INTRODUCTION TO COMMUNICATION SYSTEMS
(CT216)

Matlab code and Results :

Lab Group - 2 : Project Group - 2

Prof. Yash Vasavada
Mentor TA : Shrey Andhariya

TOPIC : CONVOLUTIONAL CODING

Group Members :

1) **Janavi Ramani - 202201158**
2) **Shyam Ghetiya - 202201161**
3) **Nishil Patel - 202201166**
4) **Tathya Prajapati - 202201170**
5) **Sunil Rathva - 202201177**
6) **Vedant Savani - 202201178**
7) **Nitin Kanzariya - 202201181**
8) **Akshat Joshi - 202201185**
9) **Bansi Patel - 202201190**
10) **Bhoomish Patel - 202201414**

Honor code :

- The work that we are presenting is our own work.
- We have not copied the work (the code,the results,etc.) that someone else has done.
- Concepts,understanding and insights we will be describing are our own.
- We make this pledge truthfully.
- We know that violation of this solemn pledge can carry grave consequences.

```
function ans = encoding(kc, generation_polynomials, inp)
    n = length(generation_polynomials);
    ans = [];
    regval = 0;
    for i = 1:length(inp)
        curbit = inp(i);
        regval = bitor(bitshift(regval, -1), bitshift(curbit, (kc - 1)));
        for j = 1:n
            cur_poly = generation_polynomials(j);
            output = 0;
            p=dec2bin(cur_poly,kc);
            p=fliplr(p)-'0';
  rv=dec2bin(regval,kc)-'0';
            output=0;
 for k=1:kc
                output=xor(output,(p(k) & rv(k)));
 end
            ans = [ans, output];
 end
 end
 end
 function ans = decoding_soft(kc, generation_polynomials, inp)
    ns = bitshift(1, (kc-1));
    n = length(generation_polynomials);
    mtr = ones(floor(length(inp)/n) + 1, ns) * 1e9;
    previous_states = cell(floor(length(inp)/n) + 1, 1);
 for i = 1:length(previous_states)
        previous_states{i} = cell(ns, 1);
 for j = 1:ns
            previous_states{i}{j} = {1e9, 1e9};
 end
 end
    idx = 0;
 for t = 1:(length(inp)/n)
 for st = 0:ns-1
 if (t == 1) && (st == 0)
                mtr(t, st+1) = 0;
 end
 for input_bits = 0:1
                curstate = st;
                next_state = bitor(bitshift(curstate, -1),(input_bits * ...
bitshift(1, (kc-2))));
                euclidian_distance = 0;
 for i = 1:n
                    output = 0;
                    poly = generation_polynomials(i);
                    regvalue = bitor(curstate, bitshift(input_bits, (kc-1)));
 for k = 0:(kc-1)
                        output = bitxor(output, bitand(bitshift(poly, -k),
...
```

```
1) & bitand(bitshift(regvalue, -(kc-k-1)), 1));
end
                    output=1-2*output;
                    euclidian_distance=euclidian_distance+(abs(output-...
inp(idx+i))*abs(output-inp(idx+i)));
 end
                euclidian=sqrt(euclidian_distance);
                a = mtr(t+1, next_state+1);
                b = mtr(t, curstate+1);
 if a > b + euclidian_distance
                    mtr(t+1, next_state+1) = b + euclidian_distance;
                     previous_states{t+1}{next_state+1} =
{curstate,input_bits};
 end
 end
 end
        idx = idx + n;
 end
    ans = [];
    temp = previous_states{(length(inp)/n)+1}{1};
    ans = [temp{2} ans];
    cur = (length(inp)/n);
 while cur >1
        temp = previous_states{cur}{temp{1}+1};
        ans = [temp{2} ans];
        cur = cur - 1;
 end
 end



 function modulated_op = modulator(encoded_message,sigma)
     s = 1 - 2 * encoded_message; % BPSK modulation
     modulated_op= s + sigma * randn(1, length(encoded_message));
 end

 function ans = decoding(kc, generation_polynomials, inp)
    ns = bitshift(1, (kc-1));
    n = length(generation_polynomials);
    mtr = ones(floor(length(inp)/n) + 1, ns) * 1e9;
    previous_states = cell(floor(length(inp)/n) + 1, 1);
    for i = 1:length(previous_states)
        previous_states{i} = cell(ns, 1);
        for j = 1:ns
            previous_states{i}{j} = {1e9, 1e9};
        end
    end
    idx = 0;
    for t = 1:(length(inp)/n)
        for st = 0:ns-1
```

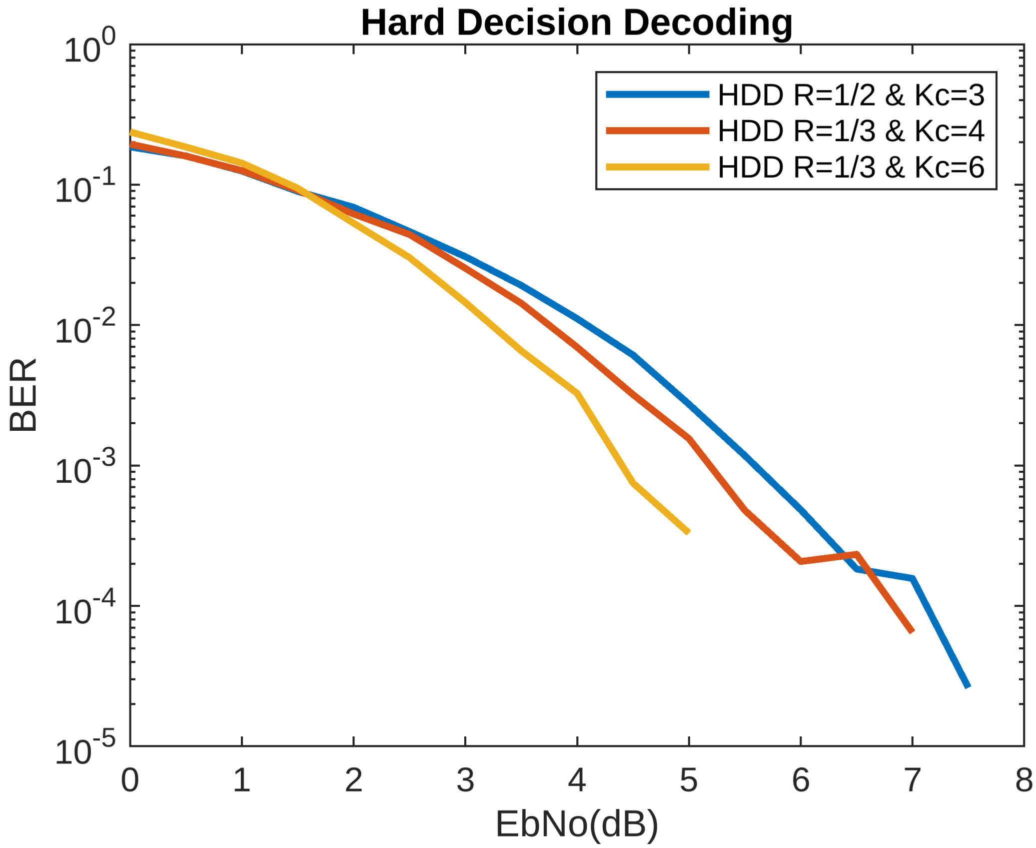
```
            if (t == 1) && (st == 0)
                mtr(t, st+1) = 0;
            end
            for input_bits = 0:1
                curstate = st;
                next_state = bitor(bitshift(curstate, -1),(input_bits *
bitshift(1, (kc-2))));
                hamming_distance = 0;
                for i = 1:n
                    output = 0;
                    poly = generation_polynomials(i);
                    regvalue = bitor(curstate, bitshift(input_bits, (kc-1)));
                    for k = 0:(kc-1)
                        output = bitxor(output, bitand(bitshift(poly, -k),
1) & bitand(bitshift(regvalue, -(kc-k-1)), 1));
                    end
                    if output ~= inp(idx+i)
                        hamming_distance = hamming_distance + 1;
                    end
                end
                a = mtr(t+1, next_state+1);
                b = mtr(t, curstate+1);
                if a > b + hamming_distance
                    mtr(t+1, next_state+1) = b + hamming_distance;
                    previous_states{t+1}{next_state+1} =
{curstate,input_bits};
                end
            end
        end
        idx = idx + n;
    end
    ans = [];
    temp = previous_states{(length(inp)/n)+1}{1};
    ans = [temp{2} ans];
    cur = (length(inp)/n);
    while cur >1
        temp = previous_states{cur}{temp{1}+1};
        ans = [temp{2} ans];
        cur = cur - 1;
    end
 end



EbNodB = 0:0.5:10;
 R1 = 1/2;
 R2 = 1/3;
 R3 = 1/3;
 n1 =2;
 n2 =3;
```

```
n3 =3;
k1 =1;
k2 =1;
k3 =1;
kc1 = 3;
kc2 = 4;
kc3 = 6;

practical_error_1 = zeros(1,length(EbNodB));
practical_error_2 = zeros(1,length(EbNodB));
practical_error_3 = zeros(1,length(EbNodB));
%theoratical_error = zeros(1,length(EbNodB));
%soft_error=zeros(1,length(EbNodB));
idx =1;
idx2=1;
N = 750;
for j=EbNodB
EbNo = 10^(j/10);
sigma1 = sqrt (1/(2*R1*EbNo));
sigma2 = sqrt (1/(2*R2*EbNo));
sigma3 = sqrt (1/(2*R3*EbNo));
BER_th = 0.5*erfc(sqrt(EbNo));
Nerrs1 = 0;
Nerrs2 = 0;
Nerrs3 = 0;
for i = 1 : N
msg = randi ([0 1],1,100); %generate random message

msg1=[msg zeros(1,kc1-1)];
msg2=[msg zeros(1,kc2-1)];
msg3=[msg zeros(1,kc3-1)];

encoded_array1=encoding(kc1,[5 7],msg1);
encoded_array2=encoding(kc2,[11 13 15],msg2);
encoded_array3=encoding(kc3,[39 43 61],msg3);

modulated_message1=modulator(encoded_array1,sigma1);
modulated_message2=modulator(encoded_array2,sigma2);
modulated_message3=modulator(encoded_array3,sigma3);

demodualted_message1=modulated_message1<0;
demodualted_message2=modulated_message2<0;
demodualted_message3=modulated_message3<0;

decoded_message1=decoding(kc1,[5 7],demodualted_message1);
decoded_message2=decoding(kc2,[11 13 15],demodualted_message2);
decoded_message3=decoding(kc3,[39 43 61],demodualted_message3);

%soft_decoded_message=decoding_soft(kc,[5 7],modulated_message);
%Nerr_soft=Nerr_soft+sum(soft_decoded_message~=msg);
```

```
Nerrs1=Nerrs1+sum(msg1 ~= decoded_message1);
Nerrs2=Nerrs2+sum(msg2 ~= decoded_message2);
Nerrs3=Nerrs3+sum(msg3 ~= decoded_message3);

end
j
practical_error_1(idx) = (Nerrs1/(N*length(msg1)));
practical_error_2(idx) = (Nerrs2/(N*length(msg2)));
practical_error_3(idx) = (Nerrs3/(N*length(msg3)));
idx = idx+1;
idx2 = idx2+1;
end
```

```
j = 0
j = 0.5000
j = 1
j = 1.5000
j = 2
j = 2.5000
j = 3
j = 3.5000
j = 4
j = 4.5000
j = 5
j = 5.5000
j = 6
j = 6.5000
j = 7
j = 7.5000
j = 8
j = 8.5000
j = 9
j = 9.5000
j = 10
```

```
semilogy(EbNodB,practical_error_1,'LineWidth',2.0);
hold on;
semilogy(EbNodB,practical_error_2,'LineWidth',2.0);
semilogy(EbNodB,practical_error_3,'LineWidth',2.0);
legend('HDD R=1/2 & Kc=3','HDD R=1/3 & Kc=4','HDD R=1/3 & Kc=6');
title('Hard Decision Decoding');
%xlim([0 1]);
xlabel('EbNo(dB)');
ylabel('BER');
hold off;
```



Hard Decision Decoding
HDD R=1/2 & Kc=3
HDD R=1/3 & Kc=4
HDD R=1/3 & Kc=6
BER
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
1
2
3
4
5
6
7
8
EbNo(dB)