Numerical Analysis
Project – Phase 1

Name	ID
1-Sara Yasser Mohamed Sobhy	18010774
2-Mennatullah Mahmoud Saad Mohamed	19016713
3- Rowan Nasser Mostafa Edrees	19015686
4-Nourhan Ahmed Arafa Mohamed	19016812

a- Pseudo code:

Gauss Elimination:

```
class GaussienElimination:
  function backSubstitution(array y):
      scaling(y)
      for r=0 to n
         for c=0 to n
             A[r][c]= y[r][c]
             b[r]= y[r][c+1]
      print(A,b,n)
      for i=n-1 to 0
        for j=i+1 to n-1
           sum = Round( (b[i]- sum) / A[i][j]*x[j])
        x[i]= Round( (b[i]- sum)/A[i][i] )
      return x
  function singular(A):
      for i=0 to n-1
         for j=0 to n-1
            if( A[i][j] == 0){ add 1 to check
              if(check == n){ print singular
                              end}
         check =0
```

```
function print(A,b,n):void
        for r=0 to n-1
           print {
          for c=0 to n-1
             print  A[r][c]
             if(c not equal n-1) print ,
          print }
          print { b[i] }

     function Round(valie, digit): double
       scale = 10^ digit
      return  round(value * scale) / scale

     function scaling(y):double[][]
      for i=0 to n-1
          for j=0 to n
            if(y[i][j] not equal 0
               y[i][j]= Round(y[i]/temp)
      return y
     function Gauss([][]A, []b): double[][]
       singular()
       for j=0 to n-1
          max = j
```

```
    for i=j+1 to n-1
       if(A[i][j]> A[max][j]
       max = i

    if(max not equal j)
      temp = A[j]  A[j] = A[max] A[max] = temp
        t  = b[j] b[j] = b[max] b[max] = t
      Print(A,b,n)
     singular(A)

     for i=j+1 to n-1
       if( A[i][j] not equal 0)
          alpha = Round(A[i][j]/A[j][i]
       b[i]= Round(b[i]- alpha*A[j][m])
       for m=j to n-1
          A[i][m] = Round(A[i][m] - alpha * A[j][m]
       print(A,b,n)
scaling(y)
return y
```

Gauss-Jordan:

```
class GaussJordanElimination extends GaussianElimination :
   function jordan(A, b): void
      // call function Gauss from GaussElimination class
```

```
y =Gauss(A,b)
 for r=0 to r=n-1
   for c=0 to c=n-1
      A[r][c] = y[r][c];
      b[r]= y[r][c+1]
 for j=n-1 to j=1
    for i = 0 to i = n-1
       if(A[i][j] not equal 0 and i not equal j)
            alpha = Round( A[i][j] / A[j][j] )
            b[i] =  Round( (b[i] - alpha * b[j])
            A[i][j] =  Round( (A[i][j] - alpha * A[j][j]);
            Print(A,b,n);
singular(A)
print b
```

LU Decomposition:

Doolittle:

```
class lu extends GaussianElimination{
    function Doolittle ([][] A,[] b) : void
   singular(A);
       for j = 0 to j =n-1
```

```
int max = j;
for  i = j + 1 to i = n-1
    if (abs(A[i][j]) > abs(A[max][j]))
        max = i;

if(max not equal j)
    double[] temp = A[j] A[j] = A[max] A[max] = temp
    double   t    = b[j] b[j] = b[max] b[max] = t
    Print(A,b,n)

for i = 0 to i= n-1
    for  k = i to k = n-1
        sum = 0;
        for  j = 0 to j = i-1
            sum = Round( (sum + Round((lower[i][j] *
upper[j][k]),digit)), digit)
        upper[i][k] = Round((A[i][k] - sum), digit);
      singular(upper)

      for  k = i to k = n-1
        if (i equal k)
           lower[i][i] = 1
```

```
          else
            sum = 0;
            for j = 0 to j = i-1
              sum =  Round( (sum + Round((lower[k][j] * upper[j][i]), digit)), digit); }
            lower[k][i] =  Round( (Round(A[k][i] - sum, digit) / upper[i][i]), digit);
          singular(lower)
     printSingle(lower)
     printSingle(upper)

     for i = 0 to i = n-1
       sum = 0.0
      for j = 0 to j = i-1
       sum = sum + lower[i][j] * d[j]
      d[i] = Round( (Round(b[i] - sum, digit) / lower[i][i]),digit)

     for ( i = 0; i < n; i++)
       print(d[i])
```

```
x = backSubstitution(y)
for i = 0 to i = n-1
    print(x[i])
```

crout:

```
class croutLU extends lu:
    function crout ([][] A,[] b) : void
        singular(A);
        for i = 0 to i =n-1
            int max = j;
         for  j = 0 to j = n-1
              if (abs(A[i][j]) > abs(A[max][j]))
                  max = i;
          for i = 0 to i= n-1
               upper[i][i]=1
          for j = 0 to j = n-1
                  sum = 0;
                 for  i = j to j = n-1
                    sum = Round( (sum + Round((lower[i][k] *
upper[k][i]),digit)), digit)
              lower[i][k] = Round((A[i][k] - sum), digit)
```

```
                for  i = j to i = n-1
                   sum = 0;
                   for k= 0 to k = j-1
                      sum =  Round( (sum + Round((lower[j][k] *
upper[k][i]), digit)), digit); }
                  upper[j][i] =  Round( (Round(A[j][i] - sum, digit) /
lower[j][i]), digit);
                 singular(lower)


        printSingle(lower)
        printSingle(upper)


        for i = 0 to i = n-1
           sum = 0.0
          for j = 0 to j = i-1
            sum = sum + lower[i][j] * d[j]
          d[i] = Round( (Round(b[i] - sum, digit) /
lower[i][i]),digit)


        for ( i = 0; i < n; i++)
           print(d[i])
```

x = backSubstitution(y)

for i = 0 to i = n-1

print(x[i])

Gauss-Seidil:

public class GaussSeidil {

define double 2d array M as a Coefficient array

define double array vector as a variables vector

define double array xi as a Initial Guess vector

define int iterationsNum as a one of the stooping conditions

define double epsilon which is Absolute relative error as a one of the stooping conditions

define String array solutionsArray of size 2

intialize int precision with value 5

public GaussSeidil(double[][] M, double[] vector, double[] xi, int iterationsNum, double epsilon, int precision) {

//use this constructor to take the values of this element from the user

}

public GaussSeidil(double[][] M, double[] vector, double[] xi, int iterationsNum, double epsilon) //this constructor used when the user don’t enter persision value

public double Round(double value, int digit) {

initialize double scale with value[Math.pow(10, digit)]

the return value is(Math.round(value * scale) / scale)

```
}

// method to check whether matrix is diagonally dominant or not
public boolean converges() {

        for i = 0 to i < M.length increase with 1 every iteration
{
          Initialize double diagonal with absolute value of M[i][i]
          Intialze double sumRowElements with value 0
          for j = 0 to j < M.length increase with 1 every iteration
                  {
                    if (i not equal j)
                         sumRowElements increase by absolute value of
M[i][j]);
                   }
                    if (sumRowElements bigger than or equal diagonal)
                                  return false;
                }
                return true;
        }
        public String[] solve() {
                initialize String steps with empty value
                initialize String stat with value ("X = { ")
```

```
                if (the equations not converges) {
                        steps += "The solution could not converge"
                } else {

                        Initialize int iterations with value 0;
                        define int n with value of the size of main matrix
                        define double array X with size of main matrix
                        define double array P with size of main matrix
                        X = xi;
                        while (true) {
                        for i = 0 to i < n ,increase with 1
                        {
                                Initialize double sum = vector[i]
steps += "x(new)" + i + " = (" + vector[i]
for j = 0 to i < n ,increase with 1
                                {
                                  if (j not equal i)
 {
                                        sum -= M[i][j] * X[j]
                                        steps += " - " + M[i][j] + " * " + X[j]
                                  }
                                }
```

```
                    X[i] = Round(1 / M[i][i] * sum, precision);
                    steps += ") / " + (M[i][i]) + " = " + X[i] + " & "
               }
               iterations++;
               initialize boolean stop with value true;
               for i = 0 to i < n and stop ,increase with 1
               {
                if (absolute value of(X[i] - P[i]) bigger than  epsilon)
                              stop = false
                if (stop OR iterations == iterationsNum)
                         break the loop
                P = (double[]) X.clone();
               }
               for i = 0 to i < n,increase with 1
{
                    stat += X[i] + " ";
               }
               stat += "}";
               steps += ", \n , run time = ";
          }
          solutionsArray[0] = stat;
          solutionsArray[1] = steps;
          return solutionsArray; }  }
```

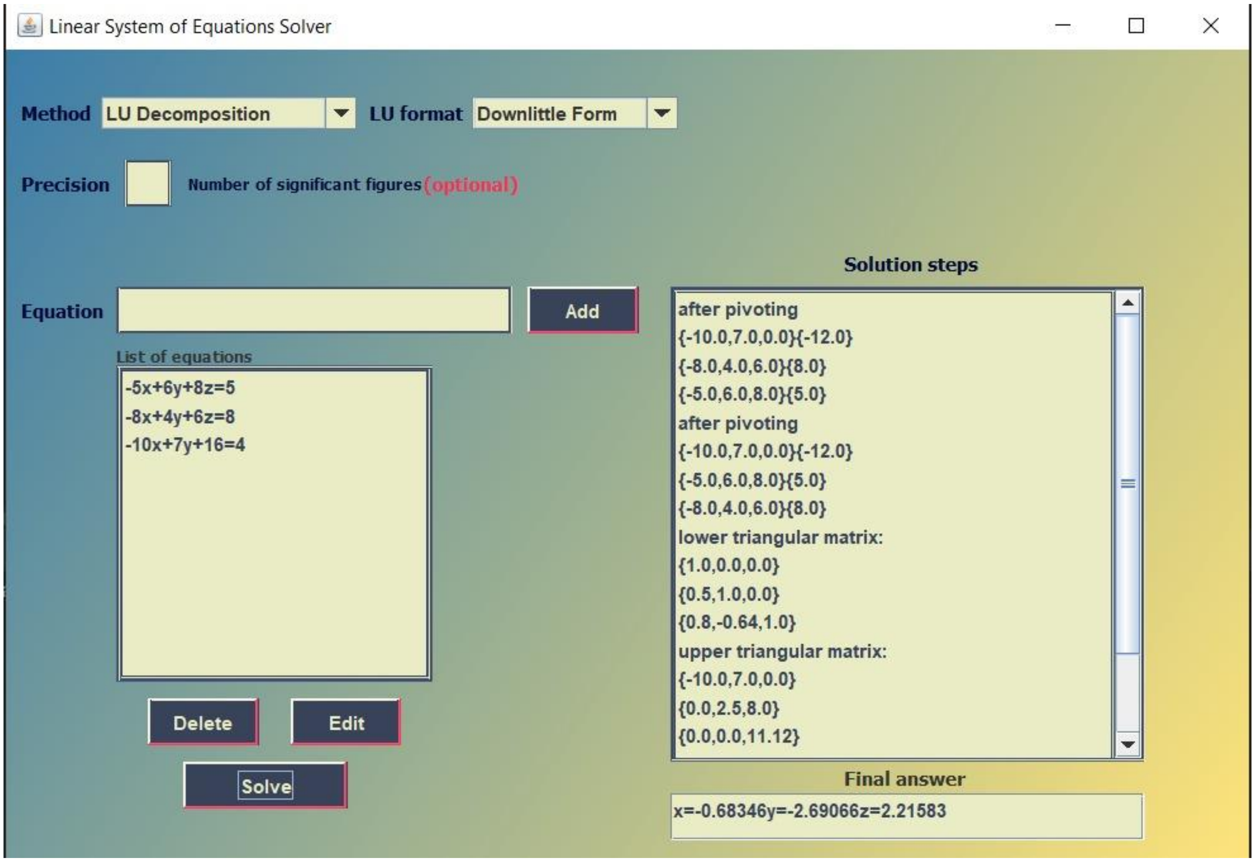
Jacobi-Iteration:

```
Class JacobiIteration inherits from Class GaussSeidil all
attributes and
public class JacobiIteration extends GaussSeidil {

     public String[] solve() {
          initialize String steps with empty value
          initialize String stat with value ("X = { ")

          if (the equations not converges) {
               steps += "The solution could not converge"
          } else {

               Initialize int iterations with value 0;
               define int n with value of the size of main
matrix
               define double array X with size of main matrix
               and fill it with zeros
               define double array P with size of main matrix
               P = xi
               while (true) {
               for i = 0 to i < n ,increase with 1
               {
                    Initialize double sum = vector[i]
                    steps += "x(new)" + i + " = (" + vector[i]
                    for j = 0 to i < n ,increase with 1
                    {
                      if (j not equal i)
                      {
                         sum -= M[i][j] * P[j]
                         steps += " - " + M[i][j] + " * " + P[j]
                      }
                    }

                    X[i] = Round(1 / M[i][i] * sum, precision);
                    steps += ") / " + (M[i][i]) + " = " + X[i] +
" & "
               }
               iterations++;
               initialize boolean stop with value true;
               for i = 0 to i < n and stop ,increase with 1
               {
                 if (absolute value of(X[i] - P[i]) bigger than
epsilon)
                             stop = false
```

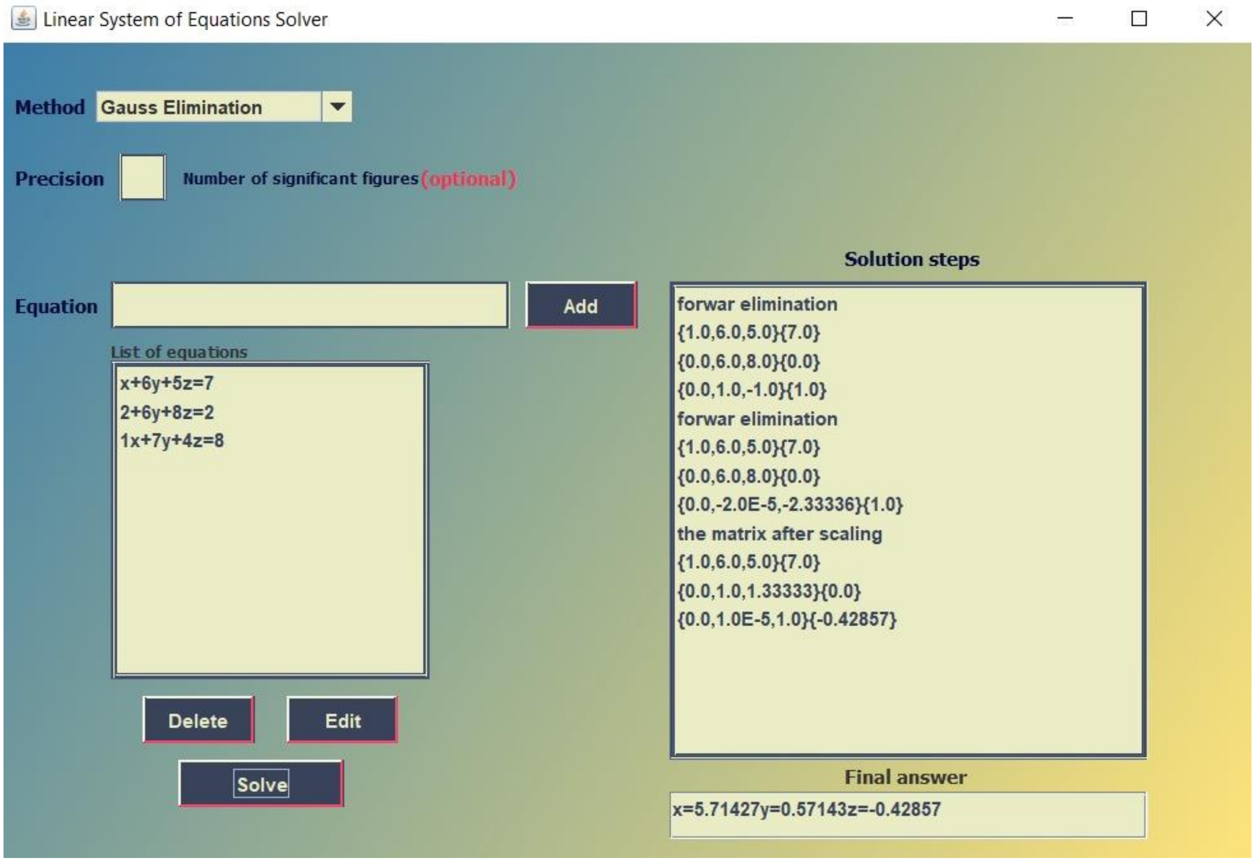
```
                if (stop OR iterations == iterationsNum)
                        break the loop
                P = (double[]) X.clone();
              }
              for i = 0 to i < n,increase with 1
              {
                   stat += X[i] + " ";
              }
              stat += "}";
              steps += ", \n , run time = ";
         }
         solutionsArray[0] = stat;
         solutionsArray[1] = steps;
         return solutionsArray;
      }
  }
```

b- Sample runs:

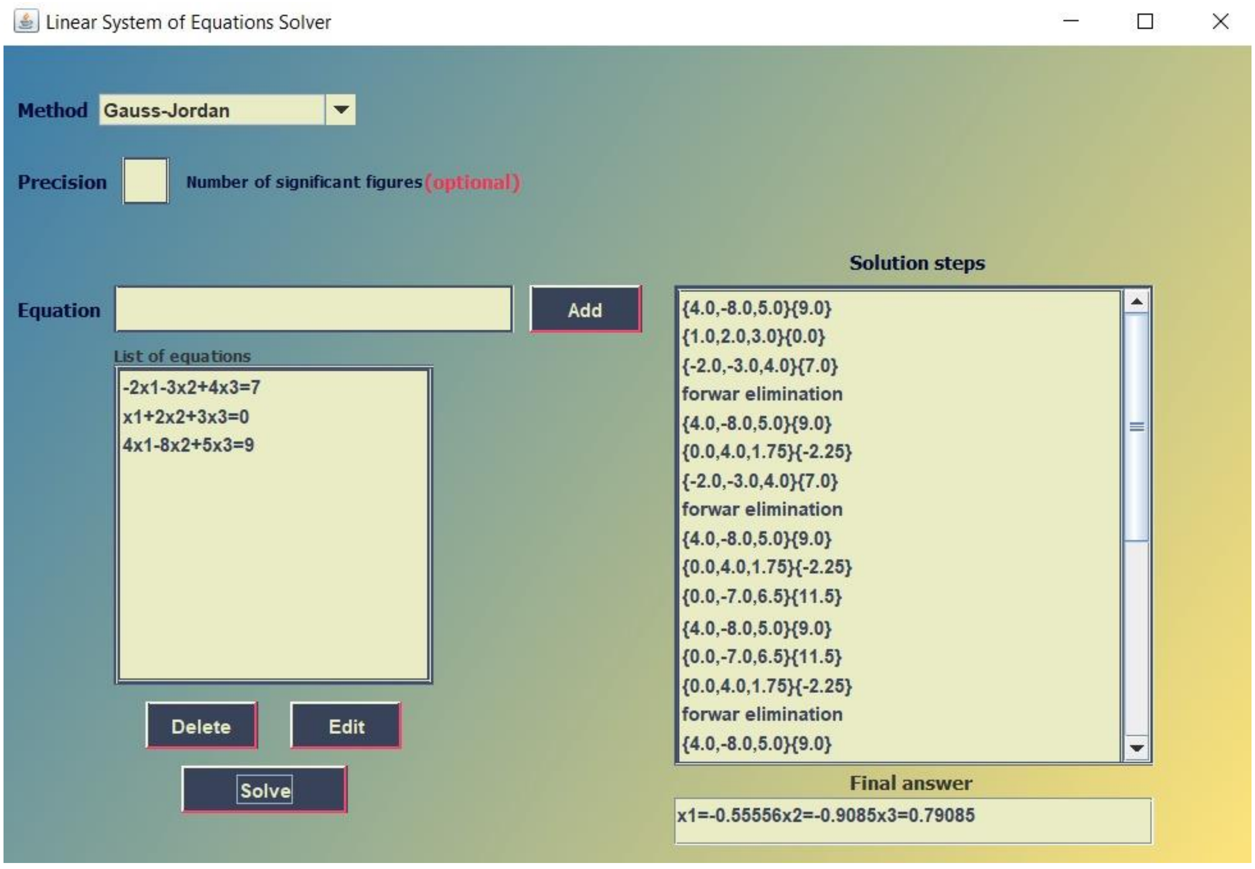
lu(downlittle):



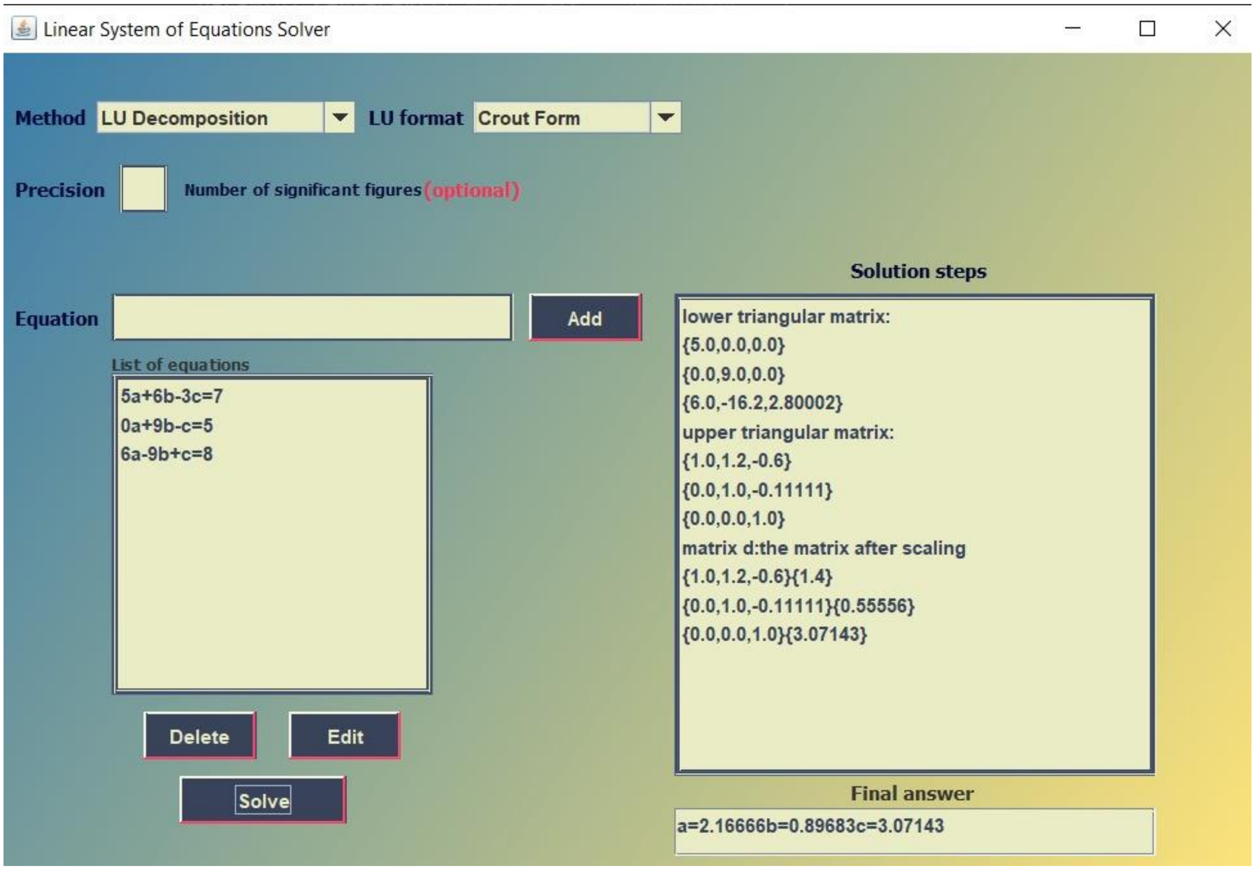
Gauss elimination:



<u>Gauss Jordan:</u>


Linear System of Equations Solver
Method
Gauss-Jordan
Precision
Number of significant figures
(optional)
Solution steps
Equation
Add
List of equations
-2x1-3x2+4x3=7
x1+2x2+3x3=0
4x1-8x2+5x3=9
{4.0,-8.0,5.0}{9.0}
{1.0,2.0,3.0}{0.0}
{-2.0,-3.0,4.0}{7.0}
forwar elimination
{4.0,-8.0,5.0}{9.0}
{0.0,4.0,1.75}{-2.25}
{-2.0,-3.0,4.0}{7.0}
forwar elimination
{4.0,-8.0,5.0}{9.0}
{0.0,4.0,1.75}{-2.25}
{0.0,-7.0,6.5}{11.5}
{4.0,-8.0,5.0}{9.0}
{0.0,-7.0,6.5}{11.5}
{0.0,4.0,1.75}{-2.25}
forwar elimination
{4.0,-8.0,5.0}{9.0}
Delete
Edit
Solve
Final answer
x1=-0.55556x2=-0.9085x3=0.79085

<u>lu(crout):</u>



c- Comparison:

d- Data structure used:

• The main data structure used in the code is the array which we used for many reasons:

- using split method which fill the array for specific size.
- it is multidimensional data structure which ease creating the matrix
- we have the number of equations which enables us to create the matrix easily without loss in memory.

- it is easy to access, add and remove elements from the matrix