ASSIGNMENT 1

NUMERICAL ANALYSIS

ANS GOMAA ABD ELSAYED
18010421
AMR IBRAHIM AHMED
18011169
SAMER MOHAMED EL SAYED
18010778
SEIF MOHAMED MAHMOUD
18010834
ABDELRAHMAN ASHRAF HASSAN
18012538

1-Pseudo-code

a- Gauss Elimination:-

```
for j=1 to n-1
        //checking if the pivot is zero
        If ajj =0
                K=j
                for k=k+1 to n
                        if akj =0
                                continue
                        break
        //swapping the rows
        row number j = row number k
        row number k = row number j


        //forward elimination
        for i =1+s"s=0" to n-1
                M=a(i+1)j / ajj
                for p = each column
                a(i+1)p = a(i+1)p – M * ajp
                b(i+1) =b(i+1)-M*bj
        s=s+1

//backward substitution
xn = bn / ann
for i = n-1 downto 1
        sum = 0
        for j = i+1 to n
                sum = sum + aij * xj
        xi = (bi – sum) / aii
```

b- Gauss Elimination using pivoting:-

```
// pivoting matrix
for i=1 to n-1{
        for j = i+1 to n {
                if (abs(aji) > abs(aii) {
                        //swapping rows
                        row i = row j
                        row j = row i
                }
        }
}

//gauss elimination
for j=1 to n-1
        //checking if the pivot is zero
        If ajj =0
                K=j
                for k=k+1 to n
                        if akj =0
                                continue
                        break
        //swapping the rows
        row number j = row number k
        row number k = row number j


        //forward elimination
        for i =1+s"s=0" to n-1
                M=a(i+1)j / ajj
                for p = each column
                a(i+1)p = a(i+1)p – M * ajp
                b(i+1) =b(i+1)-M*bj
        s=s+1

//backward substitution
xn = bn / ann
for i = n-1 downto 1
        sum = 0
```

for j = i+1 to n

sum = sum + a_{ij} * x_j

$x_i = (b_i – sum) / a_{ii}$

c- Gauss Jordan:-

```
//gauss elimination
for j=1 to n-1
        //checking if the pivot is zero
        If a_jj =0
                K=j
                for k=k+1 to n
                        if a_kj =0
                                continue
                        break
        //swapping the rows
        row number j = row number k
        row number k = row number j


        //forward elimination
        for i =1+s"s=0" to n-1
                M=a_(i+1)j / a_jj
                for p = each column
                a_(i+1)p = a_(i+1)p – M * a_jp
                b_(i+1) =b_(i+1)-M*b_j
        s=s+1

//getting answers
for j=n-1 downto 1 {
        sum=0
        for i=1 to n-j
                sum=sum+a_j(n+1-i) * x_(n+1-i)
        x_j =(a_j(n+1) -sum) / a_jj
}
x=transpose(x)
```

d- LU Decomposition:-

Doolittle Decomposition

```
//checking if it can be decomposed
[~, n] = size(a(1,:))
a = round(a,precision,'significant')
   x = ones(n, 1)
   o = 1 : n
   s = ones(n, 1)
   er = 0
   [a, o, er] = decompose(a, n, o, s, er, precision)
   If er = -1 {
      // can't be solved
      x = -1
      return
   else
      x = substitute(a, o, n, b, x, precision)

//decomposition
decompose(a, n, o, s, er){
   //finding scales
   for h = 1 to n {
      s(h) = abs(a(h, 1))
      for w = 2 to n {
         if(s(h) < abs(a(h, w))) {
            s(h) = abs(a(h, w))
         }
      }
   }
   for k = 1 to n-1 {
      o = pivot(a, o, s, n, k, precision)
      if(~isfinite(abs(a(o(k),k)) / s(o(k)))) {
         er = -1
         return
      }
      for r = k+1 to n {
         a(o(r), k) = round(a(o(r), k) / a(o(k), k),precision,'significant')
         for c = k+1 to n {
            a(o(r), c) = a(o(r), c) - round((a(o(r), k) * a(o(k), c)),precision,'significant')
            a(o(r), c) = round(a(o(r), c),precision,'significant')
```

```
            }
        }
    }
    if( ~isfinite(abs(a(o(n),n)) / s(o(n))) ) {
        er = -1
     }
}

pivot(a, o, s, n, k, precision){
    p = k
    big = abs( round(a(o(k),k) / s(o(k)),precision,'significant') )
    for i = k+1 to n {
        dummy = abs(round(a(o(i),k) / s(o(i)),precision,'significant'))
        if(dummy > big) {
            big = dummy
            p = i
        }
    }
    // swapping the rows indeces
    dummy = o(p)
    o(p) = o(k)
    o(k) = dummy
}
substitute(a, o, n, b, x, precision){
    y = ones(n, 1)
    y(o(1)) = b(o(1))
   // forward substitution
    for q = 2 to n {
        sum = b(o(q))
        for p = 1 to q-1 {
            sum = sum - round(a(o(q), p) * y(o(p)),precision,'significant')
            sum = round(sum,precision,'significant')
        }
        y(o(q)) = sum
    }

    // backward substitution
    x(n) = y(o(n)) / a(o(n), n)
    for q = n-1 downto 1 {
        sum = 0
```

```
    for p = q+1 to n {
      sum = sum + round(a(o(q), p) * x(p),precision,'significant')
      sum = round(sum,precision,'significant')
    }
    x(q) = round((y(o(q)) - sum) / a(o(q), q),precision,'significant')
  }
}
```

Crout Decomposition

```
croutLU(a, b, precision) {
  [~, n] = size(a)
  a = round(a,precision,'significant')
  x = ones(n, 1)
  o = 1 : n
  s = ones(n, 1)
  er = 0
  [a, o, er] = decompose(a, n, o, s, er, precision)
  if(er == -1) {
    // can't be solved
    x = -1
    return
  }
  else
    [x, er] = substitute(a, o, n, b, x, er, precision)
    if(er == -1) {
      x = -1
      return
    }
  }
}
decompose(a, n, o, s, er, precision) {
  //finding scales
  for h = 1 to n {
    s(h) = abs(a(h, 1))
    for w = 2 to n {
      if(s(h) < abs(a(h, w))) {
        s(h) = abs(a(h, w))
      }
    }
  }
```

```
   o = pivot(a, o, s, n, 1, precision)
   if(~isfinite(abs(a(o(1),1)) / s(o(1)))) {
      er = -1
      return
   }
   for j = 2 to n {
      a(o(1), j) = round(a(o(1), j) / a(o(1), 1),precision,'significant')
      if(~isfinite(a(o(1), j))) {
         er = -1
         return
      }
   }
   for i = 2 to n {
      for j = 2 to n {
         if( j <= i) {
            // Lij
            for k = 1 to j-1{
               a(o(i), j) = a(o(i), j) - round(a(o(i), k) * a(o(k), j),precision,'significant')
               a(o(i), j) = round(a(o(i), j),precision,'significant')
            }
         else
            a(o(i), j) = (a(o(i), j) - round(a(o(i), 1:o(i) - 1) * a(1:o(i) - 1,
j),precision,'significant')) / a(o(i), i)
            a(o(i), j) = round(a(o(i), j),precision,'significant')
         }
      }
   }
}
pivot(a, o, s, n, k, precision) {
   p = k
   big = abs( round(a(o(k),k) / s(o(k)),precision,'significant') )
   for i = k+1 to n {
      dummy = abs(round(a(o(i),k) / s(o(i)),precision,'significant'))
      if(dummy > big) {
         big = dummy
         p = i
      }
   }
   // swapping the rows indeces
   dummy = o(p)
```

```
    o(p) = o(k)
    o(k) = dummy
}
substitute(a, o, n, b, x, er, precision) {
    y = ones(n, 1)
    y(o(1)) = b(o(1)) / a(o(1), 1)
    if( ~isfinite(y(o(1))) ) {
        er = -1
        return
    }
    // forward substitution
    for i = 2 to n {
        sum = 0
        for j = 1 to i-1 {
            sum = sum + round(a(o(i), j) * y(o(j)),precision,'significant')
            sum = round(sum,precision,'significant')
        }
        y(o(i)) = round((b(o(i)) - sum),precision,'significant')/ a(o(i), i)
        y(o(i)) = round(y(o(i)),precision,'significant')
        if( ~isfinite(y(o(i))) ) {
            er = -1
            return
        }
    }
    // backward substitution
    x(n) = y(o(n))
    for i = n-1 downto 1 {
        sum = y(o(i))
        for j = i+1 to n {
            sum = sum - round(a(o(i), j) * x(j),precision,'significant')
            sum = round(sum,precision,'significant')
        }
        x(i) = sum
    }
}
```

Cholesky Decomposition

```
choleskyD(a, b, precision) {
    [~, n] = size(a)
    a = round(a,precision,'significant')
```

```
x = ones(n, 1)
for i = 1 to n {
   for j = 1 to I {
      sum = 0
      if (j == i) {
         for k = 1 to j – 1 {
            sum = sum + round(a(j,k) * a(j,k),precision,'significant')
            sum = round(sum,precision,'significant')
         }
         a(j,j) = round(sqrt(a(j,j) - sum),precision,'significant')
      else
         for k = 1 to j-1 {
            sum = sum + round(a(i,k) * a(j,k),precision,'significant')
            sum = round(sum,precision,'significant')
         }
         a(i,j) = round((a(i,j) - sum),precision,'significant') / a(j,j)
         a(i,j) = round(a(i,j),precision,'significant')
      }
   }
}
for i = 1 to n {
   if(a(i,i) == 0) {
      x = -1
      return
   }
}
for i = 1 to n {
   for j = i+1 to n {
      a(i, j) = a(j , i)
   }
}
y = ones(n, 1)
// forward substitution
y(1) = b(1) / a(1, 1)
for i = 2 to n {
   sum = 0
   for j = 1 to  i-1 {
      sum = sum + round(a(i, j) * y(j),precision,'significant')
      sum = round(sum,precision,'significant')
   }
```

```
    y(i) = round((b(i) - sum),precision,'significant') / a(i, i)
    y(i) = round(y(i),precision,'significant')
  }
  // backward substitution
  x(n) = round(y(n) / a(n, n),precision,'significant')
  for q = n-1 downto 1 {
    sum = 0
    for p = q+1 to n {
      sum = sum + round(a(q, p) * x(p),precision,'significant')
      sum = round(sum,precision,'significant')
    }
    x(q) = round((y(q) - sum),precision,'significant') / a(q, q)
    x(q) = round(x(q),precision,'significant')
  }
}
```

e- Gauss Seidil:-

//Gauss seidel iteration

- *Check the precision of the input matrices.*
- *Check if the matrix is diagonally dominant.*
    - *if not then try all permutations of the input matrix to get the nearest diagonally matrix of the input to have more chance for convergence.*

- *loop over the number of iterations given*
    - *iterate over the rows of the matrix to get Xi = (Bi – aj * Xjnew– ak * Xk new- ……. – az * Xz new) / ai.*
    - *round after each operation to achieve the precision required.*
    - *Calculate the error in each iteration to check the convergence*

//Gauss seidel relative error

- *Check the precision of the input matrices.*
- *Check if the matrix is diagonally dominant.*
    - *if not then try all permutations of the input matrix to get the nearest diagonally matrix of the input to have more chance for convergence.*

- *loop until you get error less than the tolerance limit*
    - *iterate over the rows of the matrix to get Xi = (Bi – aj * Xj new– ak * Xk new- ……. – az * Xz new) / ai.*
    - *round after each operation to achieve the precision required.*
    - *Calculate the error in each iteration to check the convergence and check if we can get the relative error required or not.*
        - *Break if it does not converge.*

f- Jacobi Iteration:-

//Jacobi iteration

- *Check the precision of the input matrices.*
- *Check if the matrix is diagonally dominant.*
    - *if not then try all permutations of the input matrix to get the nearest diagonally matrix of the input to have more chance for convergence.*

- *loop over the number of iterations given*
    - *save the old guess.*
    - *iterate over the rows of the matrix to get Xi = (Bi – aj * Xj old– ak * Xk old- ……. – az * Xz old) / ai.*
    - *round after each operation to achieve the precision required.*
    - *Calculate the error in each iteration to check the convergence*

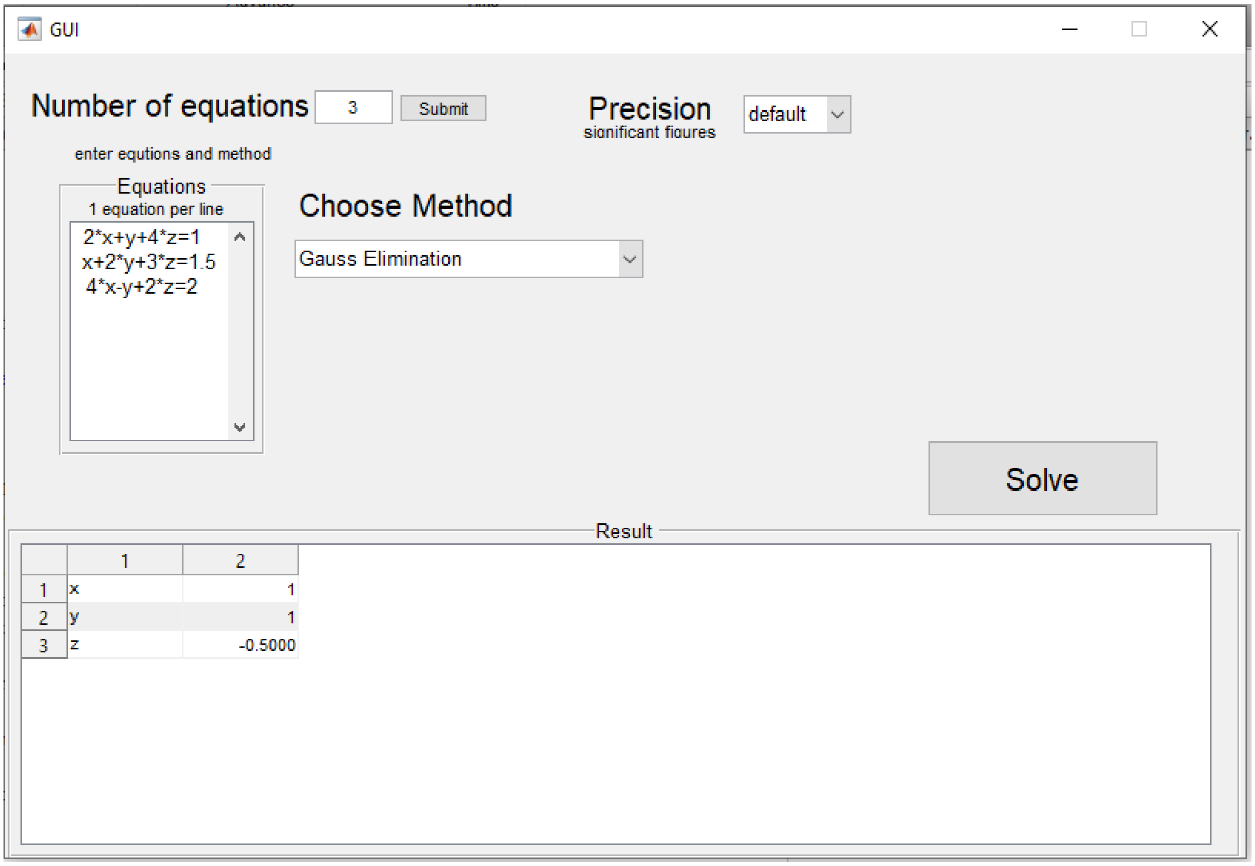
//Jacobi relative error

- *Check the precision of the input matrices.*
- *Check if the matrix is diagonally dominant.*
    - *if not then try all permutations of the input matrix to get the nearest diagonally matrix of the input to have more chance for convergence.*

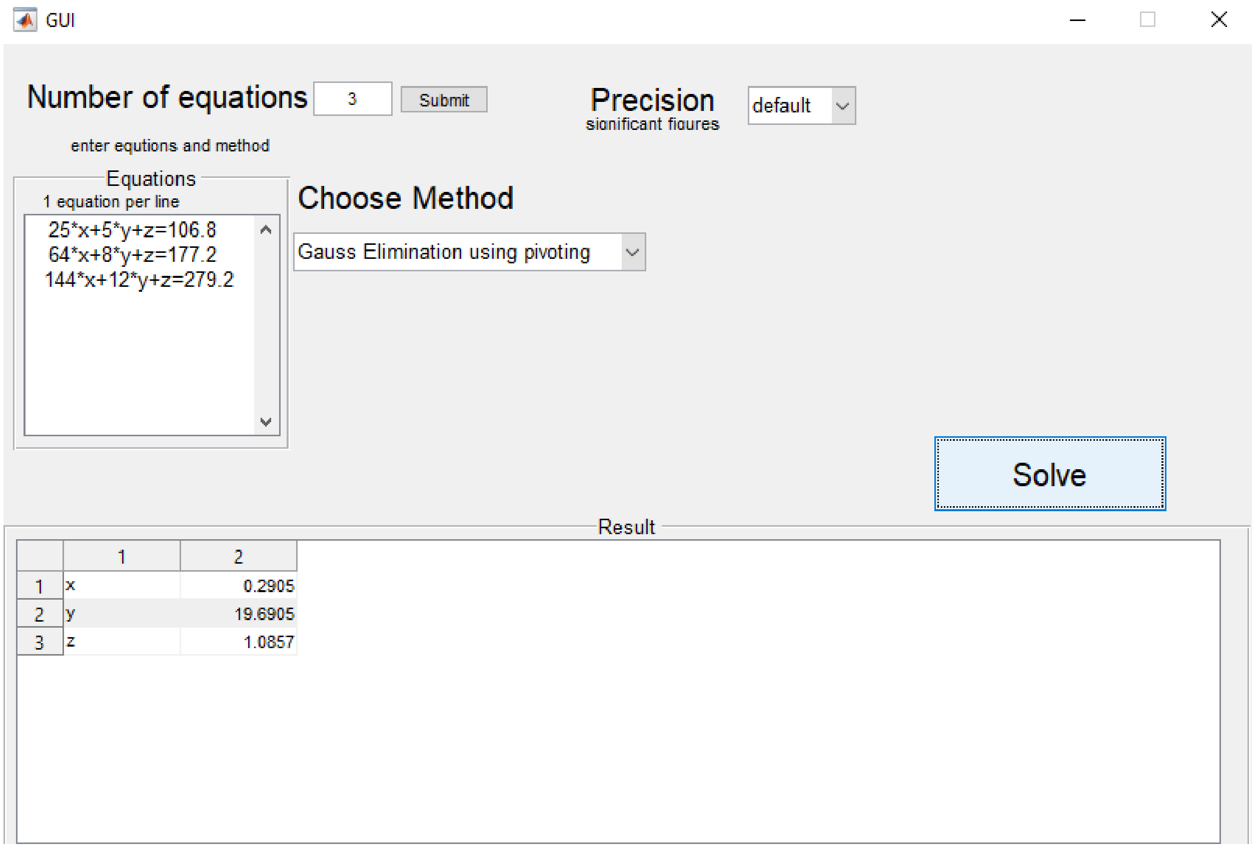
- *loop until you get error less than the tolerance limit*
    - *save the old guess.*
    - *iterate over the rows of the matrix to get Xi = (Bi – aj * Xj old– ak * Xk old- ……. – az * Xz old) / ai.*
    - *round after each operation to achieve the precision required.*
    - *Calculate the error in each iteration to check the convergence and check if we can get the relative error required or not.*
        - *Break if it does not converge.*

2-Sample runs

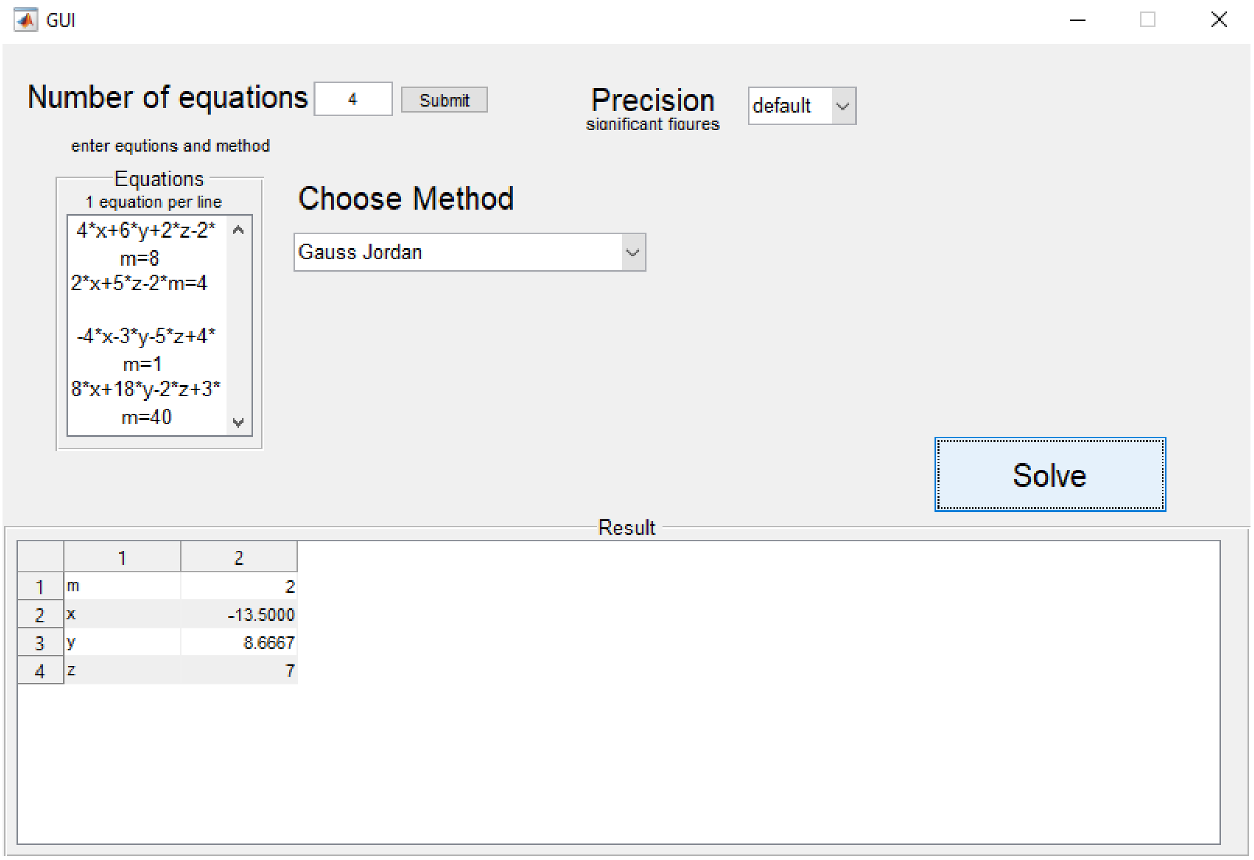
a- Gauss Elimination:-



b- Gauss Elimination using pivoting:-

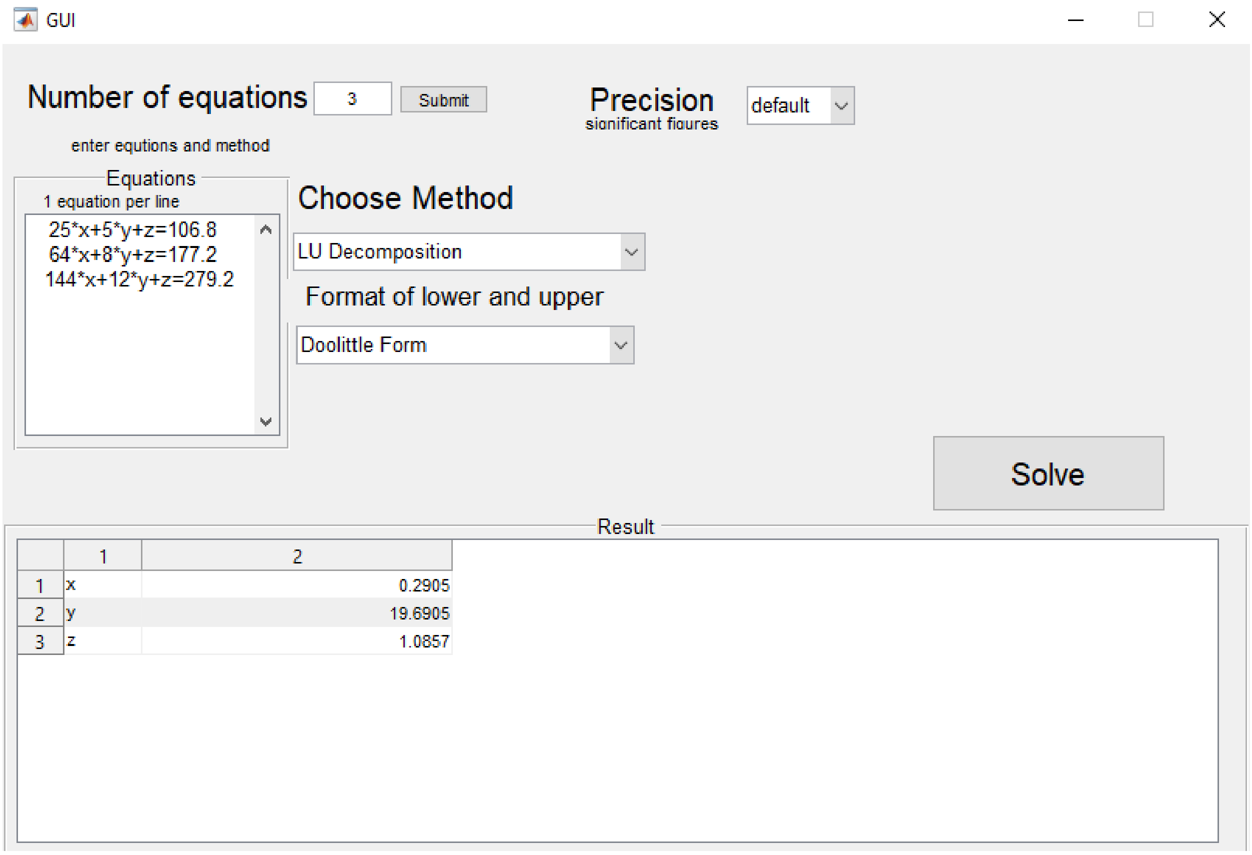


c- Gauss Jordan:-

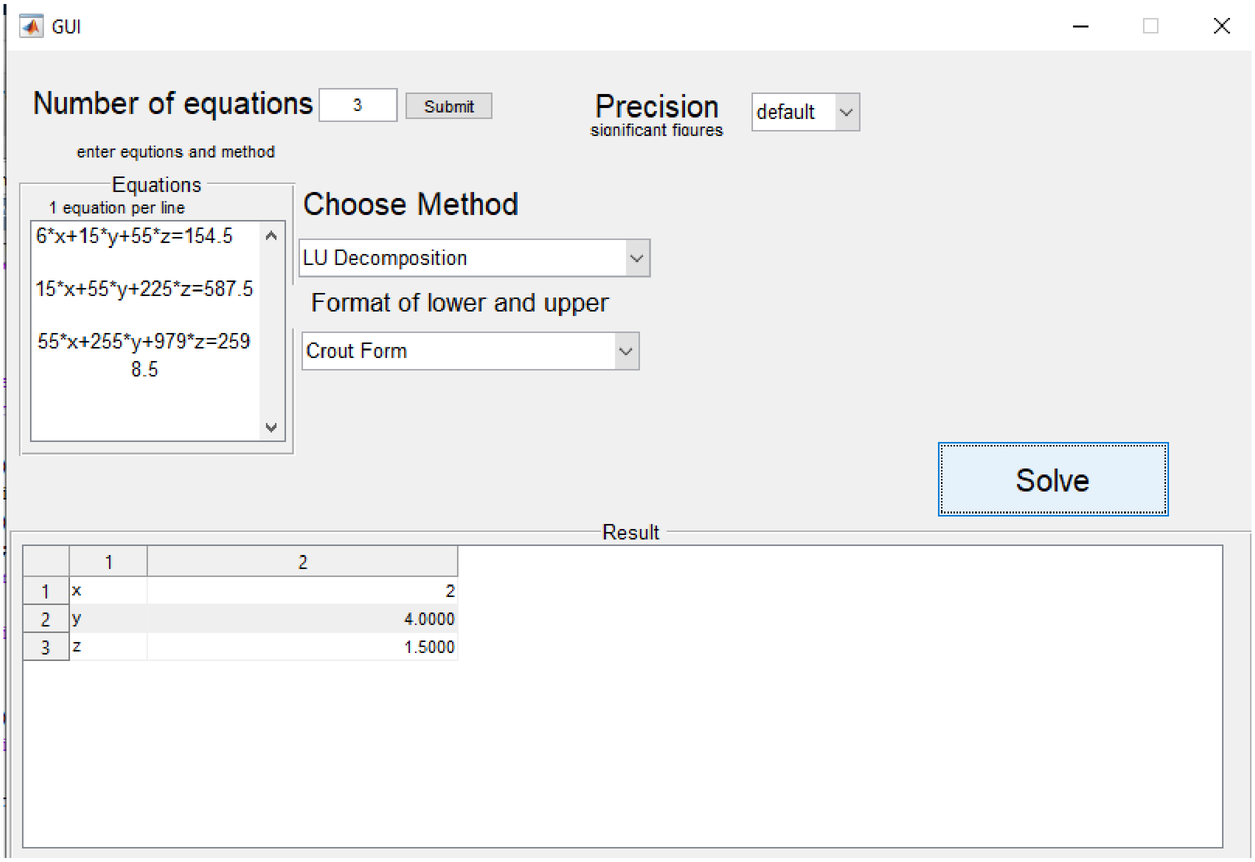


d- LU Decomposition:-

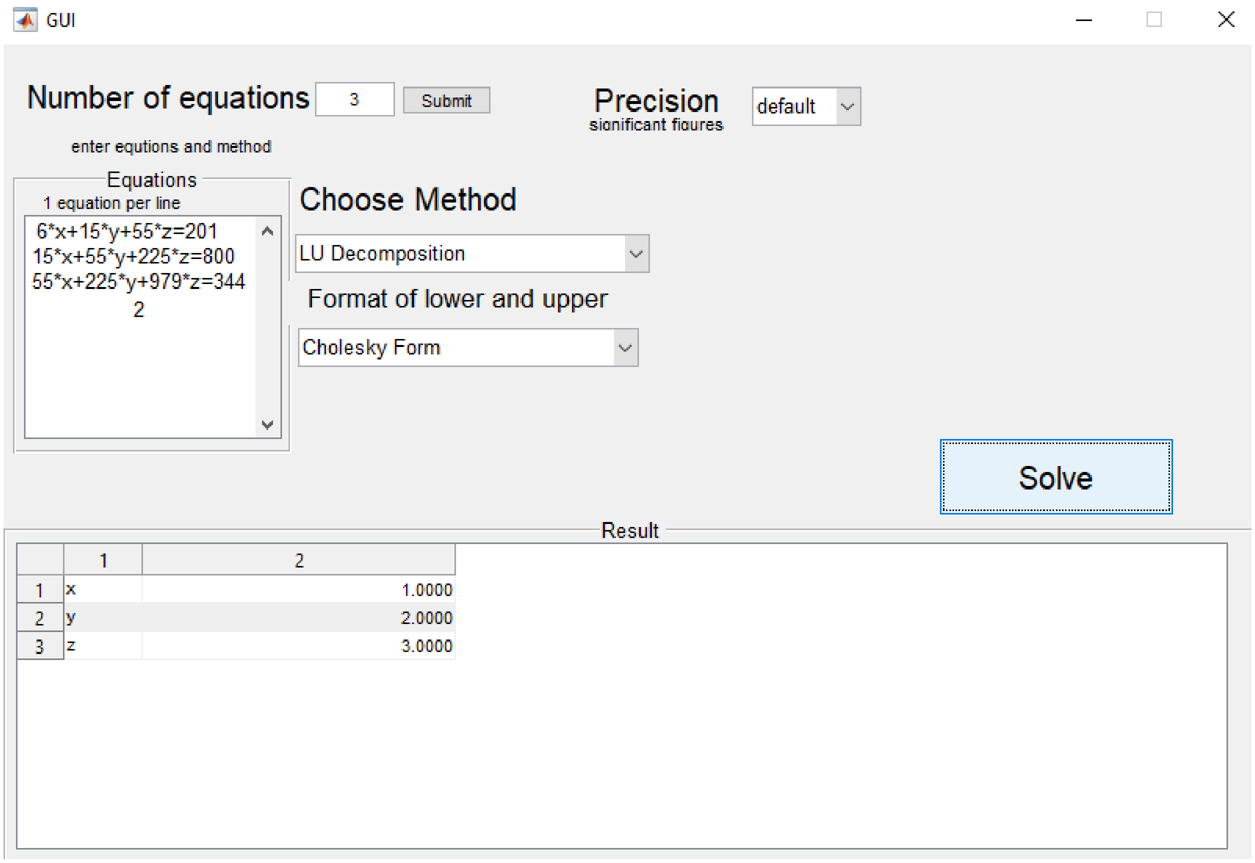
Dolittle



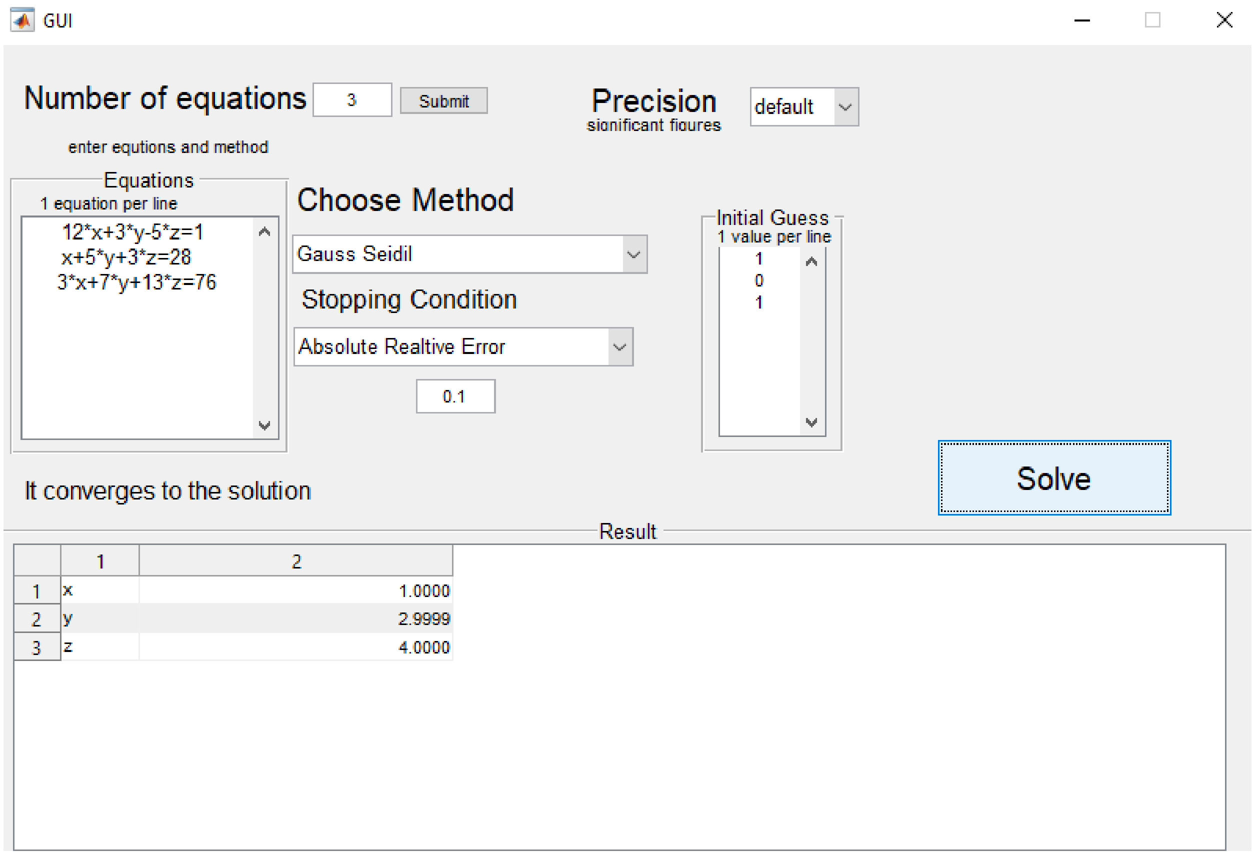
Crout

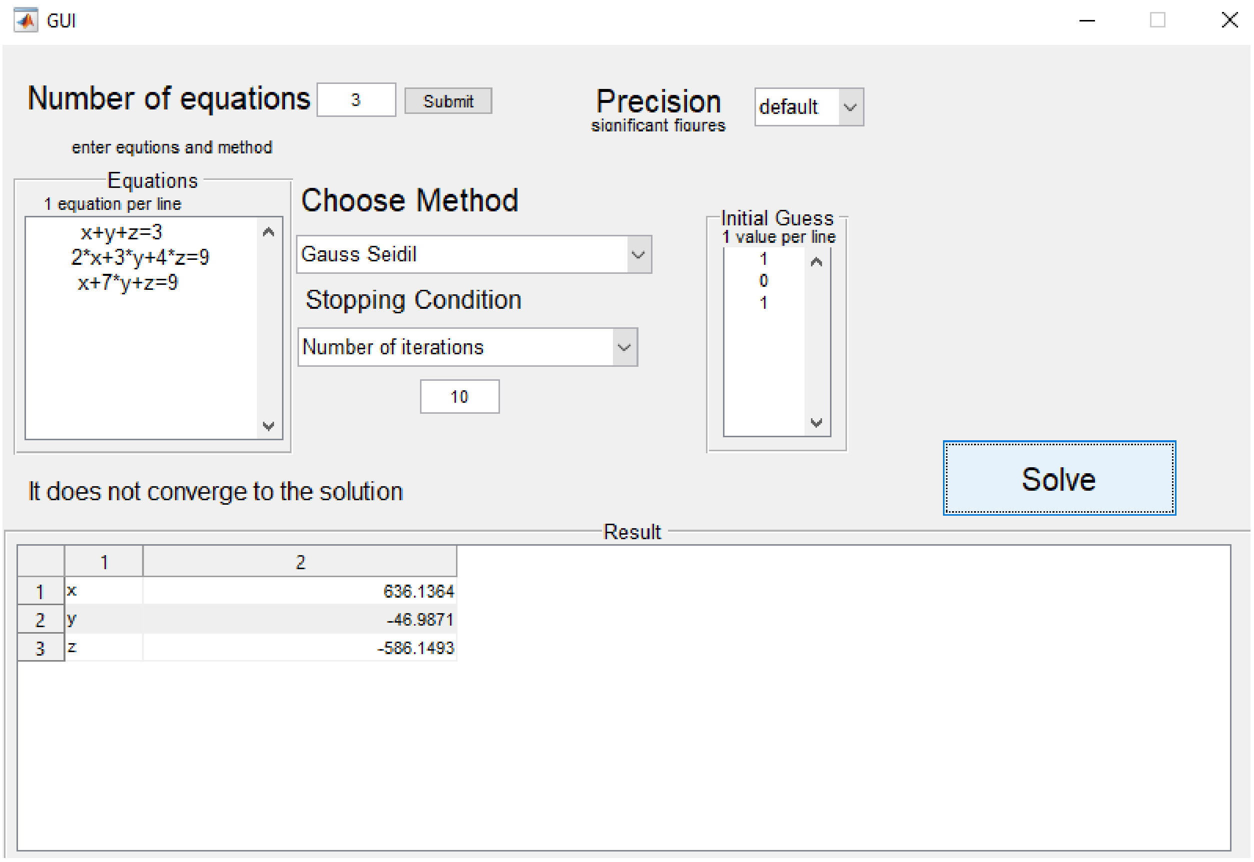

GUI
Number of equations
3
Submit
Precision
significant figures
default
enter equtions and method
Equations
1 equation per line
6*x+15*y+55*z=154.5
15*x+55*y+225*z=587.5
55*x+255*y+979*z=2598.5
Choose Method
LU Decomposition
Format of lower and upper
Crout Form
Solve
Result
1
2
x 2
y 4.0000
z 1.5000

Chelosky

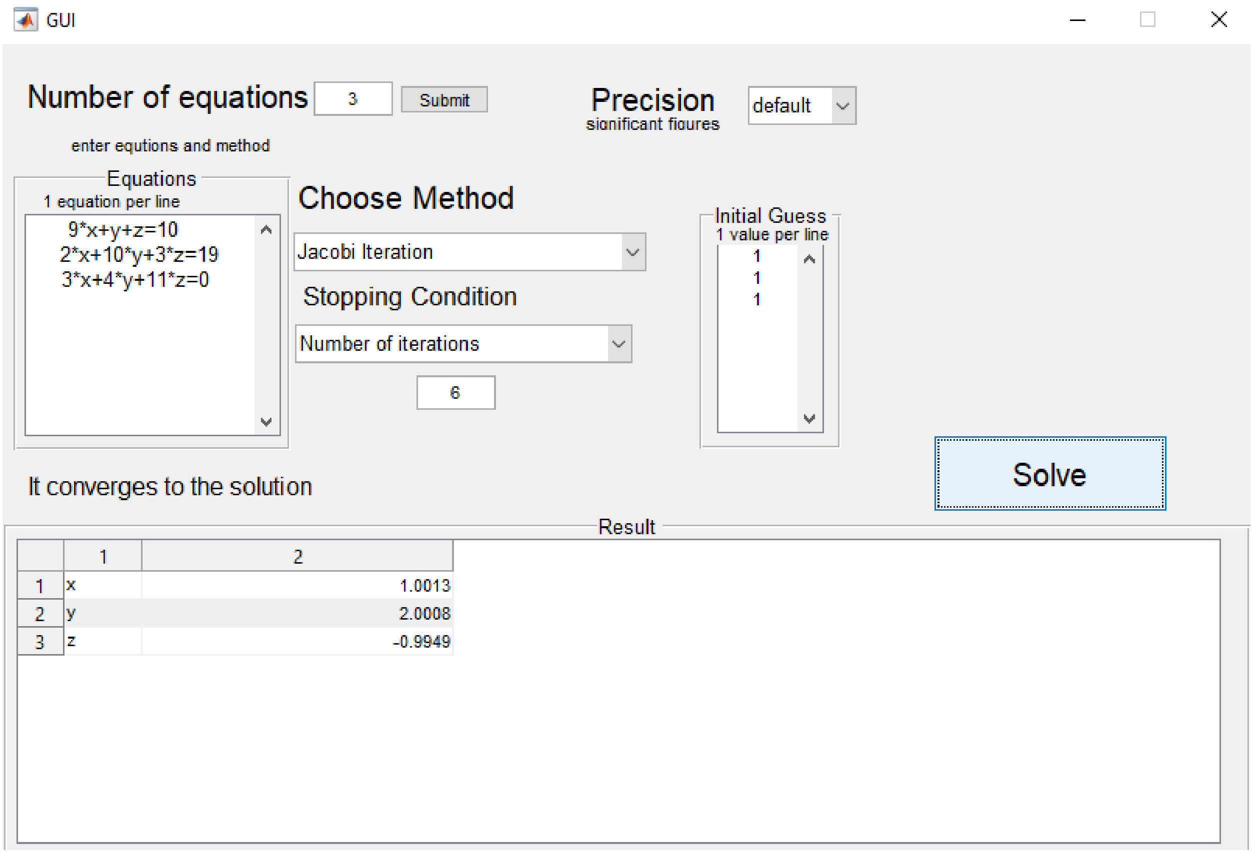

GUI
Number of equations
3
Submit
Precision
significant figures
default
enter equtions and method
Equations
1 equation per line
6*x+15*y+55*z=201
15*x+55*y+225*z=800
55*x+225*y+979*z=344
2
Choose Method
LU Decomposition
Format of lower and upper
Cholesky Form
Solve
Result
1
2
1 x 1.0000
2 y 2.0000
3 z 3.0000

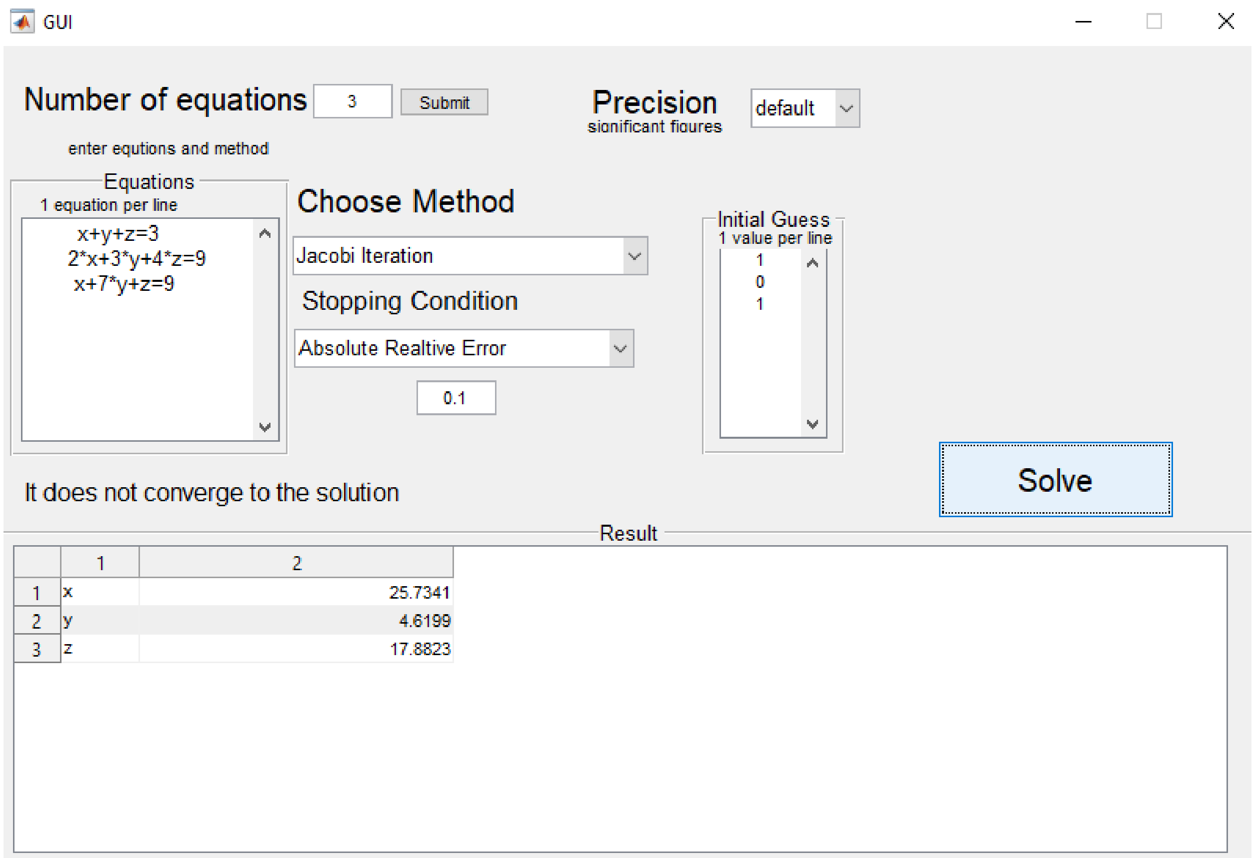
e- Gauss Seidil:-




GUI
Number of equations
3
Submit
Precision
significant figures
default
enter equtions and method
Equations
1 equation per line
x+y+z=3
2*x+3*y+4*z=9
x+7*y+z=9
Choose Method
Gauss Seidil
Stopping Condition
Number of iterations
10
Initial Guess
1 value per line
1
0
1
Solve
It does not converge to the solution
Result
x 636.1364
y -46.9871
z -586.1493

f- Jacobi Iteration:-




GUI
Number of equations
3
Submit
Precision
significant figures
default
enter equtions and method
Equations
1 equation per line
x+y+z=3
2*x+3*y+4*z=9
x+7*y+z=9
Choose Method
Jacobi Iteration
Stopping Condition
Absolute Realtive Error
0.1
Initial Guess
1 value per line
1
0
1
Solve
It does not converge to the solution
Result
1
2
1 x 25.7341
2 y 4.6199
3 z 17.8823

3-Date structure used

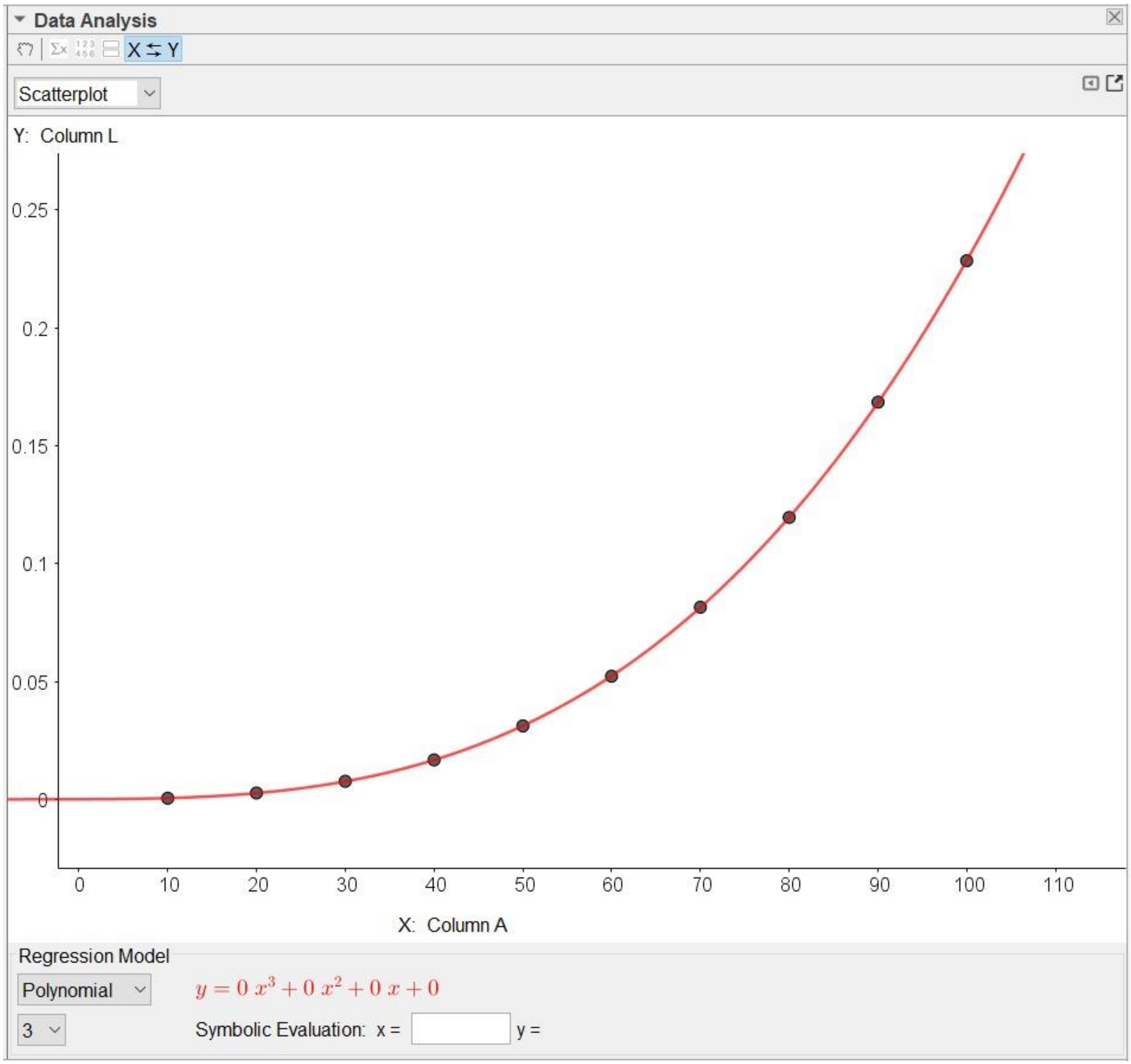
Arrays

4-Comparison between different methods

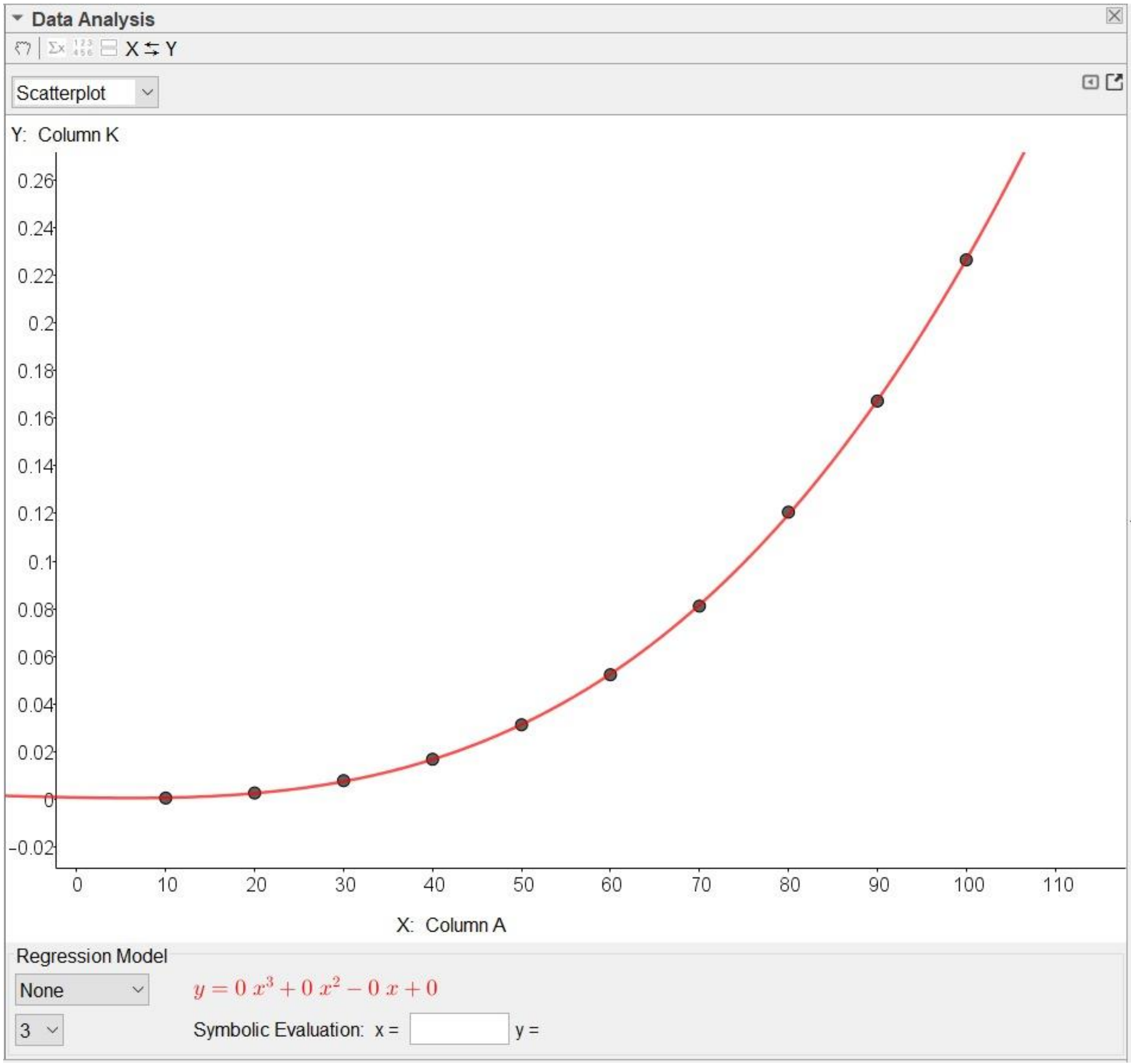
Method Name	***Time complexity***	***Convergence***	***Best Case***	***Worst Case***	***Precisions***
Gauss Elimination	*K** $O(n^3)$	*Converges*	*Matrix that doesn't require pivoting*	*Singular Matrix*	*Precise*
Gauss Elimination using pivoting	*K** $O(n^3)$	*Converges*	*Matrix that doesn't require pivoting*	*Singular Matrix*	*Precise*
Gauss Jordan	*K** $O(n^3)$	*Converges*	*Matrix that doesn't require pivoting*	*Singular Matrix*	*Precise*
LU Decomposition	$O(n^3)$*+ K** $O(n^2)$	*Converges*	*Matrix that doesn't require pivoting* *Chelosky-> Positive definite matrix*	*Singular Matrix* *Chelosky ->Not positive definite matrix*	*Precise*
Gauss Seidil	*N** $O(n^2)$	*It may not converge or it converges very slowly* *If Diagonally dominant guarantee for convergence*	*Matrix is Diagonally dominant*	*There are no combinations to make it Diagonally dominant*	*Gauss Seidil is faster to get to the required precision*
Jacobi Iteration	*N** $O(n^2)$	*If Diagonally dominant guarantee for convergence*	*Matrix is Diagonally dominant*	*There are no combinations to make it Diagonally dominant*	*Gauss Seidil is faster*

5-Analysis of the behavior

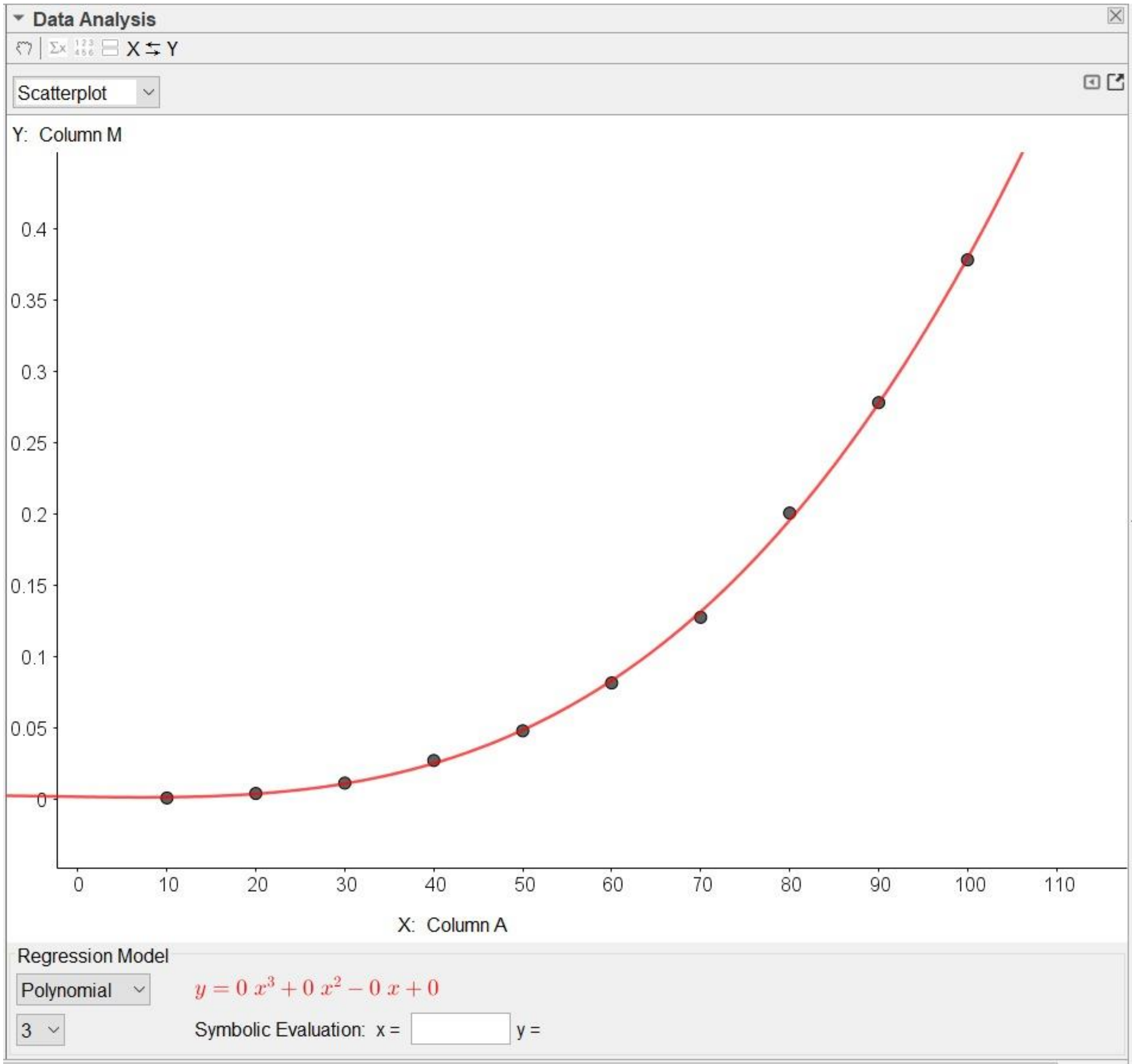
a- Gauss Elimination:-



b- Gauss Elimination using pivoting:-

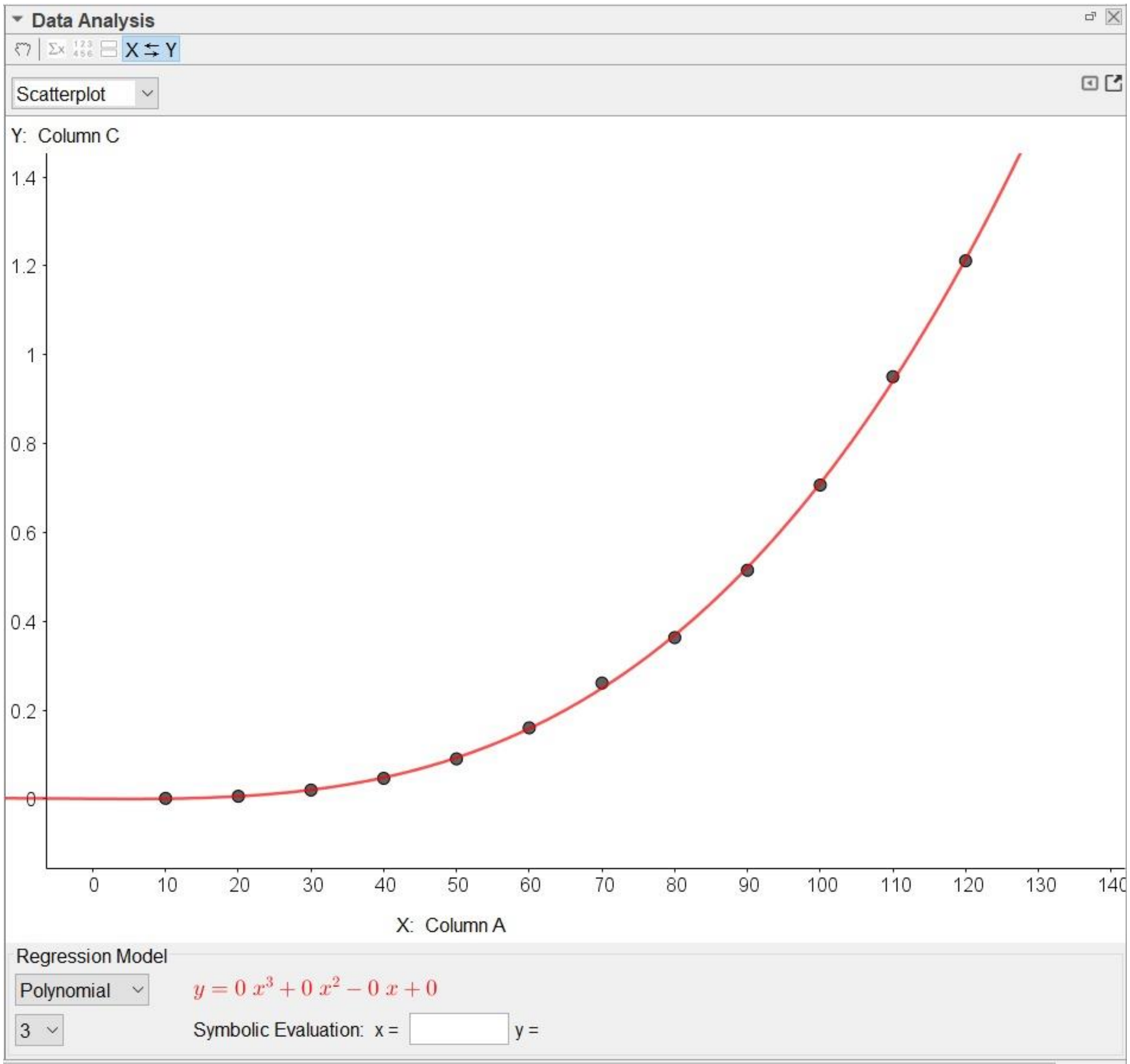


c- Gauss Jordan:-

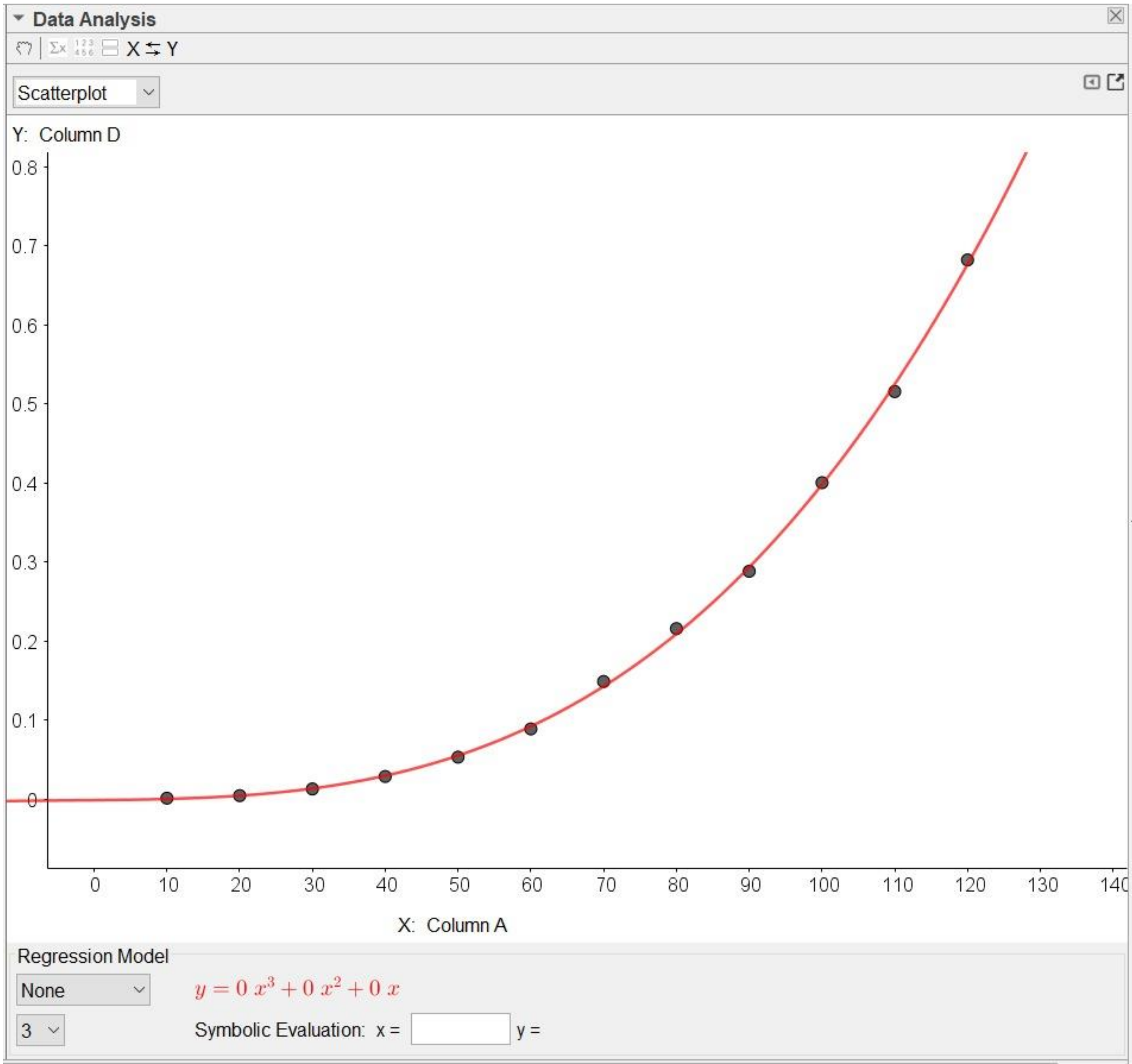


d- LU Decomposition:-

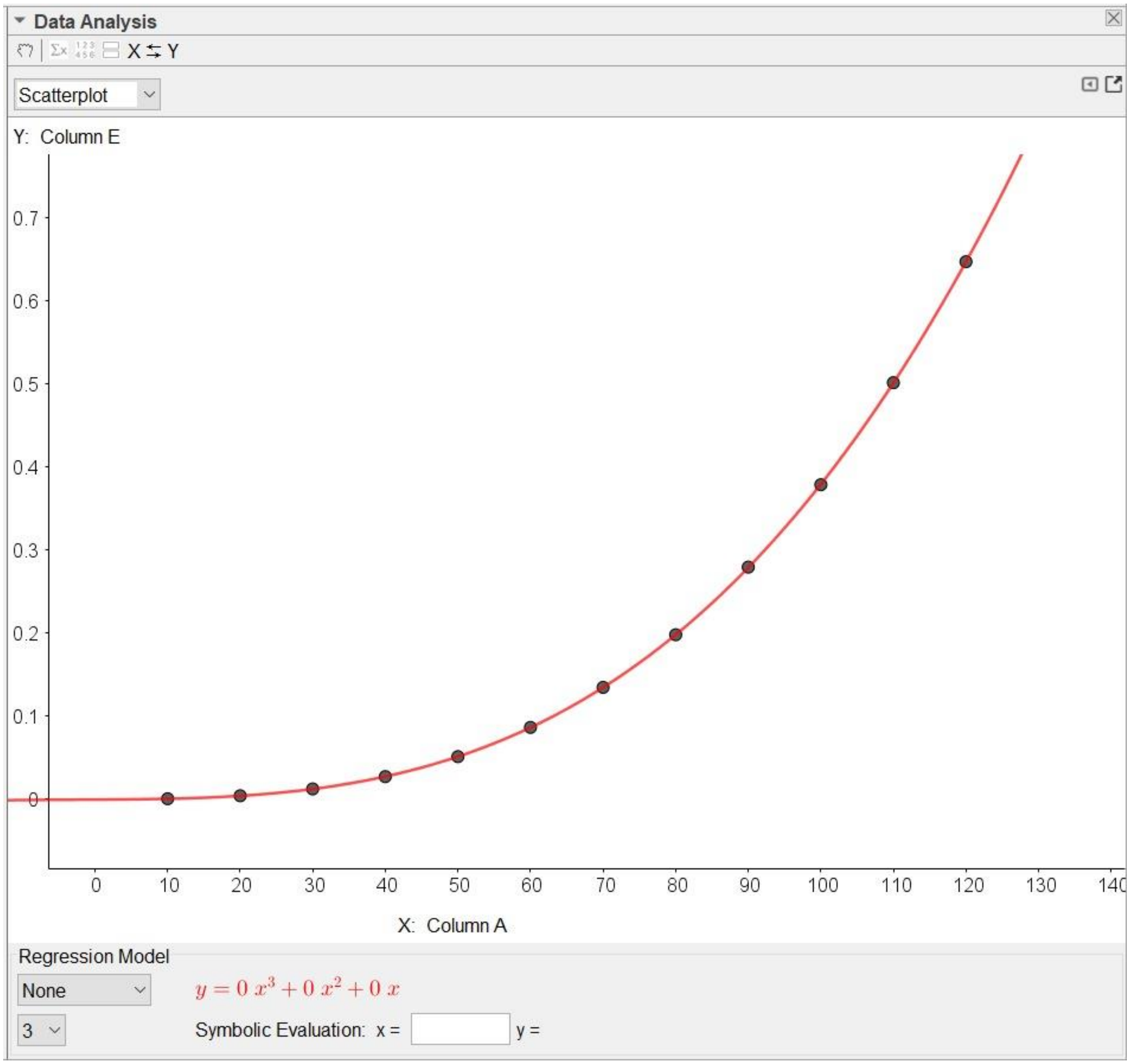
Doolittle decomposition



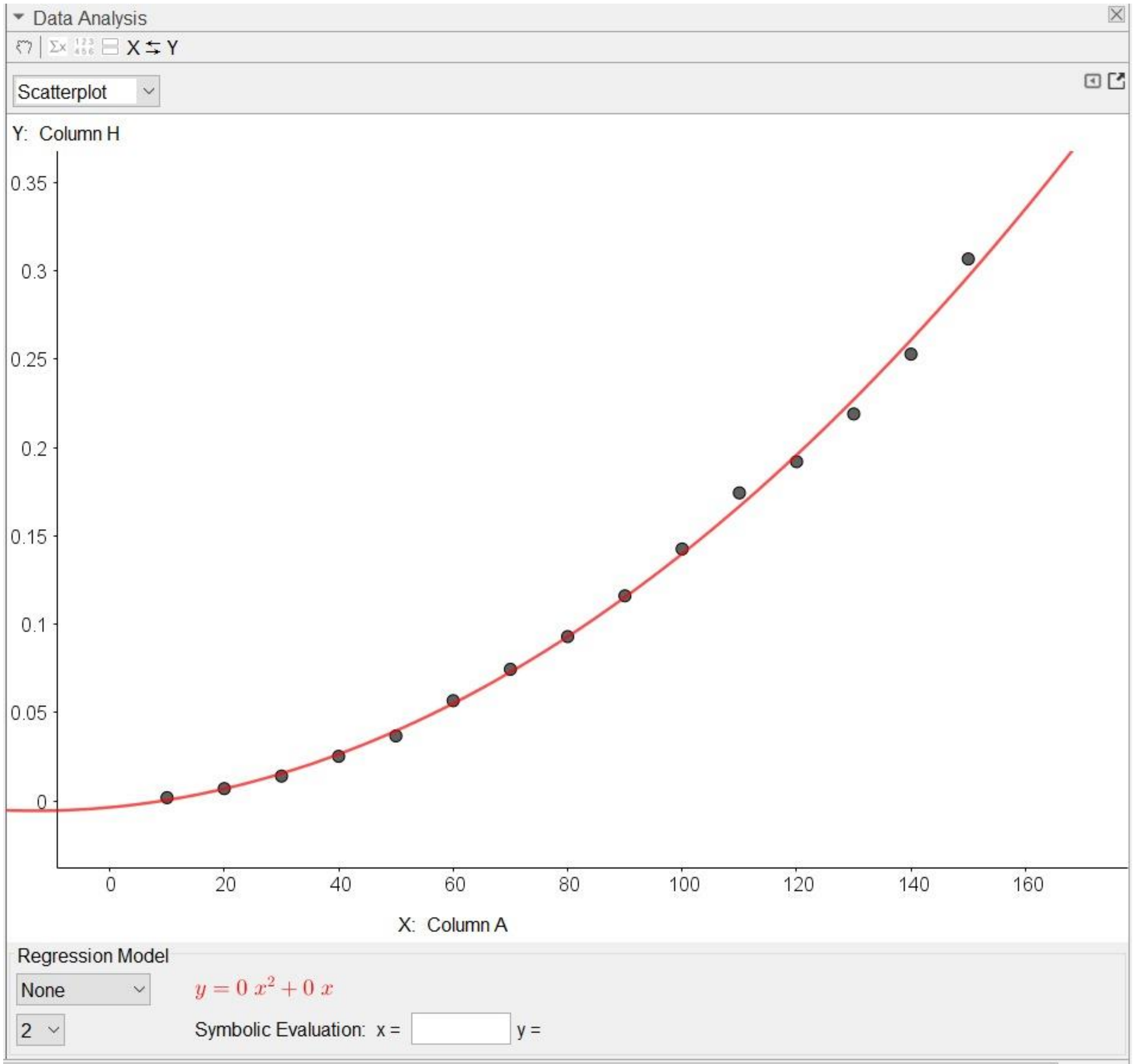
Crout Decomposition



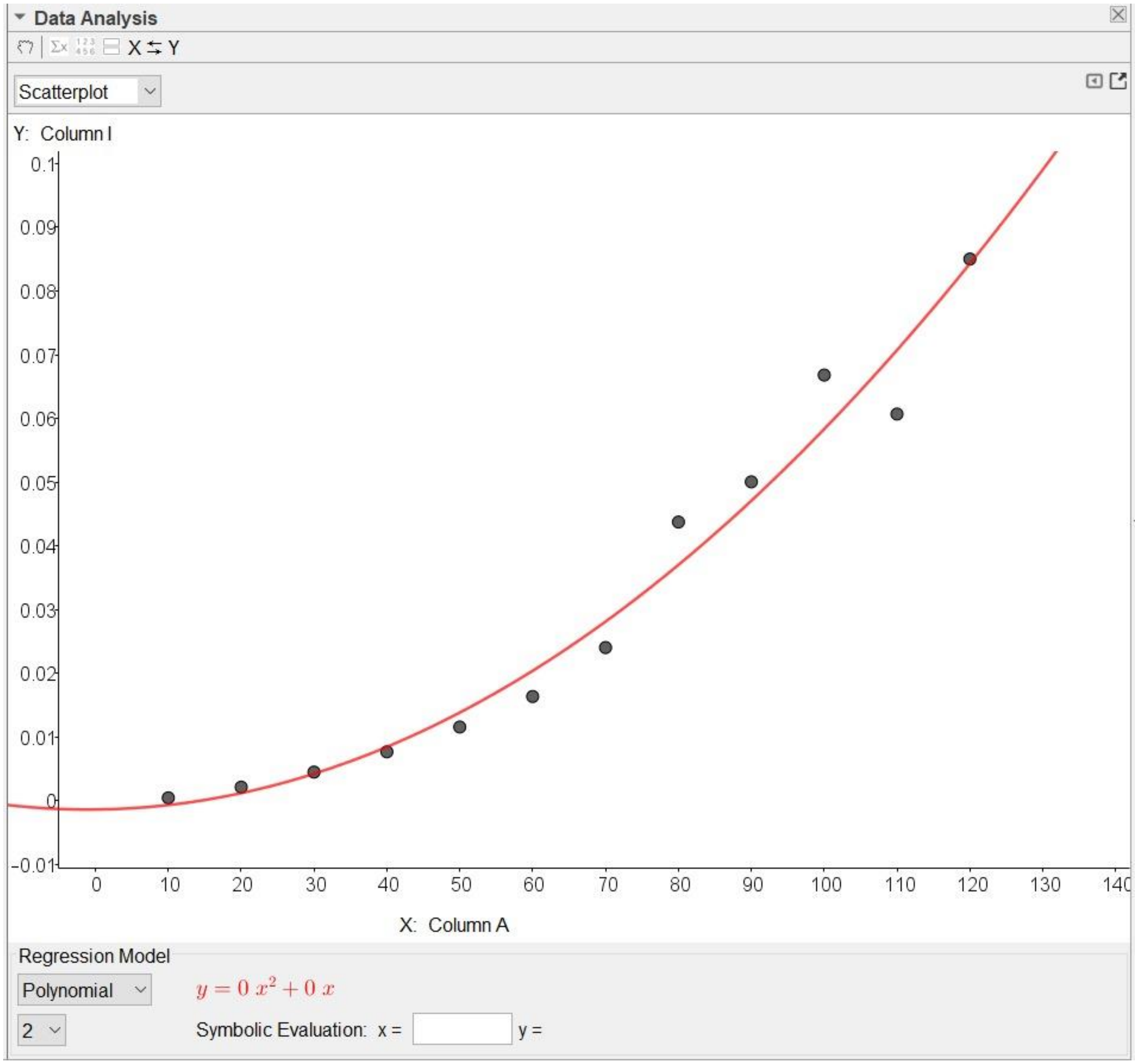
Cholesky Decomposition



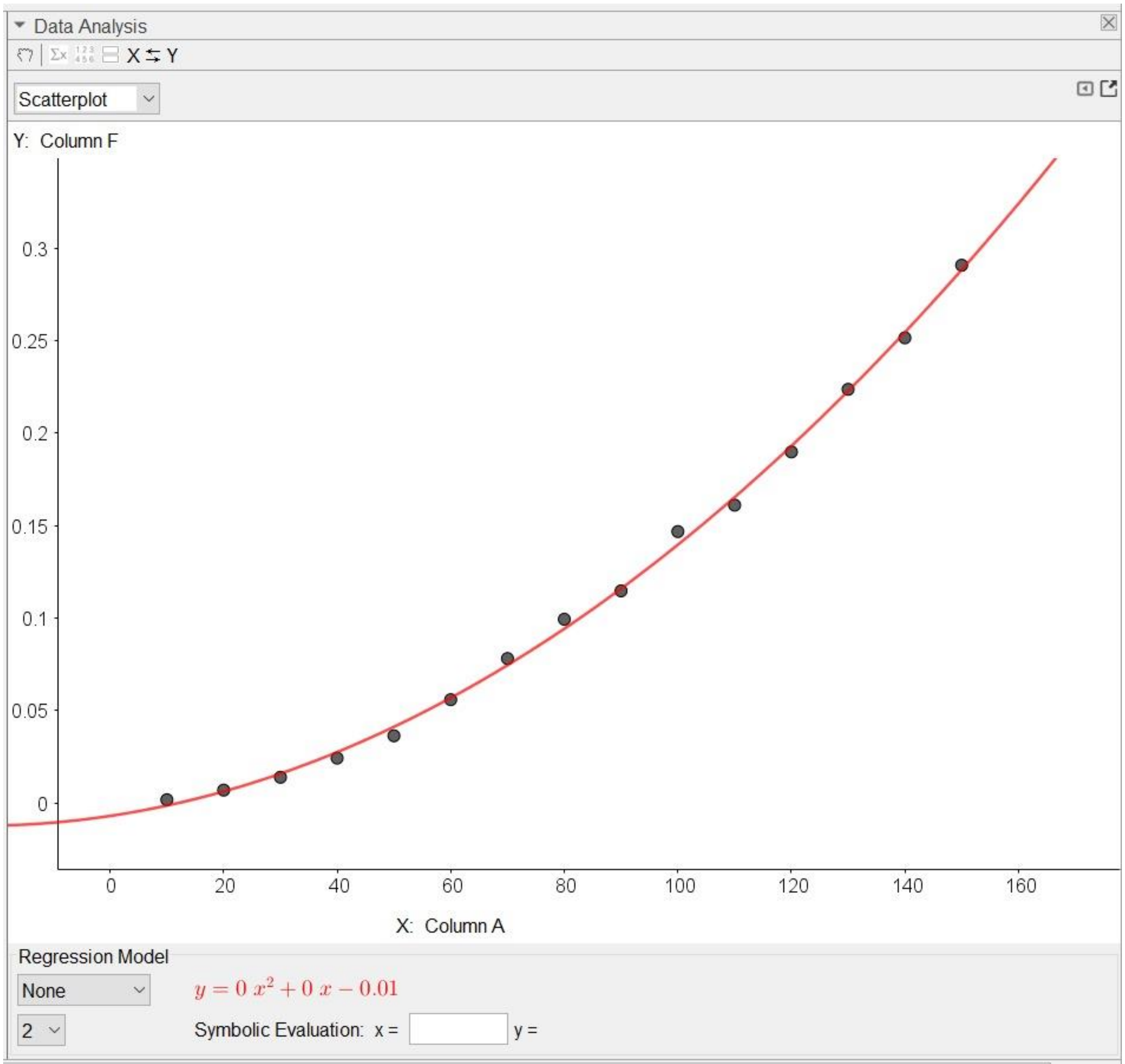
e- Gauss Seidil:-



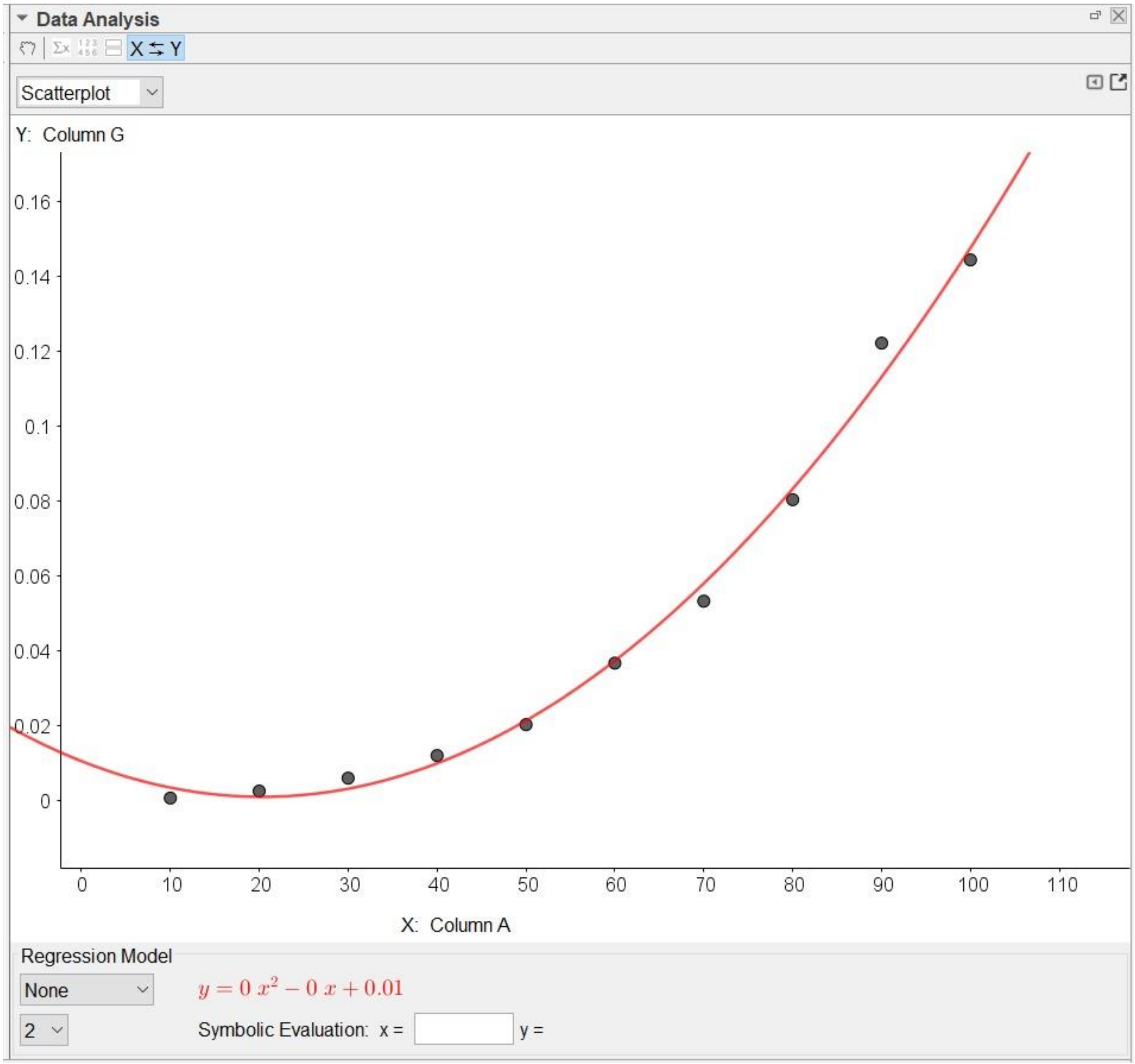
- Gauss Seidil relative error:-



f- Jacobi Iteration:-



- Jacobi relative error:-



	A	B	C	D	E	F	G	H	I	J	K
1	10	0.0013529	0.00119	0.00073283	0.00185	0.00058393	0.0018592	0.00047293	0.00046463	0.00045503	0.00062693
2	20	0.006312	0.0044151	0.0043478	0.0070468	0.0024476	0.0069961	0.0021654	0.0025953	0.0027146	0.003828
3	30	0.020166	0.01302	0.012565	0.014043	0.0059024	0.014078	0.0045303	0.0077616	0.0076878	0.011082
4	40	0.046634	0.028706	0.027429	0.024365	0.011975	0.025263	0.0077025	0.016753	0.016704	0.02695
5	50	0.090348	0.053181	0.051423	0.036447	0.020201	0.036813	0.011582	0.031256	0.031147	0.047825
6	60	0.16025	0.088926	0.08661	0.055944	0.036638	0.056808	0.016388	0.052255	0.052221	0.081463
7	70	0.26103	0.14878	0.13472	0.078177	0.053231	0.074563	0.024066	0.081127	0.081506	0.12751
8	80	0.36336	0.21572	0.19804	0.099396	0.080326	0.093035	0.043752	0.12057	0.11957	0.20078
9	90	0.51507	0.2883	0.27949	0.11475	0.12217	0.11612	0.050059	0.16726	0.16844	0.27831
10	100	0.70674	0.40044	0.37859	0.14682	0.14444	0.14265	0.066823	0.22654	0.22847	0.37863
11	110	0.95049	0.51578	0.50129	0.16108		0.17441	0.060697			
12	120	1.2116	0.68237	0.64702	0.18983		0.19205	0.085027			
13	130				0.22364		0.21904				
14	140				0.25149		0.25292				
15	150				0.29077		0.30681				
16	number of rows	doolittleLU	croutLU	choleskyD	Jacobi$_i$terations	Jacobi$_r$elativeError	Gauss$_S$eidel$_i$teration	Gauss$_S$eidel$_r$elativeError	gaussEliminationWithPivoting	gaussElimination	gaussJordan
17											
18											
19											
20											
21											
22											
23											
24											
25											