CT216-Lab06

Rakshit Pandhi-202201426

March 3, 2024

1 Analysis Exercise

Lab-6

* Analysis Exercise

1. We applied WLLN for showing that probability of qN success is :-

$$q^{qN}(1-q)^{N-qN}$$

For finding no of such sequences we used :-

$$k \times q^{qN}(1-q)^{N-qN} = 1 \quad \text{(Prob. of whole typical set)}$$

$$\therefore k = q^{-qN}(1-q)^{qN-N}$$

Taking $\log_2()$ both sides →

$$\log_2 k = -qN\log_2 q + (qN-N)\log_2(1-q)$$

$$= -qN\log_2 q + qN\log_2(1-q) - N\log_2(1-q)$$

$$= +N\left(-q\log_2 q - (1-q)\log_2(1-q)\right)$$

$$= NH_2(q)$$

$$\therefore \boxed{k = 2^{NH_2(q)}} \quad \text{—(i)}$$

Now, using Sterling's approx. →

$$\log_2 \binom{N}{r} \approx N H_2\left(\frac{r}{N}\right)$$

Here $r = qN$

$$\therefore \log_2 \binom{N}{qN} \approx N H_2(q)$$

$$\therefore \binom{N}{qN} \approx 2^{N H_2(q)} \quad \text{---(ii)}$$

Therefore from (i) & (ii) we get the same result that # of such binary sequences $\left(\binom{N}{qN}\right)$ is $2^{N H_2(q)}$ as shown directly from Shannon's Noiseless Channel Coding.

2.

(a) We are asked the # of times X_k occur, which is basically expectatⁿ of X_k in the sequence.

$$\therefore E[X_k] = N p_k$$

Here as $N \to \infty$

$$\therefore E[X_k] = \lim_{N \to \infty} N p_k$$

(b) Probability of these sequences can be calculated as :-

$$P_{sequence} = \lim_{N \to \infty} \prod_{k=1}^{K} p_k^{N p_k} \quad \left[\begin{array}{l}(N \to \infty,\ P_{sequence} \to 1) \\ (\text{We know } E[x] = N p_k)\end{array}\right.$$

Teacher's Signature:.................

(c) Since our Universal set has been cutoff from $\mathcal{X}^N$, so we wouldn't see any other sequence other than the typical sequence

Hence →

$$Z \cdot \lim_{N\to\infty} \prod_{k=1}^{K} p_k^{N p_k} = 1 \quad (Z \text{ is \# of typical sequences})$$

Taking $\log_2()$ both sides:

$$\therefore Z = \lim_{N\to\infty} \prod_{k=1}^{K} p_k^{-N p_k}$$

$$\log_2(Z) = -N \sum_{k=1}^{K} p_k \log_2(p_k)$$

Entropy of K-ary source $= H_2(X) = -\sum_{k=1}^{K} p_k \log_2(p_k)$

~~$\therefore \log_2(X) = N H_2(X)$~~

$$\therefore \log_2(Z) = -N(-H_2(X))$$

$$\therefore \boxed{Z = 2^{N H_2(X)}}$$

* Part - 2

1. Linearity Property says if c_1 & c_2 are valid codeword then so are $c_1 \pm c_2$.

Let $c_1 = [c_{1,1}\ c_{1,2} \ldots c_{1,n}]^T$

$c_2 = [c_{2,1}, c_{2,2}, \ldots c_{2,n}]^T$

As c_1 & $c_2 \rightarrow$ codeword $Hc_1 = 0$ & $Hc_2 = 0$

$c_1 + c_2 = [c_{1,1} + c_{2,1}, c_{1,2} + c_{2,2} \ldots \ldots c_{1,n} + c_{2,n}]^T$

$c_1 - c_2 = [c_{1,1} - c_{2,1}, c_{1,2} - c_{2,2} \ldots \ldots c_{1,n} - c_{2,n}]^T$

$[H]_{k \times n}[c_1 + c_2]_{n \times 1} = [H][c_1] + [H][c_2]$ (∵ Using Linearity property of Matrix Multiplication)

$= \cancel{H}\ 0 + 0$

$= \boxed{0}$

Similarly

$[H]_{k \times n}[c_1 - c_2]_{n \times 1} = 0 - 0 = 0$

Hence as $S_{c_1+c_2} = 0 = S_{c_1-c_2} \rightarrow c_1 + c_2$ & $c_1 - c_2$ are also valid codeword.

Teacher's Signature:......

2. Syndrome Prop: I says that if code is linear S remains unchanged when transmitter sends c' or c.

Let

$r = [r_1, r_2 \ldots r_n]^T$, $c = [c_1, c_2 \ldots c_n]^T$,

$e = [e_1, e_2 \ldots e_n]^T$

$r = c + e \rightarrow r_1 = c_1 + e_1, r_2 = c_2 + e_2 \ldots \rightarrow r_n = c_n + e_n$

As c is valid codeword

$S_{r_c} = [H][r]$

$S_{r_c} = [H][c+e]$

$= [H][c] + [H][e]$

$= 0 + S_e$

If any other codeword c'

then

$S_{r_{c'}} = [H][r']$

$= [H][c'+e]$

$= [H][c'] + [H][e]$

$= 0 + S_e$

$\therefore S_{r_c} = S_{r_{c'}}$

$\therefore$ We showed that if code is linear S vector depends only on e vector.

Teacher's Signature

3. Syndrom Prop II says that we can determine the errors by calculating S (unless $c = e$)

Let us say there is no error

$\therefore r = c$

$\therefore S = [H][r]$

$= [H][c]$

$S = 0$ (as $c \rightarrow$ valid codeword)

Now, introducing error $\rightarrow r' = c + e$

$\therefore S = [H][r']$

$= [H][c] + [H][e]$

$\boxed{S = 0 + S_e \neq 0}$

As $S \neq 0 \rightarrow$ error detected.

But if we take $e = c'$ (c' is valid codeword)

$r'' = c + e = c + c' = c''$ (valid codeword)

(As we proved that $c + c'$ is also a valid codeword using linearity)

$\therefore S = [H][r'']$

$= [H][c + c']$

$= [H][c'']$

$\boxed{S = 0}$

Hence proved

4. All zero $(n\times 1)$ is always a valid codeword. (regardless of G & H)

$\therefore z_{n\times 1} = [0, \ldots 0]^T$

A codeword should satisfy
$s = [H][C] = 0$

Here
$s_{k\times 1} = H_{k\times n} z_{n\times 1} = 0$ ([0] vector * any vector is [0])

$\therefore$ All Zero vector is always a codeword regardless what G & H are.

5. To show that $d^H_{min} = w_{min}$ let us prove through contradiction by taking $d^H_{min} \leq w_{min}$.

$c_1 = z_{n\times 1}$ (all zero codeword → valid)
$c_2 =$ min. Hamming wt codeword

As we took $d^H_{min} > w_{min}$, $\therefore$ there exists 2 sequences who distance is greater than w_{min}.

But as we took c_1 & c_2 as $c_1 = z_{n\times 1}$ &
$c_2 = c_{w_{min}}$

The distance b/w them is equal to w_{min}, which is lesser than what we assumed.

Hence $d^H_{min} > w_{min}$ is false

Teacher's Signature

Now $d^H_{min} < w_{min}$

This can also be proved false in the same way by taking $c_1 = [0]_{n \times 1}$ & $c_2 = c_{w_{min}}$.

We are assuming $d^H_{min} < w_{min}$, but for c_1 & c_2 $d^H = w_{min}$ which in turn is d^H_{min}

$\boxed{\text{Hence only case possible is } d^H_{min} = w_{min} \text{ for linear codes.}}$

6. $r = \frac{4}{7}$ $\therefore k = 4 ; n = 7 ; q = n - k = 3$

Let parity $\Rightarrow$
$$\left.\begin{aligned} p_1 &= m_0 + m_1 + m_2 \\ p_2 &= m_1 + m_2 + m_3 \\ p_3 &= m_0 + m_1 + m_3 \end{aligned}\right\} \text{modulo-2 sum.}$$

$$\therefore H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}_{3 \times 7}$$

Let us take 1 msg. $M_1 = [\underset{m_0}{1}\ \underset{m_1}{0}\ \underset{m_2}{1}\ \underset{m_3}{1}]^T$ $(p_1 = 0 ; p_2 = 0 ; p_3 = 0)$

$\boxed{c_1 = [1011000]^T}$

$$s = [H][c_1] = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix}_{\text{mod } 2} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Valid

Teacher's Signature:

DATE / /

Now let us say 1 bit got flipped during transmission:

$$c_2 = [1001000]^T \text{ instead of } [1011000]$$

$$\therefore s = [H][c_2]$$

$$= \begin{bmatrix} & " & \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}_{mod 2}$$

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

Error detected. And we can check "where" by taking help from P_1, P_2, P_3

Since for 1 & 2 error is detected, P_1 & P_2 has some anamoly in it.

⇒ Since P_3 is correct, m_0, m_1, m_3 are considered to be correct, error lies in m_2. Hence can be corrected by.

* Now if we introduce 1 more error (2-bit error) i.e.

$$c_3 = [1101000]^T$$

$$s = [H][c_3]$$

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

Teacher's Signature:

PAGE NO.
DATE / /

So here 1 & 2 are received correctly as 1^{st} 2 rows of s are 0 but 3^{rd} row has 1 so some error regarding ~~B~~ or we can say in m_0, m_1, m_2

But as P_1 & P_2 has no anamoly, so we wouldn't be able to make out which bit is in error.

→ Hence we were able to correct & detect for 1 bit error but only detect for 2-bit error.

$\therefore t_c = 1$ bit

Teacher's Signature.........

```
disp('Question 1');
```

```
Question 1
```

```
disp('(7,4) Hamming Code');
```

```
(7,4) Hamming Code
```

```
n=7;
k=4;
u=n-k;

%{
 We define parity as:-
p1=m0+m1+m2
p2=m1+m2+m3
p3=m0+m1+m3

Therefore first 4 rows are 4x4 identity matrix and next 3 rows are the
parity checkers.
%}

% Parity matrix
P=[
   1 1 0 1;
   1 0 1 1;
   0 1 1 1;
   ]
```

```
P = 3×4
     1     1     0     1
     1     0     1     1
     0     1     1     1
```

```
identity_matrix=eye(k);
G=[identity_matrix;P];
disp('Generator Matrix:');
```

```
Generator Matrix:
```

```
disp(G);
```

```
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
     1     1     0     1
     1     0     1     1
     0     1     1     1
```

```
identity_parity=eye(u);
H=[P identity_parity];
```

```
disp('Parity Check Matrix:');
```

```
Parity Check Matrix:
```

```
disp(H);
```

```
     1     1     0     1     1     0     0
     1     0     1     1     0     1     0
     0     1     1     1     0     0     1
```

```
decnum=0:(2^k-1);
binnum=dec2bin(decnum);
M=zeros(k,length(decnum));

for i=1:length(decnum)
    M(:,i)=binnum(i,:)-'0';
end

disp('Matrix representing all possible length k(here 4) sequences:');
```

```
Matrix representing all possible length k(here 4) sequences:
```

```
disp(M);
```

```
     0     0     0     0     0     0     0     0     1     1     1     1     1     1     1     1
     0     0     0     0     1     1     1     1     0     0     0     0     1     1     1     1
     0     0     1     1     0     0     1     1     0     0     1     1     0     0     1     1
     0     1     0     1     0     1     0     1     0     1     0     1     0     1     0     1
```

```
C=mod(G*M,2);
disp('All possible Codeword Matrix(length 7):');
```

```
All possible Codeword Matrix(length 7):
```

```
disp(C);
```

```
     0     0     0     0     0     0     0     0     1     1     1     1     1     1     1     1
     0     0     0     0     1     1     1     1     0     0     0     0     1     1     1     1
     0     0     1     1     0     0     1     1     0     0     1     1     0     0     1     1
     0     1     0     1     0     1     0     1     0     1     0     1     0     1     0     1
     0     1     0     1     1     0     1     0     1     0     1     0     0     1     0     1
     0     1     1     0     0     1     1     0     1     0     0     1     1     0     0     1
     0     1     1     0     1     0     0     1     0     1     1     0     1     0     0     1
```

```
w=sum(C);
w_min=min(w(2:length(C)));
disp(['Minimum Hamming Weight=',num2str(w_min)]);
```

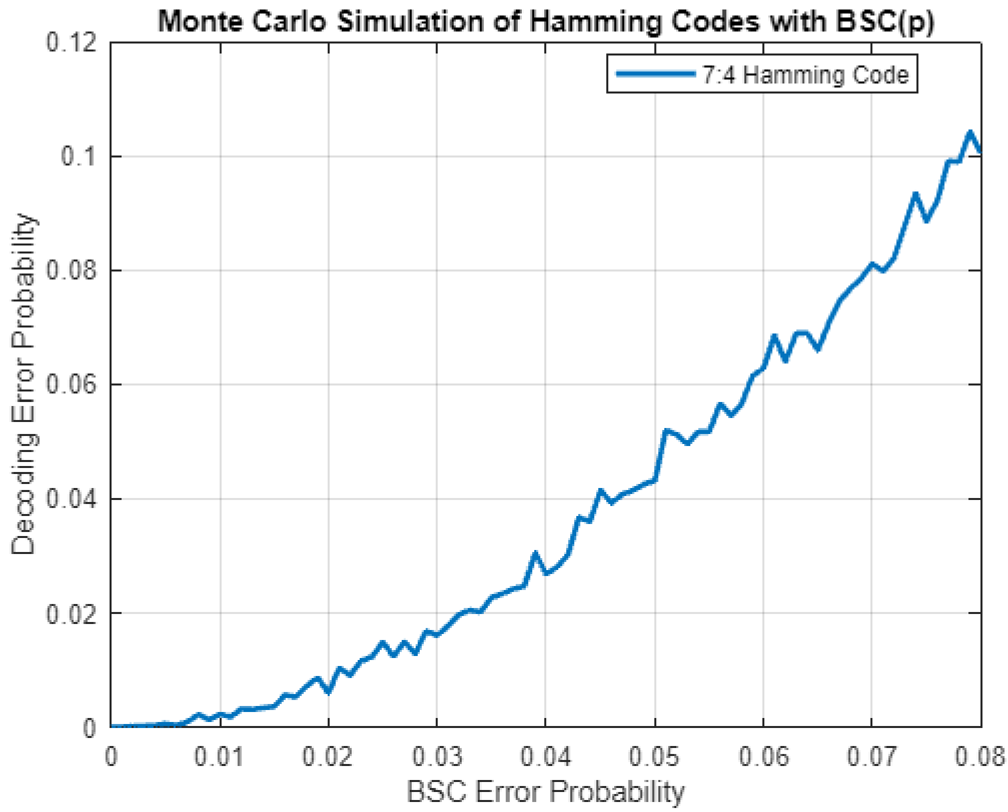
```
Minimum Hamming Weight=3
```

```


d_hmin=k;
for cdwrd1=1:length(C)
    for cdwrd2=cdwrd1+1:length(C)
```

```
        d_h=sum(xor(C(:,cdwrd1),C(:,cdwrd2)));
        d_hmin=min(d_h,d_hmin);
    end
end

disp(['Minimum Hamming Distance:',num2str(d_hmin)]);
```

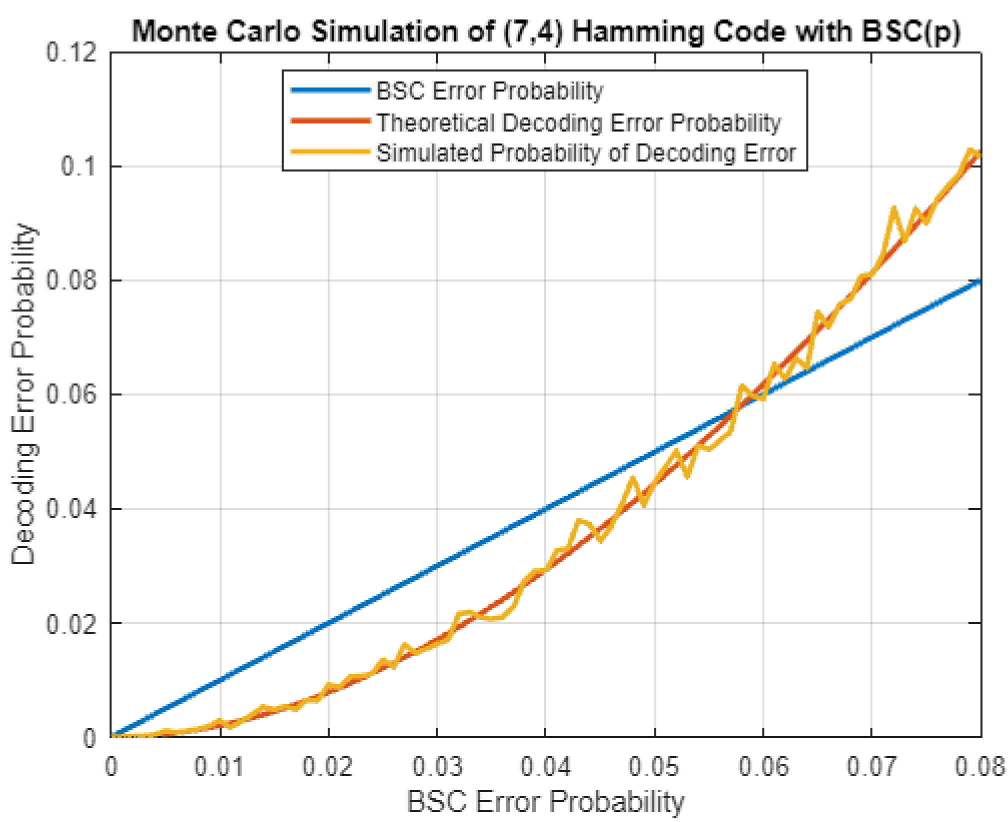
```
Minimum Hamming Distance:3
```

```
err=zeros(size(0:0.001:0.08));
nsim=1e4;

for p = 0:0.001:0.08 % Iterate over different error probabilities
for ksim = 1:nsim
m_in = randi([1, 2^k]); % Randomly select a message index
m = M(:, m_in); % Get the message corresponding to the index
c = C(:, m_in); % Get the codeword corresponding to the message
% Introduce BSC noise
bsc_noise = rand(size(c)) < p;
r = mod(c + bsc_noise, 2); % Received codeword
e=r-C;
answer = sum(abs(e));
[~,est_index] = min(answer);
% Check for error
if est_index ~= m_in
err(round(p * 1000) + 1) = err(round(p * 1000) + 1) + 1; % Increment error count
end
end
end
% Convert error counts to error probabilities
err_prob= err / nsim;
% Plotting
p_values = 0:0.001:0.08;
plot(p_values, err_prob,'LineWidth',2);
xlabel('BSC Error Probability');
legend('7:4 Hamming Code','location','best');
ylabel('Decoding Error Probability');
title('Monte Carlo Simulation of Hamming Codes with BSC(p)');
grid("on");
```




```
err = zeros(size(0:0.001:0.08)); % Simulated error count
theoretical_err = zeros(size(0:0.001:0.08)); % Theoretical error probability
nsim = 1e4; % Number of simulations
t_c=1;
for p = 0:0.001:0.08 % Iterate over different error probabilities
% Calculate theoretical probability of decoding error
% Hamming code minimum distance
d_min = 3;
% Theoretical probability of decoding error for Hamming code
for t = t_c+1:n
theoretical_err(round(p * 1000) + 1) = theoretical_err(round(p * 1000) + 1) +
nchoosek(n, t) * p^t * (1-p)^(n-t);
end
for ksim = 1:nsim
m_in = randi([1, 16]); % Randomly select a message index
m = M(:, m_in); % Get the message corresponding to the index
c = C(:, m_in); % Get the codeword corresponding to the message
% Introduce BSC noise
bsc_noise = rand(size(c)) < p;
r = mod(c + bsc_noise, 2); % Received codeword
e=r-C;
answer = sum(abs(e));
[~,est_index] = min(answer);
% Check for error
if est_index ~= m_in
```

```
err(round(p * 1000) + 1) = err(round(p * 1000) + 1) + 1; % Increment error count
end
end
end
% Convert error counts to error probabilities
err_prob = err / nsim;
% Plotting
p_values = 0:0.001:0.08;
plot(p_values, p_values,'Color', '#0072BD', 'LineWidth', 2);
hold on;
plot(p_values, theoretical_err,'Color', '#D95319', 'LineWidth', 2);
plot(p_values, err_prob,'Color', '#EDB120', 'LineWidth', 2);
legend('BSC Error Probability', 'Theoretical Decoding Error Probability','Simulated
Probability of Decoding Error',Location='best');
xlabel('BSC Error Probability');
ylabel('Decoding Error Probability');
title('Monte Carlo Simulation of (7,4) Hamming Code with BSC(p)');
grid on;
hold off;
```




```
disp('Question 2');
```

```
Question 2
```

```
disp('(14,8) RPC Code');
```

(14,8) RPC Code

```
n=14;
k=8;
u=n-k;

P=[
   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;
   1 1 1 1 0 0 0 0;
   0 0 0 0 1 1 1 1];

identity_matrix=eye(k);
G=[identity_matrix; P];
disp('Generator Matrix:');
```

```
Generator Matrix:
```

```
disp(G);
```

```
     1     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0
     0     0     1     0     0     0     0     0
     0     0     0     1     0     0     0     0
     0     0     0     0     1     0     0     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     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
     1     1     1     1     0     0     0     0
     0     0     0     0     1     1     1     1
```

```

identity_parity=eye(u);
H=[P identity_parity];
disp('Parity Check Matrix:');
```

```
Parity Check Matrix:
```

```
disp(H);
```

```
     1     0     0     0     1     0     0     0     1     0     0     0     0     0
     0     1     0     0     0     1     0     0     0     1     0     0     0     0
     0     0     1     0     0     0     1     0     0     0     1     0     0     0
     0     0     0     1     0     0     0     1     0     0     0     1     0     0
     1     1     1     1     0     0     0     0     0     0     0     0     1     0
     0     0     0     0     1     1     1     1     0     0     0     0     0     1
```

```
decnum=0:(2^k-1);
```

```
binnum=dec2bin(decnum);
M=zeros(k,length(decnum));

for i=1:length(decnum)
    M(:,i)=binnum(i,:)-'0';
end

disp('Matrix representing all possible length k(here 8) sequences:');
```

```
Matrix representing all possible length k(here 8) sequences:
```

```
disp(M);
```

```
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     1
     0     0     0     0     0     0     0     0     1     1     1     1     1     1     1     1     0     0     0
     0     0     0     0     1     1     1     1     0     0     0     0     1     1     1     1     0     0     0
     0     0     1     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     1     0     1     0     1     0     1     0
```

```
C=mod(G*M,2);
disp('All possible Codeword Matrix(length 14):');
```

```
All possible Codeword Matrix(length 14):
```

```
disp(C);
```

```
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     1
     0     0     0     0     0     0     0     0     1     1     1     1     1     1     1     1     0     0     0
     0     0     0     0     1     1     1     1     0     0     0     0     1     1     1     1     0     0     0
     0     0     1     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     1     0     1     0     1     0     1     0
     0     0     0     0     0     0     0     0     1     1     1     1     1     1     1     1     0     0     0
     0     0     0     0     1     1     1     1     0     0     0     0     1     1     1     1     0     0     0
     0     0     1     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     1     0     1     0     1     1     0     1
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     1     1     1
     0     1     1     0     1     0     0     1     1     0     0     1     0     1     1     0     0     1     1
```

```
w=sum(C);
w_min=min(w(2:length(C)));
disp(['Minimum Hamming Weight=',num2str(w_min)]);
```

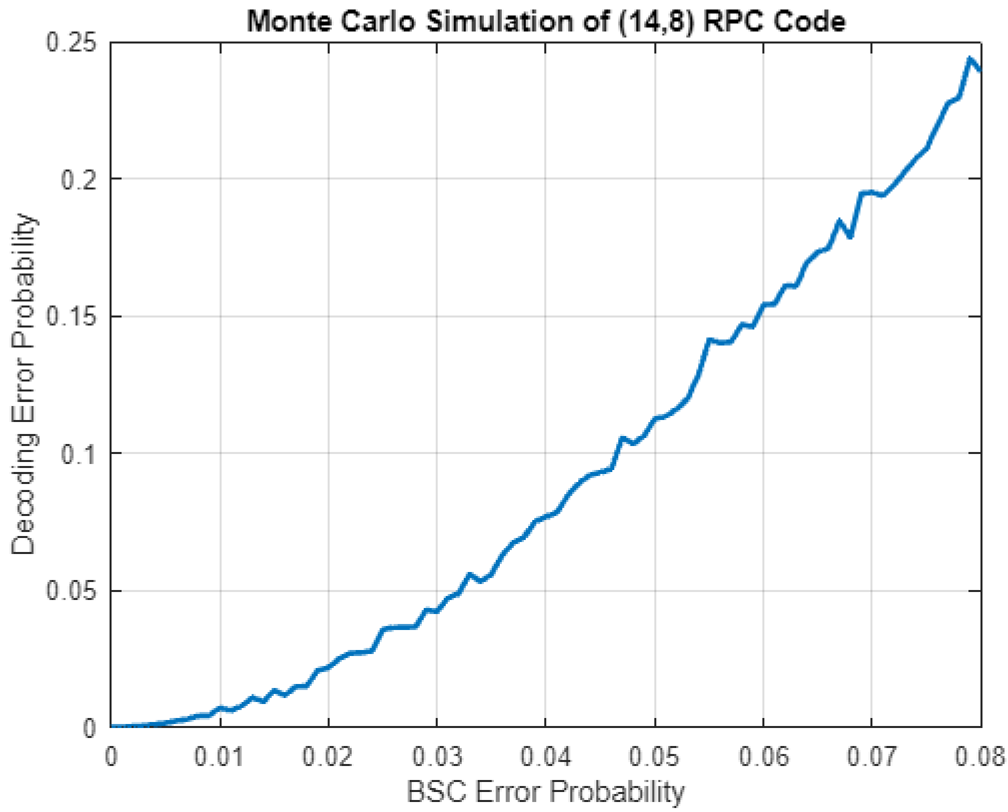
```
Minimum Hamming Weight=3
```

```

d_hmin=k;
for cdwrd1=1:length(C)
    for cdwrd2=cdwrd1+1:length(C)
        d_h=sum(xor(C(:,cdwrd1),C(:,cdwrd2)));
        d_hmin=min(d_h,d_hmin);
```

```
    end
end

disp(['Minimum Hamming Distance:',num2str(d_hmin)]);
```

```
Minimum Hamming Distance:3
```

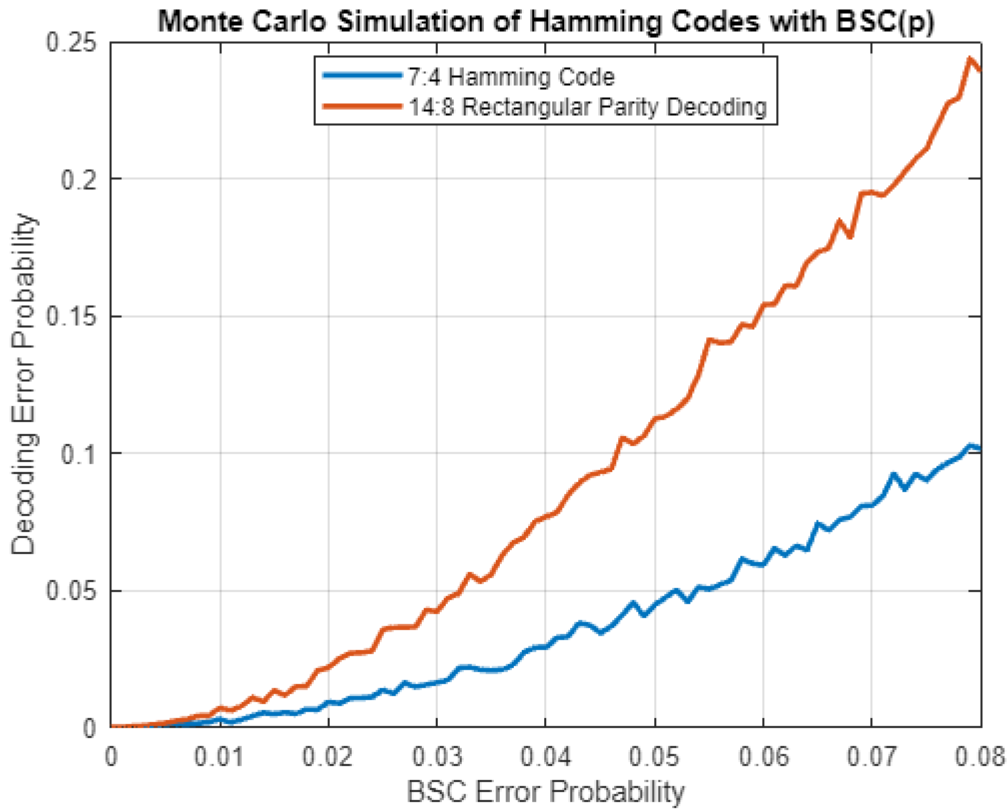
```
err=zeros(size(0:0.001:0.08));
nsim=1e4;

for p = 0:0.001:0.08 % Iterate over different error probabilities
for ksim = 1:nsim
m_in = randi([1, 2^k]); % Randomly select a message index
m = M(:, m_in); % Get the message corresponding to the index
c = C(:, m_in); % Get the codeword corresponding to the message
% Introduce BSC noise
bsc_noise = rand(size(c)) < p;
r = mod(c + bsc_noise, 2); % Received codeword
e=r-C;
answer = sum(abs(e));
[~,est_index] = min(answer);
% Check for error
if est_index ~= m_in
err(round(p * 1000) + 1) = err(round(p * 1000) + 1) + 1; % Increment error count
end
end
end
% Convert error counts to error probabilities
err_prob_1= err / nsim;
% Plotting
p_values = 0:0.001:0.08;
plot(p_values, err_prob_1,'LineWidth',2);
xlabel('BSC Error Probability');
ylabel('Decoding Error Probability');
title('Monte Carlo Simulation of (14,8) RPC Code');
grid("on");
```




```
% Comparing RPC and Hamming Code
disp('Comparing Hamming Code with RPC Code.');
```

```
Comparing Hamming Code with RPC Code.
```

```
figure;
p_values = 0:0.001:0.08;
plot(p_values, err_prob,'LineWidth',2);
hold on;
plot(p_values, err_prob_1,'LineWidth',2);
xlabel('BSC Error Probability');
legend('7:4 Hamming Code','14:8 Rectangular Parity Decoding','location','best');
ylabel('Decoding Error Probability');
title('Monte Carlo Simulation of Hamming Codes with BSC(p)');
grid("on");
hold off;
```




```
% From the above graph it is clear that the Hamming Code performs better
% though the rate are same for both.
% The reason behind this lies in the theory of perfect codes.
%{
r=4/7 and r=8/14(4/7) both have same rate but to check whether one is a
perfect code or not we show:
2^k.Volume(n,tc)=2^n or not where tc= # of correction bits and
Volume(n,tc)=sum(n choose i) { i=0 to tc)

For r=4/7
there are 16 hamming spheres and 16*8 binary sequences.
where 2^n=16*18=128
Hence a perfect Code

For r=4/7 but with k=8 and n=14
2^8 * 15 cannot be 2^14
Hence not a perfect code

Hence Hamming Code is better than RPC Code
%}
```

```
disp('Question 3');
```

Question 3

```
disp('(9,4) Product Code');
```

```
(9,4) Product Code
```

```
n=9;
k=4;
u=n-k;

P=[
  1 0 1 0;
  0 1 0 1;
  1 1 0 0;
  0 0 1 1;
  1 1 1 1];

identity_matrix=eye(k);
identity_parity=eye(u);
G=[identity_matrix; P];
H=[P identity_parity];
disp('Generator Matrix:');
```

```
Generator Matrix:
```

```
disp(G);
```

```
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
     1     0     1     0
     0     1     0     1
     1     1     0     0
     0     0     1     1
     1     1     1     1
```

```
disp('Parity Check Matrix:');
```

```
Parity Check Matrix:
```

```
disp(H);
```

```
     1     0     1     0     1     0     0     0     0
     0     1     0     1     0     1     0     0     0
     1     1     0     0     0     0     1     0     0
     0     0     1     1     0     0     0     1     0
     1     1     1     1     0     0     0     0     1
```

```
decnum=0:(2^k-1);
binnum=dec2bin(decnum);
M=zeros(k,length(decnum));

for i=1:length(decnum)
```

```
    M(:,i)=binnum(i,:)-'0';
end

disp('Matrix representing all possible length k(here 4) sequences:');
```

```
Matrix representing all possible length k(here 4) sequences:
```

```
disp(M);
```

```
     0     0     0     0     0     0     0     0     1     1     1     1     1     1     1     1
     0     0     0     0     1     1     1     1     0     0     0     0     1     1     1     1
     0     0     1     1     0     0     1     1     0     0     1     1     0     0     1     1
     0     1     0     1     0     1     0     1     0     1     0     1     0     1     0     1
```

```
C=mod(G*M,2);
disp('All possible Codeword Matrix(length 9):');
```

```
All possible Codeword Matrix(length 7):
```

```
disp(C);
```

```
     0     0     0     0     0     0     0     0     1     1     1     1     1     1     1     1
     0     0     0     0     1     1     1     1     0     0     0     0     1     1     1     1
     0     0     1     1     0     0     1     1     0     0     1     1     0     0     1     1
     0     1     0     1     0     1     0     1     0     1     0     1     0     1     0     1
     0     0     1     1     0     0     1     1     1     1     0     0     1     1     0     0
     0     1     0     1     1     0     1     0     0     1     0     1     1     0     1     0
     0     0     0     0     1     1     1     1     1     1     1     1     0     0     0     0
     0     1     1     0     0     1     1     0     0     1     1     0     0     1     1     0
     0     1     1     0     1     0     0     1     1     0     0     1     0     1     1     0
```

```
w=sum(C);
w_min=min(w(2:length(C)));
disp(['Minimum Hamming Weight=',num2str(w_min)]);
```

```
Minimum Hamming Weight=4
```

```

d_hmin=k;
for cdwrd1=1:length(C)
    for cdwrd2=cdwrd1+1:length(C)
        d_h=sum(xor(C(:,cdwrd1),C(:,cdwrd2)));
        d_hmin=min(d_h,d_hmin);
    end
end

disp(['Minimum Hamming Distance:',num2str(d_hmin)]);
```

```
Minimum Hamming Distance:4
```

```
err = zeros(size(0:0.1:1));% Initialize array to store error counts for each p
theoretical_prob = zeros(size(0:0.1:1));
nsim = 1000; % Number of simulations
```

```
for p = 0:0.1:1 % Iterate over different erasure probabilities
    for ksim = 1:nsim
        m_in = randi([1, 2^k]); % Randomly select a message index
        m = M(:, m_in); % Get the message corresponding to the index
        c = C(:, m_in); % Get the codeword corresponding to the message
        bec_noise = rand(size(c)) < p;
        % Apply BEC noise
        num = 5 * bec_noise; % Treat terms with more than 1 erasure as erasures
        r = num + c; % Received codeword
        % Set values greater than 4 to NaN
        r(r > 4) = NaN;
        theoretical_prob = binopdf(0, n, p_values) + binopdf(1, n, p_values) +
binopdf(2, n, p_values) + binopdf(3, n, p_values) + binopdf(4, n, p_values) +
binopdf(5, n, p_values) / 2;
        % Reshape the received codeword into a 3x3 matrix
        r_reshaped = [r(1), r(2), r(5); r(3), r(4), r(6); r(7), r(8), r(9)];

        % Perform iterative decoding
        for iter = 1:3 % Iterations for both row and column decoding
            % Column decoding
            for col = 1:3
                nan_indices = find(isnan(r_reshaped(:, col)));
                if length(nan_indices) == 1 % Exactly one NaN in the column
                % Find non-NaN indices
                    non_nan_indices = find(~isnan(r_reshaped(:, col)));
                    % XOR the other two elements
                    r_reshaped(nan_indices, col) =
mod(sum(r_reshaped(non_nan_indices, col)), 2);
                end
            end
            % Row decoding
            for row = 1:3
                nan_indices = find(isnan(r_reshaped(row, :)));
                if length(nan_indices) == 1 % Exactly one NaN in the row
                    % Find non-NaN indices
                    non_nan_indices = find(~isnan(r_reshaped(row, :)));
                    % XOR the other two elements
                    r_reshaped(row, nan_indices) =
mod(sum(r_reshaped(row,non_nan_indices)), 2);
               end
           end
        end
        % Check for remaining NaNs
        if any(isnan(r_reshaped), 'all')
            err(round(p * 10) + 1) = err(round(p * 10) + 1) + 1; % Increment error
count
        end
    end
end
% Convert error counts to error probabilities
```

```
err_prob = err / (nsim);
succ_prob = zeros(size(0:0.1:1)); % Initialize array to store error counts for each
p
succ_prob = 1-err_prob;
% Plotting
p_values = 0:0.1:1;
plot(p_values, succ_prob, 'r--', 'LineWidth', 2);
hold on; % Hold the plot to add more elements
% Plotting the blue dots
scatter(p_values, succ_prob, 'b', 'filled');
plot(p_values, theoretical_prob, 'g', 'LineWidth', 2);
hold off; % Release the hold
xlabel('p');
ylabel('Prob');
title('Probability of Successful Decoding for (9,4) Product Code');
ylim([0 1.2]);
grid on;
% Adding legend
legend('Success Probability (Red Dotted Line)', 'Success Probability Points (Blue
Dots)','Theoretical Analysis');
```