```
% This function generates a state table for a convolutional encoder.
function stateTable = generateStateTable(K, generators)

    numStates = 2^(K-1);
    stateTable = zeros(numStates, 4);

    for state = 0:numStates-1
        prevBits = bitget(state, K-1:-1:1);
        for inp = 0:1
            reg = [inp prevBits];
            outVec = mod(generators * reg', 2);
            out = bi2de(outVec', 'left-msb');
            nextBits = reg(1:end-1);
            nextState = bi2de(nextBits, 'left-msb');
            col = inp*2 + 1;
            stateTable(state+1, col)   = out;
            stateTable(state+1, col+1) = nextState;
        end
    end
end


% This function performs convolutional encoding on a binary message using the
specified generator polynomials.
function codeword = convolutionalEncoding(msg, generators)
    [numRows,numCols] = size(generators);
    mid_codeword = zeros(numRows,numCols+length(msg)-1);
    for row = 1 : numRows
        g = generators(row,:);
        mid_codeword(row,:) = mod(conv(msg,g),2);
    end
    codeword = mid_codeword(:).';
end


% Viterbi decoder for hard decision decoding
function decoded = viterbiDecode_hard(s, n , received)
    numStates = size(s,1);
    N = size(received,1);


    pathMetric = inf(numStates, N+1);
    prevState  = zeros(numStates, N+1);
    prevInput  = zeros(numStates, N+1);

    pathMetric(1,1) = 0;

    for t = 1:N

        r = received(t,:);
```

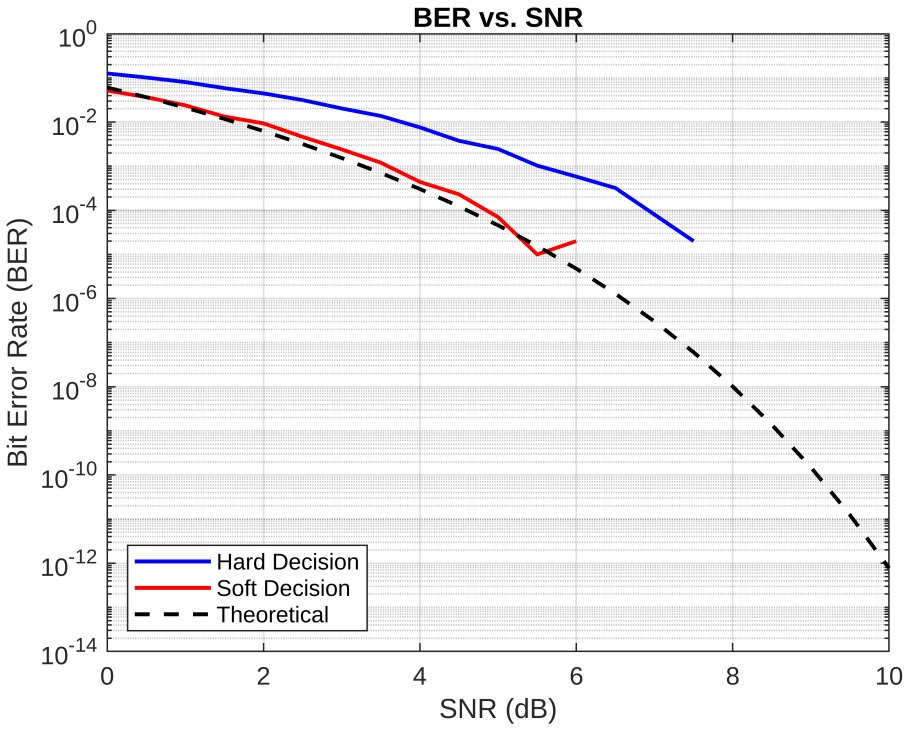
```
        for st = 1:numStates

            if isfinite(pathMetric(st,t))

                for inp = 0:1

                    col = inp*2 + 1;
                    outBits = de2bi(s(st,col), n, 'left-msb');
                    ns = s(st, col+1) + 1;
                    hd = sum(xor(outBits, r));
                    metric = pathMetric(st,t) + hd;

                    if metric < pathMetric(ns, t+1)
                        pathMetric(ns, t+1) = metric;
                        prevState(ns, t+1)  = st;
                        prevInput(ns, t+1)  = inp;
                    end

                end

            end
        end
    end


    cur = 1;
    decRev = zeros(1,N);
    for t = N+1:-1:2
        decRev(t-1) = prevInput(cur, t);
        cur = prevState(cur, t);
    end

    decoded = decRev;

end


% Viterbi decoder for a convolutional code with soft decision decoding
function decoded = viterbiDecode_soft(s, n, received)
numStates = size(s,1);
N = size(received,1);

pathMetric = inf(numStates, N+1);
prevState  = zeros(numStates, N+1);
prevInput  = zeros(numStates, N+1);

pathMetric(1,1) = 0;

for t = 1:N
```

```
    r = received(t,:);

    for st = 1:numStates

        if isfinite(pathMetric(st,t))

            for inp = 0:1
                col = inp*2 + 1;
                outBits = de2bi(s(st,col), n, 'left-msb');

                ns = s(st, col+1) + 1;
                outBits = outBits(:).';
                bpskOut = 1 - 2*outBits;
                ed = sum((bpskOut - r).^2);

                metric = pathMetric(st,t) + ed;
                if metric < pathMetric(ns, t+1)
                    pathMetric(ns, t+1) = metric;
                    prevState(ns, t+1)  = st;
                    prevInput(ns, t+1)  = inp;
                end

            end

        end

    end
end


    cur = 1;
    decRev = zeros(1,N);
    for t = N+1:-1:2
        decRev(t-1) = prevInput(cur, t);
        cur = prevState(cur, t);
    end

    decoded = decRev;

end
```

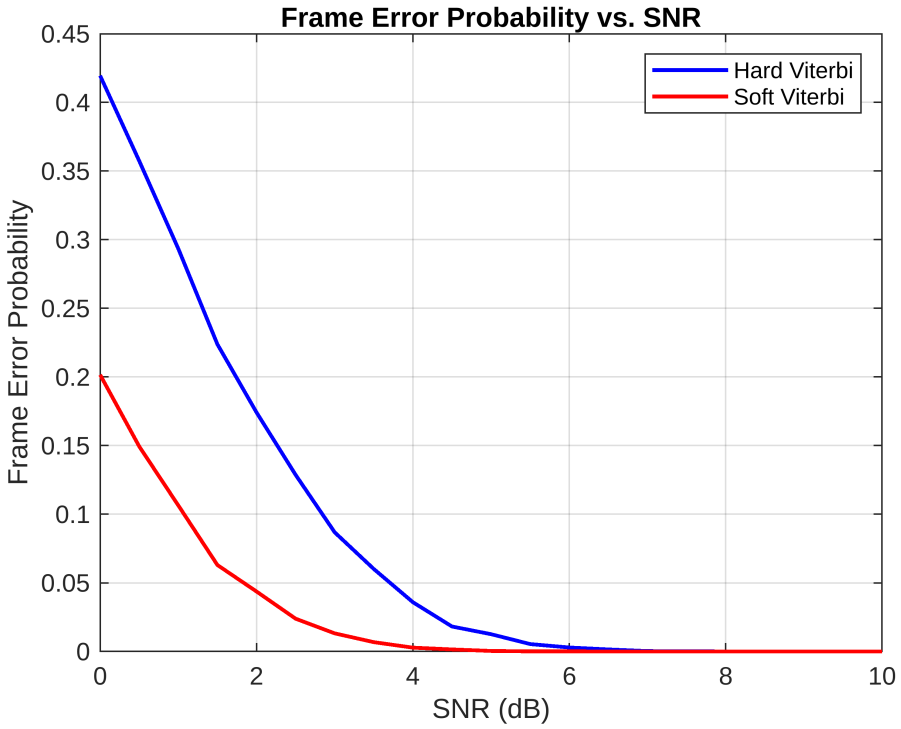
Case: 1

```
clc ;
clear all ; close all ;

K = 3;
generatorPolynomials = [1 0 1; 1 1 1];


n = size(generatorPolynomials, 1);
stateTable = generateStateTable(K, generatorPolynomials);

SNR = 0 : 0.5 : 10;
length_SNR = length(SNR);
success_hard = zeros(length_SNR, 1);
success_soft = zeros(length_SNR, 1);

bit_error_soft = zeros(length_SNR, 1);
bit_error_hard = zeros(length_SNR, 1);

for i = SNR
    trails = 10000;
    message_size = 10;
    noiseVariance = 10 ^ (- i / 10);

    curr_success_hard = 0;
    curr_success_soft = 0;

    curr_bit_error_hard = 0;
    curr_bit_error_soft = 0;

    for t = 1 : trails
        messageBits = randi([0 1], 1, message_size);
        encodedBits = convolutionalEncoding(messageBits,
generatorPolynomials);
        modulatedSignal = 1 - 2 * encodedBits;

        noiseSamples = sqrt(noiseVariance) * randn(size(modulatedSignal));
        receivedSignal = modulatedSignal + noiseSamples;

        demodulatedBits = double(receivedSignal < 0);
        numSymbols = length(demodulatedBits)/n;
        demodulatedSymbol = reshape(demodulatedBits, n, numSymbols)';

        decoded_hard = viterbiDecode_hard(stateTable, n, demodulatedSymbol);
        if isequal(messageBits, decoded_hard(1:length(messageBits)))
            curr_success_hard = curr_success_hard + 1;
        end

        Nsym = length(receivedSignal) / n;
```

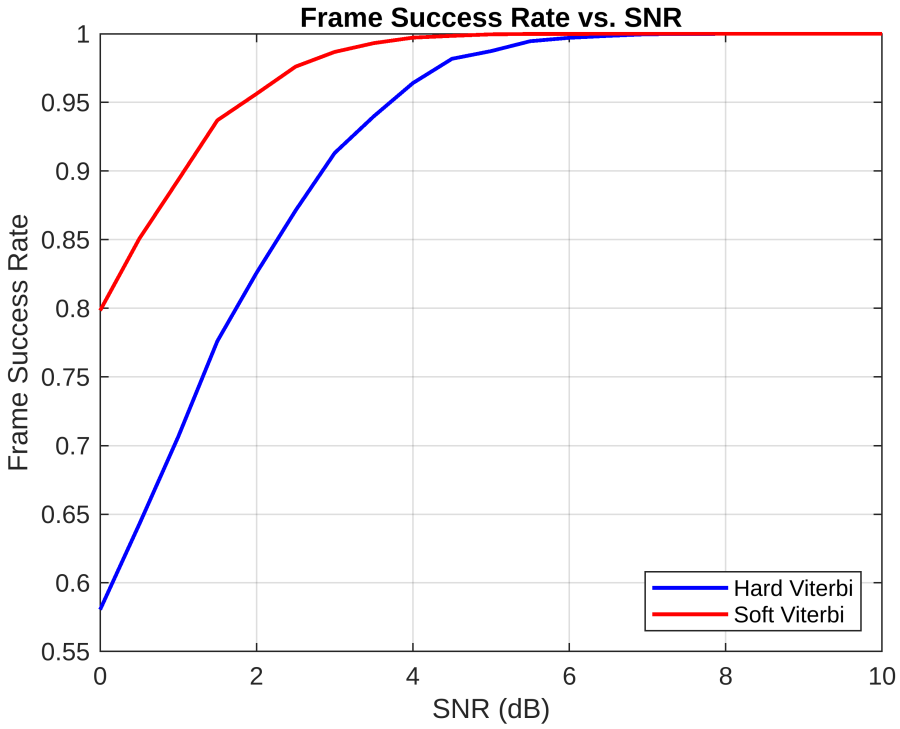
```
        structured_received = reshape(receivedSignal, n, Nsym)';
        decoded_soft = viterbiDecode_soft(stateTable, n,
structured_received);
        if isequal(messageBits, decoded_soft(1:length(messageBits)))
            curr_success_soft = curr_success_soft + 1;
        end

        curr_bit_error_hard = curr_bit_error_hard + sum(xor(messageBits,
decoded_hard(1:length(messageBits))));
        curr_bit_error_soft = curr_bit_error_soft + sum(xor(messageBits,
decoded_soft(1:length(messageBits))));
    end

    success_hard(i*2 + 1) = curr_success_hard / trails;
    success_soft(i*2 + 1) = curr_success_soft / trails;

    bit_error_hard(i*2 + 1) = curr_bit_error_hard / (length(messageBits) *
trails);
    bit_error_soft(i*2 + 1) = curr_bit_error_soft / (length(messageBits) *
trails);
end

EbN0 = 10.^(SNR / 10);
Pb_theoretical = zeros(size(EbN0));
cd = [0 0 0 0 1 1 2 4 8 16];
for i = 5:10
    Pb_theoretical = Pb_theoretical + cd(i) * qfunc(sqrt(2 * i * (1/2) *
EbN0));
end



function y = qfunc(x)
y = 0.5 * erfc(x / sqrt(2));
end


figure;
semilogy(SNR, bit_error_hard, 'b-','LineWidth',1.5); hold on;
semilogy(SNR, bit_error_soft, 'r-','LineWidth',1.5);
semilogy(SNR, Pb_theoretical, 'k--','LineWidth',1.5);
hold off;
grid on;
xlabel('SNR (dB)');
ylabel('Bit Error Rate (BER)');
legend('Hard Decision','Soft Decision','Theoretical','Location','southwest');
title('BER vs. SNR');
```




```
pErr_hard = 1 - success_hard;
pErr_soft = 1 - success_soft;

figure;
plot(SNR, pErr_hard, 'b-','LineWidth',1.5); hold on;
plot(SNR, pErr_soft, 'r-','LineWidth',1.5);
hold off;
grid on;
xlabel('SNR (dB)');
ylabel('Frame Error Probability');
legend('Hard Viterbi','Soft Viterbi','Location','northeast');
title('Frame Error Probability vs. SNR');
```




```
figure;
plot(SNR, success_hard, 'b-','LineWidth',1.5); hold on;
plot(SNR, success_soft, 'r-','LineWidth',1.5);
hold off;
grid on;
xlabel('SNR (dB)');
ylabel('Frame Success Rate');
legend('Hard Viterbi','Soft Viterbi','Location','southeast');
title('Frame Success Rate vs. SNR');
```

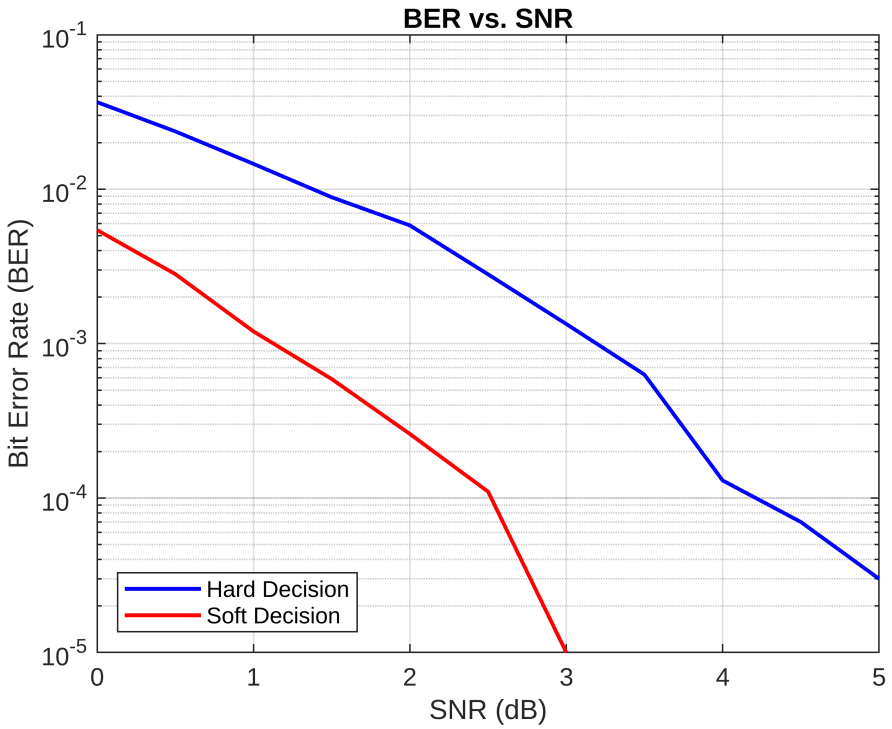


Case: 2

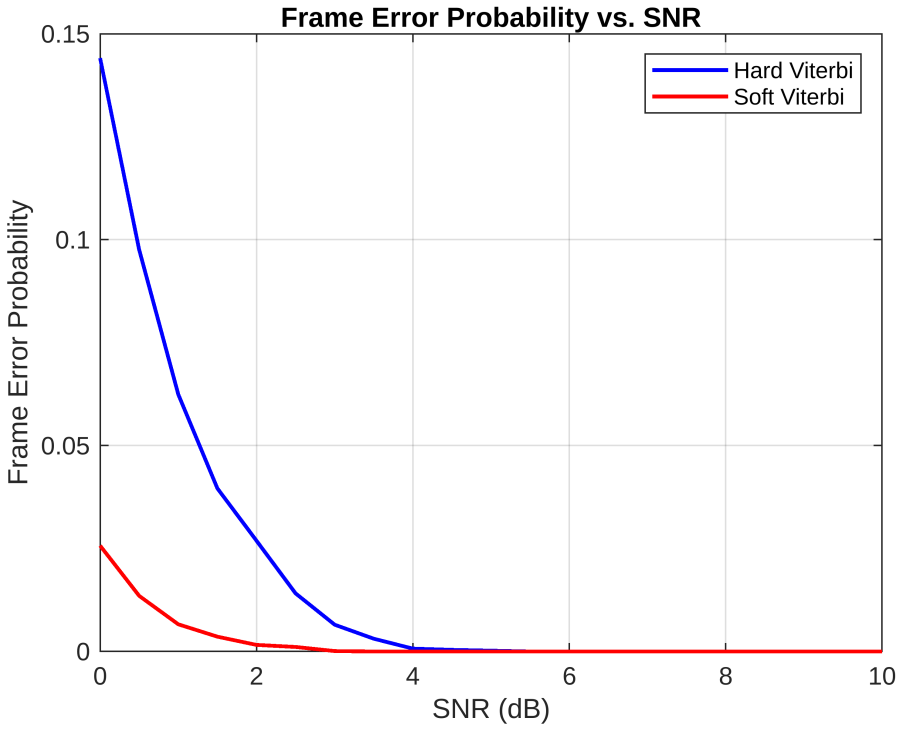
```
clc ;
clear all ; close all ;

K = 4;
generatorPolynomials  = [
    1 1 0 1;
    1 1 1 1;
    1 0 0 1
];



n = size(generatorPolynomials, 1);
stateTable = generateStateTable(K, generatorPolynomials);

SNR = 0 : 0.5 : 10;
length_SNR = length(SNR);
success_hard = zeros(length_SNR, 1);
success_soft = zeros(length_SNR, 1);

bit_error_soft = zeros(length_SNR, 1);
bit_error_hard = zeros(length_SNR, 1);

for i = SNR
```

```
    trails = 10000;
    message_size = 10;
    noiseVariance = 10 ^ (- i / 10);

    curr_success_hard = 0;
    curr_success_soft = 0;

    curr_bit_error_hard = 0;
    curr_bit_error_soft = 0;

    for t = 1 : trails
        messageBits = randi([0 1], 1, message_size);
        encodedBits = convolutionalEncoding(messageBits,
generatorPolynomials);
        modulatedSignal = 1 - 2 * encodedBits;

        noiseSamples = sqrt(noiseVariance) * randn(size(modulatedSignal));
        receivedSignal = modulatedSignal + noiseSamples;

        demodulatedBits = double(receivedSignal < 0);
        numSymbols = length(demodulatedBits)/n;
        demodulatedSymbol = reshape(demodulatedBits, n, numSymbols)';

        decoded_hard = viterbiDecode_hard(stateTable, n, demodulatedSymbol);
        if isequal(messageBits, decoded_hard(1:length(messageBits)))
            curr_success_hard = curr_success_hard + 1;
        end

        Nsym = length(receivedSignal) / n;
        structured_received = reshape(receivedSignal, n, Nsym)';
        decoded_soft = viterbiDecode_soft(stateTable, n,
structured_received);
        if isequal(messageBits, decoded_soft(1:length(messageBits)))
            curr_success_soft = curr_success_soft + 1;
        end

        curr_bit_error_hard = curr_bit_error_hard + sum(xor(messageBits,
decoded_hard(1:length(messageBits))));
        curr_bit_error_soft = curr_bit_error_soft + sum(xor(messageBits,
decoded_soft(1:length(messageBits))));
    end

    success_hard(i*2 + 1) = curr_success_hard / trails;
    success_soft(i*2 + 1) = curr_success_soft / trails;

    bit_error_hard(i*2 + 1) = curr_bit_error_hard / (length(messageBits) *
trails);
    bit_error_soft(i*2 + 1) = curr_bit_error_soft / (length(messageBits) *
trails);
end
```

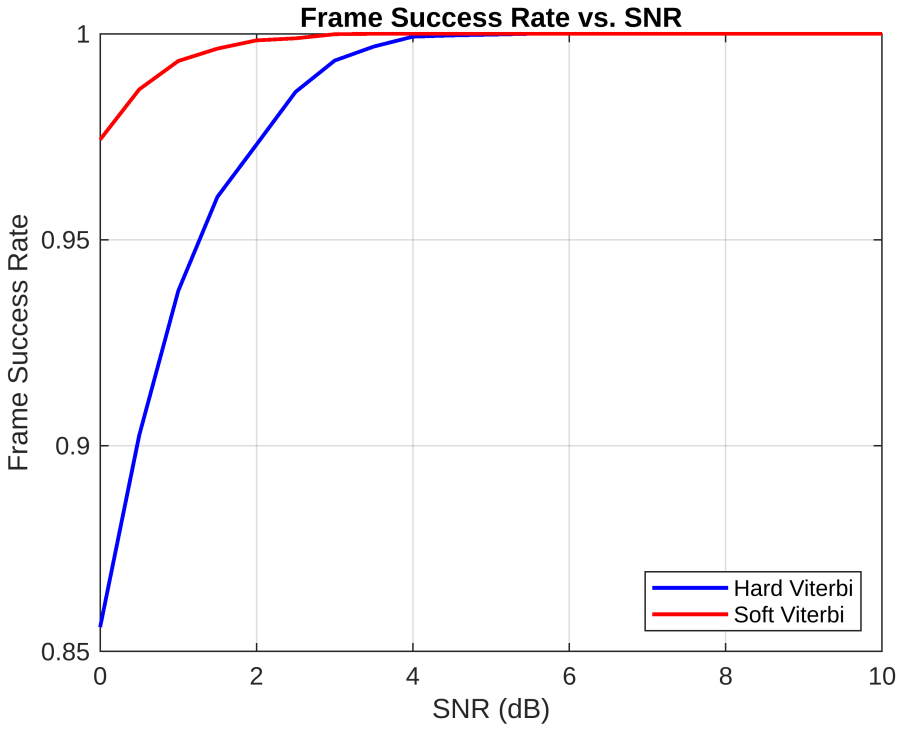
```
figure;
semilogy(SNR, bit_error_hard, 'b-','LineWidth',1.5); hold on;
semilogy(SNR, bit_error_soft, 'r-','LineWidth',1.5);
hold off;
grid on;
xlabel('SNR (dB)');
ylabel('Bit Error Rate (BER)');
legend('Hard Decision','Soft Decision','Location','southwest');
title('BER vs. SNR');
```




```
pErr_hard = 1 - success_hard;
pErr_soft = 1 - success_soft;

figure;
plot(SNR, pErr_hard, 'b-','LineWidth',1.5); hold on;
plot(SNR, pErr_soft, 'r-','LineWidth',1.5);
hold off;
grid on;
xlabel('SNR (dB)');
ylabel('Frame Error Probability');
legend('Hard Viterbi','Soft Viterbi','Location','northeast');
title('Frame Error Probability vs. SNR');
```




```
figure;
plot(SNR, success_hard, 'b-','LineWidth',1.5); hold on;
plot(SNR, success_soft, 'r-','LineWidth',1.5);
hold off;
grid on;
xlabel('SNR (dB)');
ylabel('Frame Success Rate');
legend('Hard Viterbi','Soft Viterbi','Location','southeast');
title('Frame Success Rate vs. SNR');
```

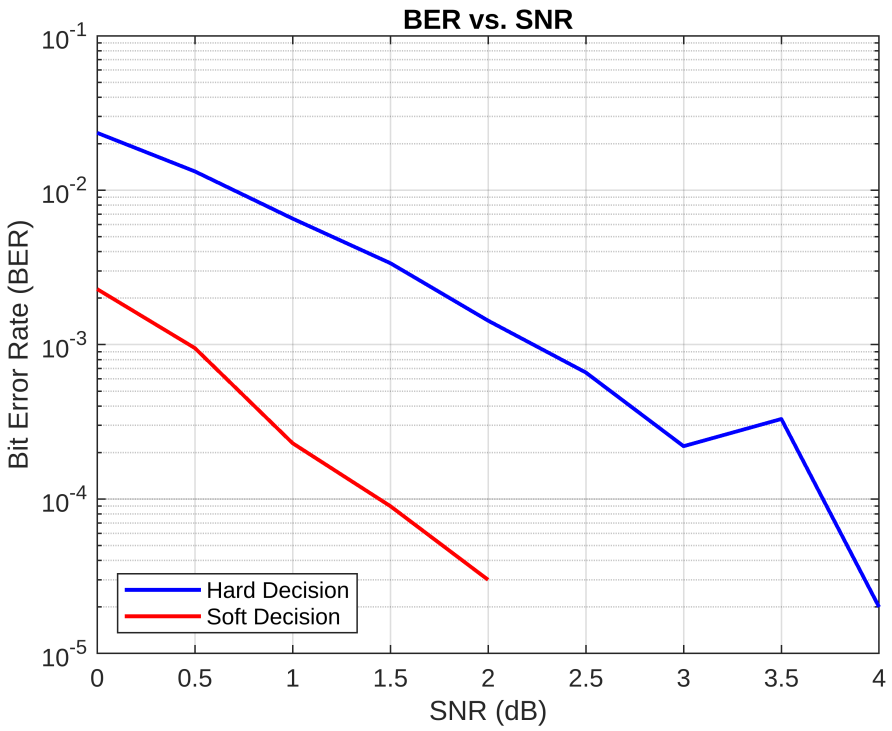


Case: 3

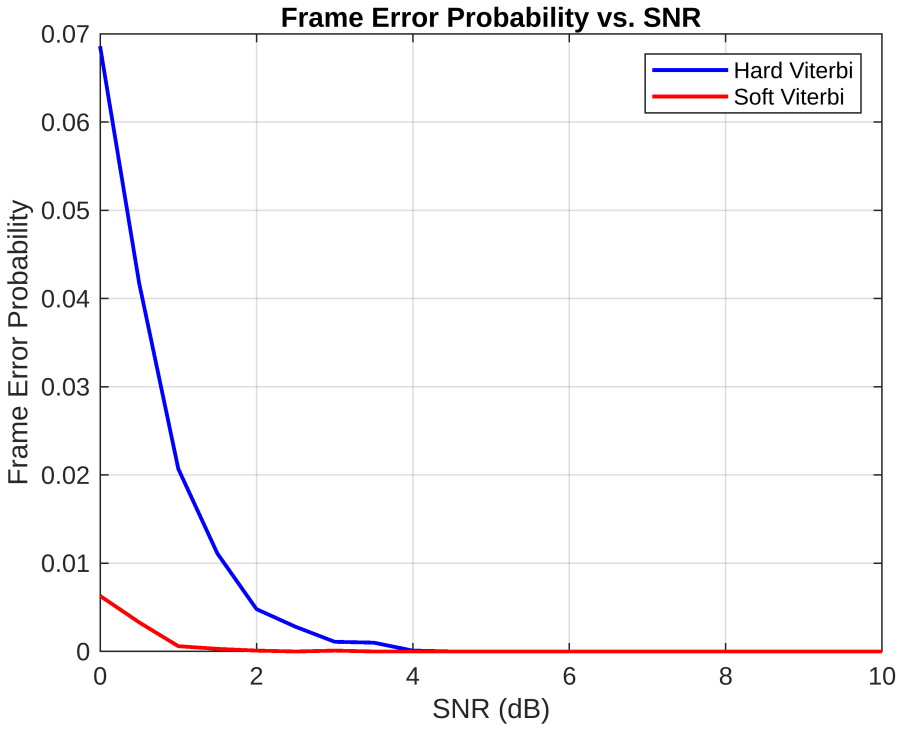
```
clc ;
clear all ; close all ;


K = 6;
generatorPolynomials = [
    1 0 1 1 1 1;
    1 1 1 0 0 1;
    1 1 0 1 0 1
];


n = size(generatorPolynomials, 1);
stateTable = generateStateTable(K, generatorPolynomials);

SNR = 0 : 0.5 : 10;
length_SNR = length(SNR);
success_hard = zeros(length_SNR, 1);
success_soft = zeros(length_SNR, 1);

bit_error_soft = zeros(length_SNR, 1);
bit_error_hard = zeros(length_SNR, 1);

for i = SNR
    trails = 10000;
```

```
    message_size = 10;
    noiseVariance = 10 ^ (- i / 10);

    curr_success_hard = 0;
    curr_success_soft = 0;

    curr_bit_error_hard = 0;
    curr_bit_error_soft = 0;

    for t = 1 : trails
        messageBits = randi([0 1], 1, message_size);
        encodedBits = convolutionalEncoding(messageBits,
generatorPolynomials);
        modulatedSignal = 1 - 2 * encodedBits;

        noiseSamples = sqrt(noiseVariance) * randn(size(modulatedSignal));
        receivedSignal = modulatedSignal + noiseSamples;

        demodulatedBits = double(receivedSignal < 0);
        numSymbols = length(demodulatedBits)/n;
        demodulatedSymbol = reshape(demodulatedBits, n, numSymbols)';

        decoded_hard = viterbiDecode_hard(stateTable, n, demodulatedSymbol);
        if isequal(messageBits, decoded_hard(1:length(messageBits)))
            curr_success_hard = curr_success_hard + 1;
        end

        Nsym = length(receivedSignal) / n;
        structured_received = reshape(receivedSignal, n, Nsym)';
        decoded_soft = viterbiDecode_soft(stateTable, n,
structured_received);
        if isequal(messageBits, decoded_soft(1:length(messageBits)))
            curr_success_soft = curr_success_soft + 1;
        end

        curr_bit_error_hard = curr_bit_error_hard + sum(xor(messageBits,
decoded_hard(1:length(messageBits))));
        curr_bit_error_soft = curr_bit_error_soft + sum(xor(messageBits,
decoded_soft(1:length(messageBits))));
    end

    success_hard(i*2 + 1) = curr_success_hard / trails;
    success_soft(i*2 + 1) = curr_success_soft / trails;

    bit_error_hard(i*2 + 1) = curr_bit_error_hard / (length(messageBits) *
trails);
    bit_error_soft(i*2 + 1) = curr_bit_error_soft / (length(messageBits) *
trails);
end
```

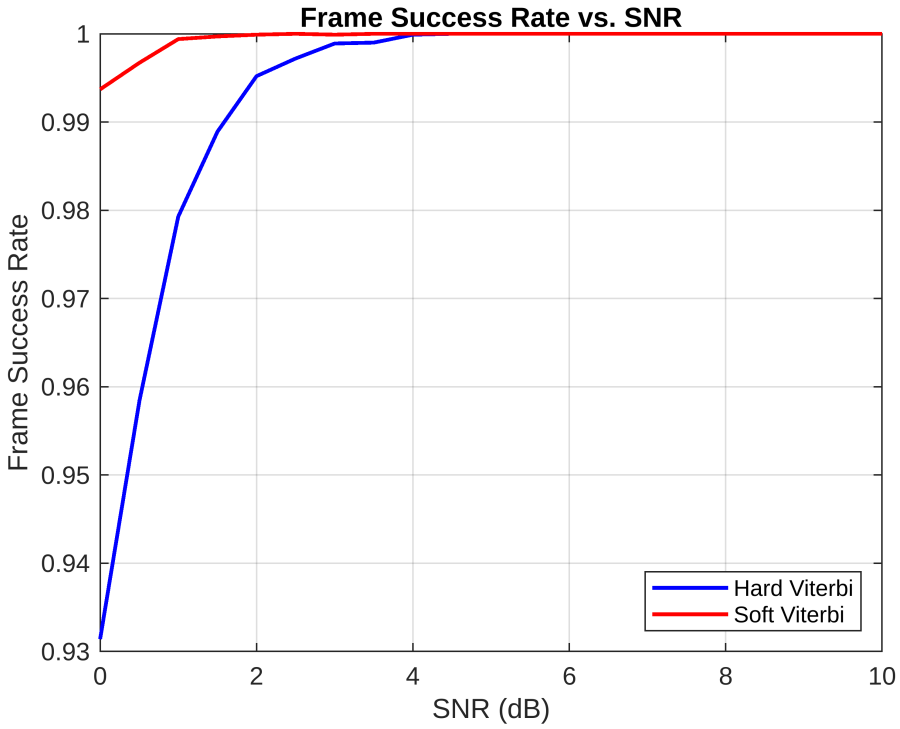
```
figure;
semilogy(SNR, bit_error_hard, 'b-','LineWidth',1.5); hold on;
semilogy(SNR, bit_error_soft, 'r-','LineWidth',1.5);
hold off;
grid on;
xlabel('SNR (dB)');
ylabel('Bit Error Rate (BER)');
legend('Hard Decision','Soft Decision','Location','southwest');
title('BER vs. SNR');
```

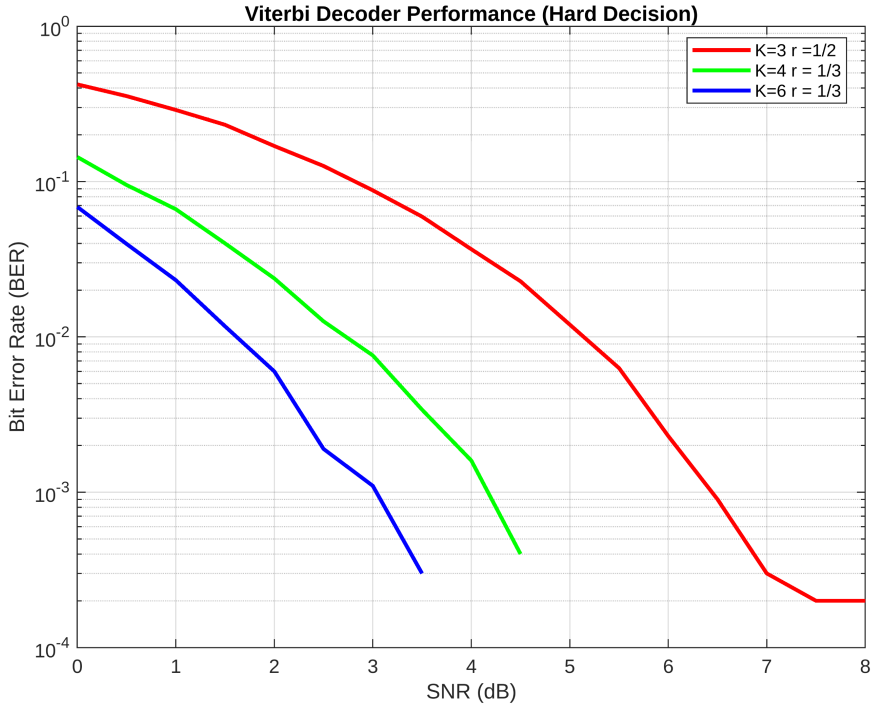



```
pErr_hard = 1 - success_hard;
pErr_soft = 1 - success_soft;

figure;
plot(SNR, pErr_hard, 'b-','LineWidth',1.5); hold on;
plot(SNR, pErr_soft, 'r-','LineWidth',1.5);
hold off;
grid on;
xlabel('SNR (dB)');
ylabel('Frame Error Probability');
legend('Hard Viterbi','Soft Viterbi','Location','northeast');
title('Frame Error Probability vs. SNR');
```




```
figure;
plot(SNR, success_hard, 'b-','LineWidth',1.5); hold on;
plot(SNR, success_soft, 'r-','LineWidth',1.5);
hold off;
grid on;
xlabel('SNR (dB)');
ylabel('Frame Success Rate');
legend('Hard Viterbi','Soft Viterbi','Location','southeast');
title('Frame Success Rate vs. SNR');
```


Frame Success Rate vs. SNR
Frame Success Rate
SNR (dB)
Hard Viterbi
Soft Viterbi
1
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0
2
4
6
8
10


Viterbi Decoder Performance (Hard Decision)
Bit Error Rate (BER)
SNR (dB)
K=3 r =1/2
K=4 r = 1/3
K=6 r = 1/3
10^0
10^-1
10^-2
10^-3
10^-4
0
1
2
3
4
5
6
7
8