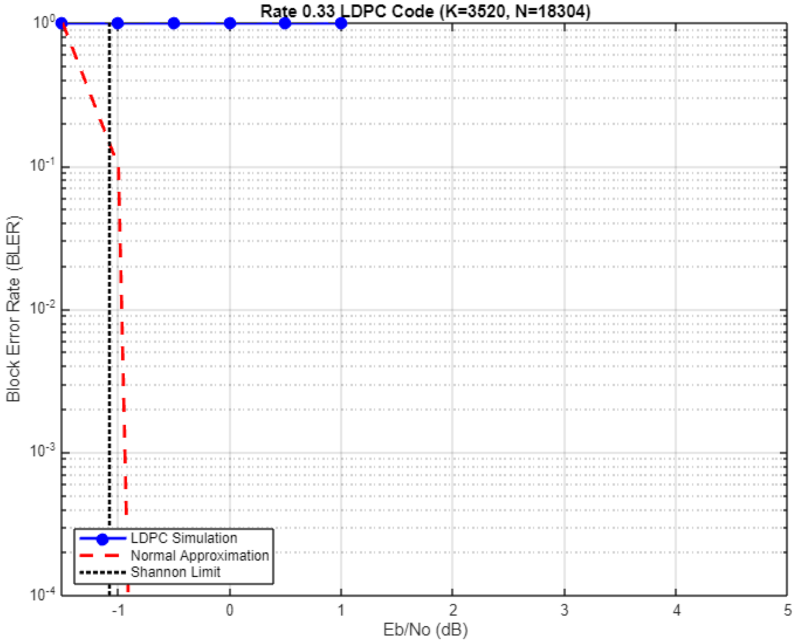
```
clear all; close all; clc;

%% Simulation Parameters (Optimized for NR_1_5_352)
baseGraph5GNR = 'NR_1_5_352';
codeRates = [1/3, 1/2, 3/5, 4/5]; % Supported code rates for BG1
EbN0dB_vec = -1.50:0.50:5.00;
max_iterations = 20;
Nsim = 10;
z = 352; % Expansion factor for NR_1_5_352
K = 10*z; % Information bits (22*52 was for NR_2_6_52)
N = 52*z; % Codeword length (68*52 was for NR_2_6_52)

%% Initialize Results Storage
results = struct();
for cr = 1:length(codeRates)
    results(cr).rate = codeRates(cr);
    results(cr).BER = zeros(size(EbN0dB_vec));
    results(cr).FER = zeros(size(EbN0dB_vec));
    results(cr).iteration_success = zeros(max_iterations, length(EbN0dB_vec));
    results(cr).Pc = zeros(size(EbN0dB_vec));
end

%% Theoretical Benchmarks
shannon_limit = 10*log10((2.^codeRates-1)./codeRates);

for cr = 1:length(codeRates)
    c_r = codeRates(cr);
    P_NA = zeros(size(EbN0dB_vec));
    for snr_idx = 1:length(EbN0dB_vec)
        EbN0dB = EbN0dB_vec(snr_idx);
        EbN0 = 10^(EbN0dB/10);
        P = c_r * EbN0;
        C = log2(1 + P);
        V = (log2(exp(1)))^2 * (P*(P + 2))/(2*(P + 1)^2);
        argument = sqrt(N/V) * (C - c_r + log2(N)/(2*N));
        P_NA(snr_idx) = qfunc(argument);
    end
    results(cr).P_NA = P_NA;
end

%% Main Simulation Loop
for cr_idx = 1:length(codeRates)
    c_r = codeRates(cr_idx);
    fprintf('\nSimulating code rate %.2f (%d/%d)\n', c_r, cr_idx,
length(codeRates));
```

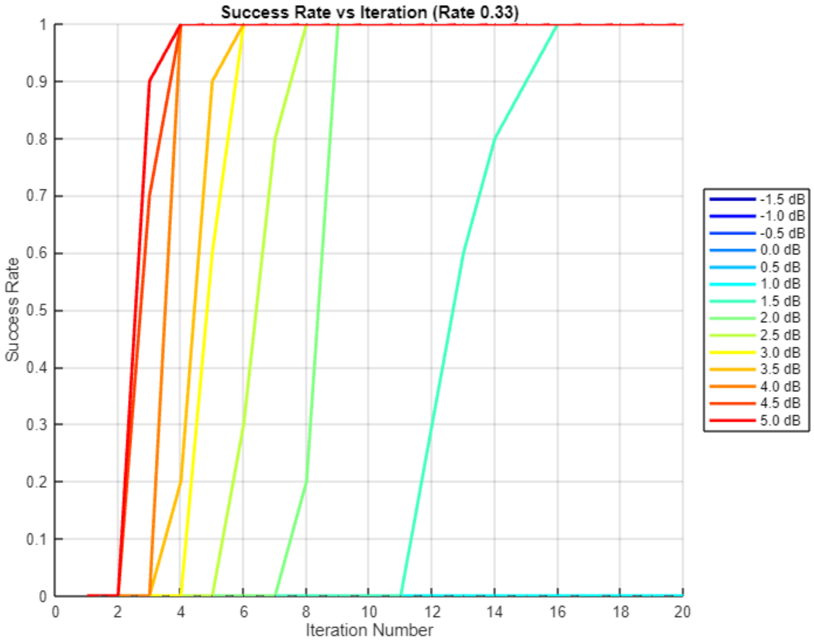
```
    % Generate parity check matrix
    [B, Hfull, z] = nrldpc_Hmatrix_352(baseGraph5GNR);
    [mb, nb] = size(B);
    kb = nb - mb;
    kNumInfoBits = kb * z;

    % Rate matching
    k_pc = kb-2;
    nbRM = ceil(k_pc/c_r)+2;
    nBlockLength = nbRM * z;
    H = Hfull(:,1:nBlockLength);
    nChecksNotPunctured = mb*z - nb*z + nBlockLength;
    H = sparse(H(1:nChecksNotPunctured,:)); % Use sparse matrix

    % Build memory-efficient Tanner graph
    [VN_to_CN_map, CN_to_VN_map] = build_tanner_graph_sparse(H);

    for snr_idx = 1:length(EbN0dB_vec)
        EbN0dB = EbN0dB_vec(snr_idx);
        EbN0 = 10^(EbN0dB/10);
        EsN0 = c_r * EbN0;
        noise_var = 1/(2*EsN0);

        bit_errors = 0;
        frame_errors = 0;
        iteration_success = zeros(1, max_iterations);
        success_count = 0;

        parfor sim = 1:Nsim % Parallel processing
            % Generate and encode message
            msg_bits = randi([0 1], 1, kNumInfoBits);
            cword = nrldpc_encode_352(B, z, msg_bits);
            cword = cword(1:nBlockLength);

            % BPSK modulation and AWGN channel
            tx_signal = 1 - 2*cword;
            noise = sqrt(noise_var) * randn(1, nBlockLength);
            rx_signal = tx_signal + noise;

            % LLR calculation and decoding
            llr = (2/noise_var) * rx_signal;
            [decoded_bits, iter_hist, final_success] = ...
                ldpc_decode_memory_optimized(llr, H, VN_to_CN_map,
CN_to_VN_map, max_iterations, msg_bits);
```

```
             % Update statistics
             iteration_success = iteration_success + iter_hist;
             success_count = success_count + final_success;
             bit_errors = bit_errors + sum(decoded_bits(1:kNumInfoBits) ~=
msg_bits);
             frame_errors = frame_errors + (sum(decoded_bits(1:kNumInfoBits) ~=
msg_bits) > 0);
         end

         % Store results
         results(cr_idx).BER(snr_idx) = bit_errors / (Nsim * kNumInfoBits);
         results(cr_idx).FER(snr_idx) = frame_errors / Nsim;
         results(cr_idx).iteration_success(:,snr_idx) = iteration_success' /
Nsim;
         results(cr_idx).Pc(snr_idx) = success_count / Nsim;

         fprintf('  SNR %.1f dB: FER=%.2e, Pc=%.2f\n', EbN0dB,
results(cr_idx).FER(snr_idx), results(cr_idx).Pc(snr_idx));
     end

     %% Plotting
     figure(cr_idx*10 + 1);
     set(gcf, 'Position', [100, 100, 800, 600]);
     semilogy(EbN0dB_vec, results(cr_idx).FER, 'b-o', 'LineWidth', 2,
'MarkerFaceColor', 'b', 'DisplayName', 'LDPC Simulation');
     hold on;
     semilogy(EbN0dB_vec, results(cr_idx).P_NA, 'r--', 'LineWidth', 2,
'DisplayName', 'Normal Approximation');
     semilogy([shannon_limit(cr_idx), shannon_limit(cr_idx)], [1e-4 1], 'k:',
'LineWidth', 2, 'DisplayName', 'Shannon Limit');
     hold off;
     grid on;
     xlabel('Eb/No (dB)'); ylabel('Block Error Rate (BLER)');
     title(sprintf('Rate %.2f LDPC Code (K=%d, N=%d)', c_r, K, N));
     legend('Location', 'southwest');
     ylim([1e-4 1]); xlim([min(EbN0dB_vec) max(EbN0dB_vec)]);

     figure(cr_idx*10 + 2);
     set(gcf, 'Position', [100, 100, 800, 600]);
     hold on;
     colors = jet(length(EbN0dB_vec));
     for snr_idx = 1:length(EbN0dB_vec)
         plot(1:max_iterations, results(cr_idx).iteration_success(:,snr_idx),
...
```

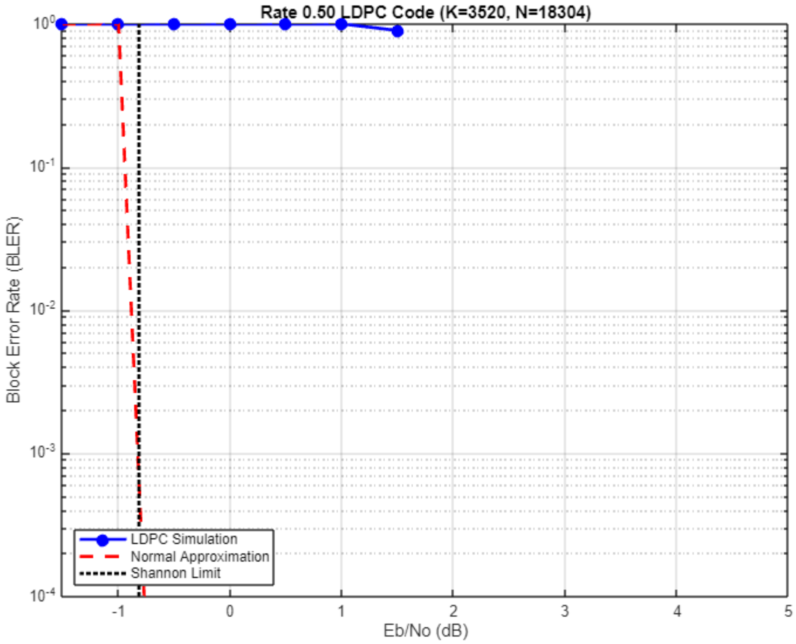
```
            'Color', colors(snr_idx,:), 'LineWidth', 2, ...
            'DisplayName', sprintf('%.1f dB', EbN0dB_vec(snr_idx)));
    end
    hold off;
    grid on;
    xlabel('Iteration Number'); ylabel('Success Rate');
    title(sprintf('Success Rate vs Iteration (Rate %.2f)', c_r));
    legend('Location', 'eastoutside');
    ylim([0 1]);
end
```

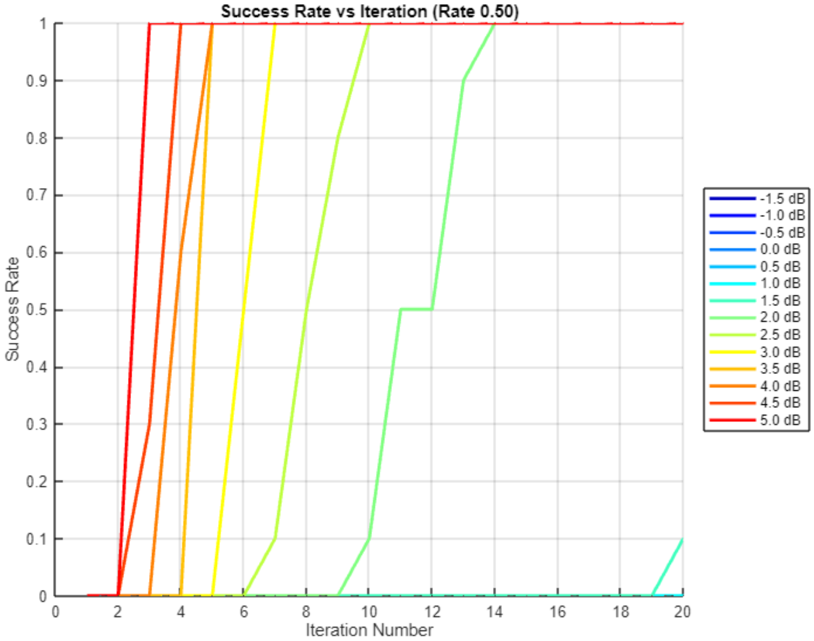
```
Simulating code rate 0.33 (1/4)
  SNR -1.5 dB: FER=1.00e+00, Pc=0.00
  SNR -1.0 dB: FER=1.00e+00, Pc=0.00
  SNR -0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 0.0 dB: FER=1.00e+00, Pc=0.00
  SNR 0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 1.0 dB: FER=1.00e+00, Pc=0.00
  SNR 1.5 dB: FER=0.00e+00, Pc=1.00
  SNR 2.0 dB: FER=0.00e+00, Pc=1.00
  SNR 2.5 dB: FER=0.00e+00, Pc=1.00
  SNR 3.0 dB: FER=0.00e+00, Pc=1.00
  SNR 3.5 dB: FER=0.00e+00, Pc=1.00
  SNR 4.0 dB: FER=0.00e+00, Pc=1.00
  SNR 4.5 dB: FER=0.00e+00, Pc=1.00
  SNR 5.0 dB: FER=0.00e+00, Pc=1.00
```



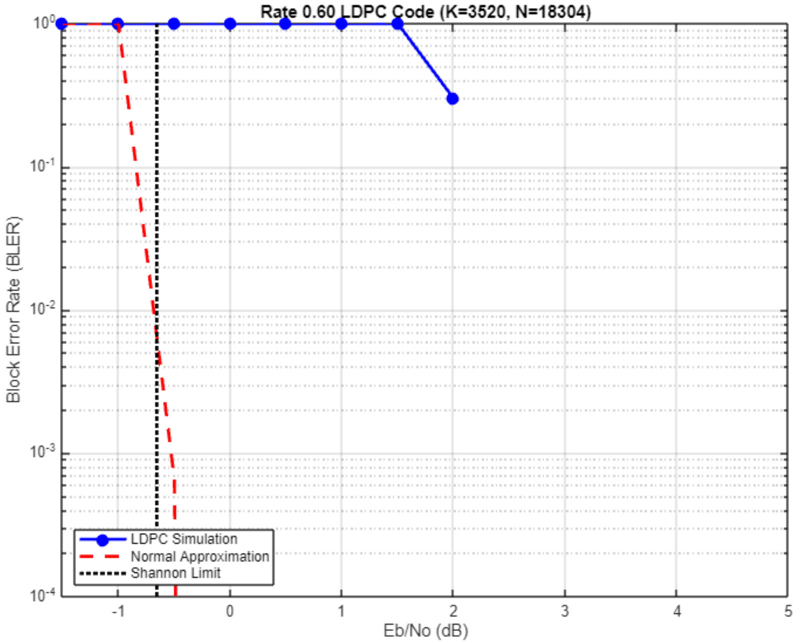


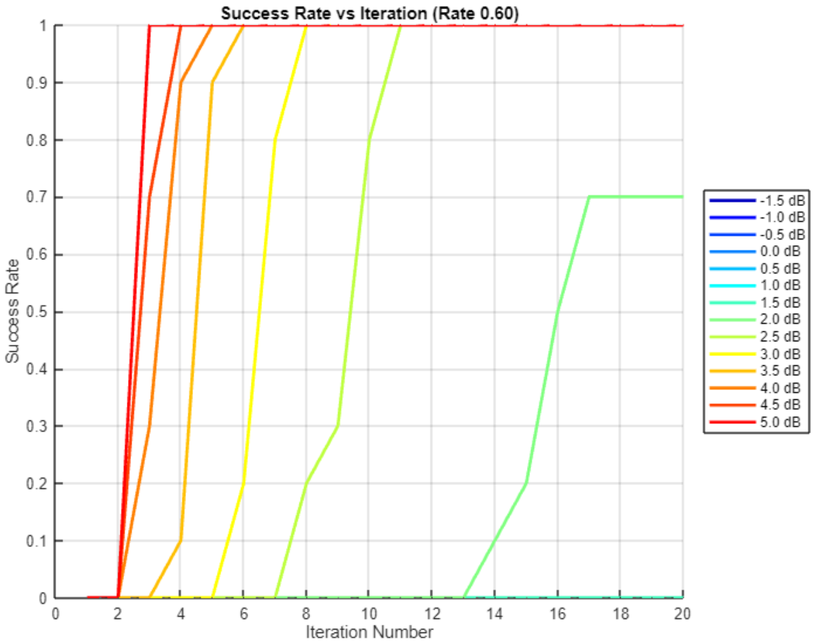
```
Simulating code rate 0.50 (2/4)
  SNR -1.5 dB: FER=1.00e+00, Pc=0.00
  SNR -1.0 dB: FER=1.00e+00, Pc=0.00
  SNR -0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 0.0 dB: FER=1.00e+00, Pc=0.00
  SNR 0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 1.0 dB: FER=1.00e+00, Pc=0.00
  SNR 1.5 dB: FER=9.00e-01, Pc=0.10
  SNR 2.0 dB: FER=0.00e+00, Pc=1.00
  SNR 2.5 dB: FER=0.00e+00, Pc=1.00
  SNR 3.0 dB: FER=0.00e+00, Pc=1.00
  SNR 3.5 dB: FER=0.00e+00, Pc=1.00
  SNR 4.0 dB: FER=0.00e+00, Pc=1.00
  SNR 4.5 dB: FER=0.00e+00, Pc=1.00
  SNR 5.0 dB: FER=0.00e+00, Pc=1.00
```



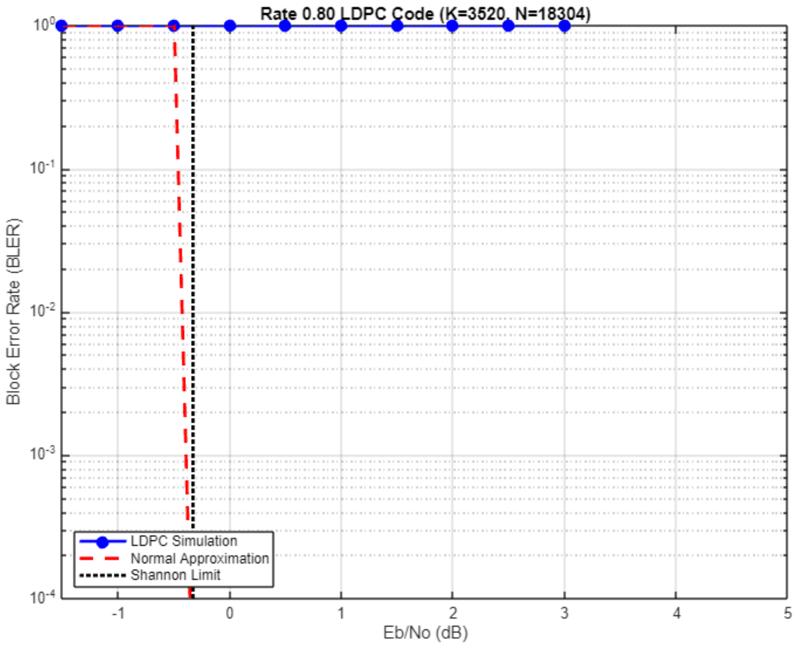


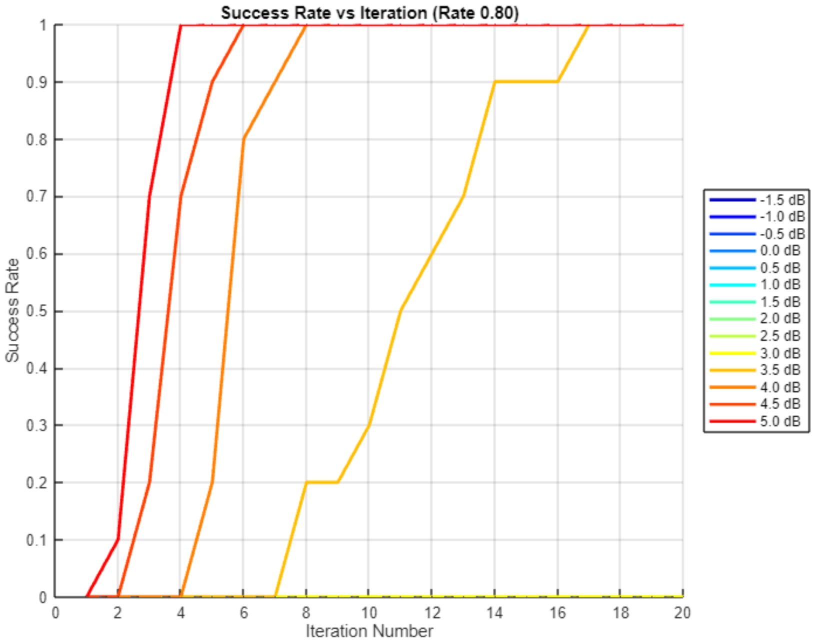
```
Simulating code rate 0.60 (3/4)
  SNR -1.5 dB: FER=1.00e+00, Pc=0.00
  SNR -1.0 dB: FER=1.00e+00, Pc=0.00
  SNR -0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 0.0 dB: FER=1.00e+00, Pc=0.00
  SNR 0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 1.0 dB: FER=1.00e+00, Pc=0.00
  SNR 1.5 dB: FER=1.00e+00, Pc=0.00
  SNR 2.0 dB: FER=3.00e-01, Pc=0.70
  SNR 2.5 dB: FER=0.00e+00, Pc=1.00
  SNR 3.0 dB: FER=0.00e+00, Pc=1.00
  SNR 3.5 dB: FER=0.00e+00, Pc=1.00
  SNR 4.0 dB: FER=0.00e+00, Pc=1.00
  SNR 4.5 dB: FER=0.00e+00, Pc=1.00
  SNR 5.0 dB: FER=0.00e+00, Pc=1.00
```





```
Simulating code rate 0.80 (4/4)
  SNR -1.5 dB: FER=1.00e+00, Pc=0.00
  SNR -1.0 dB: FER=1.00e+00, Pc=0.00
  SNR -0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 0.0 dB: FER=1.00e+00, Pc=0.00
SNR 0.5 dB: FER=1.00e+00, Pc=0.00
SNR 1.0 dB: FER=1.00e+00, Pc=0.00
SNR 1.5 dB: FER=1.00e+00, Pc=0.00
SNR 2.0 dB: FER=1.00e+00, Pc=0.00
SNR 2.5 dB: FER=1.00e+00, Pc=0.00
SNR 3.0 dB: FER=1.00e+00, Pc=0.00
SNR 3.5 dB: FER=0.00e+00, Pc=1.00
SNR 4.0 dB: FER=0.00e+00, Pc=1.00
SNR 4.5 dB: FER=0.00e+00, Pc=1.00
SNR 5.0 dB: FER=0.00e+00, Pc=1.00
```





```
Unrecognized function or variable 'Simulating'.
```

```matlab
%% Combined Performance Plot
figure(100);
set(gcf, 'Position', [100, 100, 900, 700]);
hold on;
colors = lines(length(codeRates));
markers = {'o', 's', 'd', '^'};

for cr = 1:length(codeRates)
    semilogy(EbN0dB_vec, results(cr).FER, ...
        'Color', colors(cr,:), 'Marker', markers{cr}, ...
        'LineWidth', 2, 'MarkerFaceColor', colors(cr,:), ...
        'DisplayName', sprintf('Rate %.2f LDPC', codeRates(cr)));

    semilogy(EbN0dB_vec, results(cr).P_NA, '--', ...
        'Color', colors(cr,:), 'LineWidth', 1.5, ...
        'DisplayName', sprintf('Rate %.2f NA', codeRates(cr)));
end

for cr = 1:length(codeRates)
    plot([shannon_limit(cr), shannon_limit(cr)], [1e-4 1], ':', ...
        'Color', colors(cr,:), 'LineWidth', 1.5, ...
        'DisplayName', sprintf('Rate %.2f Shannon', codeRates(cr)));
end

hold off;
grid on;
xlabel('Eb/No (dB)'); ylabel('Block Error Rate (BLER)');
title('LDPC Performance vs Theoretical Benchmarks');
legend('Location', 'southwest');
set(gca, 'FontSize', 12);
ylim([1e-4 1]); xlim([min(EbN0dB_vec) max(EbN0dB_vec)]);

%% Memory-Optimized Functions for NR_1_5_352
function [B,H,z] = nrldpc_Hmatrix_352(BG)
    % Load the base graph file
    load('NR_1_5_352.txt', 'NR_1_5_352');
    B = NR_1_5_352;
    [mb,nb] = size(B);
    z = 352;

    % Create sparse matrix directly
    [rows, cols, shifts] = find(B ~= -1);
    num_entries = length(rows);

    % Pre-calculate total number of non-zero entries
    total_nnz = num_entries * z;
    row_inds = zeros(total_nnz, 1);
    col_inds = zeros(total_nnz, 1);

    % Build indices for sparse matrix
    current_pos = 1;
    for idx = 1:num_entries
        r = rows(idx);
        c = cols(idx);
        s = B(r,c);

        % Calculate base positions
        base_row = (r-1)*z;
        base_col = (c-1)*z;

        % Create shifted indices for this block
        block_rows = (1:z) + base_row;
        block_cols = mod((1:z) + s - 1, z) + 1 + base_col;

        % Store indices
        end_pos = current_pos + z - 1;
        row_inds(current_pos:end_pos) = block_rows;
        col_inds(current_pos:end_pos) = block_cols;

        current_pos = end_pos + 1;
    end

    H = sparse(row_inds, col_inds, 1, mb*z, nb*z);
end

function [VN_to_CN_map, CN_to_VN_map] = build_tanner_graph_sparse(H)
    [num_CNs, num_VNs] = size(H);
    [rows, cols] = find(H);

    % Pre-allocate
    VN_to_CN_map = cell(num_VNs,1);
    CN_to_VN_map = cell(num_CNs,1);

    % Build VN connections
    for vn = 1:num_VNs
        VN_to_CN_map{vn} = rows(cols == vn)';
    end
```

```
     % Build CN connections
     for cn = 1:num_CNs
         CN_to_VN_map{cn} = cols(rows == cn)';
     end
 end

 function [decoded_bits, iteration_history, final_success] = ...
     ldpc_decode_memory_optimized(llr, H, VN_to_CN_map, CN_to_VN_map, max_iter,
original_msg)

     num_VNs = length(VN_to_CN_map);
     num_CNs = length(CN_to_VN_map);
     kNumInfoBits = length(original_msg);

     % Use cell arrays for message storage
     VN_to_CN_msgs = cell(num_VNs,1);
     CN_to_VN_msgs = cell(num_CNs,1);

     % Initialize VN messages
     for vn = 1:num_VNs
         cn_list = VN_to_CN_map{vn};
         VN_to_CN_msgs{vn} = llr(vn) * ones(size(cn_list));
     end

     iteration_history = zeros(1, max_iter);
     final_success = 0;

     for iter = 1:max_iter
         % Check node updates
         for cn = 1:num_CNs
             vn_list = CN_to_VN_map{cn};
             incoming = zeros(size(vn_list));

             % Collect incoming messages
             for i = 1:length(vn_list)
                 vn = vn_list(i);
                 cn_pos_in_vn = find(VN_to_CN_map{vn} == cn, 1);
                 incoming(i) = VN_to_CN_msgs{vn}(cn_pos_in_vn);
             end

             sign_prod = prod(sign(incoming));
             abs_incoming = abs(incoming);

             % Compute outgoing messages
             if isempty(CN_to_VN_msgs{cn})
                 CN_to_VN_msgs{cn} = zeros(size(vn_list));
             end

             for i = 1:length(vn_list)
                 min1 = min(abs_incoming([1:i-1, i+1:end]));
                 CN_to_VN_msgs{cn}(i) = 0.8 * sign_prod * sign(incoming(i)) *
min1;
             end
         end

         % Variable node updates and hard decision
         decoded_bits = zeros(1, num_VNs);
         for vn = 1:num_VNs
             cn_list = VN_to_CN_map{vn};
             total = llr(vn);

             % Sum all incoming messages
             for i = 1:length(cn_list)
                 cn = cn_list(i);
                 vn_pos_in_cn = find(CN_to_VN_map{cn} == vn, 1);
                 total = total + CN_to_VN_msgs{cn}(vn_pos_in_cn);
             end

             % Update outgoing messages
             for i = 1:length(cn_list)
                 cn = cn_list(i);
                 vn_pos_in_cn = find(CN_to_VN_map{cn} == vn, 1);
                 VN_to_CN_msgs{vn}(i) = total - CN_to_VN_msgs{cn}(vn_pos_in_cn);
             end

             decoded_bits(vn) = (total < 0);
         end

         % Check for success
         current_success = isequal(decoded_bits(1:kNumInfoBits), original_msg);
         iteration_history(iter) = current_success;

         if current_success
             final_success = 1;
             iteration_history(iter+1:end) = 1;
             break;
         end
     end
 end
```

```matlab
function cword = nrldpc_encode_352(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

    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);

    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
```