CT216 - INTRODUCTION TO COMMUNICATION SYSTEM

Project Report

Project :- LDPC Codes For 5G NR

Lab Group - 3, Project Group - 16

Under the guidance of Prof. Yash Vasavda

Mentor :- Akshat Jindal

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 solemn pledge can carry grave consequences.

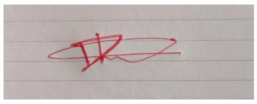
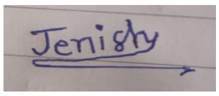
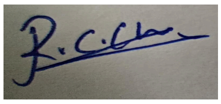
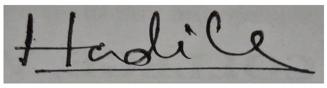
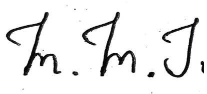
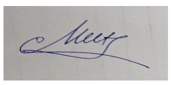
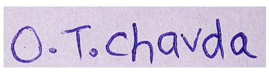
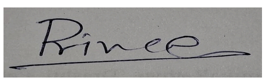
Student ID	Name	Signature
202301168	DHRUVIKA RATHOD	
202301169	BHAVYA THAKKAR	
202301171	ZEEL PARMAR	
202301172	MACWAN JENISH DENISH	
202301173	RUDRA CHAUHAN	
202301174	RATHVA HARDIKKUMAR CHHABIRAM	
202301175	JETHVA MANTHAN MANISHBHAI	
202301176	BARASARA MEET JITENDRABHAI	
202301177	CHAVDA OM TUSHARBHAI	
202301178	PATEL PRINCE JITUBHAI	

Table of Contents:

```
% Some needed funtions


function [B,H,z] = nrldpc_Hmatrix(BG)

    % load the base graph given in input (BG)
    load(sprintf('%s.txt',BG),BG);
    if strcmp(BG, 'NR_2_6_52')
        z = 52;
        BG = NR_2_6_52;
    else
        z = 352;
        BG = NR_1_5_352;
    end
    B = BG;
    [n, m] = size(B);
    % Initialize H matrix
    H = zeros(n*z, m*z);
    Iz = eye(z);
    I0 = zeros(z);

    % Lifting
    for i=1:n
        rw = (i-1)*z+(1:z);
        for j=1:m
            col = (j-1)*z+(1:z);
            if B(i, j)==-1
                H(rw, col) = I0;
            else
                H(rw, col) = circshift(Iz, -B(i, j));
                % -B(i, j) because we want to shift Iz up (or right) and
circshift default shifts the matrix downward
            end
        end
    end
end

function y = mul_sh(x, k)
    % x : intput bloack
    % k : -1 or shift
    % y : output
    if(k==-1)
        y = zeros(1, length(x));
    else
        y = [x(k+1:end) x(1:k)]; % multiplication by shifted identity
    end
end

function cword = nr5g_encoder(B, z, msg)
    [n, m] = size(B);
```

```
    cword = zeros(1, m*z);

    % Place the message bits in the beginning of the codeword
    cword(1:(m-n)*z) = msg;

    tm = zeros(1, z);
    for i=1:4
        for j=1:m-n
            % Multiply message blocks with base matrix entries
            tm = mod(tm+mul_sh(msg((j-1)*z+1:j*z), B(i, j)), 2);
        end

    end
    if B(2, m-n+1)==-1
        p1_sh = B(3, m-n+1);
    else
        p1_sh = B(2, m-n+1);
    end

    % Compute parity p1
    cword((m-n)*z+1:(m-n+1)*z) = mul_sh(tm, z-p1_sh); % got p1
    % z-p1_sh, because Ik inverse will be I(z-k) (Ik is k right shifted
identity)

    % Compute parity p2, p3, p4
    for i=1:3
        tm = zeros(1, z);
        for j=1:m-n+i
            % Multiply message blocks with base matrix entries
            tm = mod(tm+mul_sh(cword((j-1)*z+1:j*z), B(i, j)), 2);
        end
        cword((m-n+i)*z+1:(m-n+i+1)*z) = tm;
    end

    % Compute remaining parity from p5 to pn
    for i=5:n
        tm = zeros(1, z);
        for j=1:m-n+4
            tm = mod(tm+mul_sh(cword((j-1)*z+1:j*z), B(i, j)), 2);
        end
        cword((m-n+i-1)*z+1:(m-n+i)*z) = tm;
    end

    % totalparity = n*z;
    % info = m-n-2;
    % nbRM = ceil(info/coderate)+2;
    % nBlocklen = nbRM*z;
    %
    % needed_p = totalparity - (m*z - nBlocklen);
    % total_bits = m*z-n*z+needed_p;
```

```
    %
    % c_word = zeros(1, total_bits);
    % c_word(1:total_bits) = cword(1:total_bits);

    %this puncturing will be done seperatly in simulation after getting full
codeword
end
```

Hard Decision Decoding Simulation :-

```
baseGraph = 'NR_2_6_52';
% Coderates
coderate = [1/4 1/3 1/2 3/5];
Eb_no_db = 0:0.5:10;
colors = lines(length(Eb_no_db));

% Lifting
[B, Hfull, z] = nrldpc_Hmatrix(baseGraph);

% for storing the outputs
decoding_error = zeros(length(coderate), length(Eb_no_db));
bit_error = zeros(length(coderate), length(Eb_no_db));

% Number of simulations
nsim = 1000;

% Maximum iterations
max_it = 20;
iterations = 1:1:max_it;

for rr=1:length(coderate)
    [n, m] = size(B);
    cr = coderate(rr);

    % Adjusting H matrix for specific coderate
    totalparity = n*z;
    info = m-n-2;
    needed_blocks = ceil(info/cr)+2;
    nBlocklen = needed_blocks*z;
    needed_p = totalparity - (m*z - nBlocklen);
    total_bits = n*z-m*z+nBlocklen;
    H = Hfull(:, 1:nBlocklen);
    H = H(1:total_bits, :);


    [row, col] = size(H);
    infob = col-row;
```

```
    % Mapping for which check nodes connectd to a VNi
    vn_to_cn = cell(col, 1);
    % Mapping for which variable nodes connectd to a CNi
    cn_to_vn = cell(row, 1);
    % VN->CN and CN->VN msg storing matrix L
    L = zeros(row, col);

    % Mapping
    for i=1:col
        for j=1:row
            if H(j, i)==1
                vn_to_cn{i, 1} = [vn_to_cn{i, 1} j];
            end
        end
    end
    for i=1:row
        for j=1:col
            if H(i, j)==1
                cn_to_vn{i, 1} = [cn_to_vn{i, 1} j];
            end
        end
    end

    % To store output for iteration success (prob. of getting success on
iteration i)
    itr_success = zeros(length(Eb_no_db), max_it);

    for eb=1:length(Eb_no_db)
        SNR = Eb_no_db(eb);
        SNRL = 10^(SNR/10);
        sigma = sqrt(1/(2*SNRL*cr));
        success = 0;
        error = 0;

        vn_sum = zeros(1, col);
        for sim=1:nsim
            % Generating random msg
            org = randi([0 1], 1, (m-n)*z);
            % Encoding of msg
            encoded_msg = nr5g_encoder(B, z, org);

            % Puncturing
            encoded_msg = encoded_msg(1:nBlocklen);

            % BPSK modulation
            modulated = 1-2.*encoded_msg;
            % Adding noise
            noise = sigma*randn(1, nBlocklen);
            recevied_sig = modulated+noise;
```

```
            % Demodulation
            recevied = (recevied_sig<0);
            % prev to check that decode codeword is similar as previous
iteration?
            prev = recevied;
            % estimated codeword
            c_aprox = zeros(1, col);


            for it=1:max_it
                if it==1
                    % For first iteration every VN will send its value to
connected CNs (Because there is nothing for majority)
                    for i=1:col
                        for j=vn_to_cn{i, 1}
                            % msg to ith VN to jth CN
                            L(j, i) = recevied(1, i);
                        end
                    end
                else
                    for i=1:col
                        for j=vn_to_cn{i, 1}
                            % subtracting the value sent by the jth cn to
avoid positive feedback loop
                            total = vn_sum(1, i)-L(j, i);
                            % msg to ith VN to jth CN
                            L(j, i) = total>(length(vn_to_cn{i, 1})/2);
                        end
                    end
                end

                for i=1:row
                    xr_val = 0;
                    for j=cn_to_vn{i, 1}
                        xr_val = mod((xr_val+L(i, j)), 2);
                    end

                    for j=cn_to_vn{i, 1}
                        L(i, j) = mod((xr_val+L(i, j)), 2);
                    end
                end

                % If count of 1s are >(degree of VN+1) then decode to bit 1
(+1 because we will count VN's own value also)
                for i=1:col
                    sum_1 = recevied(1, i);
                    cor_col = L(:, i);
                    sum_1 = sum_1 + sum(cor_col);
                    vn_sum(1, i) = sum_1;
```

```
                    c_aprox(1, i) = sum_1>((length(vn_to_cn{i, 1})+1)/2);
                end

                % check if its original msg
                check = 1;
                for i=1:infob
                    if(c_aprox(i) ~= org(i))
                        check = 0;
                        break
                    end
                end
                if check==1
                    success = success+1;
                    % if yes then we got the success
                    % And if we get success in this iteration we will also
get success in remaining iterations
                    for j=it:max_it
                        itr_success(eb, j) = itr_success(eb, j)+1;
                    end
                    break;
                end

                % check if decoded codeword is same as previous iteration
                check2 = 1;
                for i=1:col
                    if c_aprox(1, i) ~= prev(1, i)
                        check2 = 0;
                        break;
                    end
                end
                % if yes then break
                if check2==1
                    break;
                end

                % set prev to decoded codeword
                prev = c_aprox;
            end
            % Count for error bits
            for i=1:col
                if c_aprox(1, i)~=encoded_msg(1, i)
                    error = error+1;
                end
            end
        end
        % calculation for decoding error for coderate coderate(rr) and SNR
Eb_no_db(eb)
        decoding_error(rr, eb) = (nsim-success)/nsim;
        % calculation for bit error for coderate coderate(rr) and SNR
Eb_no_db(eb)
```

```
        bit_error(rr, eb) = error/(nsim*col);

        % we want to plot in logarithmic scale so if bit_error is 0 then
log(0) will -inf(which will be ignored in graph) so make it so small value
        if bit_error(rr, eb)==0
            bit_error(rr, eb) = 1e-305;
        end
    end
    % disp(decoding_error(rr, :));

    % Iteration success probability (Performace graph)
    figure;
    for i=1:length(Eb_no_db)
        plot(iterations,itr_success(i, :)./nsim,'Color',colors(i,:));
        xlabel("Iteration Number");
        ylabel("Success Probability");
        title(['Iteration Success Probability for hard decision decoding,
Coderate = ', num2str(cr)]);

legend('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');

        grid on;
        hold on;
    end
end


% Performance Graphs
% Decoding error probability
for i=1:length(coderate)
    figure;
    plot(Eb_no_db, decoding_error(i, :), 'LineWidth', 2);
    xlabel("Eb/No (dB)");
    ylabel("Decoding error probability");
    title(['Hard Decision Decoding Error Probability, Coderate = ',
num2str(coderate(i))]);
    grid on;
end


% Bit error probability with normalization comparison
for j=1:length(coderate)
    figure;
    %shannon
    r = coderate(j);
    N = 512;
    EbNo = 10.^(Eb_no_db/10);
```

```
    PN_e = zeros(size(EbNo));
    log2e = log2(exp(1));

    for i = 1:length(EbNo)
        P = r * EbNo(i);
        C = log2(1 + P);
        V = (log2e)^2 * (P * (P + 2)) / (2 * (P + 1)^2);
        NA_term = sqrt(N / V) * (C - r + log2(N)/(2*N));
        PN_e(i) = qfunc(NA_term);
    end


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

    semilogy(Eb_no_db, PN_e, 'r-', 'LineWidth', 2);
    hold on;
    xline(shannonLimit_dB, '--b');
    hold on;
    semilogy(Eb_no_db, bit_error(j, :), 'b-', 'LineWidth', 2);
    legend('Normal Approximation', 'Shannon Limit', 'Simulation');
    grid on;
    hold on;
    xlabel("Eb/No (dB)");
    ylabel("Bit error probability");
    title(['Hard Decision Bit Error Probability, Coderate = ',
num2str(coderate(j))]);
    grid on;

    ylim([1e-30, 100000]);  % 100000, so that legend don't cover the actual
graph (So that graph is visible)
    xlim([shannonLimit_dB-0.2, 10]);
end

% Decoding error probability comparison for all coderates
figure;
for i=1:length(coderate)
    plot(Eb_no_db, decoding_error(i, :), 'LineWidth', 2);
    xlabel("Eb/No (dB)");
    ylabel("Decoding error probability");
    title('Hard Decision Decoding Error Probability Comparison');
    legend('Coderate = 1/4', 'Coderate = 1/3', 'Coderate = 1/2', 'Coderate =
3/5');
    grid on;
    hold on;
end


% Bit error probability comparison for all coderates
figure;
for i=1:length(coderate)
```

```
    plot(Eb_no_db, bit_error(i, :), 'LineWidth', 2);
    xlabel("Eb/No (dB)");
    ylabel("Bit error probability");
    title('Hard Decision Bit Error Probability Comparison');
    legend('Coderate = 1/4', 'Coderate = 1/3', 'Coderate = 1/2', 'Coderate =
3/5');
    grid on;
    hold on;
end
```

Soft Decision Decoding Simulation :-

```
baseGraph = 'NR_2_6_52';
% Coderates
coderate = [1/4 1/3 1/2 3/5];
Eb_no_db = 0:0.5:10;
colors = lines(length(Eb_no_db));

% Lifting
[B, Hfull, z] = nrldpc_Hmatrix(baseGraph);

% for storing the outputs
decoding_error = zeros(length(coderate), length(Eb_no_db));
bit_error = zeros(length(coderate), length(Eb_no_db));

% Number of simulations
nsim = 1000;

% Maximum iterations
max_it = 20;
iterations = 1:1:max_it;

for rr=1:length(coderate)
    [n, m] = size(B);
    cr = coderate(rr);

    % Adjusting H matrix for specific coderate
    totalparity = n*z;
    info = m-n-2;
    needed_blocks = ceil(info/cr)+2;
    nBlocklen = needed_blocks*z;
    needed_p = totalparity - (m*z - nBlocklen);
    total_bits = n*z-m*z+nBlocklen;
    H = Hfull(:, 1:nBlocklen);
    H = H(1:total_bits, :);


    [row, col] = size(H);
```

```
    infob = col-row;

    % Mapping for which check nodes connectd to a VNi
    vn_to_cn = cell(col, 1);
    % Mapping for which variable nodes connectd to a CNi
    cn_to_vn = cell(row, 1);
    % VN->CN and CN->VN msg storing matrix L
    L = zeros(row, col);

    % Mapping
    for i=1:col
        for j=1:row
            if H(j, i)==1
                vn_to_cn{i, 1} = [vn_to_cn{i, 1} j];
            end
        end
    end
    for i=1:row
        for j=1:col
            if H(i, j)==1
                cn_to_vn{i, 1} = [cn_to_vn{i, 1} j];
            end
        end
    end


    % To store output for iteration success (prob. of getting success on
iteration i)
    itr_success = zeros(length(Eb_no_db), max_it);

    for eb=1:length(Eb_no_db)
        SNR = Eb_no_db(eb);
        SNRL = 10^(SNR/10);
        sigma = sqrt(1/(2*SNRL*cr));
        success = 0;
        error = 0;

        vn_sum = zeros(1, col);
        for sim=1:nsim
            % Generating random msg
            org = randi([0 1], 1, (m-n)*z);
            % Encoding of msg
            encoded_msg = nr5g_encoder(B, z, org);

            % Puncturing
            encoded_msg = encoded_msg(1:nBlocklen);

            % BPSK modulation
            modulated = 1-2.*encoded_msg;
            % Adding noise
```

```
            noise = sigma*randn(1, nBlocklen);
            recevied_sig = modulated+noise;

            % Demodulation
            recevied = (recevied_sig<0);
            % prev to check that decode codeword is similar as previous
iteration?
            prev = recevied;
            % estimated codeword
            c_aprox = zeros(1, col);

            for it=1:max_it
                if it==1
                    % For first iteration every VN will send its value to
connected CNs
                    for i=1:col
                        for j=vn_to_cn{i, 1}
                            % msg to ith VN to jth CN
                            L(j, i) = recevied_sig(1, i);
                        end
                    end
                else
                    for i=1:col
                        for j=vn_to_cn{i, 1}
                            % subtracting the value sent by the jth cn to
avoid positive feedback loop
                            total = vn_sum(1, i)-L(j, i);
                            % msg to ith VN to jth CN
                            L(j, i) = total;
                        end
                    end
                end

                % Min-sum algo
                for i=1:row
                    mini1 = 1e10;
                    mini2 = 1e10;
                    ind = -1;
                    total_sign = 1;

                    for j=cn_to_vn{i, 1}
                        val = abs(L(i, j));
                        if val<=mini1
                            mini2 = mini1;
                            mini1 = val;
                            ind = j;
                        elseif val<=mini2
                            mini2 = val;
                        end
```

```
                    if(L(i, j)~=0)
                        if(L(i, j)<0)
                            total_sign = total_sign*-1;
                        end
                    end
                end

                for j=cn_to_vn{i, 1}
                    if j~=ind
                        L(i, j) = total_sign*sign(L(i, j))*mini1;
                    else
                        L(i, j) = total_sign*sign(L(i, j))*mini2;
                    end
                end

            end

            % Add values sent by all CNs
            for i=1:col
                sum_1 = recevied_sig(1, i);
                tm = L(:, i);
                sum_1 = sum_1+sum(tm);
                vn_sum(1, i) = sum_1;
            end

            % Estimate codeword
            c_aprox = (vn_sum<0);

            %check if its the original msg
            check = 1;
            for i=1:infob
                if(c_aprox(i) ~= org(i))
                    check = 0;
                    break
                end
            end
            if check==1
                success = success+1;
                % if yes then we got the success
                % And if we get success in this iteration we will also
get success in remaining iterations
                for j=it:max_it
                    itr_success(eb, j) = itr_success(eb, j)+1;
                end
                break;
            end

            % check if decoded codeword is same as previous iteration
            check2 = 1;
            for i=1:col
```

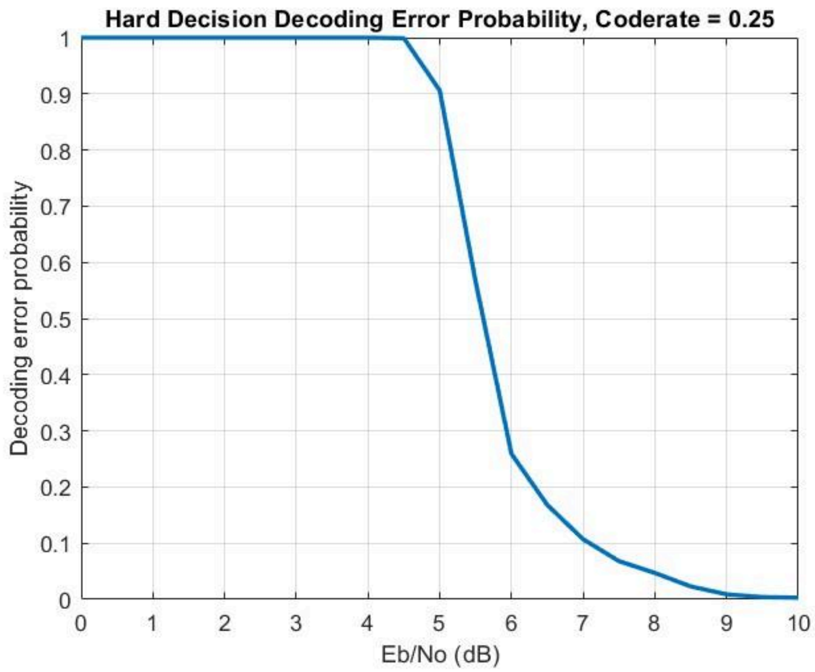
```
                if c_aprox(1, i) ~= prev(1, i)
                    check2 = 0;
                    break;
                end
            end
            % if yes then break
            if check2==1
                break;
            end

            % set prev to decoded codeword
            prev = c_aprox;
        end
        % Count for error bits
        for i=1:col
            if c_aprox(1, i)~=encoded_msg(1, i)
                error = error+1;
            end
        end
    end
    % calculation for decoding error for coderate coderate(rr) and SNR
Eb_no_db(eb)
    decoding_error(rr, eb) = (nsim-success)/nsim;
    % calculation for bit error for coderate coderate(rr) and SNR
Eb_no_db(eb)
    bit_error(rr, eb) = error/(nsim*col);

    % we want to plot in logarthmic scale so if bit_error is 0 then
log(0) will -inf(which will be ignored in graph) so make it so small value
    if bit_error(rr, eb)==0
        bit_error(rr, eb) = 1e-305;
    end
end
% disp(decoding_error);



% Iteration success probability (Performace graph)
figure;
for i=1:length(Eb_no_db)
    plot(iterations,itr_success(i, :)./nsim,'Color',colors(i,:));
    xlabel("Iteration Number");
    ylabel("Success Probability");
    title(['Iteration Success Probability for soft decision decoding,
Coderate = ', num2str(cr)]);

legend('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');

    grid on;
```

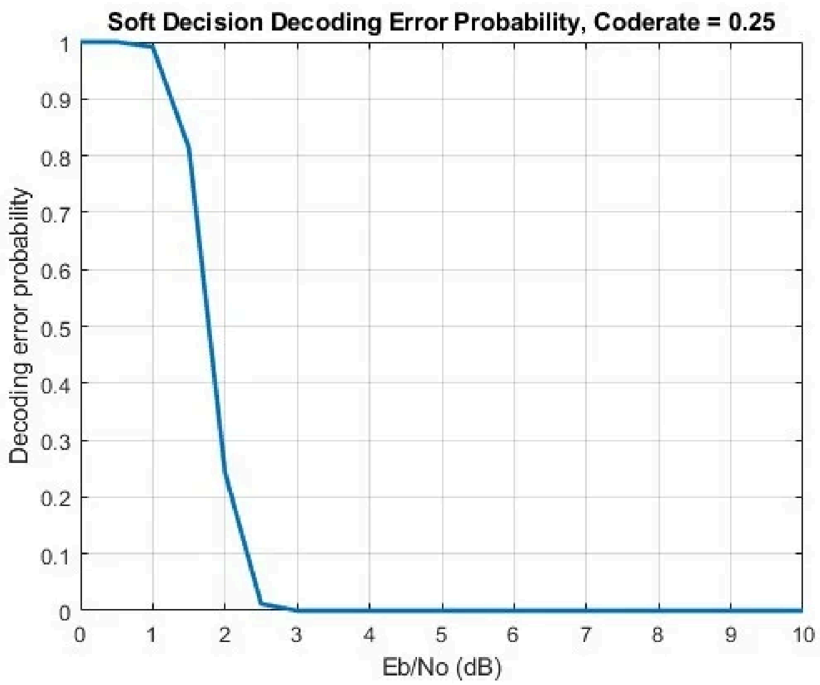
```
        hold on;
    end

end

% Performance Graphs
% Decoding error probability
for i=1:length(coderate)
    figure;
    plot(Eb_no_db, decoding_error(i, :), 'LineWidth', 2);
    xlabel("Eb/No (dB)");
    ylabel("Decoding error probability");
    title(['Soft Decision Decoding Error Probability, Coderate = ',
num2str(coderate(i))]);
    grid on;
end


% Bit error probability with normalization comparison
for j=1:length(coderate)
    figure;
    %shannon

    r = coderate(j);
    N = 512;
    EbNo = 10.^(Eb_no_db/10);
    PN_e = zeros(size(EbNo));
    log2e = log2(exp(1));

    for i = 1:length(EbNo)
        P = r * EbNo(i);
        C = log2(1 + P);
        V = (log2e)^2 * (P * (P + 2)) / (2 * (P + 1)^2);
        NA_term = sqrt(N / V) * (C - r + log2(N)/(2*N));
        PN_e(i) = qfunc(NA_term);
    end
    shannonLimit_dB = 10 * log10((2^r - 1)/r);

    semilogy(Eb_no_db, PN_e, 'r-', 'LineWidth', 2);
    hold on;
    xline(shannonLimit_dB, '--b');
    hold on;
    semilogy(Eb_no_db, bit_error(j, :), 'b-', 'LineWidth', 2);
    legend('Normal Approximation', 'Shannon Limit', 'Simulation');
    grid on;
    hold on;
    xlabel("Eb/No (dB)");
    ylabel("Bit error probability");
    title(['Soft Decision Bit Error Probability, Coderate = ',
num2str(coderate(j))]);
```

```
    grid on;

    ylim([1e-30, 100000]);  % 100000, so that legend don't cover the actual
graph (So that graph is visible)
    xlim([shannonLimit_dB-0.2, 10]);
end


% Decoding error probability comparison for all coderates
figure;
for i=1:length(coderate)
    plot(Eb_no_db, decoding_error(i, :), 'LineWidth', 2);
    xlabel("Eb/No (dB)");
    ylabel("Decoding error probability");
    title('Soft Decision Decoding Error Probability Comparison');
    legend('Coderate = 1/4', 'Coderate = 1/3', 'Coderate = 1/2', 'Coderate =
3/5');
    grid on;
    hold on;
end

% Bit error probability comparison for all coderates
figure;
for i=1:length(coderate)
    plot(Eb_no_db, bit_error(i, :), 'LineWidth', 2);
    xlabel("Eb/No (dB)");
    ylabel("Bit error probability");
    title('Soft Decision Bit Error Probability Comparison');
    legend('Coderate = 1/4', 'Coderate = 1/3', 'Coderate = 1/2', 'Coderate =
3/5');
    grid on;
    hold on;
end
```

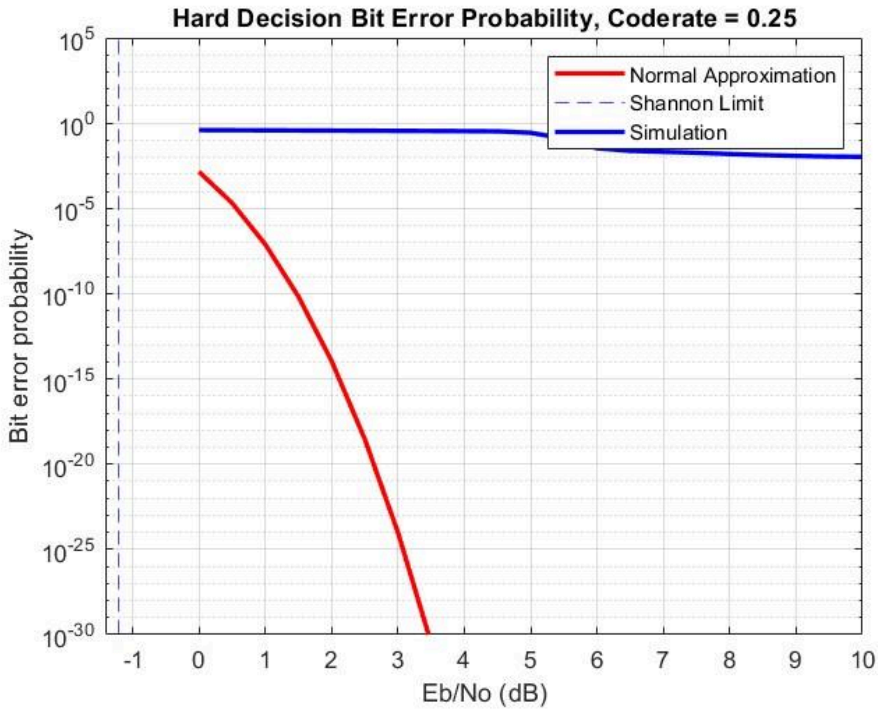
Graph Analysis (For BG2)

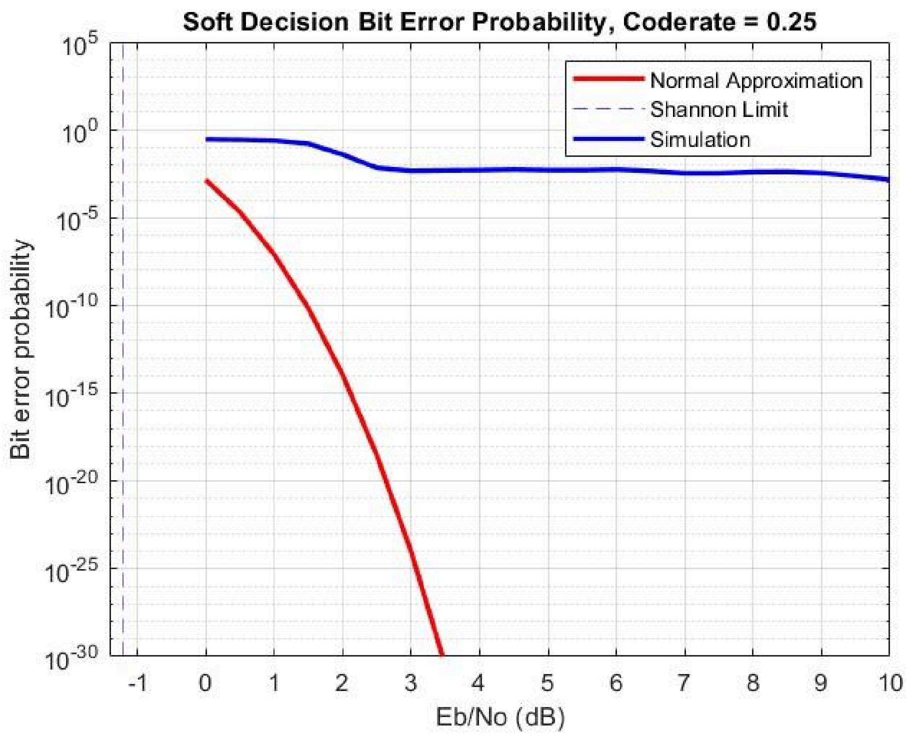
- **Graphs for Coderate ¼**
  - **Decoding Error Probability**



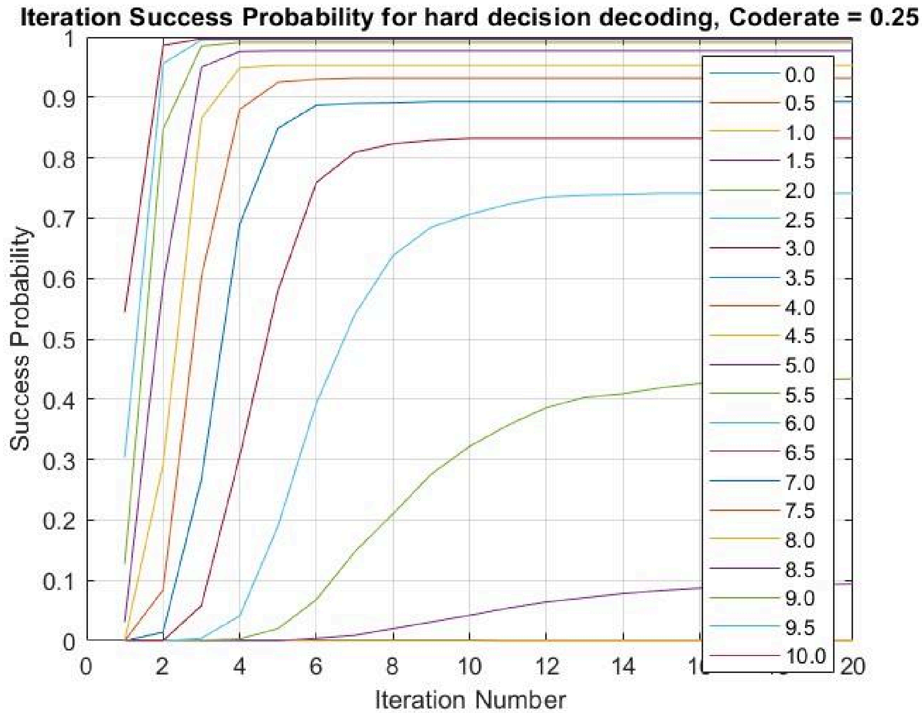


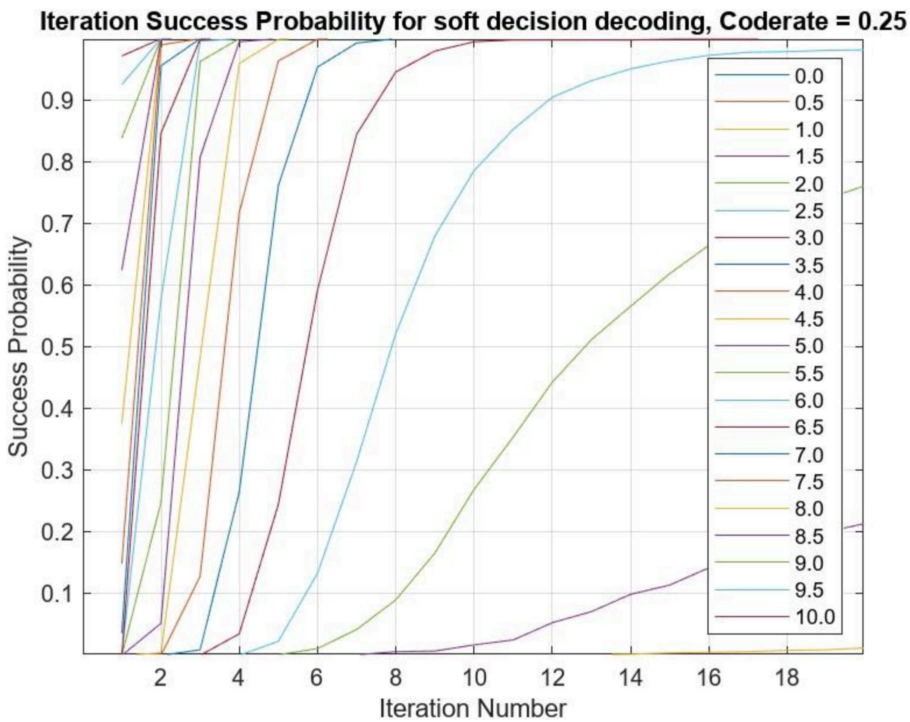
- **Bit Error Probability**



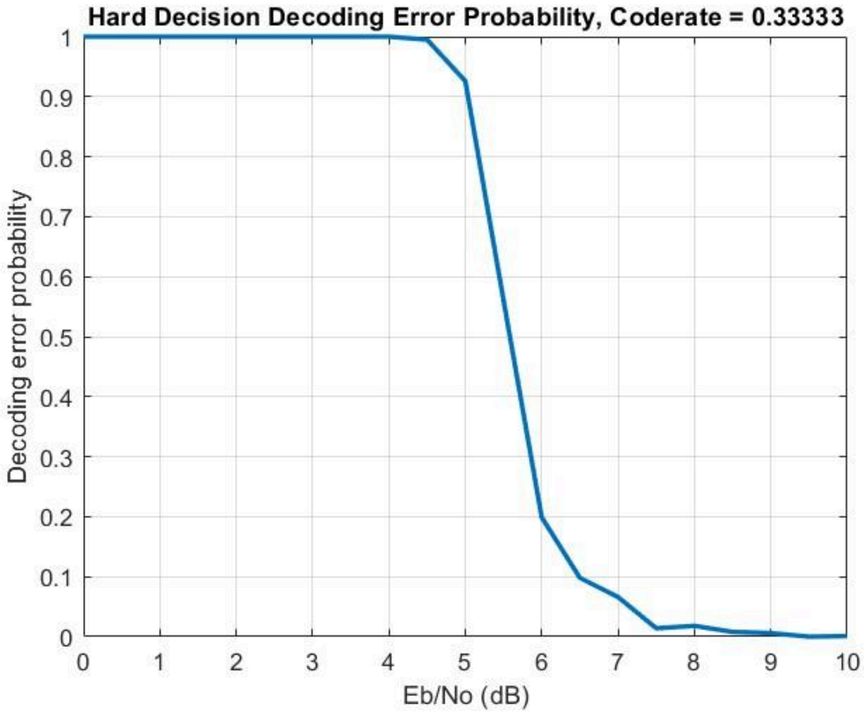


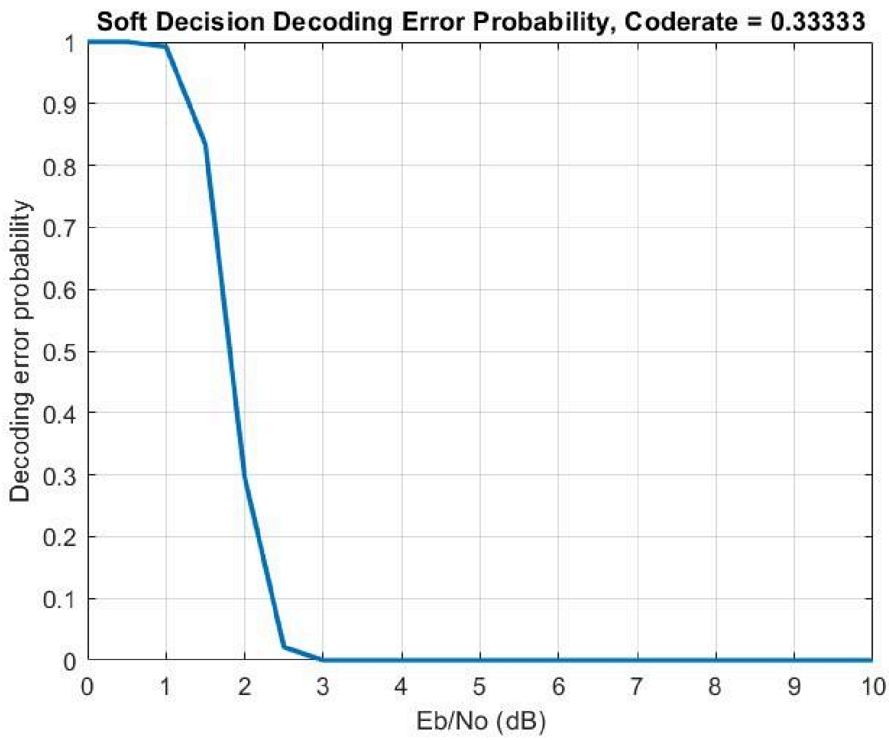
○ **Iteration Success Probability**



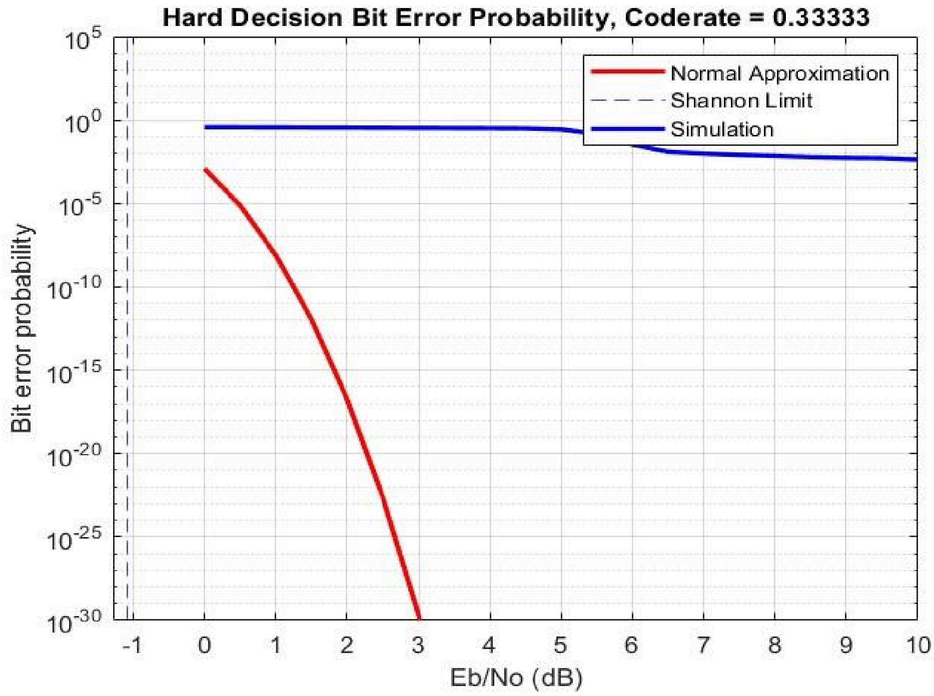


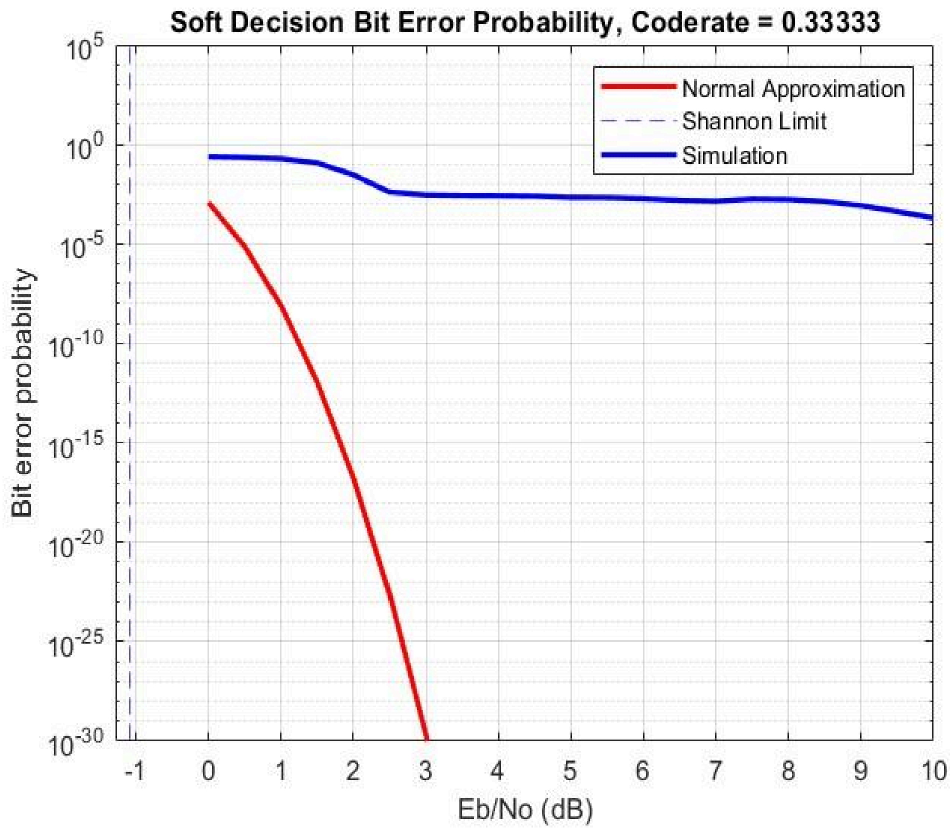
- **Graphs for Coderate 1/3**
    - **Decoding Error Probability**



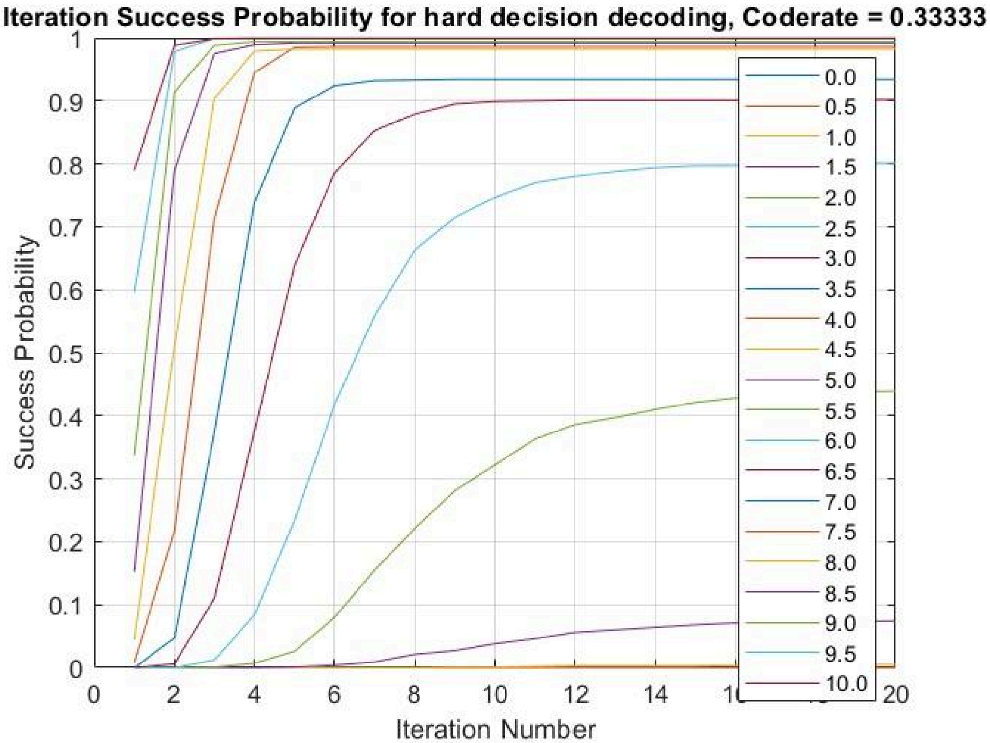


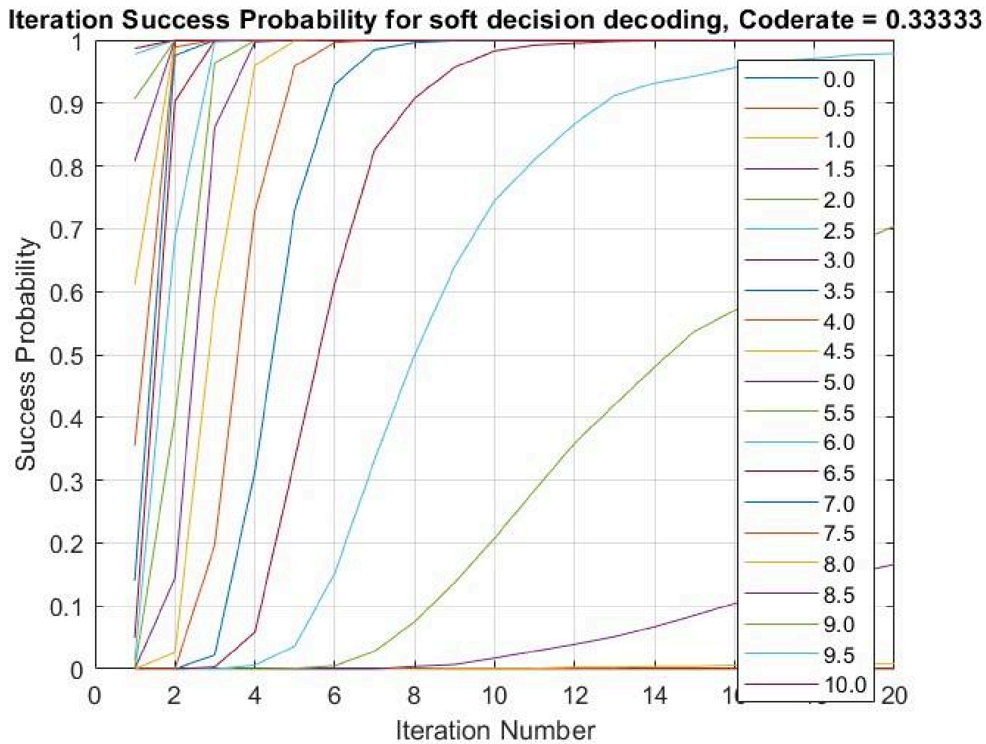
- **Bit Error Probability**



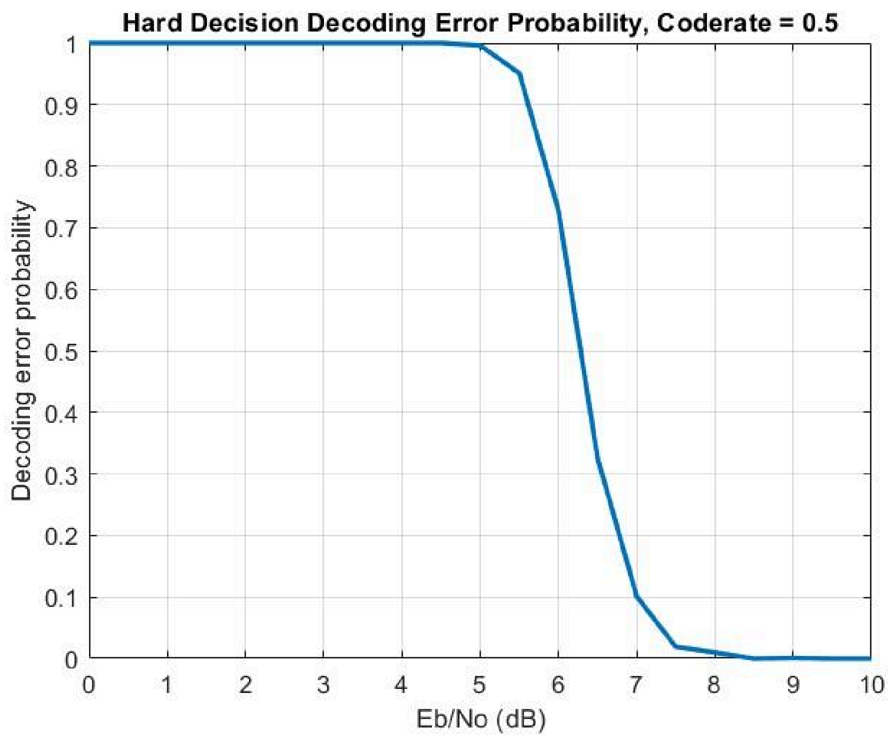


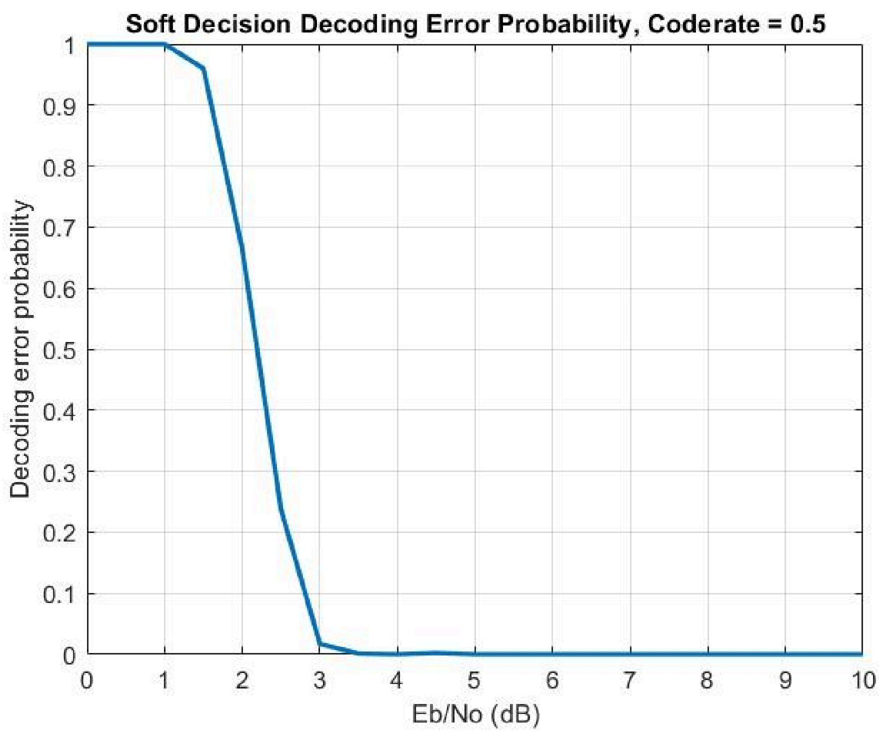
○ **Iteration Success Probability**



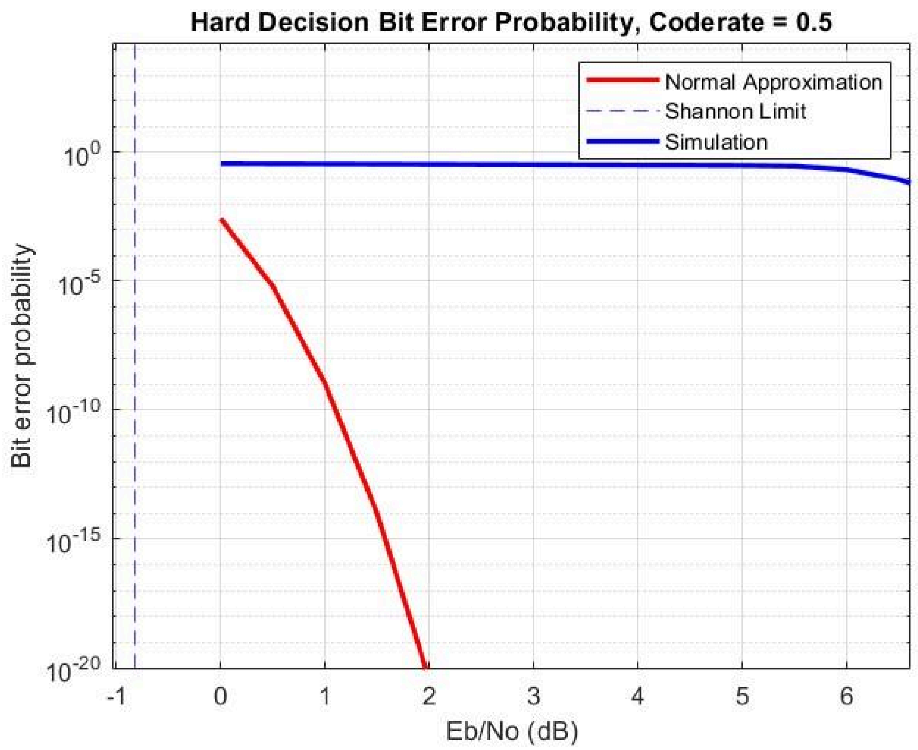


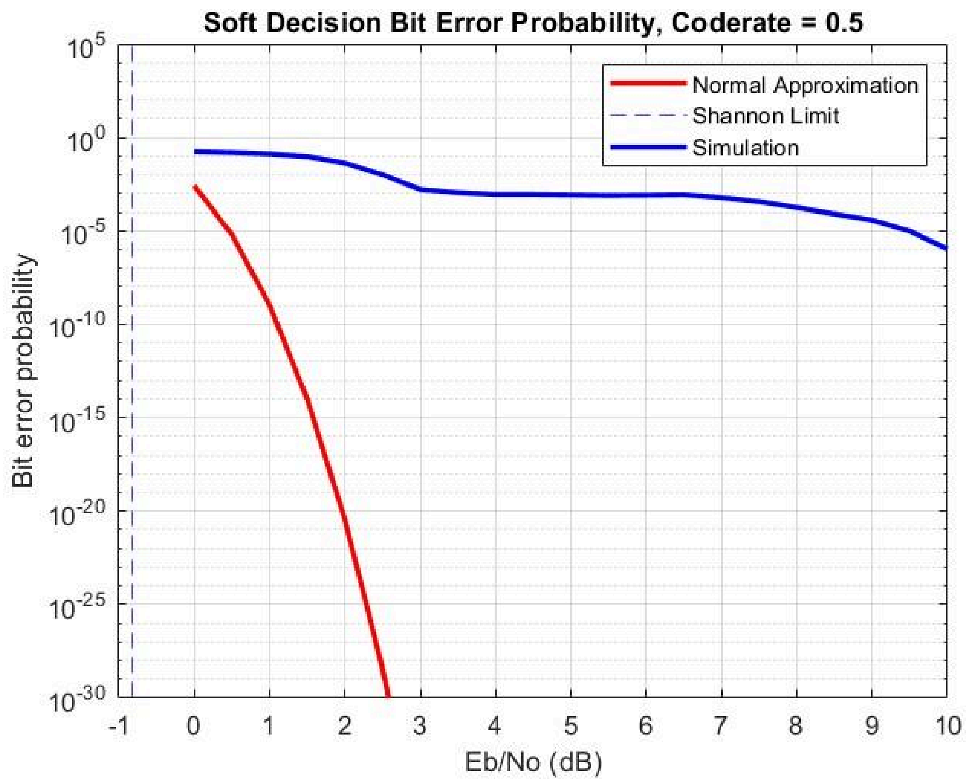
- **Graphs for Coderate 1/2**
  - **Decoding Error Probability**



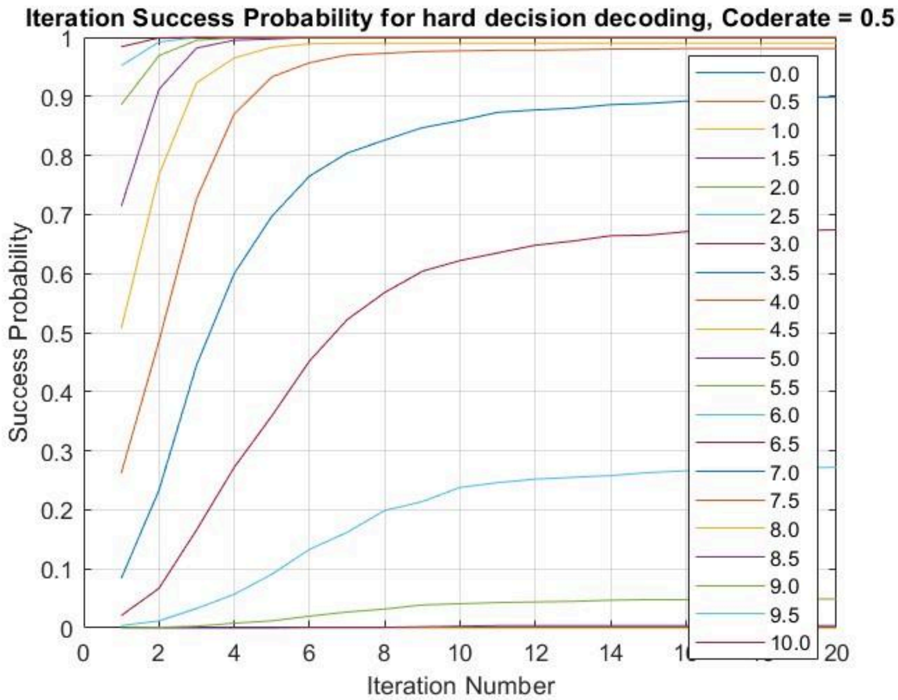


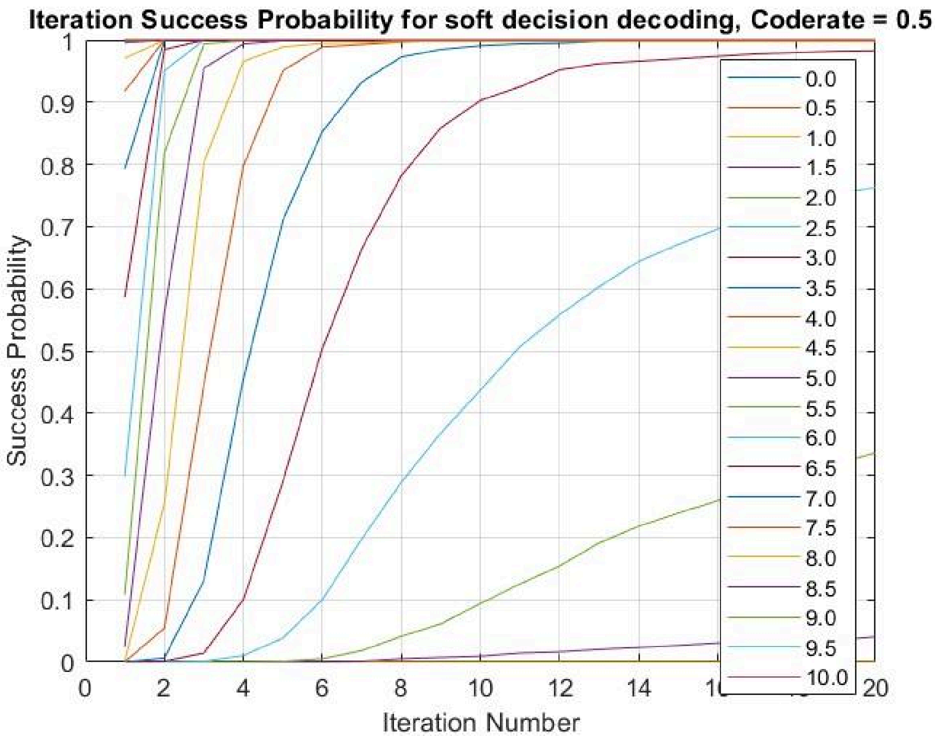
○ **Bit Error Probability**



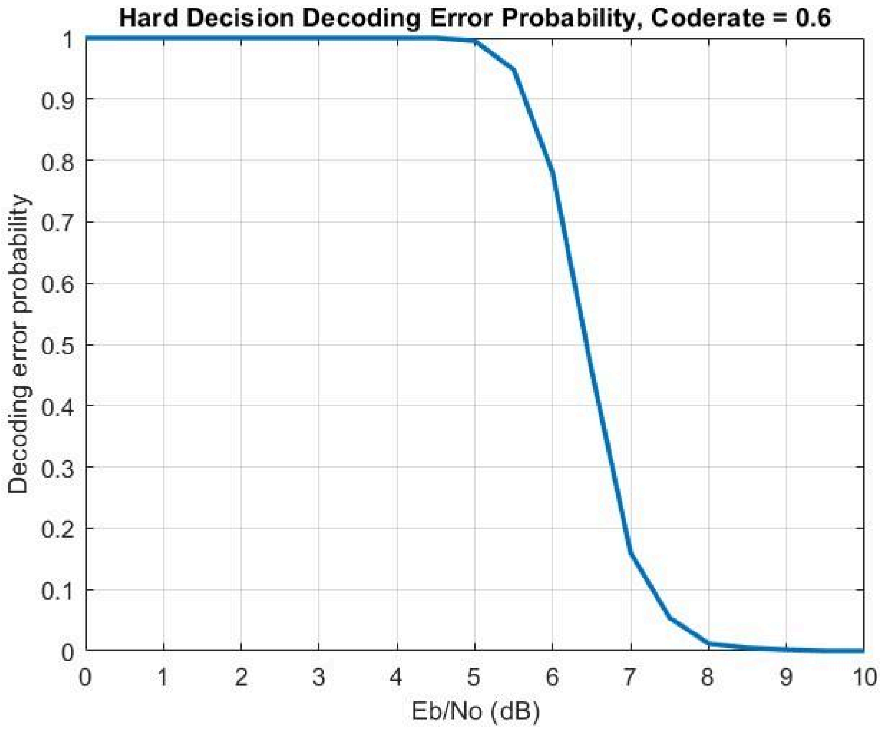


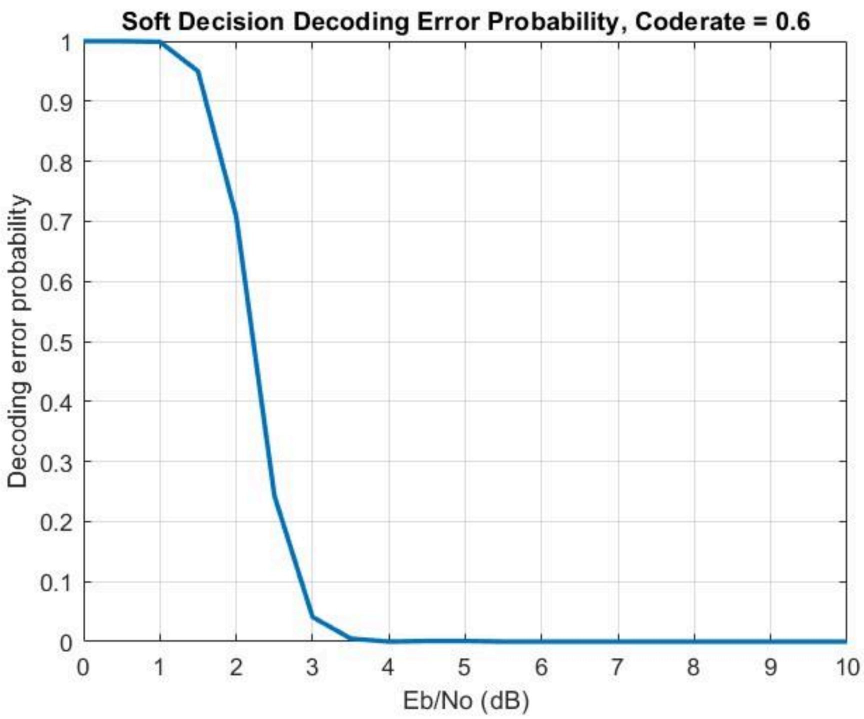
- Iteration Success Probability



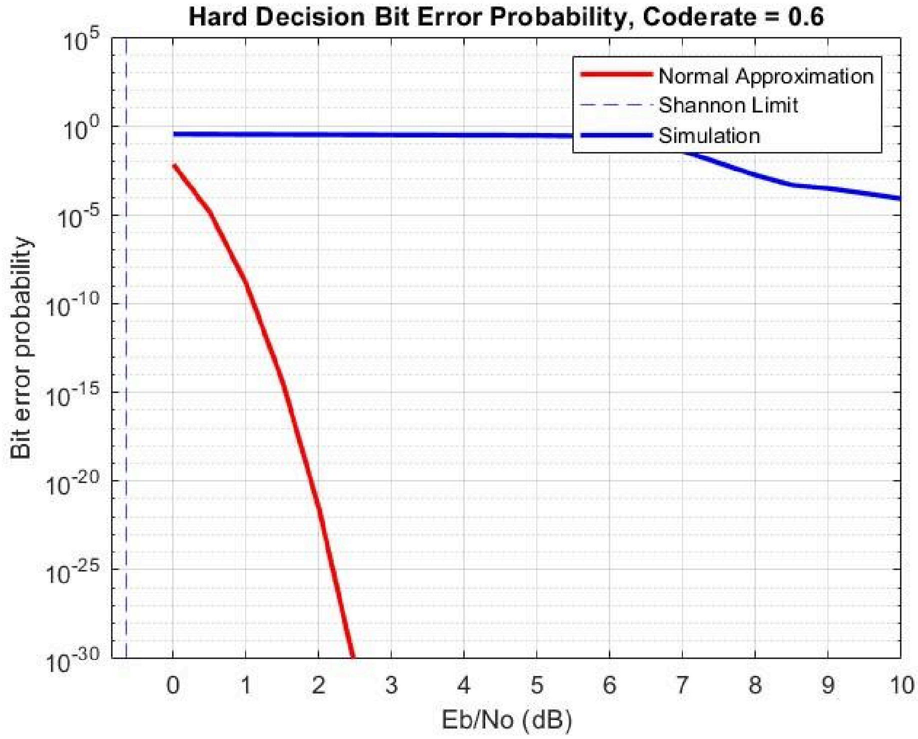


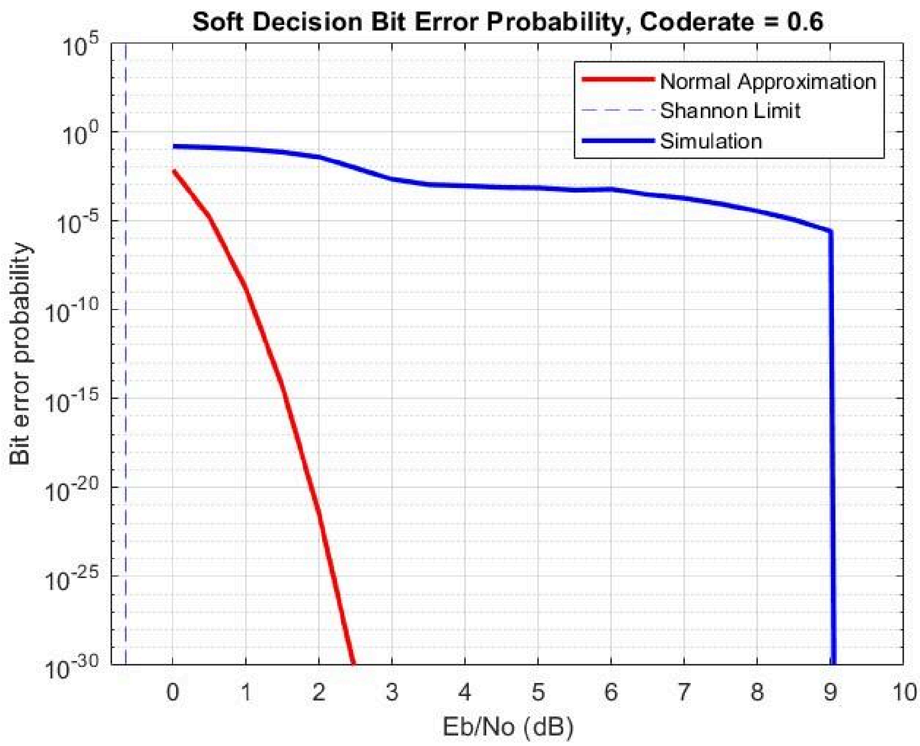
- **Graphs for Coderate 3/5**
    - **Decoding Error Probability**



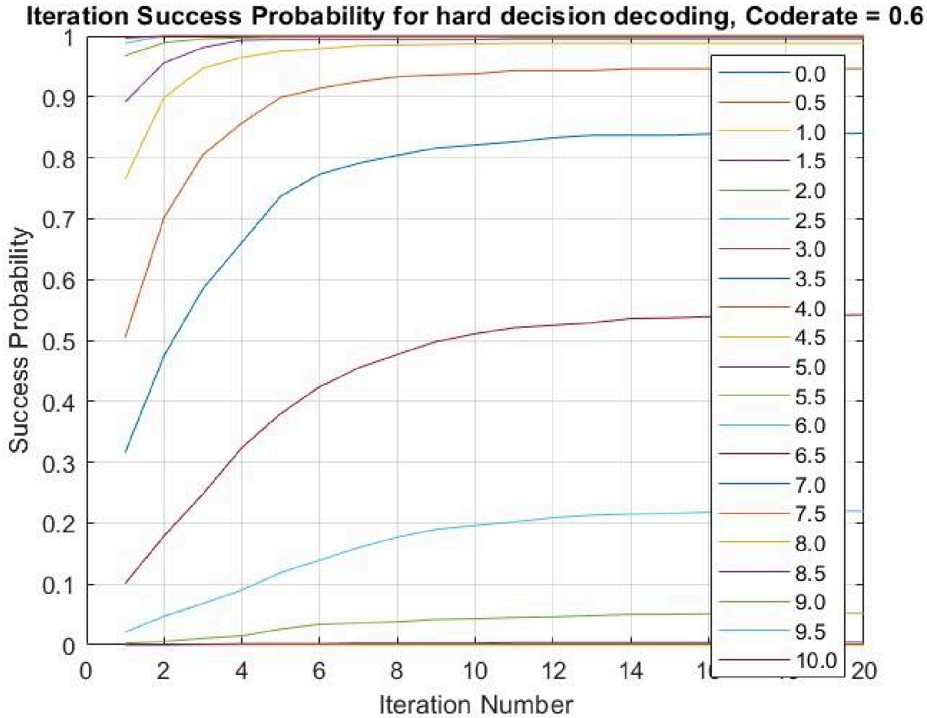


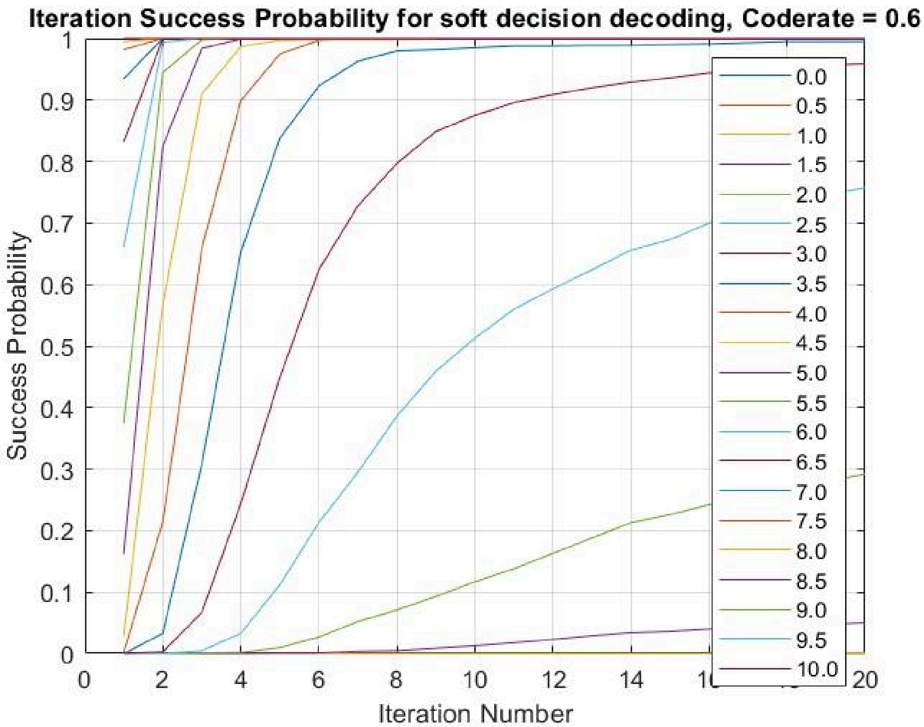
- **Bit Error Probability**



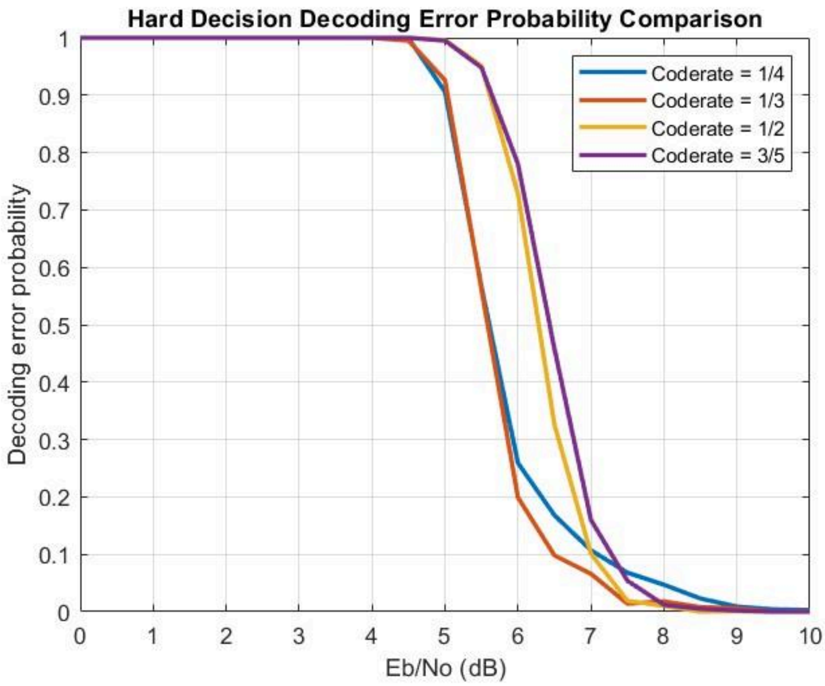


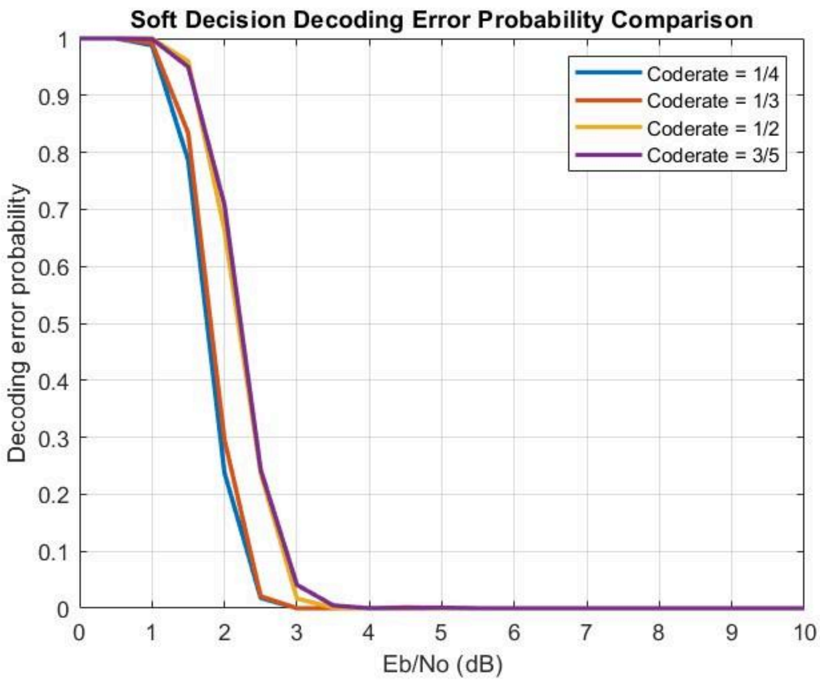
○ Iteration Success Probability



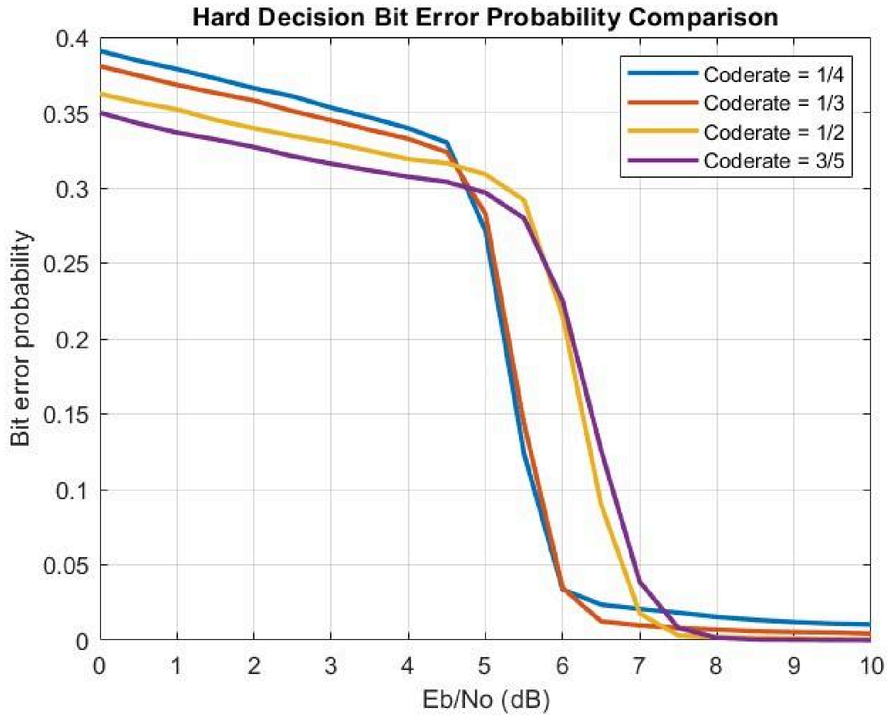


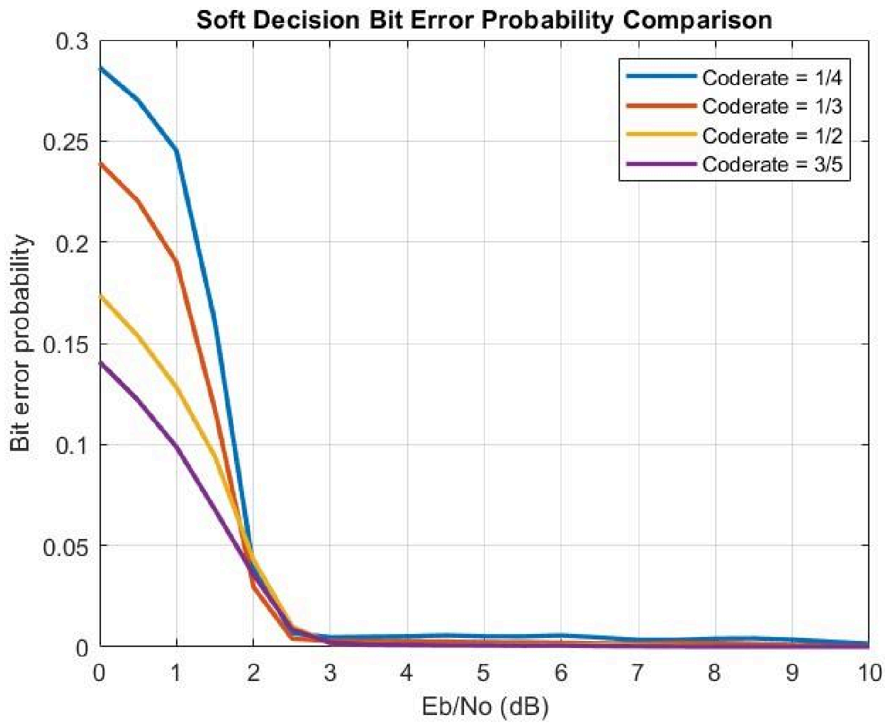
- **Comparison Graphs**
  - **Decoding Error Probability**





- ## Bit Error Probability





Derivations & Proofs :-

⇒ Log-Likelihood Ratio (LLR):

$$P(c_1=0|r_i) = \frac{f(r_i|c_1=0)\,P(c_1=0)}{f(r_i)}$$

$$P(c_1=1|r_i) = \frac{f(r_i|c_1=1)\,P(c_1=1)}{f(r_i)}$$

$$\therefore \quad LR = \frac{P(c_1=0|r_i)}{P(c_1=1|r_i)}$$

$$= \frac{f(r_i|c_1=0)}{f(r_i|c_2=1)}$$

$$= \frac{\frac{1}{\sqrt{2\pi\sigma^2}}\,e^{-\frac{1}{2}\left(\frac{r_i-\mu_1}{\sigma}\right)^2}}{\frac{1}{\sqrt{2\pi\sigma^2}}\,e^{-\frac{1}{2}\left(\frac{r_i-\mu_2}{\sigma}\right)^2}}$$

Now when $c_1=0 \Rightarrow \mu_1 = 1$ ($\because$ 0 is mapped with 1)

$r = s + n$

when $c_1 = 0 \Rightarrow s = 1$

$\therefore E[r] = E[1] + E[n]$

$\therefore E[r] = 1 \quad (\because E[n] = 0)$

Now, when $c_1 = 1 \Rightarrow \mu_2 = -1$ ($\because$ 1 is mapped with -1)

$r = s + n$

when $c_1 = 1 \Rightarrow s = -1$

$\therefore E[r] = E[-1] + E[n]$

$\therefore E[r] = -1 \quad (\because E[n] = 0)$

$$\therefore \quad LR = e^{-\frac{1}{2}\left(\frac{r_i-1}{\sigma}\right)^2 + \frac{1}{2}\left(\frac{r_i+1}{\sigma}\right)^2}$$

$$= e^{2r_i/\sigma^2}$$

$\therefore$ Log-Likelihood Ratio (LLR) $= \log e^{2r_i/\sigma^2}$

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

we use LLR is soft decision decoding, specifically in min-sum algo

⇒ In min-sum algo we only care about the sign and the relative magnitude of LLRs so $2/\sigma^2$ is +ve so if we ignore it sign and relative magnitudes of LLRs will remain unchanged and calculation will be easy

$\Rightarrow \therefore \quad LLR \approx r_i$

$\Rightarrow$ LLR For repetition codes ($n=3$)

$$\therefore \; LR = \frac{P(c_1=0 \mid r_1 r_2 r_3)}{P(c_1=1 \mid r_1 r_2 r_3)}$$

$$= \frac{f(r_1 r_2 r_3 \mid c_1=0)}{f(r_1 r_2 r_3 \mid c_1=1)}$$

$$= \frac{f(r_1 \mid c_1=0)}{f(r_1 \mid c_1=1)} \times \frac{f(r_2 \mid c_1=0)}{f(r_2 \mid c_1=1)} \times \frac{f(r_3 \mid c_1=0)}{f(r_3 \mid c_1=1)} \quad (\because r_i\text{'s are independent})$$

$$= e^{2r_1/\sigma^2} \cdot e^{2r_2/\sigma^2} \cdot e^{2r_3/\sigma^2}$$

$$\therefore \; LLR = \log e^{\frac{2}{\sigma^2}(r_1+r_2+r_3)}$$

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

$$\therefore \; LLR \simeq r_1+r_2+r_3$$

$\Rightarrow$ This can be generalized for n bits repetition code

$$\therefore \; LLR \simeq r_1+r_2+r_3+\dots+r_n$$

$\Rightarrow$ ~~2:3~~ Output LLR for 2:3 SPC.

$$m_1, m_2 \xrightarrow[\text{Encode}]{} c_1, c_2, c_3 \xrightarrow[\text{BPSK}]{} S \xrightarrow[\text{AWGN}]{} S+n \xrightarrow[\text{SISO Decode}]{} L_1, L_2, L_3$$

$$l_{ext,1} = \log\left(\frac{P(c_1=1 \mid r_1)}{P(c_1=0 \mid r_1)}\right), \quad l_2 = \log\left(\frac{P(c_2=1 \mid r_2)}{P(c_2=0 \mid r_2)}\right), \quad l_3 = \log\left(\frac{P(c_3=1 \mid r_3)}{P(c_3=0 \mid r_3)}\right)$$

$$\therefore \; = \log\left(\frac{P_1}{1-P_1}\right) \qquad = \log\left(\frac{P_2}{1-P_2}\right) \qquad = \log\left(\frac{P_3}{1-P_3}\right)$$

$$P_1 = P_2 P_3 + (1-P_2)(1-P_3), \quad 1-P_1 = P_2(1-P_3) + (1-P_2)P_3$$

$$\therefore \; P_1 - (1-P_1) = (P_2 - (1-P_2))(P_3 - (1-P_3))$$

$$\therefore \; \frac{P_1 - (1-P_1)}{P_1 + (1-P_1)} = \frac{P_2 - (1-P_2)}{P_2 + (1-P_2)} \cdot \frac{P_3 - (1-P_3)}{P_3 + (1-P_3)}$$

$$\therefore \; \frac{1 - \frac{1-P_1}{P_1}}{1 + \frac{1-P_1}{P_1}} = \frac{1 - \frac{1-P_2}{P_2}}{1 + \frac{1-P_2}{P_2}} \cdot \frac{1 - \frac{1-P_3}{P_3}}{1 + \frac{1-P_3}{P_3}}$$

$$\therefore \; \frac{1 - e^{-l_{ext,1}}}{1 + e^{-l_{ext,2}}} = \frac{1 - e^{-l_2}}{1 + e^{-l_2}} \cdot \frac{1 - e^{-l_3}}{1 + e^{-l_3}}$$

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

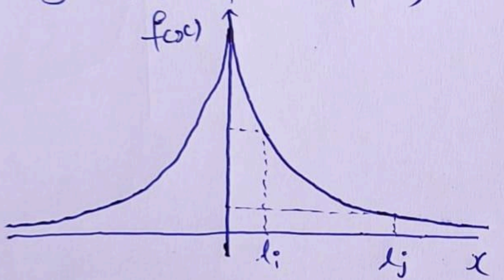
$\therefore \log\left|\tanh\left(\frac{|l_{ext,1}|}{2}\right)\right| = \log\left|\tanh\left(\frac{l_2}{2}\right)\right| + \log\left|\tanh\left(\frac{l_3}{2}\right)\right|$

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

$\Rightarrow f(x) = f^{-1}(x)$

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

$\Rightarrow$ graph for $f(x)$



according to the graph $\min(l_2, l_3)$ will contribut more in $f(l_2) + f(l_3)$

$\Rightarrow$ we can write $f(l_2) + f(l_3) \cong f(\min(l_2, l_3))$

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

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

$\Rightarrow$ This is called min-sum approximation.