CT-216

Introduction To Communication Systems

Lab-6

Kashish Patel

ID:202101502

HONOUR CODE:

I, Kashish Patel, ID :- 202101502 , declare that

→ the work that I am presenting is done by me and my lab partner Hardik (ID :- 202101506)

→ We have not copied the work (Matlab code, results, etc.) that someone else has done

→ Concepts, understanding and insights we would be describing are our own

→ Wherever we have relied on an existing work that is not our own, We have provided a proper reference citation

→ We make this pledge truthfully. We know that violation of this solemn pledge can carry grave consequences

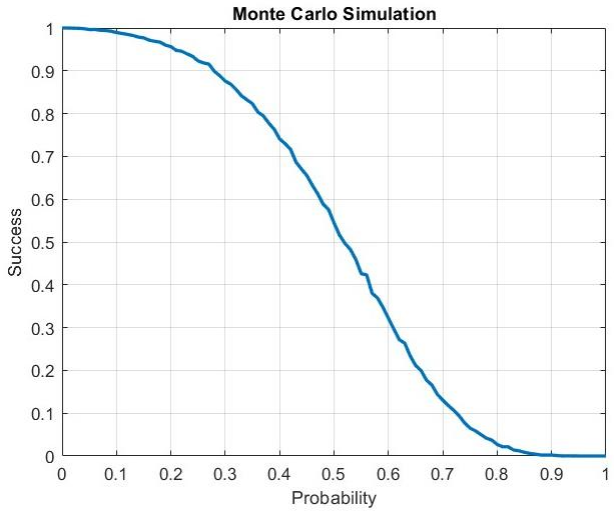
- We both have already demonstrated our code to sir and as it was a joint effort, we both have produced the same report.

In the questions, we are asked to implement the LDPC hardcode and softcode to try and decode the message and also perform Monte Carlo Simulation on the decoding algorithm and obtain the convergence graphs and compare the results with theoretical convergence graphs.

➢ For Hard Coding:

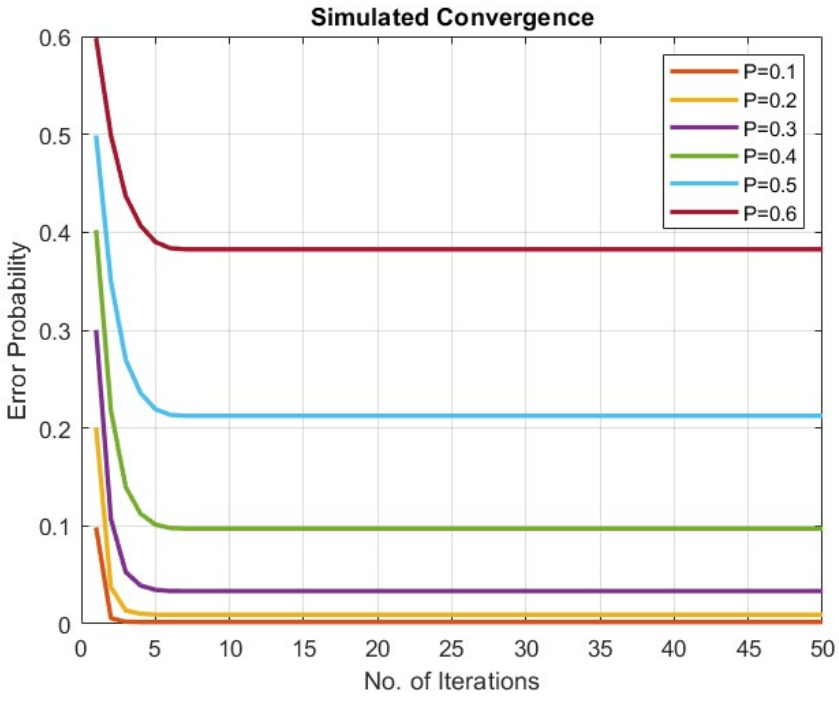
1) The H-matrix is of size 9 * 12.

Graph OF Monte Carlo Simulation(9*12):



Here the graph obtained is between the probability of error and the probability of successfully decoding it.

Graph OF Convergence(9*12):



:: Code ::

```cpp
#include <bits/stdc++.h>
using namespace std;


int find_dc(vector<vector<int>> &hmat, int n, int u)   // This function will find the degree of check node from given H matrix
{
    int dc = 0;

    for(int i=0; i<1; i++)
    {
        for(int j=0; j<n; j++)
        {
            if(hmat[i][j] == 1)
            {
                dc++;
            }
        }
    }

    return dc;
}

int find_dv(vector<vector<int>> &hmat, int n, int u)   // This function will find the degree of variable node from given H matrix
{
    int dv = 0;

    for(int i=0; i<1; i++)
    {
        for(int j=0; j<u; j++)
        {
            if(hmat[j][i] == 1)
            {
                dv++;
            }
        }
    }

    return dv;
}
```

```
void connect_checkNode_with_variableNode(vector<vector<int>> &hmat, int n, int
u, vector<vector<pair<int, int>>> &cn_graph)
{                                   // This function is for connection of
check node with variable node
    for (int i = 0; i < u; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (hmat[i][j] == 1)
            {
                cn_graph[i].push_back({j + 1, -1});
            }
        }
    }
}

void connect_variableNode_with_checkNode(vector<vector<int>> &hmat, int n, int
u, vector<vector<pair<int, int>>> &vn_graph)
{                                 // This function is for connection of variable
node with check node
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < u; j++)
        {
            if (hmat[j][i] == 1)
            {
                vn_graph[i].push_back({j + 1, -1});
            }
        }
    }
}

vector<float> assign_probability(float p)   // This function will assign
probability from 0 to 1 with increment of 0.01
{
    vector<float> prob_arr(101);

    for (int i = 0; i < 101; i++)
    {
        prob_arr[i] = p;

        p = p + 0.01;
    }

    return prob_arr;
}

int main()
```

```
{
    int n, u;

    cout << "Enter Row(u) and Column(n) of H matrix : " << endl;

    cin >> u >> n;

    vector<vector<int>> hmat(u, vector<int>(n, 0));

    vector<int> correct(101, 0);
    vector<int> val_vn(n);
    vector<vector<pair<int, int>>> cn_graph(u);
    vector<vector<pair<int, int>>> vn_graph(n);
    vector<float> probability_array(101);

    cout << "Enter H matrix : " << endl;
                                                    // Taking input of H
matrix
    for (int i = 0; i < u; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> hmat[i][j];
        }
    }


    int dc = find_dc(hmat, n, u);
    int dv = find_dv(hmat, n, u);

    connect_checkNode_with_variableNode(hmat,n,u,cn_graph);
    connect_variableNode_with_checkNode(hmat,n,u,vn_graph);

    probability_array = assign_probability(0);

    srand(time(NULL));

    for (int outer_ind = 0; outer_ind < 101; outer_ind++)
    {
        int Nsim = 10000;

        for (int Ksim = 1; Ksim <= Nsim; Ksim++) // Loop for Monte - Carlo
experiment
        {

            vector<int> original_signal(n, 0);     // Original message signal

            vector<int> signal_with_noise(n);     // Signal with noise
(Recieved signal)
```

```
            for (int i = 0; i < n; i++)
            {
                float tpr = ((float)rand() / (RAND_MAX + 1)); // Random number
generator function

                if (tpr >= probability_array[outer_ind])
                {
                    signal_with_noise[i] = original_signal[i];
                }
                else
                {
                    signal_with_noise[i] = -1;  // Set this as erasure bit
                }

                val_vn[i] = signal_with_noise[i];
            }

            int t_flag = 0;
            int zero_erasure_case = 1;
            int itr_cnt = 0;

            while (itr_cnt < 100)     // Maximum 100 iterations are allowed
            {                        // after 100 iteration loop will be
terminated automatically
                if(t_flag == 1)
                {
                    break;
                }

                if(zero_erasure_case == u)
                {
                    break;
                }

/*************************************  VN is sending message to
CN  ************************************/
/*********************************************  START  ********************
**********************************/
                for (int i = 0; i < n; i++)
                {
                    int k = val_vn[i];

                    int c1 = vn_graph[i][0].first;
                    int c2 = vn_graph[i][1].first;
                    int c3 = vn_graph[i][2].first;

                    for (int j = 0; j < dc; j++)
```

```
                    {
                        if (cn_graph[c1 - 1][j].first == i + 1)
                        {
                            cn_graph[c1 - 1][j].second = k;
                        }
                    }
                    for (int j = 0; j < dc; j++)
                    {
                        if (cn_graph[c2 - 1][j].first == i + 1)
                        {
                            cn_graph[c2 - 1][j].second = k;
                        }
                    }
                    for (int j = 0; j < dc; j++)
                    {
                        if (cn_graph[c3 - 1][j].first == i + 1)
                        {
                            cn_graph[c3 - 1][j].second = k;
                        }
                    }
                }

/*******************************************  END  ***********************
*******************************/


/************************************  CN is sending message to
VN  ***************************************/
/*******************************************  START  *********************
*********************************/


                t_flag = 1;
                zero_erasure_case = 0;

                for (int i = 0; i < u; i++)
                {
                    int erasure_count = 0;

                    for (auto it : cn_graph[i])
                    {
                        if (it.second == -1)
                        {
                            erasure_count++;
                        }
                    }

                    if (erasure_count == 1)
                    {
```

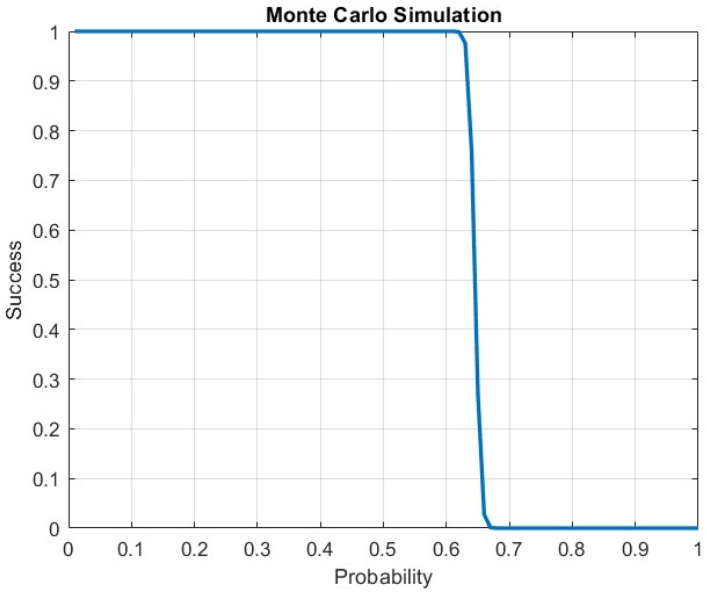
```
                        int cn_itr = 0;

                        for (auto &it : cn_graph[i])
                        {
                            if (it.second == -1)
                            {
                                int t_vn;

                                t_vn = it.first;

                                if (cn_itr == 0)
                                {
                                    val_vn[t_vn - 1] = (cn_graph[i][1].second
+ cn_graph[i][2].second + cn_graph[i][3].second) % 2;

                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second =
(cn_graph[i][1].second + cn_graph[i][2].second + cn_graph[i][3].second) % 2;
                                        }
                                    }
                                }
                                else if (cn_itr == 1)
                                {
                                    val_vn[t_vn - 1] = (cn_graph[i][0].second
+ cn_graph[i][2].second + cn_graph[i][3].second) % 2;

                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second =
(cn_graph[i][0].second + cn_graph[i][2].second + cn_graph[i][3].second) % 2;
                                        }
                                    }
                                }
                                else if (cn_itr == 2)
                                {
                                    val_vn[t_vn - 1] = (cn_graph[i][0].second
+ cn_graph[i][1].second + cn_graph[i][3].second) % 2;

                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
```

```
                                        vn_it.second =
(cn_graph[i][0].second + cn_graph[i][1].second + cn_graph[i][3].second) % 2;
                                    }
                                }
                            }
                            else if (cn_itr == 3)
                            {
                                val_vn[t_vn - 1] = (cn_graph[i][0].second
+ cn_graph[i][1].second + cn_graph[i][2].second) % 2;

                                for (auto &vn_it : vn_graph[t_vn - 1])
                                {
                                    if (vn_it.first == i + 1)
                                    {
                                        vn_it.second =
(cn_graph[i][0].second + cn_graph[i][1].second + cn_graph[i][2].second) % 2;
                                    }
                                }
                            }
                            t_flag = 0;
                        }
                        cn_itr++;
                    }
                }
                else if (erasure_count == 0)
                {
                    zero_erasure_case++;
                }

            }

/*****************************************  END  ***********************
*********************************/

            itr_cnt++;   // Increment the iteration number
        }

        t_flag = 0;

        for (int q = 0; q < n; q++)     // Checking the decoded signal
with original signal
        {
            if (val_vn[q] != original_signal[q])
            {
                t_flag++;
                break;
            }
        }
```

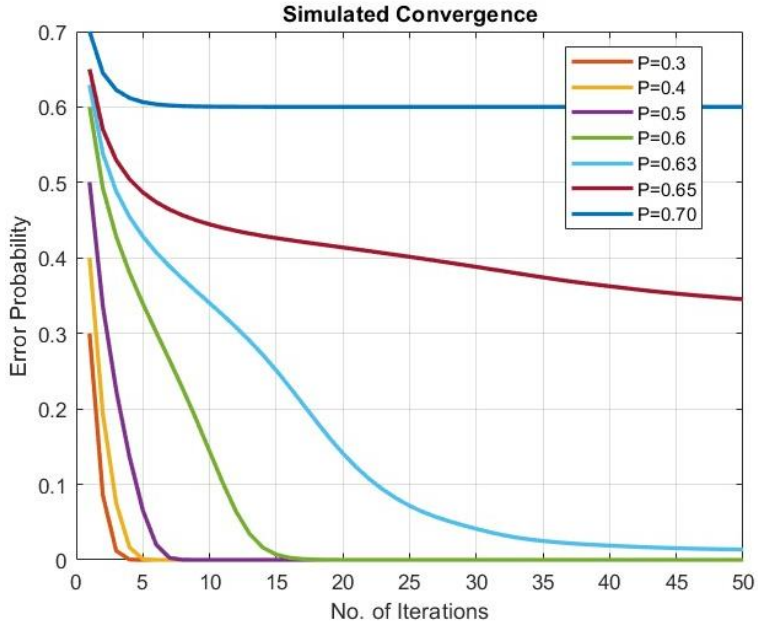
```
            if (t_flag == 0)
            {
                correct[outer_ind] = correct[outer_ind] + 1;
            }
        }
    }

    cout << "Probability of Successful Decoding : " << endl;

    cout << endl;

    for (int i = 0; i < 101; i++)
    {
        cout << float(correct[i]) / 10000 << endl;      // Printing the
probability of successful decoding
    }
}
```

2) The H-matrix is of size 3792 * 5056 .

Graph OF Monte Carlo Simulation(3792*5056):

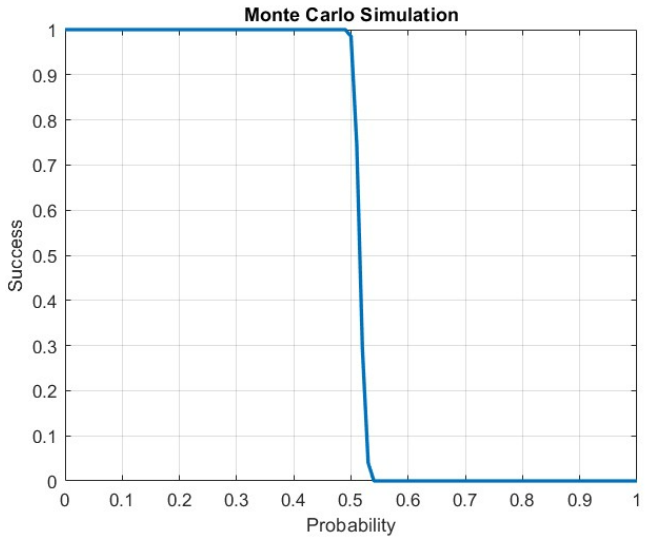


Graph OF Convergence(3792*5056):

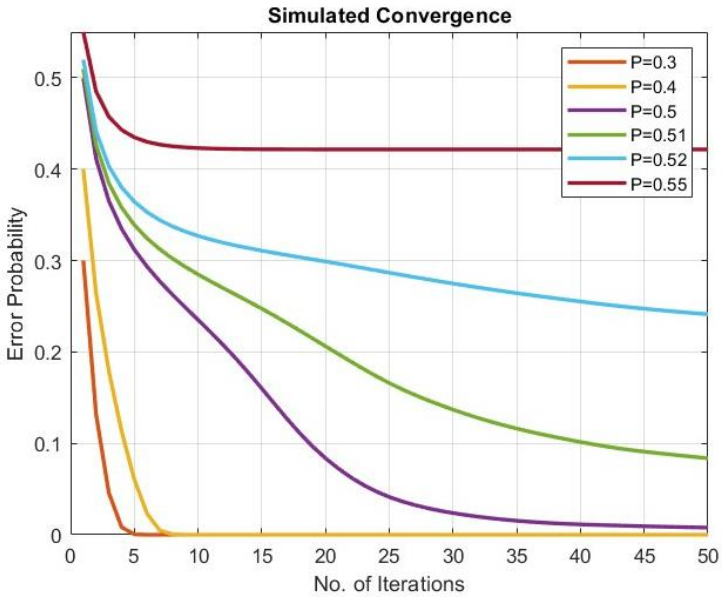


3) The H-matrix is of size 3000*5000.

Graph OF Monte Carlo Simulation(3000*5000):



Graph OF Convergence(3000*5000):



:: Code ::

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, u;

    u = 3000;
    n = 5000;

    vector<int> correct(101, 0);
    vector<int> val_vn(n);
    vector<vector<pair<int, int>>> cn_graph(u);
    vector<vector<pair<int, int>>> vn_graph(n);

    int **hmat = new int *[u]; // Read H matrix from .txt file

    for (int i = 0; i < u; i++)
    {
        hmat[i] = new int[n];
    }

    ifstream fin;
    fin.open("h_matrix_q3.txt");
```

```
if (!fin)
{
    cout << "Cannot open the file" << endl;
    exit(0);
}

int inRow = 0, inCol = 0;

char data;
while (!fin.eof())
{
    fin >> data;

    if (inCol == n)
    {

        inCol = 0;
        inRow++;
    }

    hmat[inRow][inCol] = data - 48;
    inCol++;

    if (inRow == u - 1 && inCol == n)
    {
        break;
    }
}

fin.close();

int dv = 0;
int dc = 0;

for (int i = 0; i < 1; i++)
{
    for (int j = 0; j < n; j++)
    {
        if (hmat[i][j] == 1)
        {
            dc++;
        }
    }
}

for (int i = 0; i < 1; i++)
{
```

```
        for (int j = 0; j < u; j++)
        {
            if (hmat[j][i] == 1)
            {
                dv++;
            }
        }
    }

    for (int i = 0; i < n; i++)     // Connection of VN to CN
    {
        for (int j = 0; j < u; j++)
        {
            if (hmat[j][i] == 1)
            {
                vn_graph[i].push_back({j + 1, -1});
            }
        }
    }

    for (int i = 0; i < u; i++)    // Connection of CN to VN
    {
        for (int j = 0; j < n; j++)
        {
            if (hmat[i][j] == 1)
            {
                cn_graph[i].push_back({j + 1, -1});
            }
        }
    }

    float pr[101];

    pr[0] = 0;

    float p = 0;

    for (int i = 0; i < 101; i++)
    {
        pr[i] = p;

        p = p + 0.01;
    }

    srand(time(NULL));

    for (int outer_ind = 0; outer_ind < 101; outer_ind++)
    {
```

```
        int Nsim = 1000;

        for (int Ksim = 1; Ksim <= Nsim; Ksim++)
        {

            vector<int> original_signal(n, 0); // Original Signal

            vector<int> signal_with_noise(n); // Signal with noise

            for (int i = 0; i < n; i++)
            {
                float tpr = ((float)rand() / (RAND_MAX + 1));

                if (tpr >= pr[outer_ind])
                {
                    signal_with_noise[i] = original_signal[i];
                }
                else
                {
                    signal_with_noise[i] = -1;
                }

                val_vn[i] = signal_with_noise[i];
            }

            int t_flag = 1;
            int zero_erasure_case = 1;
            int itr_cnt = 0;

            while (t_flag != 0 && zero_erasure_case != u && itr_cnt < 100)  //
Maximum 100 iterations are allowed
            {

                // Variable node to Check node

                for (int i = 0; i < n; i++)
                {
                    int k = val_vn[i];

                    int chk1 = vn_graph[i][0].first;
                    int chk2 = vn_graph[i][1].first;
                    int chk3 = vn_graph[i][2].first;

                    for (int j = 0; j < dc; j++)
                    {
                        if (cn_graph[chk1 - 1][j].first == i + 1)
                        {
                            cn_graph[chk1 - 1][j].second = k;
```

```
                }
            }
            for (int j = 0; j < dc; j++)
            {
                if (cn_graph[chk2 - 1][j].first == i + 1)
                {
                    cn_graph[chk2 - 1][j].second = k;
                }
            }
            for (int j = 0; j < dc; j++)
            {
                if (cn_graph[chk3 - 1][j].first == i + 1)
                {
                    cn_graph[chk3 - 1][j].second = k;
                }
            }
        }

        // Sending message to VN from CN
        // Using modulo two sum

        t_flag = 0;

        zero_erasure_case = 0;

        for (int i = 0; i < u; i++)
        {
            int erasure_count = 0;

            for (auto it : cn_graph[i])
            {
                if (it.second == -1)
                {
                    erasure_count++;
                }
            }

            if (erasure_count == 1)
            {
                int cn_itr = 0;

                for (auto &it : cn_graph[i])
                {
                    if (it.second == -1)
                    {
                        int t_vn;

                        t_vn = it.first;
```

```
                                if (cn_itr == 0)
                                {
                                    val_vn[t_vn - 1] = (cn_graph[i][1].second
+ cn_graph[i][2].second + cn_graph[i][3].second + cn_graph[i][4].second) % 2;

                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second =
(cn_graph[i][1].second + cn_graph[i][2].second + cn_graph[i][3].second +
cn_graph[i][4].second) % 2;
                                        }
                                    }
                                }
                                else if (cn_itr == 1)
                                {
                                    val_vn[t_vn - 1] = (cn_graph[i][0].second
+ cn_graph[i][2].second + cn_graph[i][3].second + cn_graph[i][4].second) % 2;

                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second =
(cn_graph[i][0].second + cn_graph[i][2].second + cn_graph[i][3].second +
cn_graph[i][4].second) % 2;
                                        }
                                    }
                                }
                                else if (cn_itr == 2)
                                {
                                    val_vn[t_vn - 1] = (cn_graph[i][0].second
+ cn_graph[i][1].second + cn_graph[i][3].second + cn_graph[i][4].second) % 2;

                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second =
(cn_graph[i][0].second + cn_graph[i][1].second + cn_graph[i][3].second +
cn_graph[i][4].second) % 2;
                                        }
                                    }
                                }
                                else if (cn_itr == 3)
                                {
```

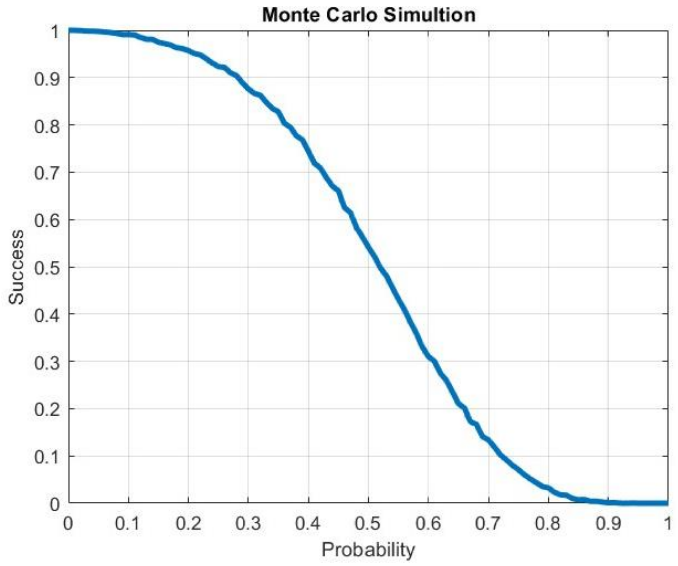
```
                                    val_vn[t_vn - 1] = (cn_graph[i][0].second
+ cn_graph[i][1].second + cn_graph[i][2].second + cn_graph[i][4].second) % 2;

                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second =
(cn_graph[i][0].second + cn_graph[i][1].second + cn_graph[i][2].second +
cn_graph[i][4].second) % 2;
                                        }
                                    }
                                }
                                else if (cn_itr == 4)
                                {
                                    val_vn[t_vn - 1] = (cn_graph[i][0].second
+ cn_graph[i][1].second + cn_graph[i][2].second + cn_graph[i][3].second) % 2;

                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second =
(cn_graph[i][0].second + cn_graph[i][1].second + cn_graph[i][2].second +
cn_graph[i][3].second) % 2;
                                        }
                                    }
                                }
                                t_flag = 1;
                            }

                            cn_itr++;
                        }
                    }
                    else if (erasure_count == 0)
                    {
                        zero_erasure_case++;
                    }
                }

                itr_cnt++;
            }

            t_flag = 0;

            for (int q = 0; q < n; q++)
            {
                if (val_vn[q] != original_signal[q])
```

```
                {
                    t_flag++;
                    break;
                }
            }

            if (t_flag == 0)
            {
                correct[outer_ind] = correct[outer_ind] + 1;
            }
        }
    }

    cout << "Probability of Successful Decoding : " << endl;

    cout << endl;

    for (int i = 0; i < 101; i++)
    {
        cout << float(correct[i]) / 1000 << endl;
    }
}
```

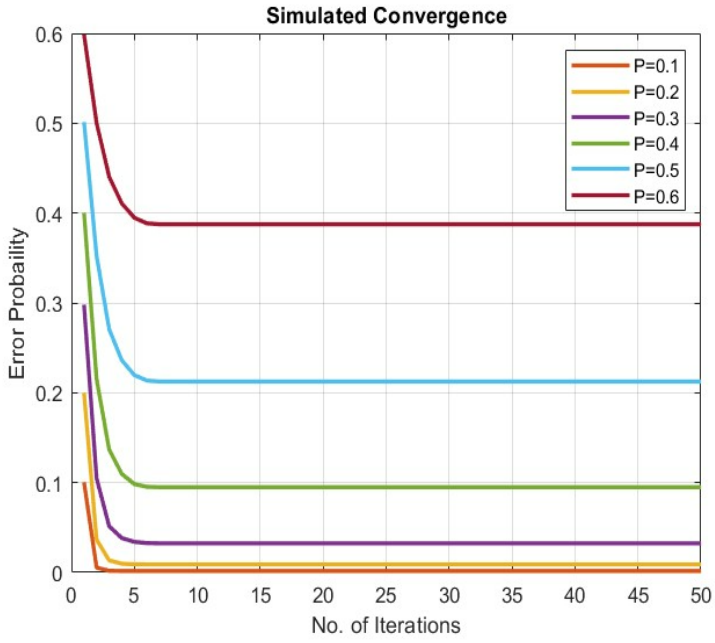
➢ For Soft Coding:

1) The H-matrix is of size 9 * 12.

Graph OF Monte Carlo Simulation(9*12):



Graph OF Convergence(9*12):



:: Code ::

```
#include <bits/stdc++.h>
using namespace std;

int find_dc(vector<vector<int>> &hmat, int n, int u)   // Find degree of check
node
{
    int dc = 0;

    for(int i=0; i<1; i++)
    {
        for(int j=0; j<n; j++)
        {
            if(hmat[i][j] == 1)
            {
                dc++;
            }
        }
    }

    return dc;
}

int find_dv(vector<vector<int>> &hmat, int n, int u)  // Find degree of
variable node
{
    int dv = 0;

    for(int i=0; i<1; i++)
    {
        for(int j=0; j<u; j++)
        {
            if(hmat[j][i] == 1)
            {
                dv++;
            }
        }
    }

    return dv;
}

void connect_checkNode_with_variableNode(vector<vector<int>> &hmat, int n, int
u, vector<vector<pair<int, float>>> &cn_graph)
{                                   // Establish connection of CN to VN
```

```
    for (int i = 0; i < u; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (hmat[i][j] == 1)
            {
                cn_graph[i].push_back({j + 1, -1});
            }
        }
    }
}

void connect_variableNode_with_checkNode(vector<vector<int>> &hmat, int n, int
u, vector<vector<pair<int, float>>> &vn_graph)
{                                       // Establish connection of VN to CN
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < u; j++)
        {
            if (hmat[j][i] == 1)
            {
                vn_graph[i].push_back({j + 1, -1});
            }
        }
    }
}

vector<float> assign_probability(float p)
{
    vector<float> prob_arr(101);

    for (int i = 0; i < 101; i++)
    {
        prob_arr[i] = p;

        p = p + 0.01;
    }

    return prob_arr;
}

int main()
{
    int n, u;

    cout << "Enter Row(u) and Column(n) of H matrix : " << endl;

    cin >> u >> n;
```

```
    vector<vector<int>> hmat(u, vector<int>(n, 0));
    vector<int> correct(101, 0);
    vector<vector<pair<int, float>>> cn_graph(u);
    vector<vector<pair<int, float>>> vn_graph(n);
    vector<float> final(n);
    vector<float> val_cn(3);
    vector<float> probability_array(101);

    float constant_1, vnd_1, vnd_0;

    cout << "Enter H matrix : " << endl;

    for (int i = 0; i < u; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> hmat[i][j];
        }
    }

    int dc = find_dc(hmat, n, u);
    int dv = find_dv(hmat, n, u);

    vector<float> val_vn(n);

    connect_checkNode_with_variableNode(hmat,n,u,cn_graph);

    connect_variableNode_with_checkNode(hmat,n,u,vn_graph);

    probability_array = assign_probability(0);

    srand(time(NULL));

    for (int outer_ind = 0; outer_ind < 101; outer_ind++)
    {

        int Nsim = 10000;

        for (int Nsim_itr = 1; Nsim_itr <= Nsim; Nsim_itr++) // Monte - Carlo
Experiment for 10000 times
        {

            vector<int> original_signal(n, 0); // Original Signal

            vector<int> signal_with_noise(n); // Signal with noise

            for (int i = 0; i < n; i++)
```

```
            {
                float tpr = ((float)rand() / (RAND_MAX + 1));

                if (tpr > probability_array[outer_ind])
                {
                    signal_with_noise[i] = original_signal[i];
                }
                else
                {
                    signal_with_noise[i] = -1;
                }

                if (signal_with_noise[i] == 0)
                {
                    val_vn[i] = 0;

                    for (int ci = 0; ci < 3; ci++)
                    {
                        vn_graph[i][ci].second = 0;
                    }
                }
                else
                {
                    val_vn[i] = 0.5;    // Assigning probability 0.5 for
erasure bit

                    for (int ci = 0; ci < 3; ci++)
                    {
                        vn_graph[i][ci].second = 0.5;
                    }
                }
            }

            int t_flag = 1;
            int zero_erasure_case = 1;
            int itr_cnt = 0;

            while (itr_cnt < 100)     // 100 iterations are allowed
            {
                if(t_flag == 0)
                {
                    break;
                }

                if(zero_erasure_case == u)
                {
                    break;
                }
```

```
                // Sending message to CN from VN

                for (int i = 0; i < n; i++)
                {
                    int c1 = vn_graph[i][0].first;
                    int c2 = vn_graph[i][1].first;
                    int c3 = vn_graph[i][2].first;

                    for (int j = 0; j < dc; j++)
                    {
                        if (cn_graph[c1 - 1][j].first == i + 1)
                        {
                            cn_graph[c1 - 1][j].second =
vn_graph[i][0].second;
                        }
                    }
                    for (int j = 0; j < dc; j++)
                    {
                        if (cn_graph[c2 - 1][j].first == i + 1)
                        {
                            cn_graph[c2 - 1][j].second =
vn_graph[i][1].second;
                        }
                    }
                    for (int j = 0; j < dc; j++)
                    {
                        if (cn_graph[c3 - 1][j].first == i + 1)
                        {
                            cn_graph[c3 - 1][j].second =
vn_graph[i][2].second;
                        }
                    }
                }

                // Sending message to VN from CN
                // Using bernard's equation

                t_flag = 0;
                zero_erasure_case = 0;

                for (int i = 0; i < u; i++)
                {
                    int erasure_count = 0;

                    for (auto it : cn_graph[i])
                    {
                        if (it.second == 0.5)
```

```
                        {
                            erasure_count++;
                        }
                    }

                    if (erasure_count == 1)
                    {
                        int cn_itr = 0;

                        for (auto &it : cn_graph[i])
                        {
                            if (it.second == 0.5)
                            {
                                int t_vn;

                                t_vn = it.first;

                                if (cn_itr == 0)
                                {
                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second = (0.5 - 0.5 * (1 - 2
* cn_graph[i][1].second) * (1 - 2 * cn_graph[i][2].second) * (1 - 2 *
cn_graph[i][3].second));
                                        }
                                    }
                                }
                                else if (cn_itr == 1)
                                {
                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second = (0.5 - 0.5 * (1 - 2
* cn_graph[i][0].second) * (1 - 2 * cn_graph[i][2].second) * (1 - 2 *
cn_graph[i][3].second));
                                        }
                                    }
                                }
                                else if (cn_itr == 2)
                                {
                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
```

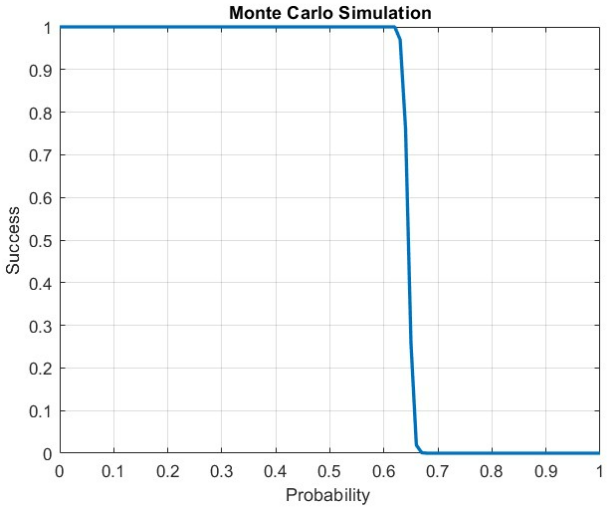
```
                                        vn_it.second = (0.5 - 0.5 * (1 - 2
* cn_graph[i][0].second) * (1 - 2 * cn_graph[i][1].second) * (1 - 2 *
cn_graph[i][3].second));
                                    }
                                }
                            }
                            else if (cn_itr == 3)
                            {
                                for (auto &vn_it : vn_graph[t_vn - 1])
                                {
                                    if (vn_it.first == i + 1)
                                    {
                                        vn_it.second = (0.5 - 0.5 * (1 - 2
* cn_graph[i][0].second) * (1 - 2 * cn_graph[i][1].second) * (1 - 2 *
cn_graph[i][2].second));
                                    }
                                }
                            }
                            t_flag = 1;
                        }

                        cn_itr++;
                    }
                }
                else if (erasure_count == 0)
                {
                    zero_erasure_case++;
                }
            }

            for (int i = 0; i < n; i++)      // This loop is compution
Conditional probability for VN would be 1
            {
                for (int j = 0; j < 3; j++)
                {
                    val_cn[j] = vn_graph[i][j].second;
                }

                for (int j = 0; j < 3; j++)
                {
                    vnd_1 = val_vn[i] * val_cn[(j + 1) % 3] * val_cn[(j +
2) % 3];

                    vnd_0 = (1 - val_vn[i]) * (1 - val_cn[(j + 1) % 3]) *
(1 - val_cn[(j + 2) % 3]);

                    constant_1 = 1 / (vnd_0 + vnd_1);
```

```
                        vn_graph[i][j].second = constant_1 * vnd_1;
                    }
                }

                itr_cnt++;
            }

            for (int i = 0; i < n; i++)      // Making decision from
likelihood ratio
            {
                final[i] = (val_vn[i] / (1 - val_vn[i])) *
(vn_graph[i][0].second / (1 - vn_graph[i][0].second)) * (vn_graph[i][1].second
/ (1 - vn_graph[i][1].second)) * (vn_graph[i][2].second / (1 -
vn_graph[i][2].second));

                if (final[i] > 1)
                {
                    final[i] = 1;
                }
                else if (final[i] == 1)
                {
                    final[i] = 0.5;
                }
                else
                {
                    final[i] = 0;
                }
            }

            t_flag = 0;

            for (int i = 0; i < n; i++)         //  Checking with original
signal
            {
                if (final[i] != original_signal[i])
                {
                    t_flag++;
                    break;
                }
            }

            if (t_flag == 0)
            {
                correct[outer_ind] = correct[outer_ind] + 1;
            }
        }
    }
```

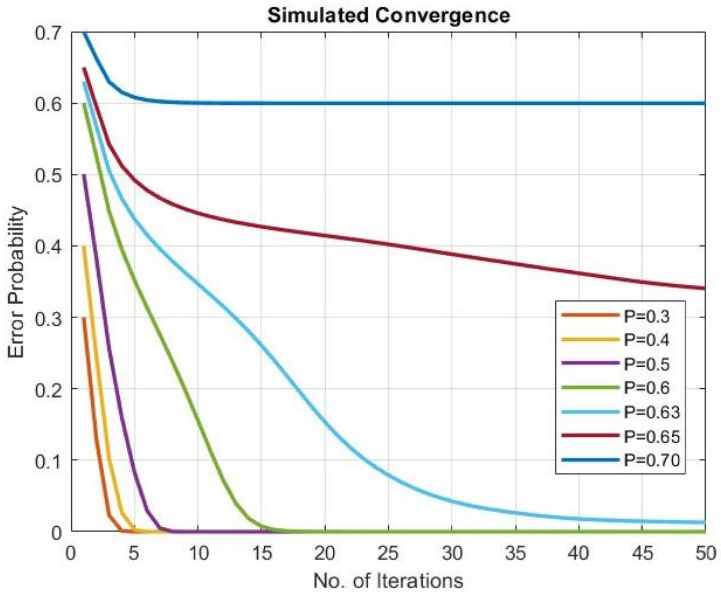
```
    cout << "Probability of Successful Decoding : " << endl;

    cout << endl;

    for (int i = 0; i < 101; i++)
    {
        cout << float(correct[i]) / 10000 << endl;
    }
}
```

2) The H-matrix is of size 3792 * 5056 .

Graph OF Monte Carlo Simulation(3792*5056):

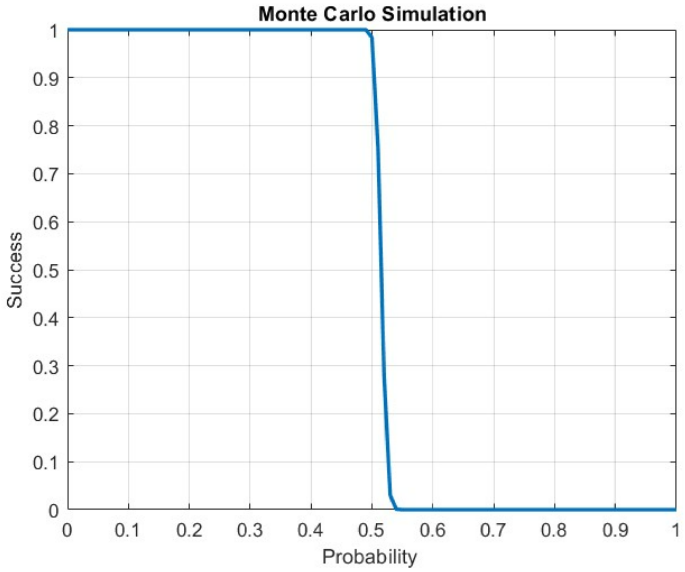


Graph OF Convergence(3792*5056):

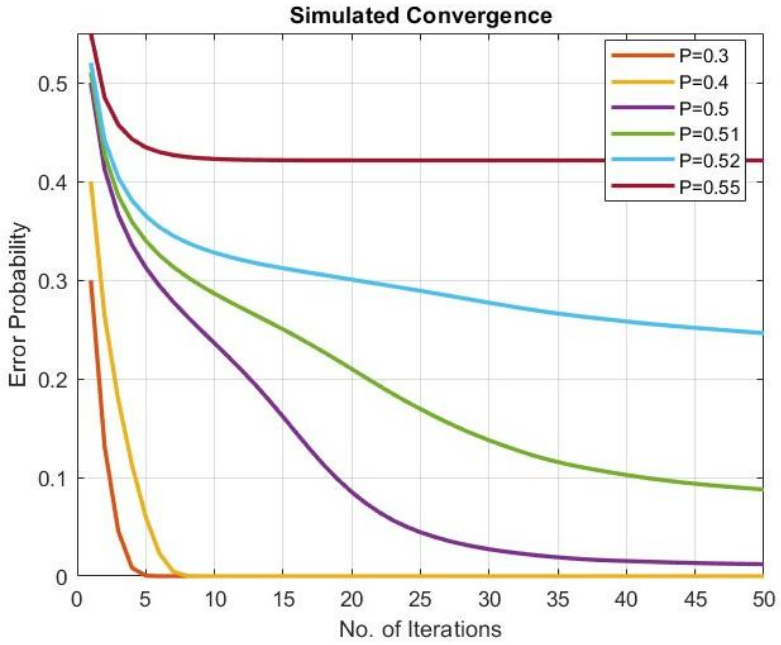


3) The H-matrix is of size 3000*5000.

Graph OF Monte Carlo Simulation(3000*5000):



Graph OF Convergence(3000*5000):



:: Code ::

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, u;

    u = 3000;
    n = 5000;

    vector<int> correct(101, 0);
    vector<float> val_cn(3);
    vector<float> val_vn(n);
    vector<vector<pair<int, float>>> cn_graph(u);
    vector<vector<pair<int, float>>> vn_graph(n);
    vector<float> final(n);

    float constant_1, vnd_1, vnd_0;

    int **hmat = new int *[u];

    for (int i = 0; i < u; i++)
    {
```

```
        hmat[i] = new int[n];
    }

    ifstream fin;
    fin.open("h_matrix_q3.txt");

    if (!fin)
    {
        cout << "Cannot open the file" << endl;
        exit(0);
    }

    int inRow = 0, inCol = 0;

    char data;
    while (!fin.eof())  // Reading h_matrix_q3 as a text file

    {
        fin >> data;

        if (inCol == n)
        {

            inCol = 0;
            inRow++;
        }

        hmat[inRow][inCol] = data - 48;
        inCol++;

        if (inRow == u - 1 && inCol == n)
        {
            break;
        }
    }
    fin.close();

    for (int i = 0; i < n; i++)
    {                                                    // Establish
connection of VN to CN
        for (int j = 0; j < u; j++)
        {
            if (hmat[j][i] == 1)
            {
                vn_graph[i].push_back({j + 1, -1});
            }
        }
    }
```

```
    for (int i = 0; i < u; i++)
    {                                                   // Establish
connection of CN to VNs
        for (int j = 0; j < n; j++)
        {
            if (hmat[i][j] == 1)
            {
                cn_graph[i].push_back({j + 1, -1});
            }
        }
    }

    int dv = 0;
    int dc = 0;

    for(int i=0; i<1; i++)       // Find degree of check node
    {
        for(int j=0; j<n; j++)
        {
            if(hmat[i][j] == 1)
            {
                dc++;
            }
        }
    }

    for(int i=0; i<1; i++)         // Find degree of variable node
    {
        for(int j=0; j<u; j++)
        {
            if(hmat[j][i] == 1)
            {
                dv++;
            }
        }
    }

    float probability_array[101];

    float p = 0;

    for (int i = 0; i < 101; i++)
    {
        probability_array[i] = p;
        p = p + 0.01;
    }
```

```
    srand(time(NULL));

    for (int outer_ind = 0; outer_ind < 101; outer_ind++)
    {
        int Nsim = 1000;

        for (int Nsim_itr = 1; Nsim_itr <= Nsim; Nsim_itr++) // Monte - Carlo
Experiment for 1000 times
        {
            vector<int> original_signal(n, 0); // Original Signal
            vector<int> signal_with_noise(n); // Signal with noise

            for (int i = 0; i < n; i++)
            {
                float tpr = ((float)rand() / (RAND_MAX + 1)); // Random number
generator

                if (tpr > probability_array[outer_ind])
                {
                    signal_with_noise[i] = original_signal[i];
                }
                else
                {
                    signal_with_noise[i] = -1;
                }

                if (signal_with_noise[i] == 0)
                {
                    val_vn[i] = 0;

                    for (int ci = 0; ci < dv; ci++)
                    {
                        vn_graph[i][ci].second = 0;
                    }
                }
                else
                {
                    val_vn[i] = 0.5;       // Assigning 0.5 as a probability
for erasure bits

                    for (int ci = 0; ci < dv; ci++)
                    {
                        vn_graph[i][ci].second = 0.5;
                    }
                }
            }

            int t_flag = 1;
```

```
            int zero_erasure_case = 1;
            int itr_cnt = 0;

            while (t_flag != 0 && zero_erasure_case != 9 && itr_cnt <
100)    // Maximum 100 iterations are allowed
            {

                // Sending message to CN from VN

                for (int i = 0; i < n; i++)
                {
                    int c1 = vn_graph[i][0].first;
                    int c2 = vn_graph[i][1].first;
                    int c3 = vn_graph[i][2].first;

                    for (int j = 0; j < dc; j++)
                    {
                        if (cn_graph[c1 - 1][j].first == i + 1)
                        {
                            cn_graph[c1 - 1][j].second =
vn_graph[i][0].second;
                        }
                    }
                    for (int j = 0; j < dc; j++)
                    {
                        if (cn_graph[c2 - 1][j].first == i + 1)
                        {
                            cn_graph[c2 - 1][j].second =
vn_graph[i][1].second;
                        }
                    }
                    for (int j = 0; j < dc; j++)
                    {
                        if (cn_graph[c3 - 1][j].first == i + 1)
                        {
                            cn_graph[c3 - 1][j].second =
vn_graph[i][2].second;
                        }
                    }
                }

                // Sending message to VN from CN
                // Using bernard's equation

                t_flag = 0;
                zero_erasure_case = 0;

                for (int i = 0; i < u; i++)
```

```
                {
                    int erasure_count = 0;

                    for (auto it : cn_graph[i])
                    {
                        if (it.second == 0.5)
                        {
                            erasure_count++;
                        }
                    }

                    if (erasure_count == 1)
                    {
                        int cn_itr = 0;

                        for (auto &it : cn_graph[i])
                        {
                            if (it.second == 0.5)
                            {
                                int t_vn;

                                t_vn = it.first;

                                if (cn_itr == 0)
                                {
                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second = (0.5 - 0.5 * (1 - 2
* cn_graph[i][1].second) * (1 - 2 * cn_graph[i][2].second) * (1 - 2 *
cn_graph[i][3].second) * (1 - 2 * cn_graph[i][4].second));
                                        }
                                    }
                                }
                                else if (cn_itr == 1)
                                {
                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second = (0.5 - 0.5 * (1 - 2
* cn_graph[i][0].second) * (1 - 2 * cn_graph[i][2].second) * (1 - 2 *
cn_graph[i][3].second) * (1 - 2 * cn_graph[i][4].second));
                                        }
                                    }
                                }
                                else if (cn_itr == 2)
```

```
                                {
                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second = (0.5 - 0.5 * (1 - 2
* cn_graph[i][0].second) * (1 - 2 * cn_graph[i][1].second) * (1 - 2 *
cn_graph[i][3].second) * (1 - 2 * cn_graph[i][4].second));
                                        }
                                    }
                                }
                                else if (cn_itr == 3)
                                {
                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second = (0.5 - 0.5 * (1 - 2
* cn_graph[i][0].second) * (1 - 2 * cn_graph[i][1].second) * (1 - 2 *
cn_graph[i][2].second) * (1 - 2 * cn_graph[i][4].second));
                                        }
                                    }
                                }
                                else if (cn_itr == 4)
                                {
                                    for (auto &vn_it : vn_graph[t_vn - 1])
                                    {
                                        if (vn_it.first == i + 1)
                                        {
                                            vn_it.second = (0.5 - 0.5 * (1 - 2
* cn_graph[i][0].second) * (1 - 2 * cn_graph[i][1].second) * (1 - 2 *
cn_graph[i][2].second) * (1 - 2 * cn_graph[i][3].second));
                                        }
                                    }
                                }
                                t_flag = 1;
                            }

                            cn_itr++;
                        }
                    }
                    else if (erasure_count == 0)
                    {
                        zero_erasure_case++;
                    }
                }
```

```
                for (int i = 0; i < n; i++)    // This loop is for compution
of Conditional probability for VN would be 1
                {
                    for (int j = 0; j < 3; j++)
                    {
                        val_cn[j] = vn_graph[i][j].second;
                    }

                    for (int j = 0; j < 3; j++)
                    {
                        vnd_1 = val_vn[i] * val_cn[(j + 1) % 3] * val_cn[(j +
2) % 3];

                        vnd_0 = (1 - val_vn[i]) * (1 - val_cn[(j + 1) % 3]) *
(1 - val_cn[(j + 2) % 3]);

                        constant_1 = 1 / (vnd_0 + vnd_1);

                        vn_graph[i][j].second = constant_1 * vnd_1;
                    }
                }

                itr_cnt++;
            }

            t_flag = 0;

            for (int i = 0; i < n; i++)      // Making decision from
likelihood ratio
            {
                final[i] = (val_vn[i] / (1 - val_vn[i])) *
(vn_graph[i][0].second / (1 - vn_graph[i][0].second)) * (vn_graph[i][1].second
/ (1 - vn_graph[i][1].second)) * (vn_graph[i][2].second / (1 -
vn_graph[i][2].second));

                if (final[i] > 1)
                {
                    final[i] = 1;
                }
                else if (final[i] == 1)
                {
                    final[i] = 0.5;
                }
                else
                {
                    final[i] = 0;
                }
            }
```

```
            for (int i = 0; i < n; i++)      //  Checking with original signal
            {
                if (final[i] != original_signal[i])
                {
                    t_flag++;
                    break;
                }
            }

            if (t_flag == 0)
            {
                correct[outer_ind] = correct[outer_ind] + 1;
            }
        }
    }

    cout << "Probability of Successful Decoding : " << endl;

    cout << endl;

    for (int i = 0; i < 101; i++)
    {
        cout << float(correct[i]) / 1000 << endl;
    }
}
```

❖ Observations:

- From the graphs we can observe that the graphs of convergence tend to converge at zero as the number of iterations becomes a large number.

❖ Conclusion:

- We learned how to write an LDPC soft and hard code to decode any given message and also compared the obtained results with that obtained by theoretical analysis.