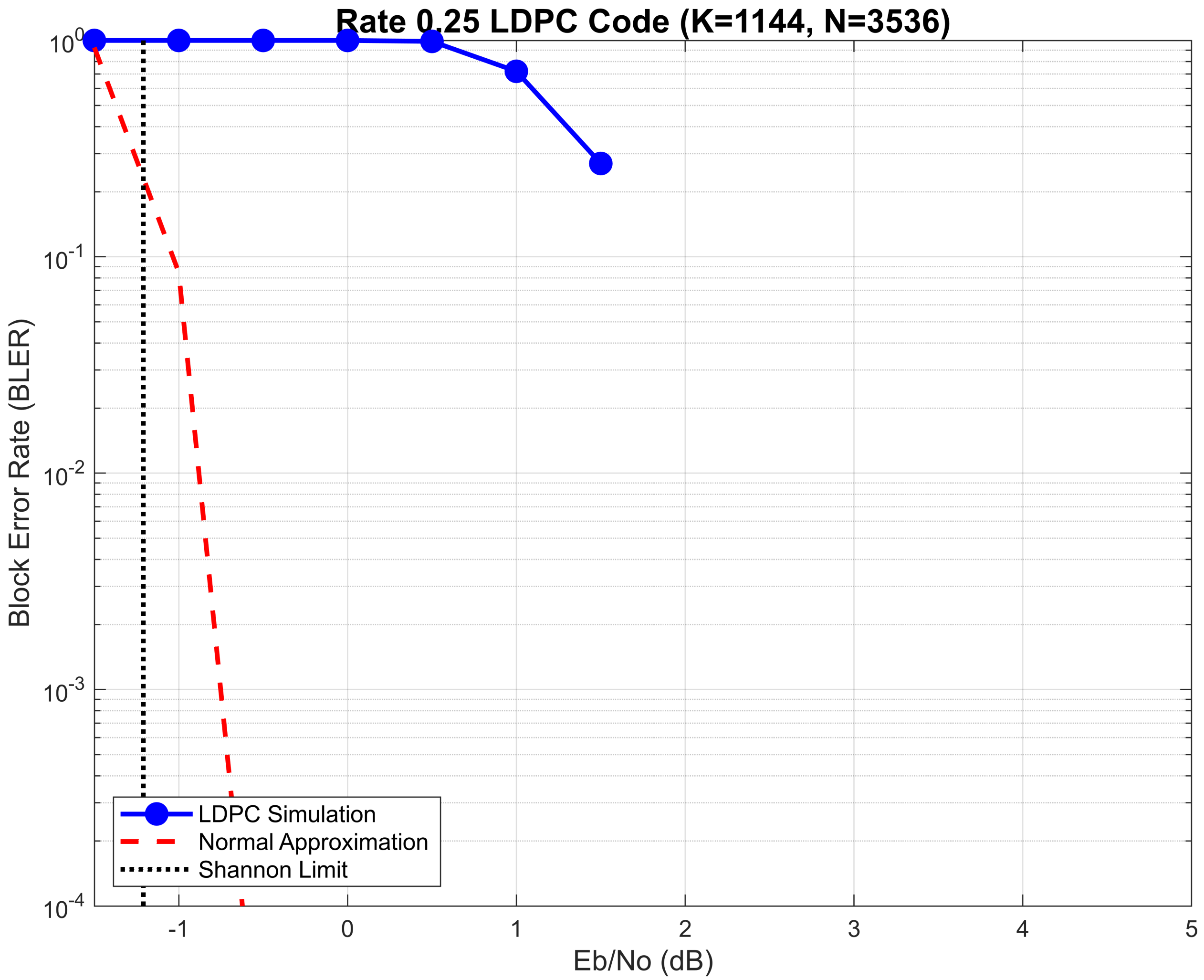
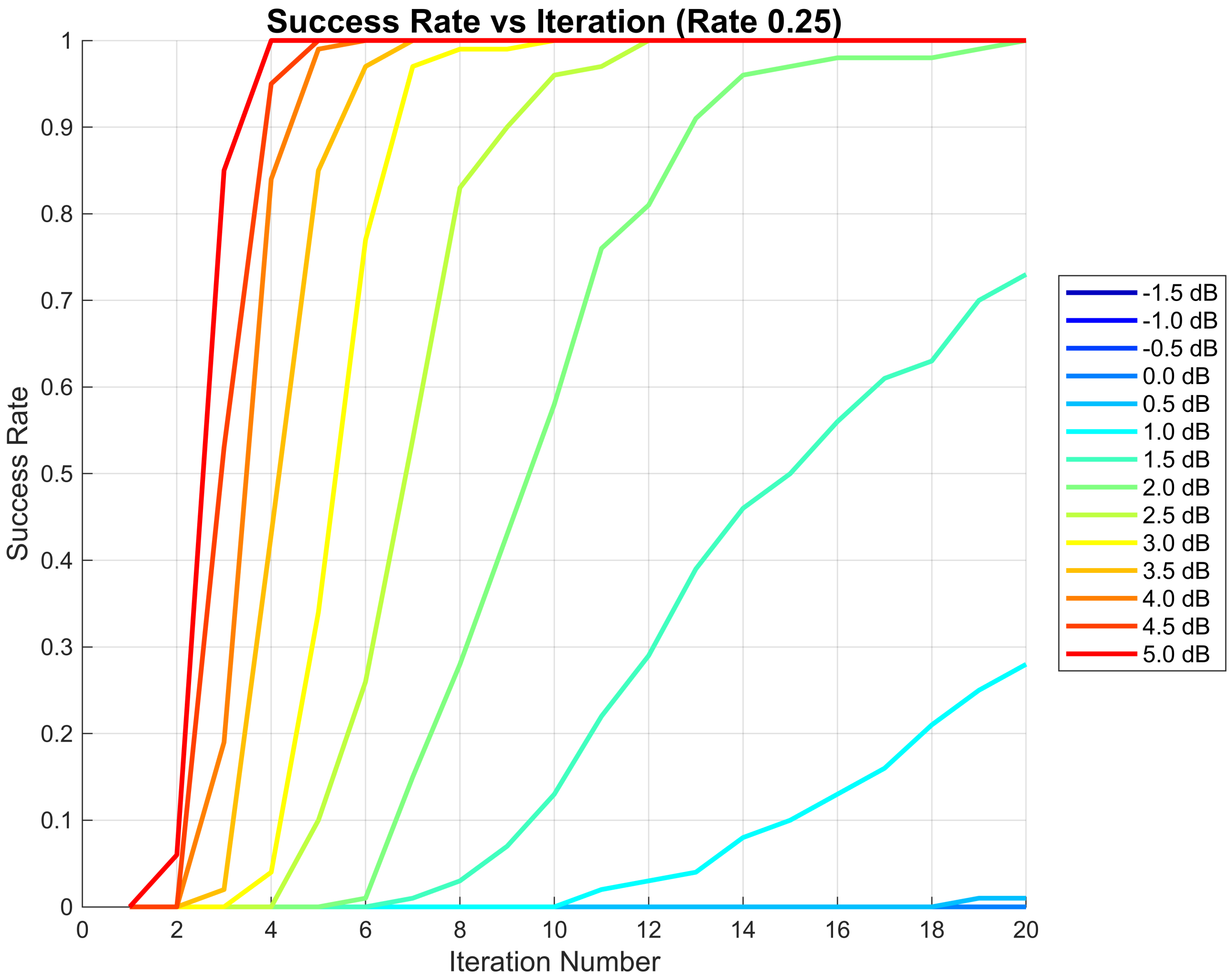
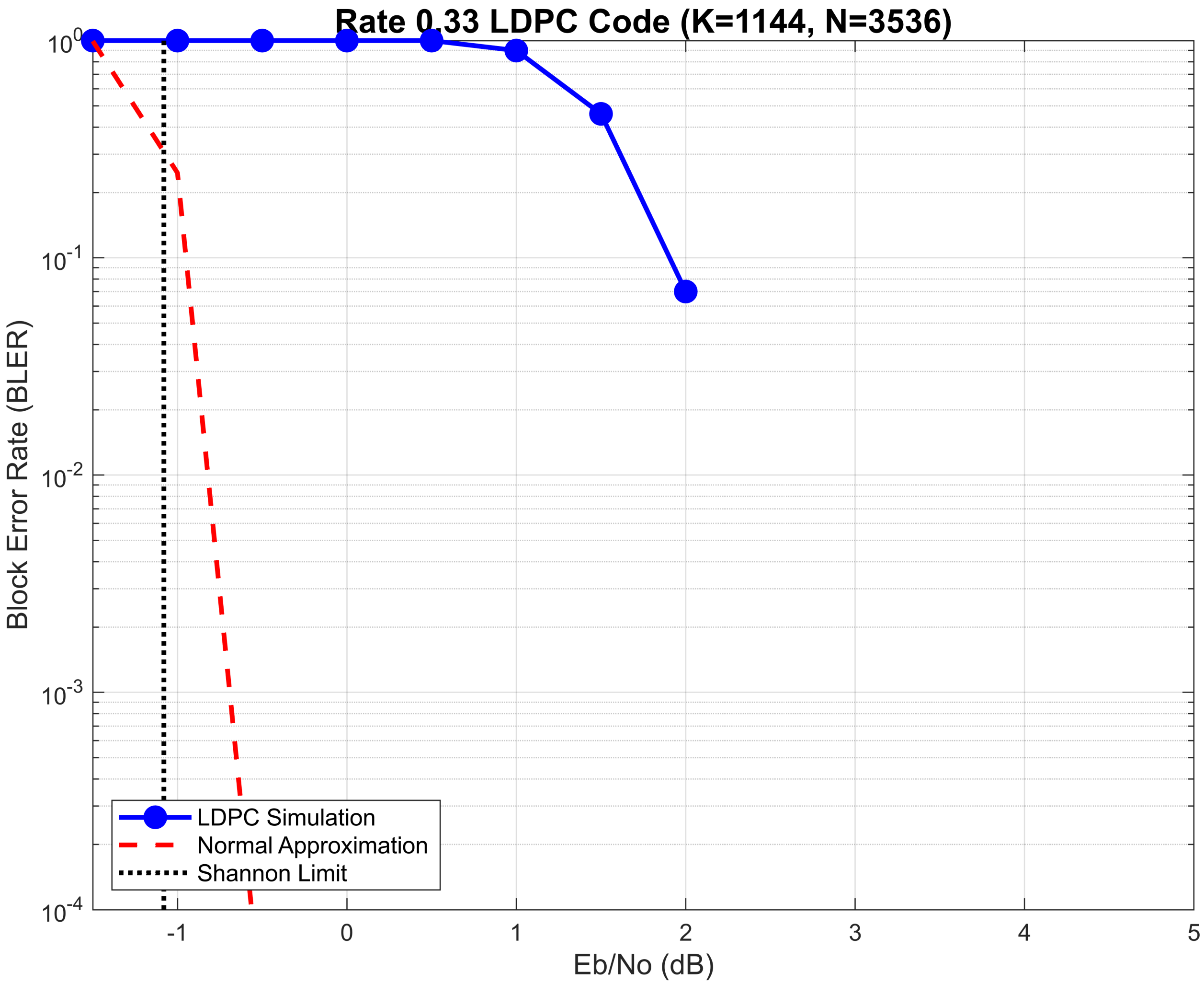
```
clear all; close all; clc;

%% Simulation Parameters
baseGraph5GNR = 'NR_2_6_52';
codeRates = [1/4, 1/3, 1/2, 3/5]; % Code rates to simulate
EbN0dB_vec = -1.50:0.50:5.00;
max_iterations = 20; % Maximum decoding iterations
Nsim = 100;
K = 22*52; % Information bits
N = 68*52; % Codeword length

%% 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)); % Probability of successful decoding
end

%% Calculate Theoretical Benchmarks (Shannon Limit and Normal Approximation)
shannon_limit = 10*log10((2.^codeRates-1)./codeRates); % In dB

% Pre-calculate Normal Approximation for each code rate
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(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 = H(1:nChecksNotPunctured,:);

    % Build Tanner graph
    [VN_to_CN_map, CN_to_VN_map] = build_tanner_graph(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; % For Pc calculation

        for sim = 1:Nsim
            % Generate and encode message
            msg_bits = randi([0 1], 1, kNumInfoBits);
            cword = nrldpc_encode(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_c332(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;

            % Calculate errors
            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; % C.3.2 Pc(imax)

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

    %% Generate Individual Plots for Each Code Rate

    % Plot 1: Decoding Error Probability vs Eb/No with theoretical curves
    figure(cr_idx*10 + 1);
    set(gcf, 'Position', [100, 100, 800, 600]);

    % Plot simulation results
    semilogy(EbN0dB_vec, results(cr_idx).FER, 'b-o', ...
        'LineWidth', 2, 'MarkerFaceColor', 'b', 'MarkerSize', 8, ...
        'DisplayName', 'LDPC Simulation');
    hold on;

    % Plot Normal Approximation
    semilogy(EbN0dB_vec, results(cr_idx).P_NA, 'r--', ...
        'LineWidth', 2, 'DisplayName', 'Normal Approximation');

    % Plot Shannon limit
    shannon_line = ones(size(EbN0dB_vec)) * 0.5; % Placeholder for vertical line
    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)', 'FontSize', 12);
    ylabel('Block Error Rate (BLER)', 'FontSize', 12);
    title(sprintf('Rate %.2f LDPC Code (K=%d, N=%d)', c_r, K, N), 'FontSize', 14);
    legend('Location', 'southwest', 'FontSize', 10);
    ylim([1e-4 1]);
    xlim([min(EbN0dB_vec) max(EbN0dB_vec)]);

    % Plot 2: Success Rate vs Iteration Number (unchanged)
    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', 'FontSize', 12);
    ylabel('Success Rate', 'FontSize', 12);
    title(sprintf('Success Rate vs Iteration (Rate %.2f)', c_r), 'FontSize', 14);
    legend('Location', 'eastoutside', 'FontSize', 10);
    ylim([0 1]);
end
```

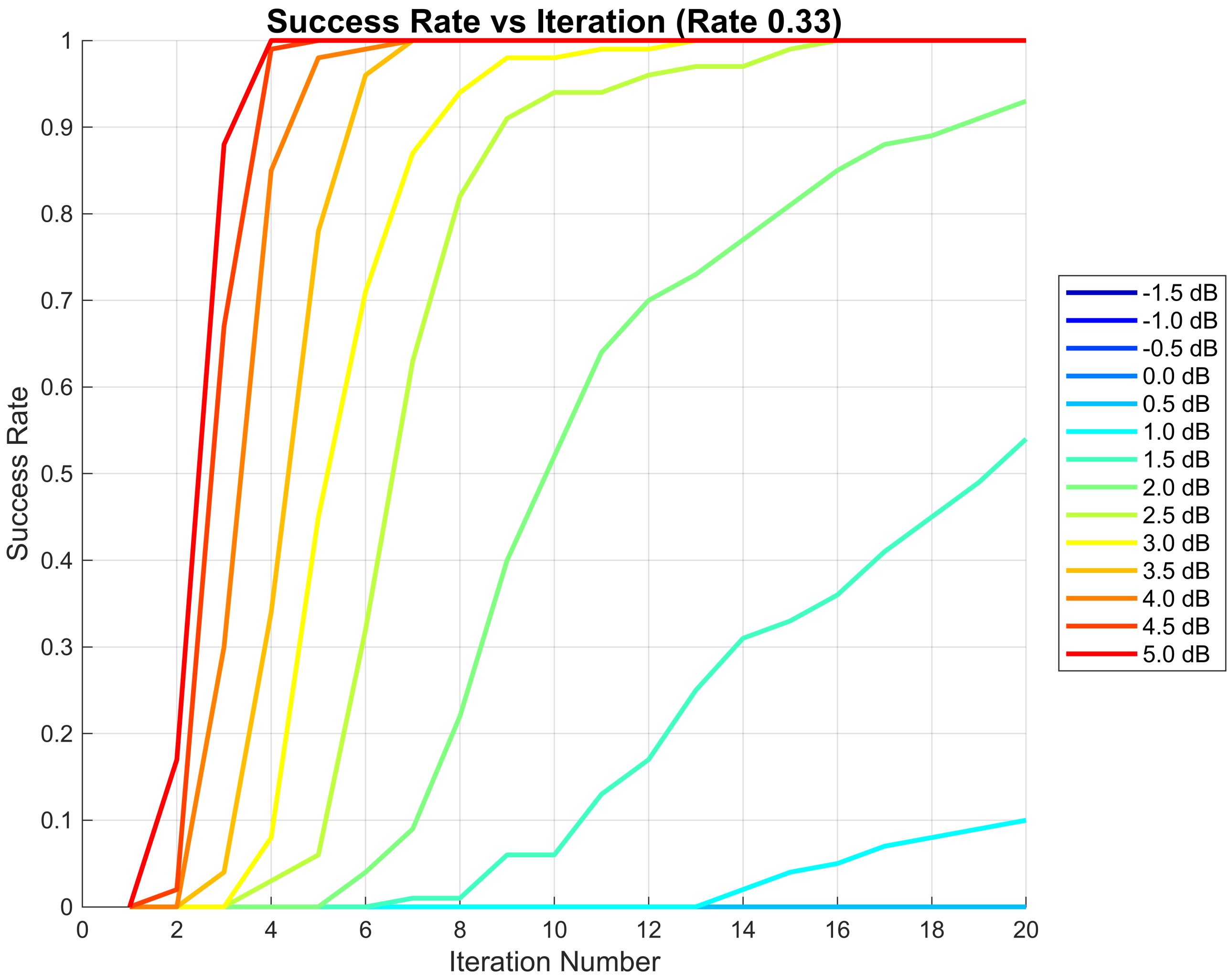
```
Simulating code rate 0.25 (1/4)
  SNR -1.5 dB: FER=1.00e+00, Pc=0.00
  SNR -1.0 dB: FER=1.00e+00, Pc=0.00
  SNR -0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 0.0 dB: FER=1.00e+00, Pc=0.00
  SNR 0.5 dB: FER=9.90e-01, Pc=0.01
  SNR 1.0 dB: FER=7.20e-01, Pc=0.28
  SNR 1.5 dB: FER=2.70e-01, Pc=0.73
  SNR 2.0 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
```


Rate 0.25 LDPC Code (K=1144, N=3536)
Block Error Rate (BLER)
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
-1
0
1
2
3
4
5
Eb/No (dB)
LDPC Simulation
Normal Approximation
Shannon Limit

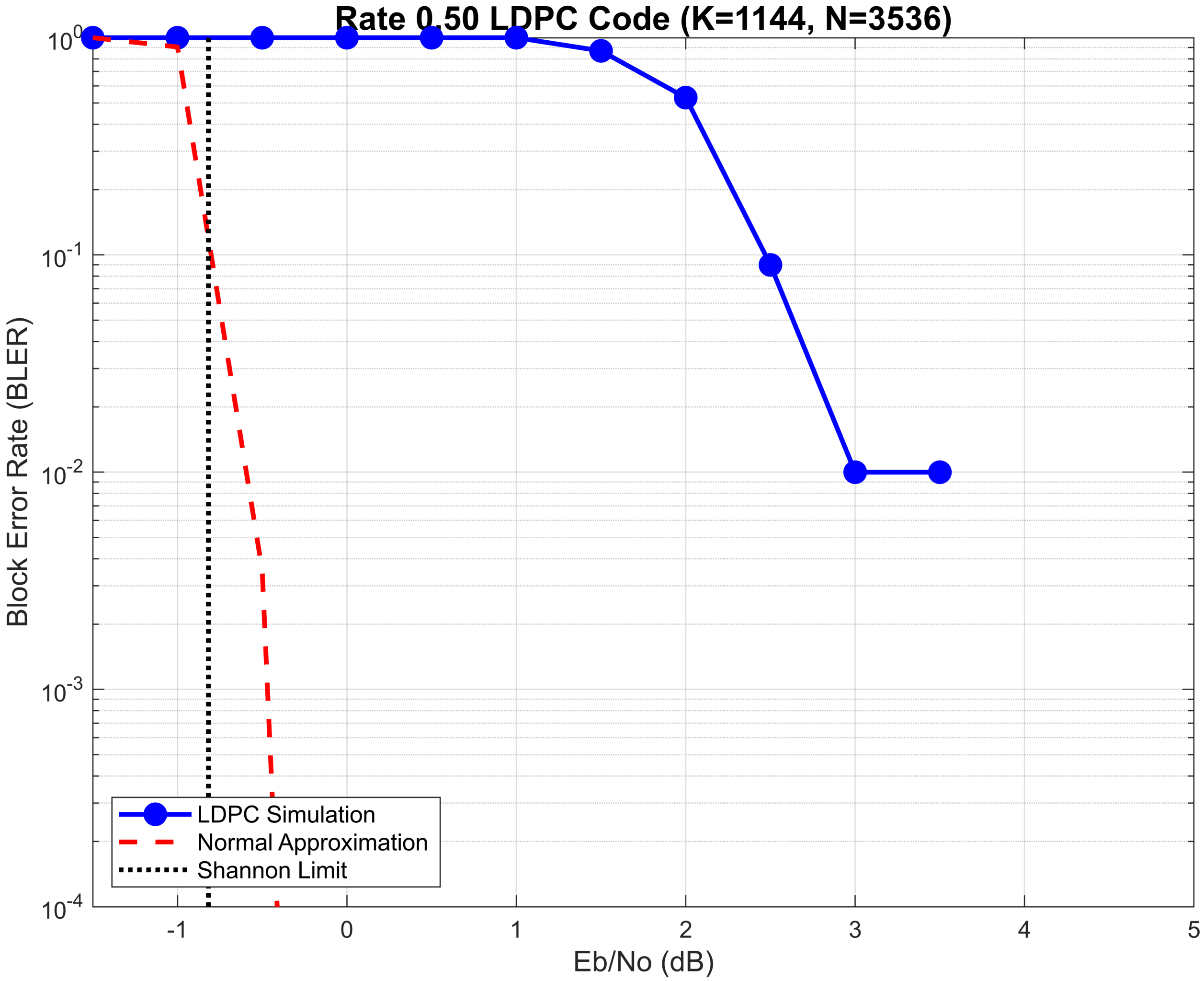


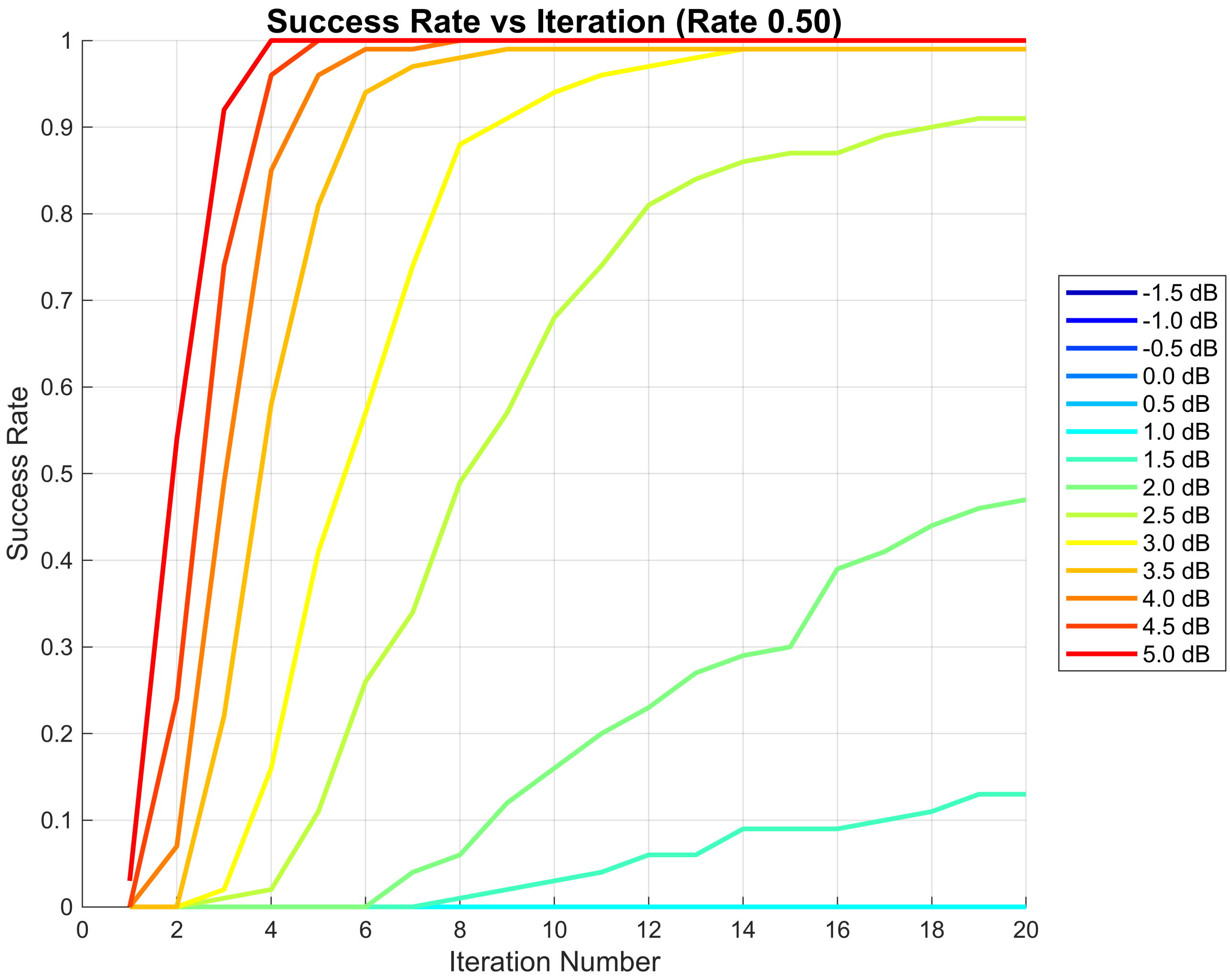
```
Simulating code rate 0.33 (2/4)
  SNR -1.5 dB: FER=1.00e+00, Pc=0.00
  SNR -1.0 dB: FER=1.00e+00, Pc=0.00
  SNR -0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 0.0 dB: FER=1.00e+00, Pc=0.00
  SNR 0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 1.0 dB: FER=9.00e-01, Pc=0.10
  SNR 1.5 dB: FER=4.60e-01, Pc=0.54
  SNR 2.0 dB: FER=7.00e-02, Pc=0.93
  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
```


Rate 0.33 LDPC Code (K=1144, N=3536)
Block Error Rate (BLER)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
-1
0
1
2
3
4
5
Eb/No (dB)
LDPC Simulation
Normal Approximation
Shannon Limit

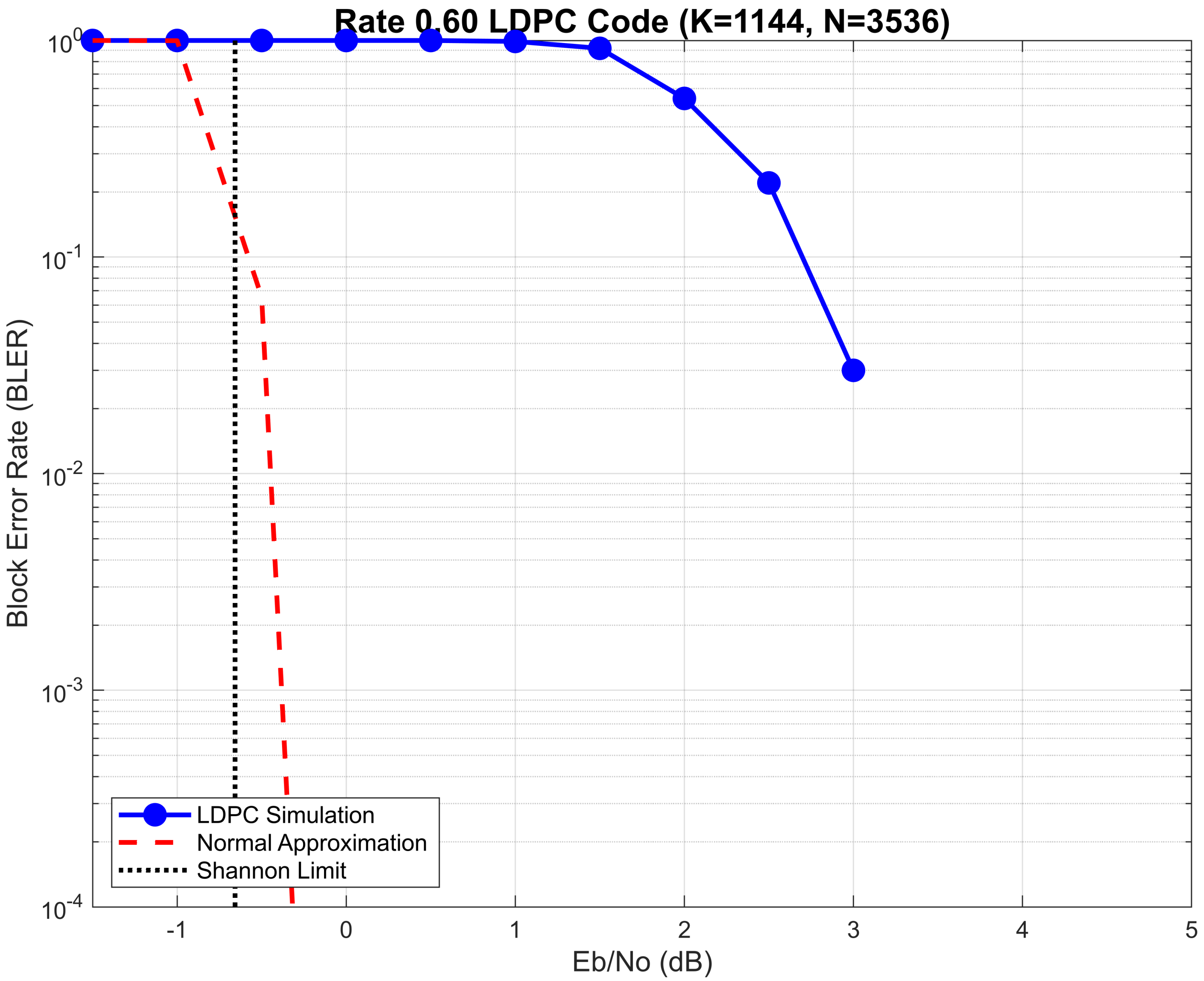


```
Simulating code rate 0.50 (3/4)
  SNR -1.5 dB: FER=1.00e+00, Pc=0.00
  SNR -1.0 dB: FER=1.00e+00, Pc=0.00
  SNR -0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 0.0 dB: FER=1.00e+00, Pc=0.00
  SNR 0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 1.0 dB: FER=1.00e+00, Pc=0.00
  SNR 1.5 dB: FER=8.70e-01, Pc=0.13
  SNR 2.0 dB: FER=5.30e-01, Pc=0.47
  SNR 2.5 dB: FER=9.00e-02, Pc=0.91
  SNR 3.0 dB: FER=1.00e-02, Pc=0.99
  SNR 3.5 dB: FER=1.00e-02, Pc=0.99
  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
```

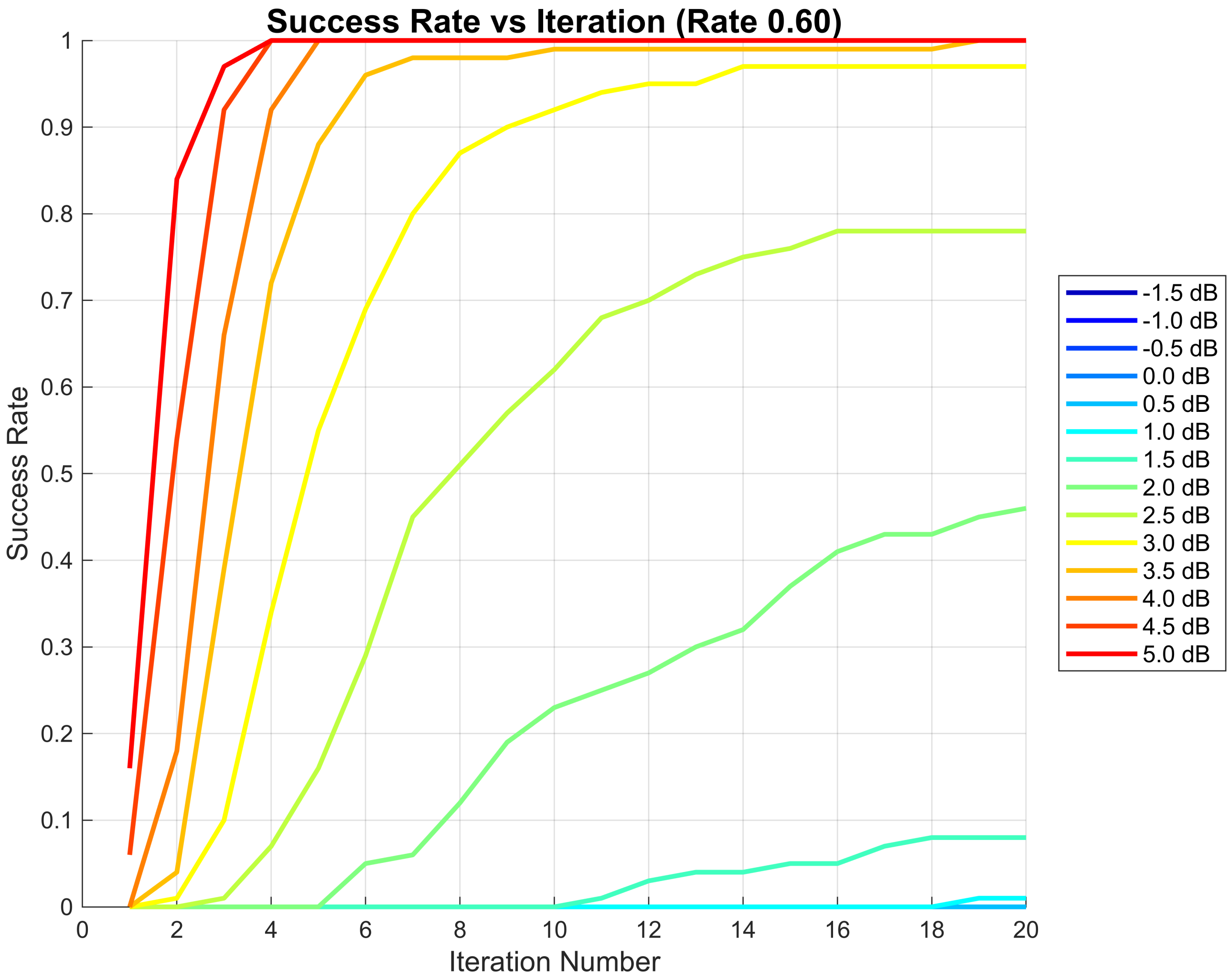

Rate 0.50 LDPC Code (K=1144, N=3536)
Block Error Rate (BLER)
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
-1
0
1
2
3
4
5
Eb/No (dB)
LDPC Simulation
Normal Approximation
Shannon Limit



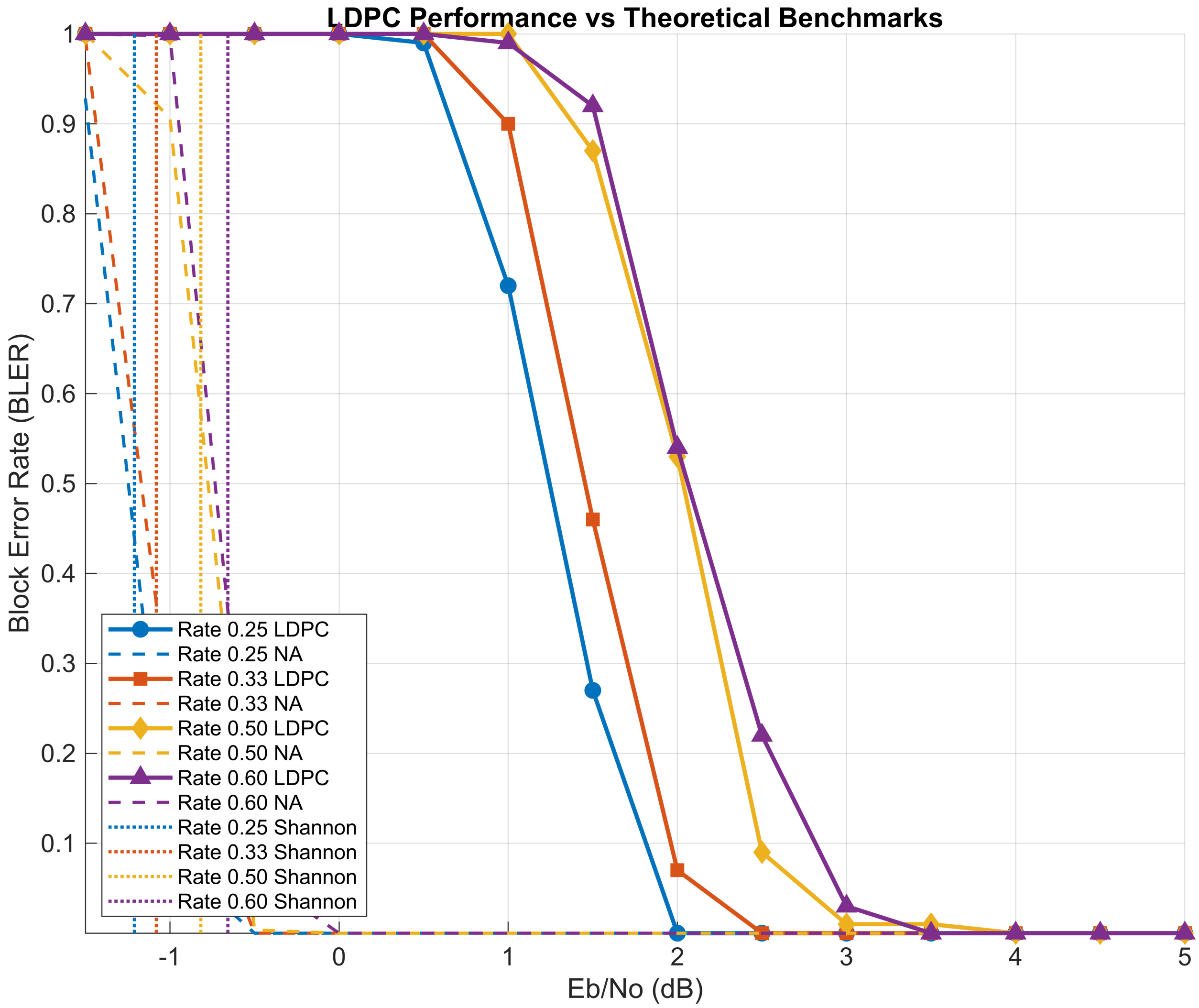
```
Simulating code rate 0.60 (4/4)
  SNR -1.5 dB: FER=1.00e+00, Pc=0.00
  SNR -1.0 dB: FER=1.00e+00, Pc=0.00
  SNR -0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 0.0 dB: FER=1.00e+00, Pc=0.00
  SNR 0.5 dB: FER=1.00e+00, Pc=0.00
  SNR 1.0 dB: FER=9.90e-01, Pc=0.01
  SNR 1.5 dB: FER=9.20e-01, Pc=0.08
  SNR 2.0 dB: FER=5.40e-01, Pc=0.46
  SNR 2.5 dB: FER=2.20e-01, Pc=0.78
  SNR 3.0 dB: FER=3.00e-02, Pc=0.97
  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
```



Rate 0.60 LDPC Code (K=1144, N=3536)
Block Error Rate (BLER)
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
-1
0
1
2
3
4
5
Eb/No (dB)
LDPC Simulation
Normal Approximation
Shannon Limit



```
%% Generate Combined Performance Plot with Theoretical Benchmarks
figure(100);
set(gcf, 'Position', [100, 100, 900, 700]);
hold on;

colors = lines(length(codeRates));
markers = {'o', 's', 'd', '^'};

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

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

% Plot Shannon limits
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)', 'FontSize', 14);
ylabel('Block Error Rate (BLER)', 'FontSize', 14);
title('LDPC Performance vs Theoretical Benchmarks', 'FontSize', 16);
legend('Location', 'southwest', 'FontSize', 10);
set(gca, 'FontSize', 12);
ylim([1e-4 1]);
xlim([min(EbN0dB_vec) max(EbN0dB_vec)]);
```



```
%% Helper Functions
function [decoded_bits, iteration_history, final_success] = ...
    ldpc_decode_c332(llr, H, VN_to_CN_map, CN_to_VN_map, max_iter, original_msg)

    [num_CNs, num_VNs] = size(H);
    VN_msgs = zeros(num_CNs, num_VNs);
    CN_msgs = zeros(num_CNs, num_VNs);
    iteration_history = zeros(1, max_iter);
    kNumInfoBits = length(original_msg);

    % Initialize with channel LLRs
    for vn = 1:num_VNs
        cn_list = VN_to_CN_map{vn};
        VN_msgs(cn_list, vn) = llr(vn);
    end
    final_success = 0;
    for iter = 1:max_iter
        % Check node updates (min-sum with scaling)
        for cn = 1:num_CNs
            vn_list = CN_to_VN_map{cn};
            incoming = VN_msgs(cn, vn_list);
            sign_prod = prod(sign(incoming));
            abs_incoming = abs(incoming);

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

        % Variable node updates
        decoded_bits = zeros(1, num_VNs);
        for vn = 1:num_VNs
            cn_list = VN_to_CN_map{vn};
            total = llr(vn) + sum(CN_msgs(cn_list, vn));

            for cn = cn_list
                VN_msgs(cn, vn) = total - CN_msgs(cn, vn);
            end

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

        % Track success at each iteration
        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; % Fill remaining iterations
            break;
        end
    end
end

%% Rest of the helper functions remain unchanged
function [VN_to_CN_map, CN_to_VN_map] = build_tanner_graph(H)
    [num_CNs, num_VNs] = size(H);
    VN_to_CN_map = cell(num_VNs, 1);
    CN_to_VN_map = cell(num_CNs, 1);

    for vn = 1:num_VNs
        VN_to_CN_map{vn} = find(H(:, vn))';
    end
    for cn = 1:num_CNs
        CN_to_VN_map{cn} = find(H(cn, :));
    end
end

function [B,H,z] = nrldpc_Hmatrix(BG)
    load(sprintf('%s.txt',BG),BG);
    B = NR_2_6_52;
    [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

    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
```