CT216

Information and Communication Systems

Prof. Yash Vasavada

(Mentor TA: Aditya Pithadiya)

LDPC Decoding for 5G NR

(End of the Semester Project)

Lab: 5

Project Group: 21

NAME	STUDENT ID
Raiyani Rudra	202301223
Anushka Prajapati	202301224
Dwarkesh Vaghasiya	202301225
Gothi Priyanshi	202301226
Patel Vedant	202301227
Patel Naitikkumar	202301228
Patel Apurv	202301230
Vora Kresha	202301231
Gaadhe Jayansh	202301232
Ayush	202301233

INDEX

We declare that

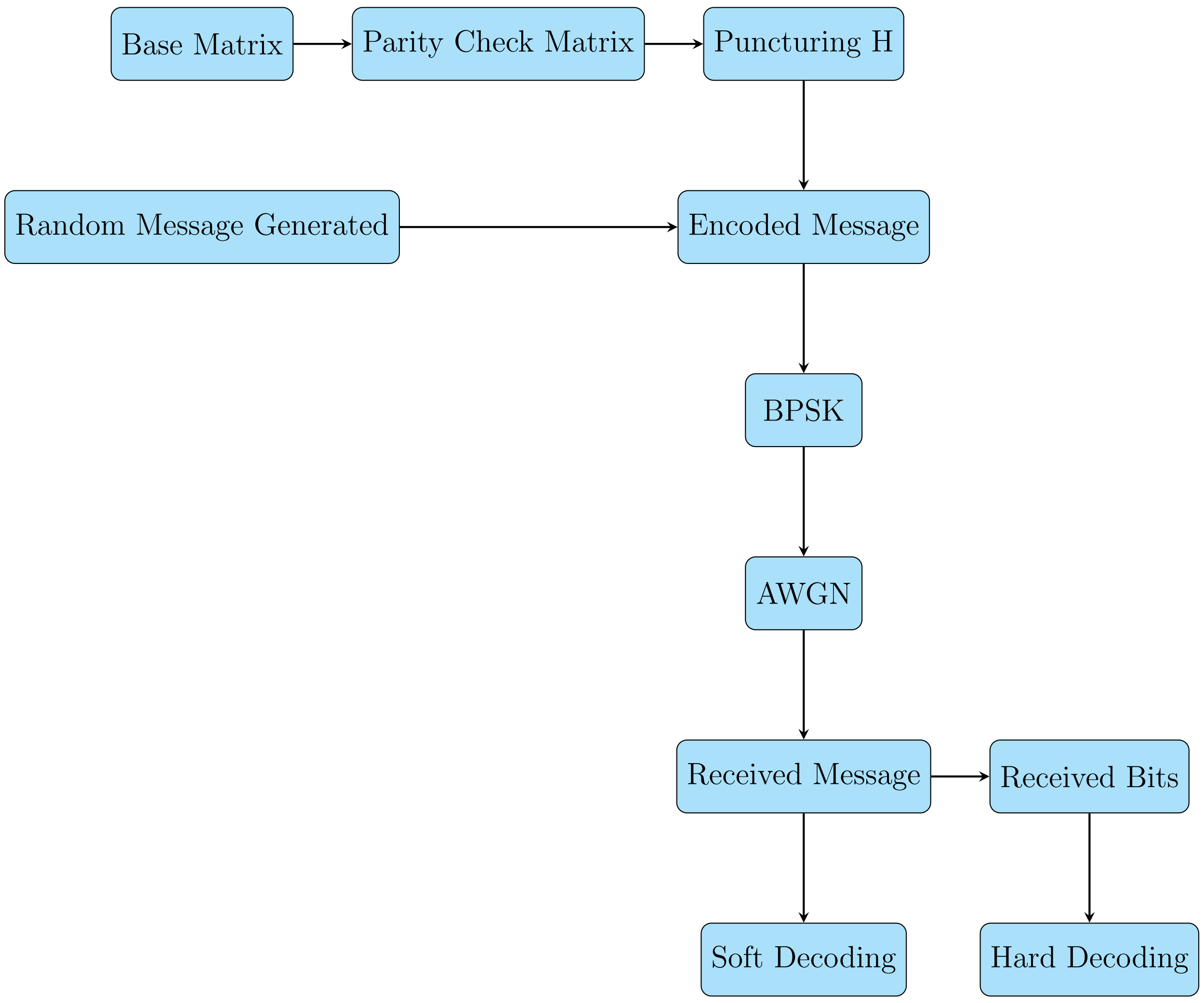
- The work that we are presenting is our own work.
- We have not copied the work (Matlab code, results, etc.) that someone else has done.
- Concepts, understanding, and insights we will be describing are our own.
- Wherever we have relied on an existing work that is not our own, We have provided a proper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

Rudra [illegible] [illegible] [illegible] [illegible]

[illegible] APURV [illegible] [illegible] [illegible]

Figure 1: Signatures of Group Members

LDPC in 5G NR – Program Flow



About LDPC and 5G NR

Low-Density Parity-Check (LDPC) codes are linear error-correcting codes introduced by Robert Gallager in the 1960s. Characterized by a sparse parity-check matrix, LDPC codes achieve near-Shannon limit performance using iterative decoding.

5G New Radio (5G NR) is the global standard for a flexible and scalable radio interface in 5G systems. To meet the demands of higher data rates, reliability, and massive connectivity, LDPC codes have been adopted by 3GPP for data channels in 5G NR.

Why LDPC in 5G?

- Excellent error correction with low complexity
- Scalable to different block lengths and code rates
- Efficient parallelizable decoding suitable for hardware

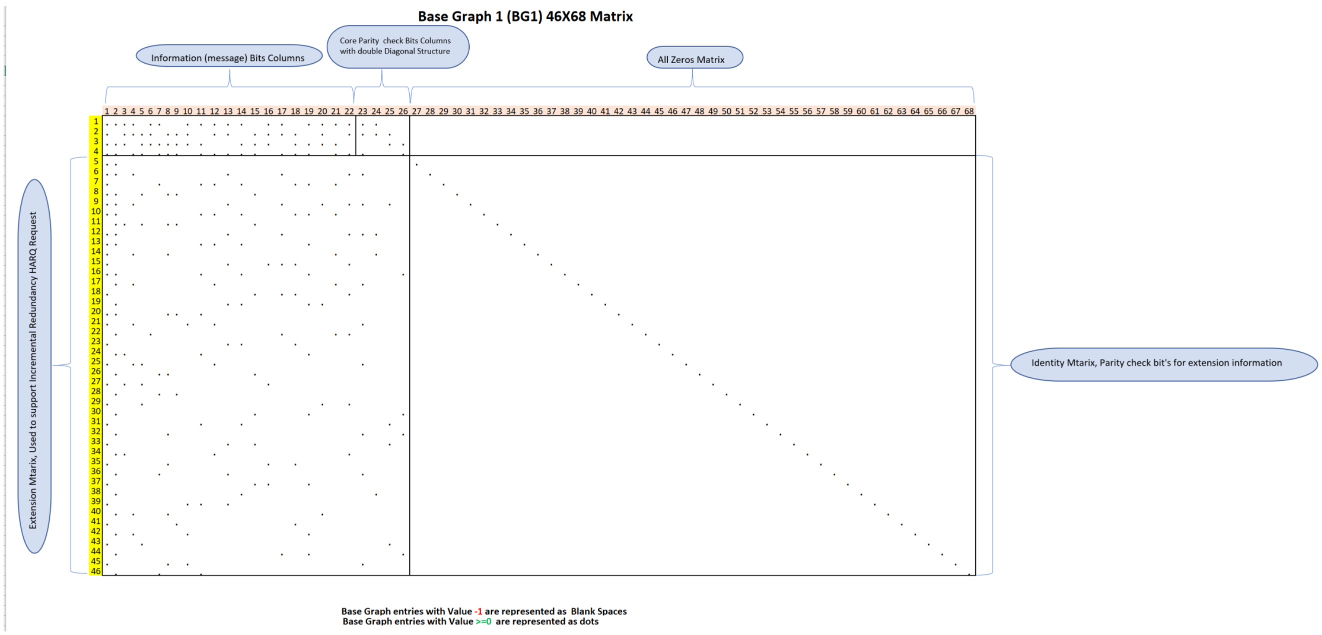
Encoding

Base Graphs (BG1 and BG2):

- **BG1** is used for larger transport block sizes (e.g., $K \geq 3840$) and higher code rates. It provides a larger number of variable and check nodes.
- **BG1** has dimensions of 46×68 and up to 22 information bits.
- **BG2** is used for smaller transport block sizes and offers better decoding performance for low code rates.
- **BG2** has dimensions of 42×52 and up to 10 information bits.
- To get x/y code rate: Suppose we need to puncture $a \cdot z$ bits. Using the formula

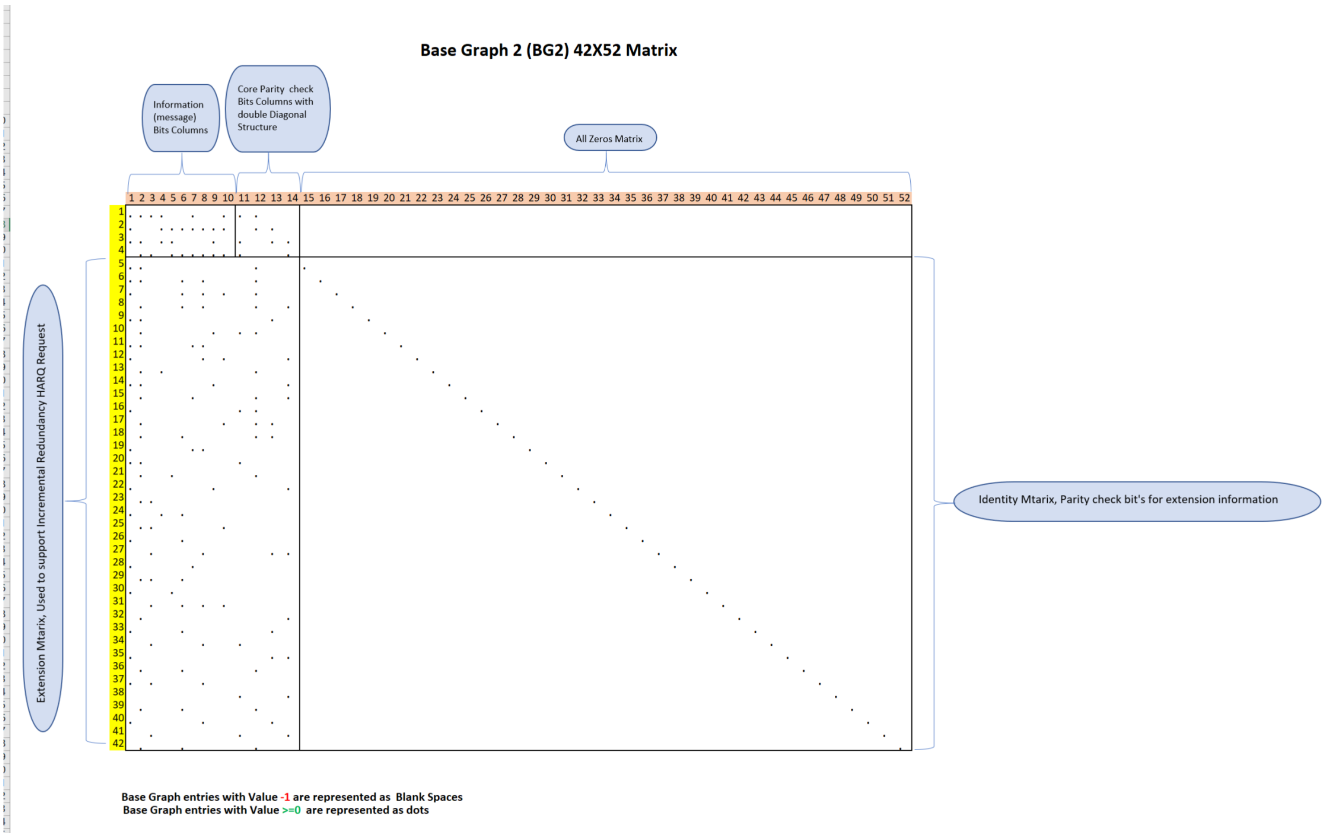
  $$\frac{\mathbf{(n-m)\cdot z}}{\mathbf{(n-2)\cdot z - a\cdot z}} = \frac{\mathbf{x}}{\mathbf{y}}$$

  We can find the lower limit of the code rate for base graphs. For BG2, the lower limit is $\mathbf{1/5}$. For BG1, the lower limit is $\mathbf{1/3}$.
- Each base graph is a compact template, which is later ”lifted” to match the required block length by replacing each entry with a circulant permutation matrix or a zero matrix.



Base Graph 1 (BG1) 46X68 Matrix
Information (message) Bits Columns
Core Parity check Bits Columns with double Diagonal Structure
All Zeros Matrix
Identity Mtarix, Parity check bit's for extension information
Extension Mtarix, Used to support Incremental Redundancy HARQ Request
Base Graph entries with Value -1 are represented as Blank Spaces
Base Graph entries with Value >=0 are represented as dots

Figure 2: Structure of BG1



Base Graph 2 (BG2) 42X52 Matrix
Information (message) Bits Columns
Core Parity check Bits Columns with double Diagonal Structure
All Zeros Matrix
Identity Mtarix, Parity check bit's for extension information
Extension Mtarix, Used to support Incremental Redundancy HARQ Request
Base Graph entries with Value -1 are represented as Blank Spaces
Base Graph entries with Value >=0 are represented as dots

Figure 3: Structure of BG2

Base Graph to Parity-Check Matrix (H) Conversion

Each base graph (BG1 or BG2) in 5G NR LDPC represents a compact template for the parity-check matrix H. This matrix is structured in a quasi-cyclic (QC) manner, meaning it consists of repeating patterns formed by submatrices arranged in a structured grid.

- A base graph is a small matrix (e.g., 46x68 for BG1) containing integer values.
- Each non-negative integer in the base graph refers to a submatrix.
- The lifting factor Z determines the size of each circulant permutation matrix.
- The final parity-check matrix H is formed by replacing each entry:
  - $-1 \rightarrow$ All-zero submatrix of size $Z \times Z$
  - $a \geq 0 \rightarrow$ a submatrix of size $Z \times Z$ by circularly right-shifting an identity matrix by a positions.
- The result is a large sparse matrix used in encoding and decoding.

$$H_{BG}=\begin{bmatrix} 0 & -1 & -1 & 1 \\ -1 & 2 & 3 & -1 \end{bmatrix} \quad H=\left[\begin{array}{cccc|cccc|cccc|cccc} 1&0&0&0 & 0&0&0&0 & 0&0&0&0 & 0&1&0&0 \\ 0&1&0&0 & 0&0&0&0 & 0&0&0&0 & 0&0&1&0 \\ 0&0&1&0 & 0&0&0&0 & 0&0&0&0 & 0&0&0&1 \\ 0&0&0&1 & 0&0&0&0 & 0&0&0&0 & 1&0&0&0 \\ \hline 0&0&0&0 & 0&0&1&0 & 0&0&0&1 & 0&0&0&0 \\ 0&0&0&0 & 0&0&0&1 & 1&0&0&0 & 0&0&0&0 \\ 0&0&0&0 & 1&0&0&0 & 0&1&0&0 & 0&0&0&0 \\ 0&0&0&0 & 0&1&0&0 & 0&0&1&0 & 0&0&0&0 \end{array}\right]$$

Figure 4: Effect of puncturing on the parity-check matrix

5G NR adjusts code rates using puncturing, repetition, and shortening—all while keeping the same encoder and decoder. To see how puncturing works, let's look at its effect on the parity-check matrix (H).

Puncturing in H Matrix:

- Puncturing is a "rate adaptation mechanism" employed in error-correcting codes, wherein a subset of the encoded bits is deliberately excluded from transmission.
- This approach enables dynamic rate adjustment without necessitating modifications to the encoder or decoder implementations.
- In LDPC, this is realized by removing or skipping certain columns in the parity-check matrix H during transmission.
- 5G NR uses rate matching (puncturing, repetition, shortening) to adjust code rates without changing the encoder or decoder.

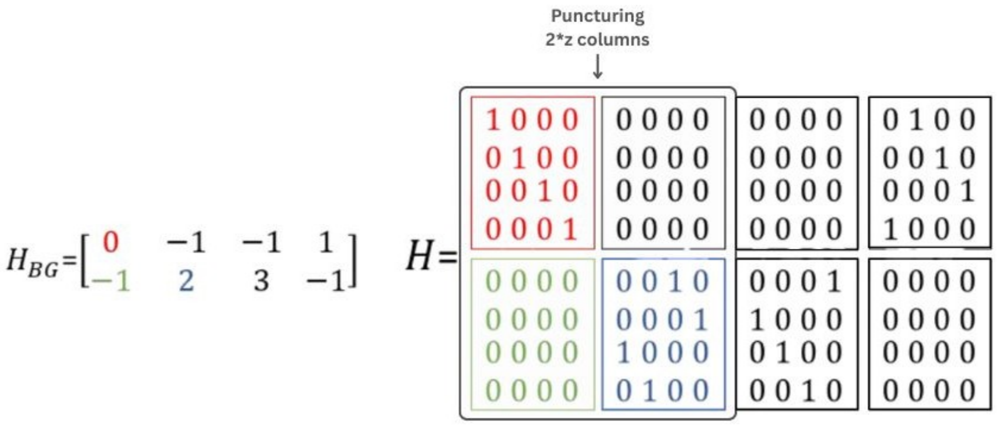


Figure 5: Effect of puncturing on the parity-check matrix

Encoding Process:

- The encoder takes message bits and produces parity bits using the parity-check matrix H.
- The full codeword is the concatenation of message and parity bits.
- $\mathbf{c} = \mathbf{u} \cdot G$, where G is the generator matrix.
- The parity bits of the encoded codeword can be simply found by solving linear equations. As mentioned earlier, the double diagonal structure of the base matrix makes this task easier.
- We use the fact that:

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

  where

  $$C = [m_1\ m_2\ \dots\ m_k\ p_1\ p_2\ \dots\ p_{n-k}]$$

  is the encoded codeword, and the size of the matrix H is $k \times n$.

BPSK Modulation and AWGN Channel in LDPC Systems

BPSK Modulation

Binary Phase Shift Keying (BPSK) is a digital modulation scheme that maps binary bits to two distinct phases of a carrier signal. In LDPC systems, this modulation is performed after encoding.

- The LDPC encoder outputs a binary codeword $\mathbf{c} = \{c_1, c_2, \dots, c_n\}$.

- Each bit $c_i \in \{0, 1\}$ is mapped to a BPSK symbol $x_i \in \{+1, -1\}$ using the rule:

$$x_i = 1 - 2c_i$$

- This results in $c_i = 0 \Rightarrow x_i = +1$ and $c_i = 1 \Rightarrow x_i = -1$.

AWGN Channel Model

The Additive White Gaussian Noise (AWGN) channel model introduces random noise into the signal:

$$r_i = x_i + n_i$$

Where:

- x_i is the transmitted BPSK symbol,
- $n_i \sim \mathcal{N}(0, \sigma^2)$ is the Gaussian noise,
- r_i is the received symbol.

The noise is:

- **Additive:** Noise is simply added to the signal.
- **White:** Uniform power across frequencies.
- **Gaussian:** Follows a normal distribution.

Soft Decision Decoding

Soft decision decoding is a powerful technique that leverages the reliability of each received bit rather than making a binary decision (0 or 1) immediately. This approach is essential in modern communication systems like 5G NR, where high performance under low Signal-to-Noise Ratio (SNR) conditions is required. It forms the basis for iterative decoding algorithms such as Belief Propagation and its simplification — the Min-Sum algorithm.

Log-Likelihood Ratio (LLR) Input

The Log-Likelihood Ratio (LLR) is a quantitative measure of how likely it is that a received bit corresponds to a 0 or a 1. It is defined as:

$$LLR = \log\left(\frac{P(x = 0 \mid r)}{P(x = 1 \mid r)}\right)$$

Where:

- x is the transmitted bit.
- r is the received signal after modulation and channel effects.
- $P(x = 0 \mid r)$ and $P(x = 1 \mid r)$ are the conditional probabilities.

For BPSK over an AWGN channel, where bit 0 is mapped to +1 and bit 1 to -1, and noise is Gaussian with variance σ^2, the LLR simplifies to:

$$LLR = \frac{2r}{\sigma^2}$$

This value indicates:

- A large positive LLR $\rightarrow$ high confidence the bit is 0
- A large negative LLR $\rightarrow$ high confidence the bit is 1
- LLR near zero $\rightarrow$ uncertainty

LLRs serve as the input to the variable nodes in the Tanner graph for iterative decoding algorithms.

Min-Sum Algorithm

The Min-Sum algorithm is a simplified version of the Belief Propagation algorithm used in soft-decision decoding of LDPC codes. It reduces the computational burden by replacing hyperbolic tangent operations with simple minimum and sign operations.

Step 1: Initialization
Each variable node is initialized with the corresponding LLR value from the channel which it sends to the connected check nodes:

$$m_{v\to c}^{(0)} = LLR_v$$

Step 2: Check Node Update
Each check node computes messages to connected variable nodes based on the minimum of incoming messages:

$$m_{c\to v} = \left(\prod_{v' \in N(c) \setminus v} \text{sign}(m_{v'\to c}) \right) \cdot \min_{v' \in N(c) \setminus v} |m_{v'\to c}|$$

Step 3: Variable Node Update
Each variable node updates its message to a check node using the sum of LLR and all incoming check node messages (excluding the target check node):

$$m_{v\to c} = LLR_v + \sum_{c' \in N(v) \setminus c} m_{c'\to v}$$

Step 4: Hard Decision
After a fixed number of iterations or convergence, the final LLR is:

$$LLR_v^{final} = LLR_v + \sum_{c' \in N(v)} m_{c'\to v}$$

The decoded bit is:

$$\hat{x}_v = \begin{cases} 0 & \text{if } LLR_v^{final} \geq 0 \\ 1 & \text{otherwise} \end{cases}$$

Step 5: Hard Decision

Convert the final belief vector to hard bits:

$$[-1.0, -0.4, 1.1, -0.6, 0.4, 0.7, -0.7] \Rightarrow [0, 0, 1, 0, 1, 1, 0]$$

Step 6: Check for Valid Codeword

Verify if the hard-decoded codeword satisfies:

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

If yes, decoding is successful. If not, repeat from Step 3.

Hard Decision (Bit-Flipping Algorithm)

This algorithm corrects errors in received codewords by iteratively flipping bits that violate parity-check constraints.

Initialization

Let:

- r = Received noisy codeword (binary vector)
- H = Parity-check matrix
- MaxIter = Maximum allowed iterations

Step 1: Send Initial Messages

Each **VN** holds the received bit **r** and sends its current bit value (0 or 1) to all connected **CN** in the Tanner graph.

Step 2: CN $\rightarrow$ VN

For each **CN**:

- Compute the **modulo-2 sum** of all incoming messages.
- **If the sum = 0:**
  - The parity constraint is satisfied; send back the same bit values to connected variable nodes.
- **If the sum $\neq 0$:**
  - The parity constraint fails; send back flipped bit values to the connected **VNs**.

Step 3: VN $\rightarrow$ CN

Each variable node collects feedback from all connected check nodes and applies majority voting:

- If most check nodes suggest flipping, update the bit ($0 \leftrightarrow 1$).
- Else, keep the current bit.

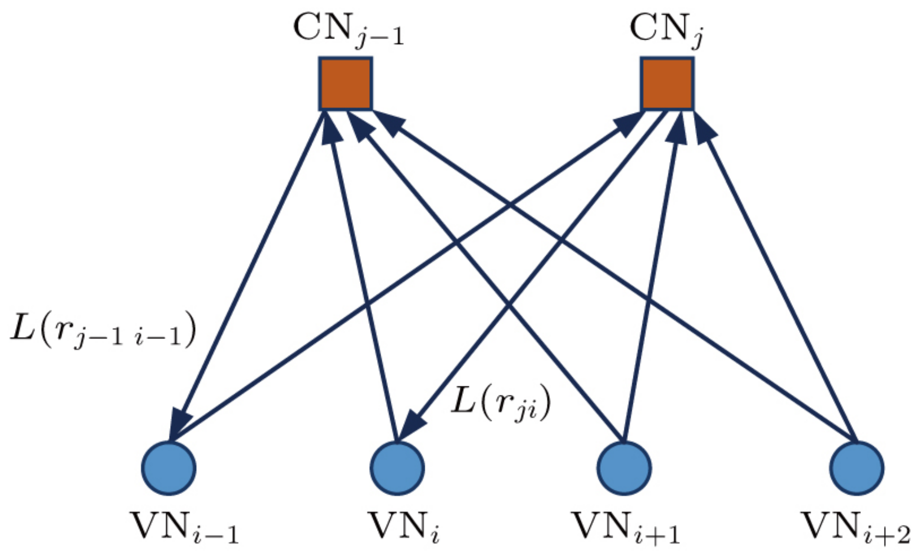


Figure 6: Tanner Graph

Step 4: Termination

- A valid codeword is found ($H \cdot r = 0$)
- Maximum number of iterations is reached
- The estimated codeword doesn't change between iterations

Example

Received codeword (r): [1, 0, 0, 0, 1, 0, 0, 0]
Assume an error at position C_2.

- Check nodes detect inconsistency in parity equations.
- After several iterations, C_2 flips to 1, yielding corrected codeword: [1, 1, 0, 0, 1, 0, 0, 0]

The Shannon Limit

The Shannon limit, also known as **Shannon's Theorem** or the **Noisy-Channel Coding Theorem**, defines the *maximum rate* at which information can be transmitted **reliably** over a noisy communication channel. It sets a **theoretical upper bound** on the channel capacity, which is the maximum achievable data rate without errors.

The channel capacity C is given by:

$$C = B \log_2 \left(1 + \frac{S}{N}\right)$$

where:

- B is the **bandwidth** of the channel,
- S is the **signal power**,
- N is the **noise power**.

According to the Shannon limit:

- If the information rate $R \leq C$, it is possible to transmit information *reliably* with an **arbitrarily small probability of error** using appropriate coding techniques.
- If $R > C$, then **errors are unavoidable**, regardless of the coding method used.

LDPC

1.Hard Decision Decoding

```
QDbaseGraph5GNR = 'NR_2_6_52';
codeRates = [1/4, 1/3, 1/2, 3/5];

[B, Hfull, z] = nrldpc_Hmatrix(QDbaseGraph5GNR);
[mb, nb] = size(B);
kb = nb - mb;

EbNodB = 0:0.5:10;
Nsim = 1000;
maxItr = 20;

figure;
hold on;

for i = 1:length(codeRates)
    codeRate = codeRates(i);
    kNumInfoBits = kb * z;
    k_pc = kb - 2;
    nbRM = ceil(k_pc / codeRate) + 2;
    nBlockLength = nbRM * z;

    H = Hfull(:, 1:nBlockLength);
    nChecksNotPunctured = mb*z - nb*z + nBlockLength;
    H = H(1:nChecksNotPunctured, :);

    [u, n] = size(H);
    k = n - u;

    vn2cn = cell(1, n);
    for l = 1:n
        vn2cn{l} = find(H(:, l))';
    end

    cn2vn = cell(1, u);
    for l = 1:u
        cn2vn{l} = find(H(l, :));
    end

    plotvec = zeros(1, length(EbNodB));

    for idx = 1:length(EbNodB)
        jEb = EbNodB(idx);
        EbNo = 10^(jEb/10);
        sigma = sqrt(1 / (2 * codeRate * EbNo));
        errors = 0;
        for NsimIdx = 1:Nsim
            b = randi([0 1], [kNumInfoBits 1]);
            c = nrldpc_encode(B, z, b');
            c = c(1:nBlockLength)';

            s = 1 - 2 * c;
            r = s + sigma * randn(nBlockLength, 1);
            r_hard = (r < 0);
            VN = r_hard;

            msg_CN_2_VN = cell(1, n);

            for itr = 1:maxItr
                if itr == 1
                    for i = 1:u
                        sumxor = mod(sum(VN(cn2vn{i})), 2);
                        for j = 1:length(cn2vn{i})
                            idx_vn = cn2vn{i}(j);
                            tempxor = mod(sumxor + VN(idx_vn), 2);
                            msg_CN_2_VN{idx_vn}(end + 1) = tempxor;
                        end
                    end
                else
                    msg_VN_2_CN = cell(1, u);
                    for i = 1:n
                        cnt = r_hard(i);
                        for j = 1:length(vn2cn{i})
                            cnt = cnt + msg_CN_2_VN{i}(j);
                        end
                        for j = 1:length(vn2cn{i})
                            tempcnt = cnt - msg_CN_2_VN{i}(j);
                            msg_VN_2_CN{vn2cn{i}(j)}(end+1) = double(tempcnt
> (length(vn2cn{i}) / 2));
                        end
                    end

                    tempVN = zeros(1, n);
                    for i = 1:n
                        cnt1 = r_hard(i);
                        for j = 1:length(vn2cn{i})
                            cnt1 = cnt1 + msg_CN_2_VN{i}(j);
                        end
                        tempVN(i) = double(cnt1 > ((length(vn2cn{i}) + 1)/
2));
                    end

                    if isequal(tempVN, VN)
                        break;
                    end
                    VN = tempVN;

                    msg_CN_2_VN = cell(1, n);
                    for i = 1:u
                        sumxor = mod(sum(msg_VN_2_CN{i}), 2);
                        for j = 1:length(cn2vn{i})
                            idx_vn = cn2vn{i}(j);
                            tempxor = mod(sumxor + msg_VN_2_CN{i}(j), 2);
                            msg_CN_2_VN{idx_vn}(end + 1) = tempxor;
                        end
                    end
                end
            end

            decoded = VN(1:kNumInfoBits);
            errors = errors + sum(decoded ~= b');
        end


        plotvec(idx) = errors / (Nsim * kNumInfoBits);
        fprintf('Rate %.2f | Eb/No = %.1f dB | BER = %.5f\n', codeRate, jEb,
plotvec(idx));
    end

    semilogy(EbNodB, plotvec, 'LineWidth', 2);
end

legend('Rate 1/4', 'Rate 1/3', 'Rate 1/2', 'Rate 3/5', 'Location',
'SouthWest');
xlabel('Eb/No (dB)');
ylabel('Bit Error Rate (BER)');
title('5G NR LDPC Hard-Decision Decoding using Tanner Graph (Nsim=1000)');
grid on;

function [B,H,z] = nrldpc_Hmatrix(BG)
    load(sprintf('%s.txt', BG), BG);
    B = eval(BG);
    [mb, nb] = size(B);
    z = 52;
    H = zeros(mb*z, nb*z);
    Iz = eye(z); I0 = zeros(z);
    for kk = 1:mb
        tmpvecR = (kk-1)*z + (1:z);
        for kk1 = 1:nb
            tmpvecC = (kk1-1)*z + (1:z);
            if B(kk, kk1) == -1
                H(tmpvecR, tmpvecC) = I0;
            else
                H(tmpvecR, tmpvecC) = circshift(Iz, -B(kk, kk1));
            end
        end
    end
end

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

    for i = 1:4
        for j = 1:n-m
            temp = mod(temp + mul_sh(msg((j-1)*z+1:j*z), B(i,j)), 2);
        end
    end

    p1_sh = B(2,n-m+1);
    if p1_sh == -1
        p1_sh = B(3,n-m+1);
    end
    cword((n-m)*z+1:(n-m+1)*z) = mul_sh(temp, z - p1_sh);

    for i = 1:3
        temp = zeros(1,z);
        for j = 1:n-m+i
            temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z), B(i,j)), 2);
        end
        cword((n-m+i)*z+1:(n-m+i+1)*z) = temp;
    end

    for i = 5:m
        temp = zeros(1,z);
        for j = 1:n-m+4
            temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z), B(i,j)), 2);
        end
        cword((n-m+i-1)*z+1:(n-m+i)*z) = temp;
    end
end

function y = mul_sh(x,k)
    if k == -1
        y = zeros(1,length(x));
    else
        y = [x(k+1:end), x(1:k)];
    end
end
```

2.Soft Decision Decoding

```matlab
baseGraph5GNR = 'NR_2_6_52';
codeRates = [1/2,1/3,1/4,3/5];

[B, Hfull, z] = nrldpc_Hmatrix(baseGraph5GNR);
[mb, nb] = size(B);
kb = nb - mb;

EbNodB = 0:0.5:10;
Nsim = 1000;
maxItr = 20;

figure;
hold on;

for i = 1:length(codeRates)
    codeRate = codeRates(i);
    kNumInfoBits = kb * z;
    k_pc = kb - 2;
    nbRM = ceil(k_pc / codeRate) + 2;
    nBlockLength = nbRM * z;

    H = Hfull(:, 1:nBlockLength);
    nChecksNotPunctured = mb*z - nb*z + nBlockLength;
    H = H(1:nChecksNotPunctured, :);

    [u, n] = size(H);
    k = n - u;

    berVec = zeros(1, length(EbNodB));

    for idx = 1:length(EbNodB)
        jEb = EbNodB(idx);
        EbNo = 10^(jEb/10);
        sigma = sqrt(1 / (2 * codeRate * EbNo));
        succ = 0;

        for NsimIdx = 1:Nsim
            b = randi([0 1], [kNumInfoBits, 1]);
            c = nrldpc_encode(B, z, b');
            c = c(1:nBlockLength)';

            s = 1 - 2 * c;
            r = s + sigma * randn(nBlockLength, 1);

            decodedBits = SDD(H, r, maxItr);
            decoded = (decodedBits < 0);

            decode = decoded(1:kNumInfoBits);
            if sum(xor(decode,b)) == 0
                succ = succ + 1;
            end

        end

        berVec(idx) = (Nsim - succ)/Nsim;
        fprintf('Rate is: %.3f || Eb/No = %0.2f dB || BER = %.6f\n',
codeRate, EbNodB(idx), berVec(idx));
    end

    semilogy(EbNodB, berVec, 'LineWidth', 2);
end

legend('Rate 1/2','Rate 1/3','Rate 1/4','Rate 3/5', 'Location', 'SouthWest');
xlabel('Eb/No (dB)');
ylabel('Decoding error Probability');
title('5G NR LDPC Soft-Decision Decoding (Nsim=1000)');
grid on;
hold off;

function decode = SDD(H, received,maxItr)
    [row,col] = size(H);
    L = zeros(row,col);

    for i=1:row
        for j=1:col
            if H(i,j) ~= 0
                L(i,j) = received(j);
            end
        end
    end

    for itr=1:maxItr
        for i=1:row
            min1 = inf;
            min2 = inf;
            countEvenOdd = 0;

            for j=1:col
                if L(i,j) < 0
                countEvenOdd = countEvenOdd + 1;
                end
            end

               tol1=1e-10;
               tol2 = 1e-10;

            for j=1:col
                if L(i,j) ~= 0
                    val = abs(L(i,j));
                    if val < min1 - tol1
                        min2 = min1;
                        min1 = val;
                    elseif(val > min1+tol1) && (val < min2-tol1)
                        min2 = val;
                    end
                end
            end

            p = mod(countEvenOdd,2);

            for j = 1:col
              if L(i,j) ~= 0
                value = abs(L(i,j));
                if abs(value - min1) < tol2
                    if p == 0
                        L(i,j) = sign(L(i,j)) * min2;
                    else
                        L(i,j) = -sign(L(i,j)) * min2;
                    end
                else
                    if p == 0
                        L(i,j) = sign(L(i,j)) * min1;
                    else
                        L(i,j) = -sign(L(i,j)) * min1;
                    end
                end
              end
            end
        end

        for j=1:col
            sum = 0;
            for i=1:row
                sum = sum + L(i,j);
            end

            sum = sum + received(j);

            for i=1:row
                if L(i,j) ~= 0
                    L(i,j) = sum - L(i,j);
                end
            end

            received(j) = sum;
        end
        end

        decode = received;

end

function [B, H, z] = nrldpc_Hmatrix(BG)
    load(sprintf('%s.txt', BG), BG);
    B = eval(BG);
    [mb, nb] = size(B);
    z = 52;
    H = zeros(mb*z, nb*z);
    Iz = eye(z); I0 = zeros(z);
    for kk = 1:mb
        tmpvecR = (kk-1)*z + (1:z);
        for kk1 = 1:nb
            tmpvecC = (kk1-1)*z + (1:z);
            if B(kk, kk1) == -1
                H(tmpvecR, tmpvecC) = I0;
            else
                H(tmpvecR, tmpvecC) = circshift(Iz, -B(kk, kk1));
            end
        end
    end
end

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

    for i = 1:4
        for j = 1:n-m
            temp = mod(temp + mul_sh(msg((j-1)*z+1:j*z), B(i,j)), 2);
        end
    end

    p1_sh = B(2, n-m+1);
    if p1_sh == -1
        p1_sh = B(3, n-m+1);
    end
    cword((n-m)*z+1:(n-m+1)*z) = mul_sh(temp, z - p1_sh);

    for i = 1:3
        temp = zeros(1, z);
        for j = 1:n-m+i
            temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z), B(i,j)), 2);
        end
        cword((n-m+i)*z+1:(n-m+i+1)*z) = temp;
        end

        for i = 5:m
            temp = zeros(1, z);
            for j = 1:n-m+4
                temp = mod(temp + mul_sh(cword((j-1)*z+1:j*z), B(i,j)), 2);
            end
            cword((n-m+i-1)*z+1:(n-m+i)*z) = temp;
        end
end

function y = mul_sh(x, k)
    if k == -1
        y = zeros(1, length(x));
    else
        y = [x(k+1:end), x(1:k)];
    end
end
```

3.Comparison with Shannon

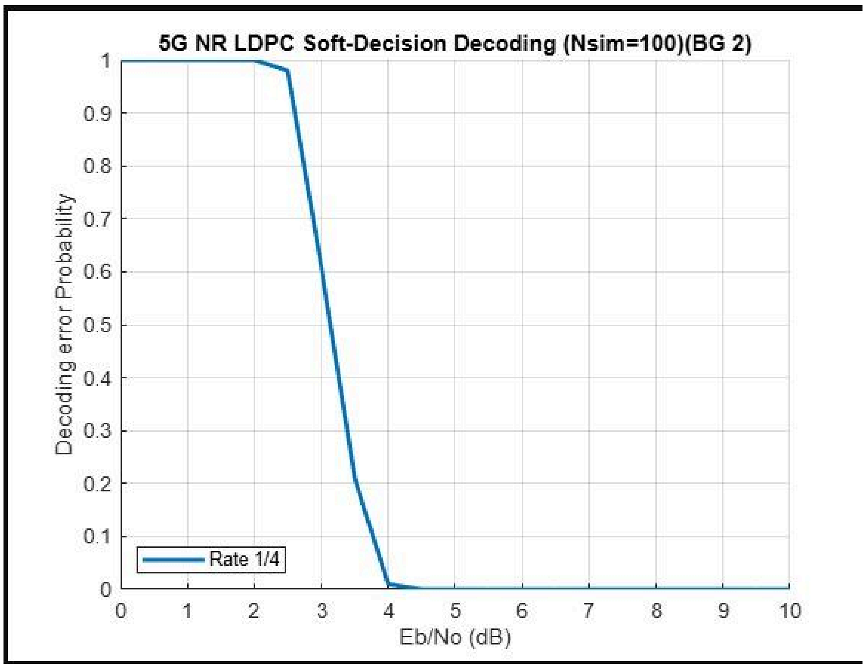
```
r = 1/4;
K = 128;
N = K / r;

EbNodB = 0:0.1:6;
EbNo_linear = 10.^(EbNodB / 10);

Pe_NA = zeros(size(EbNodB));

for i = 1:length(EbNodB)
    P = r * EbNo_linear(i);
    C = log2(1 + P);
    V = (log2(exp(1)))^2 * (P * (P + 2)) / (2 * (P + 1)^2);
    term = sqrt(N/V) * (C - r + log2(N) / (2 * N));
    Pe_NA(i) = qfunc(term);
end

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

figure;
semilogy(EbNodB_14, ber_14, 'ks-','LineWidth',2, 'MarkerFaceColor','w');hold
on;
semilogy(EbNodB, Pe_NA, 'b-', 'LineWidth', 2);
xline(shannon_limit_dB, 'r--', 'LineWidth', 2, ...
    'Label', sprintf('Shannon Limit (%.2f dB)', shannon_limit_dB), ...
    'LabelVerticalAlignment', 'bottom');
```

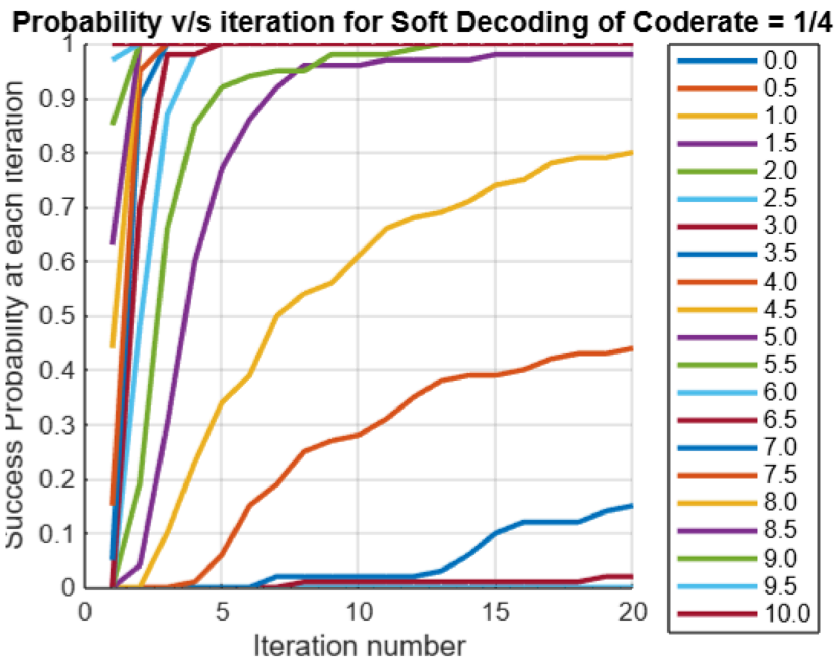
```
grid on;
xlabel('Eb/N0 (dB)');
ylabel('Block Error Probability');
title('Simulated BER vs Normal Approximation vs Shannon Limit (Rate = 1/4, N
= 512)');
legend('Simulated BER','Normal Approximation', 'Shannon Limit', 'Location',
'SouthWest');
```

Using Base Graph2

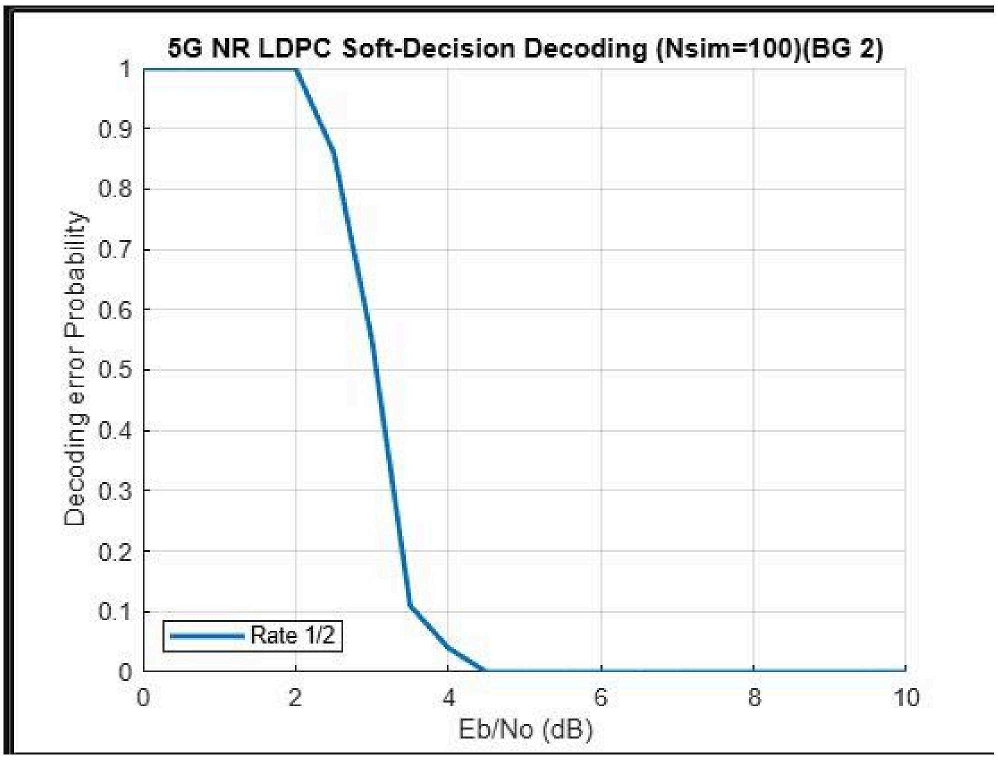
Soft Decoding :

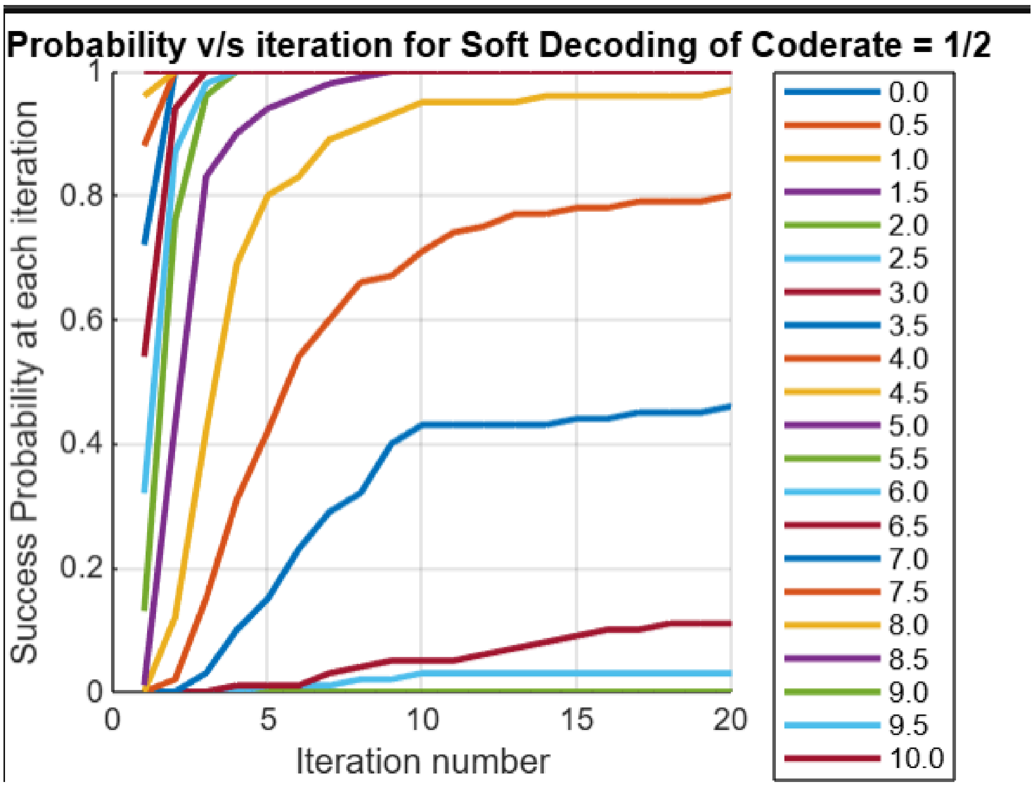
Rate = ¼;



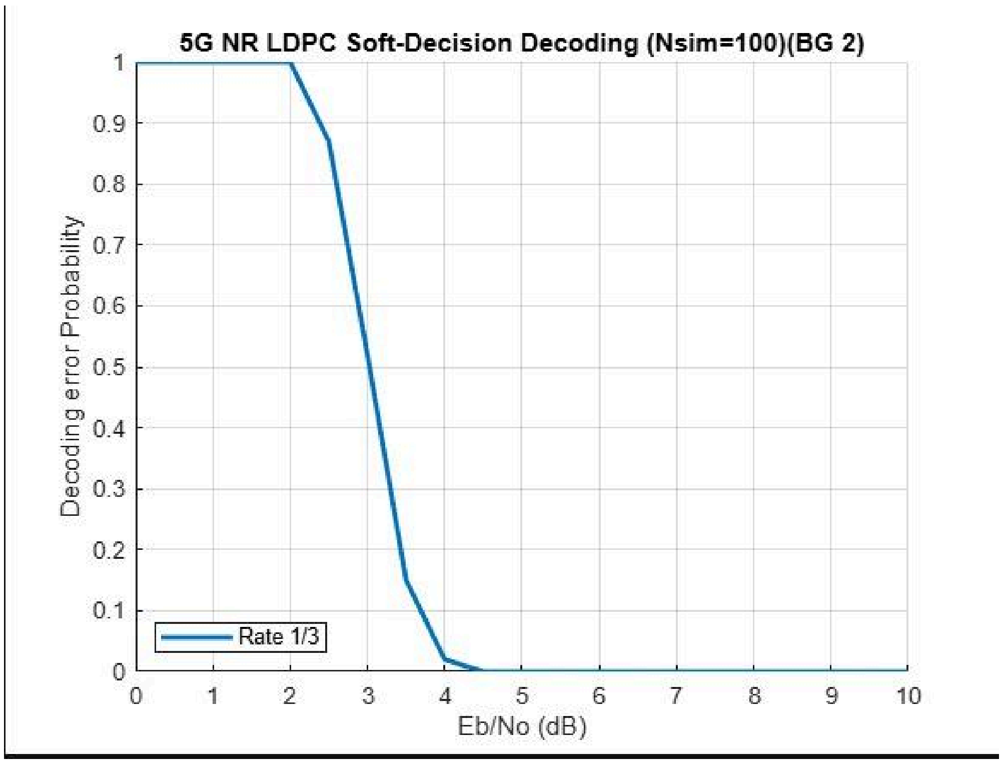


Rate = ½;

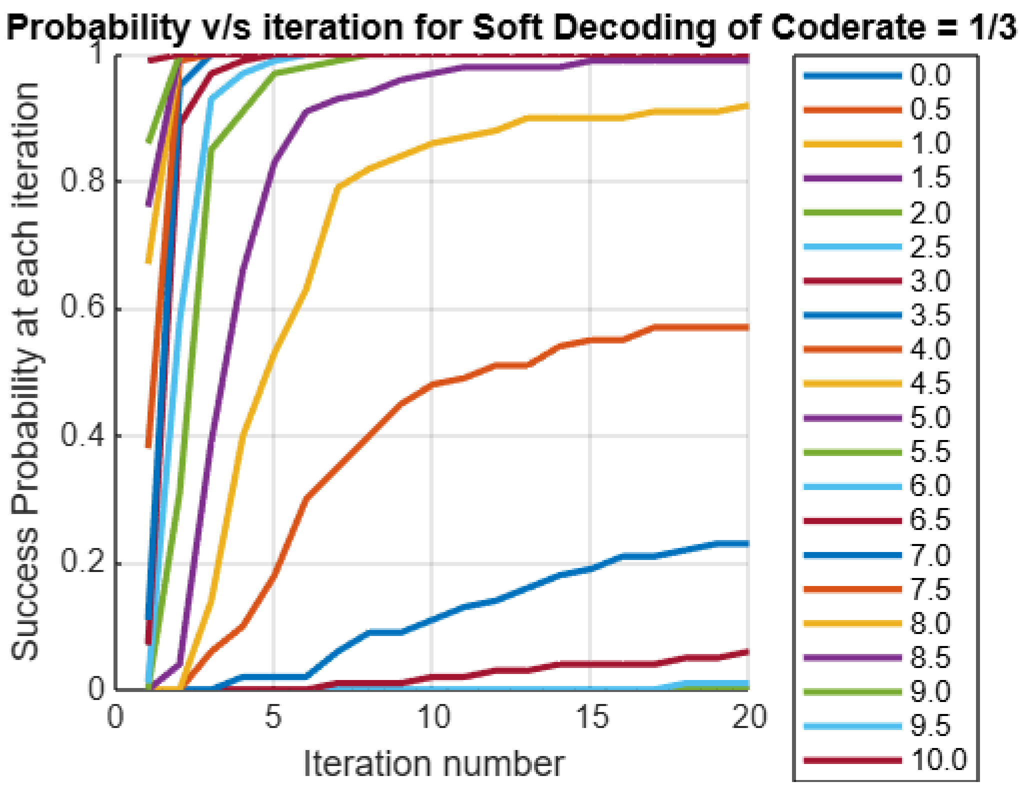

5G NR LDPC Soft-Decision Decoding (Nsim=100)(BG 2)
Decoding error Probability
Eb/No (dB)
Rate 1/2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
2
4
6
8
10


Probability v/s iteration for Soft Decoding of Coderate = 1/2
Success Probability at each iteration
Iteration number
0
0.2
0.4
0.6
0.8
1
0
5
10
15
20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0

Rate = ⅓;

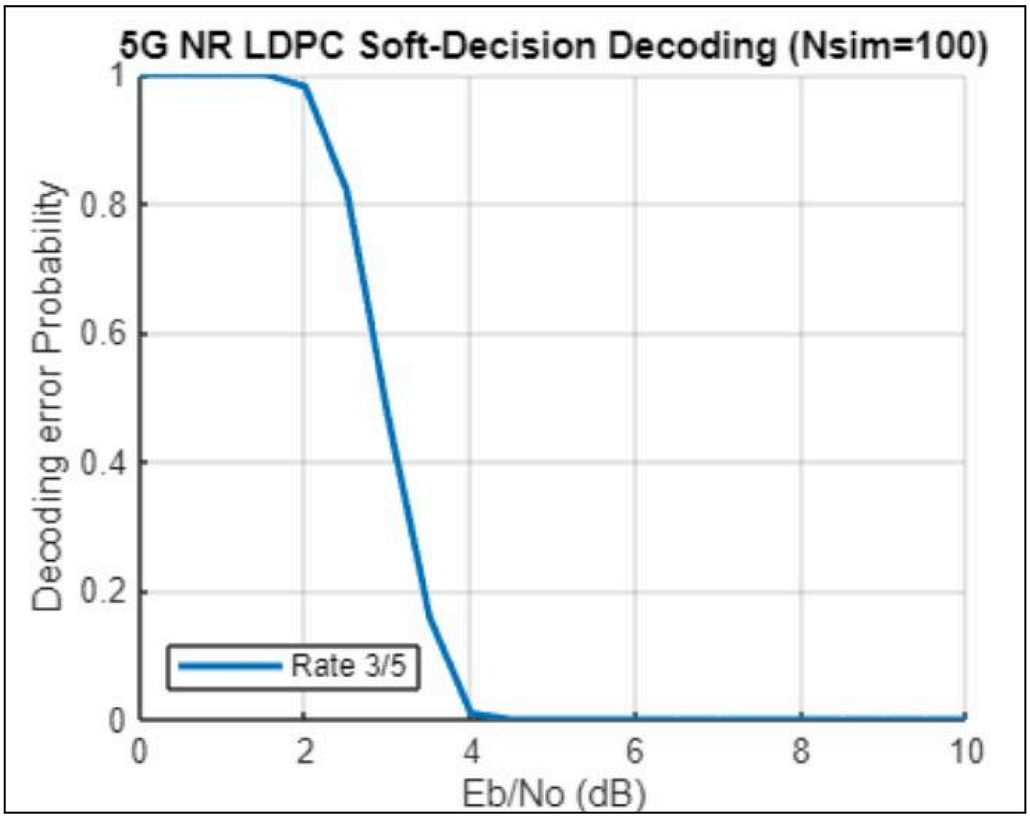


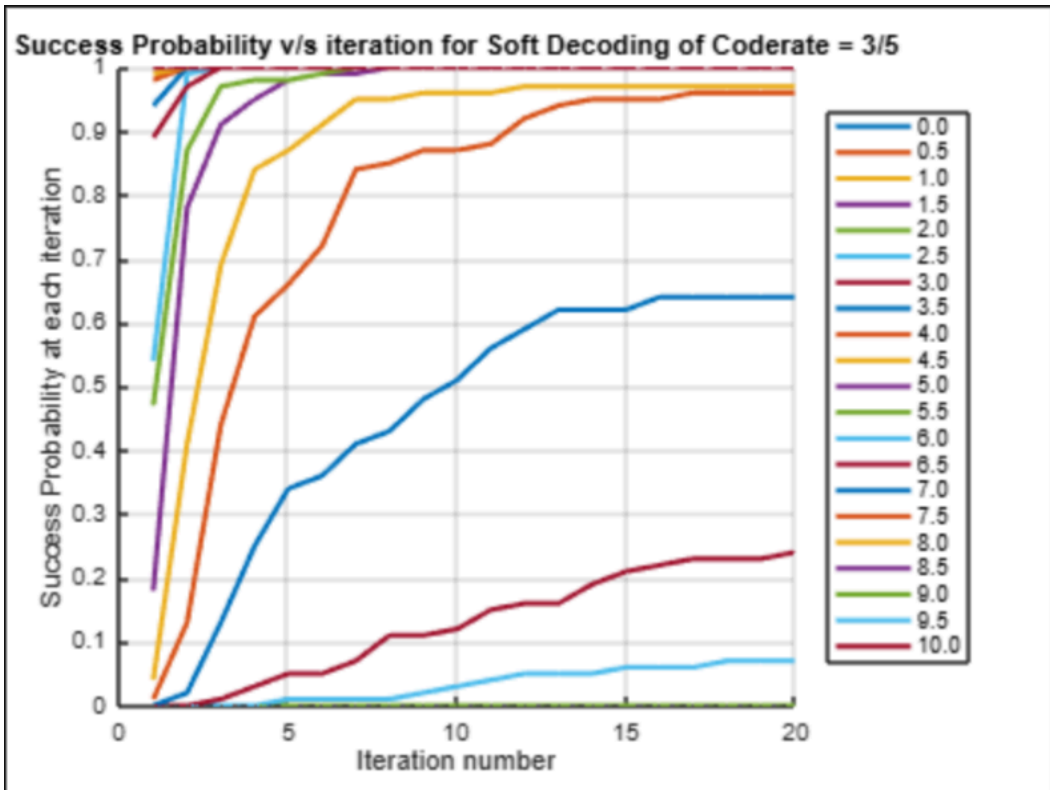
5G NR LDPC Soft-Decision Decoding (Nsim=100)(BG 2)
Decoding error Probability
Eb/No (dB)
Rate 1/3



Probability v/s iteration for Soft Decoding of Coderate = 1/3
Success Probability at each iteration
Iteration number
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0

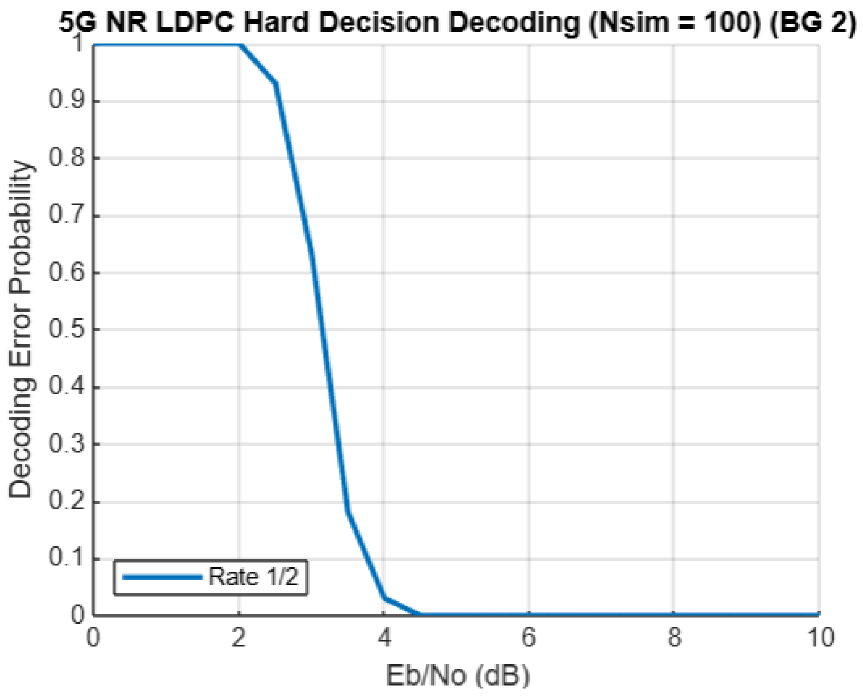
Rate = ⅗;

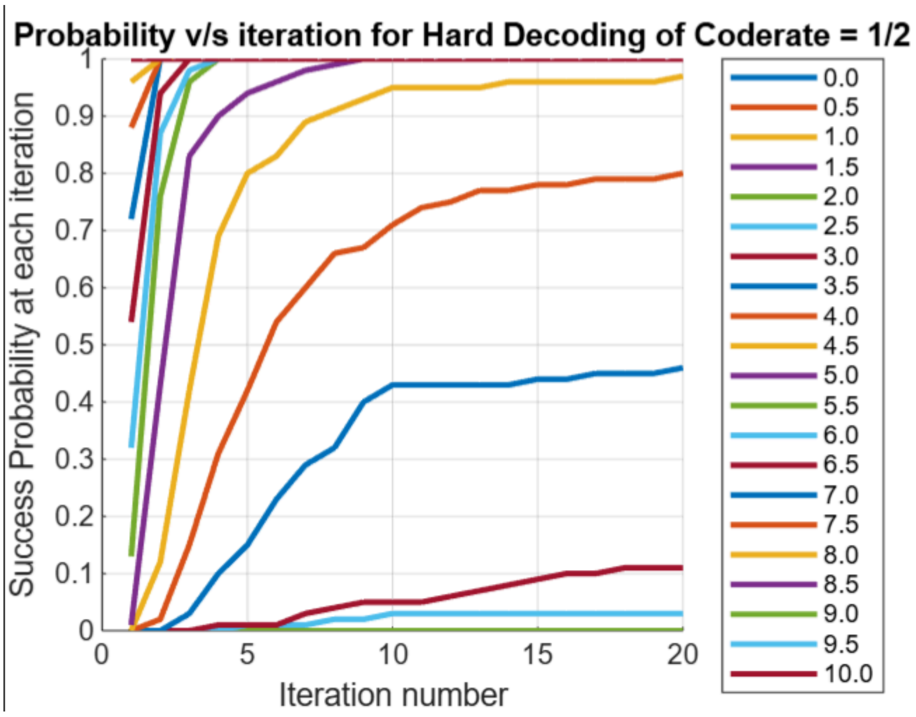

5G NR LDPC Soft-Decision Decoding (Nsim=100)
Decoding error Probability
Eb/No (dB)
Rate 3/5
0
0.2
0.4
0.6
0.8
1
0
2
4
6
8
10


Success Probability v/s iteration for Soft Decoding of Coderate = 3/5
Success Probability at each iteration
Iteration number
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0

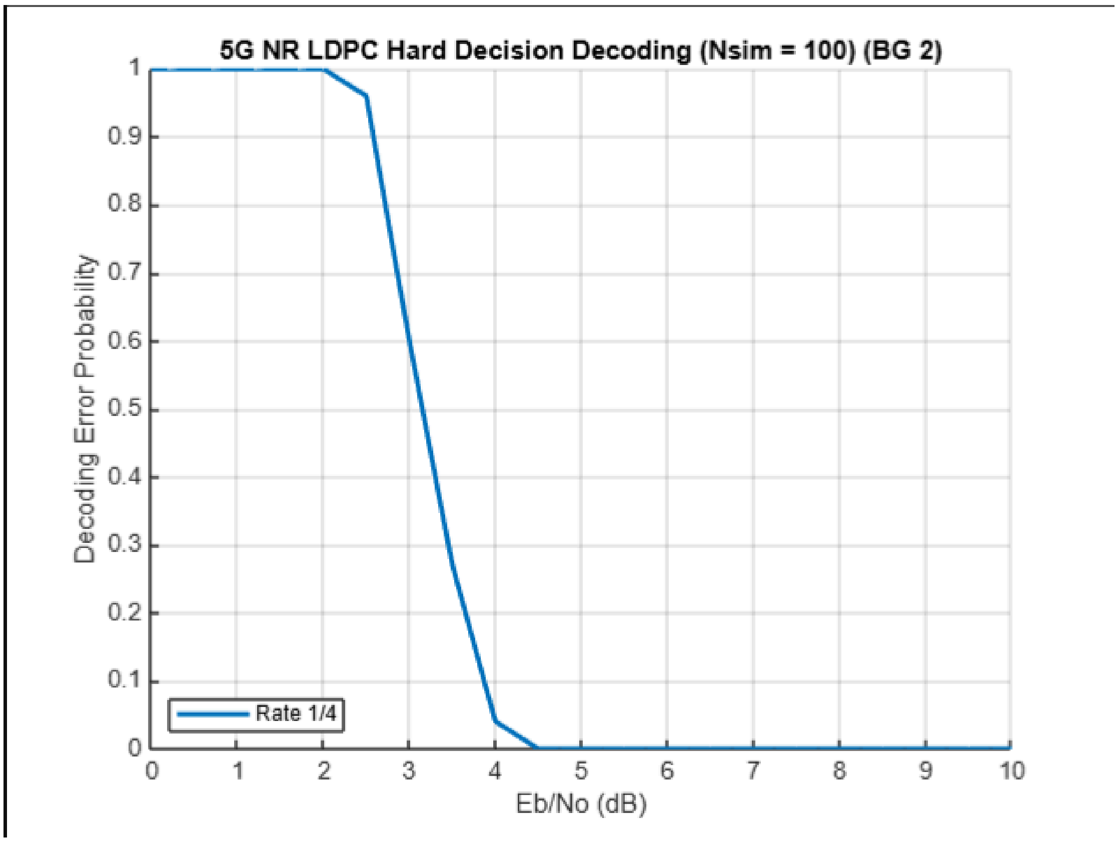
Hard Decoding :

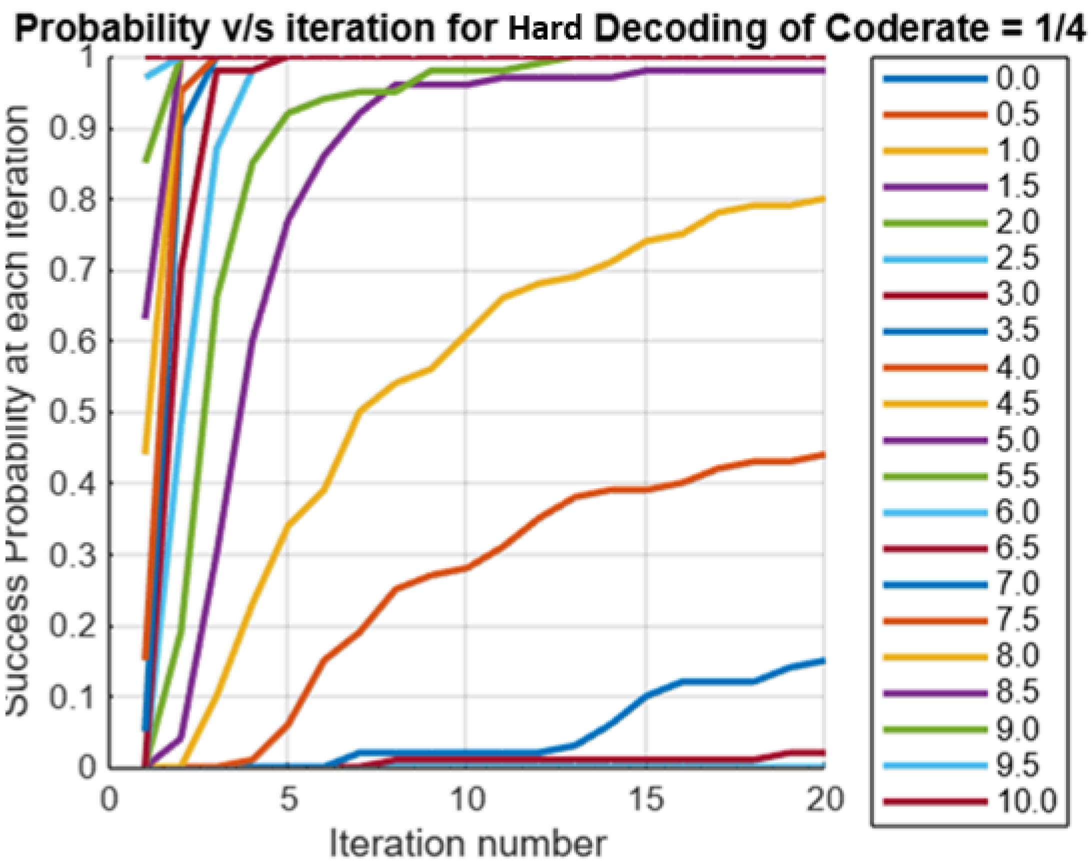
Rate = ½;



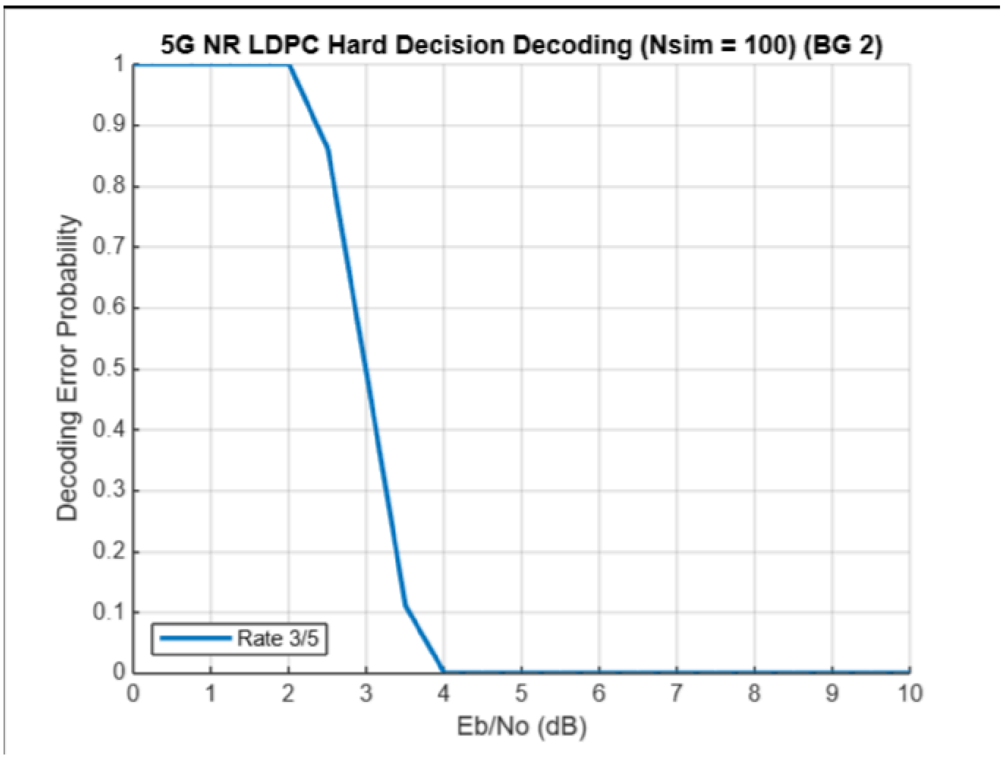


Rate = ¼;

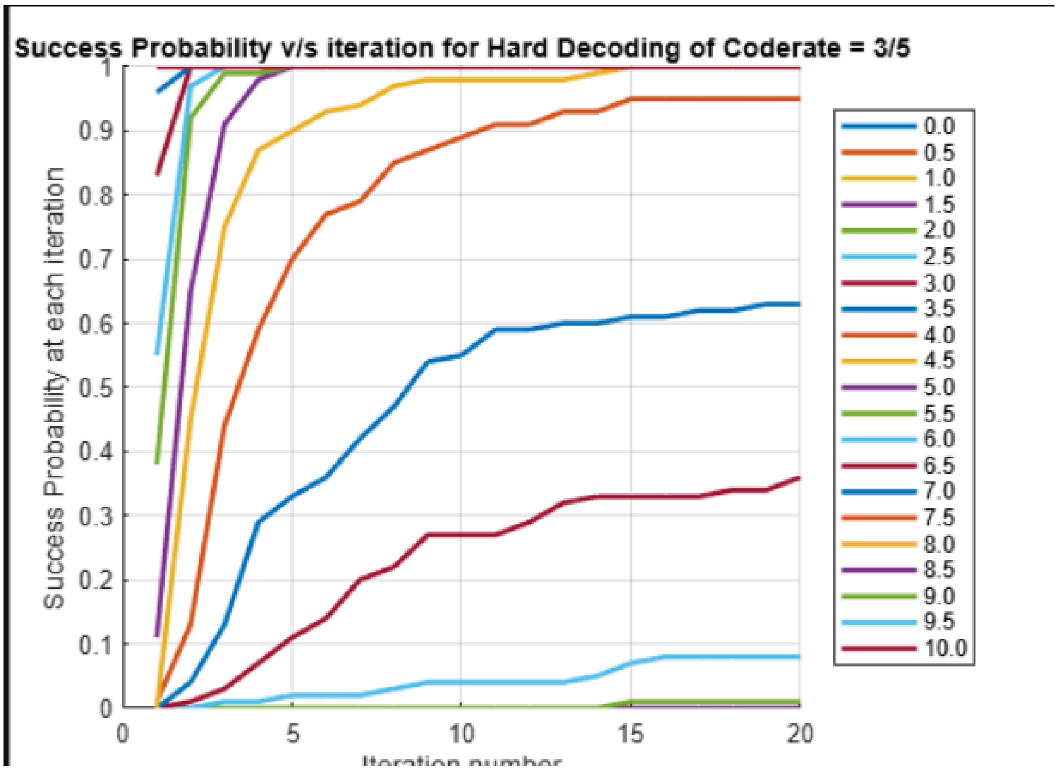

5G NR LDPC Hard Decision Decoding (Nsim = 100) (BG 2)
Decoding Error Probability
Eb/No (dB)
Rate 1/4
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
1
2
3
4
5
6
7
8
9
10


Probability v/s iteration for Hard Decoding of Coderate = 1/4
Success Probability at each iteration
Iteration number
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0

Rate = ⅗;

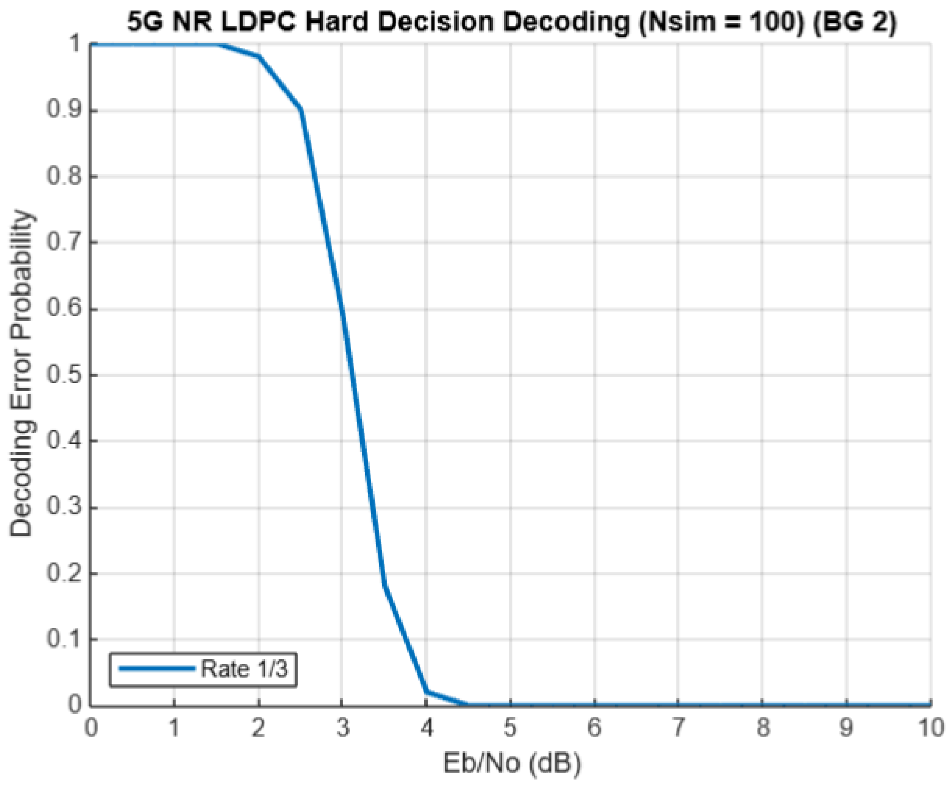


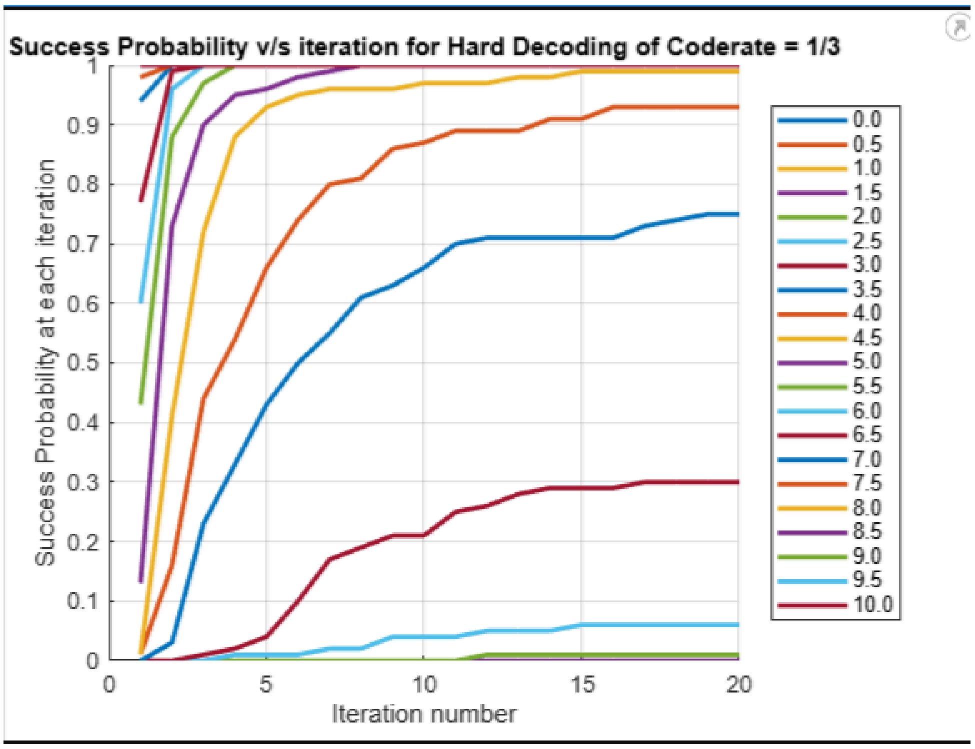
5G NR LDPC Hard Decision Decoding (Nsim = 100) (BG 2)
Decoding Error Probability
Eb/No (dB)
Rate 3/5

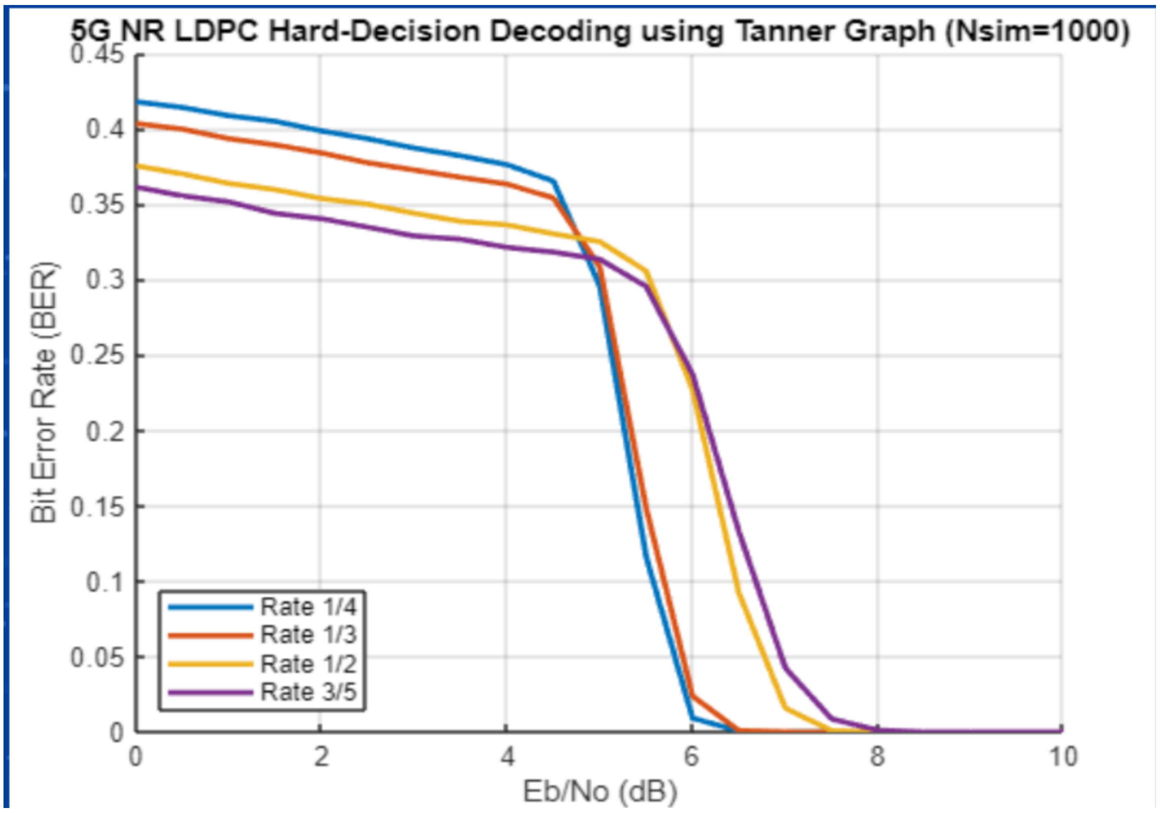


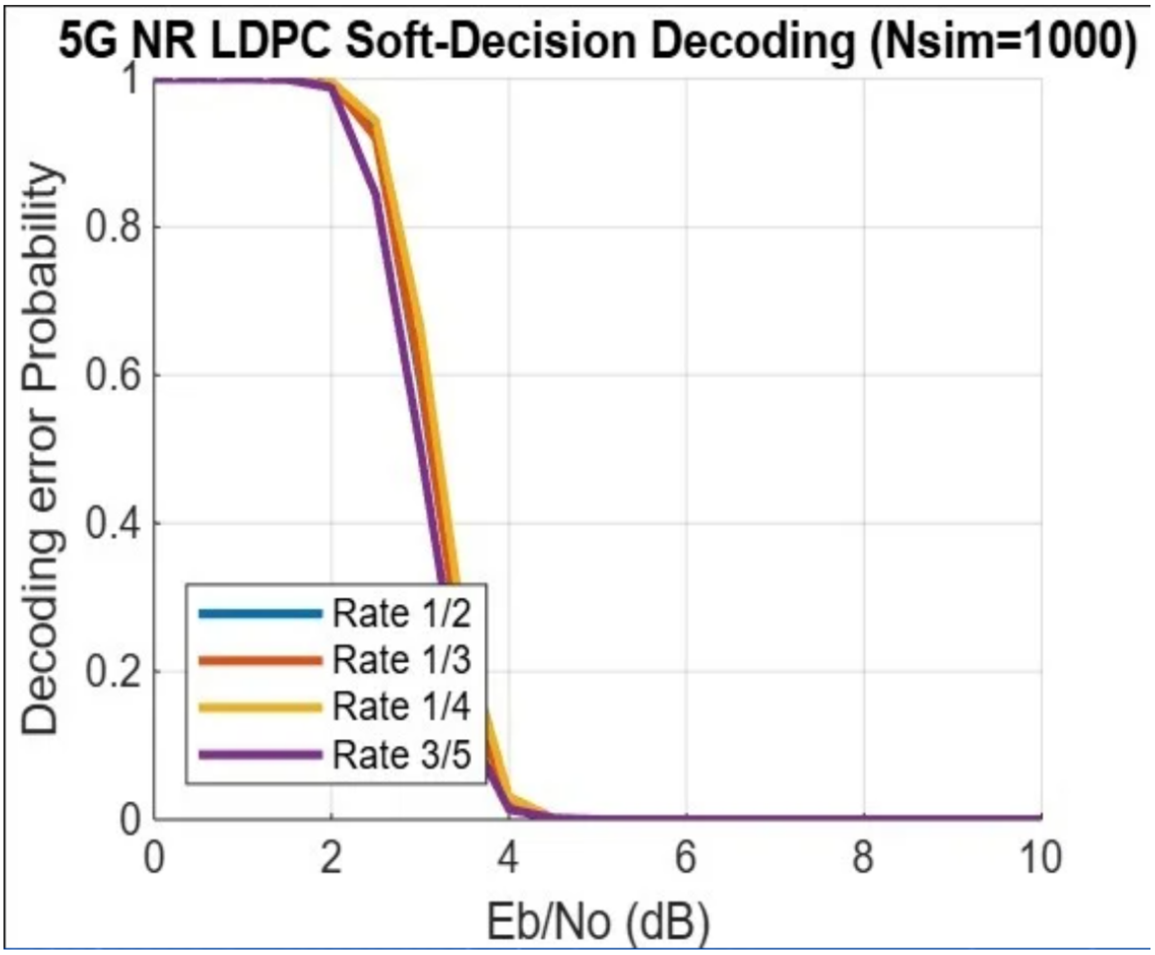
Success Probability v/s iteration for Hard Decoding of Coderate = 3/5
Success Probability at each iteration
Iteration number
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0

Rate = ⅓;


5G NR LDPC Hard Decision Decoding (Nsim = 100) (BG 2)
Decoding Error Probability
Eb/No (dB)
Rate 1/3
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
1
2
3
4
5
6
7
8
9
10

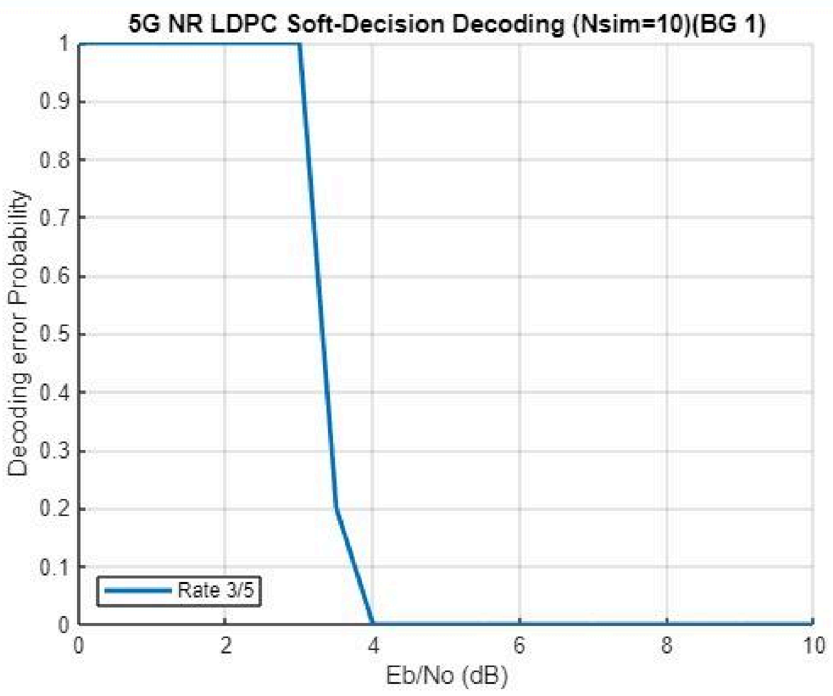

Success Probability v/s iteration for Hard Decoding of Coderate = 1/3
Success Probability at each iteration
Iteration number
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20


5G NR LDPC Hard-Decision Decoding using Tanner Graph (Nsim=1000)
Bit Error Rate (BER)
Eb/No (dB)
Rate 1/4
Rate 1/3
Rate 1/2
Rate 3/5

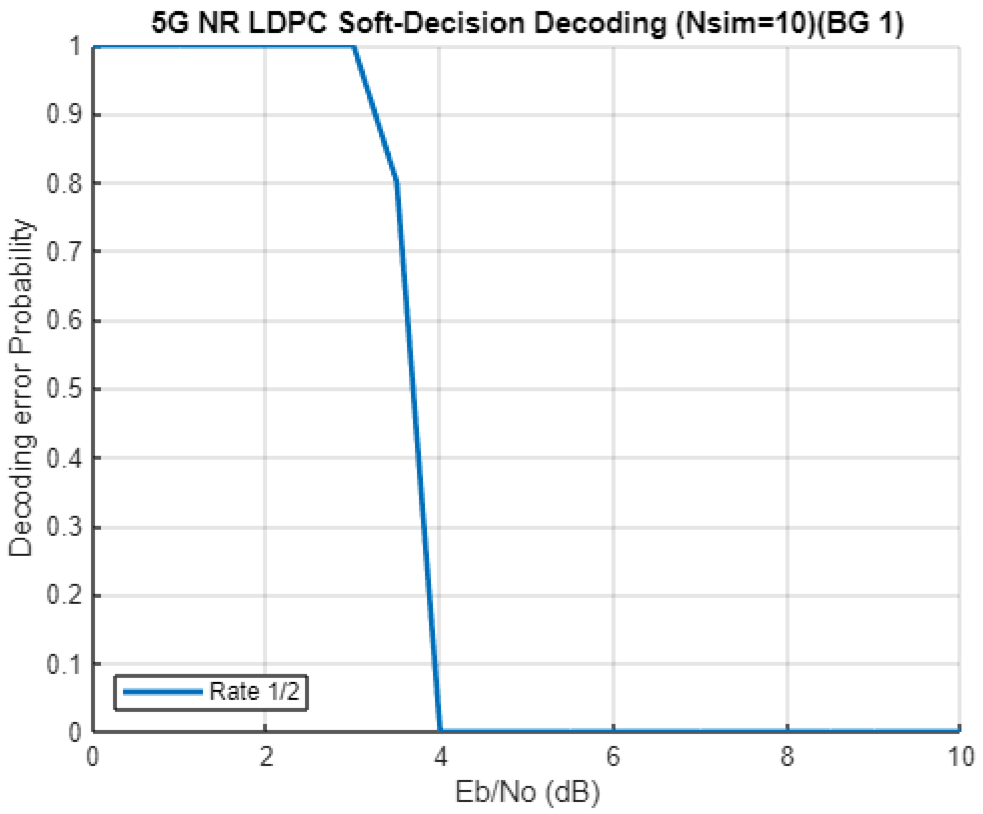

5G NR LDPC Soft-Decision Decoding (Nsim=1000)
Decoding error Probability
Eb/No (dB)
Rate 1/2
Rate 1/3
Rate 1/4
Rate 3/5

USING BG1 Soft Decoding

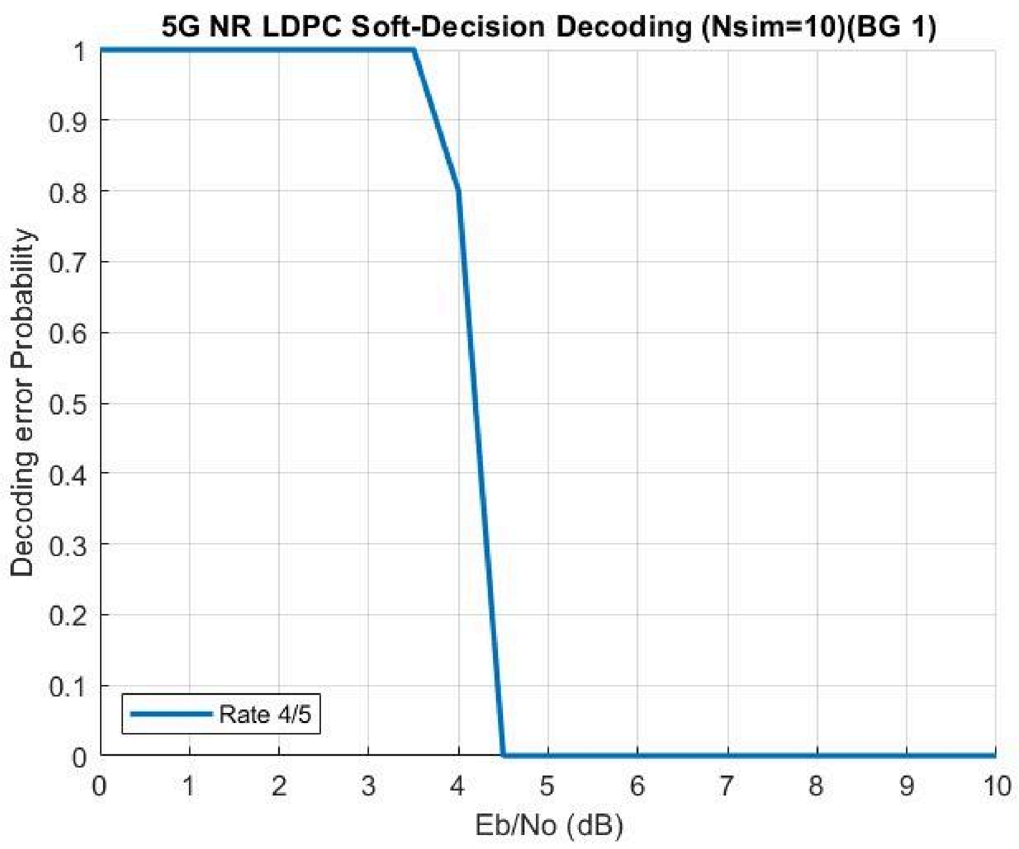
Rate : 3/5



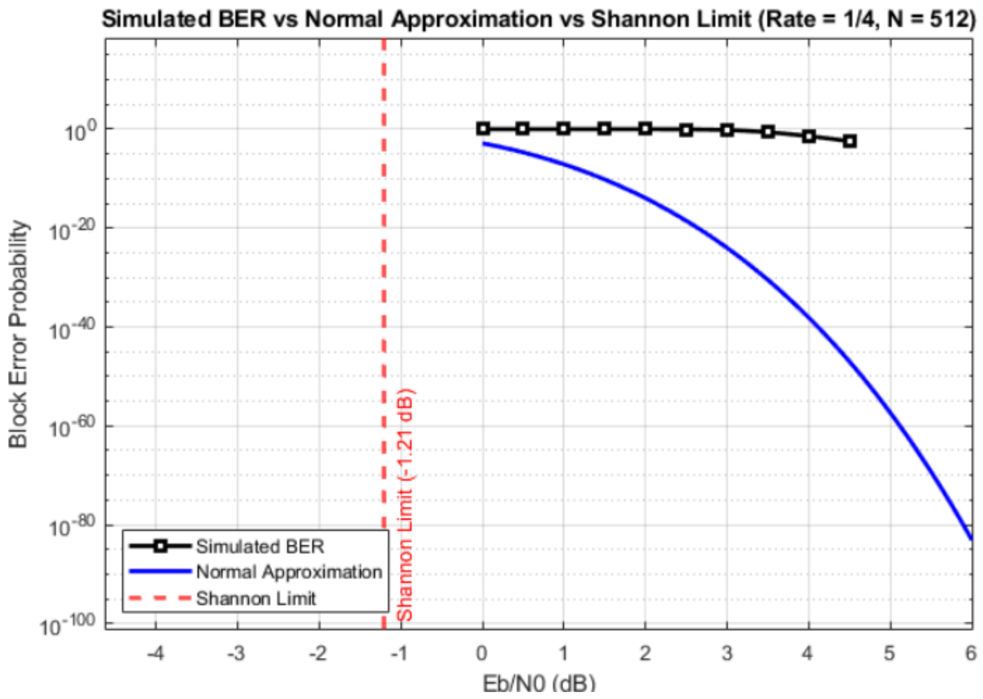
Rate : 1/2



Rate:4/5



Normal Approximation and Shannon Limit



Conclusion

In this project, we explored the role of LDPC codes in 5G NR systems, focusing on their encoding and decoding processes. We implemented the transformation from base graphs to parity-check matrices, applied puncturing techniques, and simulated BPSK modulation over an AWGN channel. Using both soft and hard decision decoding methods, we observed the error correction capabilities of LDPC codes and their proximity to the Shannon limit.

This study reinforces why LDPC is a preferred choice for modern communication systems like 5G NR due to its efficiency, flexibility, and performance under noisy conditions.

References

[1] Gallager, R. G. (1962). *Low-density parity-check codes.* IRE Transactions on Information Theory.

[2] Zhang, J., Chen, X., & Letaief, K. B. (2019). *An overview of channel coding for 5G NR cellular communications.* APSIPA Transactions on Signal and Information Processing, 8, e19. Cambridge University Press. https://doi.org/10.1017/ATSIP.2019.12

[3] Chelikani, V. K. (2023). *5G NR DL SCH LDPC Channel Coding and Base Graph Selection.* LinkedIn. Available at: `https://www.linkedin.com/pulse/5g-nr-dl-sch-ldpc-channel-coding-base-graph-selection-chelikani/` [Accessed 17 Apr. 2025].

[4] *An example of QC-LDPC code and its base matrix*, ResearchGate, 2025, `https://www.researchgate.net/figure/An-example-of-QC-LDPC-code-and-its-base-matrix_fig1_378024852`, Accessed: 2025-04-17.

[5] *GC-like LDPC Code Construction and its NN-aided Decoder Implementation*, Journal Name, Volume, Year, pages, `https://link_to_article`, Accessed: 2025-04-17.

[6] Muhammad Saleem, Ihsan Ullah, Gohar Khan, Muhammad Alhussein, and Nimra Rahim, *Improved LDPC Decoding for 5G Wireless Communication Systems Using Threshold-Based Bit Flipping Algorithm*, Wireless Communications and Mobile Computing, vol. 2022, Article ID 3076517, 10 pages, 2022. `https://onlinelibrary.wiley.com/doi/10.1155/2022/3076517`

[7] Shi Jin-Jing, Li Bo-Peng, Huang Duan, *Reconciliation for CV-QKD using globally-coupled LDPC codes*, Chinese Physics B, **29**, no. 4, 040301, 2020, `https://cpb.iphy.ac.cn/article/2020/2027/cpb_29_4_040301.html`.

[8] MacKay, D. J. C. (1999). *Good error-correcting codes based on very sparse matrices.* IEEE Transactions on Information Theory.