Numerical Computations Project Phase#1

Fall Semester 2021

Veronica Romany Hanna	19016156
Norah Ahmed Hassan	19016810
Mariam Tarek Mostafa	19016637
Mariam Mohamed Ahmed	19017274
Yara Hossam Abdelaziz	19016871

Problem Statement

- Implementation of a program which takes an input of a system of linear equations and solves it using the desired method.
- Compare and analyze the behavior of the different numerical methods used for solving system of linear equations:
    1. Gauss Elimination.
    2. Gauss Jordan.
    3. LU Decomposition.
    4. Gauss Seidel.
    5. Jacobi Iteration.

Design Decisions

1. The program is written in Python.
2. The equations can be of any format.
3. Number of equations must be equal to the number of variables. Otherwise, an error will be displayed.
4. Invalid input format generates an error.
5. Partial pivoting is applied when applicable.
6. Users can set the decimal rounding value. If not chosen, the default value is 4.
7. Coefficients must be numbers.
8. Each variable can be of any combination of letters and numbers but it must start with a letter.

9. Any operator out of (+,-,/,.,=) is considered invalid.
10. In Gauss Seidel and Jacobi methods, the diagonal elements must be nonzero. Otherwise, an error will be displayed.

Pseudocode for each method

1. Gauss Elimination

Start

Get A,B, numberofunknowns,noofdecimal

If rankA != rankAug

    Return System is inconsistent and there is no solution

Else if rankA equals rankAug but less than number of unknowns

    Return system has infinite number of solution

Else

    For counter i in matrix A

    Get the maximum abs element in column i

        If(the element in the diagonal !=maxelement)

        for k in range(i+1,n):   //pivoting is applied

            if abs(A[k,i])> abs(A[i,i]):

                A[[i,k]]= A[[k,i]]

                f[[i,k]]= f[[k,i]]

```
                End if
        For j in range(i+1,n)      //elimination
            m = A[j,i]/A[i,i]       //get the multiply
            roundm=round(m,noofdecimal)
            A[j,:] =A[j,:] - roundm*A[i,:]
            f[j] = f[j] - roundm*f[i]
       End For
```

Return Back_Subs(A,f,noofdecimal) //after finishing pivioting and forward elimination we call backward substitution function

Function Back_Subs(A,f,noofdecimal):

- Start
- Get A,f,noofdecimal
- Declare x the solution vector

```
x[last index] = round(f[n-1]/A[n-1,n-1],noofdecimal)
for i in range(n-2,-1,-1):
        sum_ = 0
        for j in range(i+1,n):
        sum_ = round(sum_ + A[i,j]*x[j],noofdecimal)
        x[i] = round((f[i] - sum_)/A[i,i],noofdecimal)
End for
```

return x

2. Gauss Jordan

Gauss Jordan

Start

Get A,B, numberofunknowns,noofdecimal

If rankA != rankAug

Return System is inconsistent and there is no solution

Else if rankA equals rankAug but less than number of unknowns

Return system has infinite number of solution

Else

```
 For counter i in matrix A
 Get the maximum abs element in column i
     If(the element in the diagonal !=maxelement )
     For k in range(i+1,n):   //pivoting is applied
         if abs(A[k,i])> abs(A[i,i]):
             A[[i,k]]= A[[k,i]]
               f[[i,k]]= f[[k,i]]
         End if
     End for
     End if
```

```
End For
 For counter i in matrix A
        For counter j in matrix A
      If i not equal j
           pro = A[j][i] / A[i][i];
            roundmo=round(pro,noofdecimal)
            for k in range(n):
                 A[j][k] = A[j][k] - (A[i][k]) * roundmo;
         End For
              f[j]=f[j]-f[i]*roundmo
            End IF
            End For
      End For
for i in matrix A:
      f[i] = f[i]/A[i][i]
End For
     x = np.zeros(n)
 For i in range(n):
     x[i] = round(f[i],noofdecimal)
End for
return 0,x,A,runtime
```

End

3. LU Decomposition

```
Function forward_substitution(L, b ,sigFigs):
    X[0] = b[0] / L[0]
    For i=0 to n-1
        Temp[i] = b[i]
        For j=0 to i-1
            Temp = Temp - (L[i,j] * x[j])
        X[i] = round(Temp / L[i,i], sigFigs)
Return X
Function backward_substitution(U, d, sigFigs)
    For i = n-1 to 0
      Temp = d[i]
      For j=i+1 to n-1
        Temp -= U[i,j] * x[j]
      X[i] = round(Temp / U[i,i], sigFigs)
Return x
Function doolittle(A, b,sigFigs, unknownsNumber):
 U = zero matrix
 L = identity matrix
```

If rankA != rankAug

Return System is inconsistent and there is no solution

Else if rankA equals rankAug but less than number of unknowns

Return system has infinite number of solution

Else

For counter i in matrix A

Get the maximum abs element in column i

If(the element in the diagonal !=maxelement)

For k=i+1 to n-1:   //pivoting is applied

if abs(A[k,i])> abs(A[i,i]):

A[[i,k]]= A[[k,i]]

f[[i,k]]= f[[k,i]]

For k=i+1 to n-1

A[k,i] **=** A[k,i]**/**A[i,i]

For j=i+1 to n-1

A[k,j] **-=** A[k,i]*A[i,j]

Extract L and U from compact representation of A

d = forward_substitution(L, b, sigFigs)

x =backward_substitution(U, d, sigFigs)

Return L,U,x

Function crout(A, b, sigFigs, unknownsNumber):

```
 L = zero matrix
U = identity matrix
 If rankA != rankAug
        Return System is inconsistent and there is no solution
Else if rankA equals rankAug but less than number of unknowns
        Return system has infinite number of solution
Else
     For i=0 to n-1
       For j=i to n-1
         Lower sum = 0
         For k=0 to j-1
           Lower sum += L[j,k] * U[k,i]
         L[j,i] = A[j,i] - Lower sum
       For j=i+1 to n-1
         Upper sum = 0
```

For k=0 to i-1

Upper sum += L[i,k] * U[k,j]

U[i,j] = (A[i,j] -Upper sum) / L[i, i]

d = forward_substitution(L, b, sigFigs)

x =backward_substitution(U, d, sigFigs)

Return L, U, x

Function chelosky(A, b, sigFigs, unknownsNumber):

If rankA != rankAug

Return System is inconsistent and there is no solution

Else if rankA equals rankAug but less than number of unknowns

Return system has infinite number of solution

Else if matrixA equals matrix A transpose

L = zero matrix

For j=0 to n-1

Fo r=j to n-1

If diagonal element

summation = 0

For k=0 to j-1

```
      summation+= L[i,k] power 2
     L[i,j] =squareroot(A[i,j]-summation)
    Else:
     summation = 0
     For k=0 ti j-1
          summation += L[i,k]*L[j,k]
     L[i,j] = (A[i,j] - summation) / L[j,j]
  d = forward_substitution(L, b, sigFigs)
 x =backward_substitution(U, d, sigFigs)
 Return L,L transpose,x
Else:
 Return "System is not symmetric"
```

4. Gauss Seidel

```
for i=0 to length Array
     for j=0 to length Array-i-1
       if A[j][0] < A[j + 1][0]
          swap A[j] with A[j+1]
while iteration < maximum numbers of iterations
     for i=0 to length Array
          new initial guess [i] = B[i] - A[i][j]*old intialGuess[j] / A[i][i]
```

old initial guess[i] = new initial guess[i]

Ea = norm of new initial guess and old initial guess

If (Ea < Absoluter relative error)

Return new initial guess

Return new initial guess

5. Jacobi Iteration

function jacobi(x, A, b ,tolerance ,maxIteraions, precision):

FOR k = 0 to maxIteraions:

FOR i = 0 to n-1

sum ← 0

FOR j = 0 to n-1

IF i == j:

sum = sum + b_i rounded to the specified precision

ENDIF

IF i != j:

sum ← sum - x_j*A_{ij} rounded to the specified precision

ENDIF

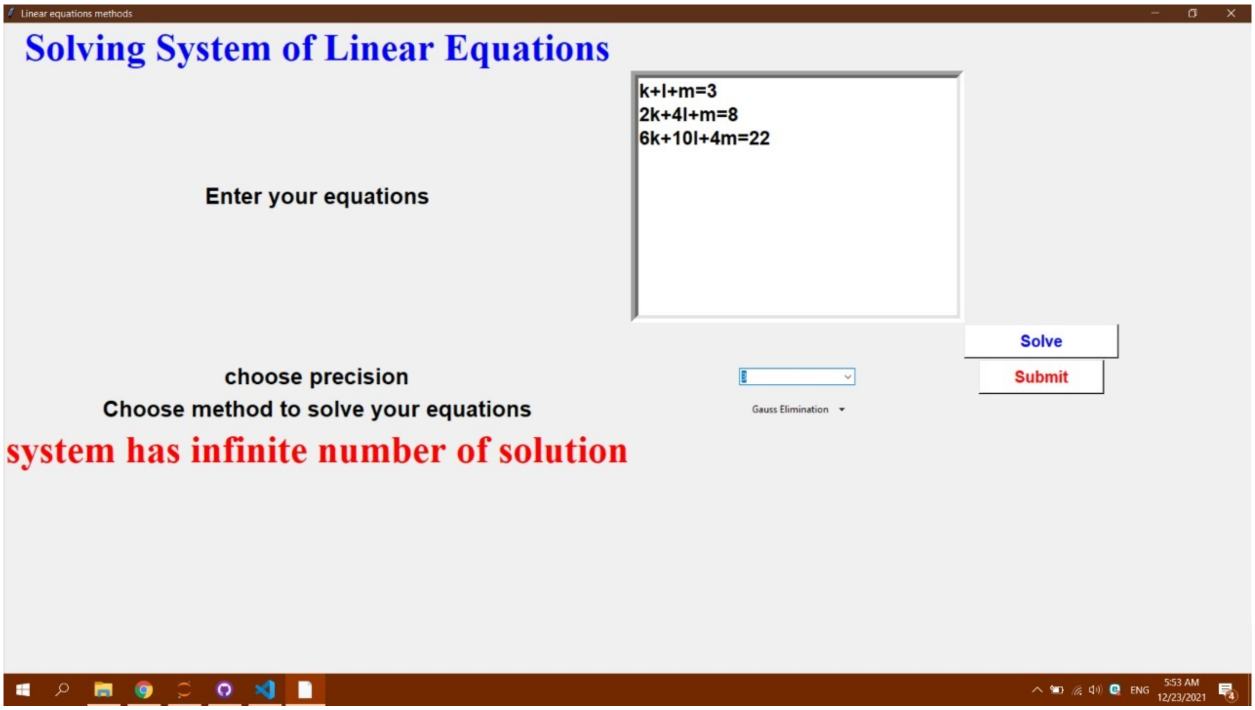
IF A_{ii} == 0:

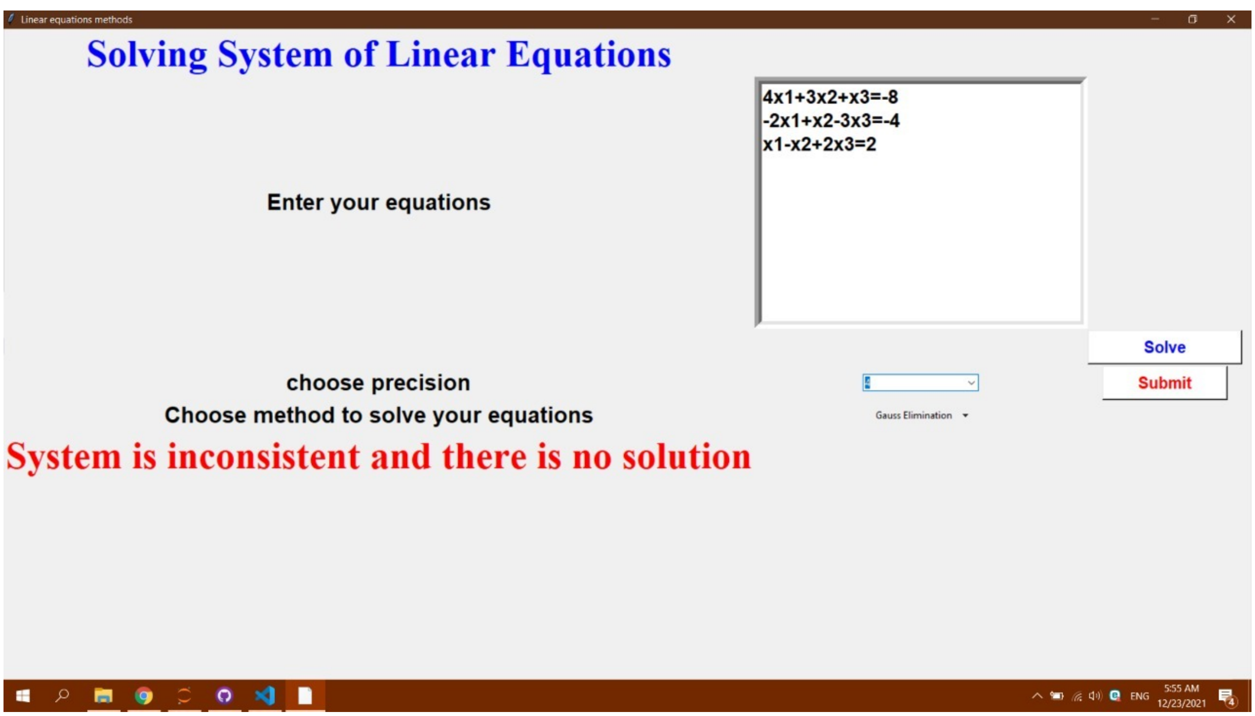
return x, Error

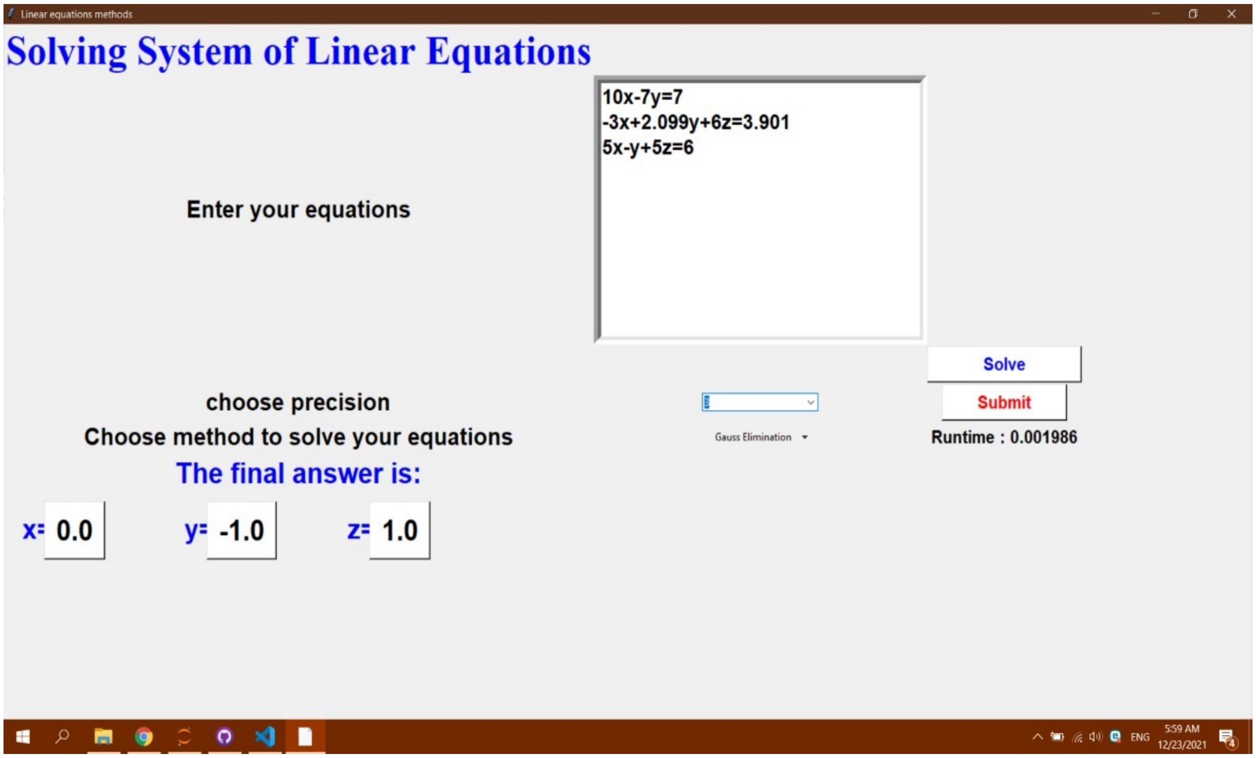
```
    ENDIF
  x_newᵢ ← sum / Aᵢᵢ rounded to the specified precision
    ENDFOR
 Ea ← calculate norm of x-new - x
 IF  Ea < tol
  return x_new, noError
    ENDIF
    ENDFOR
         x ← x_new
ENDFOR
 return x, noError
```

Sample runs

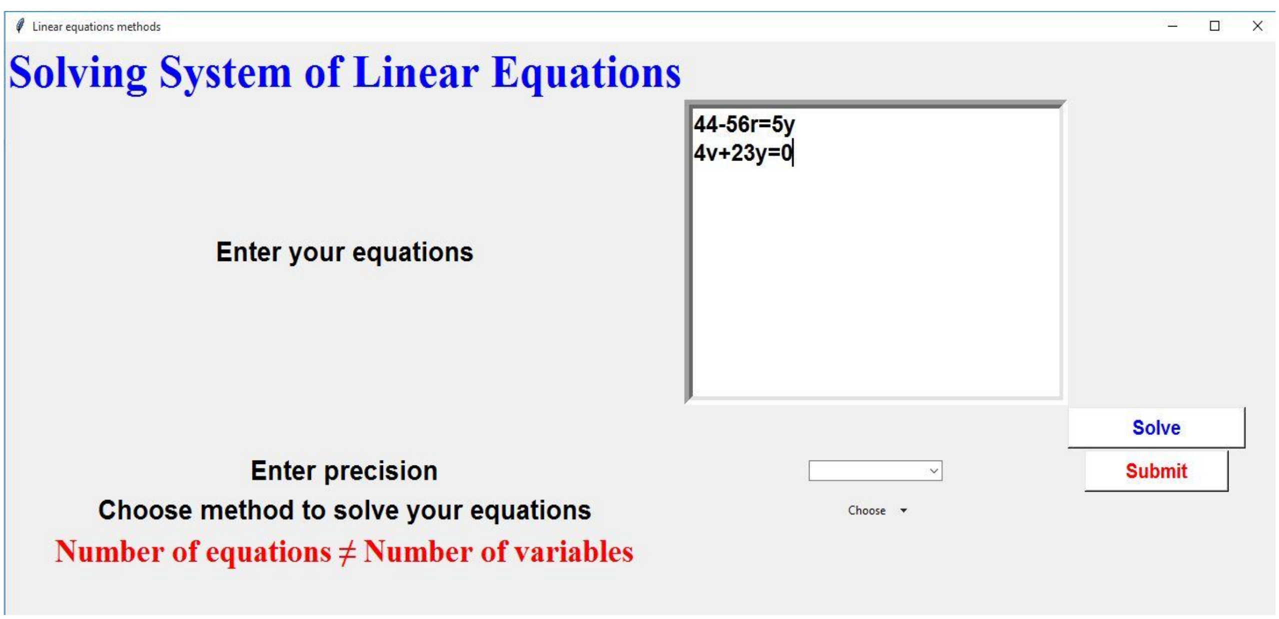
1. Gauss Elimination

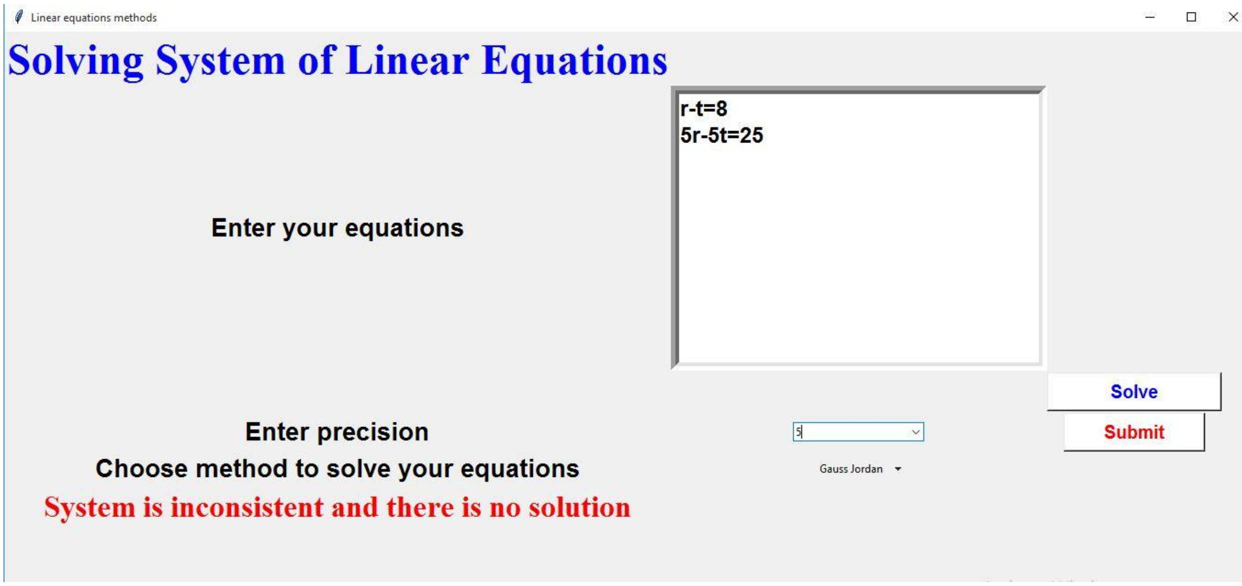


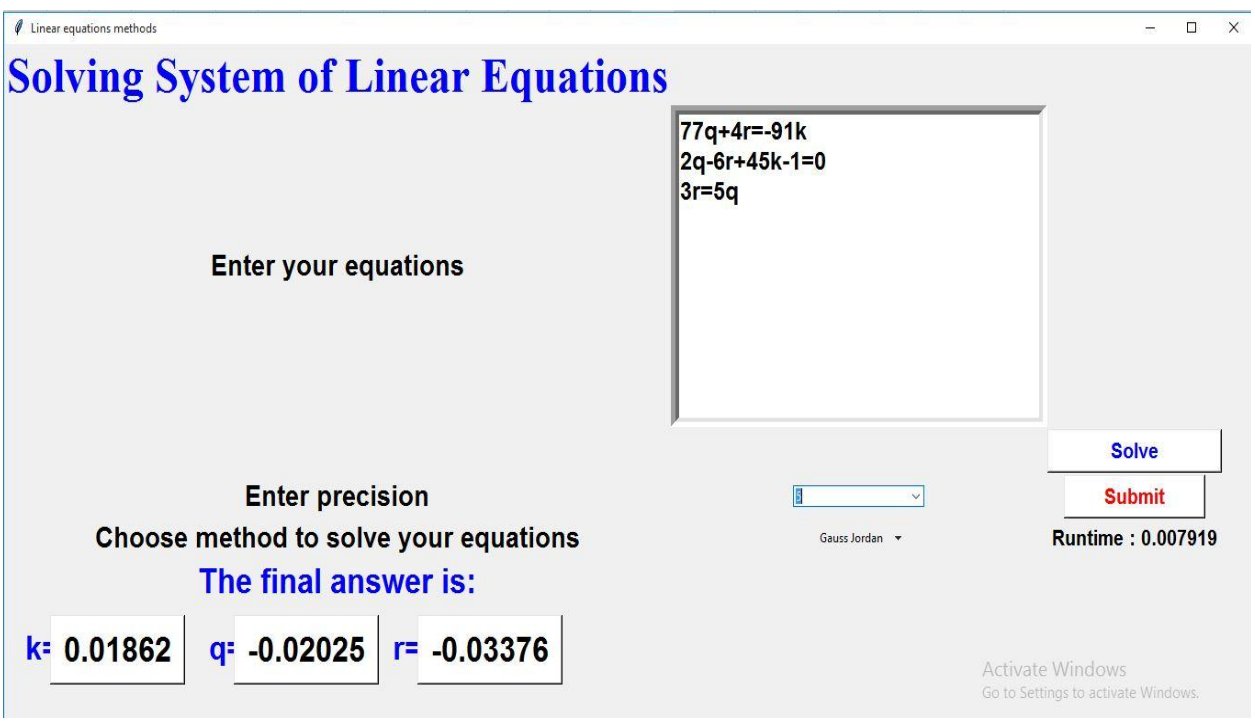


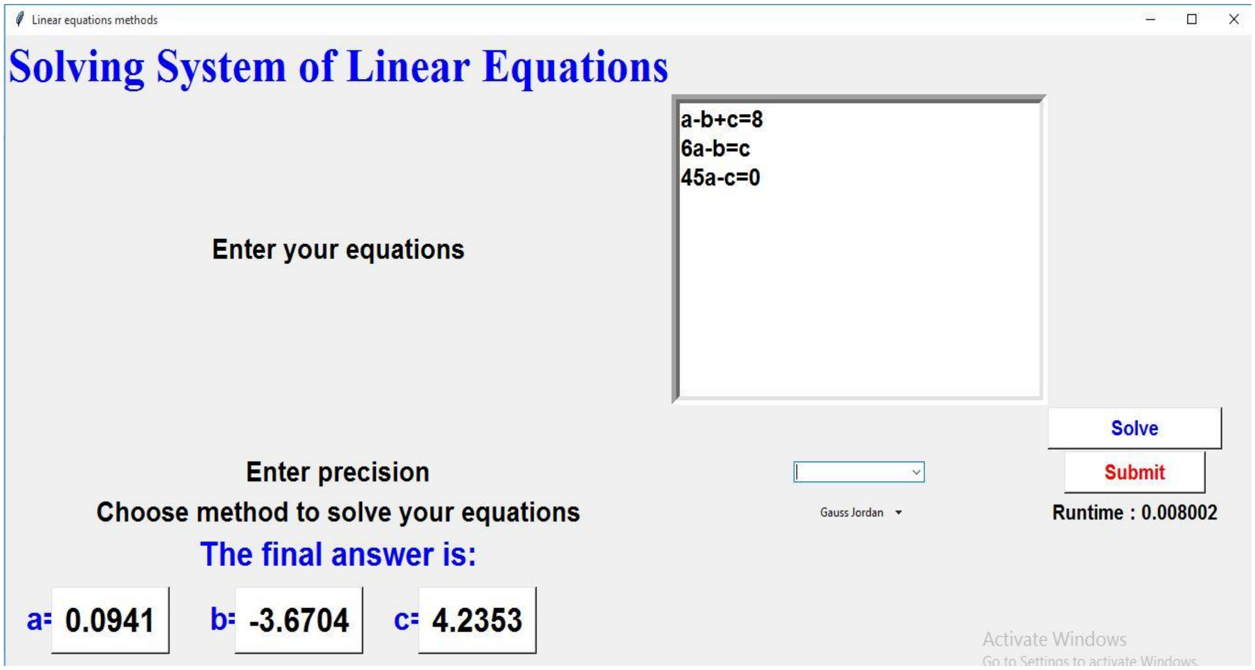


2. Gauss Jordan

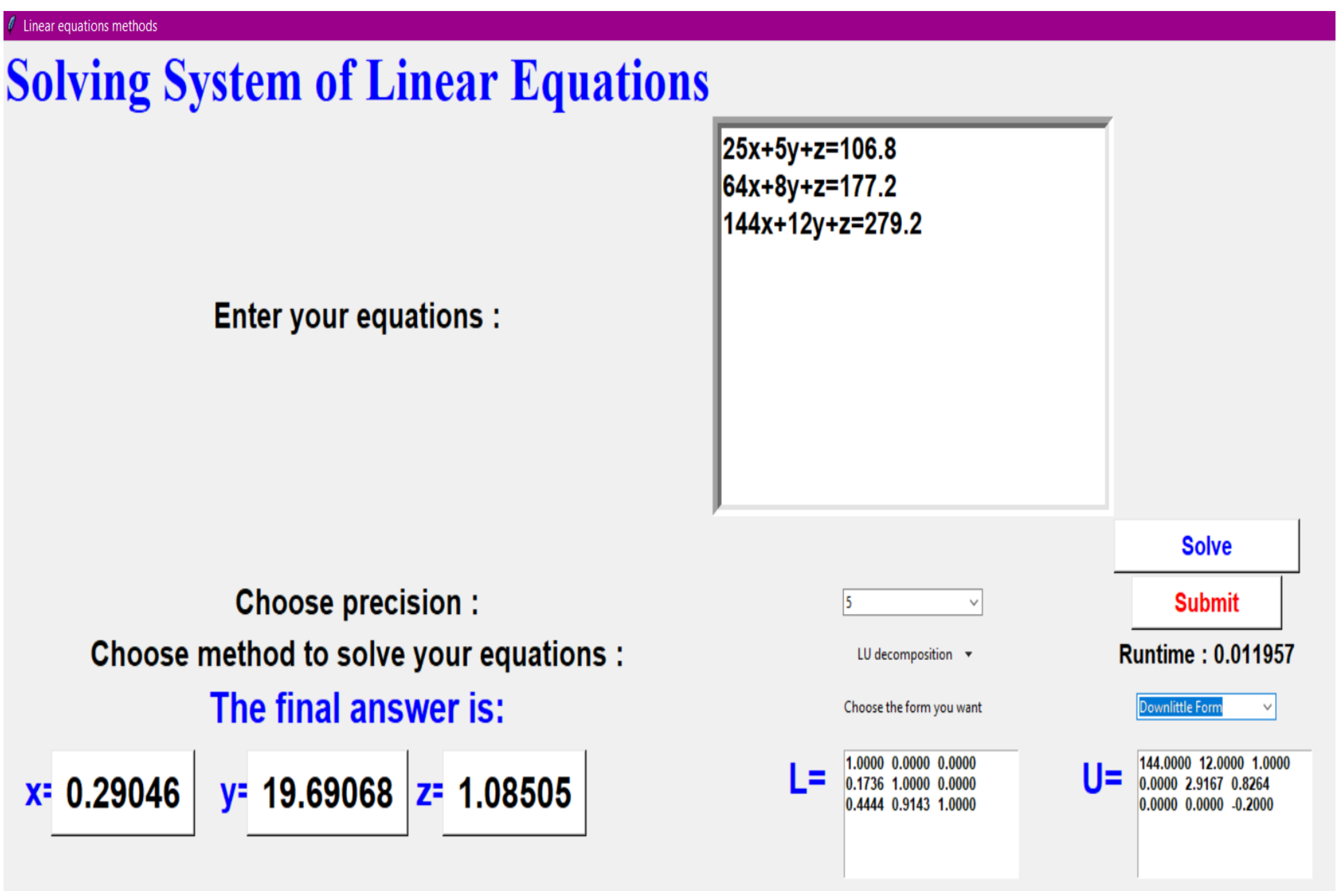


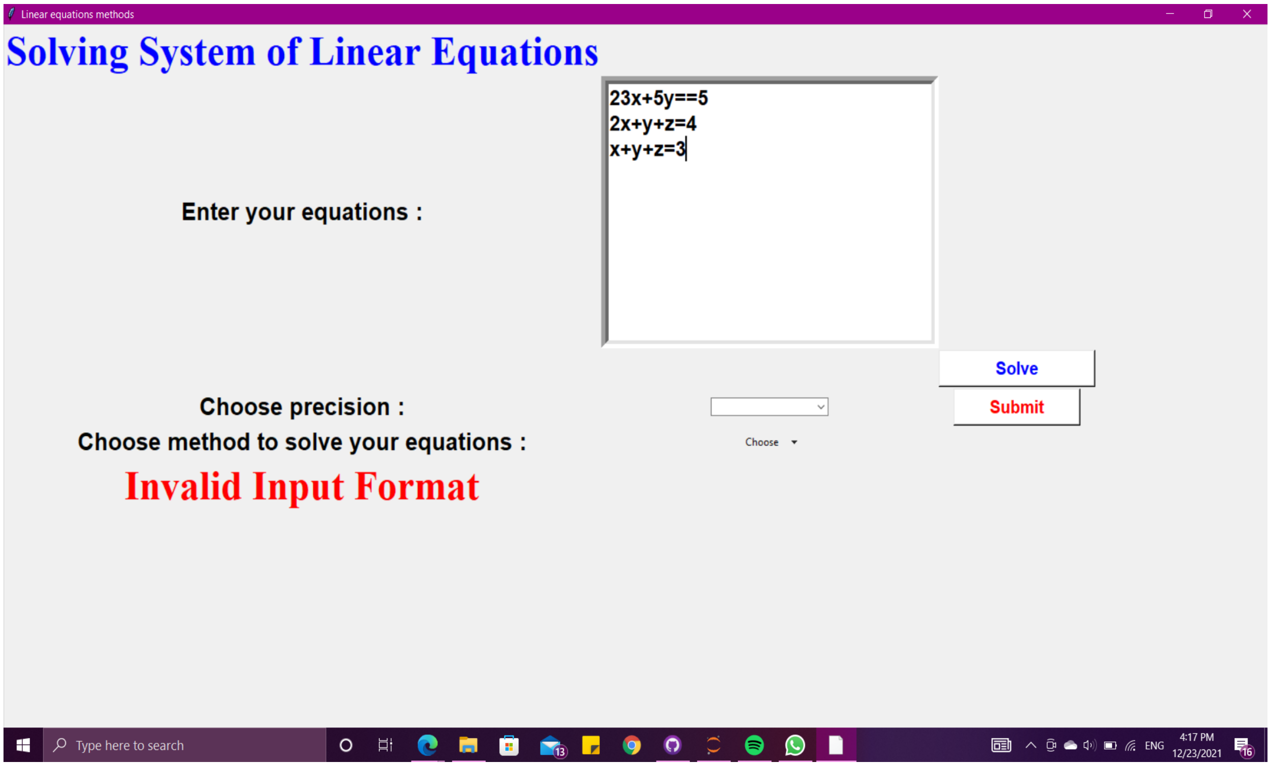

Linear equations methods
Solving System of Linear Equations
Enter your equations
r-t=8
5r-5t=25
Solve
Enter precision
5
Submit
Choose method to solve your equations
Gauss Jordan
System is inconsistent and there is no solution


Linear equations methods
Solving System of Linear Equations
Enter your equations
77q+4r=-91k
2q-6r+45k-1=0
3r=5q
Solve
Enter precision
Submit
Choose method to solve your equations
Gauss Jordan
Runtime : 0.007919
The final answer is:
k= 0.01862
q= -0.02025
r= -0.03376
Activate Windows
Go to Settings to activate Windows.

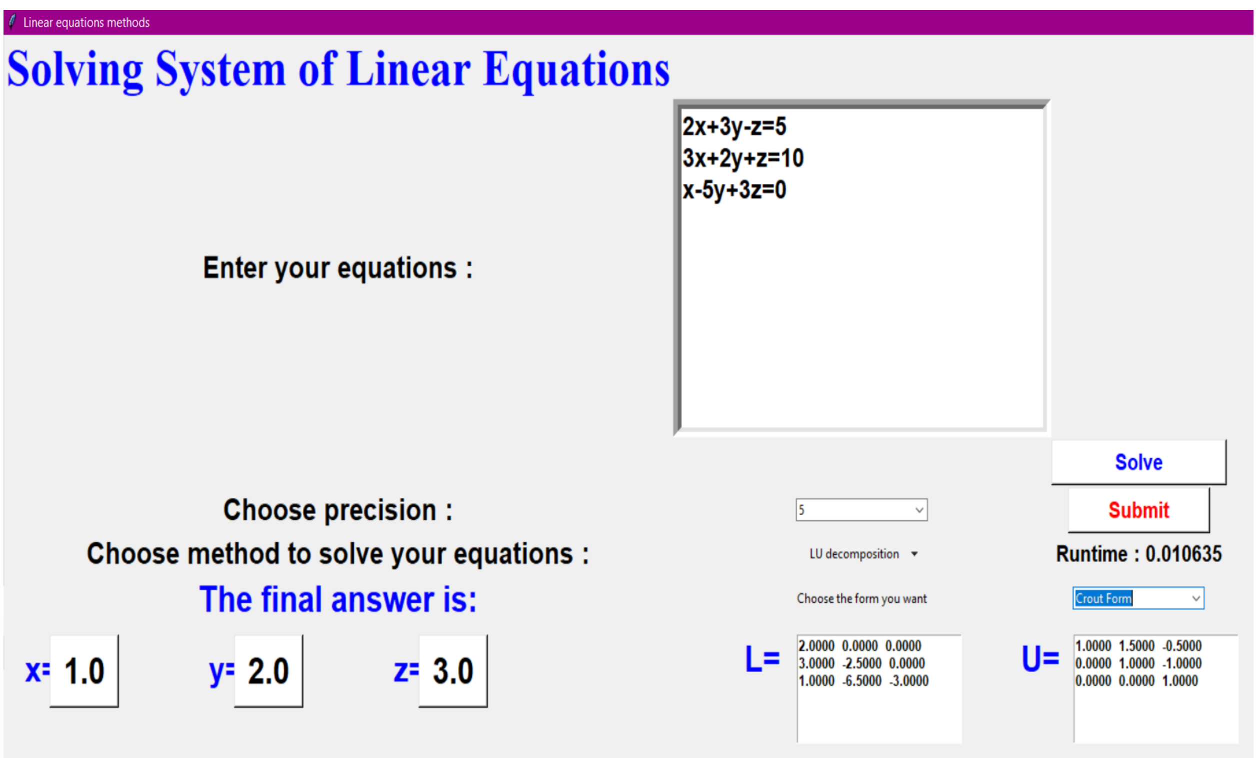


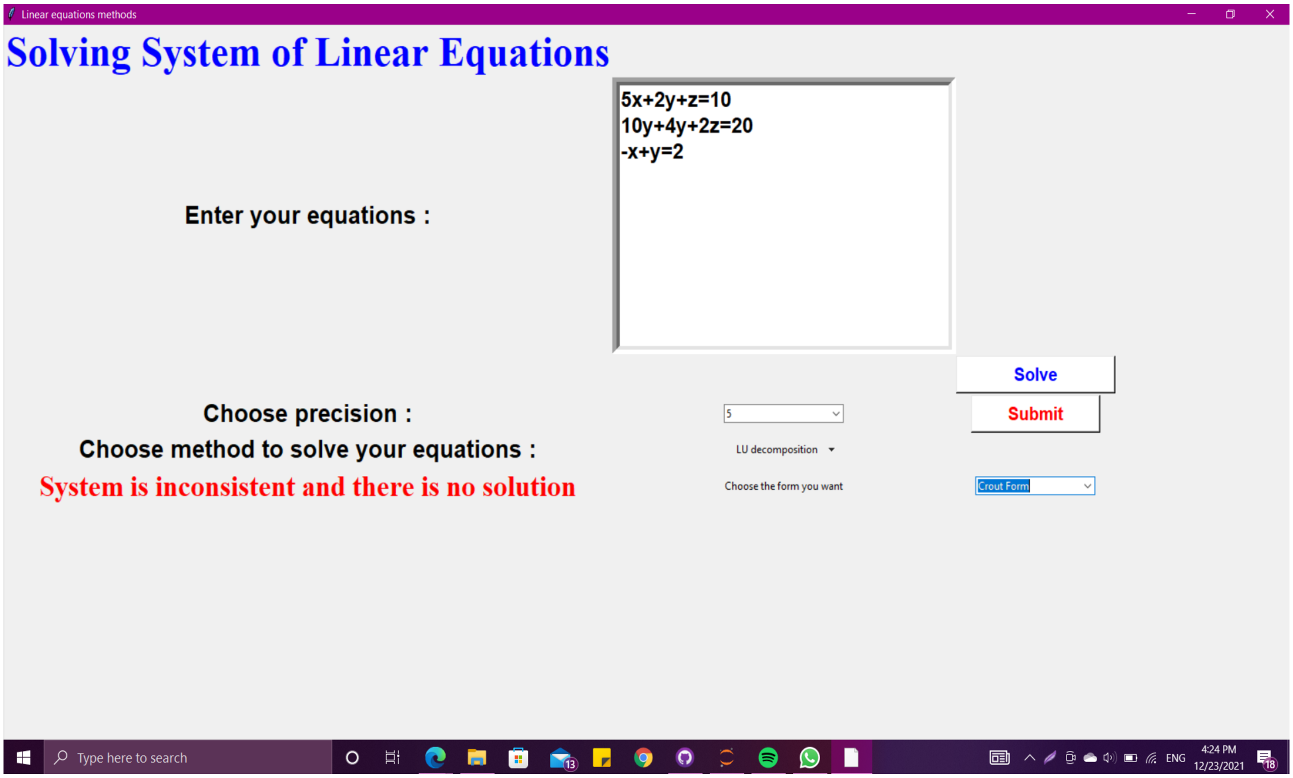
3. LU Decomposition
    a. Doolittle



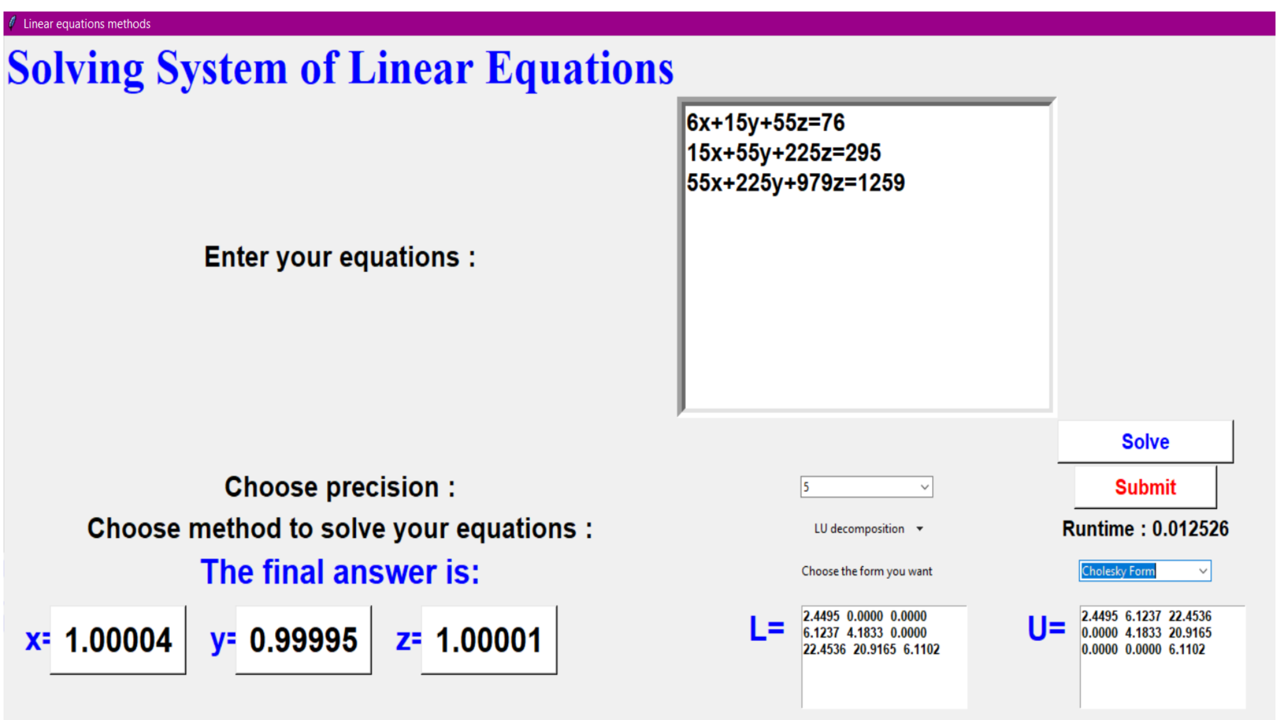

Linear equations methods
Solving System of Linear Equations
23x+5y==5
2x+y+z=4
x+y+z=3
Enter your equations :
Solve
Submit
Choose precision :
Choose method to solve your equations :
Choose
Invalid Input Format
Type here to search
ENG
4:17 PM
12/23/2021

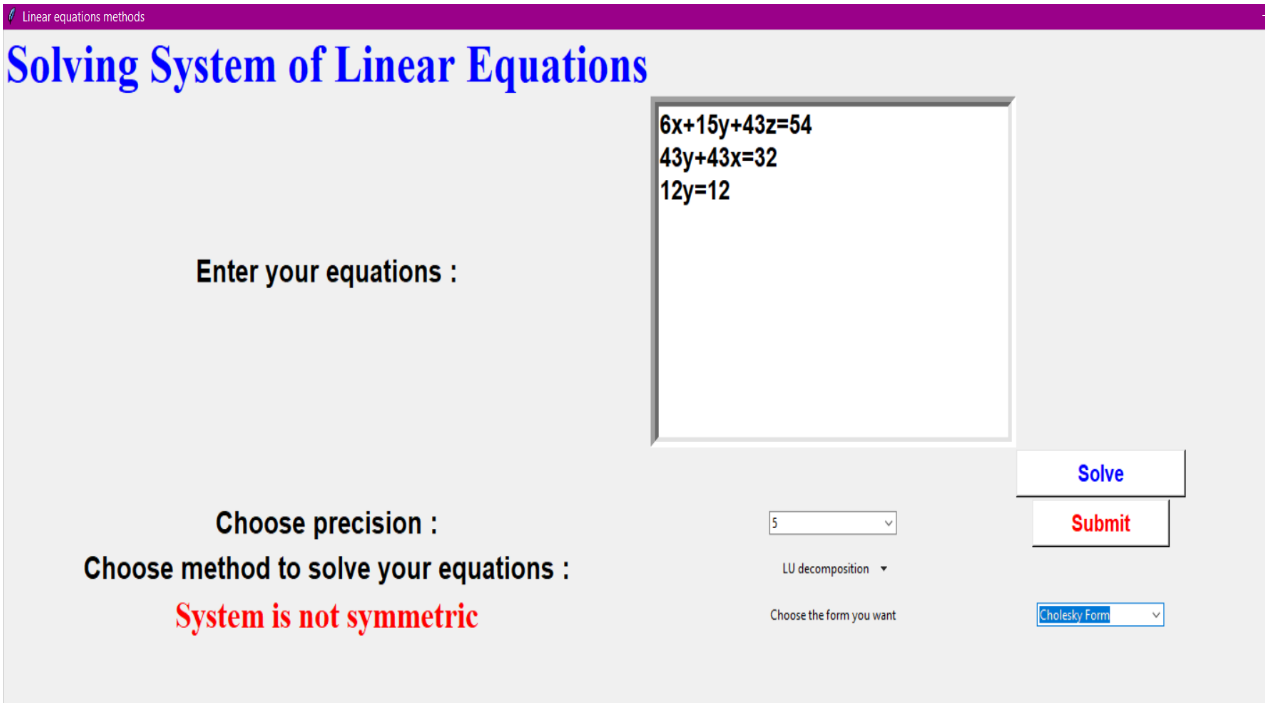
b. Crout


Linear equations methods
Solving System of Linear Equations
2x+3y-z=5
3x+2y+z=10
x-5y+3z=0
Enter your equations :
Solve
Submit
Choose precision :
5
Choose method to solve your equations :
LU decomposition
Runtime : 0.010635
The final answer is:
Choose the form you want
Crout Form
x= 1.0
y= 2.0
z= 3.0
L=
2.0000 0.0000 0.0000
3.0000 -2.5000 0.0000
1.0000 -6.5000 -3.0000
U=
1.0000 1.5000 -0.5000
0.0000 1.0000 -1.0000
0.0000 0.0000 1.0000

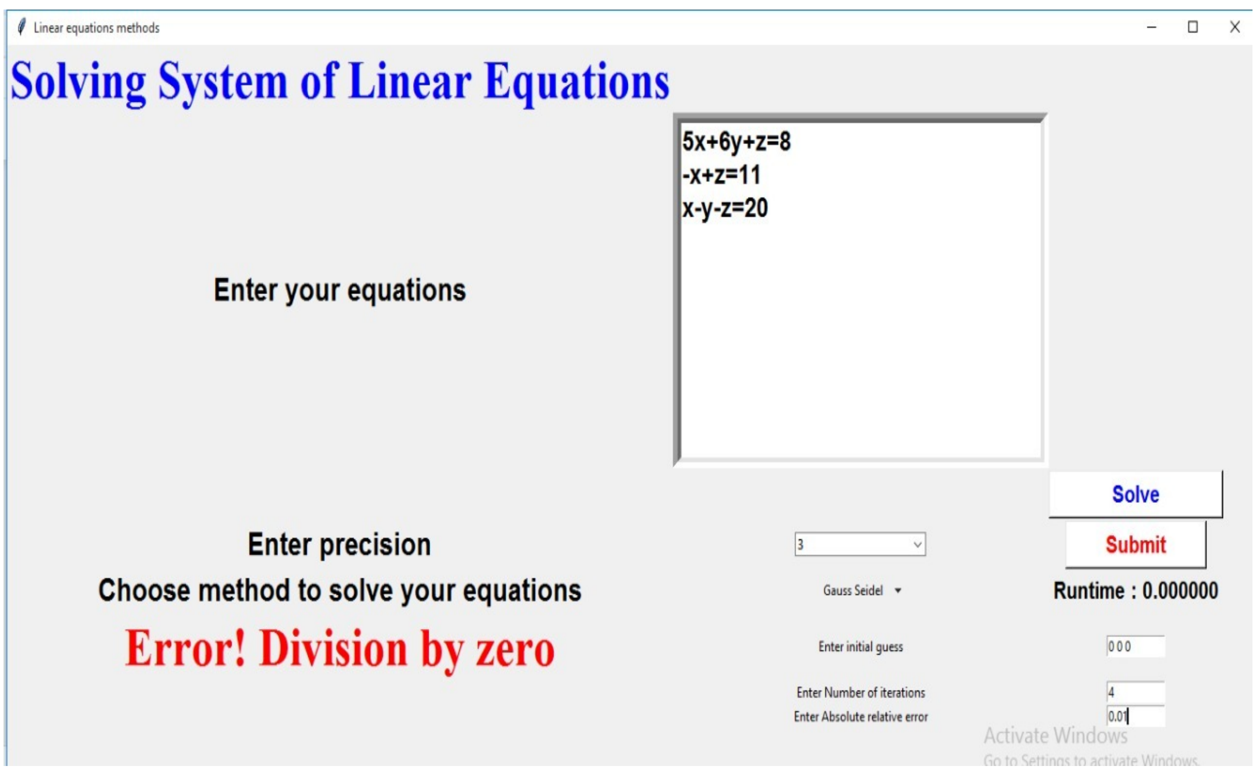

Linear equations methods
Solving System of Linear Equations
Enter your equations :
5x+2y+z=10
10y+4y+2z=20
-x+y=2
Solve
Choose precision :
5
Submit
Choose method to solve your equations :
LU decomposition
System is inconsistent and there is no solution
Choose the form you want
Crout Form
Type here to search
ENG
4:24 PM
12/23/2021

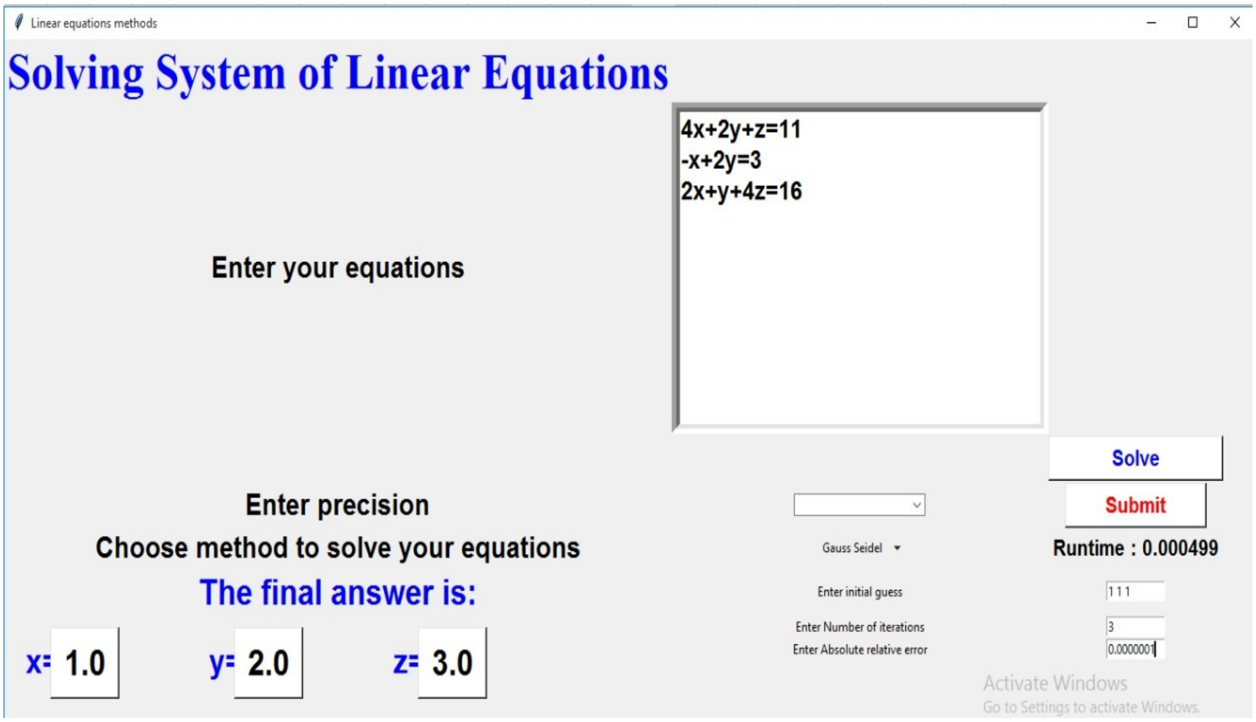
c. Cholesky


Linear equations methods
Solving System of Linear Equations
6x+15y+55z=76
15x+55y+225z=295
55x+225y+979z=1259
Enter your equations :
Solve
Choose precision :
5
Submit
Choose method to solve your equations :
LU decomposition
Runtime : 0.012526
The final answer is:
Choose the form you want
Cholesky Form
x= 1.00004
y= 0.99995
z= 1.00001
L=
2.4495 0.0000 0.0000
6.1237 4.1833 0.0000
22.4536 20.9165 6.1102
U=
2.4495 6.1237 22.4536
0.0000 4.1833 20.9165
0.0000 0.0000 6.1102

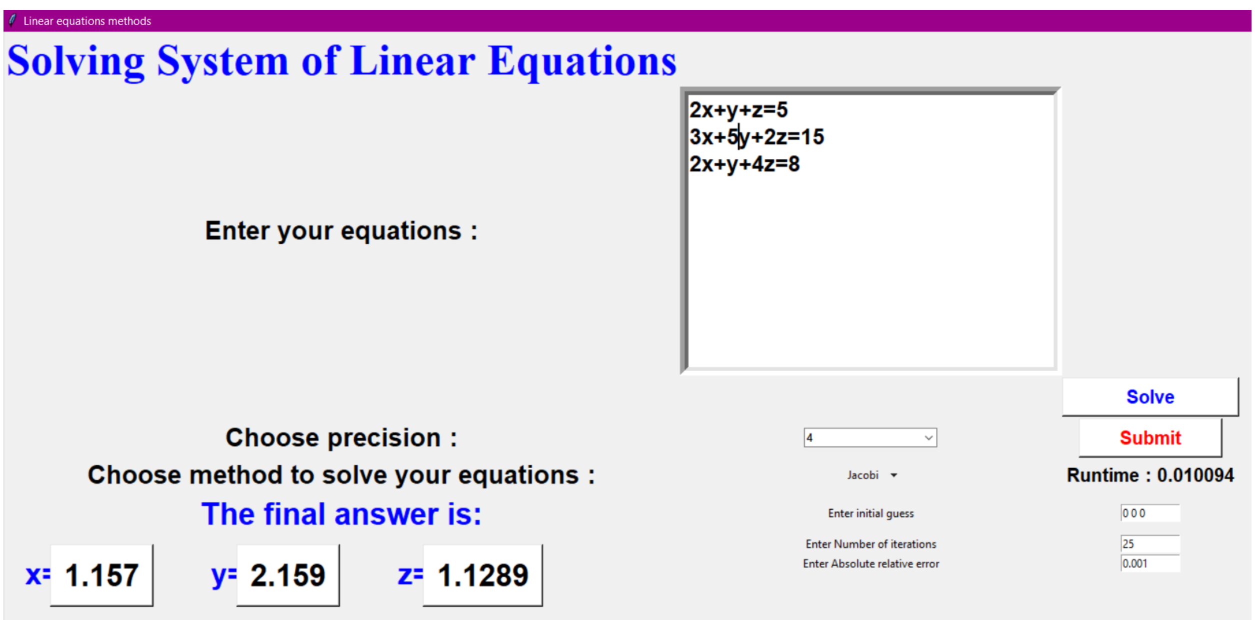


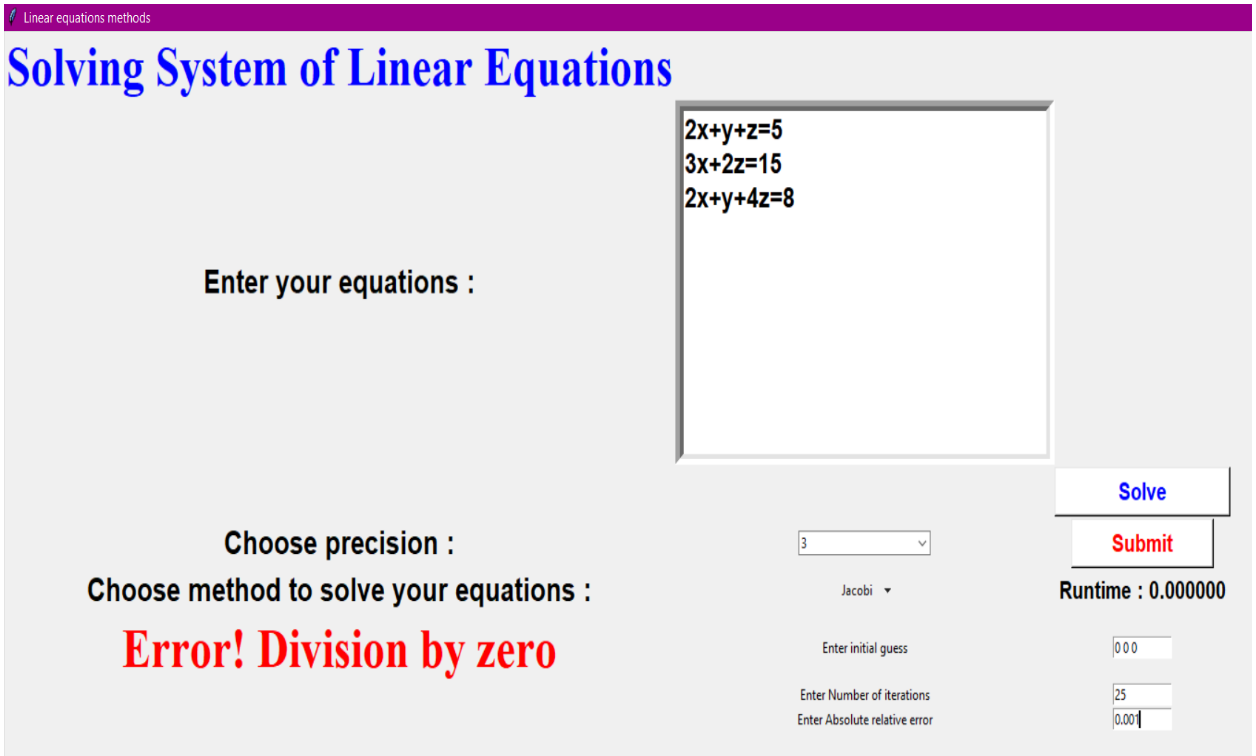
4. Gauss Seidel

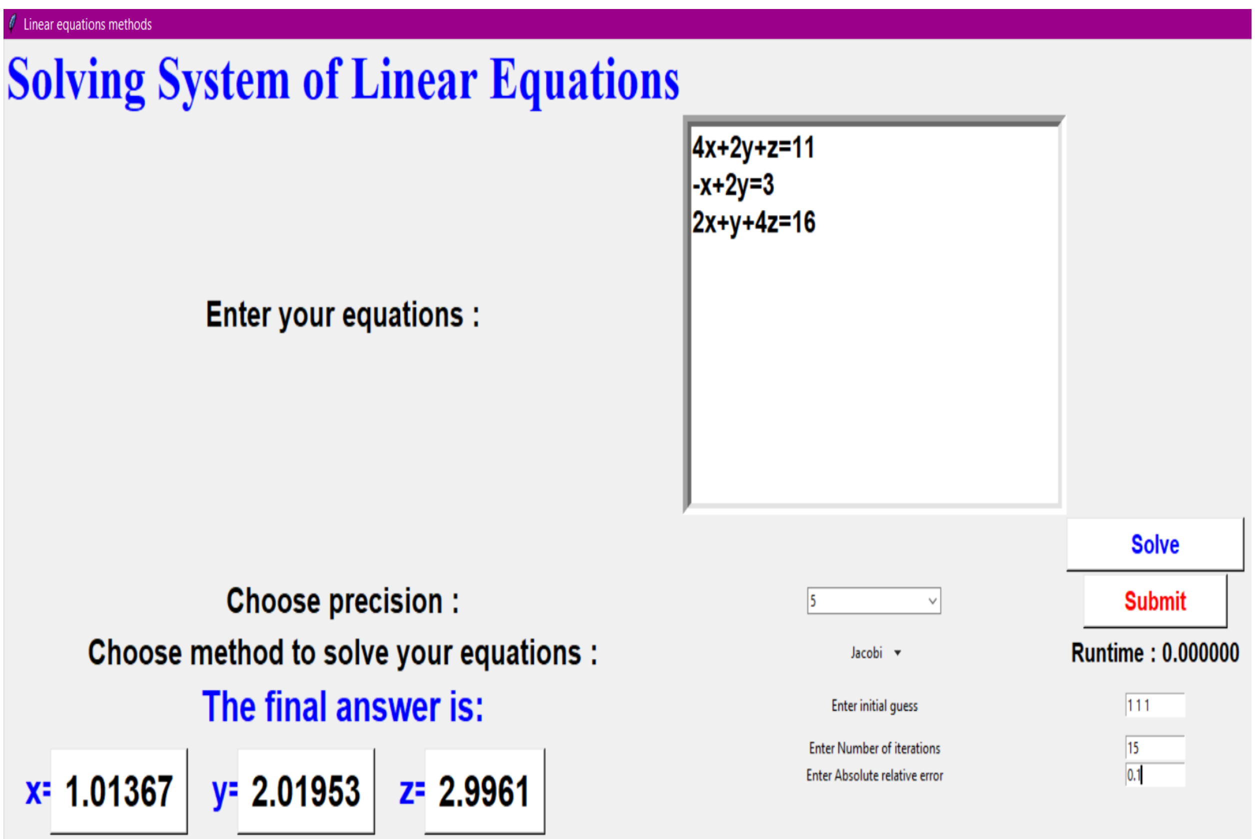




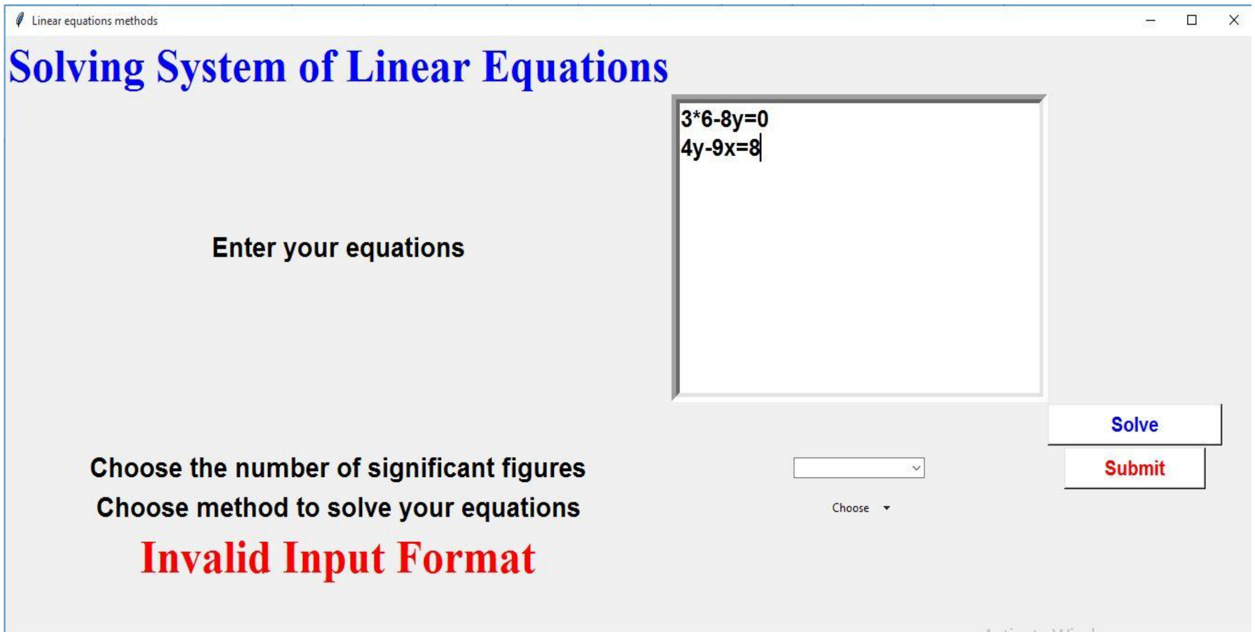
5. Jacobi Iteration




Linear equations methods
Solving System of Linear Equations
Enter your equations :
2x+y+z=5
3x+2z=15
2x+y+4z=8
Solve
Choose precision :
3
Submit
Choose method to solve your equations :
Jacobi
Runtime : 0.000000
Error! Division by zero
Enter initial guess
0 0 0
Enter Number of iterations
25
Enter Absolute relative error
0.001


Linear equations methods
Solving System of Linear Equations
Enter your equations :
4x+2y+z=11
-x+2y=3
2x+y+4z=16
Solve
Choose precision :
5
Submit
Choose method to solve your equations :
Jacobi
Runtime : 0.000000
The final answer is:
Enter initial guess
1 1 1
Enter Number of iterations
15
Enter Absolute relative error
0.1
x= 1.01367
y= 2.01953
z= 2.9961

6. Input Validation



Comparison between different methods

P.O.C	Time Complexity	Convergence	Best Case	Approximate error
Gauss Elimination	$O(n^3)$	Naive Gauss method will converge as it's a Direct Method with a constraint that the Coefficient Matrix is not Singular.	When the absolute of the pivot element is bigger than the absolute of the other elements under it , so it will do the same steps which are equal in complexity	A direct method so it calculates an exact solution for equations

Gauss-Jordan	$O(n^3)$	Gauss-Jordan method will converge as it's a Direct Method with a constraint that the Coefficient Matrix is not Singular.	it will always do the same steps for any matrix so all the cases are equal in complexity	a direct method so it calculates an exact solution for equations but there may be some errors due to round off errors in floating point arithmetic

LU Decomposition	$O(n^3)+2*O(n^2)$	LU Decomposition with All its forms will converge as it's a Direct Method with a constraint that the Coefficient Matrix is not Singular. There is also a constraint in Cholesky decomposition for symmetric matrices that the matrix is positive definite.	a Direct Method and it will always do the same steps for any matrix so all the cases are equal in complexity	an exact solution as it's a direct method but there may be some errors due to round off errors in floating point arithmetic

Gauss-Seidel	$O(n^2)$	Iterations are repeated until the convergence criterion is satisfied. As any other iterative method, the Gauss-Seidel method has problems. It may not converge or it converges very slowly. If the coefficient matrix A is Diagonally Dominant Gauss-Seidel is guaranteed to converge.	If the solution is converging & the coefficient matrix A is diagonally dominant, the best available estimates will be employed	More iteration will give more precision so the approximate errors will decreased

Jacobi-Iteration	O (n^2)	Iterations are repeated until the convergence criterion is satisfied. It may not converge or it converges very slowly. If the coefficient matrix A is diagonally dominant, Jacobi is guaranteed to converge.	if the solution is converging, the best available estimates will be employed	More iteration will give more precision so the approximate errors will decreased

Data structures used

- Arrays, Lists: used for storing the matrix form of the equations and results.
- Sets : Used while filtering the variables from equations to avoid repetition.