```
clear;
clc;

baseGraph = 'NR_2_6_52';
coderate = [1/4 1/3 1/2 3/5];
Eb_no_db = 0:0.5:10;

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

Nsim = 500;

max_itr = 20;
iterations = 1:max_itr;
```

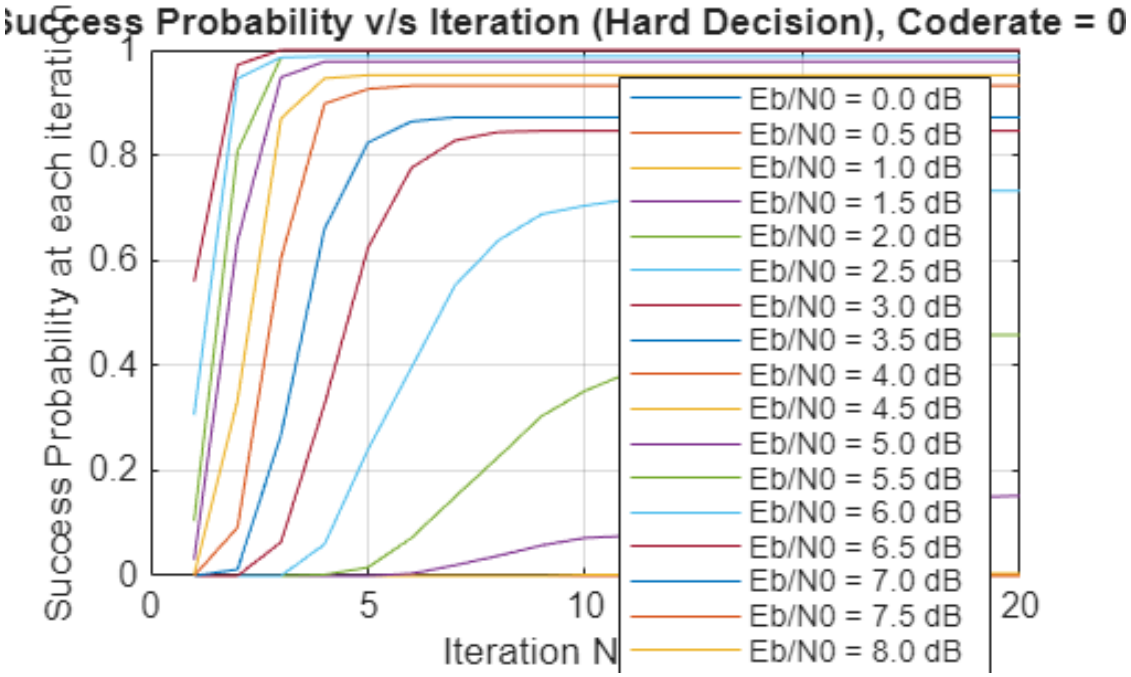
```
bit_error_all = zeros(4, 20);
decoding_error_all = zeros(4, 20);

for index = 1:length(coderate)

    [n, m] = size(B);
    codeword = coderate(index);
    totalparity = n*z;
    q = m-n-2;
    nbRM = ceil(q/codeword)+2;
    nBlocklen = nbRM*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;
    [vn_to_cn_conn, cn_to_vn_conn, VN_to_CN_Message, CN_to_VN_Message] =
Connection(H);

    decoding_error = zeros(1, length(Eb_no_db));
    bit_error = zeros(1, length(Eb_no_db));

    iter = 1;
    figure;

    for ind = 1:length(Eb_no_db)

        SNR = Eb_no_db(ind);
        SNRL = 10^(SNR/10);
        sigma = sqrt(1/(2*SNRL*codeword));

        success = 0;
        error1 = 0;
```

```
        itr_success = zeros(1, max_itr);
        vn_sum = zeros(1, col);

        for sim = 1:Nsim

            b = randi([0 1], 1, (m-n)*z);
            c = nrldpc_encode(B, z, b);
            c = c(1:nBlocklen);

            s = 1 - 2.*c;
            noise = sigma * randn(1, nBlocklen);
            recevied_sig = s + noise;

            r = (recevied_sig < 0);
            prev = r;
            c_hat = zeros(1, col);

            for itr = 1:max_itr

                if itr == 1
                    for i = 1:col
                        for j = vn_to_cn_conn{i}
                            VN_to_CN_Message(j, i) = r(1, i);
                        end
                    end
                else
                    for i = 1:col
                        for j = vn_to_cn_conn{i}
                            total = vn_sum(1, i) - CN_to_VN_Message(j, i);
                            VN_to_CN_Message(j, i) = total >
(length(vn_to_cn_conn{i})/2);
                        end
                    end
                end

                for i = 1:row
                    value = 0;
                    for j = cn_to_vn_conn{i}
                        value = mod((value + VN_to_CN_Message(i, j)), 2);
                    end
                    for j = cn_to_vn_conn{i}
                        CN_to_VN_Message(i, j) = mod((value + VN_to_CN_Message(i,
j)), 2);
                    end
                end

                for i = 1:col
                    sum1s = r(1, i);
                    cor_col = CN_to_VN_Message(:, i);
                    sum1s = sum1s + sum(cor_col);
                    vn_sum(1, i) = sum1s;
```

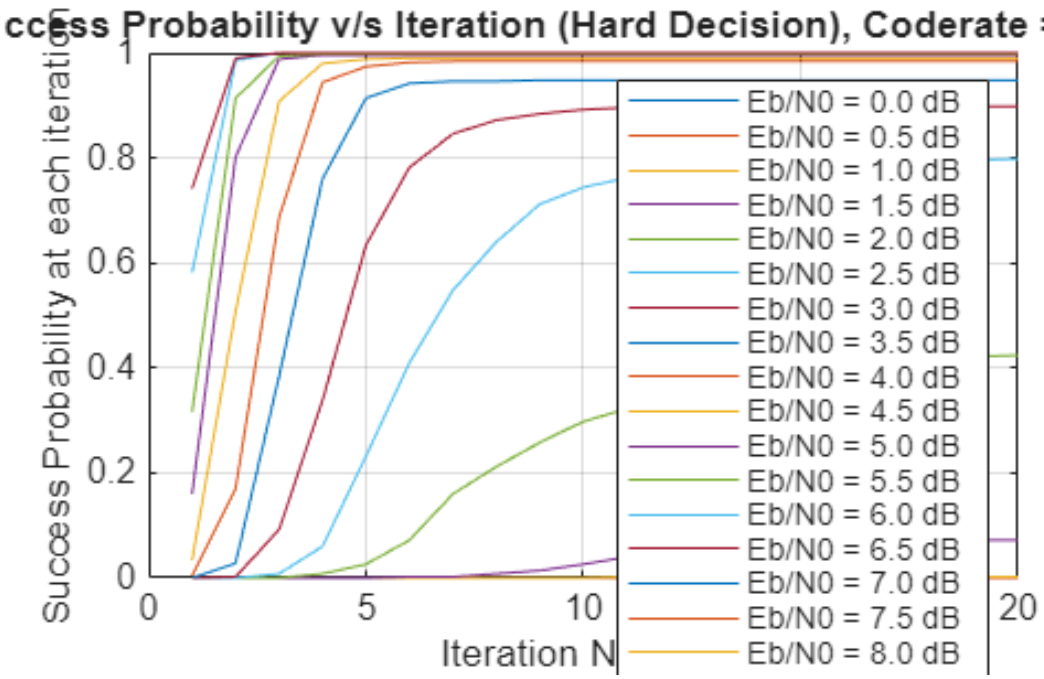
```
                    c_hat(1, i) = sum1s > ((length(vn_to_cn_conn{i}) + 1)/2);
                end

                if sum(xor(c_hat(1:infob), b)) == 0
                    success = success + 1;
                    for j = itr:max_itr
                        itr_success(1, j) = itr_success(1, j) + 1;
                    end
                    break;
                end

                if sum(xor(prev, c_hat)) == 0
                    break;
                end

                prev = c_hat;
            end

            error1 = error1 + sum(c_hat ~= c);
        end

        plot(iterations, itr_success ./ Nsim, 'DisplayName', sprintf('Eb/N0 = %.1f
dB', Eb_no_db(ind)));
        xlabel("Iteration Number");
        ylabel("Success Probability at each iteration");
        title(['Success Probability v/s Iteration (Hard Decision), Coderate = ',
num2str(codeword)]);
        legend('show');
        grid on;
        hold on;

        decoding_error(1, iter) = (Nsim - success) / Nsim;
        bit_error(1, iter) = error1 / (Nsim * col);
        fprintf('coderate: %.2f | BER: %.5f | DER: %.5f\n', coderate(index),
bit_error(1, iter), decoding_error(1, iter));
        iter = iter + 1;
    end

    % Store results in 2D matrices
    bit_error_all(index, 1:length(Eb_no_db)) = bit_error;
    decoding_error_all(index, 1:length(Eb_no_db)) = decoding_error;

    hold off;
end
```

```
coderate: 0.25 | BER: 0.39029 | DER: 1.00000
coderate: 0.25 | BER: 0.38406 | DER: 1.00000
coderate: 0.25 | BER: 0.38006 | DER: 1.00000
coderate: 0.25 | BER: 0.37207 | DER: 1.00000
coderate: 0.25 | BER: 0.36529 | DER: 1.00000
coderate: 0.25 | BER: 0.35945 | DER: 1.00000
coderate: 0.25 | BER: 0.35267 | DER: 1.00000
coderate: 0.25 | BER: 0.34603 | DER: 1.00000
coderate: 0.25 | BER: 0.33882 | DER: 1.00000
```

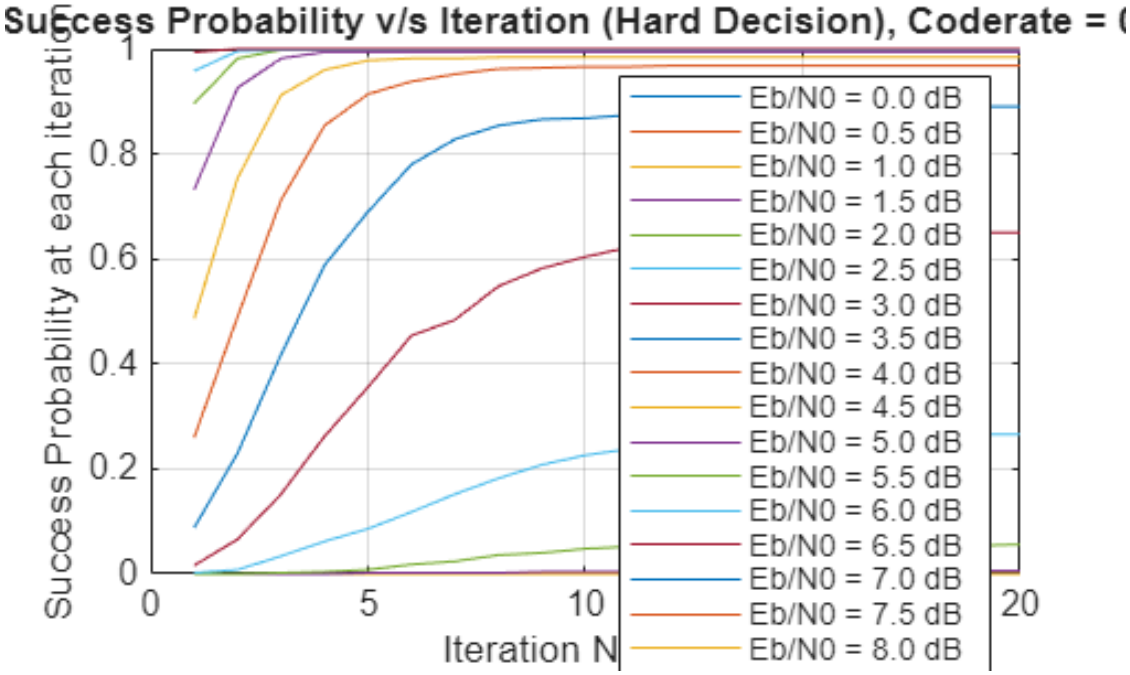
```
coderate: 0.25 | BER: 0.32875 | DER: 0.99600
coderate: 0.25 | BER: 0.25959 | DER: 0.84800
coderate: 0.25 | BER: 0.11797 | DER: 0.54200
coderate: 0.25 | BER: 0.03338 | DER: 0.26800
coderate: 0.25 | BER: 0.02346 | DER: 0.15400
coderate: 0.25 | BER: 0.02056 | DER: 0.12800
coderate: 0.25 | BER: 0.01775 | DER: 0.06800
coderate: 0.25 | BER: 0.01595 | DER: 0.04800
coderate: 0.25 | BER: 0.01396 | DER: 0.02200
coderate: 0.25 | BER: 0.01146 | DER: 0.01200
coderate: 0.25 | BER: 0.01147 | DER: 0.01200
coderate: 0.25 | BER: 0.01057 | DER: 0.00000
```



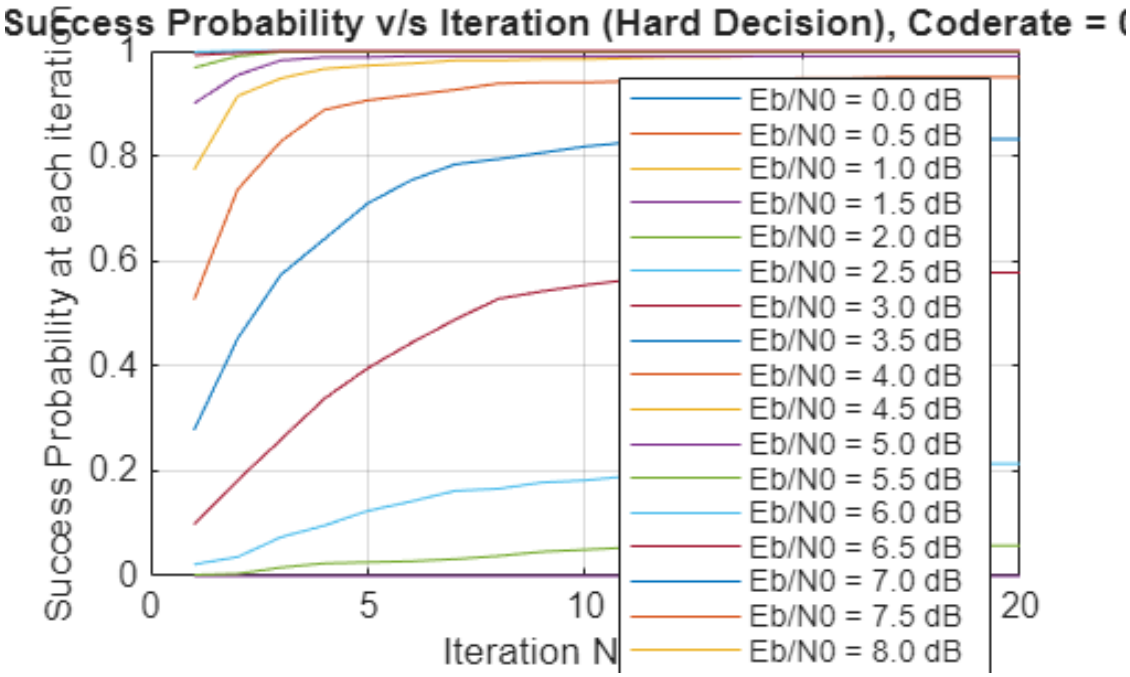

```
coderate: 0.33 | BER: 0.38024 | DER: 1.00000
coderate: 0.33 | BER: 0.37439 | DER: 1.00000
coderate: 0.33 | BER: 0.36808 | DER: 1.00000
coderate: 0.33 | BER: 0.36363 | DER: 1.00000
coderate: 0.33 | BER: 0.35699 | DER: 1.00000
coderate: 0.33 | BER: 0.35092 | DER: 1.00000
coderate: 0.33 | BER: 0.34463 | DER: 1.00000
coderate: 0.33 | BER: 0.33851 | DER: 1.00000
coderate: 0.33 | BER: 0.33220 | DER: 1.00000
coderate: 0.33 | BER: 0.32468 | DER: 0.99800
coderate: 0.33 | BER: 0.28382 | DER: 0.92800
coderate: 0.33 | BER: 0.14707 | DER: 0.57600
coderate: 0.33 | BER: 0.03695 | DER: 0.20200
coderate: 0.33 | BER: 0.01300 | DER: 0.10200
coderate: 0.33 | BER: 0.01017 | DER: 0.05200
coderate: 0.33 | BER: 0.00843 | DER: 0.01600
coderate: 0.33 | BER: 0.00687 | DER: 0.01000
coderate: 0.33 | BER: 0.00605 | DER: 0.00400
coderate: 0.33 | BER: 0.00540 | DER: 0.00200
coderate: 0.33 | BER: 0.00490 | DER: 0.00000
coderate: 0.33 | BER: 0.00414 | DER: 0.00000
```



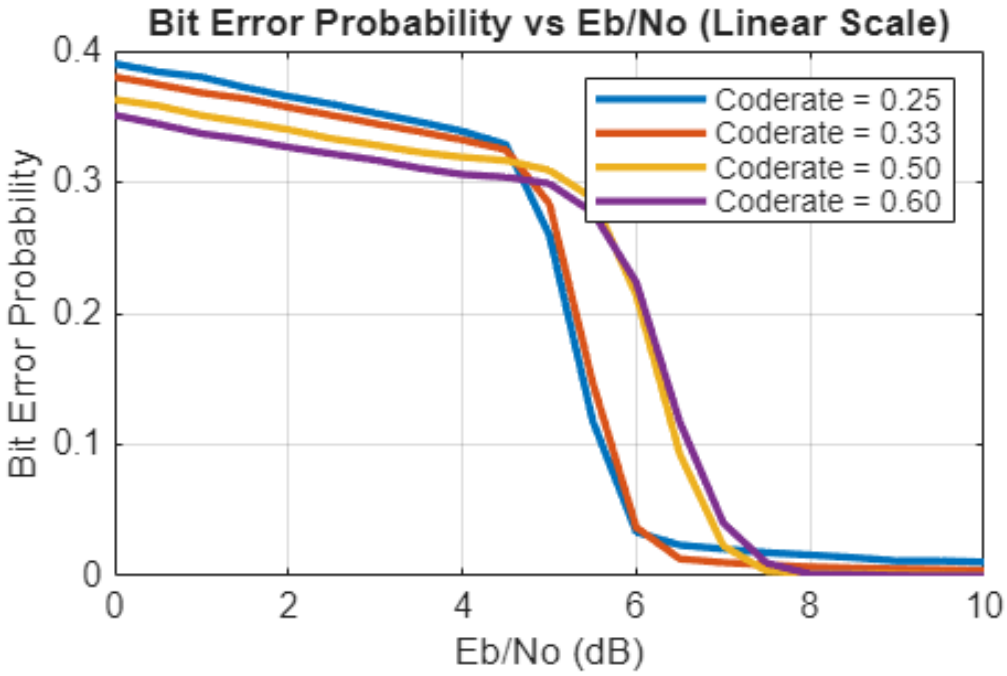

```
coderate: 0.50 | BER: 0.36298 | DER: 1.00000
coderate: 0.50 | BER: 0.35819 | DER: 1.00000
coderate: 0.50 | BER: 0.35075 | DER: 1.00000
coderate: 0.50 | BER: 0.34563 | DER: 1.00000
coderate: 0.50 | BER: 0.33996 | DER: 1.00000
coderate: 0.50 | BER: 0.33311 | DER: 1.00000
coderate: 0.50 | BER: 0.32827 | DER: 1.00000
coderate: 0.50 | BER: 0.32281 | DER: 1.00000
coderate: 0.50 | BER: 0.31908 | DER: 1.00000
coderate: 0.50 | BER: 0.31650 | DER: 1.00000
coderate: 0.50 | BER: 0.30913 | DER: 0.99400
coderate: 0.50 | BER: 0.28766 | DER: 0.94400
coderate: 0.50 | BER: 0.21405 | DER: 0.73400
coderate: 0.50 | BER: 0.09289 | DER: 0.35000
coderate: 0.50 | BER: 0.02299 | DER: 0.11000
coderate: 0.50 | BER: 0.00356 | DER: 0.03200
coderate: 0.50 | BER: 0.00203 | DER: 0.01600
coderate: 0.50 | BER: 0.00182 | DER: 0.00600
coderate: 0.50 | BER: 0.00131 | DER: 0.00000
coderate: 0.50 | BER: 0.00090 | DER: 0.00200
coderate: 0.50 | BER: 0.00051 | DER: 0.00000
```



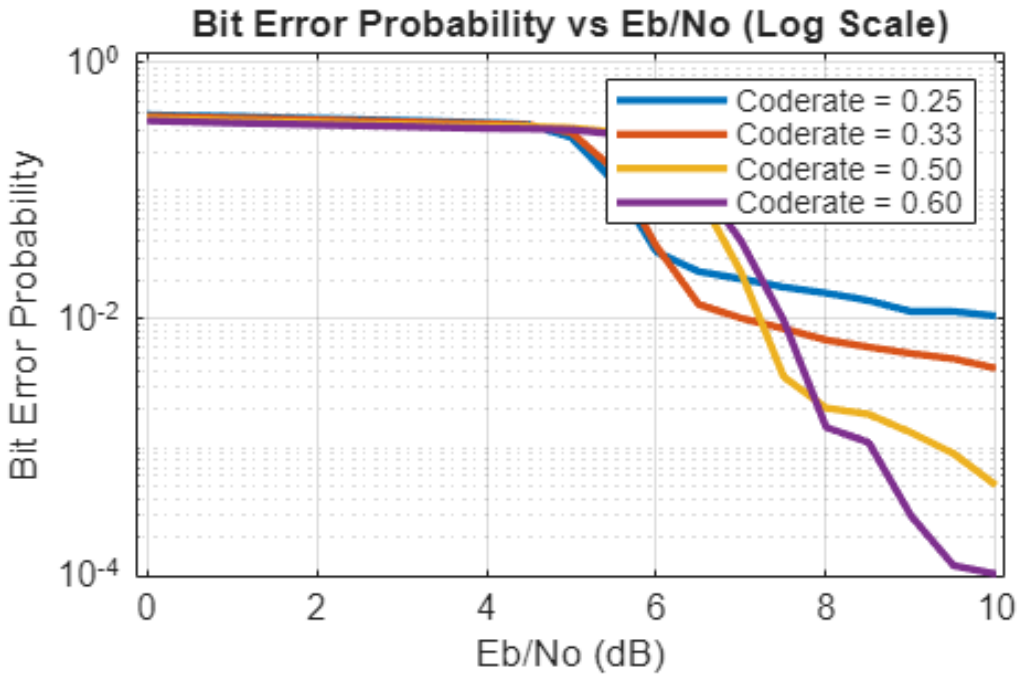

```
coderate: 0.60 | BER: 0.35118 | DER: 1.00000
coderate: 0.60 | BER: 0.34431 | DER: 1.00000
coderate: 0.60 | BER: 0.33699 | DER: 1.00000
coderate: 0.60 | BER: 0.33245 | DER: 1.00000
coderate: 0.60 | BER: 0.32659 | DER: 1.00000
coderate: 0.60 | BER: 0.32160 | DER: 1.00000
coderate: 0.60 | BER: 0.31688 | DER: 1.00000
coderate: 0.60 | BER: 0.31073 | DER: 1.00000
coderate: 0.60 | BER: 0.30591 | DER: 1.00000
coderate: 0.60 | BER: 0.30384 | DER: 1.00000
coderate: 0.60 | BER: 0.29900 | DER: 1.00000
coderate: 0.60 | BER: 0.27701 | DER: 0.94200
coderate: 0.60 | BER: 0.22375 | DER: 0.78600
coderate: 0.60 | BER: 0.11729 | DER: 0.42200
coderate: 0.60 | BER: 0.04038 | DER: 0.16800
coderate: 0.60 | BER: 0.00975 | DER: 0.05000
coderate: 0.60 | BER: 0.00144 | DER: 0.01000
coderate: 0.60 | BER: 0.00110 | DER: 0.01000
coderate: 0.60 | BER: 0.00029 | DER: 0.00200
coderate: 0.60 | BER: 0.00012 | DER: 0.00000
coderate: 0.60 | BER: 0.00010 | DER: 0.00000
```



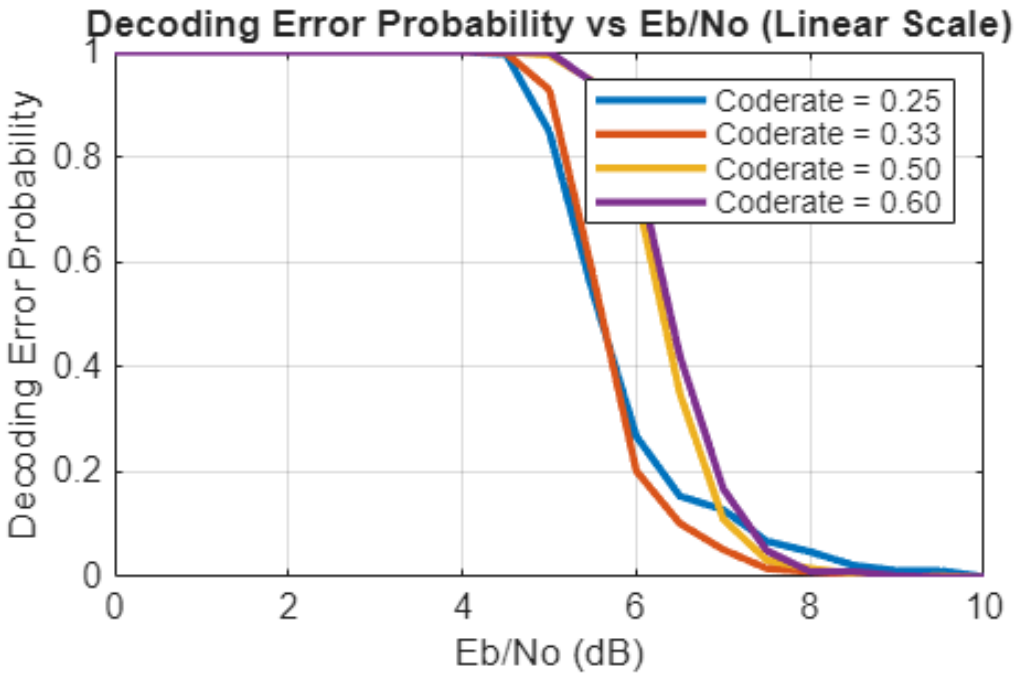

```
%% Final plots for all coderates

% BER Linear Scale
figure;
for i = 1:length(coderate)
    plot(Eb_no_db, bit_error_all(i, :), 'LineWidth', 2, 'DisplayName',
sprintf('Coderate = %.2f', coderate(i)));
    hold on;
end
xlabel('Eb/No (dB)');
ylabel('Bit Error Probability');
title('Bit Error Probability vs Eb/No (Linear Scale)');
legend('show');
grid on;
```



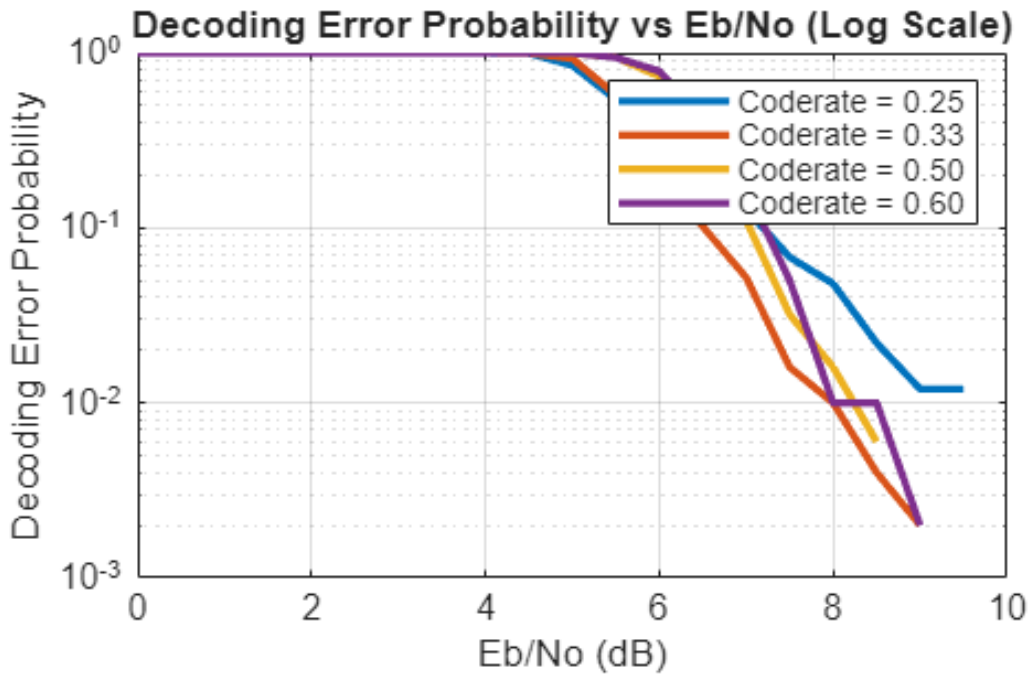

```
% BER Log Scale
figure;
for i = 1:length(coderate)
    semilogy(Eb_no_db, bit_error_all(i, :), 'LineWidth', 2, 'DisplayName',
sprintf('Coderate = %.2f', coderate(i)));
    hold on;
end
xlabel('Eb/No (dB)');
ylabel('Bit Error Probability');
title('Bit Error Probability vs Eb/No (Log Scale)');
legend('show');
grid on;
```



```
% DER Linear Scale
figure;
for i = 1:length(coderate)
    plot(Eb_no_db, decoding_error_all(i, :), 'LineWidth', 2, 'DisplayName',
sprintf('Coderate = %.2f', coderate(i)));
    hold on;
end
xlabel('Eb/No (dB)');
ylabel('Decoding Error Probability');
title('Decoding Error Probability vs Eb/No (Linear Scale)');
legend('show');
grid on;
```

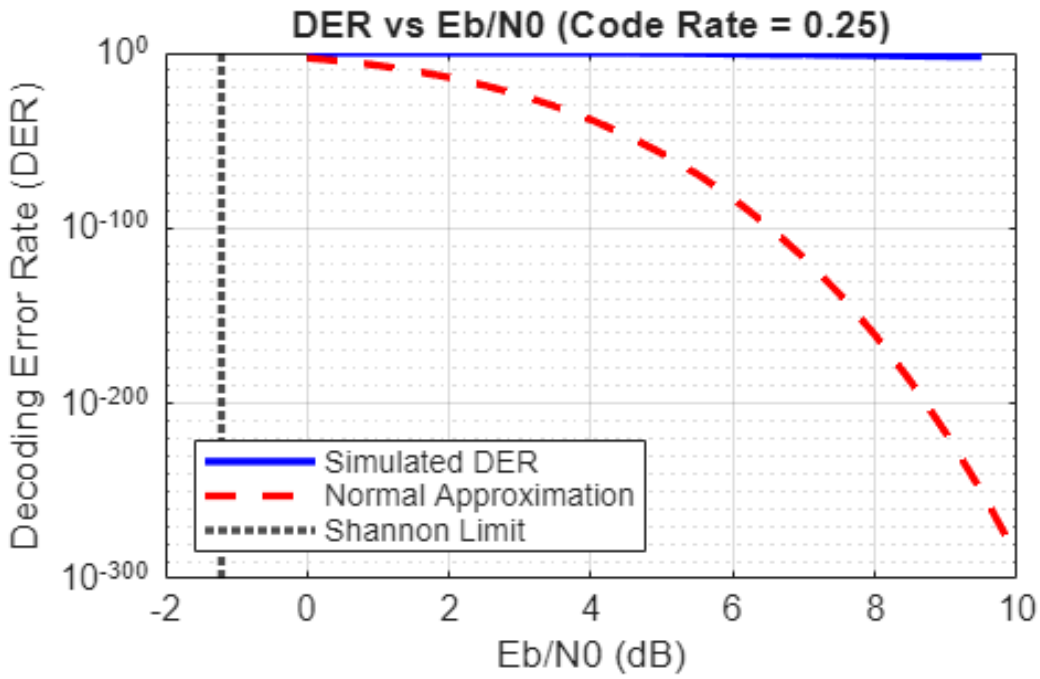


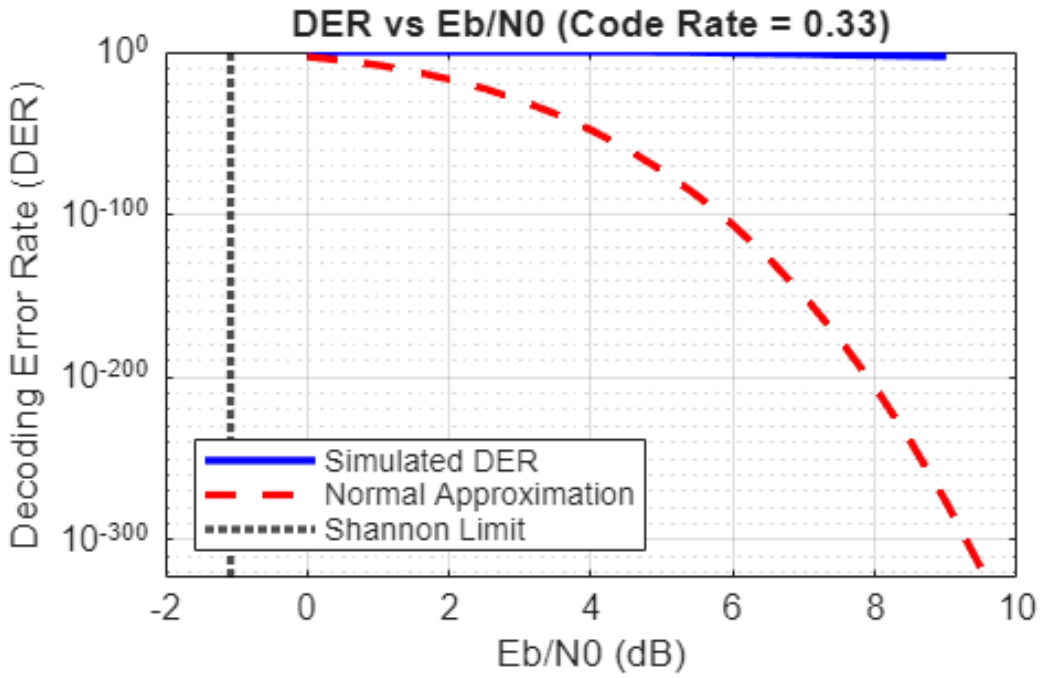

```
% DER Log Scale
figure;
for i = 1:length(coderate)
    semilogy(Eb_no_db, decoding_error_all(i, :), 'LineWidth', 2, 'DisplayName',
sprintf('Coderate = %.2f', coderate(i)));
    hold on;
end
xlabel('Eb/No (dB)');
ylabel('Decoding Error Probability');
title('Decoding Error Probability vs Eb/No (Log Scale)');
legend('show');
grid on;
```

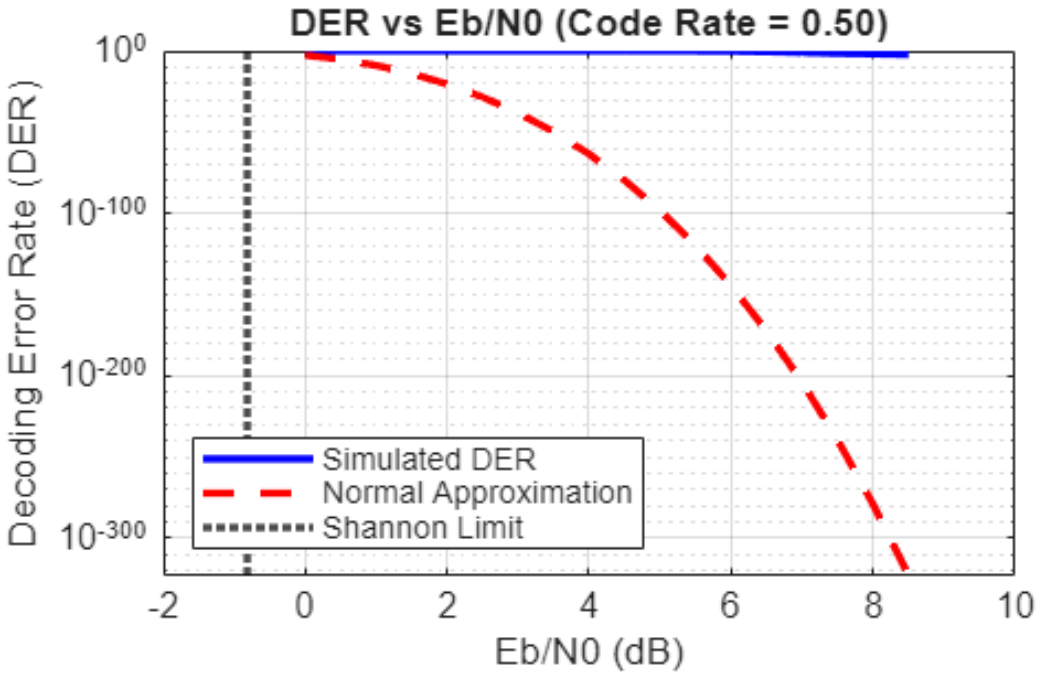


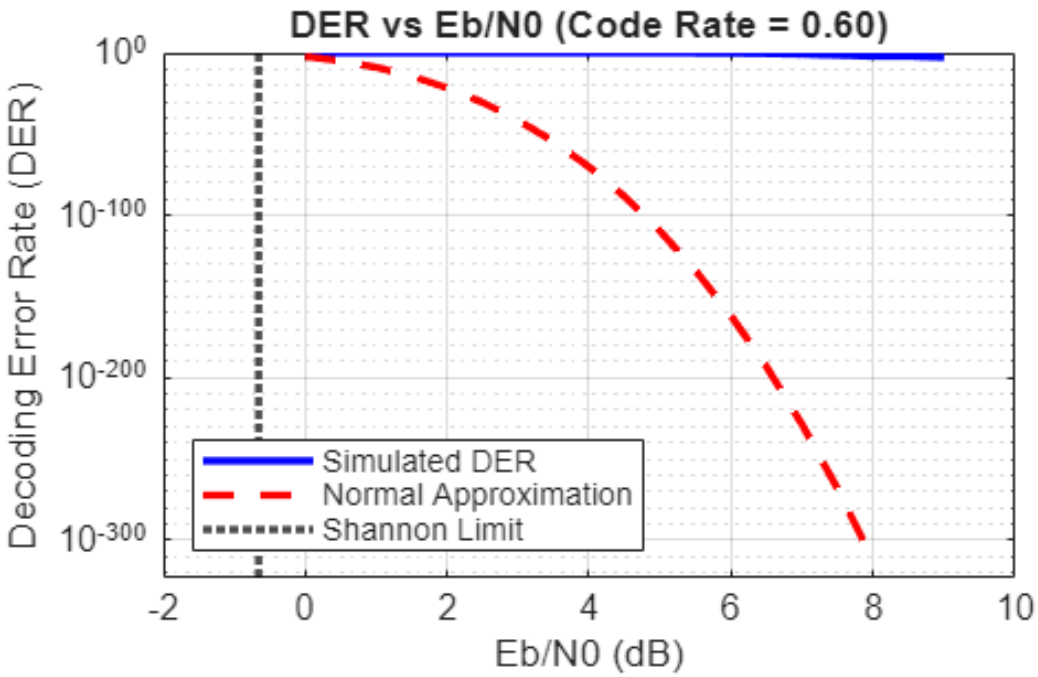

```
EbNodB = 0:0.5:10;
EbNo = 10.^(EbNodB/10);
N = 512;  % Blocklength
log2e = log2(exp(1));

for i = 1:length(coderate)
    r = coderate(i);
    PN_e = zeros(size(EbNo));

    % Calculate Normal Approximation for each Eb/N0
    for j = 1:length(EbNo)
        P = r * EbNo(j);
        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(j) = qfunc(NA_term);
    end

    % Shannon limit for current code rate
    shannonLimit_dB = 10 * log10((2^r - 1)/r);

    % Plotting for this coderate
    figure;
    semilogy(EbNodB, decoding_error_all(i, :), 'b-', 'LineWidth', 2);
    hold on;
    semilogy(EbNodB, PN_e, 'r--', 'LineWidth', 2);
    xline(shannonLimit_dB, 'k:', 'LineWidth', 2, ...
        'DisplayName', sprintf('Shannon Limit = %.2f dB', shannonLimit_dB));

    grid on;
    xlabel('Eb/N0 (dB)');
```

```
    ylabel('Decoding Error Rate (DER)');
    title(sprintf('DER vs Eb/N0 (Code Rate = %.2f)', r));
    legend('Simulated DER', 'Normal Approximation', 'Shannon Limit', 'Location',
'southwest');
end
```










```
function [vn_to_cn_conn, cn_to_vn_conn, VN_to_CN_Message, CN_to_VN_Message] =
Connection(H)

    [row, col] = size(H);
    vn_to_cn_conn = cell(col, 1);
    cn_to_vn_conn = cell(row, 1);
    VN_to_CN_Message = zeros(row, col);
    CN_to_VN_Message = zeros(row, col);

    for i=1:col
        for j=1:row
```

```
            if H(j, i)==1
                vn_to_cn_conn{i} = [vn_to_cn_conn{i} j];
            end

        end
    end


    for i=1:row
        for j=1:col
            if H(i, j)==1
                cn_to_vn_conn{i} = [cn_to_vn_conn{i} j];
            end
        end
    end

end
```

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