```
function decode = minsum(H , r)

    %L matrix
    [row , col] = size(H);


    L = zeros(row,col);

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



    maxitr = 20;

    for itr = 1:maxitr

        for i=1:row

            min1 = inf;
            min2 = inf;
            tol = 1e-10;
            count = 0;

            for j=1:col

                if L(i,j) < 0
                    count = count + 1;
                end

            end



            for j=1:col
                if L(i,j) ~= 0
                    value = abs(L(i,j));
                    if value < min1 - tol

                        min2 = min1;
                        min1 = value;
                    elseif (value > min1 + tol) && (value < min2 - tol)

                        min2 = value;
                    end
                end
            end




            p = mod(count , 2);
            vec = [1 , -1];

            tol2 = 1e-10;

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


        end



        for j=1:col

            sum = 0;

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

            sum = sum + r(j);

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

            r(j) = sum;

        end


    end

    decode = r;
end
```

```
function [decode,itr_update] = Iteration_Successs(H , r , itr_succ , b ,
kNumInfoBits)

    %L matrix
    [row , col] = size(H);


    L = zeros(row,col);

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

    maxitr = 20;

    for itr = 1:maxitr

        for i=1:row

            min1 = inf;
            min2 = inf;
            tol = 1e-10;
            count = 0;

            for j=1:col

                if L(i,j) < 0
                    count = count + 1;
                end

            end


            for j=1:col
                if L(i,j) ~= 0
                    value = abs(L(i,j));
                    if value < min1 - tol

                        min2 = min1;
                        min1 = value;
                    elseif (value > min1 + tol) && (value < min2 - tol)

                        min2 = value;
                    end
                end
            end


            p = mod(count , 2);
            vec = [1 , -1];

            tol2 = 1e-10;

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


        end


        for j=1:col

            sum2 = 0;

            for i=1:row
                sum2 = sum2 + L(i,j);
            end

            sum2 = sum2 + r(j);

            for i=1:row
                if L(i,j) ~= 0
                    L(i,j) = sum2 - L(i,j);
                end
            end

            r(j) = sum2;

        end

        c2 = (r < 0);

        if sum(xor(c2(1:kNumInfoBits) , b)) == 0
            for j=itr:maxitr
                itr_succ(j) = itr_succ(j) + 1;
            end
            break;
        end
```

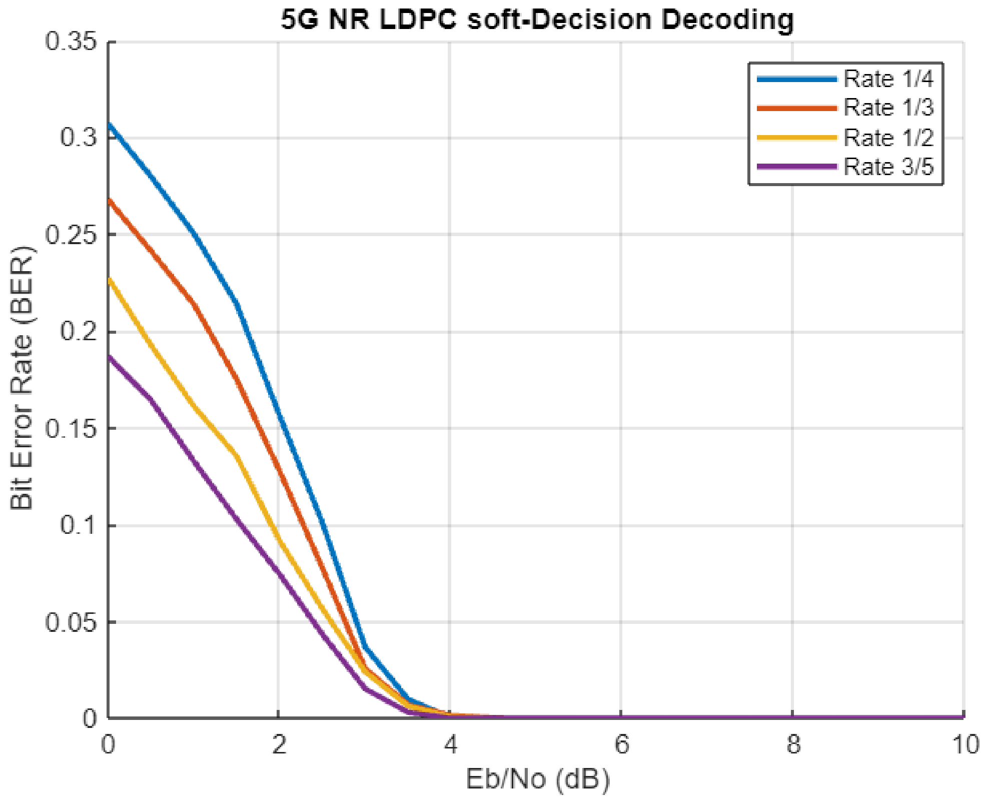
```
    end

    decode = r;
    itr_update = itr_succ;
end
```

```
clc;
clear;

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

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

EbNodB = 0:0.5:10;    % limit of Eb/No in DB
Nsim = 100;           % Number of Monte Carlo runs
max_itr = 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;

    plotvec = zeros(1, length(EbNodB));  % vector to store BER

    for idx = 1:length(EbNodB)

        value = EbNodB(idx);
        EbNo = 10^(value/10);
        sigma = sqrt(1 / (2 * codeRate * EbNo));
        total_error_bit = 0;

        for NsimIdx = 1:Nsim

            b = randi([0 1], [kNumInfoBits 1]);  % message bits
            c = nrldpc_encode(B, z, b');         % Encode
            c = c(1:nBlockLength)';

            s = 1 - 2 * c;                           % BPSK modulation
            r = s + sigma * randn(nBlockLength, 1);  % AWGN channel


            VN = minsum(H , r);

            %convert into bit
            decoded = (VN < 0);

            total_error_bit = total_error_bit + sum(decoded ~= c);
        end

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

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

```
Rate 0.25 | Eb/No = 0.0 dB | BER = 0.30696
Rate 0.25 | Eb/No = 0.5 dB | BER = 0.27972
Rate 0.25 | Eb/No = 1.0 dB | BER = 0.25026
Rate 0.25 | Eb/No = 1.5 dB | BER = 0.21381
Rate 0.25 | Eb/No = 2.0 dB | BER = 0.15626
Rate 0.25 | Eb/No = 2.5 dB | BER = 0.10085
Rate 0.25 | Eb/No = 3.0 dB | BER = 0.03665
Rate 0.25 | Eb/No = 3.5 dB | BER = 0.00975
Rate 0.25 | Eb/No = 4.0 dB | BER = 0.00023
Rate 0.25 | Eb/No = 4.5 dB | BER = 0.00000
Rate 0.25 | Eb/No = 5.0 dB | BER = 0.00000
Rate 0.25 | Eb/No = 5.5 dB | BER = 0.00000
Rate 0.25 | Eb/No = 6.0 dB | BER = 0.00000
Rate 0.25 | Eb/No = 6.5 dB | BER = 0.00000
Rate 0.25 | Eb/No = 7.0 dB | BER = 0.00000
Rate 0.25 | Eb/No = 7.5 dB | BER = 0.00000
Rate 0.25 | Eb/No = 8.0 dB | BER = 0.00000
Rate 0.25 | Eb/No = 8.5 dB | BER = 0.00000
Rate 0.25 | Eb/No = 9.0 dB | BER = 0.00000
Rate 0.25 | Eb/No = 9.5 dB | BER = 0.00000
Rate 0.25 | Eb/No = 10.0 dB | BER = 0.00000
Rate 0.33 | Eb/No = 0.0 dB | BER = 0.26788
Rate 0.33 | Eb/No = 0.5 dB | BER = 0.24125
Rate 0.33 | Eb/No = 1.0 dB | BER = 0.21366
Rate 0.33 | Eb/No = 1.5 dB | BER = 0.17496
Rate 0.33 | Eb/No = 2.0 dB | BER = 0.12767
Rate 0.33 | Eb/No = 2.5 dB | BER = 0.07746
Rate 0.33 | Eb/No = 3.0 dB | BER = 0.02558
Rate 0.33 | Eb/No = 3.5 dB | BER = 0.00692
Rate 0.33 | Eb/No = 4.0 dB | BER = 0.00141
Rate 0.33 | Eb/No = 4.5 dB | BER = 0.00023
Rate 0.33 | Eb/No = 5.0 dB | BER = 0.00000
Rate 0.33 | Eb/No = 5.5 dB | BER = 0.00000
Rate 0.33 | Eb/No = 6.0 dB | BER = 0.00000
Rate 0.33 | Eb/No = 6.5 dB | BER = 0.00000
Rate 0.33 | Eb/No = 7.0 dB | BER = 0.00000
Rate 0.33 | Eb/No = 7.5 dB | BER = 0.00000
Rate 0.33 | Eb/No = 8.0 dB | BER = 0.00000
Rate 0.33 | Eb/No = 8.5 dB | BER = 0.00000
Rate 0.33 | Eb/No = 9.0 dB | BER = 0.00000
Rate 0.33 | Eb/No = 9.5 dB | BER = 0.00000
Rate 0.33 | Eb/No = 10.0 dB | BER = 0.00000
Rate 0.50 | Eb/No = 0.0 dB | BER = 0.22733
Rate 0.50 | Eb/No = 0.5 dB | BER = 0.19256
Rate 0.50 | Eb/No = 1.0 dB | BER = 0.16111
Rate 0.50 | Eb/No = 1.5 dB | BER = 0.13529
Rate 0.50 | Eb/No = 2.0 dB | BER = 0.09162
Rate 0.50 | Eb/No = 2.5 dB | BER = 0.05645
Rate 0.50 | Eb/No = 3.0 dB | BER = 0.02394
Rate 0.50 | Eb/No = 3.5 dB | BER = 0.00623
Rate 0.50 | Eb/No = 4.0 dB | BER = 0.00073
Rate 0.50 | Eb/No = 4.5 dB | BER = 0.00000
Rate 0.50 | Eb/No = 5.0 dB | BER = 0.00000
Rate 0.50 | Eb/No = 5.5 dB | BER = 0.00000
Rate 0.50 | Eb/No = 6.0 dB | BER = 0.00000
Rate 0.50 | Eb/No = 6.5 dB | BER = 0.00000
Rate 0.50 | Eb/No = 7.0 dB | BER = 0.00000
Rate 0.50 | Eb/No = 7.5 dB | BER = 0.00000
Rate 0.50 | Eb/No = 8.0 dB | BER = 0.00000
Rate 0.50 | Eb/No = 8.5 dB | BER = 0.00000
Rate 0.50 | Eb/No = 9.0 dB | BER = 0.00000
Rate 0.50 | Eb/No = 9.5 dB | BER = 0.00000
Rate 0.50 | Eb/No = 10.0 dB | BER = 0.00000
Rate 0.60 | Eb/No = 0.0 dB | BER = 0.18696
Rate 0.60 | Eb/No = 0.5 dB | BER = 0.16417
Rate 0.60 | Eb/No = 1.0 dB | BER = 0.13273
Rate 0.60 | Eb/No = 1.5 dB | BER = 0.10263
Rate 0.60 | Eb/No = 2.0 dB | BER = 0.07429
Rate 0.60 | Eb/No = 2.5 dB | BER = 0.04331
Rate 0.60 | Eb/No = 3.0 dB | BER = 0.01514
Rate 0.60 | Eb/No = 3.5 dB | BER = 0.00302
Rate 0.60 | Eb/No = 4.0 dB | BER = 0.00000
Rate 0.60 | Eb/No = 4.5 dB | BER = 0.00018
Rate 0.60 | Eb/No = 5.0 dB | BER = 0.00000
Rate 0.60 | Eb/No = 5.5 dB | BER = 0.00000
Rate 0.60 | Eb/No = 6.0 dB | BER = 0.00000
Rate 0.60 | Eb/No = 6.5 dB | BER = 0.00000
Rate 0.60 | Eb/No = 7.0 dB | BER = 0.00000
Rate 0.60 | Eb/No = 7.5 dB | BER = 0.00000
Rate 0.60 | Eb/No = 8.0 dB | BER = 0.00000
Rate 0.60 | Eb/No = 8.5 dB | BER = 0.00000
Rate 0.60 | Eb/No = 9.0 dB | BER = 0.00000
Rate 0.60 | Eb/No = 9.5 dB | BER = 0.00000
Rate 0.60 | Eb/No = 10.0 dB | BER = 0.00000
```

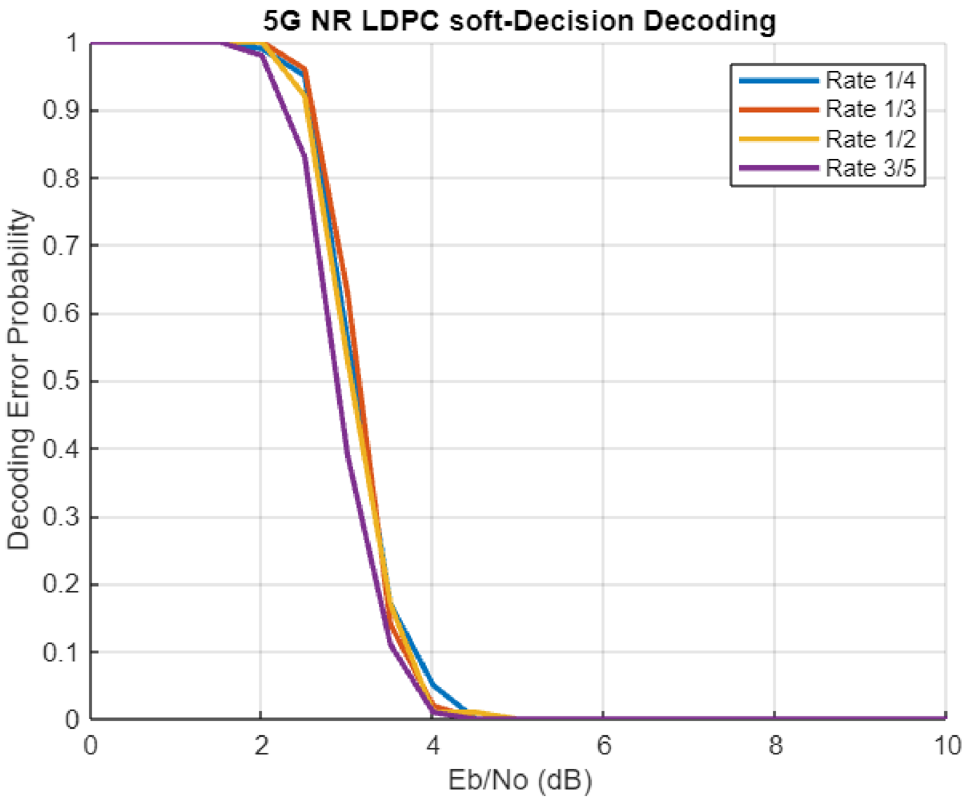
```
legend('Rate 1/4', 'Rate 1/3', 'Rate 1/2', 'Rate 3/5');
xlabel('Eb/No (dB)');
ylabel('Bit Error Rate (BER)');
title('5G NR LDPC soft-Decision Decoding');
grid on;
hold off;
```



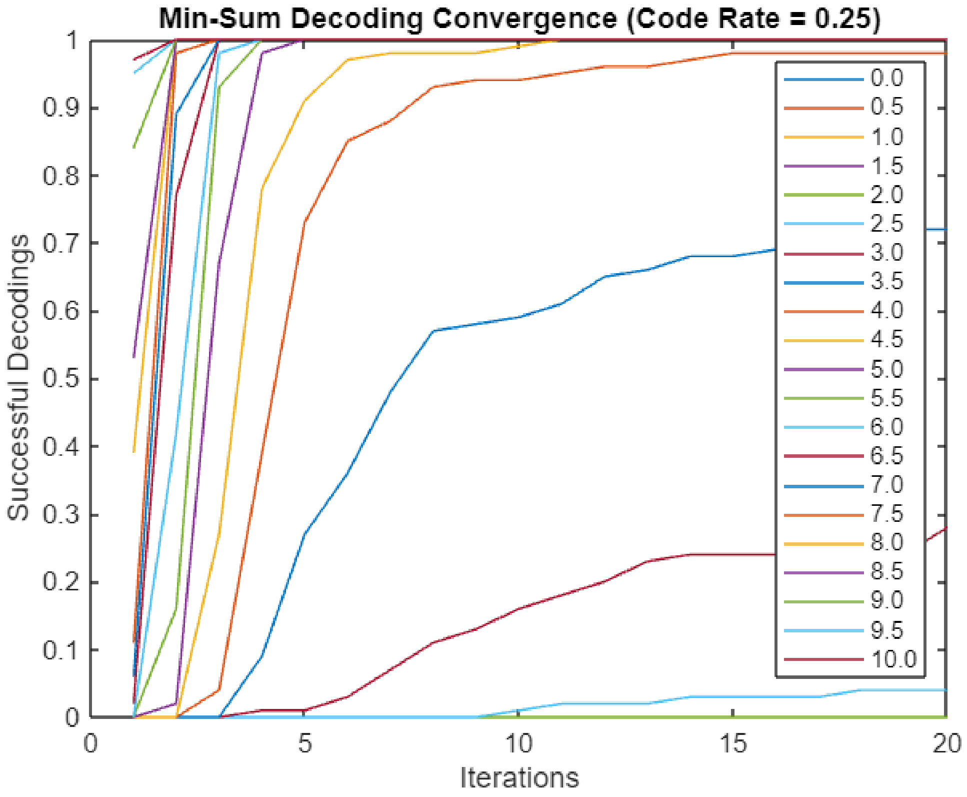
```
figure;
hold on;

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;

    decoding_error = zeros(1, length(EbNodB));  % BER for each SNR

    for idx = 1:length(EbNodB)

        value = EbNodB(idx);
        EbNo = 10^(value/10);
        sigma = sqrt(1 / (2 * codeRate * EbNo));
        success = 0;
        for NsimIdx = 1:Nsim

            b = randi([0 1], [kNumInfoBits 1]);  % message bits
            c = nrldpc_encode(B, z, b');         % Encode
            c = c(1:nBlockLength)';

            s = 1 - 2 * c;                           % BPSK modulation
            r = s + sigma * randn(nBlockLength, 1);  % AWGN channel


            VN = minsum(H , r);

            %convert into bit
            VN = (VN < 0);

            %decode
            decoded = VN(1:kNumInfoBits);

            if sum(xor(decoded,b)) == 0
                success = success + 1;
            end

        end

        decoding_error(idx) = (Nsim - success) / Nsim;
        fprintf('Rate %.2f | Eb/No = %.1f dB | Error_prob = %.5f\n', codeRate,
value, decoding_error(idx));
    end

    semilogy(EbNodB, decoding_error, 'LineWidth', 2);
end
```

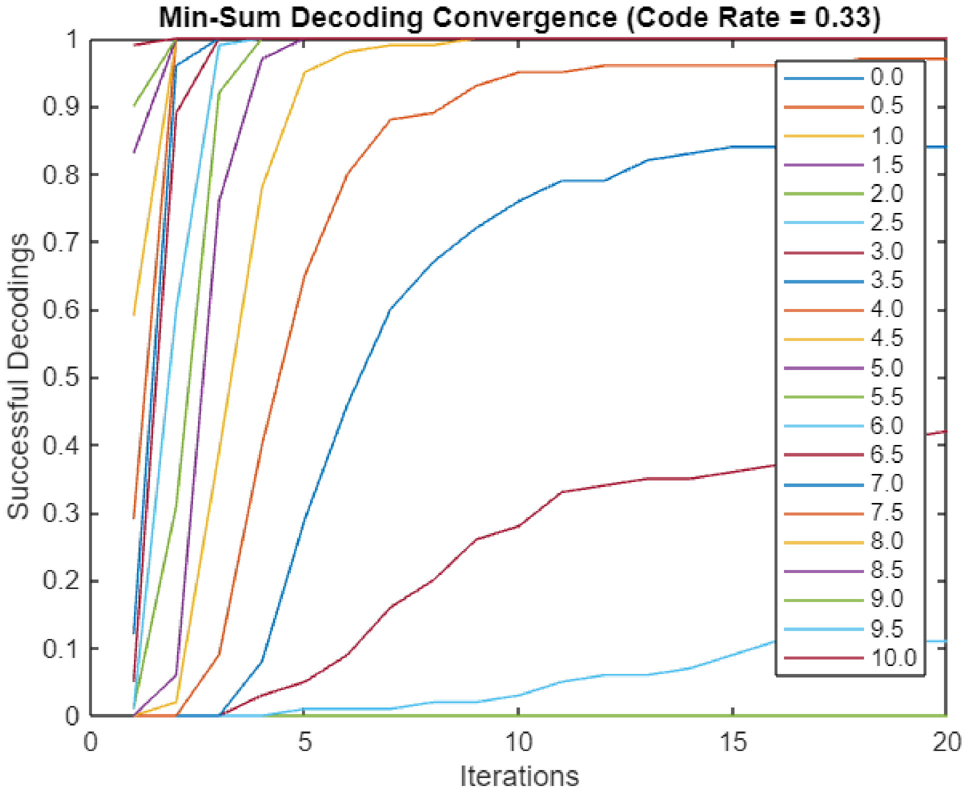
```
Rate 0.25 | Eb/No = 0.0 dB | Error_prob = 1.00000
Rate 0.25 | Eb/No = 0.5 dB | Error_prob = 1.00000
Rate 0.25 | Eb/No = 1.0 dB | Error_prob = 1.00000
Rate 0.25 | Eb/No = 1.5 dB | Error_prob = 1.00000
Rate 0.25 | Eb/No = 2.0 dB | Error_prob = 0.99000
Rate 0.25 | Eb/No = 2.5 dB | Error_prob = 0.95000
Rate 0.25 | Eb/No = 3.0 dB | Error_prob = 0.56000
Rate 0.25 | Eb/No = 3.5 dB | Error_prob = 0.17000
Rate 0.25 | Eb/No = 4.0 dB | Error_prob = 0.05000
Rate 0.25 | Eb/No = 4.5 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 5.0 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 5.5 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 6.0 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 6.5 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 7.0 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 7.5 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 8.0 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 8.5 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 9.0 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 9.5 dB | Error_prob = 0.00000
Rate 0.25 | Eb/No = 10.0 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 0.0 dB | Error_prob = 1.00000
Rate 0.33 | Eb/No = 0.5 dB | Error_prob = 1.00000
Rate 0.33 | Eb/No = 1.0 dB | Error_prob = 1.00000
Rate 0.33 | Eb/No = 1.5 dB | Error_prob = 1.00000
Rate 0.33 | Eb/No = 2.0 dB | Error_prob = 1.00000
Rate 0.33 | Eb/No = 2.5 dB | Error_prob = 0.96000
Rate 0.33 | Eb/No = 3.0 dB | Error_prob = 0.63000
Rate 0.33 | Eb/No = 3.5 dB | Error_prob = 0.14000
Rate 0.33 | Eb/No = 4.0 dB | Error_prob = 0.02000
Rate 0.33 | Eb/No = 4.5 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 5.0 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 5.5 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 6.0 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 6.5 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 7.0 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 7.5 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 8.0 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 8.5 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 9.0 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 9.5 dB | Error_prob = 0.00000
Rate 0.33 | Eb/No = 10.0 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 0.0 dB | Error_prob = 1.00000
Rate 0.50 | Eb/No = 0.5 dB | Error_prob = 1.00000
Rate 0.50 | Eb/No = 1.0 dB | Error_prob = 1.00000
Rate 0.50 | Eb/No = 1.5 dB | Error_prob = 1.00000
Rate 0.50 | Eb/No = 2.0 dB | Error_prob = 1.00000
Rate 0.50 | Eb/No = 2.5 dB | Error_prob = 0.92000
Rate 0.50 | Eb/No = 3.0 dB | Error_prob = 0.53000
Rate 0.50 | Eb/No = 3.5 dB | Error_prob = 0.17000
Rate 0.50 | Eb/No = 4.0 dB | Error_prob = 0.01000
Rate 0.50 | Eb/No = 4.5 dB | Error_prob = 0.01000
Rate 0.50 | Eb/No = 5.0 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 5.5 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 6.0 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 6.5 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 7.0 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 7.5 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 8.0 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 8.5 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 9.0 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 9.5 dB | Error_prob = 0.00000
Rate 0.50 | Eb/No = 10.0 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 0.0 dB | Error_prob = 1.00000
Rate 0.60 | Eb/No = 0.5 dB | Error_prob = 1.00000
Rate 0.60 | Eb/No = 1.0 dB | Error_prob = 1.00000
Rate 0.60 | Eb/No = 1.5 dB | Error_prob = 1.00000
Rate 0.60 | Eb/No = 2.0 dB | Error_prob = 0.98000
Rate 0.60 | Eb/No = 2.5 dB | Error_prob = 0.83000
Rate 0.60 | Eb/No = 3.0 dB | Error_prob = 0.39000
Rate 0.60 | Eb/No = 3.5 dB | Error_prob = 0.11000
Rate 0.60 | Eb/No = 4.0 dB | Error_prob = 0.01000
Rate 0.60 | Eb/No = 4.5 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 5.0 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 5.5 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 6.0 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 6.5 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 7.0 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 7.5 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 8.0 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 8.5 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 9.0 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 9.5 dB | Error_prob = 0.00000
Rate 0.60 | Eb/No = 10.0 dB | Error_prob = 0.00000
```

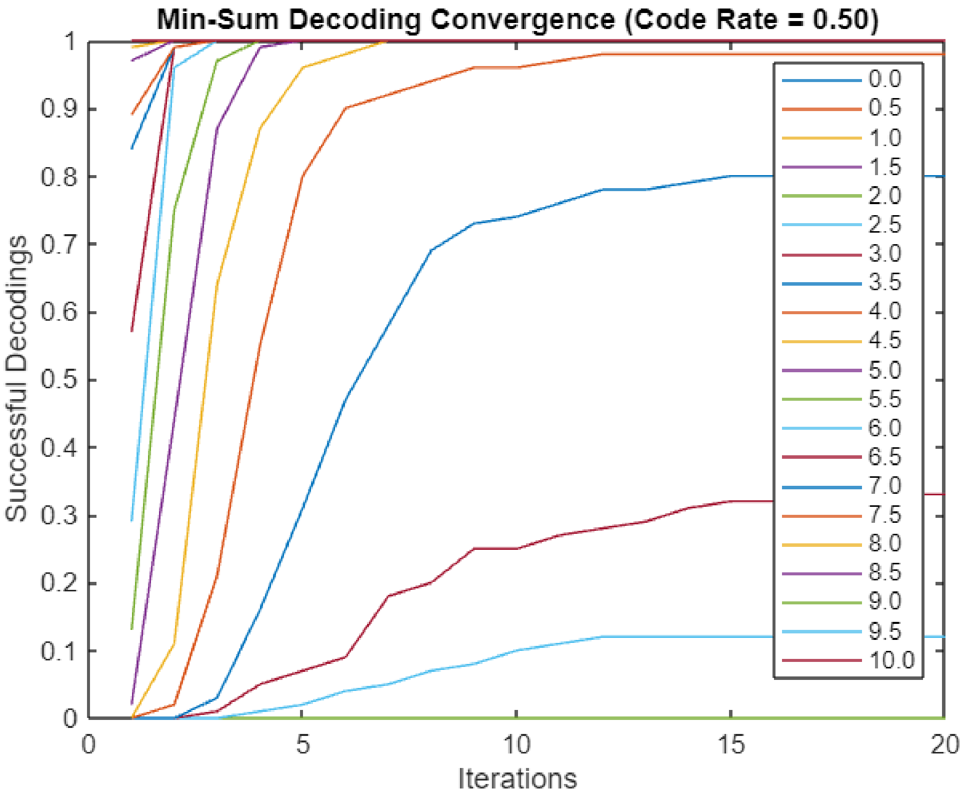
```
legend('Rate 1/4', 'Rate 1/3', 'Rate 1/2', 'Rate 3/5');
xlabel('Eb/No (dB)');
ylabel('Decoding Error Probability');
title('5G NR LDPC soft-Decision Decoding');
grid on;
hold off;
```

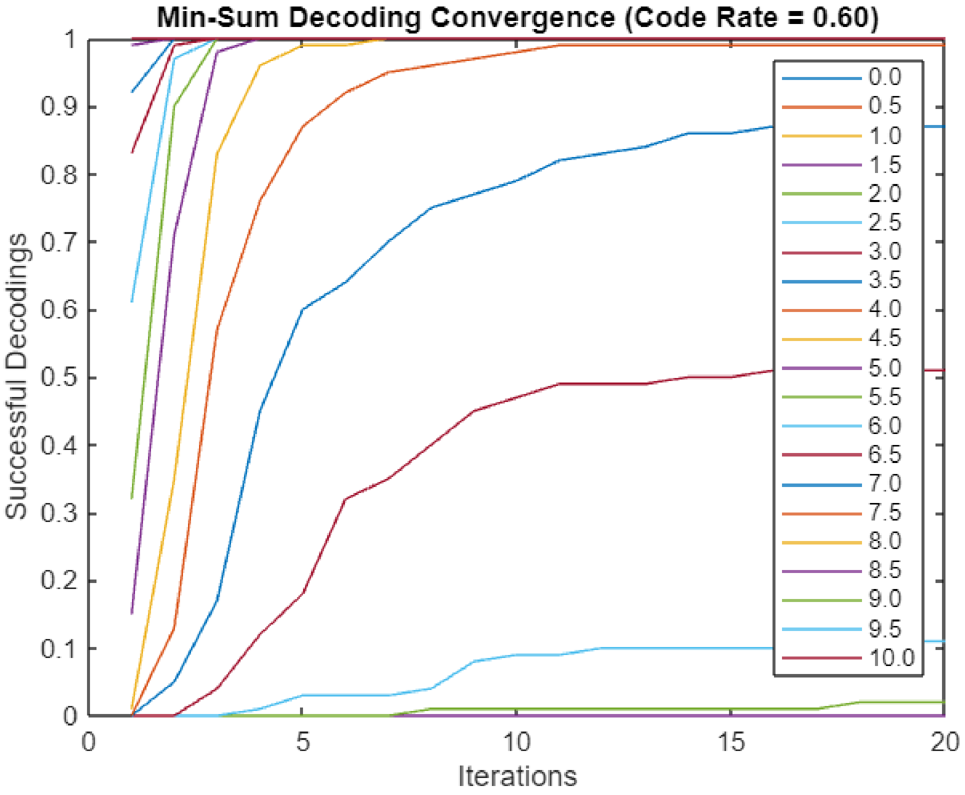


```
for i = 1:length(codeRates)
    figure;

    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;

    for idx = 1:length(EbNodB)

        value = EbNodB(idx);
        EbNo = 10^(value/10);
        sigma = sqrt(1 / (2 * codeRate * EbNo));
        total_error_bit = 0;

        itr_succ = zeros(1,max_itr);

        for NsimIdx = 1:Nsim

            b = randi([0 1], [kNumInfoBits 1]);  % Random message bits
            c = nrldpc_encode(B, z, b');         % Encode
            c = c(1:nBlockLength)';

            s = 1 - 2 * c;                           % BPSK modulation
            r = s + sigma * randn(nBlockLength, 1);  % AWGN channel


            [VN,itr_succ]  = Iteration_Successss(H , r , itr_succ , b ,
kNumInfoBits);


        end

        plot(1:max_itr,itr_succ./Nsim);
        hold on;
    end

    xlabel('Iterations');
    ylabel('Successful Decodings');
    title(sprintf('Min-Sum Decoding Convergence (Code Rate = %.2f)', codeRate));

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');
    hold off;

end
```


Min-Sum Decoding Convergence (Code Rate = 0.25)
Successful Decodings
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Iterations
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

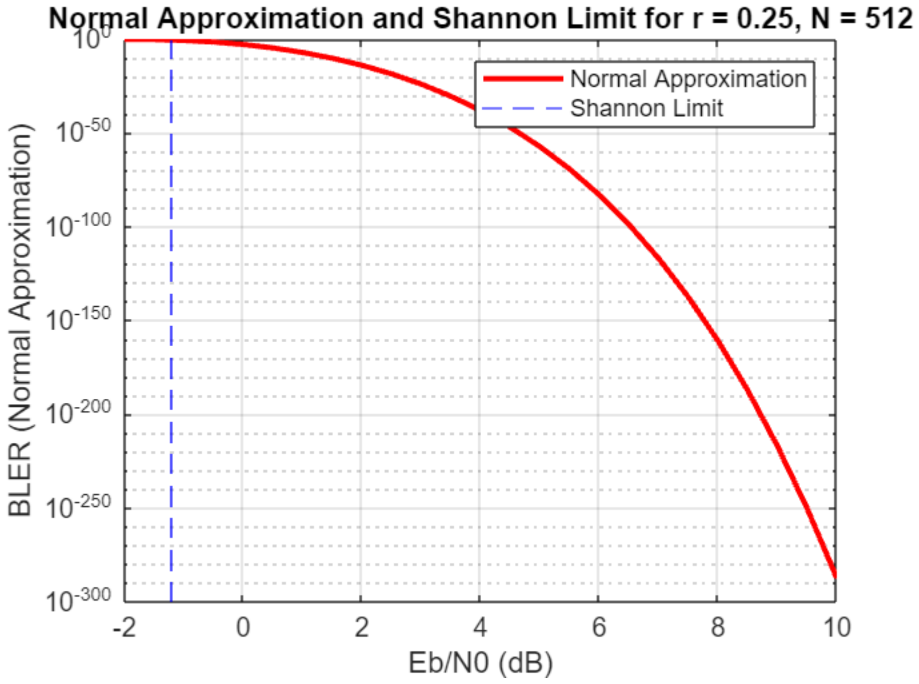

Min-Sum Decoding Convergence (Code Rate = 0.33)
Successful Decodings
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Iterations
0
5
10
15
20
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0


Min-Sum Decoding Convergence (Code Rate = 0.50)


Min-Sum Decoding Convergence (Code Rate = 0.60)

```
%shannon
r = 1/4;
```

```
N = 512;
EbNodB = -2:0.5:10;
EbNo = 10.^(EbNodB/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); %qfunc() use for calculate Q() function
end

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


semilogy(EbNodB, PN_e, 'r-', 'LineWidth', 2);
hold on;

xline(shannonLimit_dB, '--b');

xlabel('Eb/N0 (dB)');
ylabel('BLER (Normal Approximation)');
title(sprintf('Normal Approximation and Shannon Limit for r = %.2f, N = %d', r, N));
legend('Normal Approximation', 'Shannon Limit');
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
```