20220120 Lab 06

Page No.
Date :

Prerak Dave

1) Linearity

Given c_1 & c_2 are valid codewords:

$\therefore H_{(n-k)\times n}\, C_{1(n\times 1)} = \phi$

$H_{(n-k)\times n}\, C_{2(n\times 2)} = \phi$

adding,

$$H_{(n-k)\times n}\,[C_{1(n\times 1)} + C_{2(n\times 2)}] = \phi$$

because

$$H_{(n-k)\times n}\, C_{1(n\times 1)} + H_{(n-k)\times n}\, C_{2(n\times 1)} = \phi$$

$\therefore C_{1,2}$ is a valid codewords.

$\therefore$ Here n should be same for both valid code

2) $S_{n-k\times 1} = H_{n-k\times n}\, e_{n\times 1}$

$r \Rightarrow c + e$

$\therefore$ If $r = c_{n\times 1} + e_{n\times 1}$

then,

$$S_{n-k\times 1} = H_{n-k\times n}\,[c+e]_{n\times 1}$$

$$\therefore H_{n-k\times n}\cdot c_{n\times 1} + H_{n-k\times n}\, e_{n\times 1} = S_{n-k\times 1}$$

However c is a valid codeword therefore

$$H_{n-k\times n}\, c_{n\times 1} = [0]_{n-k\times 1}$$

$$\therefore S_{n-k\times 1} = H_{n-k\times n}\cdot e_{n\times 1}$$

3) We see that syndrome vector only depends on error vector.

Here,

$$S_{n-k\times 1} = [H_{n-k\times m}] \cdot [c']_{n\times 1}$$

as c' is valid codeword.

$$H_{n-k\times n}\, C'_{n\times 1} = [0]_{n-k\times 1}$$

$$\therefore S_{n-k\times 1} = [0]_{n-k\times 1}$$

⇒ Error can't be detected if error is valid codeword.

4) A valid codeword should satisfy:

$$H_{n-k\times n}\, C_{n\times 1} = [0]_{n-k\times n}$$

as $C = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{n\times 1}$, it will give the product as $[0]$, so it is a valid codeword.

5) Let c_a & c_b be codewords with min. hamming distance pair.

$$d_{min} = d_H(c_a, c_b)$$

So,

$$w_{min} \le w_H(c_a + c_b)$$

$$w_{min} \le d_H(c_a + c_b, 0)$$

$$w_{min} \le d_H\left(\begin{matrix} c_a & c_b \\ 0 & 0 \end{matrix}\right)$$

$$w_{min} \le d_{min} \quad -(1)$$

C_d ⇒ codeword with min hamming weight

$w_{min} = w_H(C_d)$

$d_{min} \leq d_H(C_d, 0)$

$\leq w_H(C_d)$

$d_{min} \leq w_{min}$ —— ②

$\therefore d_{min} = w_{min}$

⑥ The H-matrix for rate = 4/7 (Parity check)

$$H_{3\times7} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \qquad n = 7$$

$$H_x = H \begin{bmatrix} c_1 \\ c_2 \\ c_3+1 \\ \vdots \\ c_n \end{bmatrix} = H \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} + H \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

(c with error)

$$= 0 + \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$S = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Page No.
Date :

Syndrome vector corresponds to 3rd column in H matrix. So, the 3rd bit needs to be ~~filled~~ flipped.

if there is a 2-bit ~~or~~ error.

$$r = \begin{bmatrix} c_1 \\ c_{2+1} \\ c_{3+1} \\ \vdots \\ c_n \end{bmatrix}$$

$$H_r = H \begin{bmatrix} c_1 \\ c_{2+1} \\ c_{3+1} \\ \vdots \\ c_n \end{bmatrix}$$

$$= H \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= 0 + \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$S = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

~~So~~ now there is ~~no~~ ~~difference~~ error detected ~~by~~ syndrome vector.

Here, there is an error in 2nd & 3rd bit but syndrome vector is corresponding to 1st bit.

Hence, we cannot correct error with more than 1 bit ~~er~~ error.

Q.1 : (7, 4) Hamming Code

```
%  a.
n=7; k=4;
rate=k/n;

%Generator Matrix and Parity Check Matrix    [G]n*k   [H](n-k)*n

i=eye(k); parity_matrix = [1 1 0 1; 1 0 1 1; 0 1 1 1];

G = [i ; parity_matrix]; %; 1 0 1 1;0 1 1 1]
H = [parity_matrix,eye(n-k)];


message = uint32(0):uint32(power(2,k) -1);
total_message_in_binary = dec2bin(message)- '0';
total_message_in_binary = transpose(total_message_in_binary);
% collection of all possible codewords. Array of k*2^k
total_encoded_messages = mod(G*total_message_in_binary,2)
```

```
total_encoded_messages = 7×16
     0     0     0     0     0     0     0     0     1     1     1     1     1 ···
     0     0     0     0     1     1     1     1     0     0     0     0     1
     0     0     1     1     0     0     1     1     0     0     1     1     0
     0     1     0     1     0     1     0     1     0     1     0     1     0
     0     1     0     1     1     0     1     0     1     0     1     0     0
     0     1     1     0     0     1     1     0     1     0     0     1     1
     0     1     1     0     1     0     0     1     0     1     1     0     1
```

```
%  b.
wmin = n;

%couting wmin
for column = 2:power(2,k)

    count=0;
    for j = 1:n

        count = count + total_encoded_messages(j,column);

    end

    if count < wmin
          wmin=count;
    end
end


%c
% Calculating Min Hamming Distance
dminH = n;
for codeword1 = 1:power(2,k)-1
    for codeword2 = codeword1+1:power(2,k)
        dH=0;
        for row=1:n
            dH = dH+ xor(total_encoded_messages(row,codeword1),...
                total_encoded_messages(row,codeword2));

        end

       if dminH>dH
           dminH=dH;
       end

    end
end
dminH
```

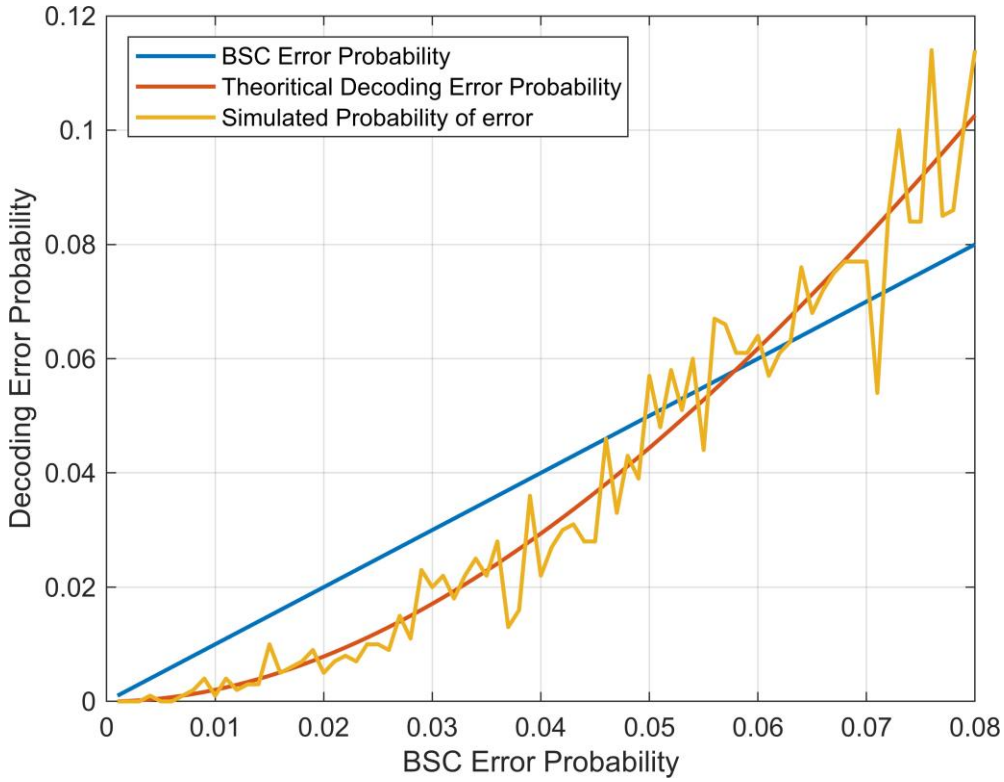
```
dminH = 3
```

Create encoded message

```
%maximum_numbers_possible_transmit_in_decimal=power(2,k)-1;

%d
%nth number codeword = 15
%randmom_message1 = total_message_in_binary(1:k,n);

%% Type your random message of k bits here %%
random_message = [0 0 1 0];
random_message = transpose(random_message);
c= mod(G*random_message,2);      %encoded message of random_message


tc = floor((dminH - 1)/2);

Nsim = 1000;
p = 0.001:0.001:0.08;
p_error_Hamming = zeros(1,length(p));
p_error_Analytical = zeros(1,length(p));

for s=1:length(p)
    favourable=0;
    %Monte Carlo: Repeating same experiment Nsim times
    for z=1:Nsim



        noisy = rand(n,1)<p(s);
        received_message = mod((noisy+c),2);


        %Decoding

        dminH1=n;

        for codeword = 1:power(2,k)
            dH=0;

            for row=1:n
                dH = dH+ xor(total_encoded_messages(row,codeword),...
                    received_message(row,1));
            end

           if dminH1>dH
               dminH1=dH;
               c_calculated = total_encoded_messages(1:n,codeword);
```

```
            end

        end


        flag=0;

        %Comparing transmitted code with the calculated codeword
        %If words are not same, then counting it as errored codeword
        for g=1:n
            if (c_calculated(g,1)~=c(g,1))
                flag=1;
                break;
            end
        end

        if(flag==1)
            favourable=favourable+1;
        end



    end




    p_error_Hamming(1,s) = favourable/Nsim;

    %e
    p_error_Analytical(s) = p_error_dijiye(n,tc,p(s));

end







figure;
plot(p,p,Linewidth=1.5);
hold on;
plot(p,p_error_Analytical,Linewidth=1.5);
hold on;
plot(p,p_error_Hamming,Linewidth=1.5);
ylabel('Decoding Error Probability');
xlabel('BSC Error Probability');
grid on;
legend('BSC Error Probability','Theoritical Decoding Error Probability'...
```

```
    ,'Simulated Probability of error','Location','northwest');
```



Q.2 : Rectangular Parity Check Code

```
%a
%Creating Total Valid Encoded Codes

n=14; k=8;

rate=k/n;
%Generator Matrix and Parity Check Matrix    [G]n*k   [H](n-k)*n


i=eye(k);
parity_matrix = [   1 1 1 1 0 0 0 0;...
                    0 0 0 0 1 1 1 1;...
                    1 0 0 0 1 0 0 0; ...
                    0 1 0 0 0 1 0 0 ;...
                    0 0 1 0 0 0 1 0; ...
                    0 0 0 1 0 0 0 1];

G = [i ; parity_matrix];


%b
total_message_in_binary = (dec2bin(0:power(2,k) -1,8)- '0')';
```

```
% collection of all possible codewords. k*2^k no array
total_encoded_messages = mod(G*total_message_in_binary,2);


H = [parity_matrix,eye(n-k)];


wmin2 = n;

%couting wmin
for column = 2:power(2,k)

    count=0;
    for j = 1:n

        count = count + total_encoded_messages(j,column);

    end

    if count < wmin2
          wmin2=count;
    end
end
wmin2
```

```
wmin2 = 3
```

```
%c
%Calculating Min Hamming Distance
dminH2 = n;
for codeword1 = 1:(2^k)-1
    for codeword2 = codeword1+1:2^k
        dH=0;
        for row=1:n
            dH = dH+ xor(total_encoded_messages(row,codeword1),...
                total_encoded_messages(row,codeword2));

        end

       if dminH2>dH
           dminH2=dH;
       end

    end
end
dminH2
```

```
dminH2 = 3
```

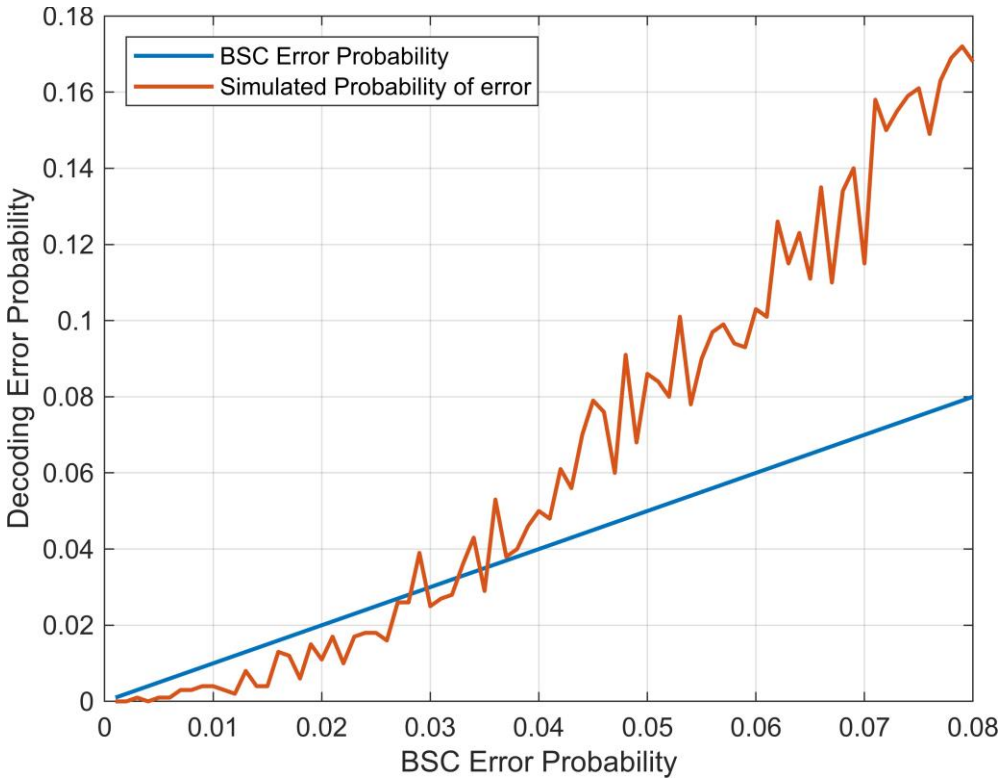
```
maximum_numbers_possible_transmit_in_decimal=power(2,k)-1;
```

```
%% Type your random message of k bits here %%
random_message = [0 0 1 0 0 0 1 1];
random_message = transpose(random_message);
c= mod(G*random_message,2)     %encoded message of random_message
```

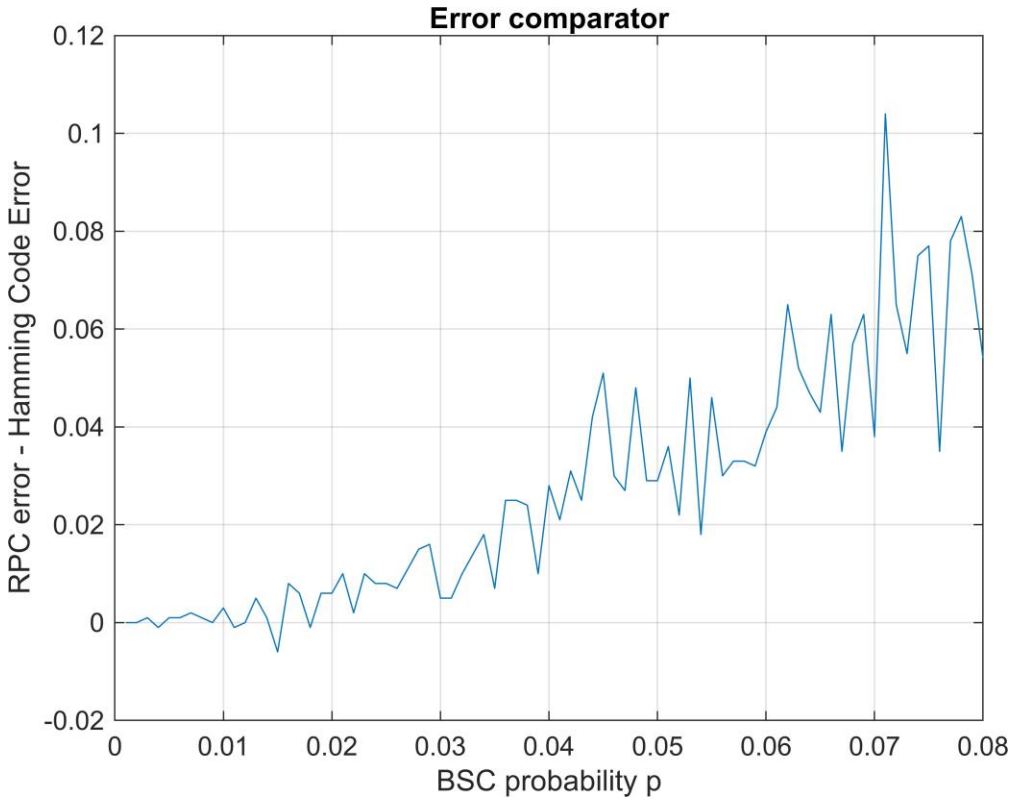
```
c = 14×1
     0
     0
     1
     0
     0
     0
     1
     1
     1
     0
     ⋮
```

```
tc = floor((dminH2 - 1)/2);

Nsim = 1000;
p = 0.001:0.001:0.08;
p_error_rpc = zeros(1,length(p));


for s=1:length(p)
    favourable=0;
    for z=1:Nsim


        noisy = rand(n,1)<p(s);
        received_message = mod((noisy+c),2);


        %Decoding

        dminH21=n;

        for codeword = 1:power(2,k)
            dH=0;

            for row=1:n
                dH = dH+ xor(total_encoded_messages(row,codeword),...
                    received_message(row,1));
            end

           if dminH21>dH
               dminH21=dH;
               c_calculated = total_encoded_messages(1:n,codeword);
```

```
            end

        end


        flag=0;
        %Comparing transmitted code with the calculated codeword
        %If words are not same, then counting it as errored codeword

        for g=1:n
            if (c_calculated(g,1)~=c(g,1))
                flag=1;
                break;
            end
        end

        if(flag==1)
            favourable=favourable+1;
        end

    end




    %d
    p_error_rpc(1,s) = favourable/Nsim;




end

    %Checking if decoded word is same as original codeword

figure;
plot(p,p,Linewidth=1.5);
hold on;
plot(p,p_error_rpc,Linewidth=1.5);
ylabel('Decoding Error Probability');
xlabel('BSC  Error  Probability');
grid on;
legend('BSC Error Probability','Simulated Probability of error','Location','northwest');
```




```
%2.e

figure;

plot(p,p_error_rpc-p_error_Hamming);
ylabel('RPC error - Hamming Code Error');
xlabel('BSC  probability  p');
title('Error comparator');
grid on;
```



Q.3 Square Parity Check Code, over the BEC(p).

```
k=4; n=9;
rate = k/n;

%a
i = eye(k); parity_matrix =[1 1 0 0 ; 0 0 1 1; 1 0 1 0 ;0 1 0 1; 1 1 1 1];

G = [i ; parity_matrix];            % n*k

H = [parity_matrix, eye(n-k)];      %(n-k)*n

%b
% collection of all possible codewords. Array of k*2^k
total_message_in_binary = (dec2bin(0:power(2,k) -1,k)- '0')';
total_encoded_messages = mod(G*total_message_in_binary,2);
```

```
H = [parity_matrix,eye(n-k)];


wmin3 = n;

%couting wmin
for column = 2:power(2,k)

    count=0; %Counting number of ones
    for j = 1:n

        count = count + total_encoded_messages(j,column);

    end

    if count < wmin3
          wmin3=count;
    end
end
wmin3
```

```
wmin3 = 4
```

```
%c
%Calculating Min Hamming Distance
dminH3 = n;
for codeword1 = 1:(2^k)-1
    for codeword2 = codeword1+1:2^k
        dH=0;
        for row=1:n
            dH = dH+ xor(total_encoded_messages(row,codeword1),...
                total_encoded_messages(row,codeword2));

        end

       if dminH3>dH
           dminH3=dH;

       end

    end
end
dminH3
```

```
dminH3 = 4
```

```
%% Type your random message of k bits here %%
random_message = [0 0 1 0];
random_message = transpose(random_message);
c= mod(G*random_message,2)     %encoded message of random_message
```

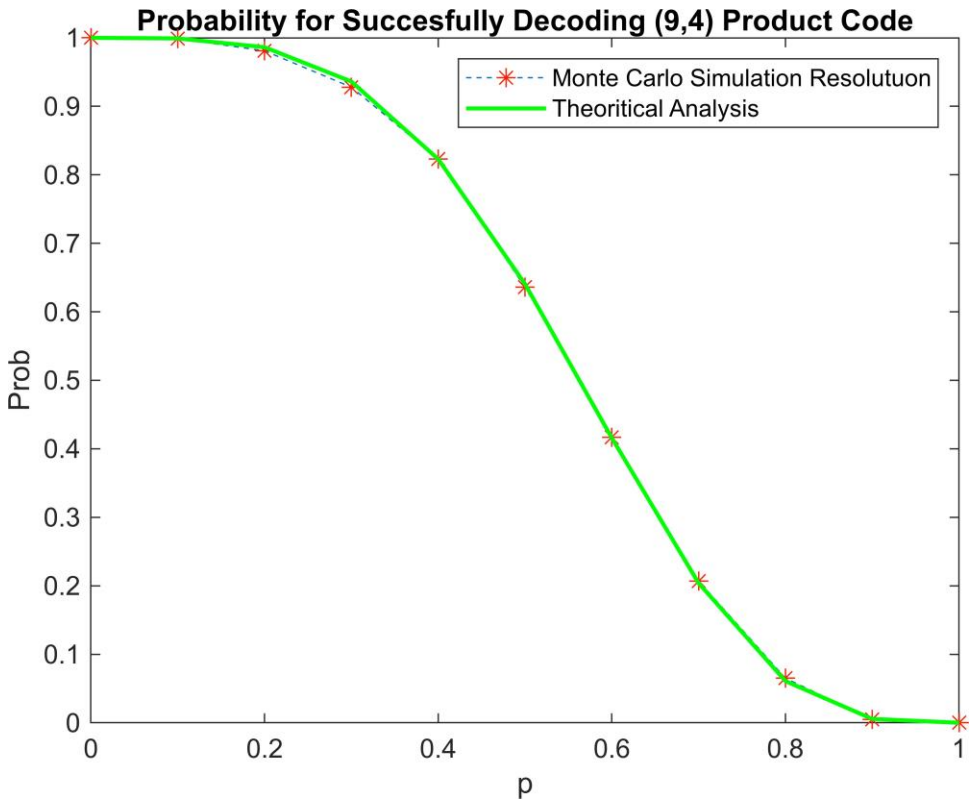
```
c = 9×1
     0
     0
     1
     0
     0
     1
     1
     0
     1
```

```
tc = floor((dminH3 - 1)/2);

Nsim = 1000;


%d using BEC instead of BSC
p=0:0.1:1;

p_error_spc = zeros(1,length(p));
p_error_spc_theoritical = zeros(1,length(p));


for s=1:length(p)
    favourable=0;
    for z=1:Nsim

        %adding noise
        noisy = rand(n,1)<p(s);
        received_message =c;
        for i = 1:n
            if(noisy(i,1)==1)
                received_message(i,1) =-1;
            end
        end


                                    %%%%%%%%%%%%%%%%%%%
                                    %Iterative Decoding%
                                    %%%%%%%%%%%%%%%%%%%
      SPC = [received_message(1,1) received_message(2,1) received_message(5,1); ...
          received_message(3,1) received_message(4,1) received_message(6,1); ...
          received_message(7,1) received_message(8,1) received_message(9,1)];
      number_of_iterations = 0;
      row_done = 0; column_done=0;
      kai_row_done = zeros(3,1);
      kai_column_done = zeros(1,3);
      dony=0;
      while(dony==0 && number_of_iterations < n/2)

            %row by row
            if(row_done<3)
```

```
        for rowr = 1:3
            if(kai_row_done(rowr,1)==0)
                counter= 0;    %checking number of -1

                for rcolumn = 1:3

                    if(SPC(rowr,rcolumn)==-1)
                        counter=counter+1;

                    end

                end

                if(counter==0)
                    row_done=row_done+1;
                    kai_row_done(rowr,1)=1;
                end

                if(counter==1)
                    column_to_correct =0;
                    value=0;
                    for column2r = 1:3

                        if(SPC(rowr,column2r)==-1)
                            column_to_correct=column2r;
                            continue;
                        end

                        value = value + SPC(rowr,column2r);

                    end
                    SPC(rowr,column_to_correct) = mod(value,2);
                    row_done=row_done+1;
                    kai_row_done(rowr,1)=1;
                end



            end

        end
    end

    %column by column


    if(column_done<3)

        for columnc = 1:3
```

```
            if(kai_column_done(1,columnc)==0)




                counter= 0;    %checking number of -1

                for rowc = 1:3

                    if(SPC(rowc,columnc)==-1)
                        counter=counter+1;

                    end

                end

                if(counter==0)
                    column_done=column_done+1;
                    kai_column_done(1,columnc)=1;
                end

                if(counter==1)
                    row_to_correct =0;
                    value=0;
                    for row2c = 1:3

                        if(SPC(row2c,columnc)==-1)
                            row_to_correct=row2c;
                            continue;
                        end

                        value = value + SPC(row2c,columnc);

                    end
                    SPC(row_to_correct,columnc) = mod(value,2);
                    column_done=column_done+1;
                    kai_column_done(1,columnc)=1;
                end
            end
        end

    end



    if(row_done==3 && column_done==3)
        dony=1;
```

```
            end
            number_of_iterations=number_of_iterations+1;
        end


        %decoding complete!!!!!


        if dony==1
            favourable = favourable+1;
        end


    end


    %d
    p_error_spc(1,s) = favourable/Nsim;

    p_error_spc_theoritical(1,s) = getmePtheoretical(p(s));
end

figure;
ploty = plot(p,p_error_spc);
ploty.LineStyle = '--';
ploty.Marker = '*';
ploty.MarkerEdgeColor = [1 0.1 0];
ploty.MarkerSize = 7.2;
ylabel('Prob');
xlabel('p');
title('Probability for Succesfully Decoding (9,4) Product Code');
hold on;
plot(p,p_error_spc_theoritical,'g',LineWidth=1.5);
legend('Monte Carlo Simulation Resolutuon','Theoritical Analysis');
```




```
function answ = p_error_dijiye(n,tc,p)

    answ=0;
    for i=0:tc
        answ = answ + nchoosek(n,i)*power(p,i)*power(1-p,n-i);
    end
```

```
    answ=1-answ;
end


function answ = getmePtheoretical(p)
    answ=0;

    for i = 0:3
        answ = answ + nchoosek(9,i)*power(p,i)*power(1-p,9-i);
    end

    answ = answ + 117*power(p,4)*power(1-p,5) + 81*power(p,5)*power(1-p,4);

end
```