```
% Load H matrix
baseGraphFileName = 'NR_2_6_52';

% coderate {1/4, 1/3, 1/2, 3/5} for 2_6_52 and coderate {1/3, 1/2, 3/5 and 4/5} for
1_5_352
R = [1/4, 1/3, 1/2, 3/5];

% Expand base matrix into full binary parity-check matrix (H matrix)
[B_mat,H_mat,z] = nrldpc_Hmatrix(baseGraphFileName);

[rowsB,colsB] = size(B_mat); kb = colsB - rowsB;

EbNoDb_Vals = 0 : 0.5 : 10;

Nsim = 500;

max_itr = 20;

% Allocate space for graph plot of Bit Error Rate(BER) vs Eb/No(dB)
BER = zeros(length(R), length(EbNoDb_Vals));

% Allocate space for graph plot of Error Probability vs Eb/No(dB)
p_error = zeros(length(R), length(EbNoDb_Vals));

% Allocate space for graph plot of Success Probabilty vs No of Iteration
succRateItr = zeros(length(EbNoDb_Vals), max_itr);

% Index for coderate
ind_cr = 1;


for codeRate = R

    infoBitsCount = kb * z; % Information bits
    k_pc = kb-2; nbRM = ceil(k_pc/codeRate)+2;
    codeLen = nbRM * z; % Encoded bits

    n = codeLen;

    H = H_mat(:,1:codeLen);
    nChecksNotPunctured = rowsB*z - colsB*z + codeLen;
    H = H(1:nChecksNotPunctured,:);

    [row, col] = size(H);

    % Tracker for current Eb/No iteration index
```

```
    ebNoIdx = 1;

    for ebnodb = EbNoDb_Vals

        Ebno = 10 ^ (ebnodb / 10); % Convert Eb/No from dB to linear scale

        sigma = sqrt(1 / (2 * codeRate * Ebno));

        for i = 1 : 1 : Nsim

            msgBits = randi([0 1],[infoBitsCount 1]);

            encodedBits = nrldpc_encode(B_mat,z,msgBits'); % Perform encoding using
the 5G NR LDPC base matrix
            encodedBits = encodedBits(1:codeLen); % Encoded Message

            s = 1 - 2 * encodedBits; % BPSK Modulator

            r = s + sigma * randn(1, n); % AWGN Channel

            % L - Adjacency matrix representing the Tanner graph
            L = r .* H;

            sumRow = r;
            prev_decoded_msg = zeros(size(encodedBits));

            for itr = 1 : 1 : max_itr

                % SPC code
                for chk = 1 : 1 : row
                    ind = find(H(chk, :) ~= 0);
                    [min1, minpos] = min(abs(L(chk, ind)));
                    min2 = min(abs(L(chk, ind([1 : minpos - 1 minpos + 1 : end]))));
                    sgn = sign(L(chk, ind));
                    prod_sgn = prod(sgn);

                    L(chk, ind) = min1 .* prod_sgn;
                    L(chk, ind(minpos)) = min2 .* prod_sgn;
                    L(chk, ind) = sgn .* L(chk, ind);
                end

                % Repetation code
                sumRow = r + sum(L);
                for var = 1 : 1 : col
                    ind = find(H(:, var) ~= 0);
                    L(ind, var) = sumRow(var) - L(ind, var);
                end

                decoded_msg = sumRow < 0;
```

```
                % Verify if the decoded message matches the original encoded
message and if the code rate index is 1
                if(decoded_msg == encodedBits & ind_cr == 1)
                    succRateItr(ebNoIdx, itr) = succRateItr(ebNoIdx, itr) + 1;
                end

                % Below Code is commented out for plotting of
                % p_success vs iteration

                % if((itr ~= 1) & (decoded_msg == prev_decoded_msg))
                %      break;
                % else
                %     prev_decoded_msg = decoded_msg;
                % end
            end

            % Check if the decoding was successful or not
            errCount = sum(decoded_msg ~= encodedBits);
            if(errCount > 0)
                BER(ind_cr, ebNoIdx) = BER(ind_cr, ebNoIdx) + errCount;
                p_error(ind_cr, ebNoIdx) = p_error(ind_cr, ebNoIdx) + 1;
            end

        end

        BER(ind_cr, ebNoIdx) = BER(ind_cr, ebNoIdx) / n / Nsim;
        p_error(ind_cr, ebNoIdx) = p_error(ind_cr, ebNoIdx) / Nsim;

        ebNoIdx = ebNoIdx + 1;
    end

    ind_cr = ind_cr + 1;

end
```

```
% Uncoded BPSK

% Convert SNR from dB to linear scale
Ebno = 10 .^ (EbNoDb_Vals ./ 10);

% Calculate BER for each SNR
ber_uncoded = 0.5 * erfc(sqrt(Ebno));

for i = 1:1:4
    plot(EbNoDb_Vals, BER(i, :), 'DisplayName',sprintf('Rate = %.2f', R(i)),
LineWidth=2);
    hold on;
end
```

```
plot(EbNoDb_Vals, ber_uncoded, 'DisplayName', 'Uncoded BPSK', LineWidth=2)
legend('show','Location','bestoutside');
title('Bit Error Rate vs EbNo (db)');
xlabel('EbNo in db');
ylabel('BER');
hold off;

for i = 1:1:4
    semilogy(EbNoDb_Vals, BER(i, :), 'DisplayName',sprintf('Rate = %.2f', R(i)),
LineWidth=2);
    hold on;
end
semilogy(EbNoDb_Vals, ber_uncoded, 'DisplayName', 'Uncoded BPSK', LineWidth=2)
legend('show','Location','bestoutside');
title('Bit Error Rate vs EbNo (db) with y axis with log scale')
xlabel('EbNo in db');
ylabel('BER in log scale');
hold off;

for i = 1:1:4
    plot(EbNoDb_Vals, p_error(i, :), 'DisplayName',sprintf('Rate = %.2f', R(i)),
LineWidth=2);
    hold on;
end
legend('show','Location','bestoutside');
title('Probability of Decoding Failure vs EbNo (db)');
xlabel('EbNo in db');
ylabel('Probability of Decoding Failure');
hold off;

for i = 1:1:4
    plot(EbNoDb_Vals, 1 - p_error(i, :), 'DisplayName',sprintf('Rate = %.2f',
R(i)), LineWidth=2);
    hold on;
end
legend('show','Location','bestoutside');
title('Probability of Success vs EbNo (db)');
xlabel('EbNo in db');
ylabel('Probability of Success');
hold off;

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

```
hold off;
```

```
function y = mul_sh(x, k)% Function to shift matrix x by k positions
% x - input matrix
% k - number of positions to shift (can be -1)

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

end

% Function to generate the binary H matrix from the base matrix
function [B_mat,H,z] = nrldpc_Hmatrix(BG)

load(sprintf('%s.txt',BG),BG);
B_mat = NR_2_6_52;
[rowsB,colsB] = size(B_mat);
z = 52;
H = zeros(rowsB*z,colsB*z);
Iz = eye(z); I0 = zeros(z);
for kk = 1:rowsB
    tmpvecR = (kk-1)*z+(1:z);
    for kk1 = 1:colsB
        tmpvecC = (kk1-1)*z+(1:z);
        if B_mat(kk,kk1) == -1
            H(tmpvecR,tmpvecC) = I0;
        else
            H(tmpvecR,tmpvecC) = circshift(Iz,-B_mat(kk,kk1));
        end
    end
end

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

end

% Function to encode message using Base Matrix
function cword = nrldpc_encode(B_mat,z,msg)

%B_mat: base matrix
%z: expansion factor
%msg: message vector, len = (cols-rows)*z
%cword: codeword vector, len = cols*z
```

```
[m,n] = size(B_mat);

cword = zeros(1,n*z);
cword(1:(n-m)*z) = msg;


%double-diagonal encoding
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_mat(i,j)),2);
    end
end
if B_mat(2,n-m+1) == -1
    p1_sh = B_mat(3,n-m+1);
else
    p1_sh = B_mat(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_mat(i,j)),2);
    end
    cword((n-m+i)*z+1:(n-m+i+1)*z) = temp;
end


% Calculate the remaining parity bits
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_mat(i,j)),2);
    end
    cword((n-m+i-1)*z+1:(n-m+i)*z) = temp;
end

end
```