GROUP: 26 LDPC Coding

Prof. Yash Vasavada

Mentor: Dhriti Goenka

GROUP MEMBERS

ID	Name
202301282	HRIDAY ASNANI
202301401	DHRUVI JOBANPUTRA
202301402	MEGH DINESHBHAI BAVARVA
202301403	SHOBHITCHANDRA CHAUDHARI
202301404	RAMANI DARSHIT
202301405	TANISH TANAWALA
202301406	KHUSHI PANDYA
202301407	KARTHIK RAMAN BALAMURUGAN
202301408	VRAJ PATEL
202301409	KIRTAN CHAUHAN

Contents

1 Introduction to LDPC Codes

- LDPC (Low-Density Parity-Check) codes were first introduced by Robert G. Gallager in 1962.
- Low-density parity-check (LDPC) codes are a class of linear block codes. The LDPC matrix is specified by a matrix containing mostly 0's and few 1's.
- The structure of LDPC codes allows them to be easily adapted to different block lengths and rates, offering flexibility for various applications.
- LDPC codes are capable of achieving performance very close to the theoretical maximum (Shannon limit) for data transmission over noisy channels, making them highly desirable in practical systems.
- In 5G NR (New Radio), LDPC codes are used primarily for channel coding of data channels, offering high reliability and near-capacity performance.

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

Figure 1: Example LDPC structure.

Here, n represents the number of columns.
J = number of 1's in each column, $J \geq 3$.
K = number of 1's in each row, $K > J$.

2 LDPC Communication System Diagram

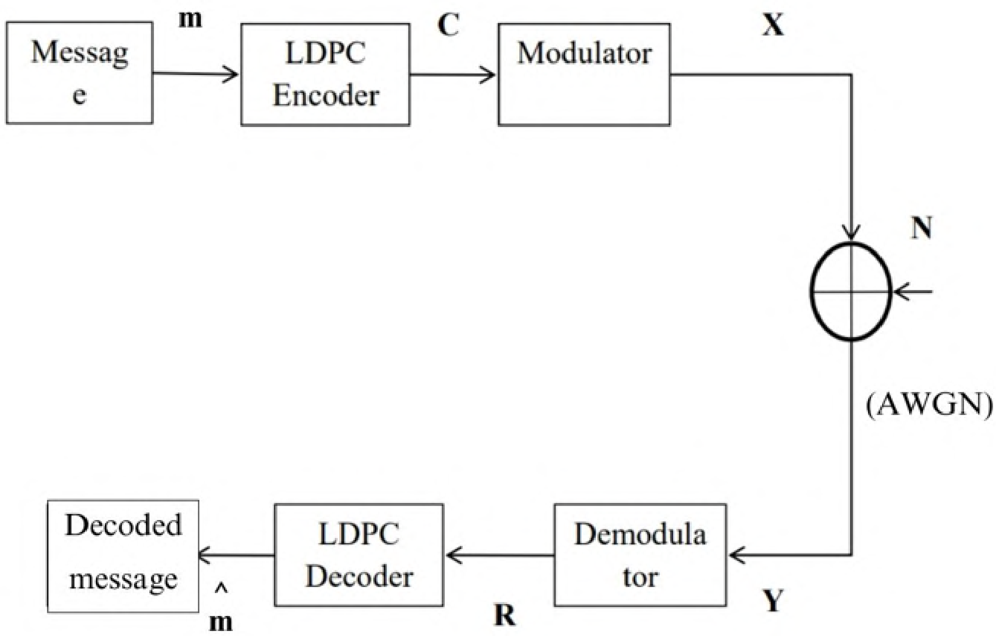


Figure 2: LDPC-based communication system with encoding, modulation, channel noise (AWGN), and decoding.

3 Definition of LDPC Code

- An **LDPC Code** is a type of linear block code with a **sparse parity-check matrix**.
- Let H be the parity-check matrix of size $(n-k) \times n$, where:
  - n: Total number of bits in the codeword
  - k: Number of message bits
- The number of 1s in the matrix is much less than the total number of entries, i.e., $\ll n(n-k)$.

4 5G NR Base Graph Matrix and Protograph Construction

- The parity-check matrix used in LDPC coding is derived from a representation known as the **Base Graph (BG)**.
- Two types of base graphs are defined in the 5G NR standard:
  - **BG1:** 46×68
  - **BG2:** 42×52
- These base matrices serve as compact templates for constructing large parity-check matrices using a method called **lifting** or **expansion**.
- Each entry in the base graph can take a value between -1 and $Z_c - 1$, where Z_c is known as the **expansion factor** (or lifting size).

- The number of information bits K depends on the chosen base graph and expansion factor Z_c:
  - For BG1: $K = 22 \cdot Z_c$
  - For BG2: $K = 10 \cdot Z_c$
- The interpretation of matrix entries during expansion is:
  - -1: Replaced with a $Z \times Z$ all-zero matrix.
  - 0: Replaced with a $Z \times Z$ identity matrix.
  - X (where $0 < X < Z_c$): Replaced with a $Z \times Z$ identity matrix circularly right-shifted by X.

5 Efficient Encoding using Double Diagonal LDPC Structure

I_k: Identity matrix of size $Z_c \times Z_c$ circularly shifted to the right by k positions.

Message vector:

$$\mathbf{m} = \begin{bmatrix} m_1 & m_2 & m_3 & m_4 \end{bmatrix}$$

Codeword vector:

$$\mathbf{c} = \begin{bmatrix} m_1 & m_2 & m_3 & m_4 & p_1 & p_2 & p_3 & p_4 \end{bmatrix}$$

$$\left[\begin{array}{cccc|cccc} I_1 & 0 & I_3 & I_1 & I_2 & I & 0 & 0 \\ I_2 & I & 0 & I_3 & 0 & I & I & 0 \\ 0 & I_4 & I_2 & I & I_1 & 0 & I & I \\ I_4 & I_1 & I & 0 & I_2 & 0 & 0 & I \end{array}\right]$$

H matrix with Z_c: 5

The parity-check matrix H satisfies:

$$H \cdot \mathbf{c}^T = \mathbf{0}$$

Expanded Equations from $H \cdot c^T = 0 \pmod 2$:

$$\begin{aligned} &(1) \quad I_1 m_1 + I_3 m_3 + I_1 m_4 + I_2 p_1 + I p_2 = 0 \\ &(2) \quad I_2 m_1 + I m_2 + I_3 m_3 + I p_2 + I p_3 = 0 \\ &(3) \quad I_4 m_2 + I_2 m_3 + I m_4 + I_1 p_1 + I p_3 + I p_4 = 0 \\ &(4) \quad I_4 m_1 + I_1 m_2 + I m_3 + I_2 p_1 + I p_4 = 0 \end{aligned}$$

Step 1: Compute p_1
Add all four equations (1) through (4) modulo 2:

$$I_1p_1 = I_1m_1 + I_3m_3 + I_1m_4 + I_2m_1 + Im_2 + I_3m_3 + I_4m_2 + I_2m_3 + Im_4 + I_4m_1 + I_1m_2 + Im_3$$

Step 2: Use Eq (1) to solve for p_2

$$Ip_2 = -(I_1m_1 + I_3m_3 + I_1m_4 + I_2p_1)$$

Step 3: Use Eq (2) to solve for p_3

$$Ip_3 = -(I_2m_1 + Im_2 + I_3m_3 + Ip_2)$$

Step 4: Use Eq (4) to solve for p_4

$$Ip_4 = -(I_4m_1 + I_1m_2 + Im_3 + I_2p_1)$$

6 Modulation

6.1 Simulation of BPSK Modulation and AWGN

1. The output of the channel encoder or the information source is a vector, whose each element is a bit $b \in \{0, 1\}$.
2. Each of these bits is sent to the BPSK modulator, which maps b to s, where

$$s = \begin{cases} +1 \text{ volts}, & \text{if } b = 0 \\ -1 \text{ volts}, & \text{if } b = 1 \end{cases}$$

   This mapping can also be implemented as $s = 1 - 2b$.
3. Let the AWGN introduce a per-symbol SNR $\gamma = \frac{E_s}{N_0}$ (in linear scale). Since BPSK symbols $s \in \{-1, +1\}$, the signal energy $E_s = 1$ Joule (proportional), hence noise power is:

$$\sigma_n^2 = \frac{1}{\gamma}$$

4. The AWGN is simulated as:

$$u = \sigma_n \cdot u_s$$

   where $u_s \sim \mathcal{N}(0, 1)$ is a standard Normal variate.

6.2 Modulation Over AWGN Channel

- a: Bits to be transmitted
- $\hat{a}$: Receiver's estimated value of transmitted bit
- $s(t)$: Waveform transmitted into the channel
- $n(t)$: White Gaussian Noise
- $r(t) = s(t) + n(t)$: Waveform received after noise
- r: Information about waveform after filtering at the receiver

6.2.1 Bit-Error Rate (BER)

- $\text{BER} = \Pr\{a \neq \hat{a}\}$

Estimating BER:

- Transmit N bits (e.g., $N = 10^7$)
- Find number of errors n_e (must be at least 100)
- $\text{BER} = \frac{n_e}{N}$

Signal-to-Noise Ratio (SNR):

- $\text{SNR} = \frac{\text{Signal Power}}{\text{Noise Power}}$
- $\text{SNR (dB)} = 10 \log_{10}\left(\frac{\text{Signal Power}}{\text{Noise Power}}\right)$

6.2.2 Continuous-Time AWGN Channel

- Signal Power $= P$
- Noise Power Spectral Density (PSD) $= \frac{N_0}{2}$
- Bandwidth $= 2W$
- $\text{SNR} = \frac{P}{N_0 W}$
- Time per symbol $= T = \frac{1}{2W}$

6.2.3 Discrete-Time AWGN Channel

- s: Symbol representing $s(t)$
- n: Noise after receive filtering
- $n \sim \mathcal{N}(0, \sigma^2)$, independent from symbol to symbol
- Assumption: Receive filter produces sufficient statistics

Discrete-Time Channel Parameters:

- Energy per symbol $E_s = PT = \frac{P}{2W}$
- Noise Energy (variance of noise): $\sigma^2 = \frac{N_0}{2}$
- Signal Energy:

$$E_s = \text{Mean of } s^2 = \frac{(-1)^2 + (+1)^2}{2} = 1$$

- $\text{SNR} = \frac{1}{\sigma^2}$

6.3 BER for BPSK over AWGN

The Bit Error Rate (BER) is expressed as:

$$\mathrm{BER} = \Pr\{s = +1\} \cdot \Pr\{n < -1\} + \Pr\{s = -1\} \cdot \Pr\{n > 1\}$$

Assuming equal probabilities for $s = \pm 1$, i.e., $\Pr\{s = \pm 1\} = 0.5$, and that n is Gaussian with variance σ^2, we get:

$$\mathrm{BER} = Q\left(\frac{1}{\sigma}\right) = Q\left(\sqrt{\mathrm{SNR}}\right)$$

Where the Q-function is defined as:

$$Q(x) = \frac{1}{2} \cdot \operatorname{erfc}\left(\frac{x}{\sqrt{2}}\right)$$

Here, `erfc` is the complementary error function.

7 Decoding

7.1 SPC and Repetition

7.1.1 Received Signal Definitions

- **Signal Definitions:**
  - $r_1 = 1 + N_1(0, \sigma^2)$
  - $r_2 = 1 + N_2(0, \sigma^2)$
  - $r_3 = 1 + N_3(0, \sigma^2)$
- **Additional Information:**
  - $N_1, N_2, N_3 \sim \mathcal{N}(0, \sigma^2)$ are independent noise terms.
  - Decoded message symbol vector: $c = [+1, +1, +1]$.

7.1.2 Log-Likelihood

Derivation of Log-Likelihood Ratio L_1

We derive the log-likelihood ratio L_1 for the received signals r_1, r_2, r_3 in a BFSK system with AWGN.

Step 1: Define the Received Signals and Their PDFs Given $r_i = 1 + N_i$, where $N_i \sim \mathcal{N}(0, \sigma^2)$, this corresponds to a transmitted signal of $+1$. For a transmitted signal of -1, we assume $r_i = -1 + N_i$. The PDFs are:

$$P(r_i \mid +1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(r_i - 1)^2}{2\sigma^2}\right), \tag{1}$$

$$P(r_i \mid -1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(r_i + 1)^2}{2\sigma^2}\right). \tag{2}$$

Step 2: Compute the Log-Likelihood Ratio for One Signal The LLR for r_i is:

$$\mathrm{LLR}_i = \log\left(\frac{P(r_i \mid +1)}{P(r_i \mid -1)}\right) = -\frac{(r_i - 1)^2}{2\sigma^2} + \frac{(r_i + 1)^2}{2\sigma^2}. \tag{3}$$

Simplify the exponent:

$$(r_i + 1)^2 - (r_i - 1)^2 = (r_i^2 + 2r_i + 1) - (r_i^2 - 2r_i + 1) = 4r_i. \tag{4}$$

Thus:

$$\text{LLR}_i = \frac{4r_i}{2\sigma^2} = \frac{2r_i}{\sigma^2}. \tag{5}$$

Step 3: Total LLR Since r_1, r_2, r_3 are independent, the total LLR is:

$$L_1 = \text{LLR}_1 + \text{LLR}_2 + \text{LLR}_3 = \frac{2r_1}{\sigma^2} + \frac{2r_2}{\sigma^2} + \frac{2r_3}{\sigma^2} = \frac{2}{\sigma^2}(r_1 + r_2 + r_3). \tag{6}$$

This matches the given form with $\lambda = 1$.

Now, the other log-likelihood terms:

$$L_2 = r_1 + r_2 + r_3 \quad \text{(intrinsic)} \tag{7}$$

$$L_2 = r_1 + r_2 \quad \text{(extrinsic)} \tag{8}$$

7.1.3 Final Approximation

Final Approximation Results

Final approximation out: intrinsic out

$$|\text{extrinsic}| = \min(|\text{intrinsic}|, |r_3|)$$

7.1.4 Summary of Key Results

Key Results Summary

Here, we summarize the main findings from the lecture notes:

- The total log-likelihood ratio for the received signals is:

$$L_1 = \frac{2}{\sigma^2}(r_1 + r_2 + r_3), \quad \text{with} \quad \lambda = 1.$$

- The log-likelihood L_2 has both intrinsic and extrinsic forms:

$$L_2 = r_1 + r_2 + r_3 \quad \text{(intrinsic)}, \quad L_2 = r_1 + r_2 \quad \text{(extrinsic)}.$$

- The final approximation for the extrinsic component is:

$$|\text{extrinsic}| = \min(|\text{intrinsic}|, |r_3|).$$

7.1.5 Repetition Code

Repetition Code Derivation

In this section, we describe how the received signals r_1, r_2, r_3 and the decoded message symbol vector $c = [+1, +1, +1]$ relate to a repetition code in the BFSK system.

A repetition code sends the same symbol multiple times to improve reliability. Here, the transmitted symbol s (either $+1$ or -1) is sent three times, resulting in the received signals r_1, r_2, r_3. The decoding process uses a majority voting rule based on individual decisions for each r_i.

Step 1: Decision Rule for Each Received Signal For each r_i, we decide the transmitted symbol as follows:

- If $\text{LLR}_i = \frac{2r_i}{\sigma^2} > 0$, i.e., $r_i > 0$, decide $d_i = +1$.
- If $r_i < 0$, decide $d_i = -1$.

This threshold is derived from the LLR computed earlier, where $\text{LLR}_i > 0$ favors $+1$.

Step 2: Individual Decisions Apply the decision rule to each received signal:

- For r_1, decide $d_1 = +1$ if $r_1 > 0$, otherwise $d_1 = -1$.
- For r_2, decide $d_2 = +1$ if $r_2 > 0$, otherwise $d_2 = -1$.
- For r_3, decide $d_3 = +1$ if $r_3 > 0$, otherwise $d_3 = -1$.

Step 3: Majority Decoding With three repetitions, the final decoded symbol is determined by majority voting:

- If at least two of d_1, d_2, d_3 are $+1$, the decoded symbol is $+1$.
- Otherwise, the decoded symbol is -1.

Given the decoded vector $c = [+1, +1, +1]$, this indicates $d_1 = +1$, $d_2 = +1$, and $d_3 = +1$, so the final decoded symbol is $+1$, consistent with a repetition code.

7.1.6 SPC Code

Single Parity-Check (SPC) Code

Proof of (3, 2) Single Parity-Check (SPC) Code Given message $M = [m_1, m_2]$, encode into $[m_1, m_2, m_3]^T$ where m_3 is the parity bit such that:

$$[(m_1 \oplus m_2) \oplus m_3 = 0]$$

$$(m_1 \oplus m_2) \oplus m_3 = (m_1 \oplus m_2 \oplus m_3) = 0$$

Encoding Generator matrix G:

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

Codeword $C = [c_1, c_2, c_3]^T$:

$$C = M \cdot G$$

Example:

$$M = [0, 1] \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = [0, 1, 1]$$

$$(m_1 \oplus m_2) = 0 \oplus 1 = 1, \quad m_3 = 1$$

$$(m_1 \oplus m_2 \oplus m_3) = 0 \oplus 1 \oplus 1 = 0$$

Parity Check Parity check matrix H:

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

Syndrome S:

$$S = H \cdot C^T \mod 2$$

If $S = 0$, no error. Example mapping:

c_1	c_2	c_3	map
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$P_2 = P((c_2 \oplus c_3)|n_2, n_3)$$

Decoding Received vector $R = [r_1, r_2, r_3]^T$:

$$[r_1, r_2, r_3]^T \mod H^T = 0$$

SISO decoder for SPC code (3, 2):

$$M = [m_1, m_2] \to [\text{SPC code}] \cdot c = [c_1, c_2, c_3] \to \hat{M} = [m_1, m_2]$$

Error Correction

$$\begin{aligned}1 - P_1 &= P_2(1 - P_3) + P_3(1 - P_2) \\ P_1 \cdot (1 - P_3) &= P_2 \cdot [P_3 - (1 - P_3)] + (1 - P_2) \cdot (1 - P_3) - P_3 \\ P_1 - (1 - P_3) &= \frac{P_2 - (1 - P_2)}{1 + (1 - P_2)/P_3} \cdot \frac{1 - (1 - P_3)/P_3}{1 + (1 - P_2)/P_3} \\ &= \frac{P_2 - (1 - P_2)}{1 + \frac{1-P_2}{P_3}} \cdot \frac{1 - \frac{1-P_3}{P_3}}{1 + \frac{1-P_2}{P_3}}\end{aligned}$$

Final:

$$[(m_1 \oplus m_2) \oplus (m_1 \oplus m_2 \oplus m_3) = 0]$$

$$\hat{M} = \text{algorithmic} \quad \text{and} \quad \hat{C} = [c_1 \oplus c_3]$$

8 Simulation Codes

8.1 Overview

This section presents the MATLAB code for simulating LDPC codes specified in the 5G NR standard using BPSK modulation over an AWGN channel. Both hard and soft decoding algorithms are implemented for the NR_2_6_52 base graph.

Support Functions

Shift Function

```
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
```

Listing 1: Shift Function for Encoding

NR LDPC H-matrix Generation

```
function [B,H,z] = nrldpc_Hmatrix(BG)
    load(sprintf('%s.txt',BG),BG);
    B = NR_2_6_52;
    [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

    [U,N]=size(H); K = N-U;
    P = H(:,1:K);
    G = [eye(K); P];
    Z = H*G;
end
```

Listing 2: NR LDPC H-matrix Generation Function

NR LDPC Encoding

```
function cword = nrldpc_encode(B,z,msg)
    %B: base matrix
    %z: expansion factor
    %msg: message vector

    [m,n] = size(B);

    cword = zeros(1,n*z); %initializing codeword vector
    cword(1:(n-m)*z) = msg;

    %double-diagonal encoding
    temp = zeros(1,z);
    for i = 1:4 %row 1 to 4
        for j = 1:n-m %message columns
            temp = mod(temp + mul_sh(msg((j-1)*z+1:j*z),B(i,j)),2);
        end
    end
    if B(2,n-m+1) == -1
        p1_sh = B(3,n-m+1);
    else
        p1_sh = B(2,n-m+1);
    end
    cword((n-m)*z+1:(n-m+1)*z) = mul_sh(temp,z-p1_sh); %p1
    %Find p2, p3, p4
    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
    %Remaining parities
    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
```

Listing 3: NR LDPC Encoding Function

Soft Decoding Simulation

```
baseGraph5GNR = 'NR_2_6_52'; % Load 5G NR LDPC base H matrix
Nsim = 500;
max_itr = 20;
FlagMatrix = zeros(Nsim, max_itr);
colors = ['r', 'g', 'b', 'm'];

EbNodB = 0:0.5:10;
ErrorProb = zeros(1, length(EbNodB));
R = [1/4 1/3 1/2 3/5];

%BER vs Eb/No(dB)
BER = zeros(length(R), length(EbNodB));

% Error Probability vs Eb/No(dB)
err = zeros(length(R), length(EbNodB));

% Success Probability vs No of Iteration
succ_itr = zeros(length(EbNodB), max_itr);

% Index for coderate
ind_cr = 1;

for codeRate = R % For base graph NR_2_6_52
    % AWGN Channel
    [B, Hfull, z] = nrldpc_Hmatrix(baseGraph5GNR);
    [mb, nb] = size(B);
    kb = nb - mb;
    InfoBits = kb * z;
    k_pc = kb - 2;
    nbRM = ceil(k_pc / codeRate) + 2;
    nBlkLength = nbRM * z;

    H = Hfull(:, 1:nBlkLength);
    nChecksUsed = mb*z - nb*z + nBlkLength;
    H = H(1:nChecksUsed, :);

    Nchecks = size(H, 1);

    Rows = size(H, 1);
    Cols = size(H, 2);

    ebno_itr=1;

    for index = 1:length(EbNodB)
        EbNo = 10^(EbNodB(index) / 10);
        sigma = sqrt(1 / (2 * codeRate * EbNo));
        ErrorinBits = 0;

```

```
        for ksim = 1:Nsim
            b = randi([0 1], [InfoBits 1]);
            c = nrldpc_encode(B, z, b');
            c = c(1:nBlkLength)';

            % BPSK Modulation
            s = 1 - 2*c;

            rec_vec = s + sigma * randn(size(s));
            rec_vec = (rec_vec < 0);

            M = zeros(Rows, Cols);  % Variable-to-Check messages
            L = zeros(Rows, Cols);  % Check-to-Variable messages
            decoded_msg = zeros(1, Cols);

            % Initialize messages from variable nodes to check nodes
            for ic = 1 : Cols
                check_nodes = find(H(:, ic));
                for j = 1 : length(check_nodes)
                    M(check_nodes(j), ic) = rec_vec(ic);  % Initial message is received bit
                end
            end

            % Begin decoding iterations
            for itr = 1 : max_itr
                % Check node update (SPC)
                for ir = 1 : Rows
                    var_nodes = find(H(ir, :));
                    for j = 1 : length(var_nodes)
                        others = M(ir, var_nodes([1:j-1, j+1:end]));
                        L(ir, var_nodes(j)) = mod(sum(others), 2);  % Parity excluding self
                    end
                end

                %Variable node update (Majority)
                for ic = 1 : Cols
                    check_nodes = find(H(:, ic));
                    total_vote = sum(L(check_nodes, ic)) + rec_vec(ic);
                    decoded_msg(ic) = total_vote > ((length(check_nodes) + 1) / 2);

                    for j = 1 : length(check_nodes)
                        other_votes = L(check_nodes([1:j-1, j+1:end]), ic);
                        cnt = sum(other_votes) + rec_vec(ic);
                        M(check_nodes(j), ic) = cnt > (length(check_nodes) / 2);
                    end
                end

                if (isequal(decoded_msg(:), c(:)) && ind_cr==2) % increment success count if decoded msd is same as codeword
                     succ_itr(ebno_itr,itr) = succ_itr(ebno_itr,itr) + 1;
                end
```

```
            end

            % Count bit errors
            bitError = sum(decoded_msg(:) ~= c(:));
            if(bitError > 0)
                BER(ind_cr, ebno_itr) = BER(ind_cr, ebno_itr) + bitError;
                err(ind_cr, ebno_itr) = err(ind_cr, ebno_itr) + 1;
            end
        end
        BER(ind_cr, ebno_itr) = BER(ind_cr, ebno_itr) / Cols / Nsim;
        err(ind_cr, ebno_itr) = err(ind_cr, ebno_itr) / Nsim;

        ebno_itr=ebno_itr+1;
    end
    ind_cr=ind_cr+1;
end

succ_itr = succ_itr/Nsim;

EbNo = 10 .^ (EbNodB ./ 10);

% Calculate Bit Error Rate (BER) for each SNR
ber_uncoded = 0.5 * erfc(sqrt(EbNo ./ 2));

figure;
for idx = 1:length(R)
    semilogy(EbNodB, BER(idx, :),'Color', colors(idx), 'LineWidth', 2);
    hold on;
end
semilogy(EbNodB, ber_uncoded,'Color','k', 'LineWidth', 2);
xlabel('EbNo in dB');
ylabel('BER');
title('BER Performance (Log Scale)');
legend('Rate = 1/4', 'Rate = 1/3', 'Rate = 1/2', 'Rate = 3/5','Uncoded BPSK');
hold off;

figure;
hold on;
grid on;
for idx = 1:length(R)
    plot(EbNodB, BER(idx, :), 'Color', colors(idx), 'LineWidth', 2);
end
plot(EbNodB, ber_uncoded,'Color','k', 'LineWidth', 2);
xlabel('E_b/N_0 (dB)');
ylabel('BER');
title('BER Performance (Linear Scale)');
legend('Rate = 1/4', 'Rate = 1/3', 'Rate = 1/2', 'Rate = 3/5', 'Uncoded BPSK');
hold off;

figure;
hold on;
legendEntries = cell(1, length(EbNodB));
for i=1:length(EbNodB)
    plot(1:max_itr,succ_itr(i,:),'LineWidth',2);
    legendEntries{i} = sprintf('Eb/N0 = %.1f dB', EbNodB(i));
```

```
end
xlabel('Iteration');
ylabel('Success Probability');
title('Success Probability vs Iteration for Different Eb/N0');
legend(legendEntries, 'Location', 'northeast');
grid on;
hold off;

figure;
hold on;
grid on;
for idx = 1:length(R)
    plot(EbNodB, 1-err(idx, :), 'Color', colors(idx), 'LineWidth', 2);
end
xlabel('E_b/N_0 (dB)');
ylabel('Probability of Success');
title('Probability of Success vs E_b/N_0 (dB)');
legend('Rate = 1/4', 'Rate = 1/3', 'Rate = 1/2', 'Rate = 3/5');
hold off;

figure;
hold on;
grid on;
for idx = 1:length(R)
    plot(EbNodB, err(idx, :), 'Color', colors(idx), 'LineWidth', 2);
end
xlabel('E_b/N_0 (dB)');
ylabel('Probability Of Decoding Failure');
title('Probability Of Decoding Failure vs E_b/N_0 (dB)');
legend('Rate = 1/4', 'Rate = 1/3', 'Rate = 1/2', 'Rate = 3/5');
hold off;
```

Listing 4: Main 5G NR LDPC Soft Decoding Simulation Script

Hard Decoding Simulation

```
baseGraph5GNR = 'NR_2_6_52'; % Base graph 2 for LDPC
R = [1/4, 1/3, 1/2, 3/5];      % Code rates
[B, Hfull, z] = nrldpc_Hmatrix(baseGraph5GNR);
[mb, nb] = size(B);
kb = nb - mb;
Ebnodb_range = 0:0.5:10;
Nsim = 100;
max_itr = 20;

BER = zeros(length(R), length(Ebnodb_range));
p_error = zeros(length(R), length(Ebnodb_range));
p_success_itr = zeros(length(Ebnodb_range), max_itr);

ind_cr = 1;
for codeRate = R
    kNumInfoBits = kb * z;
    k_pc = kb - 2;
    nbRM = ceil(k_pc/codeRate) + 2;
    nBlockLength = nbRM * z;
    n = nBlockLength;

```

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

    [row, col] = size(H);
    ebno_itr = 1;

    for ebnodb = Ebnodb_range
        Ebno = 10 ^ (ebnodb / 10);
        sigma = sqrt(1 / (2 * codeRate * Ebno));

        for i = 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(1, n);

            L = r .* H;
            sum_r = r;
            prev_decoded_msg = zeros(size(c));

            for itr = 1:max_itr
                for ir = 1:row
                    ind = find(H(ir, :) ~= 0);
                    [min1, minpos] = min(abs(L(ir, ind)));
                    min2 = min(abs(L(ir, ind([1:minpos-1 minpos+1:end])
                        )));
                    sgn = sign(L(ir, ind));
                    prod_sgn = prod(sgn);
                    L(ir, ind) = min1 .* prod_sgn;
                    L(ir, ind(minpos)) = min2 .* prod_sgn;
                    L(ir, ind) = sgn .* L(ir, ind);
                end

                sum_r = r + sum(L);
                for ic = 1:col
                    ind = find(H(:, ic) ~= 0);
                    L(ind, ic) = sum_r(ic) - L(ind, ic);
                end

                decoded_msg = sum_r < 0;

                if(decoded_msg == c && ind_cr == 1)
                    p_success_itr(ebno_itr, itr) = p_success_itr(
                        ebno_itr, itr) + 1;
                end
            end

            bitError = sum(decoded_msg ~= c);
            if bitError > 0
                BER(ind_cr, ebno_itr) = BER(ind_cr, ebno_itr) +
                    bitError;
                p_error(ind_cr, ebno_itr) = p_error(ind_cr, ebno_itr) +
                    1;
            end
        end

```

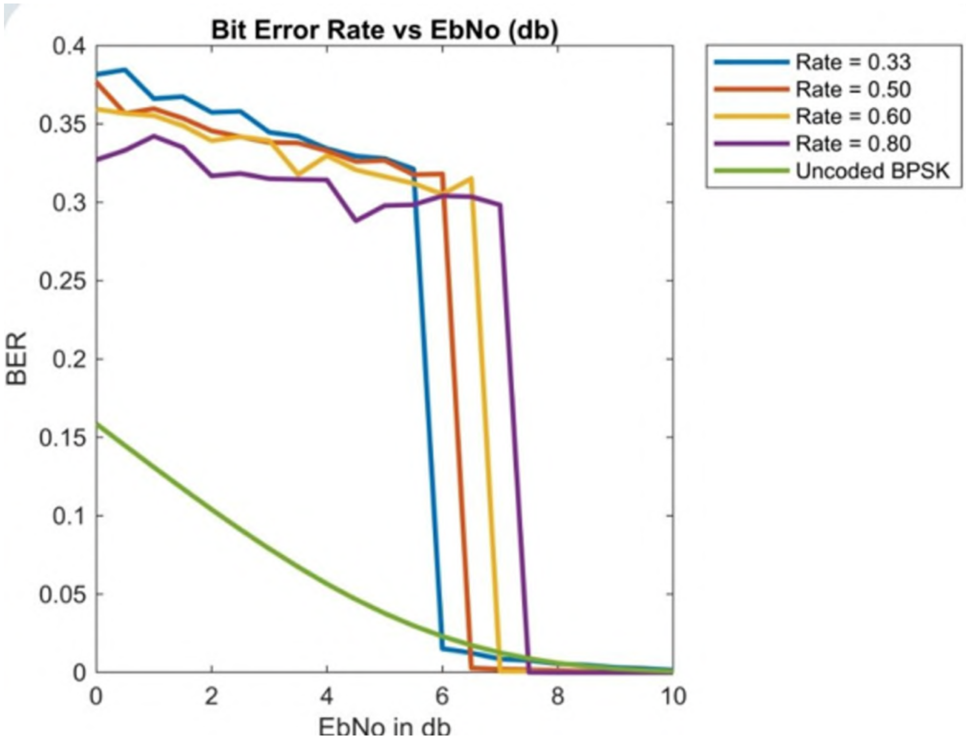
```
        BER(ind_cr, ebno_itr) = BER(ind_cr, ebno_itr) / n / Nsim;
        p_error(ind_cr, ebno_itr) = p_error(ind_cr, ebno_itr) / Nsim;
        ebno_itr = ebno_itr + 1;
    end
    ind_cr = ind_cr + 1;
end

% Uncoded BPSK Reference Curve and Plotting
Ebno = 10 .^ (Ebnodb_range ./ 10);
ber_uncoded = 0.5 * erfc(sqrt(Ebno ./ 2));
for i = 1:4
    semilogy(Ebnodb_range, BER(i, :), 'DisplayName', sprintf('Rate = %.2f', R(i)), 'LineWidth', 2);
    hold on;
end
semilogy(Ebnodb_range, ber_uncoded, 'DisplayName', 'Uncoded BPSK', 'LineWidth', 2);
legend('show','Location','bestoutside');
xlabel('Eb/No (dB)');
ylabel('BER (log scale)');
title('BER vs Eb/No with LDPC and Uncoded BPSK');
hold off;

for i = 1:4
    plot(Ebnodb_range, BER(i, :), 'DisplayName', sprintf('Rate = %.2f', R(i)), 'LineWidth', 2);
    hold on;
end
plot(Ebnodb_range, ber_uncoded, 'DisplayName', 'Uncoded BPSK', 'LineWidth', 2);
legend('show','Location','bestoutside');
xlabel('Eb/No (dB)');
ylabel('BER');
title('BER vs Eb/No with LDPC and Uncoded BPSK');
hold off;

for i = 1:4
    plot(Ebnodb_range, p_error(i, :), 'DisplayName', sprintf('Rate = %.2f', R(i)), 'LineWidth', 2);
    hold on;
end
legend('show','Location','bestoutside');
xlabel('Eb/No (dB)');
ylabel('Probability of Decoding Failure');
title('Probability of Decoding Failure vs Eb/No');
hold off;

for i = 1:4
    plot(Ebnodb_range, 1 - p_error(i, :), 'DisplayName', sprintf('Rate = %.2f', R(i)), 'LineWidth', 2);
    hold on;
end
legend('show','Location','bestoutside');
xlabel('Eb/No (dB)');
ylabel('Probability of Success');
title('Probability of Success vs Eb/No');
hold off;

```

```
for i = 1:length(Ebnodb_range)
    plot(1:max_itr, p_success_itr(i, :)/Nsim, 'DisplayName', sprintf('
       Ebnodb = %.2f', Ebnodb_range(i)), 'LineWidth', 2);
    hold on;
end
legend('show','Location','bestoutside');
xlabel('No. of Iterations');
ylabel('Probability of Success');
title('Probability of Success vs No. of Iterations');
hold off;
```

Listing 5: Main 5G NR LDPC Hard Decoding Simulation Script

9 Simulation Results: Hard vs. Soft Decoding

9.0. BER vs. Eb/No (Hard, NR_1_5_352) and BER vs. Eb/No (Soft, NR_1_5_352)



0.4

Figure 3: BER vs. Eb/No (Hard, NR_1_5_352)

0.4

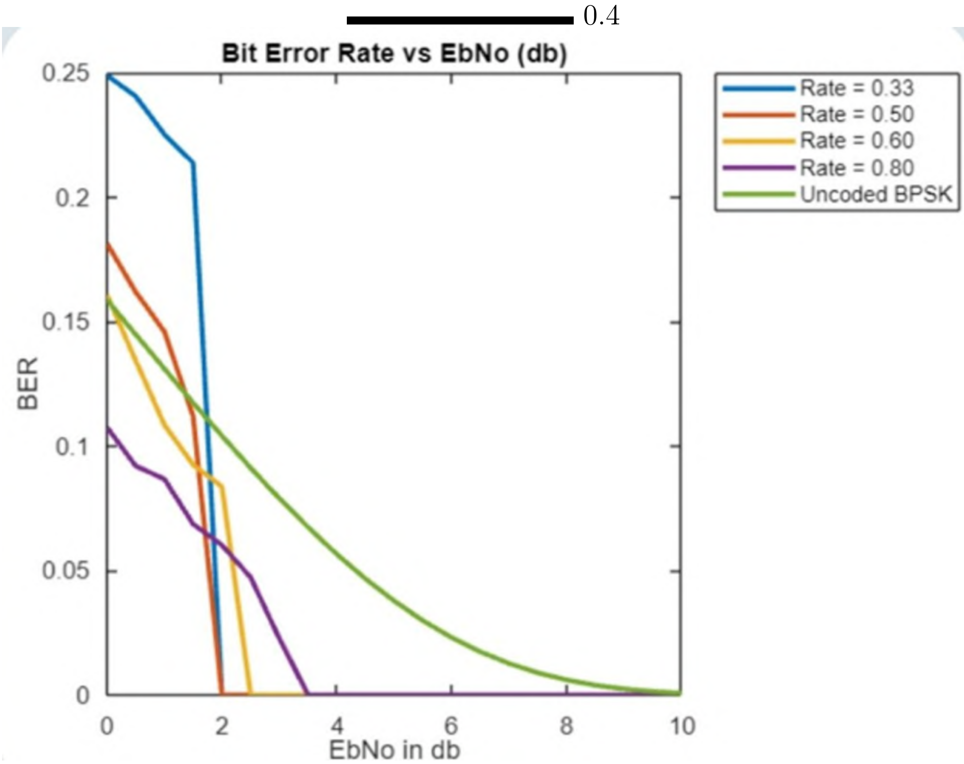
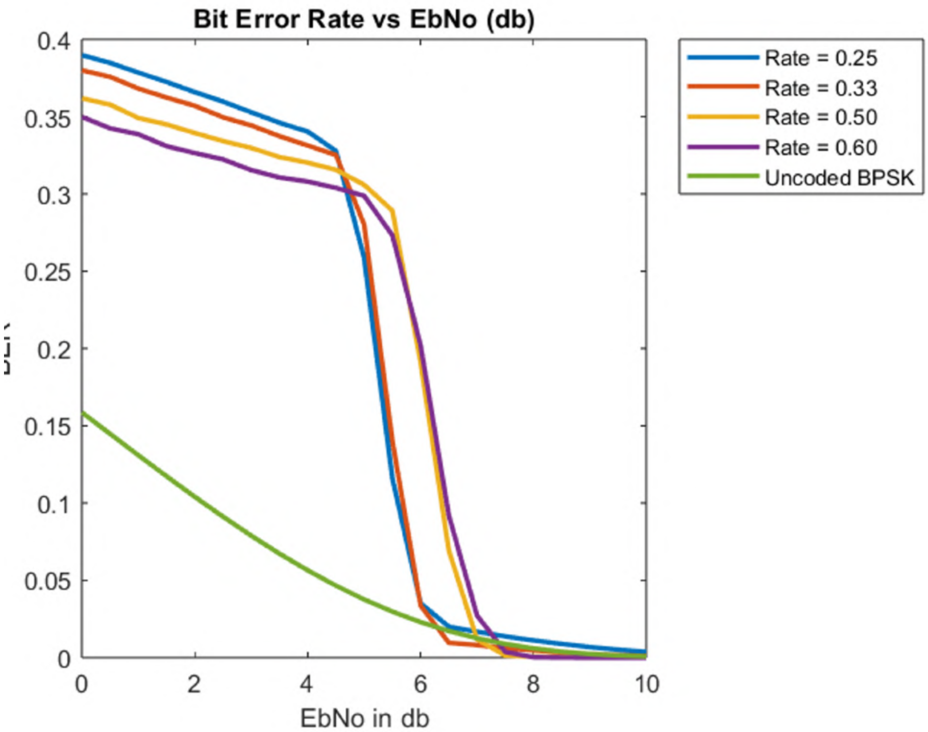


Figure 4: BER vs. Eb/No (Soft, NR_1_5_352)

Figure 5: BER vs. Eb/No (Hard, NR_1_5_352) and BER vs. Eb/No (Soft, NR_1_5_352)



0.4

Figure 6: BER vs. Eb/No (Hard, NR_2_6_52)

0.4

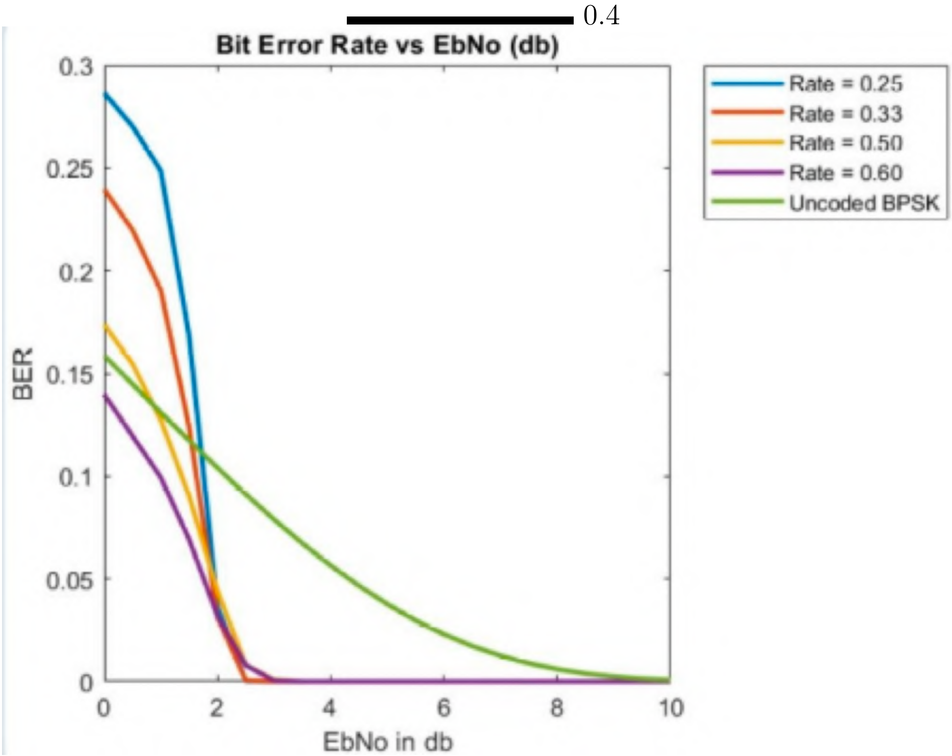
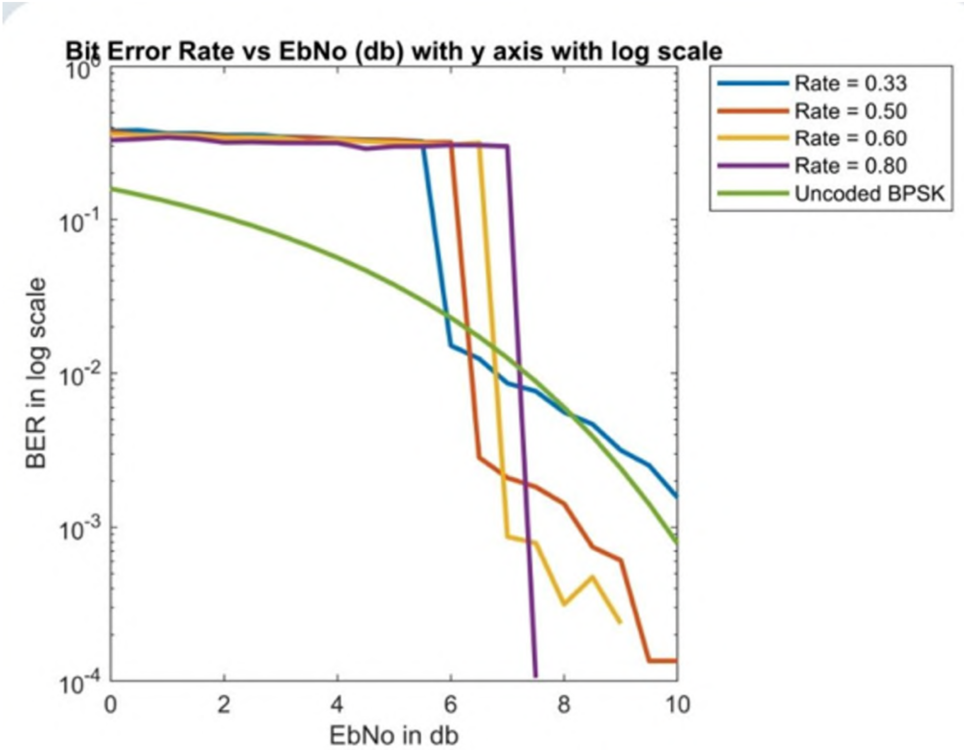


Figure 7: BER vs. Eb/No (Soft, NR_2_6_52)

Figure 8: BER vs. Eb/No (Hard, NR_2_6_52) and BER vs. Eb/No (Soft, NR_2_6_52)

9.0. Logarithmic BER vs. Eb/No (Hard, NR_1_5_352) and Logarithmic BER vs. Eb/No (Soft, NR_1_5_352)



0.4

Figure 9: Logarithmic BER vs. Eb/No (Hard, NR_1_5_352)

0.4

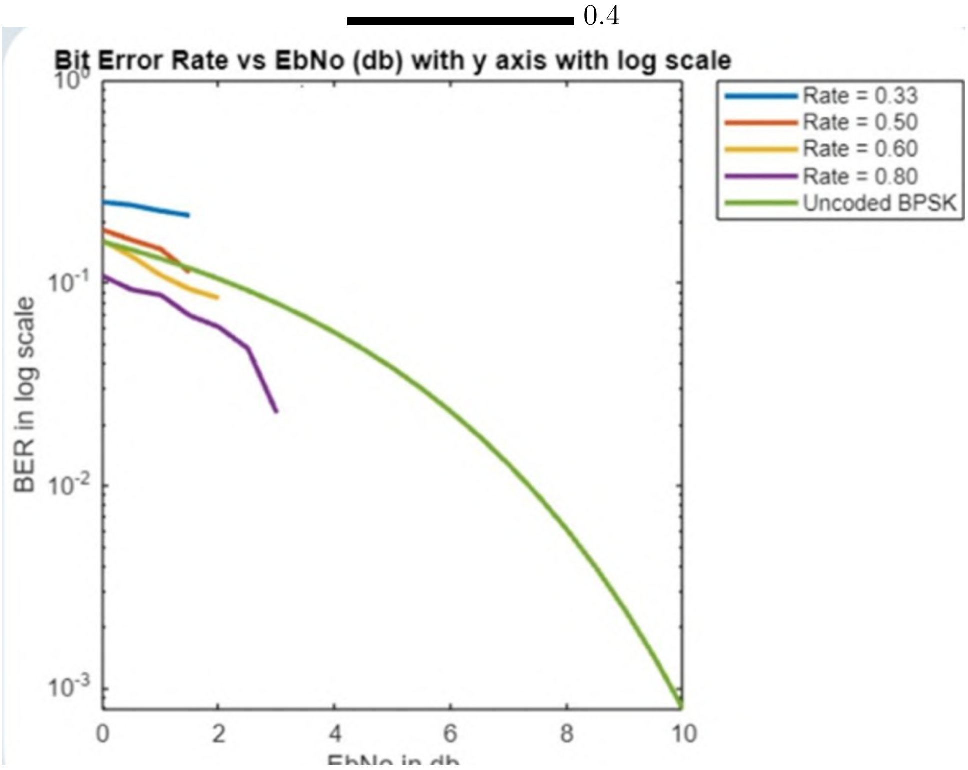
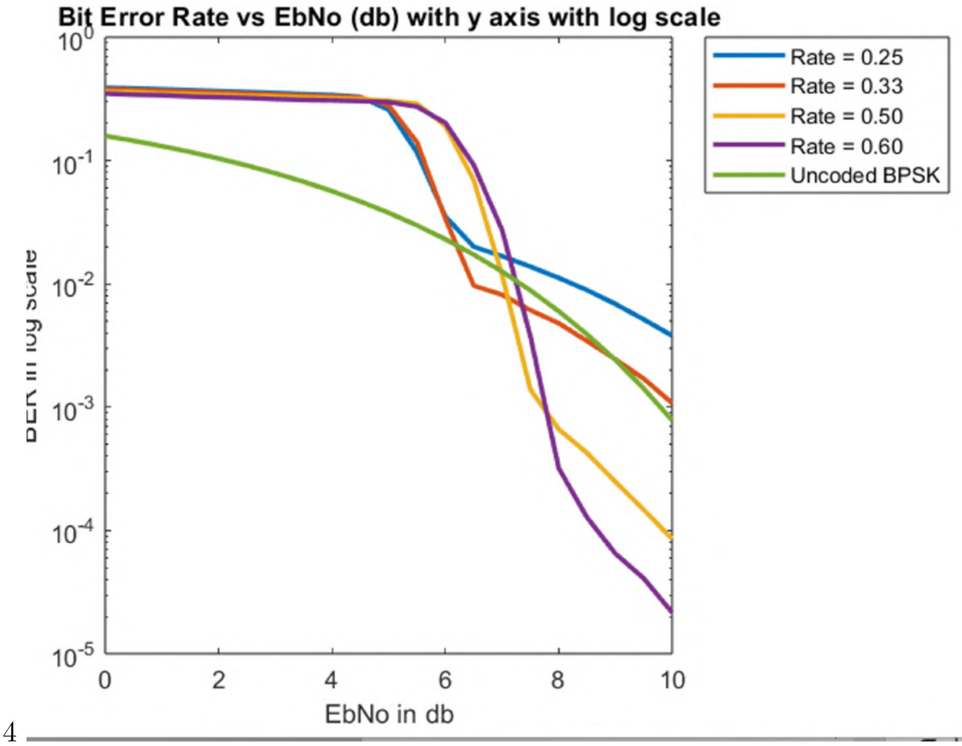


Figure 10: Logarithmic BER vs. Eb/No (Soft, NR_1_5_352)

Figure 11: Logarithmic BER vs. Eb/No (Hard, NR_1_5_352) and Logarithmic BER vs. Eb/No (Soft, NR_1_5_352)

9.0. Logarithmic BER vs. Eb/No (Hard, NR_2_6_52) and Logarithmic BER vs. Eb/No (Soft, NR_2_6_52)



0.4

Figure 12: Logarithmic BER vs. Eb/No (Hard, NR_2_6_52)

0.4

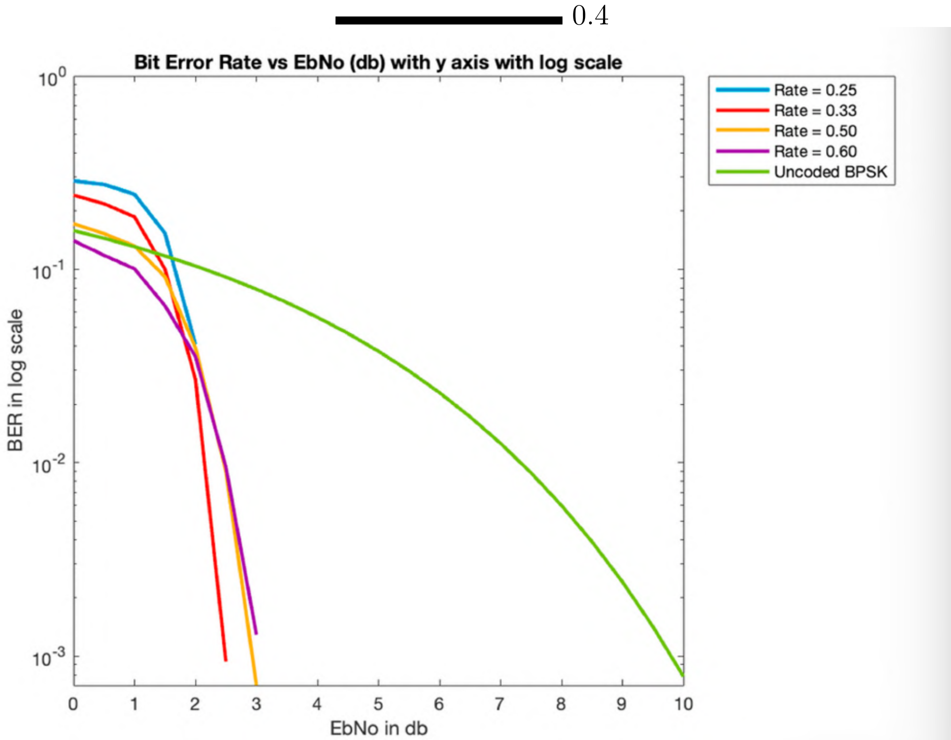
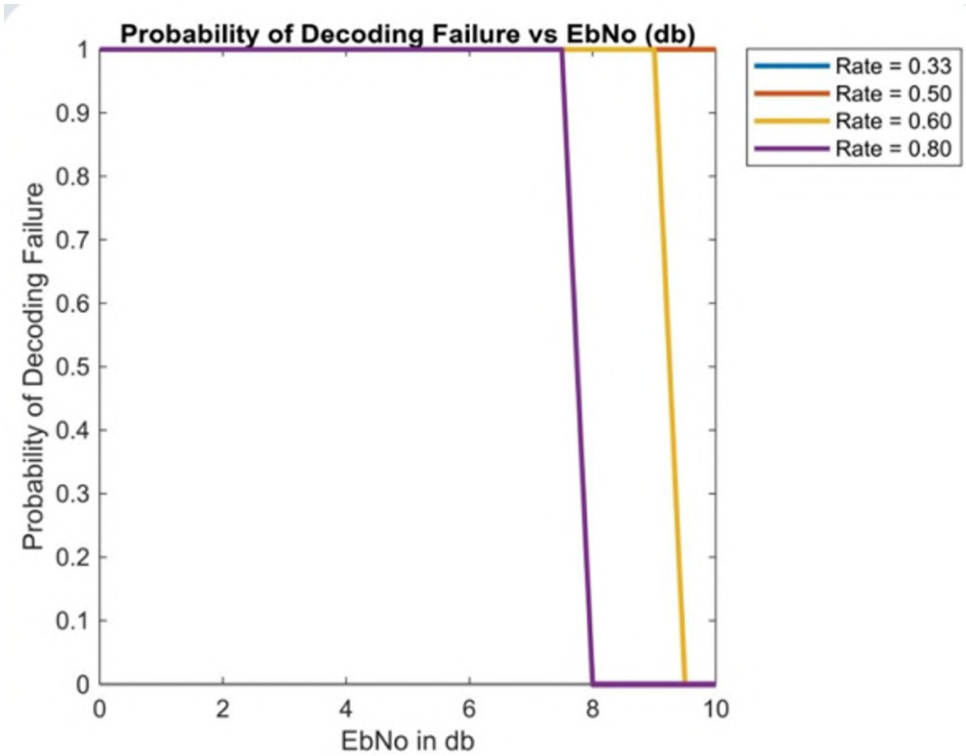


Figure 13: Logarithmic BER vs. Eb/No (Soft, NR_2_6_52)

Figure 14: Logarithmic BER vs. Eb/No (Hard, NR_2_6_52) and Logarithmic BER vs. Eb/No (Soft, NR_2_6_52)

9.0. Decoding error probability vs. Eb/No (Hard, NR_1_5_352) and Decoding error probability vs. Eb/No (Soft, NR_1_5_352)



0.4

Figure 15: Decoding error probability vs. Eb/No (Hard, NR_1_5_352)

0.4

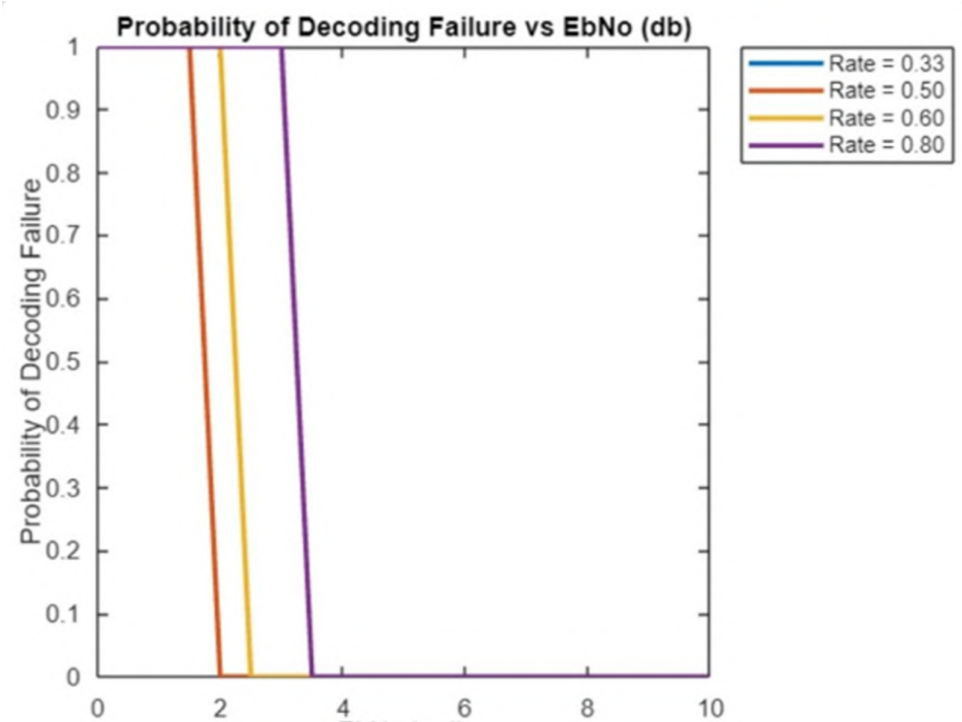
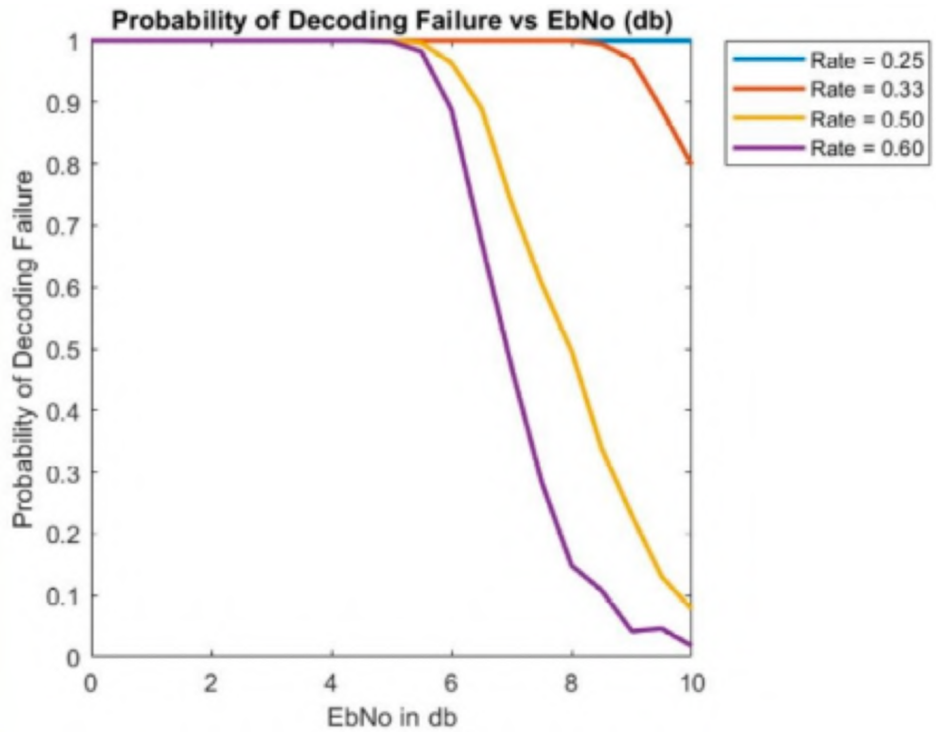


Figure 16: Decoding error probability vs. Eb/No (Soft, NR_1_5_352)

Figure 17: Decoding error probability vs. Eb/No (Hard, NR_1_5_352) and Decoding error probability vs. Eb/No (Soft, NR_1_5_352)

9.0. Decoding error probability vs. Eb/No (Hard, NR_2_6_52) and Decoding error probability vs. Eb/No (Soft, NR_2_6_52)



0.4

Figure 18: Decoding error probability vs. Eb/No (Hard, NR_2_6_52)

0.4

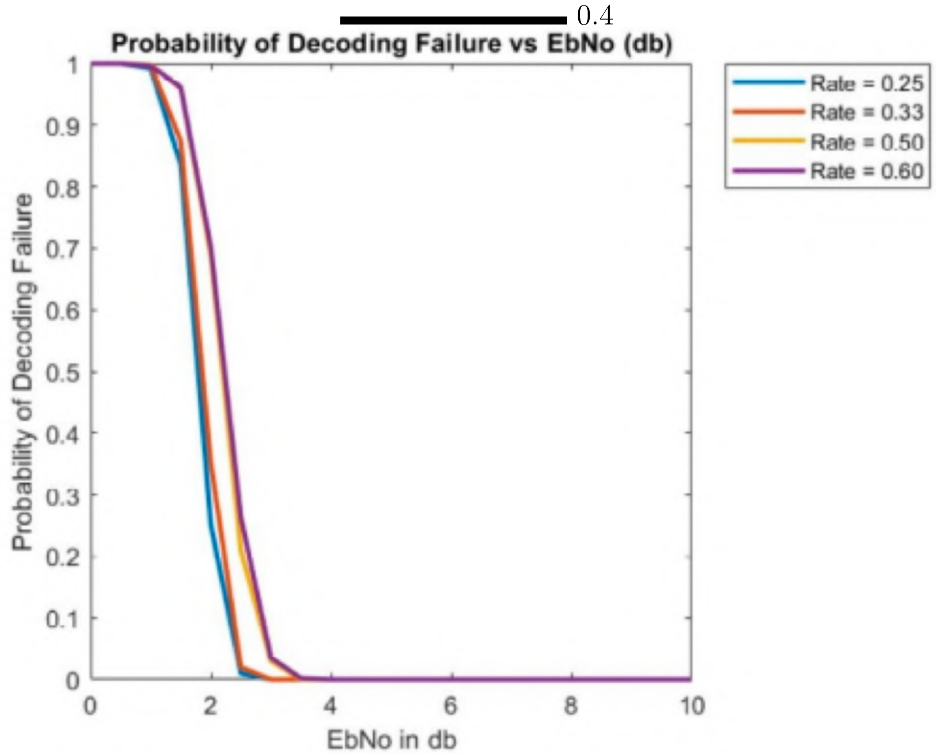
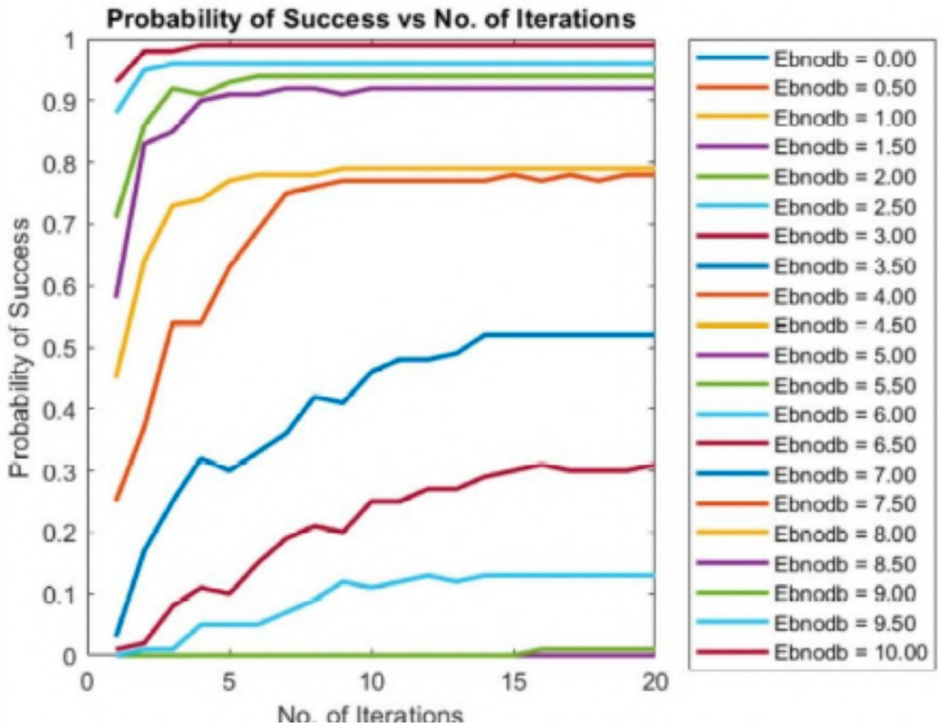


Figure 19: Decoding error probability vs. Eb/No (Soft, NR_2_6_52)

Figure 20: Decoding error probability vs. Eb/No (Hard, NR_2_6_52) and Decoding error probability vs. Eb/No (Soft, NR_2_6_52)

9.0. Success probability vs. iteration (Hard, NR_2_6_52) and Success probability vs. iteration (Soft, NR_2_6_52)



0.4

Figure 21: Success probability vs. iteration (Hard, NR_2_6_52)

0.4

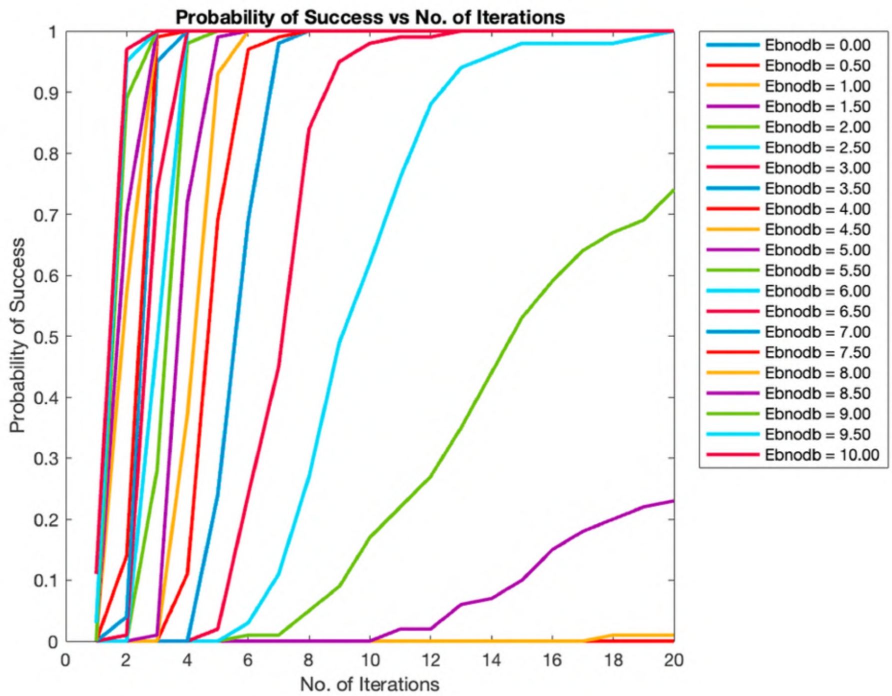
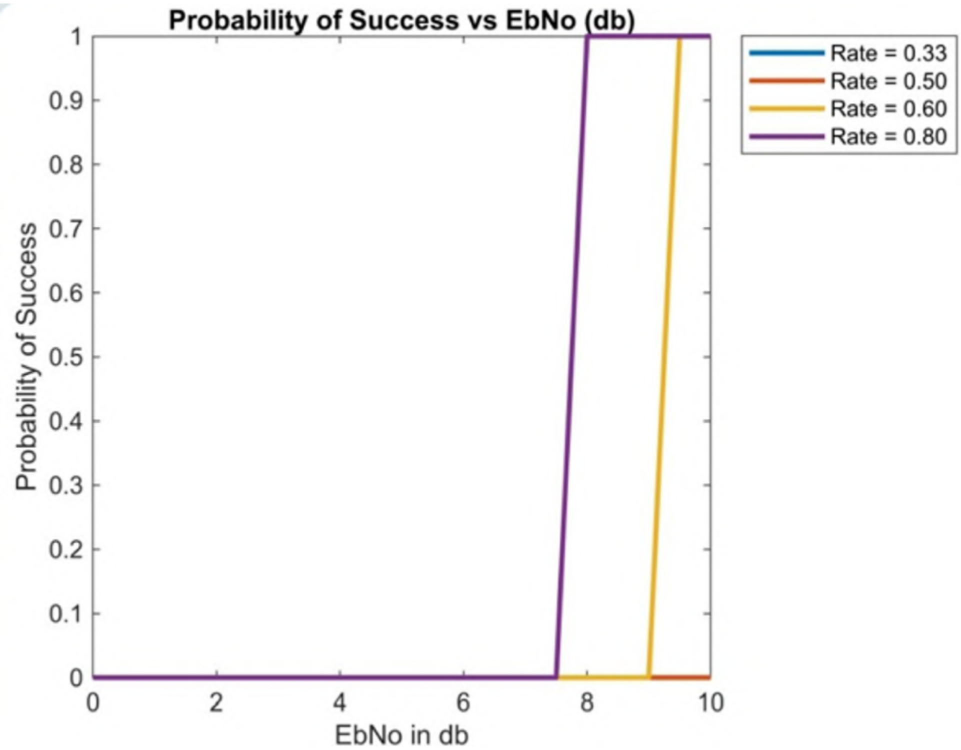


Figure 22: Success probability vs. iteration (Soft, NR_2_6_52)

Figure 23: Success probability vs. iteration (Hard, NR_2_6_52) and Success probability vs. iteration (Soft, NR_2_6_52)

9.0. Success probability vs. Eb/No (Hard, NR_1_5_352) and Success probability vs. Eb/No (Soft, NR_1_5_352)



0.4

Figure 24: Success probability vs. Eb/No (Hard, NR_1_5_352)

0.4

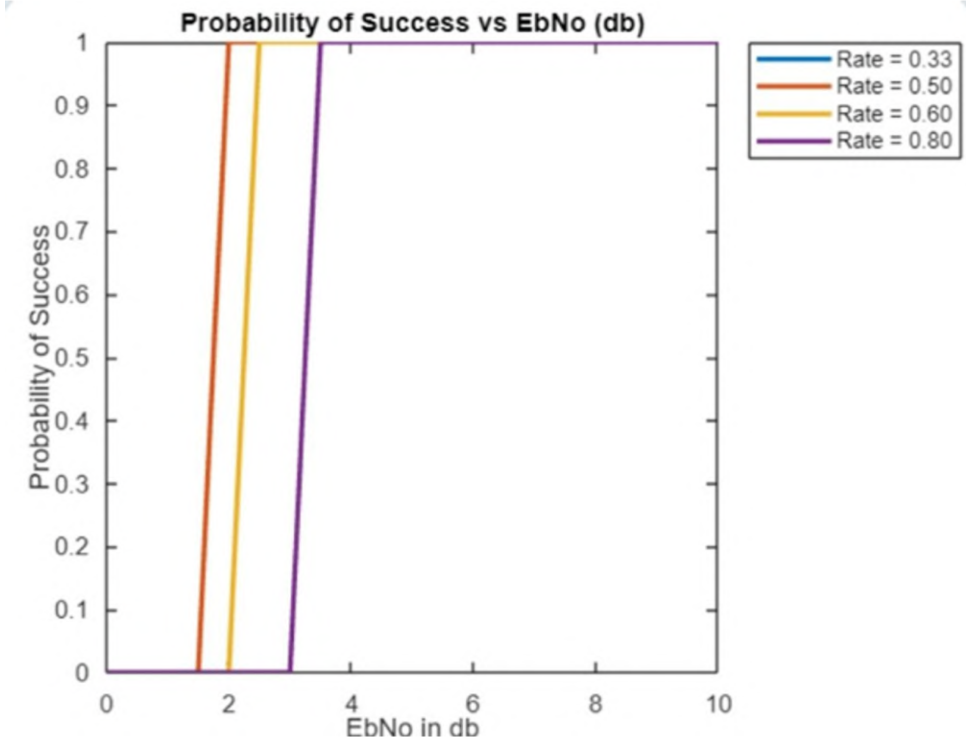
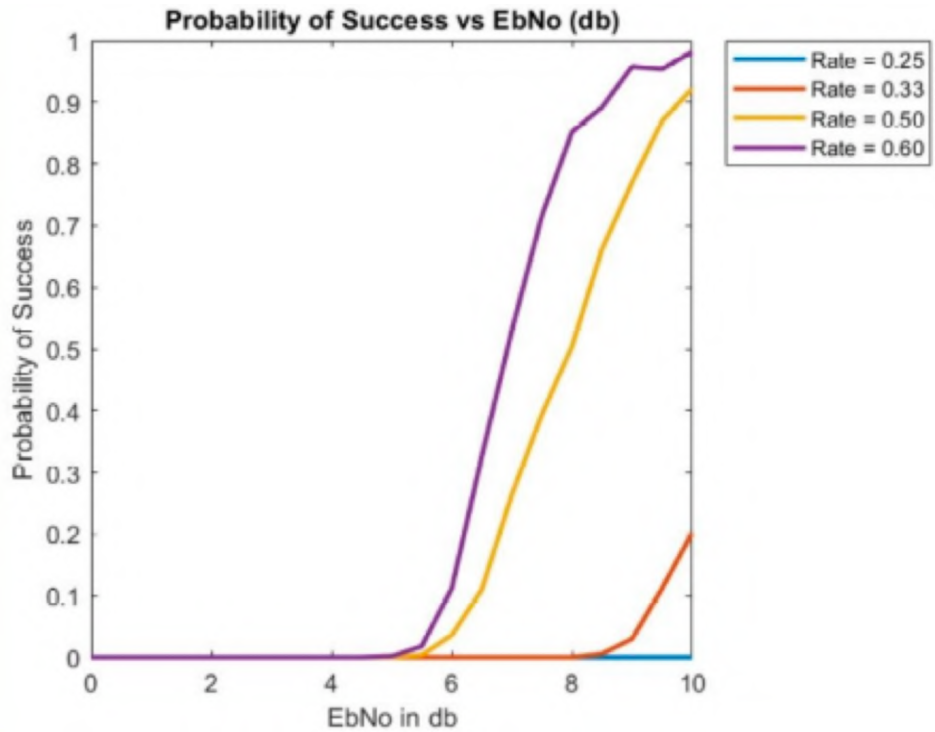


Figure 25: Success probability vs. Eb/No (Soft, NR_1_5_352)

Figure 26: Success probability vs. Eb/No (Hard, NR_1_5_352) and Success probability vs. Eb/No (Soft, NR_1_5_352)

9.0. Success probability vs. Eb/No (Hard, NR_2_6_52) and Success probability vs. Eb/No (Soft, NR_2_6_52)



0.4

Figure 27: Success probability vs. Eb/No (Hard, NR_2_6_52)

0.4

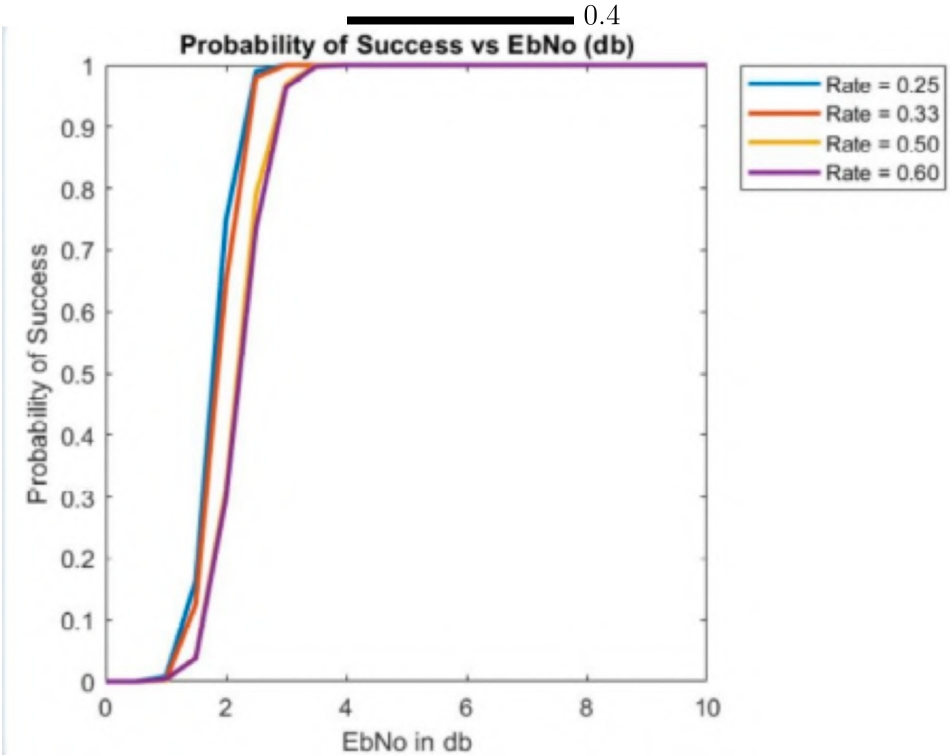


Figure 28: Success probability vs. Eb/No (Soft, NR_2_6_52)

Figure 29: Success probability vs. Eb/No (Hard, NR_2_6_52) and Success probability vs. Eb/No (Soft, NR_2_6_52)