Name: Pranay Pourkar

Roll No. : BT15EEE061

Programming Assignment

Index

1) Write a program to decide diagonal dominance (or) otherwise of given square matrix. Also find magnitude of diagonal dominance (if so)

```
//*************************************************************
#include<iostream>
#include<math.h>
#define N 4
using namespace std;
int main()
{
   int n = 4;
   int m[N][N] = { { 8, 0, -2, -3 },
                { -2, -10, 2, 2},
                { 1, -6, 12, -3},
                { -3, 2, 3, -15} };

   bool status = true;
   int diff_array[n] , delta;

    // for each row
   for (int i = 0; i < n; i++)
  {
      // for each column, finding sum of each row.
      int sum = 0;
      for (int j = 0; j < n; j++)
                        {
                          sum = sum + abs(m[i][j]);
      }

      // removing the diagonal element.
      sum = sum - abs(m[i][i]);

      // checking if diagonal element is less
      // than sum of non-diagonal element.
      if (abs(m[i][i]) < sum)
                        {
                          status = false;
                        }
                        else
                        {
                                  diff_array[i] =  abs(m[i][i]) - sum;
                                  //cout<<diff_array[i]<<"\n";
                        }

   }

   //check for diagonal dominant or not
   if(status == true)
   {
     //let delta=diff_array[0]
     delta = diff_array[0];
     for(int j=1;j<n;j++)
            {
                        if(delta>diff_array[j])
                        {
```

```
                    delta = diff_array[j];
                }
            }

            cout<<"Given Square matrix is diagonal dominant"<<"\n"<<"Magnitude of diagonal dominance is delta =
"<<delta;

        }
        else
        {
          cout<<"Given Square matrix is NOT diagonal dominant";
        }

   return 0;
}
//*************************************************************************
```

Input:

```
 8 0 -2 -3
-2 -10 2 2
 1 -6 12 -3
-3 2 3 -15
```

Output: Given Square matrix is diagonal dominant
Magnitude of diagonal dominance is delta = 2

2) Write a program for finding an estimate of $cond_{oo}(A)$ for a diagonally dominant matrix

```
//*************************************************************
#include<iostream>
#include<math.h>
#define N 4
using namespace std;
int main()
{
   int m[N][N] = { { 8, 0, -2, -3 },
                   { -2, -10, 2, 2},
                   { 1, -6, 12, -3},
                   { -3, 2, 3, -15} };

   bool status = true;
          int diff_array[N] , delta;
          int row_sum[N], max_row_sum = 0;

    // for each row
   for (int i = 0; i < N; i++)
  {

      // for each column, finding sum of each row.
      int sum = 0;
      for (int j = 0; j < N; j++)
                    {
                      sum = sum + abs(m[i][j]);
                      row_sum[i] = sum;
                      if(max_row_sum<row_sum[i])
                      {
         max_row_sum = row_sum[i];
                      }
      }

      // removing the diagonal element.
      sum = sum - abs(m[i][i]);
      // checking if diagonal element is less
      // than sum of non-diagonal element.
      if (abs(m[i][i]) < sum)
                    {
                      status = false;
                    }
                    else
                    {
                              diff_array[i] =  abs(m[i][i]) - sum;
                              //cout<<diff_array[i]<<"\n";
                    }

   }

   //check for diagonal dominant or not
   if(status == true)
   {
     //let delta=diff_array[0]
```

```
    delta = diff_array[0];
    for(int j=1;j<N;j++)
          {
                if(delta>diff_array[j])
                {
                      delta = diff_array[j];
                }
          }
          cout<<"Given Square matrix is diagonal dominant"<<"\n"<<"Magnitude of diagonal dominance is delta =
"<<delta<<"\n";
          //To find infinity norm of matrix
          float condition_no;
          condition_no = (float)max_row_sum/(float)delta;
          cout<<"Condition number of matrix is <= "<<condition_no;

         }
         else
         {
          cout<<"Given Square matrix is NOT diagonal dominant";
         }

         return 0;
}
//*************************************************************************
```

Input:

```
 8 0 -2 -3
-2 -10 2 2
 1 -6 12 -3
-3 2 3 -15
```

Output:

```
Given Square matrix is diagonal dominant
Magnitude of diagonal dominance is delta = 2
Condition number of matrix is <= 11.5
```

3) Write a program which gives solution of diagonal matrix, lower triangular, upper triangular, unitary matrix

//Part 1: A is diagonal matrix

```
//*************************************************************
```

```
#include<iostream>
#include<math.h>
#define N 3
using namespace std;

int main()
{
	float A[N][N] = { {2,0,0},
	  { 0,3,0},
	  { 0,0,4} };

	float b[N] = {2,9,8};
	float x[N];
	int i,j,k;
	for(i=0;i<N;i++)
	{
		x[i] = b[i]/A[i][i];
	}

	cout<<"Solution of Ax=b is x = ";
	for(j=0;j<N;j++)
	{
		cout<<x[j]<<" ";
	}

	return 0;
}
```

```
//*************************************************************
```

Input:

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{bmatrix} \qquad b = [2\ 9\ 8]$$

Output: Solution of Ax=b is x = [1 3 2]

//Part 2: A is lower triangular

```
//*************************************************************
```

```
#include<iostream>
#include<math.h>
#define N 3

using namespace std;
```

```
int main()
{
	float A[N][N] = { {10,0,0 },
	  { 2,-10.6,0},
	  { 3, 1.1, (-1163/106)} };

	float b[N] = {15,37,-10};
	float x[N], sum;

	int k,j;
	for(int i=0;i<N;i++)
	{
	 sum=0;
	 for(j=0;j<i;j++)
	 {
		sum = sum + (A[i][j]*x[j]);
	 }
	 x[i] = (b[i] - sum)/A[i][i];
	}

	cout<<"Solution of Ax=b is x = ";
	for(j=0;j<N;j++)
	{
		cout<<x[j]<<" ";
	}

	return 0;
}
//*************************************************************************
```

Input:

$$A = \begin{bmatrix} 10 & 0 & 0 \\ 2 & -10.6 & 0 \\ 3 & 1.1 & (-1163/106) \end{bmatrix} \qquad b = [15\ 37\ -10]$$

Output: Solution of Ax=b is x = [1.5 -3.20755 1.09717]

//Part 3: A is upper triangular

```
//*************************************************************************

#include<iostream>
#include<math.h>
#define N 3

using namespace std;

int main()
{
	float A[N][N] = { {1,0.3,0.4 },
	  { 0,1,(-11/53)},
	  { 0,0,1} };

	float b[N] = {1.5,(-170/53),1};
	float x[N], sum;
```

```
        int k,j;
        for(int i=N-1;i>=0;i--)
        {
          sum=0;
          for(j=i+1;j<N;j++)
          {
                  sum = sum + (A[i][j]*x[j]);
          }
          x[i] = (b[i] - sum)/A[i][i];
        }

        cout<<"Solution of Ax=b is x = ";
        for(j=0;j<N;j++)
        {
                cout<<x[j]<<" ";
        }

        return 0;
}
//*************************************************************
```

Input:

A = 1 0.3 0.4
0 1 (-11/53)
0 0 1

b = [1.5 (-170/53) 1]

Output: Solution of Ax=b is x = [2 -3 1]

//Part 4: A is unitary matrix

```
//*************************************************************

#include<iostream>
#include<math.h>
#define N 4

using namespace std;

int main()
{
        int A[N][N] = { {5, -2, 2, 7},
            {1, 0, 0, 3},
            {-3, 1, 5, 0},
            {3, -1, -9, 4}};

        int x[N];
        int b[N] = {1,2,3,4};

   //A* = inverse(A), A* = conj(transpose(A))
        //transpose of A
        int i,j,k;
        for(i=0;i<N;i++)
        {
                for(j=i+1;j<N;j++)
                {
                        k = A[i][j];
```

```
                        A[i][j] = A[j][i];
                        A[j][i] = k;
                }
        }

        int sum;

        for(i=0;i<N;i++)
        {
                sum = 0;
                for(j=0;j<N;j++)
                {
                        sum = sum + (A[i][j] * b[j]);
                }
                x[i] = sum;
        }

        cout<<"Solution of Ax=b is x = ";
        for(j=0;j<N;j++)
         {
                cout<<x[j]<<" ";
         }
        return 0;
}
//*************************************************************************
```

Input:

A

$$\begin{bmatrix} 5 & -2 & 2 & 7 \\ 1 & 0 & 0 & 3 \\ -3 & 1 & 5 & 0 \\ 3 & -1 & -9 & 4 \end{bmatrix}$$

b = [1 2 3 4]

Output: Solution of Ax=b is x = [10 -3 -19 29]

4) Write a program to find solution of Ax=b using gauss elimination

//**********

```
#include<iostream>
#include<math.h>

#define N 3 //order of matrix A
using namespace std;

int main()
{
        int i,j,k,p,q,r;
  float c,x[N],sum=0;

  //Augmented matrix A
  float A[N+1][N+1] = { {0.143,0.357,2.01,-5.17 },
          { -1.31,0.911,1.99,-5.46},
          { 11.2,-4.3,0.605,4.42} };

  printf("\n Elements are inserted in augmented matrix form:\n\n");

  for(j=0; j<N; j++) /* loop for the generation of upper triangular matrix*/
  {

    for(i=0; i<N; i++)
    {
      if(i>j)
      {
        c=A[i][j]/A[j][j];
        for(k=0; k<N+1; k++)
        {
          A[i][k]=A[i][k]-c*A[j][k];
        }
      }
    }
  }
  x[N-1]=A[N-1][N]/A[N-1][N-1];
  /* this loop is for backward substitution*/

  for(i=0;i<N;i++)
  {
      for(j=0;j<N+1;j++)
      {
          cout<<A[i][j]<<" ";
          }
          cout<<"\n";
      }
```

```
        cout<<"............"<<"\n";

   for(int i=N-1;i>=0;i--)
        {
          sum=0;
          for(j=i+1;j<N;j++)
          {
                sum = sum + (A[i][j]*x[j]);
          }
          x[i] = (A[i][N] - sum)/A[i][i];
        }

        cout<<"Solution of Ax=b is x = ";
        for(j=0;j<N;j++)
        {
                cout<<x[j]<<" ";
        }

        return 0;
}
//*************************************************************
```

Input: Elements are inserted in augmented matrix form

$$A|b = \begin{bmatrix} 0.143 & 0.357 & 2.01 & -5.17 \\ -1.31 & 0.911 & 1.99 & -5.46 \\ 11.2 & -4.3 & 0.605 & 4.42 \end{bmatrix}$$

Output: Solution of Ax=b is x = [-10.3127 -27.4614 3.03902]

5) Write a program to find solution of Ax=b using gauss elimination with trivial pivoting

//**********

```
#include<iostream>
#include<math.h>

#define N 4 //order of matrix A
using namespace std;

int main()
{
        int i,j,k,p,q,r;
  float c,x[N],sum=0;

  //Augmented matrix A
  float A[N][N+1] = { {1,-1,2,-1,-8 },
          { 2,-2,3,-3,-20},
          { 1,1,1,0,-2},
          {1,-1,4,3,4} };

        float b[N] = {-8,-20,-2,4};

  printf("\n Elements are inserted in augmented matrix form:\n\n");

  for(j=0; j<N; j++) /* loop for the generation of upper triangular matrix*/
  {

        //check that A[j][j] is non zero otherwise interchange
        if(A[j][j]==0)
        {
                for(k=j+1;k<N;k++)
                {
                        if(A[k][j]!=0)
                        {
                                p=k;
                                break;
                                }
                        }

                        for(k=0;k<N+1;k++)
                        {
                                q=A[j][k];
                                A[j][k] = A[p][k];
                                A[p][k] = q;
                        }

                }


    for(i=0; i<N; i++)
    {
      if(i>j)
      {
        c=A[i][j]/A[j][j];
```

```
            for(k=0; k<N+1; k++)
            {
               A[i][k]=A[i][k]-c*A[j][k];
            }
          }
       }
    }
    x[N-1]=A[N-1][N]/A[N-1][N-1];
    /* this loop is for backward substitution*/

    for(i=0;i<N;i++)
    {
            for(j=0;j<N+1;j++)
            {
                    cout<<A[i][j]<<" ";
                    }
                    cout<<"\n";
            }
            cout<<"............"<<"\n";




    for(int i=N-1;i>=0;i--)
            {
             sum=0;
             for(j=i+1;j<N;j++)
             {
                    sum = sum + (A[i][j]*x[j]);
             }
             x[i] = (A[i][N] - sum)/A[i][i];
            }

            cout<<"Solution of Ax=b is x = ";
            for(j=0;j<N;j++)
            {
                    cout<<x[j]<<" ";
            }

            return 0;
}
//*************************************************************************
```

Input: Elements are inserted in augmented matrix form

$$A|b = \begin{bmatrix} 1 & -1 & 2 & -1 & -8 \\ 2 & -2 & 3 & -3 & -20 \\ 1 & 1 & 1 & 0 & -2 \\ 1 & -1 & 4 & 3 & 4 \end{bmatrix}$$

Output: Solution of Ax=b is x = [-7 3 2 2]

6) Write a program to find solution of Ax=b using gauss elimination with partial pivoting

```
//**************************************************************************

#include<iostream>
#include<math.h>

#define N 3 //order of matrix A
using namespace std;

int main()
{
        int i,j,k,p,q,r;
  float c,x[N],sum=0,max;

  //Augmented matrix A
  float A[N+1][N+1] = { {10,-7,0,7},
          { -3,2.099,6,3.901},
          { 5,-1,5,6} };

  printf("\n Elements are inserted in augmented matrix form:\n\n");

  for(j=0; j<N; j++) /* loop for the generation of upper triangular matrix*/
  {

      //check that A[j][j] is max among row otherwise interchange with max
        max = abs(A[j][j]);
        p=j;
            for(k=j+1;k<N;k++)
            {
                if(abs(A[k][j])>max)
                {
                    max = A[k][j];
                    p=k;
                    }
                }

                for(k=0;k<N+1;k++)
                {
                    q=A[j][k];
                    A[j][k] = A[p][k];
                    A[p][k] = q;
                }



    for(i=0; i<N; i++)
    {
      if(i>j)
      {
        c=A[i][j]/A[j][j];
        for(k=0; k<N+1; k++)
```

```
            {
               A[i][k]=A[i][k]-c*A[j][k];
            }
          }
       }
       for(p=0;p<N;p++)
      {
            for(q=0;q<N+1;q++)
            {
                        cout<<A[p][q]<<" ";
                        }
                        cout<<"\n";
              }
            cout<<"............"<<"\n";

   }
   x[N-1]=A[N-1][N]/A[N-1][N-1];
   /* this loop is for backward substitution*/

   for(i=0;i<N;i++)
   {
            for(j=0;j<N+1;j++)
            {
                        cout<<A[i][j]<<" ";
                        }
                        cout<<"\n";
            }
            cout<<"............"<<"\n";

   for(int i=N-1;i>=0;i--)
            {
             sum=0;
             for(j=i+1;j<N;j++)
             {
                        sum = sum + (A[i][j]*x[j]);
             }
             x[i] = (A[i][N] - sum)/A[i][i];
            }

            cout<<"Solution of Ax=b is x = ";
            for(j=0;j<N;j++)
            {
                        cout<<x[j]<<" ";
            }
            return 0;
}
//*************************************************************************
```

Input: Elements are inserted in augmented matrix form

$$A|b = \begin{bmatrix} 10 & -7 & 0 & 7 \\ -3 & 2.099 & 6 & 3.901 \\ 5 & -1 & 5 & 6 \end{bmatrix}$$

Output: Solution of Ax=b is x = [0 -1 1]

7) Write a program to solve a linear system Ax=b using Gauss Jordan elimination

//**********

```
#include<iostream>
#include<math.h>
#define N 3
using namespace std;

int main()
{
        int i,j,k,p,q,r;
   float c,x[N],sum=0;

   //Augmented matrix A
   float A[N][N+1] = { {0,2,1,4 },
             { 1,1,2,6},
             { 2,1,1,7} };

   printf("\n Elements are inserted in augmented matrix form:\n\n");

        for(j=0; j<N; j++) /* loop for the generation of upper triangular matrix*/
   {

        //check that A[j][j] is non zero otherwise interchange
        if(A[j][j]==0)
        {
                for(k=j+1;k<N;k++)
                {
                        if(A[k][j]!=0)
                        {
                                p=k;
                                break;
                                }
                        }

                        for(k=0;k<N+1;k++)
                        {
                                q=A[j][k];
                                A[j][k] = A[p][k];
                                A[p][k] = q;
                        }

                }


      for(i=0; i<N; i++)
      {
         if(i != j)
         {
            c=A[i][j]/A[j][j];
            for(k=0; k<N+1; k++)
```

```
            {
               A[i][k]=A[i][k]-c*A[j][k];
            }
          }
       }
    }
    x[N-1]=A[N-1][N]/A[N-1][N-1];
    /* this loop is for backward substitution*/

    for(i=0;i<N;i++)
    {
          for(j=0;j<N+1;j++)
          {
                cout<<A[i][j]<<" ";
                }
                cout<<"\n";
          }
          cout<<"............"<<"\n";

    for(int i=N-1;i>=0;i--)
          {
           sum=0;
           for(j=i+1;j<N;j++)
           {
                sum = sum + (A[i][j]*x[j]);
           }
           x[i] = (A[i][N] - sum)/A[i][i];
          }

          cout<<"Solution of Ax=b is x = ";
          for(j=0;j<N;j++)
          {
                cout<<x[j]<<" ";
          }
          return 0;
}
//*************************************************************************
```

Input: Elements are inserted in augmented matrix form

$$A|b = \begin{bmatrix} 0 & 2 & 1 & 4 \\ 1 & 1 & 2 & 6 \\ 2 & 1 & 1 & 7 \end{bmatrix}$$

Output: Solution of Ax=b is x = [2.2 1.4 1.2]

8) Write a program to find inverse of non-singular matrix using Gauss Jordan elimination

```
//*************************************************************

#include<iostream>
#include<math.h>
#define N 3
using namespace std;

int main()
{
        int i,j,k,p;
        float q,r;
  float c,sum=0;

  //Augmented matrix A|I
  float A[N][N+N] = { {5,7,9,1,0,0 },
          { 4,3,8,0,1,0},
          { 7,5,6,0,0,1} };

        float A_inverse[N][N];

  printf("\n Elements are inserted in augmented matrix form, A|I \n\n");

        for(j=0; j<N; j++) /* loop for the generation of upper triangular matrix*/
  {

        //check that A[j][j] is non zero otherwise interchange
        if(A[j][j]==0)
        {
                for(k=j+1;k<N;k++)
                {
                        if(A[k][j]!=0)
                        {
                                p=k;
                                break;
                                }
                        }

                        for(k=0;k<N+N;k++)
                        {
                                q=A[j][k];
                                A[j][k] = A[p][k];
                                A[p][k] = q;
                        }

                }


    for(i=0; i<N; i++)
    {
       if(i != j)
       {
          c=A[i][j]/A[j][j];
          for(k=0; k<N+N; k++)
```

```
            {
               A[i][k]=A[i][k]-c*A[j][k];
            }
          }
       }
    }

   //reduce diagonal elements to 1
   for(i=0;i<N;i++)
   {
          q = A[i][i];
          //cout<<"p"<<p<<"P";
          for(j=0;j<N+N;j++)
          {
                    A[i][j] = A[i][j]/q;

                    if(j>=N)
                    {
                              A_inverse[i][j-N] = A[i][j];
                              }
                    }
          }

   cout<<"Inverse of matrix is"<<"\n";
   for(i=0;i<N;i++)
   {
          for(j=0;j<N;j++)
          {
                    cout<<A_inverse[i][j]<<" ";
                    }
                    cout<<"\n";
          }
          cout<<"....."<<"\n";

          return 0;
}
//*************************************************************************
```

Input: Elements are inserted in augmented matrix form

A|I =

```
5 7 9 1 0 0
4 3 8 0 1 0
7 5 6 0 0 1
```

Output: Inverse of matrix A is

```
-0.209524 0.0285714 0.27619
0.304762 -0.314286 -0.0380952
-0.00952383 0.228571 -0.12381
```

9) Write a program to decompose non-singular matrix A as product LU of lower & upper triangular matrices

i) with assumption Lii = 1 (Doolittle method)

ii) with assumption Uii =1 (Crouts method)

// Doolittle method

//**********

```
#include<iostream>
#include <bits/stdc++.h>
#include<math.h>
#define N 3

//lii = 1 doolittle method
using namespace std;

int main()
{
  int l[N][N], u[N][N];
  memset(l, 0, sizeof(l));
  memset(u, 0, sizeof(u));

  int a[N][N] = { {2,-1,-2},
                  {-4,6,3},
                  {-4,-2,8}};


  for (int i = 0; i < N; i++) {

      // Upper Triangular
      for (int k = i; k < N; k++) {

          // Summation of L(i, j) * U(j, k)
          int sum = 0;
          for (int j = 0; j < i; j++)
                                  {
              sum += (l[i][j] * u[j][k]);
                                  }
          // Evaluating U(i, k)
          u[i][k] = a[i][k] - sum;
      }

      // Lower Triangular
      for (int k = i; k < N; k++) {
          if (i == k)
              l[i][i] = 1; // Diagonal as 1
          else {

              // Summation of L(k, j) * U(j, i)
              int sum = 0;
              for (int j = 0; j < i; j++)
                  sum += (l[k][j] * u[j][i]);
```

```
            // Evaluating L(k, i)
            l[k][i] = (a[k][i] - sum) / u[i][i];
          }
        }
     }

  cout<<"A = "<<"\n";
  for(int i=0;i<N;i++)
  {
          for(int j=0;j<N;j++)
          {
                  cout<<a[i][j]<<" ";
          }
          cout<<"\n";
  }

    cout<<"\nL = "<<"\n";

  for(int i=0;i<N;i++)
  {
          for(int j=0;j<N;j++)
          {
                  cout<<l[i][j]<<" ";
          }
          cout<<"\n";
  }

  cout<<"\nU = "<<"\n";

  for(int i=0;i<N;i++)
  {
          for(int j=0;j<N;j++)
          {
                  cout<<u[i][j]<<" ";
          }
          cout<<"\n";
  }
  return 0;
}
//*************************************************************************
```

Input:

$$A = \begin{bmatrix} 2 & -1 & -2 \\ -4 & 6 & 3 \\ -4 & -2 & 8 \end{bmatrix}$$

Output:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -2 & -1 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 2 & -1 & -2 \\ 0 & 4 & -1 \\ 0 & 0 & 3 \end{bmatrix}$$

// Crouts method

```
//*************************************************************************

#include<iostream>
#include <bits/stdc++.h>
#include<math.h>
#define N 3

//uii = 1 crouts method
using namespace std;

int main()
{
  float l[N][N], u[N][N];
  memset(l, 0, sizeof(l));
  memset(u, 0, sizeof(u));

  float a[N][N] = { {10,3,4},
            {2,-10,3},
            {3,2,-10}};


  for (int i = 0; i < N; i++) {

     // Lower Triangular
     for (int k = i; k < N; k++) {

          // Summation of L(k, j) * U(j, i)
          float sum = 0;
          for (int j = 0; j < i; j++)
                                  {
            sum += (l[k][j] * u[j][i]);
                                   }

          // Evaluating L(k, i)
          l[k][i] = a[k][i] - sum;


       // Upper Triangular
     for (int k = i; k < N; k++) {

       if (i == k)
                          {
         u[i][i] = 1; // Diagonal as 1
                }
       else {
       // Summation of L(i, j) * U(j, k)
       int sum = 0;
       for (int j = 0; j < i; j++)
                          {
          sum += (l[i][j] * u[j][k]);
                           }
       // Evaluating U(i, k)
       u[i][k] = (a[i][k] - sum)/l[i][i];

      }
```

```
    }

    }
  }

  cout<<"A = "<<"\n";
  for(int i=0;i<N;i++)
  {
        for(int j=0;j<N;j++)
        {
              cout<<a[i][j]<<" ";
        }
        cout<<"\n";
  }

    cout<<"\nL = "<<"\n";

  for(int i=0;i<N;i++)
  {
        for(int j=0;j<N;j++)
        {
              cout<<l[i][j]<<" ";
        }
        cout<<"\n";
  }

  cout<<"\nU = "<<"\n";

  for(int i=0;i<N;i++)
  {
        for(int j=0;j<N;j++)
        {
              cout<<u[i][j]<<" ";
        }
        cout<<"\n";
  }
        return 0;
}
//*************************************************************************
```

Input:

$$A = \begin{bmatrix} 10 & 3 & 4 \\ 2 & -10 & 3 \\ 3 & 2 & -10 \end{bmatrix}$$

Output:

$$L = \begin{bmatrix} 10 & 0 & 0 \\ 2 & -10.6 & 0 \\ 3 & 1.1 & -10.8887 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 0.3 & 0.4 \\ 0 & 1 & -0.283019 \\ 0 & 0 & 1 \end{bmatrix}$$

10) Find Approximation of $||A||_2$ when A is lower triangular matrix

```
//*************************************************************
#include<iostream>
#include<math.h>
#define N 4
using namespace std;
int main()
{
	int i,j,k,sum;
	//A is lower triangular matrix
	float A[N][N] = {{2,0,0,0},{4,-2,0,0},{-5,6,1,0},{1,5,3,3}};
	float x[N];
	float y[N];

	//step 1
	x[0] = 1;
	y[0] = A[0][0]*x[0]
	//step 2
	for(i=1;i<N;i++)
	{
		sum=0;
		for(j=0;j<i;j++)
		{
			sum = sum + A[i][j]*x[j];
		}
		if(abs(A[i][i] + sum) > abs(A[i][i] - sum))
		{
			x[i] = 1;
		}
		else
		{
			x[i] = -1;
		}
		y[i] = (A[i][i]*x[i]) + sum;
	}
	 //step 3
	 float y_2_norm=0;
	 for(i=0;i<N;i++)
	 {
		y_2_norm = y_2_norm + (y[i]*y[i]);
	 }
	 y_2_norm = sqrt(y_2_norm);
	 cout<<"2 norm of vector y is found out to be "<<y_2_norm<<"\n";
	 cout<<"Approximation of 2 norm of matrix A estimates to "<<y_2_norm/sqrt(N)<<"\n";
	return 0;
}
//*************************************************************
```

Input:

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & -2 & 0 & 0 \\ -5 & 6 & 1 & 0 \\ 1 & 5 & 3 & 3 \end{bmatrix}$$

Output: Approximation of 2 norm of matrix A estimates to 8.42615

11) Find Approximation of $||A^{-1}||_2$ when A is lower triangular matrix

//**********

```
#include<iostream>
#include<math.h>
#define N 4
using namespace std;

int main()
{
	int i,j,k,sum;
	//A is lower triangular matrix
   float A[N][N] = {{2,0,0,0},{4,-2,0,0},{-5,6,1,0},{1,5,3,3}};
	float x[N];
	float y[N];
	int sign;

	//step 1
	x[0] = 1;
	y[0] = 1/(A[0][0]);

	//step 2
	for(i=1;i<N;i++)
	{
		sum=0;
		for(j=0;j<i;j++)
		{
			sum = sum + A[i][j]*y[j];
		}

		if(sum>0)
		{
			sign = 1;
		}
		else
		{
			sign = -1;
		}


		y[i] = (-1/A[i][i])*(sum + sign);
	}

	 //step 3
	 float y_2_norm=0;
	 for(i=0;i<N;i++)
	 {
		y_2_norm = y_2_norm + (y[i]*y[i]);
	 }

	 y_2_norm = sqrt(y_2_norm);

	 cout<<"2 norm of vector y is found out to be "<<y_2_norm<<"\n";
```

```
        cout<<"Approximation of 2 norm of matrix A inverse estimates to "<<y_2_norm/sqrt(N)<<"\n";


        return 0;
}
//*************************************************************************
```

Input:

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & -2 & 0 & 0 \\ -5 & 6 & 1 & 0 \\ 1 & 5 & 3 & 3 \end{bmatrix}$$

Output: Approximation of 2 norm of matrix A inverse estimates to 5.06211

12) For lower matrix A, find approximation of $cond_2(A)$

```
//*************************************************************************

#include<iostream>
#include<math.h>
#define N 4

using namespace std;

float approximation_of_2_norm(float A[N][N])
{
   int i,j,k,sum;
        float x[N];
        float y[N];

        //step 1
        x[0] = 1;
        y[0] = A[0][0]*x[0];

        //step 2
        for(i=1;i<N;i++)
        {
                sum=0;
                for(j=0;j<i;j++)
                {
                        sum = sum + A[i][j]*x[j];
                }

                if(abs(A[i][i] + sum) > abs(A[i][i] - sum))
                {
```

```
                        x[i] = 1;
                }
                else
                {
                        x[i] = -1;
                }
                y[i] = (A[i][i]*x[i]) + sum;
        }

        //step 3
        float y_2_norm=0;
        for(i=0;i<N;i++)
        {
                y_2_norm = y_2_norm + (y[i]*y[i]);
        }

        y_2_norm = sqrt(y_2_norm);

        //cout<<"2 norm of vector y is found out to be "<<y_2_norm<<"\n";

        return y_2_norm/sqrt(N);
}

float approximation_of_2_norm_inverse(float A[N][N])
{
        int i,j,k,sum;
        //A is lower triangular matrix
        float x[N];
        float y[N];
        int sign;

        //step 1
        x[0] = 1;
        y[0] = 1/(A[0][0]);

        //step 2
        for(i=1;i<N;i++)
        {
                sum=0;
                for(j=0;j<i;j++)
                {
                        sum = sum + A[i][j]*y[j];
                }

                if(sum>0)
                {
                        sign = 1;
                }
                else
                {
                        sign = -1;
                }


                y[i] = (-1/A[i][i])*(sum + sign);
        }
```

```
	//step 3
	float y_2_norm=0;
	for(i=0;i<N;i++)
	{
		y_2_norm = y_2_norm + (y[i]*y[i]);
	}

	y_2_norm = sqrt(y_2_norm);

	//cout<<"2 norm of vector y is found out to be "<<y_2_norm<<"\n";

	return y_2_norm/sqrt(N);
}



int main()
{
		float A[N][N] = {{2,0,0,0},{4,-2,0,0},{-5,6,1,0},{1,5,3,3}};
    float norm_A, norm_A_inverse;
    norm_A = approximation_of_2_norm(A);
    norm_A_inverse = approximation_of_2_norm_inverse(A);

    cout<<"Approximation of condition number A is : "<<norm_A*norm_A_inverse;




	return 0;
}
//**************************************************************************
```

Input:

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & -2 & 0 & 0 \\ -5 & 6 & 1 & 0 \\ 1 & 5 & 3 & 3 \end{bmatrix}$$

Output: Approximation of condition number A is : 42.6541