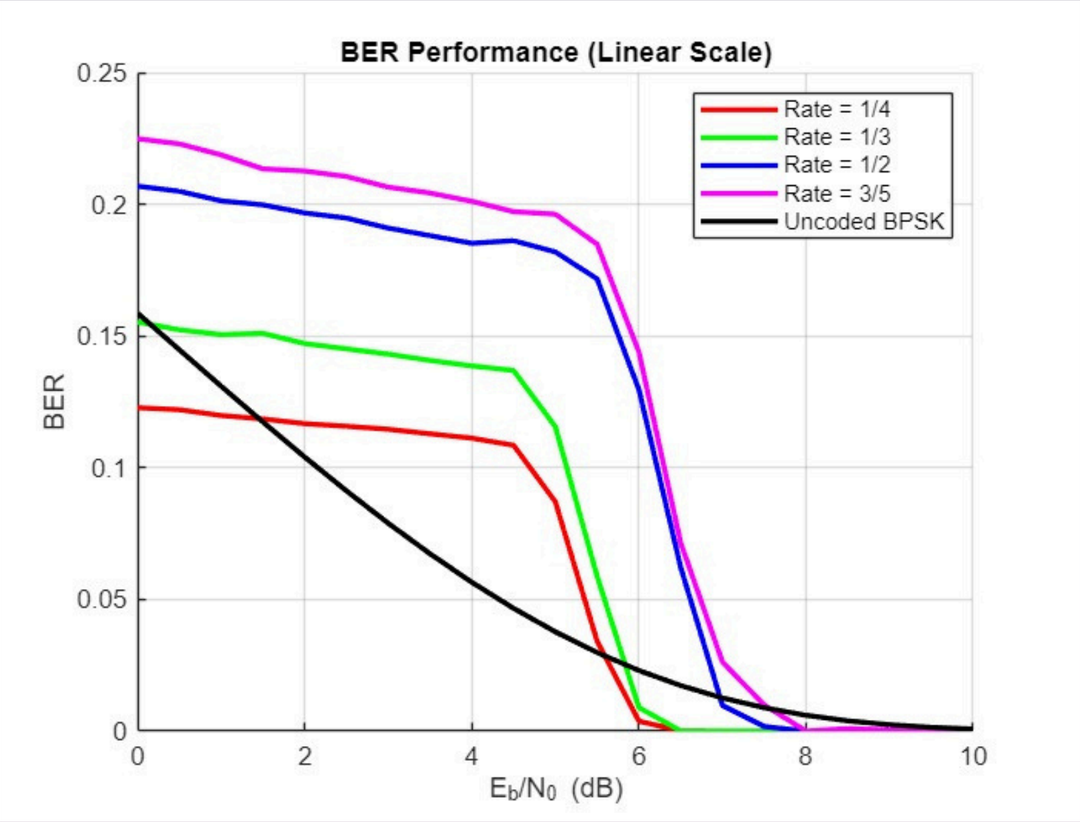


Soft Decision Decoding

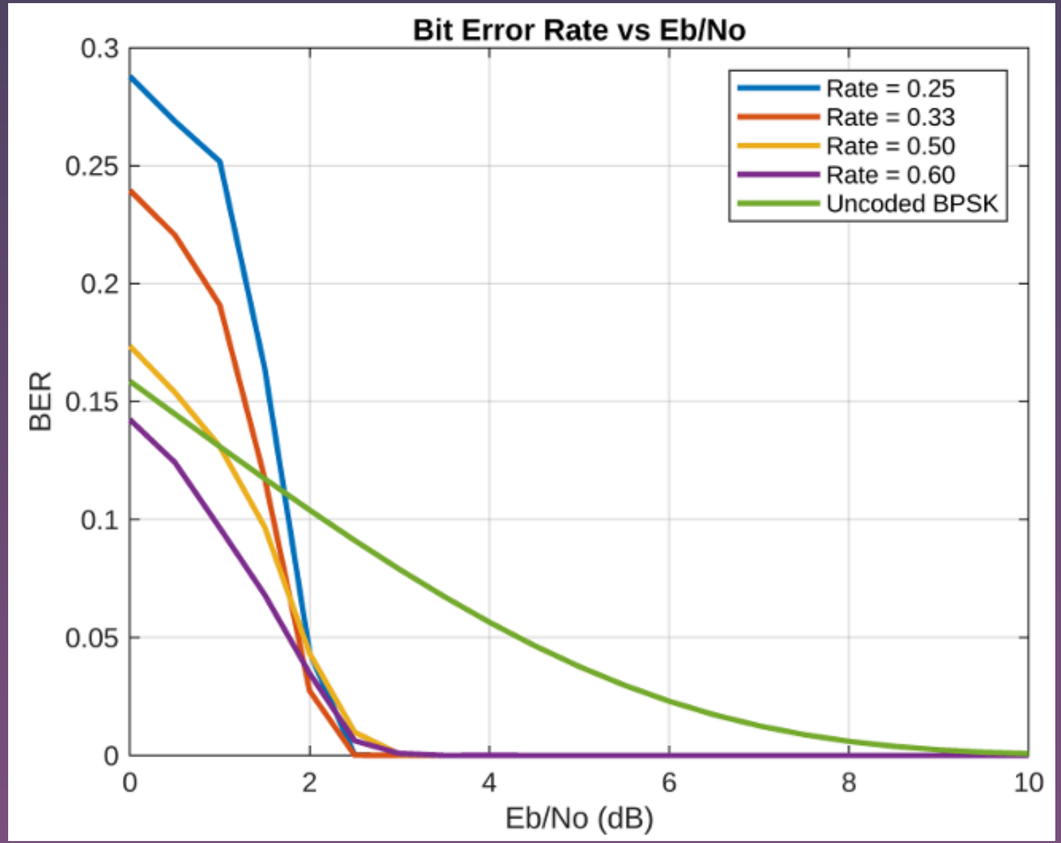


BER Vs Eb/No(Linear scale)

Hard Decision Decoding

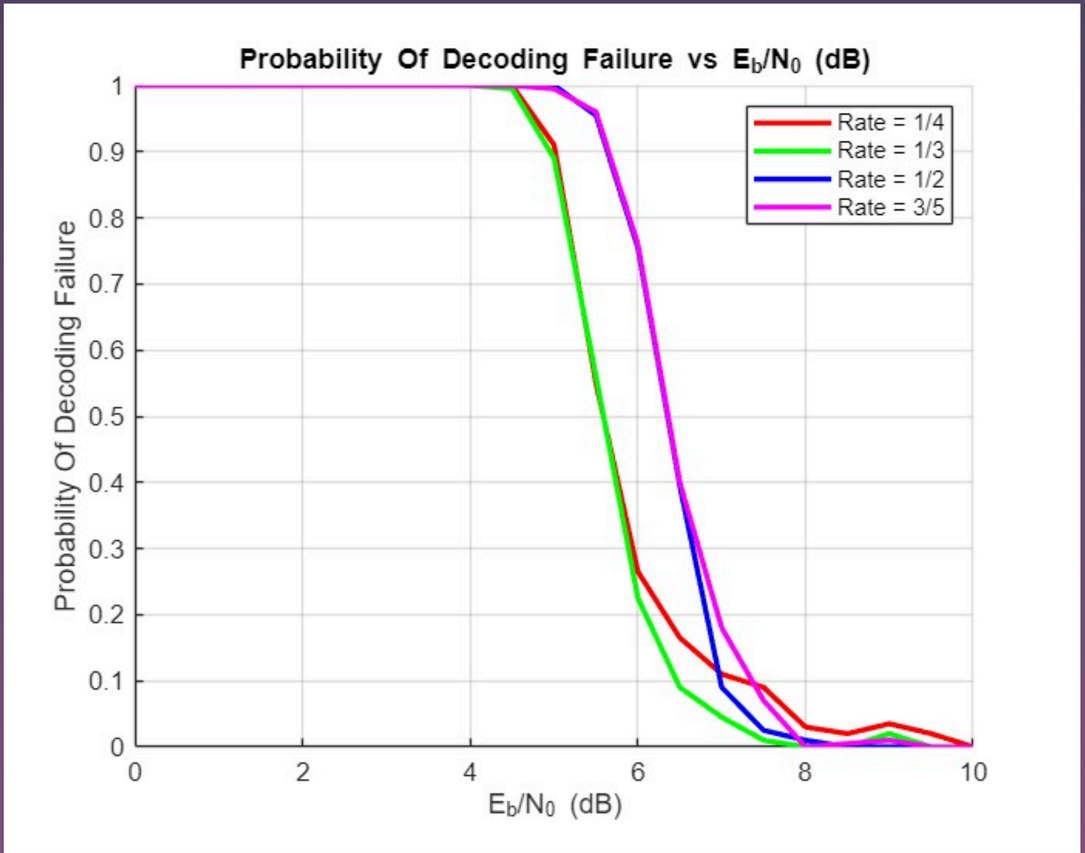


Soft Decision Decoding

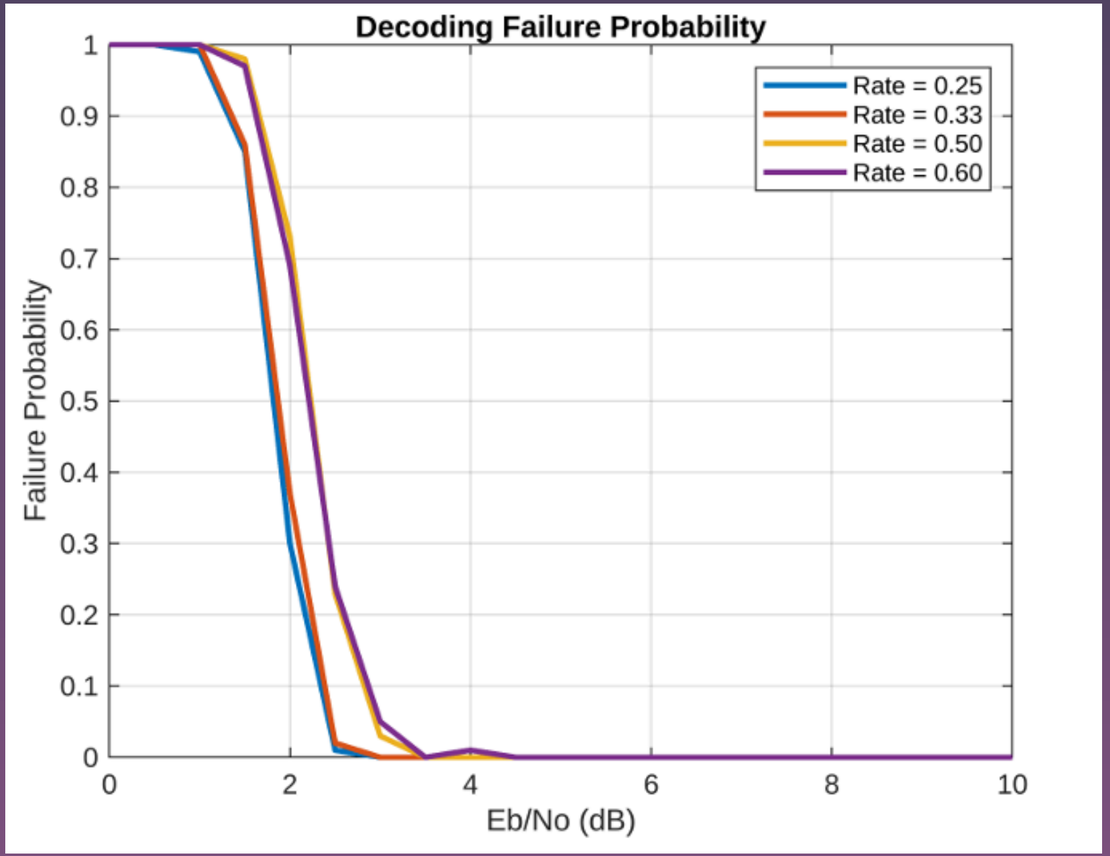


Decoding Failure Probability

Hard Decision Decoding

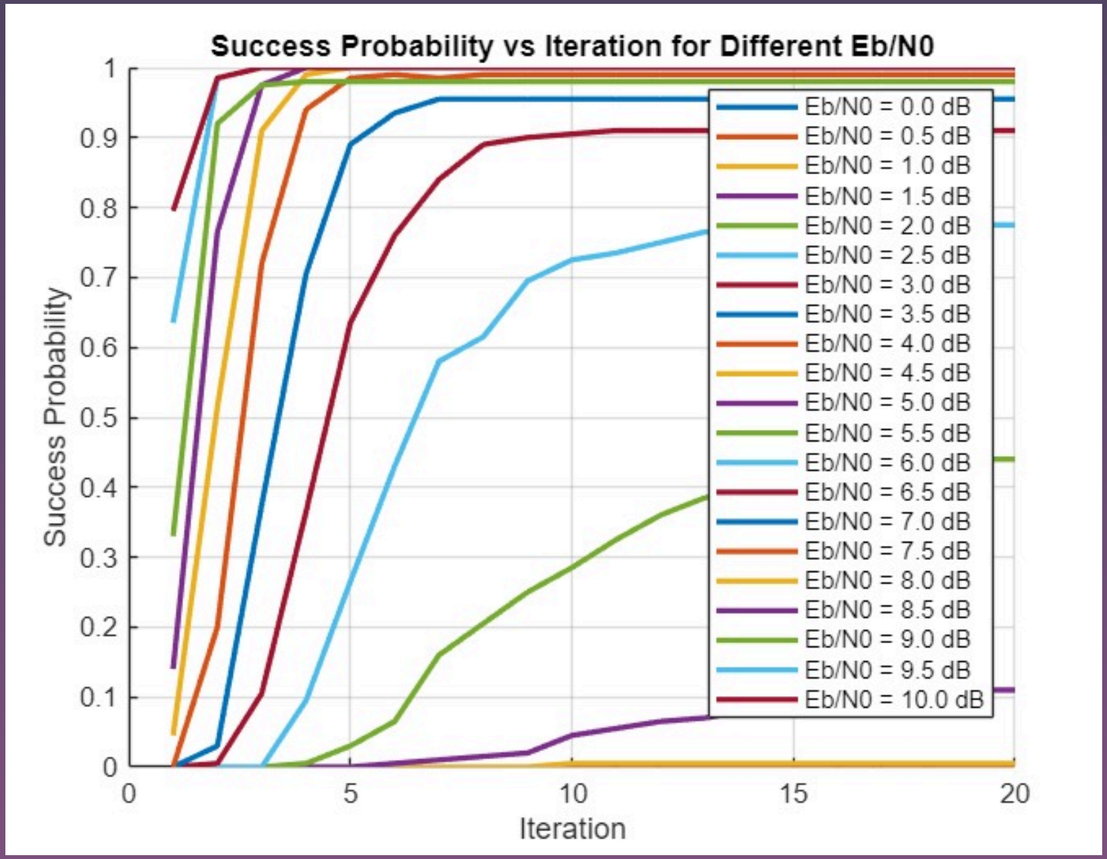


Soft Decision Decoding

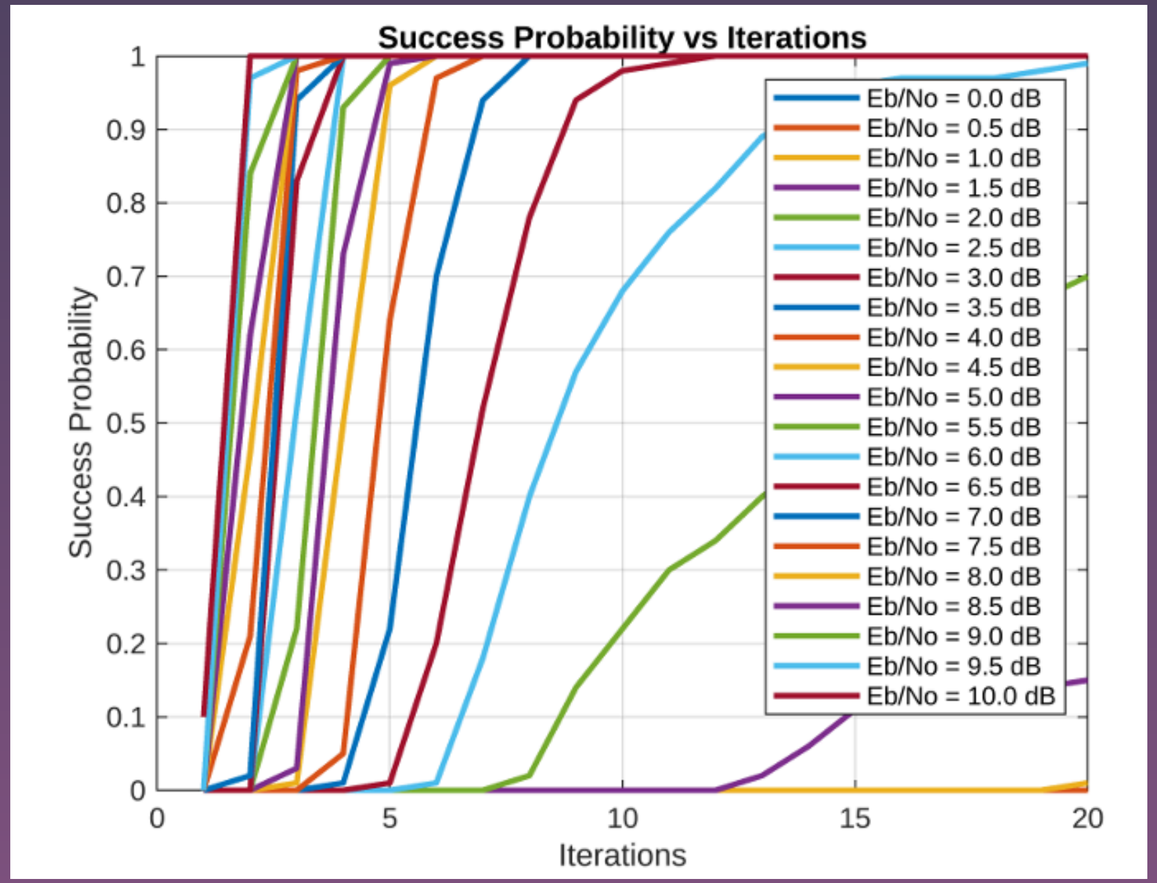


P(success) Vs Iterations for CodeRate = 0.25

Hard Decision Decoding



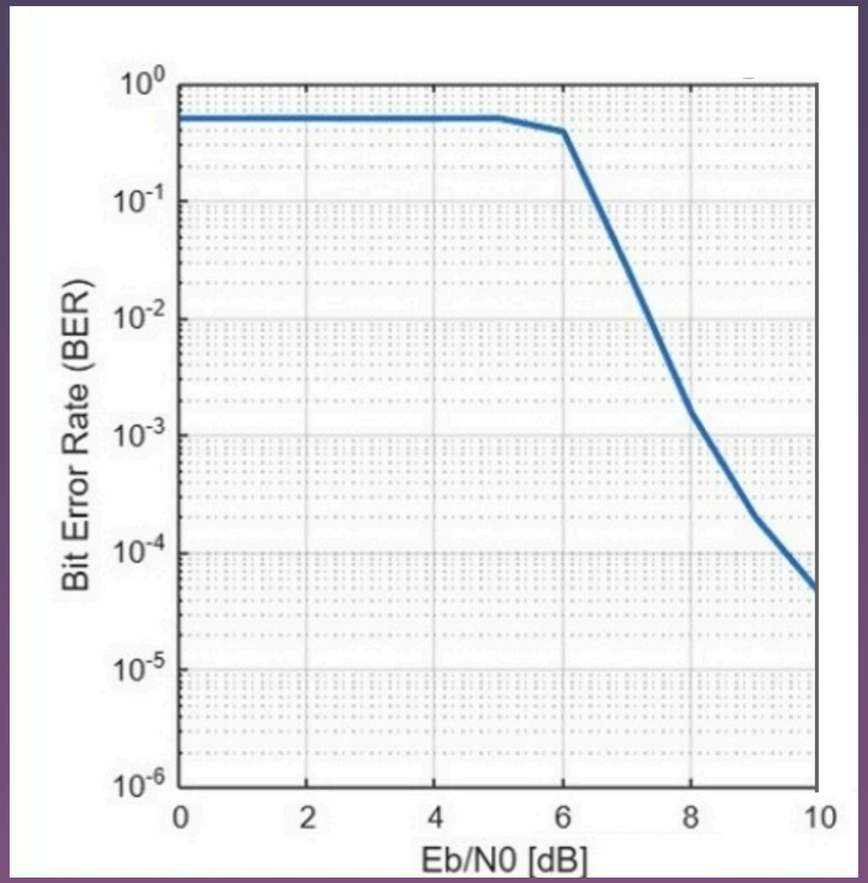
Soft Decision Decoding



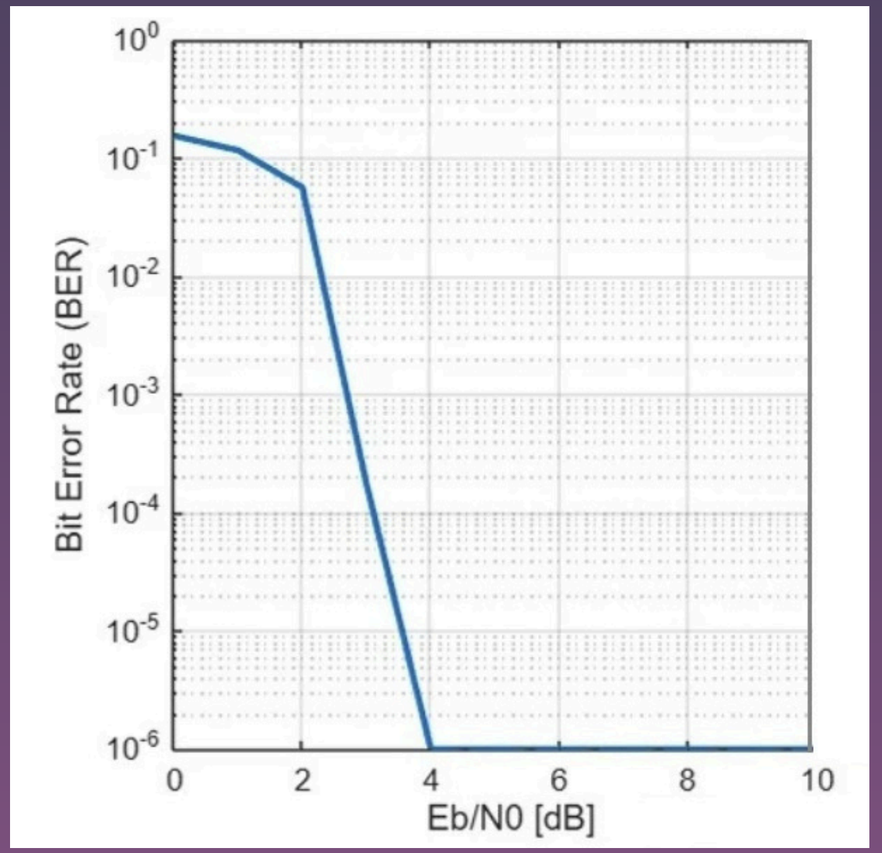
Results of Matrix NR_1_5_352

Bit Error Probability vs Eb/N0 (Log Scale) for CodeRate = 1/3

Hard Decoding

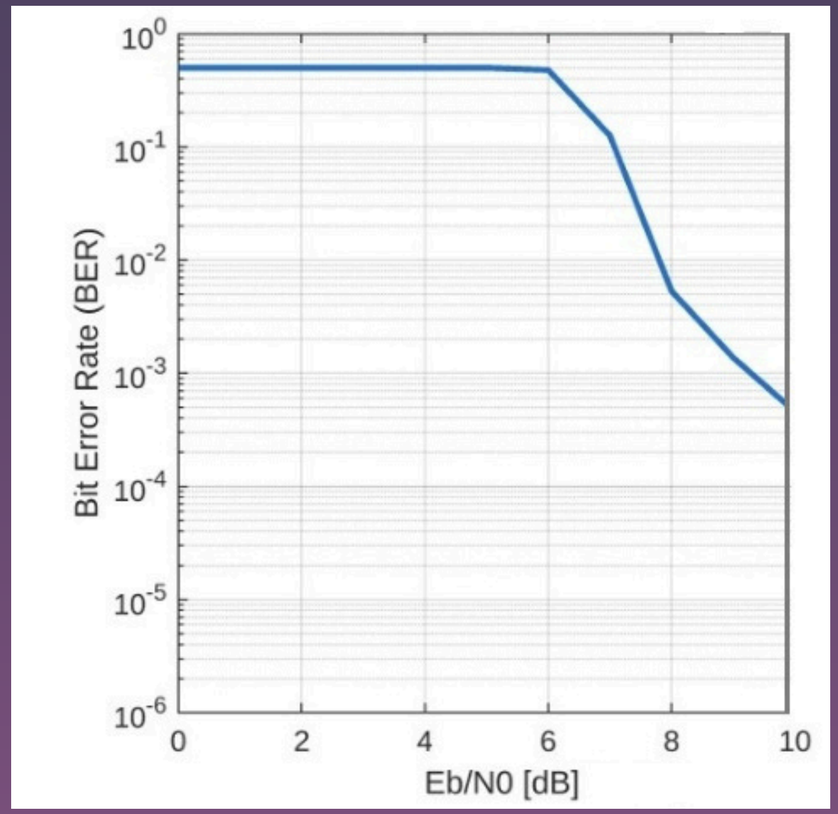


Soft Decoding

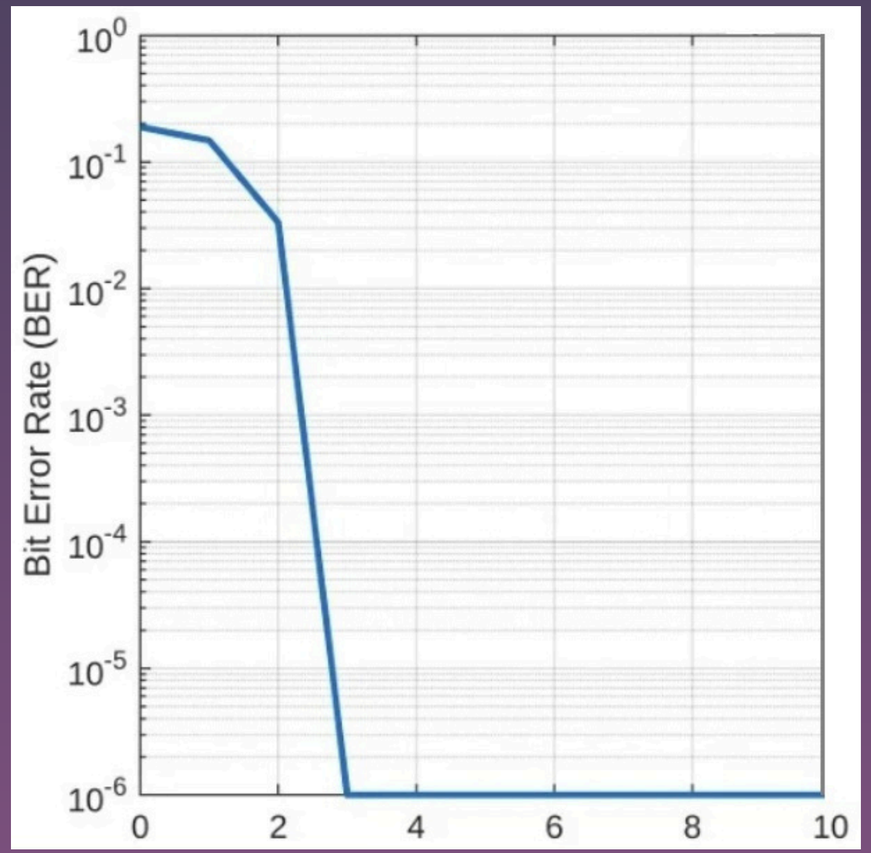


Bit Error Probability vs Eb/N0 (Log Scale) for CodeRate = 1/2

Hard Decoding

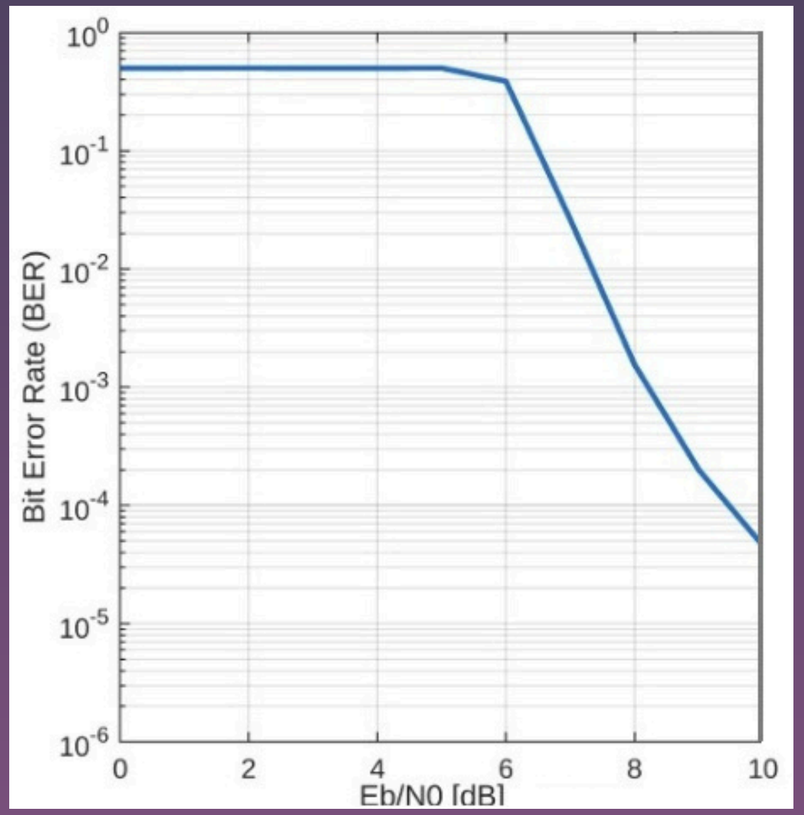


Soft Decoding

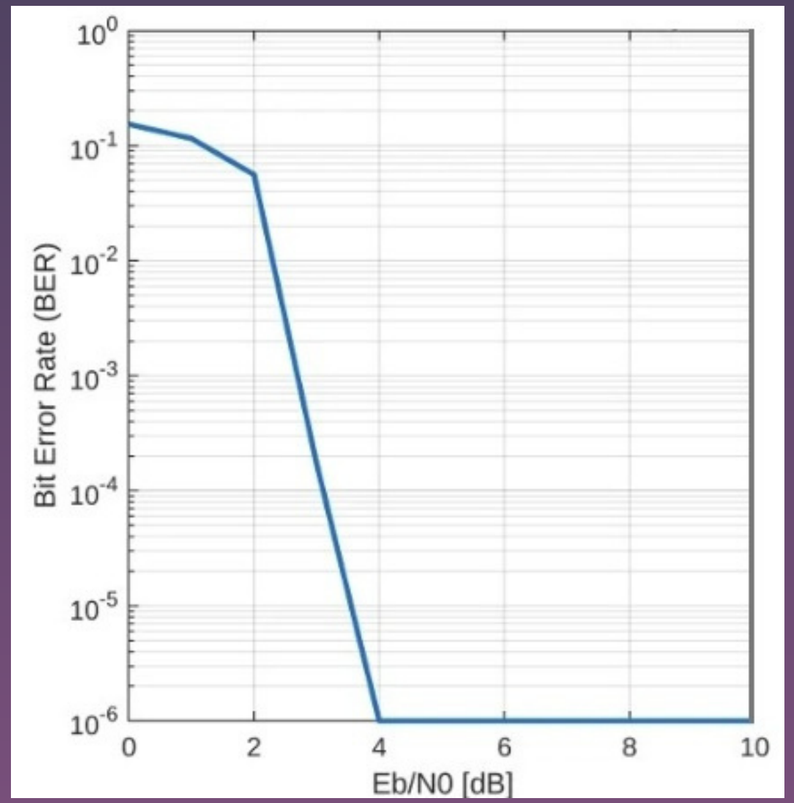


Bit Error Probability vs Eb/N0 (Log Scale) for CodeRate = 3/5

Hard Decoding

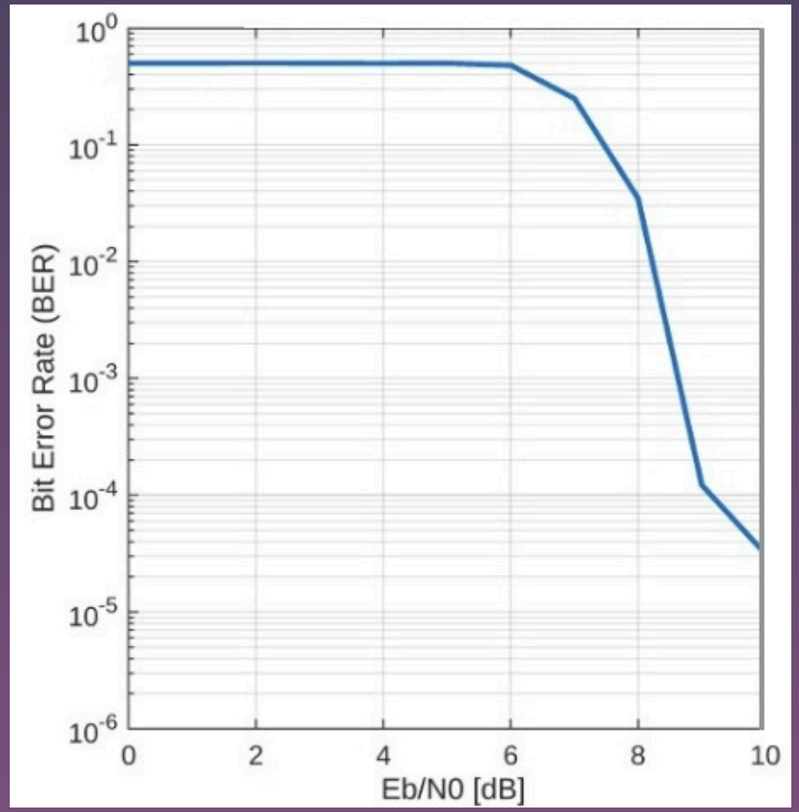


Soft Decoding

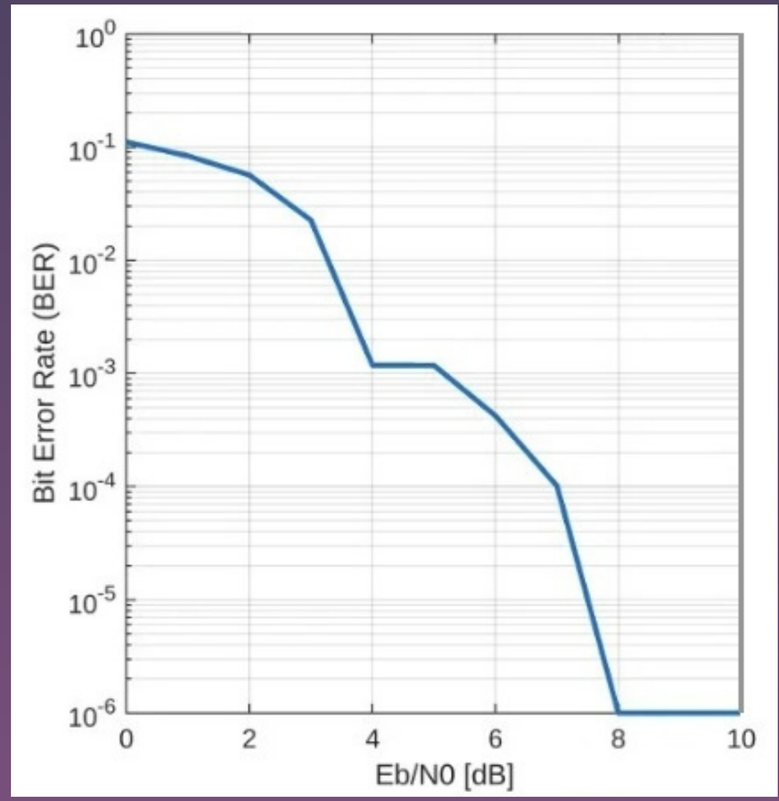


Bit Error Probability vs Eb/N0 (Log Scale) for CodeRate = 4/5

Hard Decoding

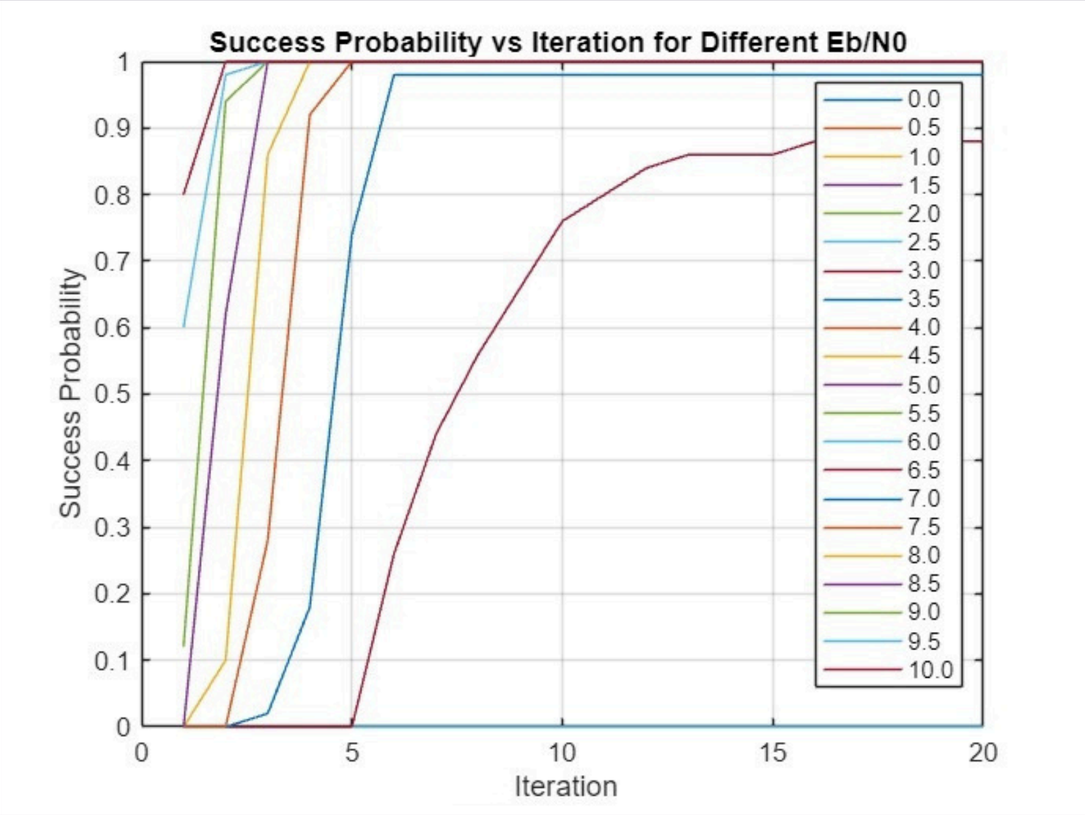


Soft Decoding

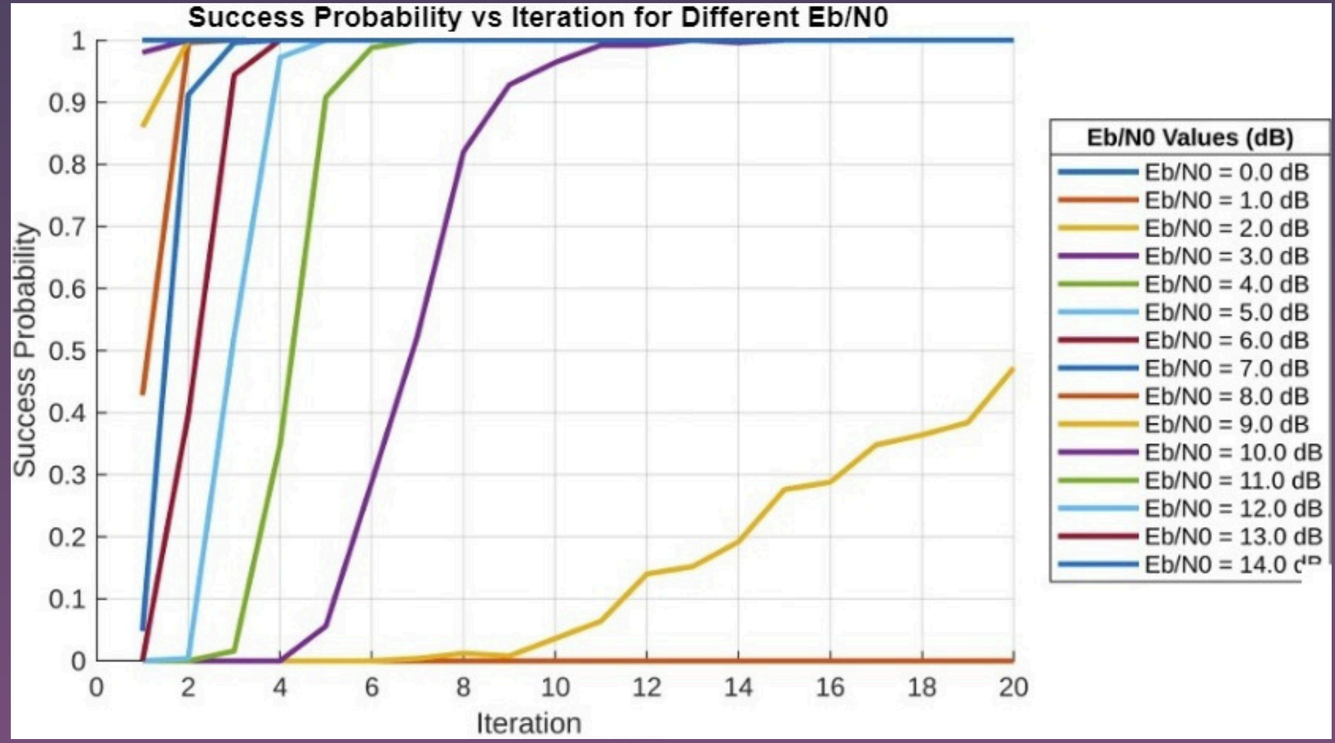


P (success) vs Iteration

Hard Decoding

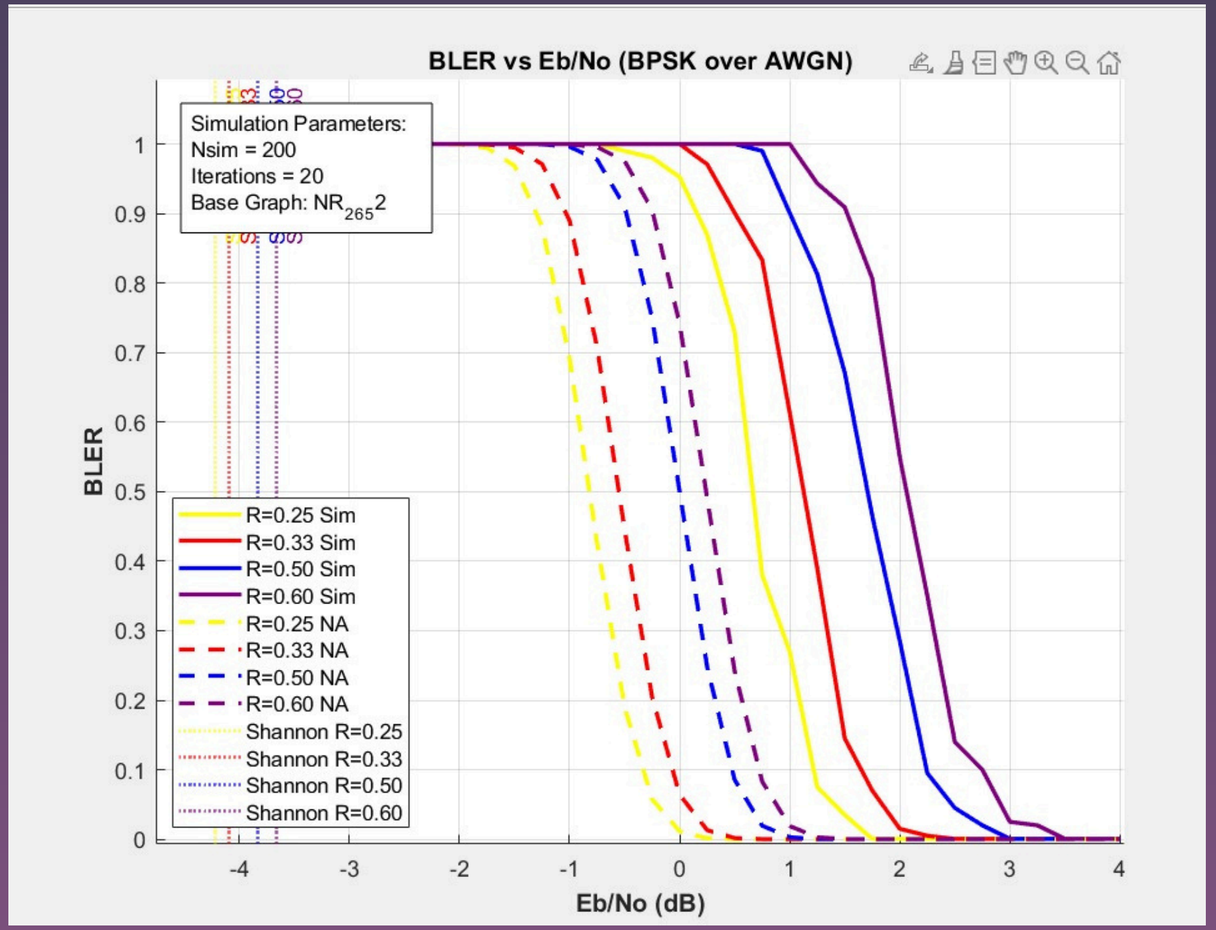


Soft Decoding



Shannon Approximation

Soft Decision Decoding



Conclusion

- Soft decoding proves to be significantly more efficient than hard decoding, as it leverages probabilistic information (beliefs) from other received bits, not just binary decisions.
- With each iteration, the accuracy of decoding improves, increasing the likelihood of successful correction of errors.
- The performance further enhances with lower code rates, as a higher number of parity bits provide more constraints, leading to faster and more reliable decoding.

REFERENCES

- Lecture Slides of Channel Coding by Professor Yash Vasavada.
- Video Lectures of NPTEL-NOM IITM by Prof. Andrew Thangaraj on LDPC codes.
- Implementation of Low-Density Parity-Check codes for 5G NR shared channels by LIFANG WANG

Team Members

202301443 - Vansh Patel

202301444 - Vagh Divyesh

202301445 - Pransu Vadsmiya

202301446 - Jainesh Patel

202301447 - Atik Vohra

202301450 - Shubham Varmora

202301452 - Prem Kukadiya

202301458 - Pranav Bavadiya

202301460 - Akshat Bhatt

202301461 - Ayush Chovatiya

202301462 - Yesha Joshi

THANK YOU :)