Project Phase 1

Team members:

- 21010528 رنا محمد علي عطية
- 21010298 أمنية طارق ابراهيم السيد
- 21011492 نوران أشرف يوسف
- 21010547 روان محمد سعيد عبد الفتاح

Pseudo-code for each method:

- ### Gauss Elimination

These are the pseudo-codes of Gauss Elimination functions with scaling, there exists another functions without scaling which are nearly the same (in implementation) as those which use scaling.

- **procedure** solve_with_scaling

```
    if rankA == rankAaugB and rankA == n  then
      Initialize S according to the input array size (n)
      //S is an n-element array for storing scaling factors
      //S[i] is the largest coefficient of row i
     for i=0 to n-1 do
       S[i] ← abs(A[i][0])
       for j=1 to n-1 do
         S[i] ← max(abs(A[i][j]), S[i])
       end for
      end for

      //A is the coefficient matrix of X (2-d array)
      //B is a (1-d array) where AX = B
     ForwardElimination_with_scaling(A, B, n, S)
     out ← BackSubstitution(A, B, n)

    else if rankA != rankAaugB then
     out ← "This linear system of equations has no solution."
    else
     out ← "This linear system of equations has infinite number of solutions."
    end if
    return out
end procedure
```

- **procedure** Pivot_with_scaling (A, B, S, n, k):

```
p = k

//Detecting the largest scaled number (pivot) in column k
big ← abs(A[k][k] / S[k])
big ← precision(big)
for i=0 to n-1 do
   dummy ← abs(A[i][k] / S[i])
    //precision function is used to round the numbers
   dummy ← precision(dummy)
   if dummy > big then
      big = dummy
      p = i
   end if
 end for

//If a new pivot is detected, swap the rows
if p != k then
   for j=0 to n-1 do
      dummy ← A[p][j]
      A [p][j] ← A[k][j]
      A[k][j] ← dummy
    end for

   //Swapping the values in B
   dummy ← B[p]
   B[p] ← B[k]
   B[k] ← dummy

   //Swapping the values in S
   dummy ← S[p]
   S[p] ← S[k]
   S[k] ← dummy
```

end procedure

- **procedure** ForwardElimination_with_scaling (A, B, n, S):

```
for k=0 to n-2 do
    Pivot_with_scaling(A, B, S, n, k)
    for i=k+1 to n-1 do
        factor ← A[i][k] / A[k][k]
        factor ← precision(factor)
        for j=k+1 to n-1 do
            A[i][j] ← A[i][j] - factor * A[k][j]
            A[i][j] ←precision(A[i][j])
        end for
        B[i] ←B[i] - factor * B[k]
        B[i] ← precision(B[i])
    end for
end for
```

end procedure

- **procedure** BackSubstitution (A, B, n):

```
Initialize X according to n
X[n-1] ← B[n-1] / A[n-1][n-1]
X[n-1] ← precision(X[n-1])
for i=n-1 to 0 do
    sum ← 0
    for j=i+1 to n-1 do
        sum ← sum + A[i][j] * X[j]
        sum ← precision(sum)
    end for
    X[i] ←  (B[i] - sum) / A[i][i]
    X[i] ← precision(X[i])
end for
return X
```

end procedure

Gauss Jordan

These are the pseudo-codes of Gauss Elimination functions with scaling, there exists another functions without scaling which are nearly the same (in implementation) as those which use scaling.

- **procedure** solve_with_scaling

```
if rankA == rankAaugB and rankA == n then
     Initialize S according to the input array size
     //S is an n-element array for storing scaling factors
     //S[i] is the largest coefficient of row i
    for i=0 to n-1 do
       S[i] = abs(A[i][0])
       for j=1 to n-1 do
          S[i] = max(abs(A[i][j]), S[i])
       end for
     end for

    ForwardElimination_with_scaling(A, B, n, S)//Same as in Gauss Elimination
    BackElimination(A, B, n)

    for k=0 to n-1 do
       B[k] ← B[k] / A[k][k]
       B[k] ← precision(B[k])
     end for
     out = B

else if rankA != rankAaugB then
    out = "This linear system of equations has no solution."
else
    out = "This linear system of equations has infinite number of solutions."
end if
return out
end procedure
```

- **procedure** BackElimination (A, B, n):

```
for k=n-1 to 0 do
    for i=k-1 to 0 do
       factor ← A[i][k] / A[k][k]
       factor ← precision(factor)
       B[i] ← B[i] - factor * B[k]
       B[i] ← precision(B[i])
     end for
 end for
end procedure
```

➢ LU Decomposition

• Doolittle:

Algorithm 1 Doolittle LU Decomposition with Pivoting

procedure DECOMPOSE
 $n \leftarrow$ size of A
 Initialize matrices P, L, and U based on A
 for $k = 1$ to $n-1$ **do**
  $pivot \leftarrow U[k,k]$
  if $|pivot| < 1e-10$ **then**
   Find the index $nonzero\_row$ of the first nonzero pivot in
   the current column below the current row
   Swap rows in U, L, and P to pivot
   $pivot \leftarrow U[k,k]$ ▷ Update pivot after pivoting
  end if
  for $i = k+1$ to n **do**
   $factor \leftarrow precision\left(\frac{U[i,k]}{pivot}\right)$
   $L[i,k] \leftarrow factor$
   $U[i,k:] \leftarrow U[i,k:] - factor \cdot U[k,k:]$
   for $j = k$ to n **do**
    $U[i,j] \leftarrow precision(U[i,j])$
   end for
  end for
 end for
 return Solve linear system using P, L, and U and vector b
end procedure

• Cholesky:

Algorithm 2 Cholesky Decomposition

procedure CHOLESKY_DECOMPOSITION(b)
 $n \leftarrow$ size of A
 Initialize matrices L, U, P, and variables for validity and error message
 Check the solvability of A and mark the decomposition as
 invalid if the matrix is not symmetric positive definite
 if Decomposition is valid **then**
  for $i = 1$ to n **do**
   for $j = 1$ to i **do**
    if $i = j$ **then**
     $sum\_val \leftarrow \text{precision}\left(\sum_{k=1}^{j} L[i,k]^2\right)$
     $L[i,i] \leftarrow \text{precision}\sqrt{A[i,i] - sum\_val}$
    else
     $sum\_val \leftarrow \text{precision}\left(\sum_{k=1}^{j} L[i,k] \cdot L[j,k]\right)$
     $L[i,j] \leftarrow \text{precision}\frac{A[i,j] - sum\_val}{L[j,j]}$
    end if
   end for
  end for
  $U \leftarrow$ transpose of L
  return Solve linear system using P, L, and U and input vector b
 else
  return Error message
 end if
end procedure

- **Crout:**

Algorithm 3 Crout LU Decomposition

procedure CroutDecompose(A, fig)
    Initialize matrices L, U, P, and variables for failure detection
    and error message
    for $j = 1$ **to** n **do**
        for $i = j$ **to** n **do**
            $L[i,j] \leftarrow precision\left(A[i,j] - \sum_{k=1}^{j-1} L[i,k] \cdot U[k,j], fig\right)$
            if $|L[j,j]| < 1e - 10$ **then**
                $err\_msg \leftarrow$ Division by zero encountered. Method failed.
                $isFailed \leftarrow True$
            else
                $U[j,i] \leftarrow precision\left(\frac{A[j,i] - \sum_{k=1}^{j-1} L[j,k] \cdot U[k,i]}{L[j,j]}, fig\right)$
            end if
        end for
    end for
    if $isFailed = False$ **then**
        return Solve linear system using P, L, and U and input vector b
    else
        return err_msg
    end if
end procedure

- **Gauss Seidel**
  - procedure (array, initial, iteration, error):

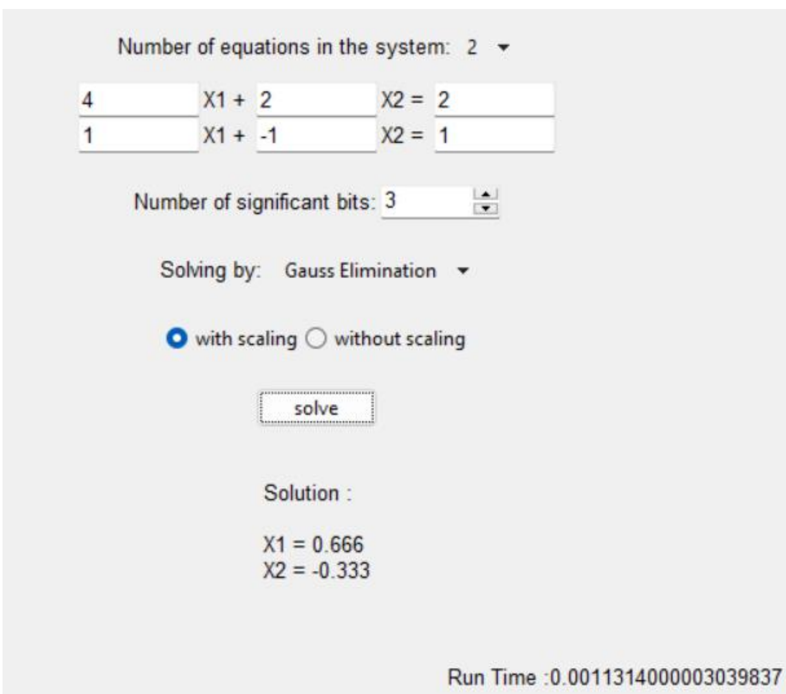
```
iterate <- 0
while iterate < iteration  and error  > ek ( k = 1 to n)
   for i = 1 to n
      x <- arrayi, n
      for j = 1 to n
         if j does not equal  i
            x <- x – arrayi, j * initialj
         end if
      end for
      x <- x / arrayi, i
      ei <- (x – intiali )/ x * 100
      initiali <- x
   end for
   iterate <- iterate + 1
end while
       end procedure
```

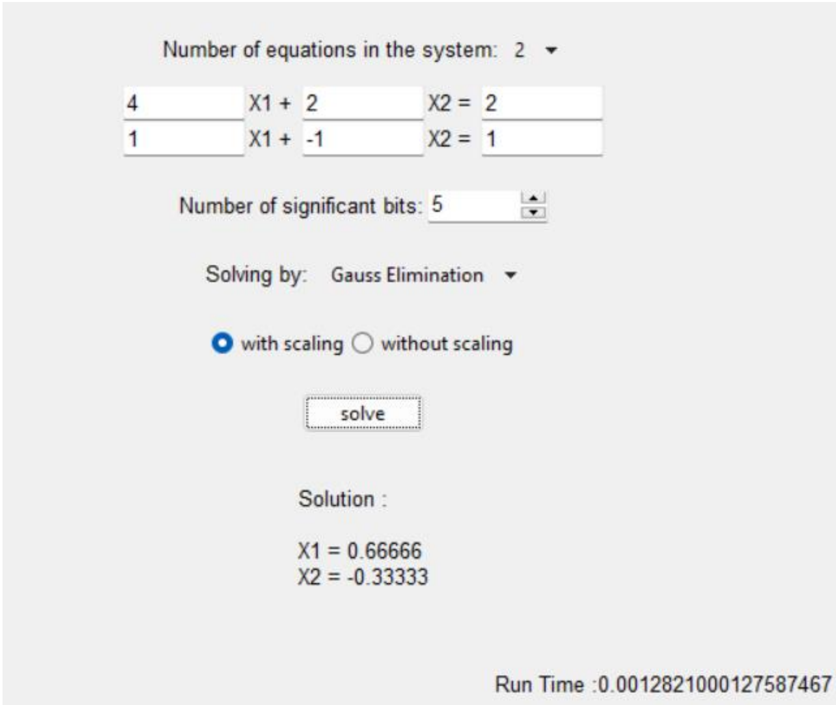
- **Jacobi Iteration**
  - procedure (array, initial, iteration, error):

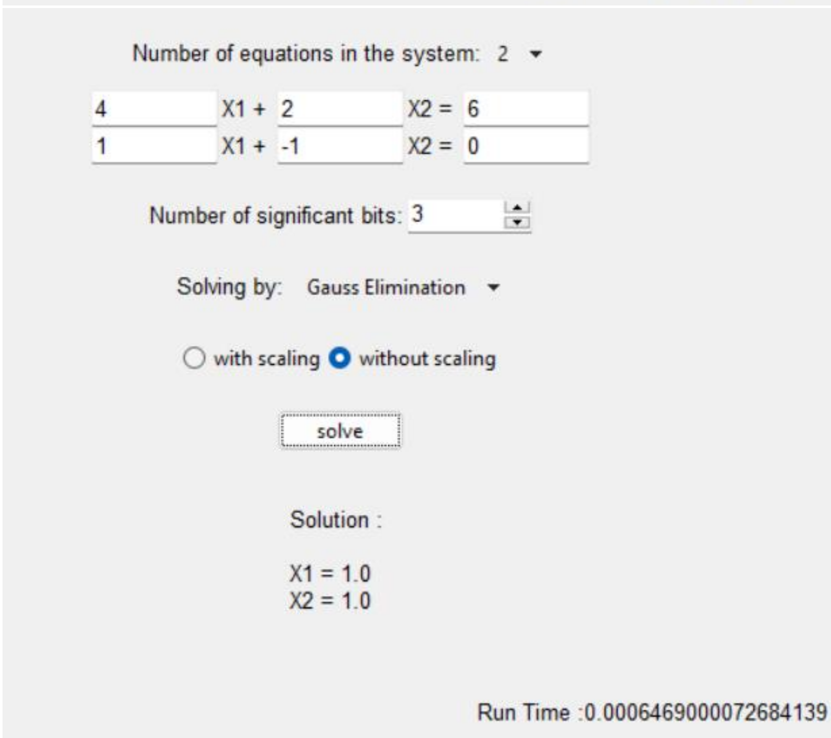
```
iterate <- 0
while iterate < iteration and error  > e^k ( k = 1 to n)
   for i = 1 to n
      x <- array^(i, n)
      for j = 1 to n
         if j does not equal  i
            x <- x – array^(i, j) * initial^j
         end if
      end for
      x <- x / array^(i, i)
      e^i <- (x – intial^i )/ x * 100
      temp^i <- x
   end for
   for k = 1 to n
      intial^k <- temp^k
   iterate <- iterate + 1
end while
end procedure
```

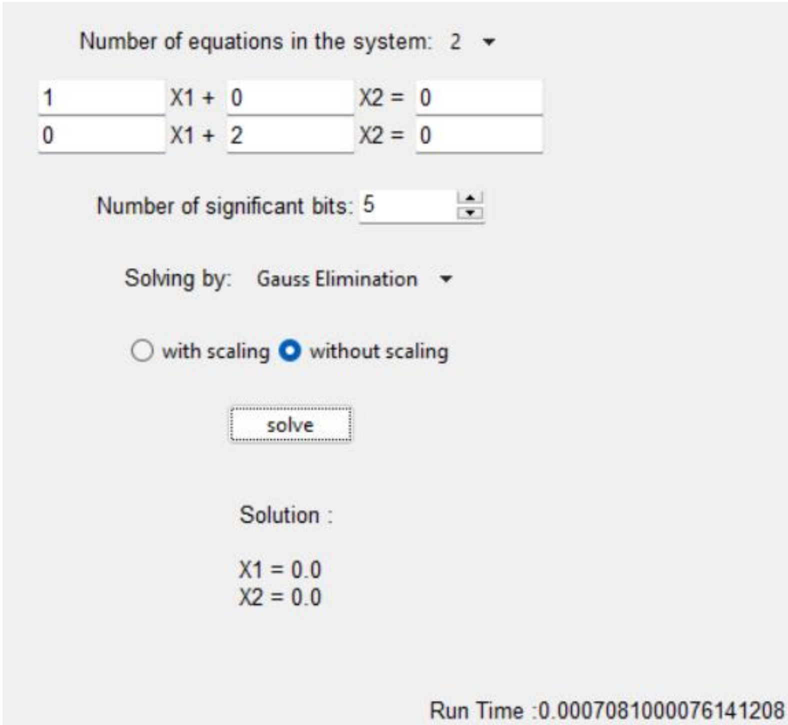
Sample runs for each method:

- **Gauss Elimination**

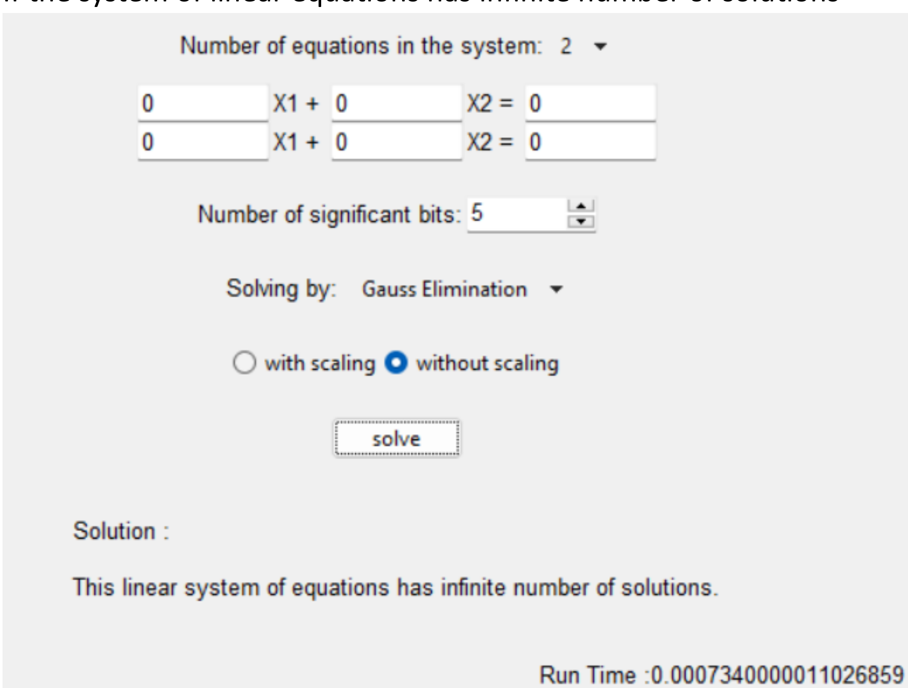



Number of equations in the system: 2
4 X1 + 2 X2 = 2
1 X1 + -1 X2 = 1
Number of significant bits: 5
Solving by: Gauss Elimination
with scaling
without scaling
solve
Solution :
X1 = 0.66666
X2 = -0.33333
Run Time :0.0012821000127587467

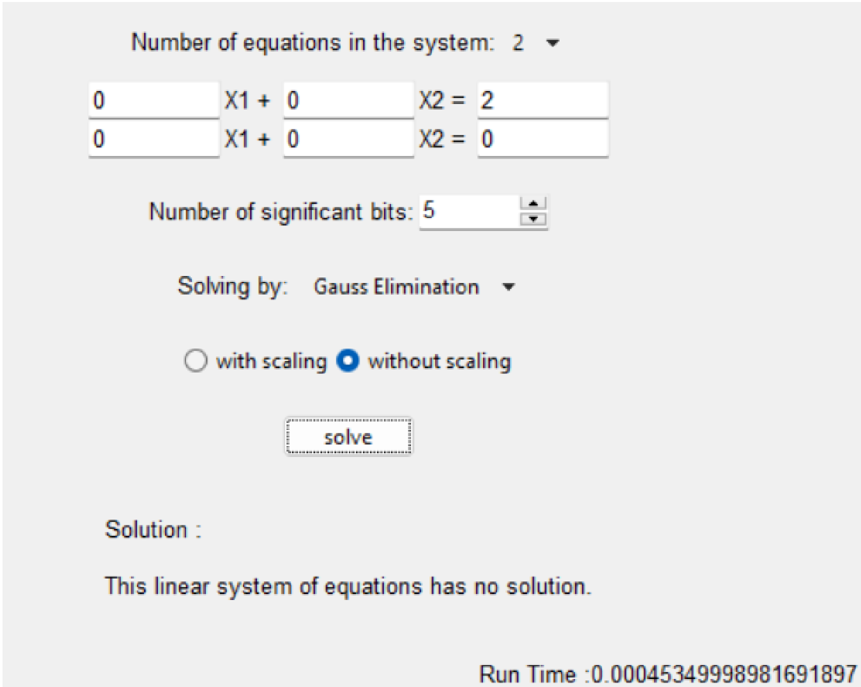

Number of equations in the system: 2
4 X1 + 2 X2 = 6
1 X1 + -1 X2 = 0
Number of significant bits: 3
Solving by: Gauss Elimination
with scaling
without scaling
solve
Solution :
X1 = 1.0
X2 = 1.0
Run Time :0.0006469000072684139



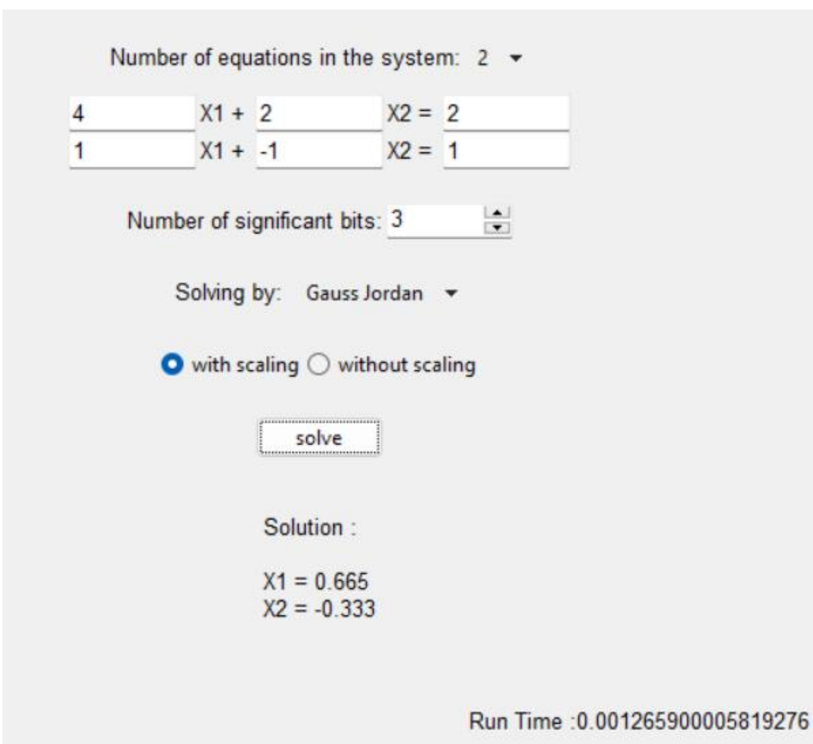
If the system of linear equations has infinite number of solutions

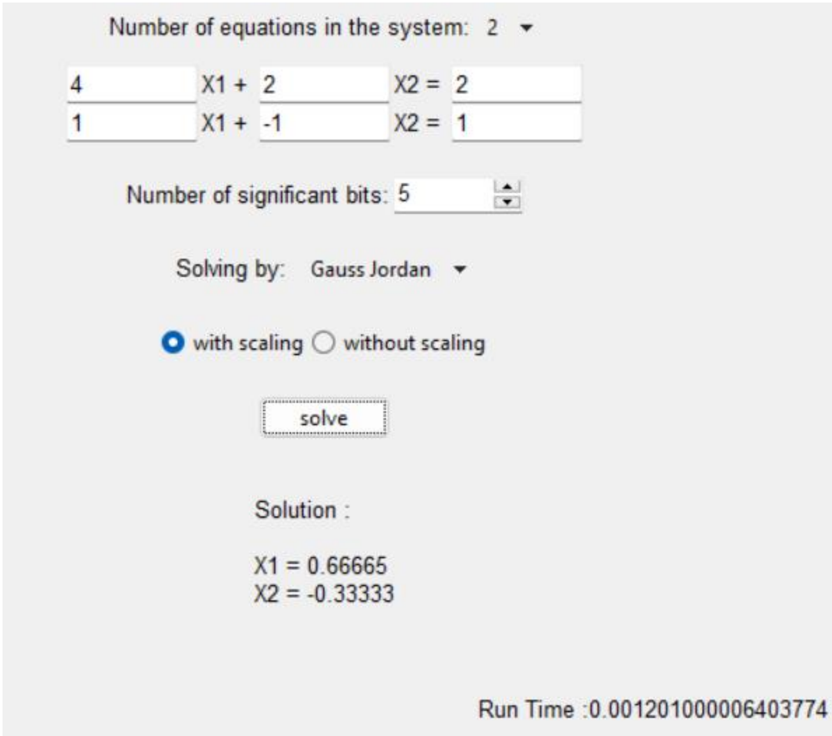


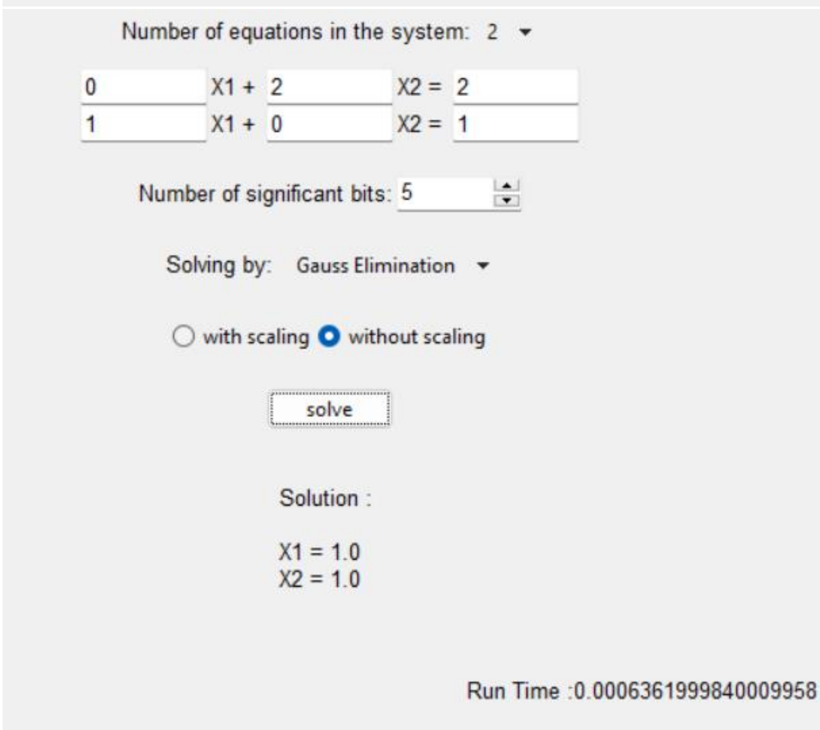
If the system of linear equations has no solution



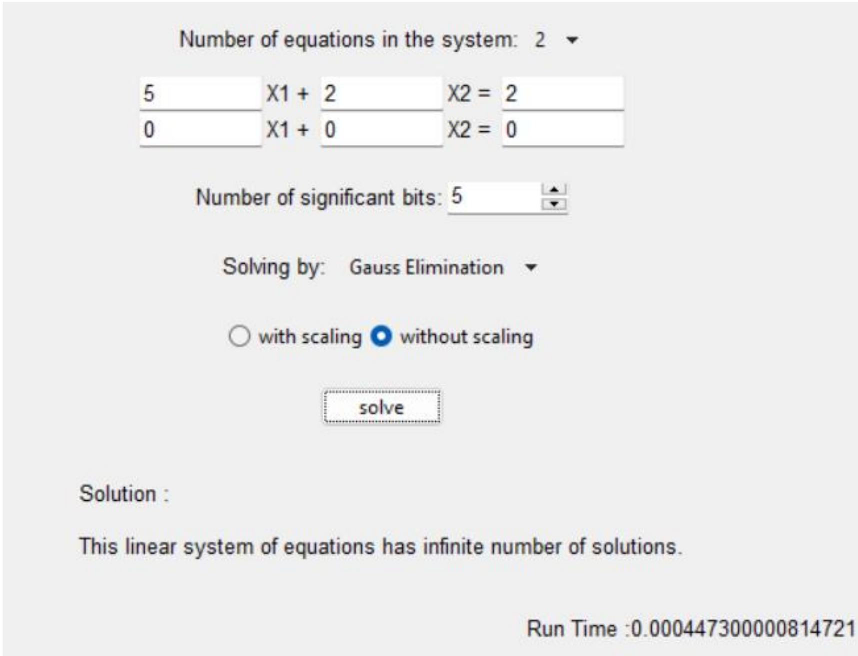
- **Gauss Jordan**



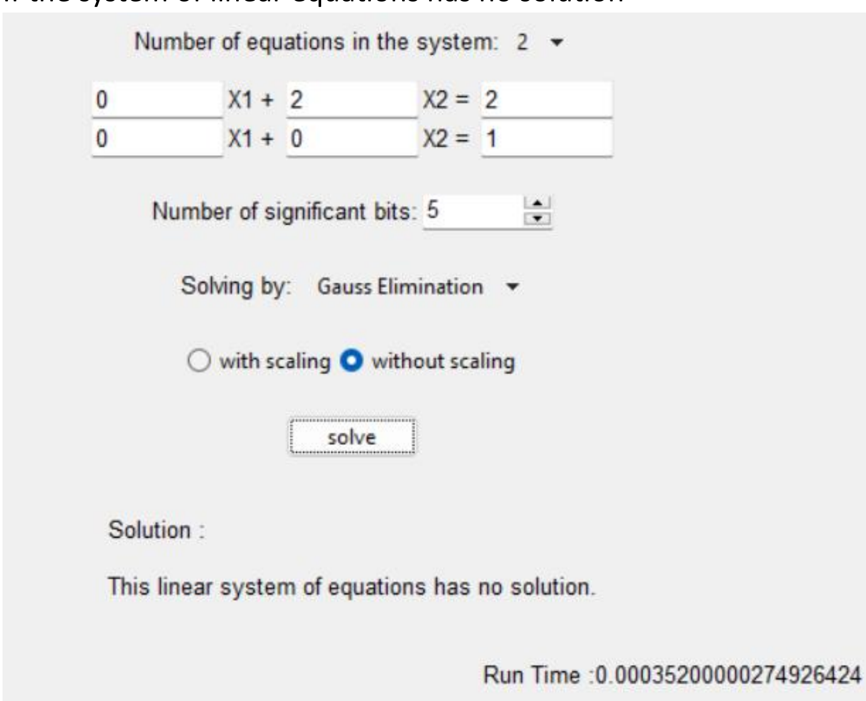

Number of equations in the system: 2
4 X1 + 2 X2 = 2
1 X1 + -1 X2 = 1
Number of significant bits: 5
Solving by: Gauss Jordan
with scaling
without scaling
solve
Solution :
X1 = 0.66665
X2 = -0.33333
Run Time :0.001201000006403774


Number of equations in the system: 2
0 X1 + 2 X2 = 2
1 X1 + 0 X2 = 1
Number of significant bits: 5
Solving by: Gauss Elimination
with scaling
without scaling
solve
Solution :
X1 = 1.0
X2 = 1.0
Run Time :0.0006361999840009958

If the system of linear equations has infinite number of solutions

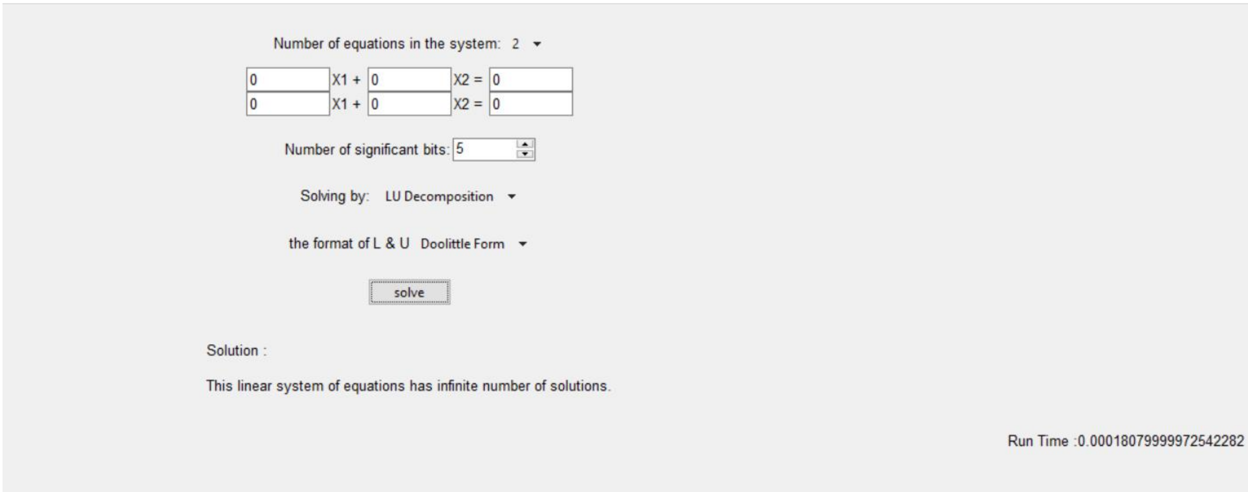


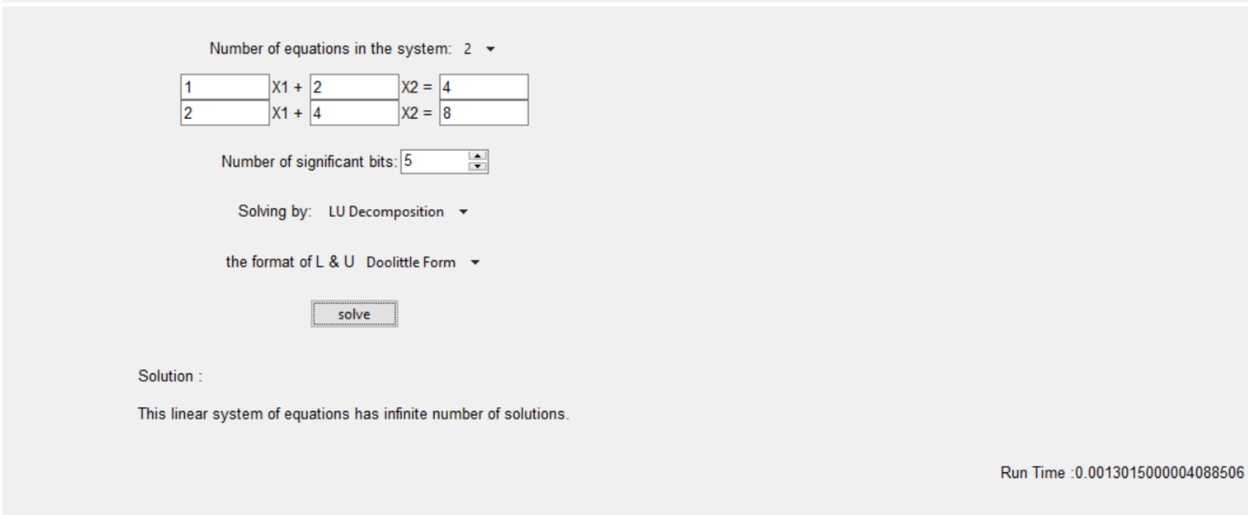
If the system of linear equations has no solution

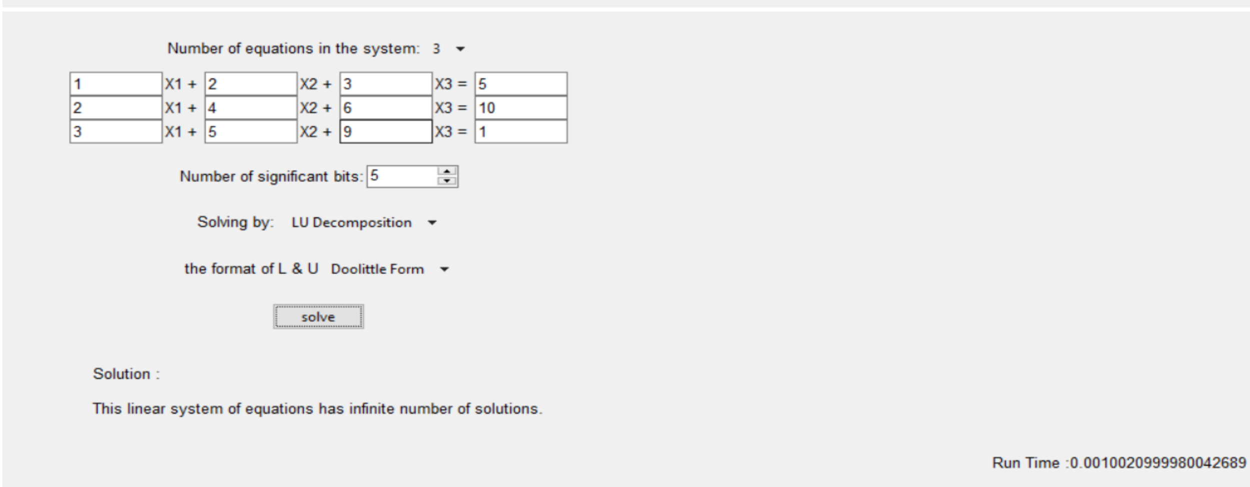


- **LU Decomposition**
    - **Doolittle**:

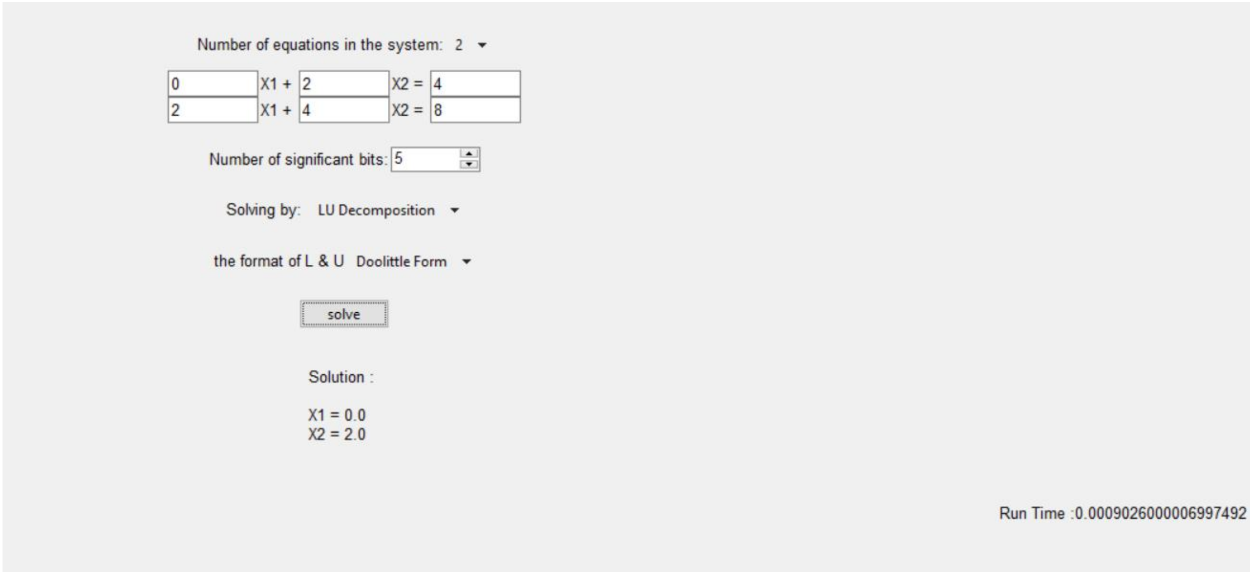
In case of infinite possibilities for solution:



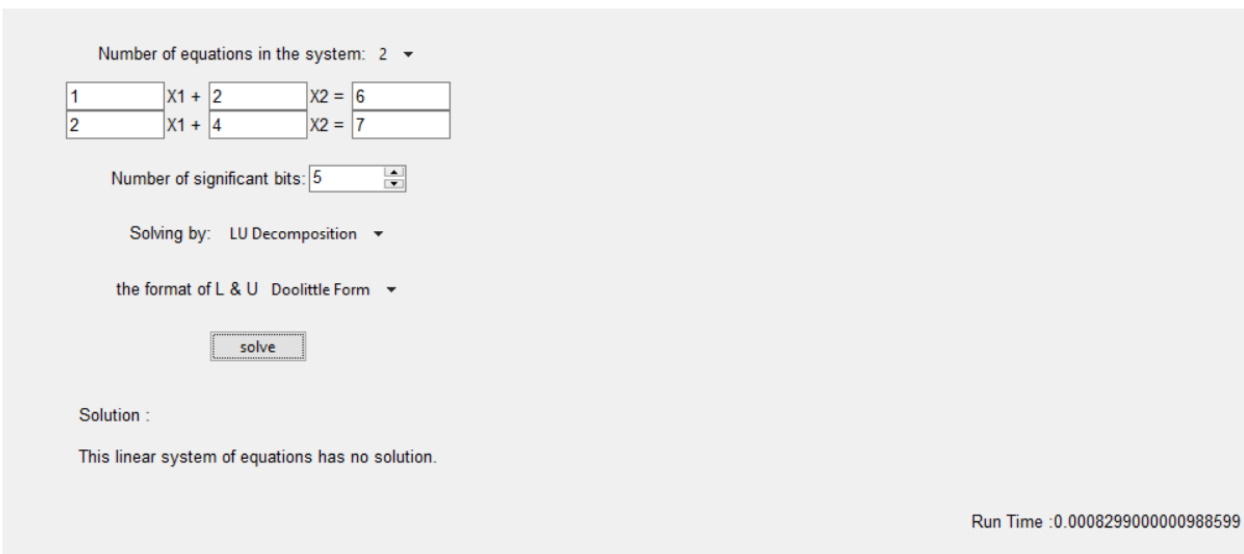




In case of zero pivot, pivoting strategy is applied where the row is interchanged with the first row containing non-zero pivot:

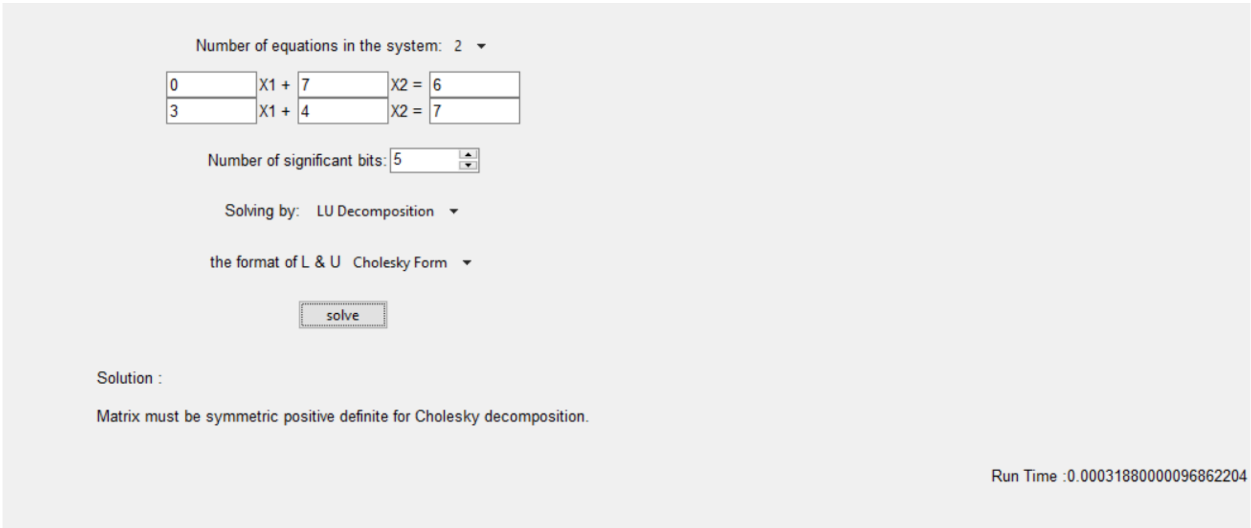


In case of no solution:

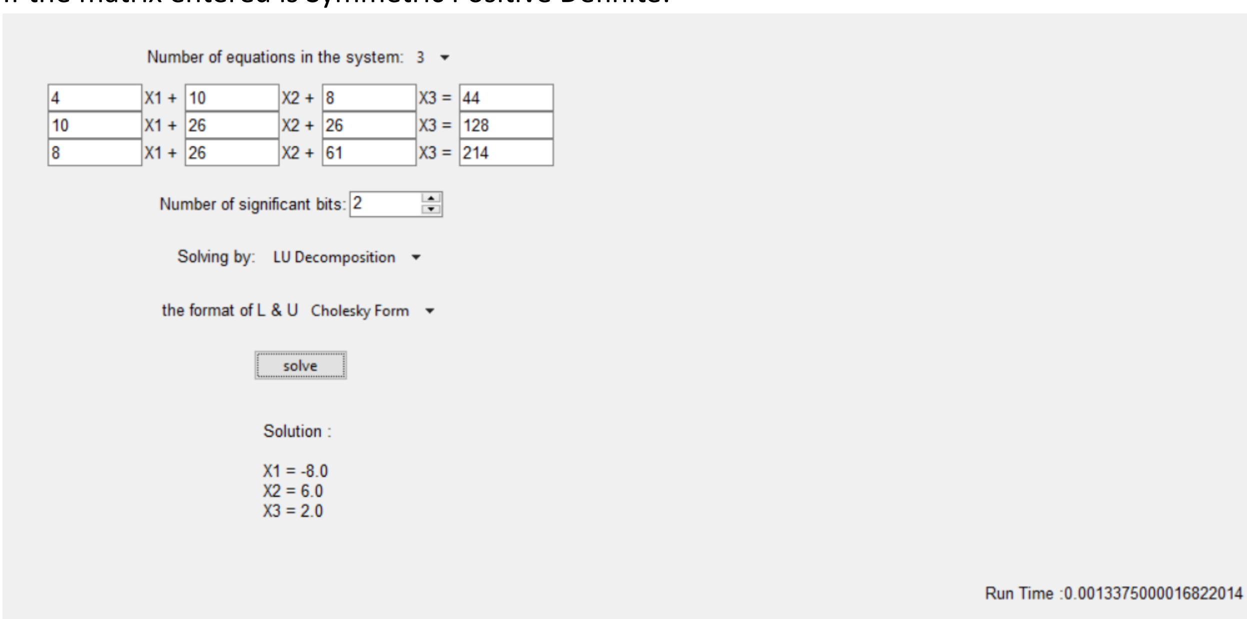


- **Cholesky:**

Cholesky method works only on special kind of matrices (Symmetric Positive Definite matrices). Hence, if the matrix is not Symmetric Positive Definite, an error shows up.

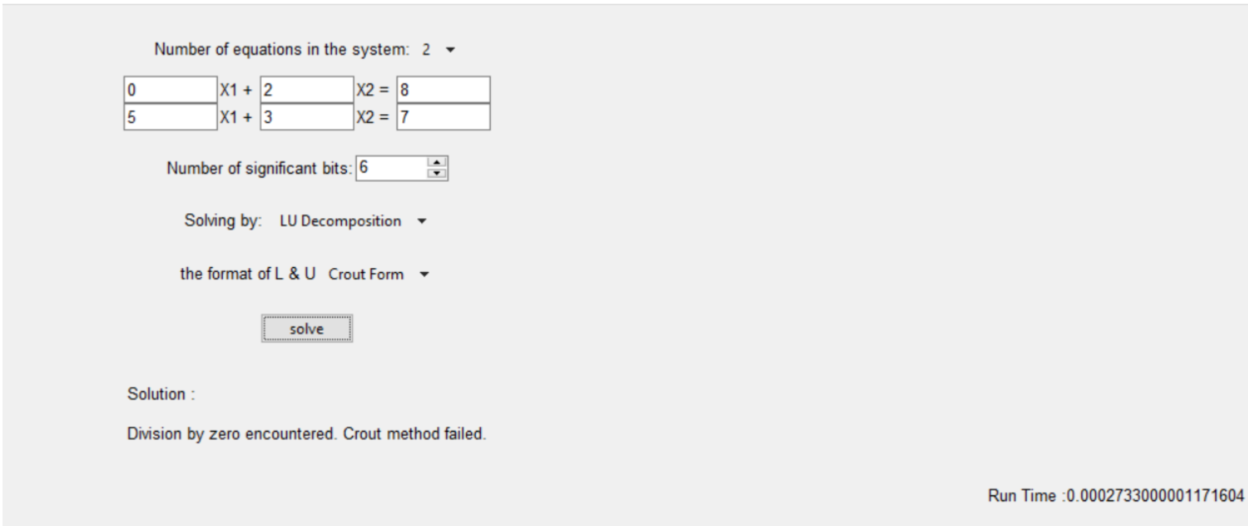


If the matrix entered is Symmetric Positive Definite:

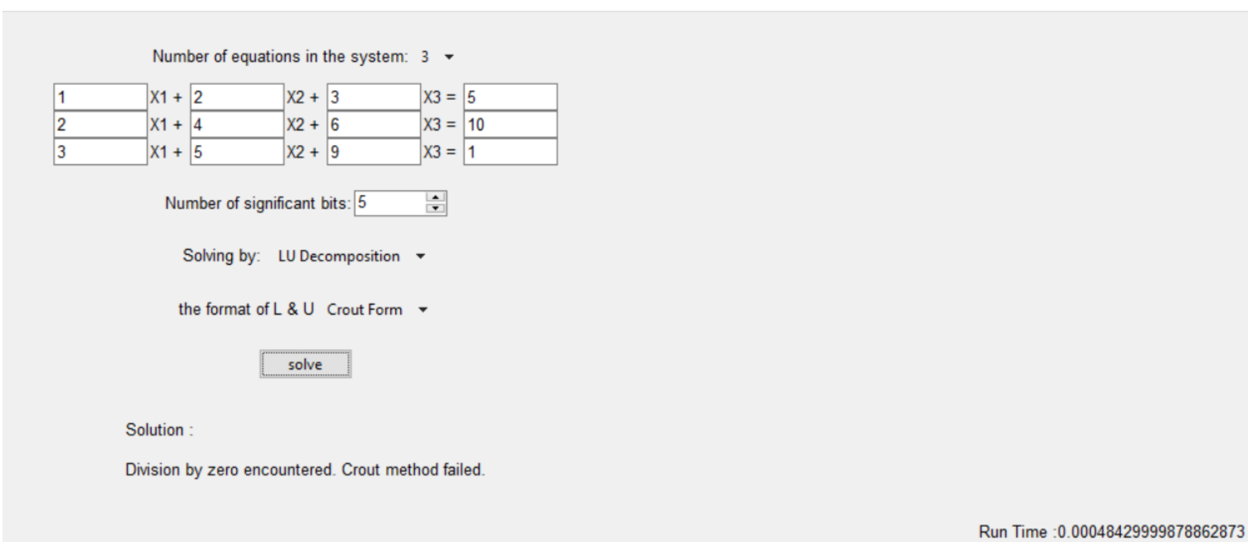


- **Crout:**

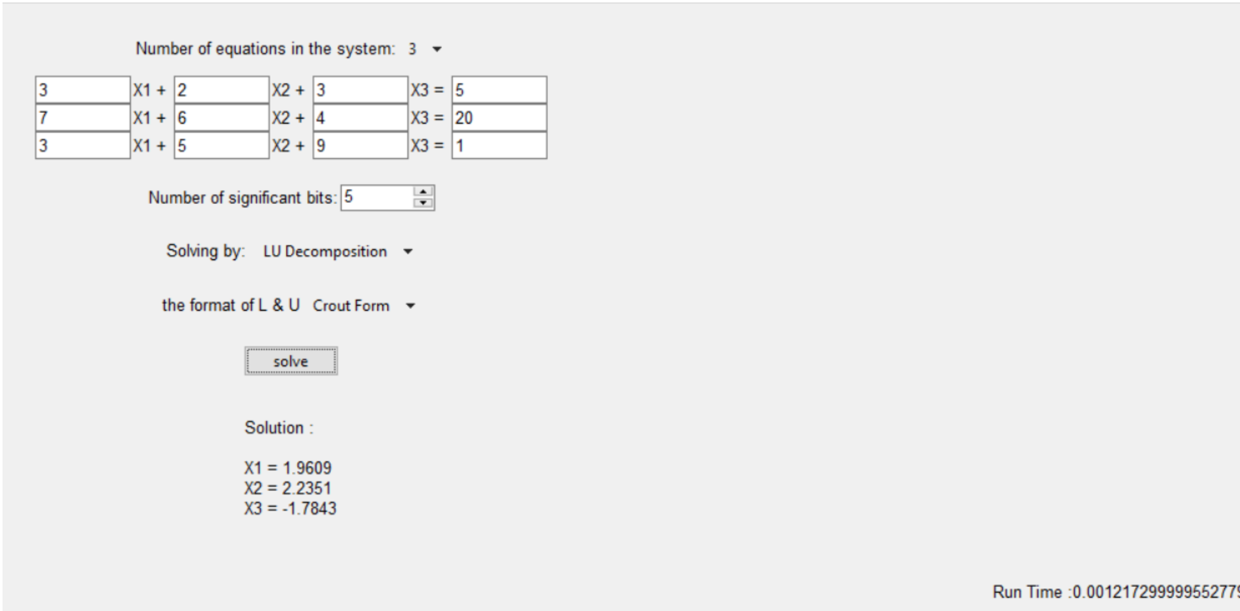
In case of zero pivot an error shows up indicating that Crout method has failed:



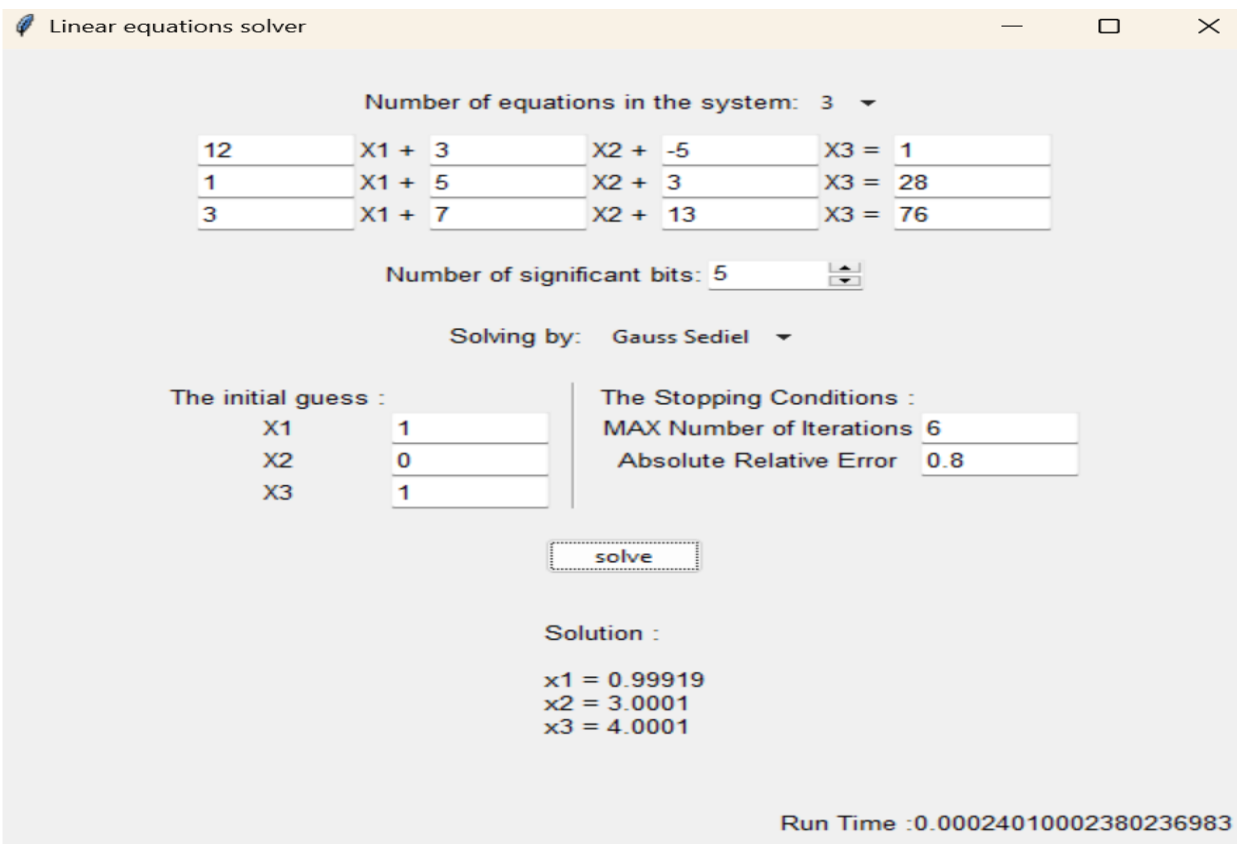
In case of two rows are multiples of each other, division by zero is encountered during elimination. Hence, it fails:



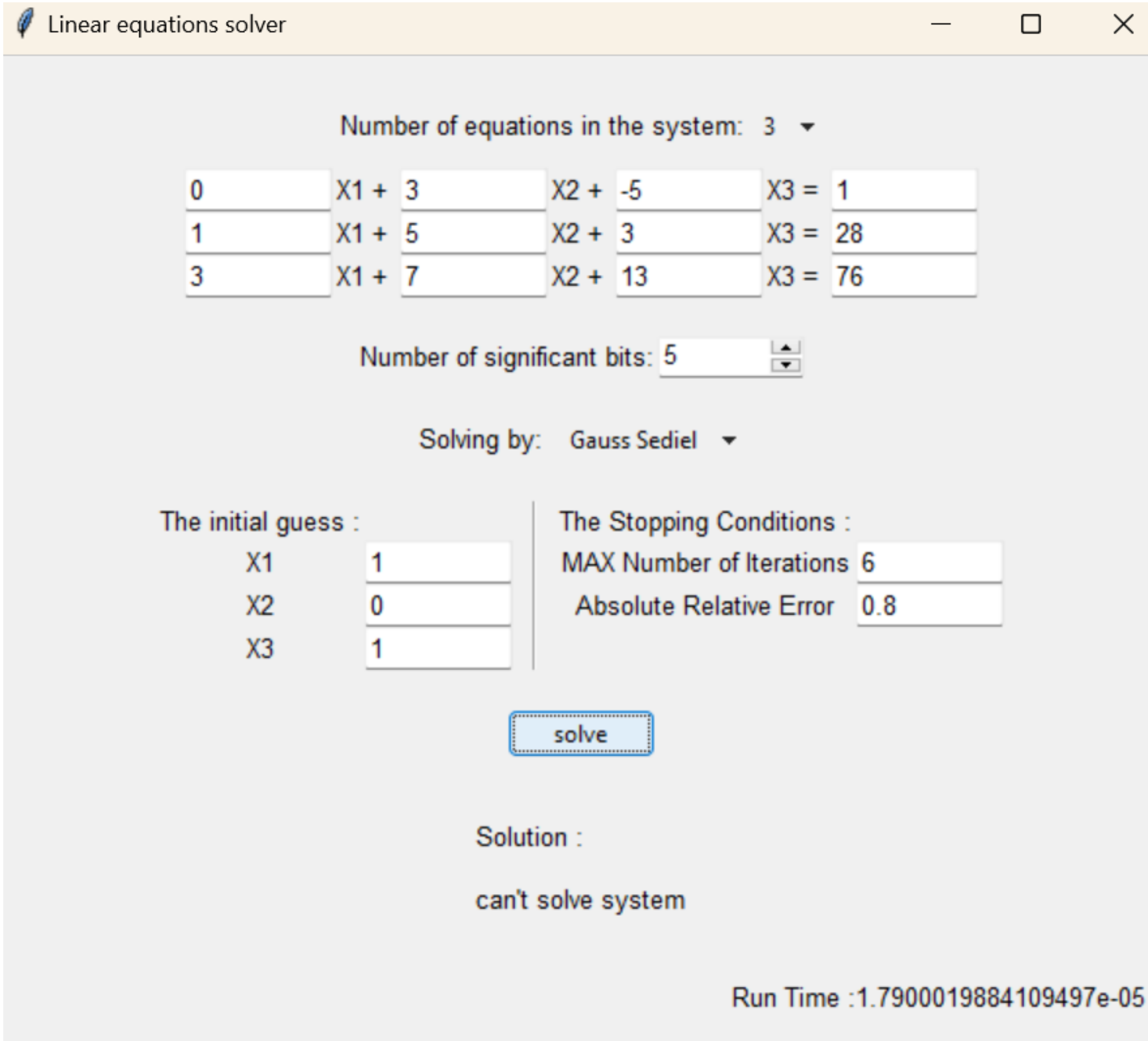
In case of unique solution:



- **Gauss seidel:**

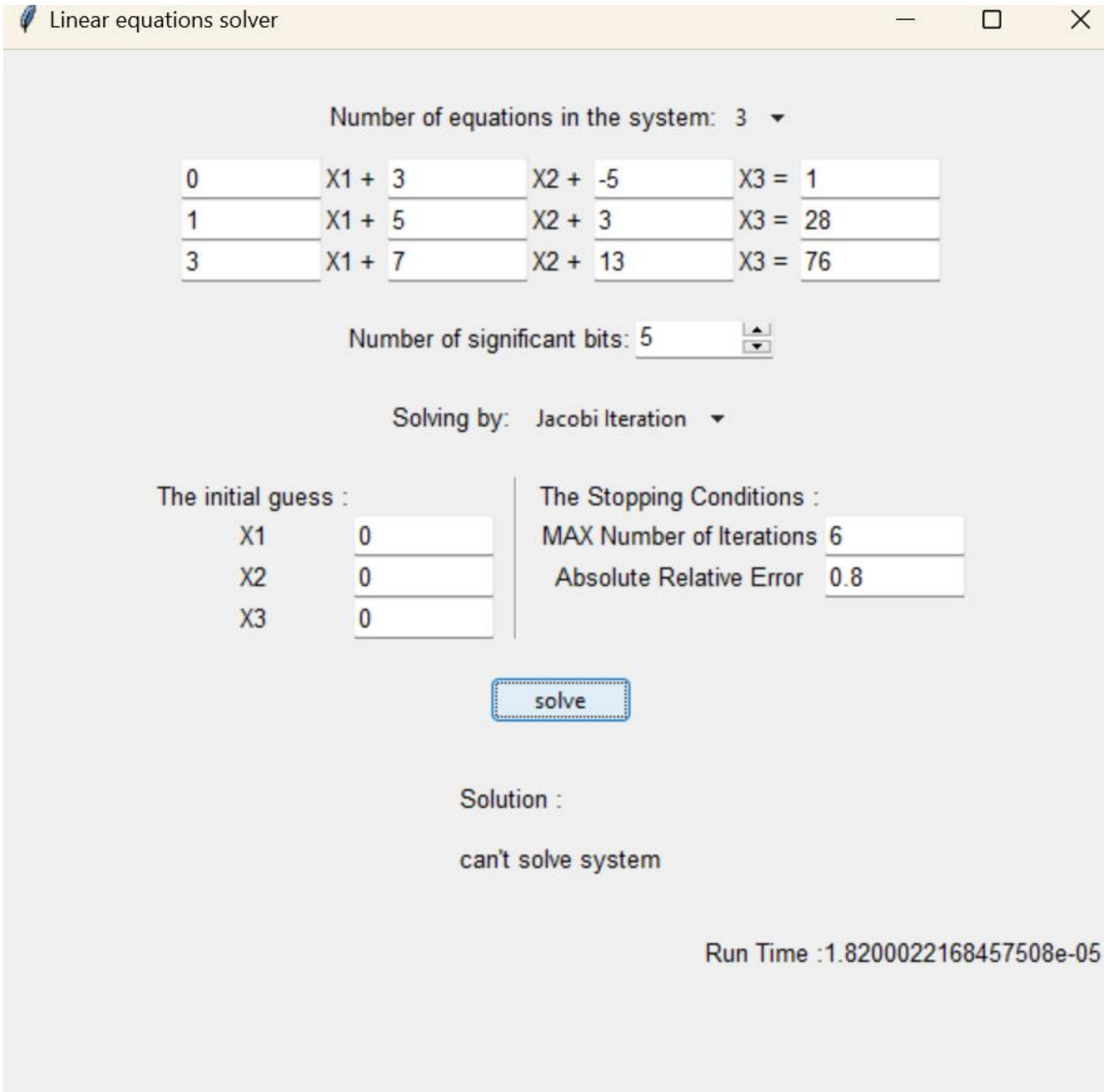


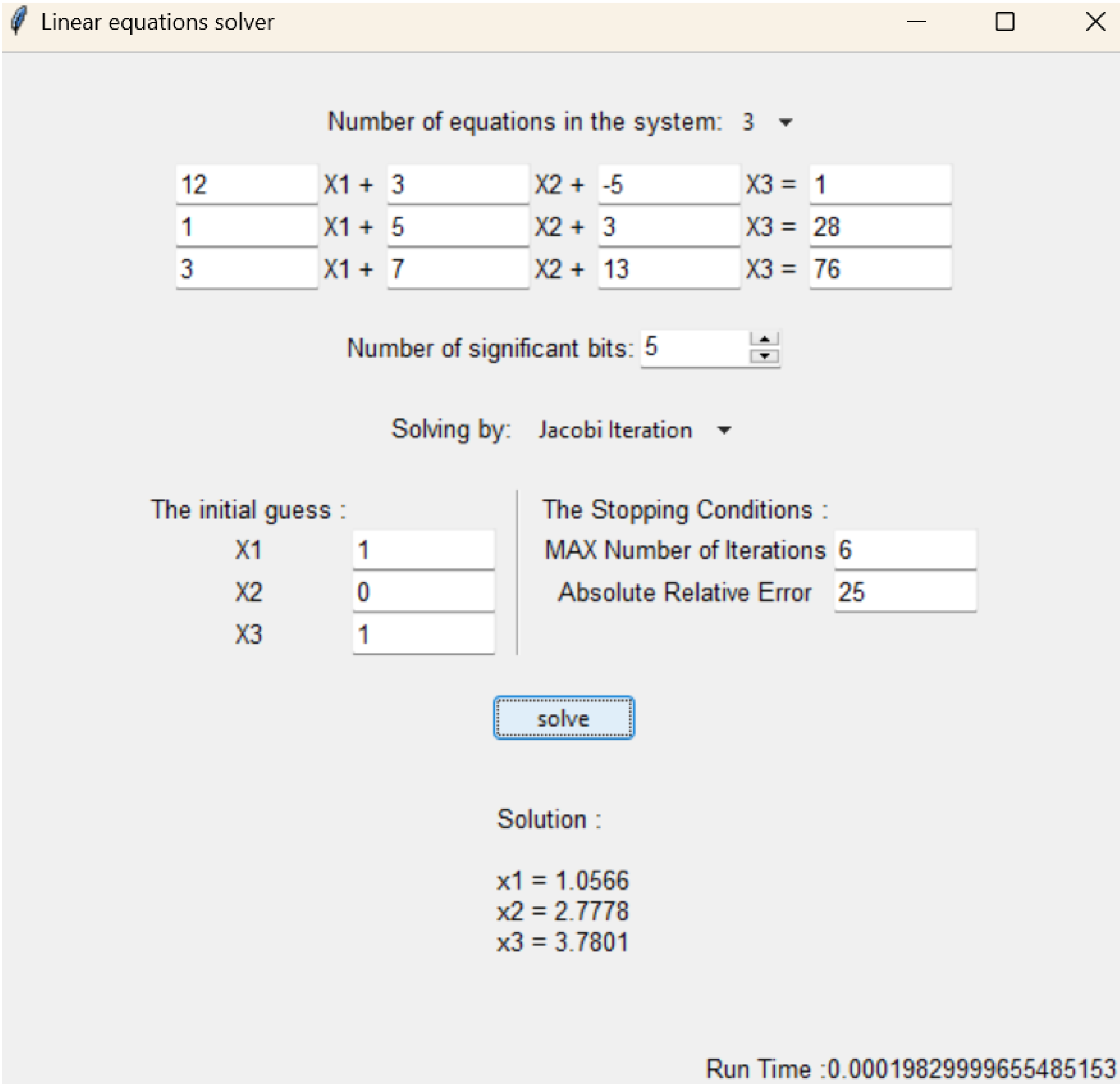
Test case when there are 0s in the diagonal



➢ Jacobi:

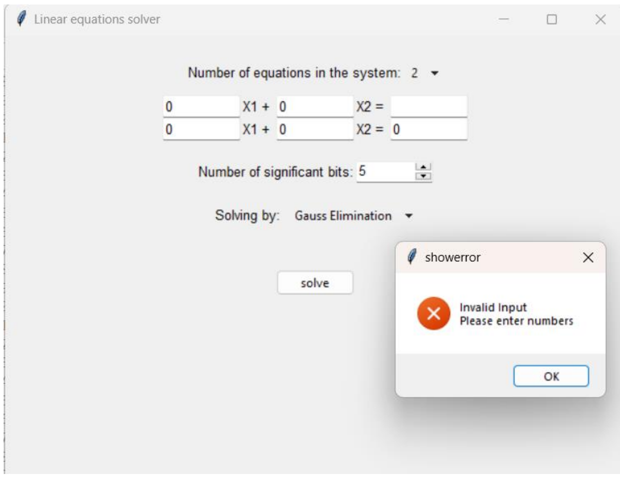
Test case when there are 0s in the diagonal

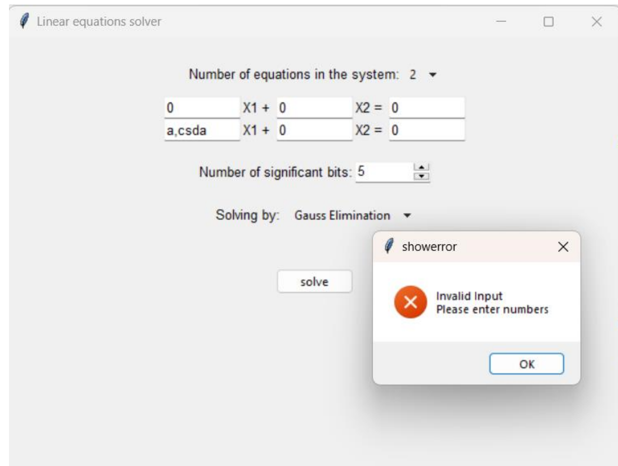



Linear equations solver
Number of equations in the system: 3
12 X1 + 3 X2 + -5 X3 = 1
1 X1 + 5 X2 + 3 X3 = 28
3 X1 + 7 X2 + 13 X3 = 76
Number of significant bits: 5
Solving by: Jacobi Iteration
The initial guess :
X1 1
X2 0
X3 1
The Stopping Conditions :
MAX Number of Iterations 6
Absolute Relative Error 25
solve
Solution :
x1 = 1.0566
x2 = 2.7778
x3 = 3.7801
Run Time :0.00019829999655485153

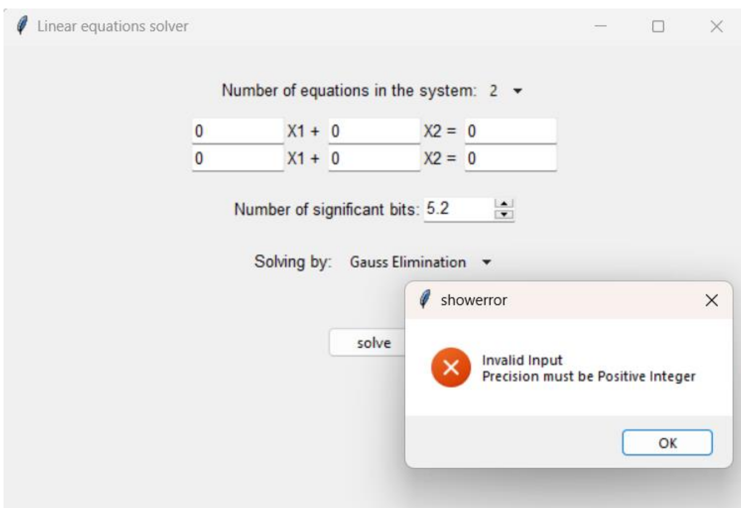
General test cases:

If the coefficients are non-numeric (none, letters, symbols … etc), a warning message appears





If the number of significant bits is not positive integer, a warning message appears



Comparison between different methods:

- **Gauss Elimination/ Gauss Jordan:**

Time Complexity:

Both Gauss Elimination and Gauss Jordan have time complexity of approximately O(n^3), where n is the size of the matrix, but Gauss Jordan is more costly when n is big.

Convergence:

The solution will always converge to the right answer and the accuracy of the answer is based on the number of the significant figures chosen.

Best Case and Approximate Errors:

Since we used scaling and pivoting in Gauss Elimination and Gauss Jordan, the numerical errors, which appears (for example) from dividing on very small number (nearly zero), has decreased which leads to decreasing the approximate error.

The approximate error is also affected by the number of significant figures the user chooses.

- **LU decomposition:**

Time Complexity:

- **Crout LU Decomposition:**

Time complexity is approximately O(n^3), Where n is the size of the matrix.

- **Doolittle LU Decomposition:**

Similar time complexity to Crout's method: O(n^3).

- **Cholesky Decomposition:**

Time complexity is approximately O $(\frac{n^3}{3})$. (For a symmetric positive definite matrix).

Convergence:

- **Doolittle:**

Generally, LU decomposition methods are guaranteed to converge for nonsingular matrices.

We applied pivoting strategies in Doolittle method. So, it improves numerical stability.

- **Crout:**

If the matrix has small or zero pivot values, convergence issues may arise as the method fails to yield an answer.

- **Cholesky Decomposition:**

Converges if the matrix is symmetric positive definite.

Best Case and Approximate Errors:

- **Doolittle:**

As we used pivoting in Doolittle method, pivoting helps select pivot elements that are more significant in magnitude, reducing the impact of round-off errors during the decomposition.

Best-case errors are generally improved because the pivot selection is more robust.

- **Crout:**

The absence of pivoting may in the implementation of the method may lead to increasing sensitivity to round-off errors, especially for ill-conditioned matrices.

- **Cholesky Decomposition:**

Best suited for symmetric positive definite systems.

May be more accurate than LU decomposition (Doolittle, Crout) for such systems.

Sensitive to the input matrix and can fail if not positive definite.

❖ Gauss Seidel / Jacobi:

Time Complexity:

- **Gauss Seidel**
  Time complexity is $O(n^2)$ (n is the number of equations).
- **Jacobi**
  Time complexity is $O(n^2)$ (n is the number of equations).

Convergence:

- **Gauss Seidel**

When the system is diagonally dominant, it will converge (it is a sufficient condition for convergence), but if it isn't diagonally dominant, we can't be sure about the convergence.

- **Jacobi**
  There is no guarantee for convergence.

<u>Best Case and Approximate Errors:</u>

- **Gauss Seidel**
  When the system is diagonally dominant, we will be sure that the system will yield an answer.
  It yields an approximate solution not exact.

- **Jacobi**
  It yields an approximate solution that is less than a specified error.

Data structure used:

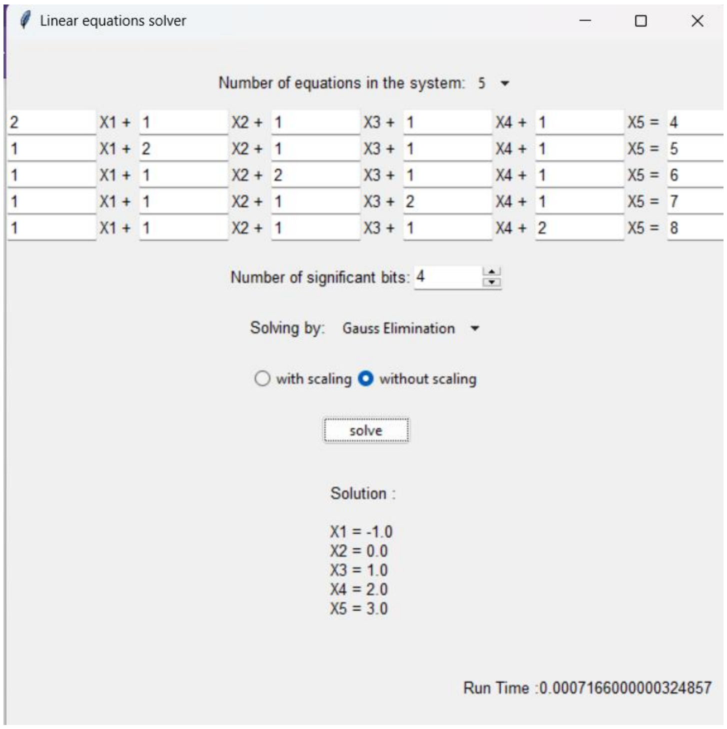
<u>NumPy Arrays (Matrices and Vectors):</u>

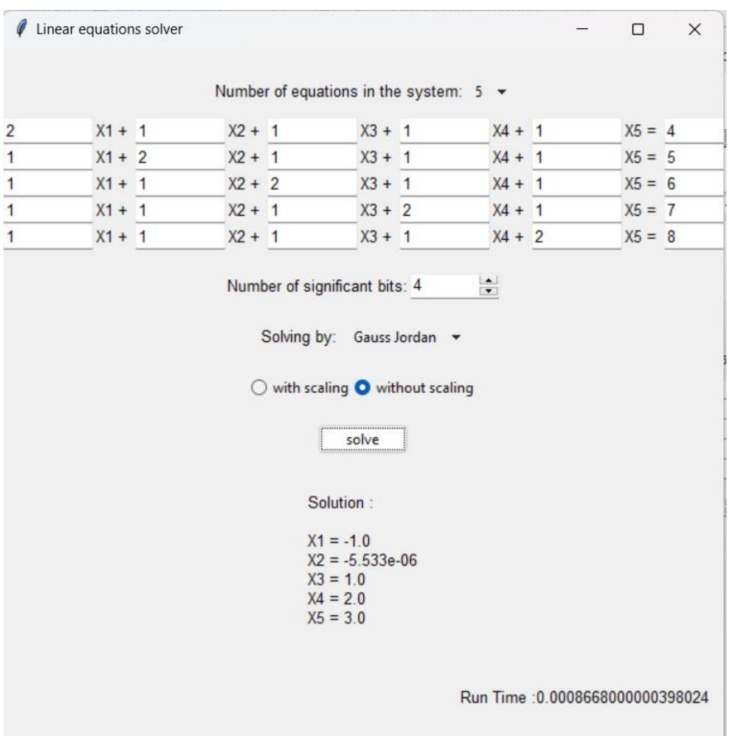
How Helpful:

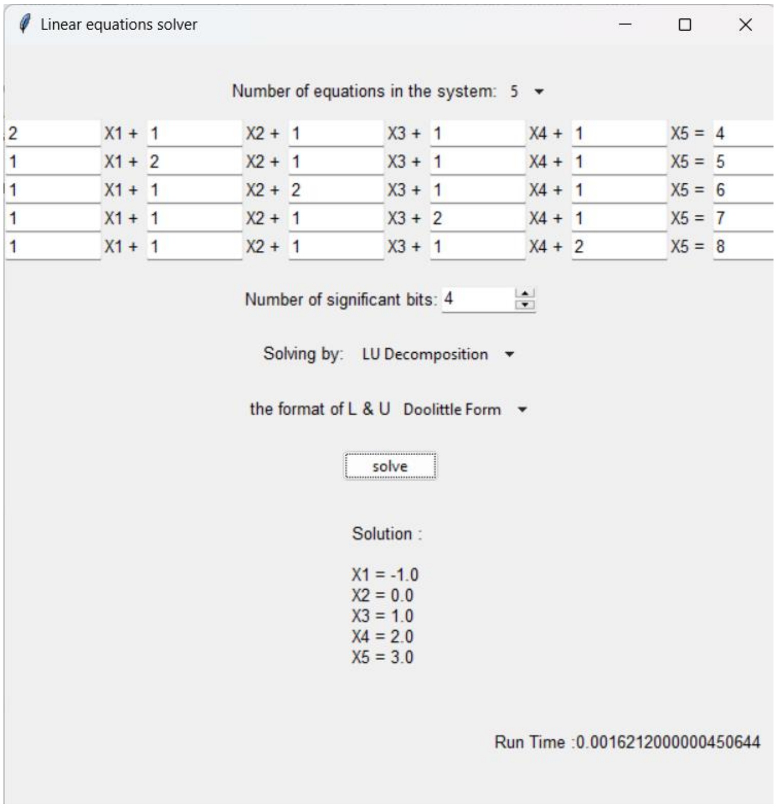
- NumPy arrays for matrix operations is highly beneficial due to their efficiency and readability.
- They provide a convenient and efficient way to represent matrices and vectors.
- NumPy provides a comprehensive set of built in functions and methods for array operations and linear algebraic operations.

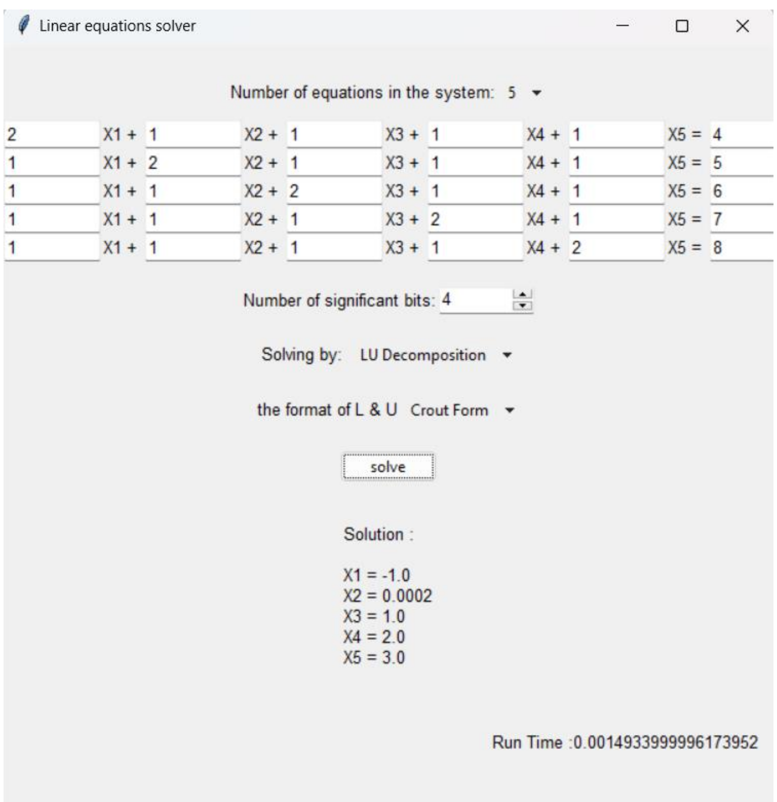
Phase1 test cases

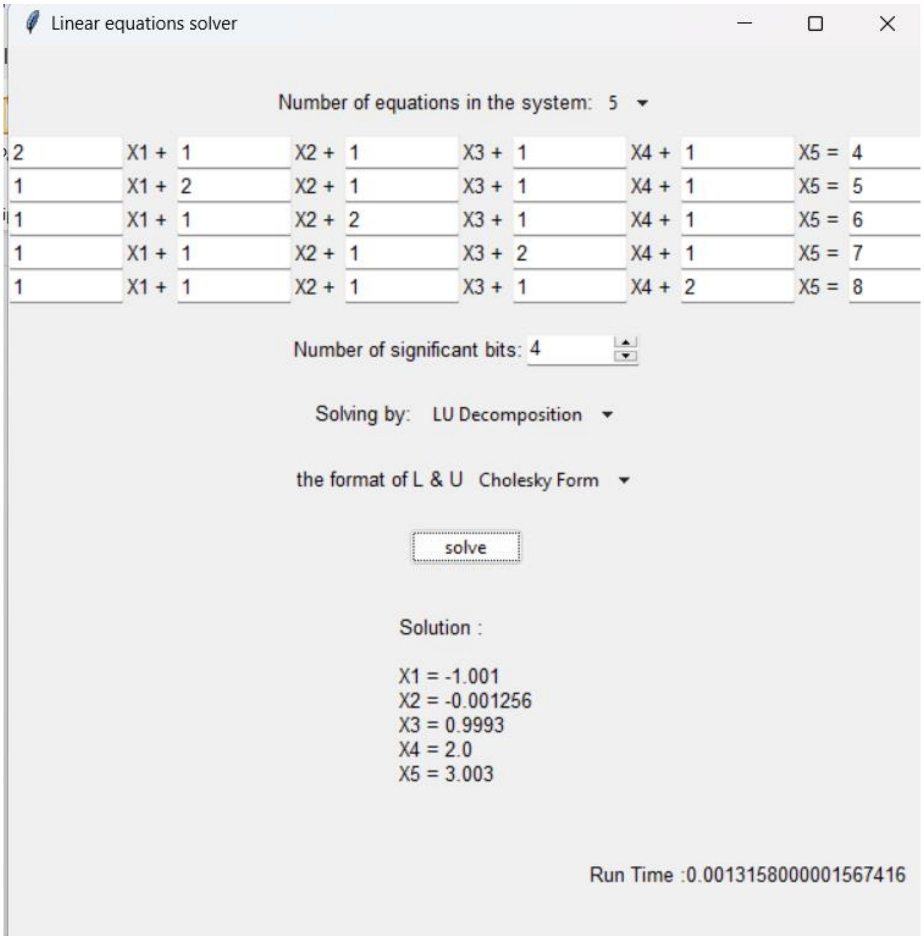
1.



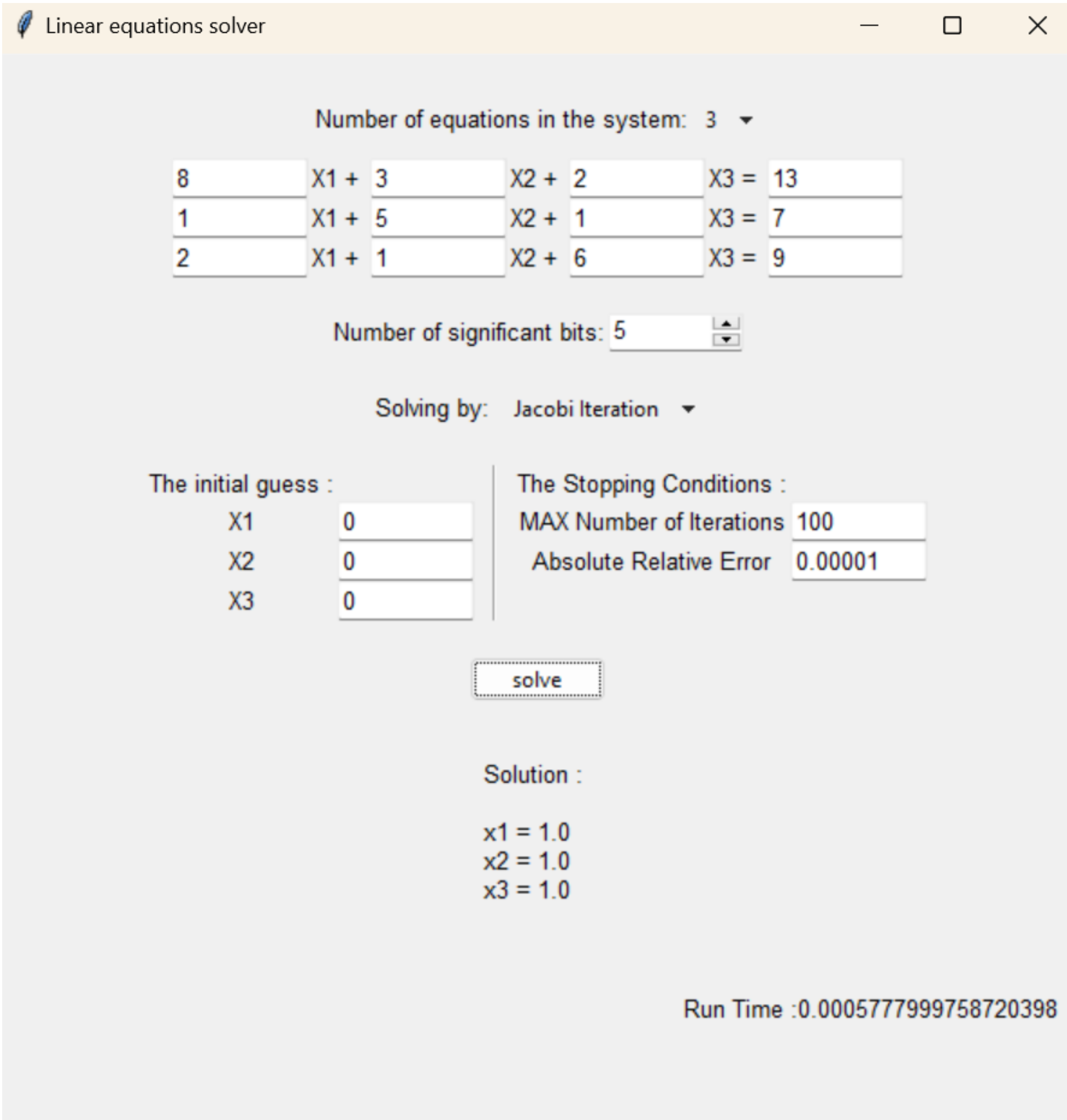


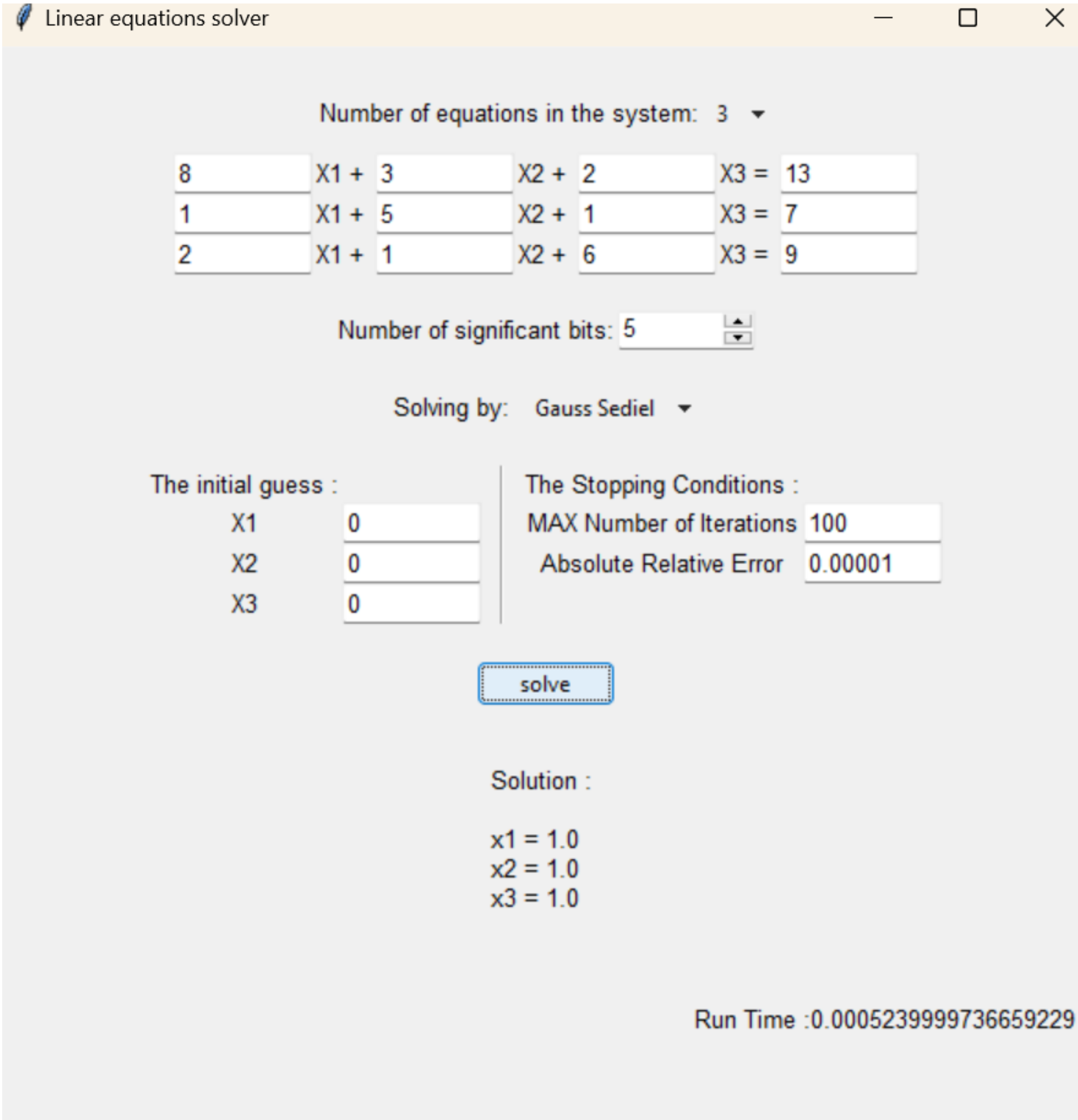

Linear equations solver
Number of equations in the system: 5
2 X1 + 1 X2 + 1 X3 + 1 X4 + 1 X5 = 4
1 X1 + 2 X2 + 1 X3 + 1 X4 + 1 X5 = 5
1 X1 + 1 X2 + 2 X3 + 1 X4 + 1 X5 = 6
1 X1 + 1 X2 + 1 X3 + 2 X4 + 1 X5 = 7
1 X1 + 1 X2 + 1 X3 + 1 X4 + 2 X5 = 8
Number of significant bits: 4
Solving by: LU Decomposition
the format of L & U Doolittle Form
solve
Solution :
X1 = -1.0
X2 = 0.0
X3 = 1.0
X4 = 2.0
X5 = 3.0
Run Time :0.0016212000000450644


Linear equations solver
Number of equations in the system: 5
2 X1 + 1 X2 + 1 X3 + 1 X4 + 1 X5 = 4
1 X1 + 2 X2 + 1 X3 + 1 X4 + 1 X5 = 5
1 X1 + 1 X2 + 2 X3 + 1 X4 + 1 X5 = 6
1 X1 + 1 X2 + 1 X3 + 2 X4 + 1 X5 = 7
1 X1 + 1 X2 + 1 X3 + 1 X4 + 2 X5 = 8
Number of significant bits: 4
Solving by: LU Decomposition
the format of L & U Crout Form
solve
Solution :
X1 = -1.0
X2 = 0.0002
X3 = 1.0
X4 = 2.0
X5 = 3.0
Run Time :0.0014933999996173952


Linear equations solver
Number of equations in the system: 5
2 X1 + 1 X2 + 1 X3 + 1 X4 + 1 X5 = 4
1 X1 + 2 X2 + 1 X3 + 1 X4 + 1 X5 = 5
1 X1 + 1 X2 + 2 X3 + 1 X4 + 1 X5 = 6
1 X1 + 1 X2 + 1 X3 + 2 X4 + 1 X5 = 7
1 X1 + 1 X2 + 1 X3 + 1 X4 + 2 X5 = 8
Number of significant bits: 4
Solving by: LU Decomposition
the format of L & U Cholesky Form
solve
Solution :
X1 = -1.001
X2 = -0.001256
X3 = 0.9993
X4 = 2.0
X5 = 3.003
Run Time :0.0013158000001567416

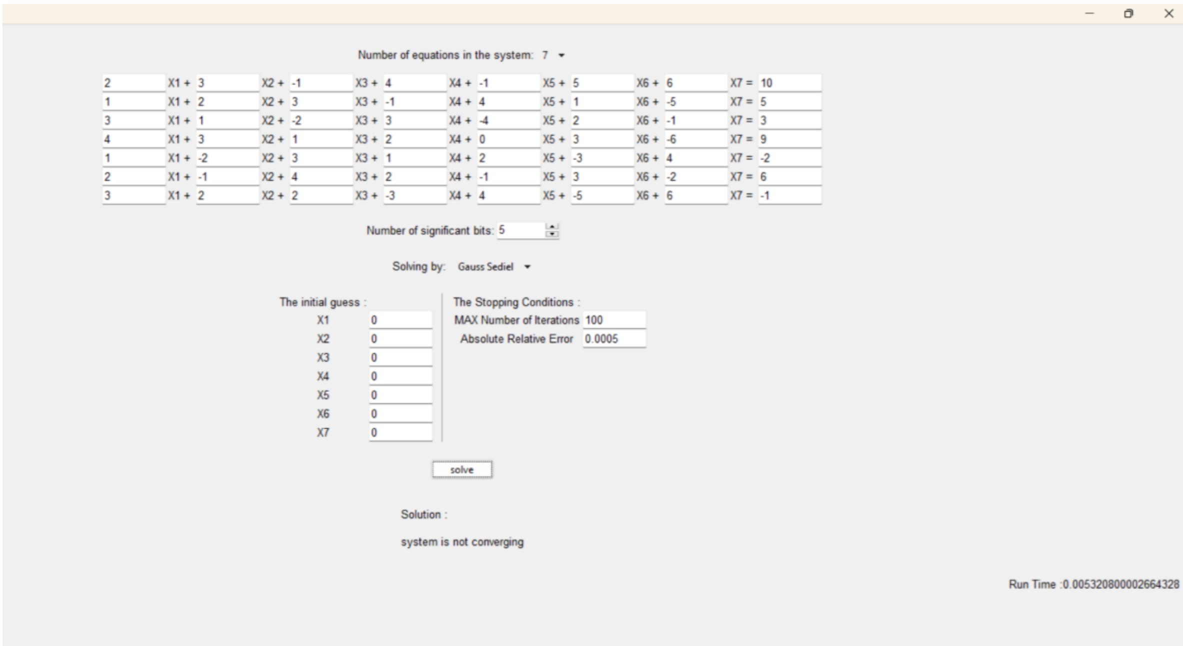
2.


Linear equations solver
Number of equations in the system: 3
8 X1 + 3 X2 + 2 X3 = 13
1 X1 + 5 X2 + 1 X3 = 7
2 X1 + 1 X2 + 6 X3 = 9
Number of significant bits: 5
Solving by: Jacobi Iteration
The initial guess :
X1 0
X2 0
X3 0
The Stopping Conditions :
MAX Number of Iterations 100
Absolute Relative Error 0.00001
solve
Solution :
x1 = 1.0
x2 = 1.0
x3 = 1.0
Run Time :0.0005777999758720398

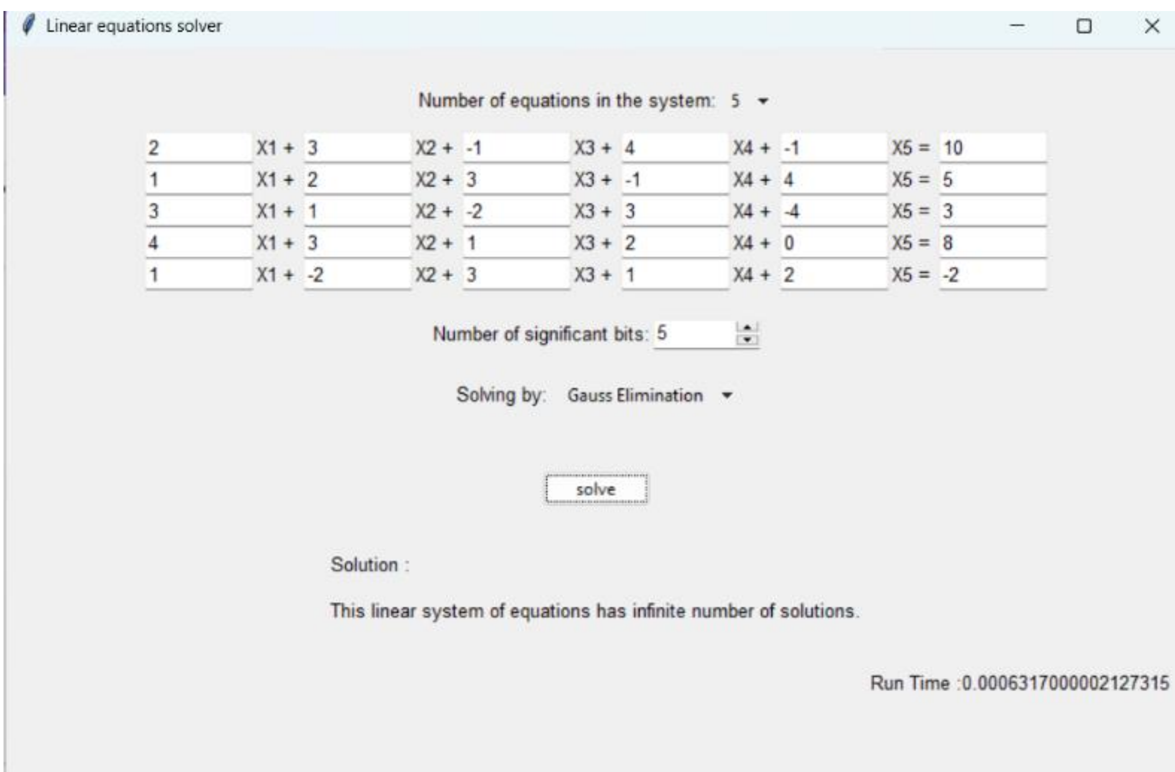


From time we can see that Gauss-Seidel converges faster than Jacobi. But both converged to the same answer.

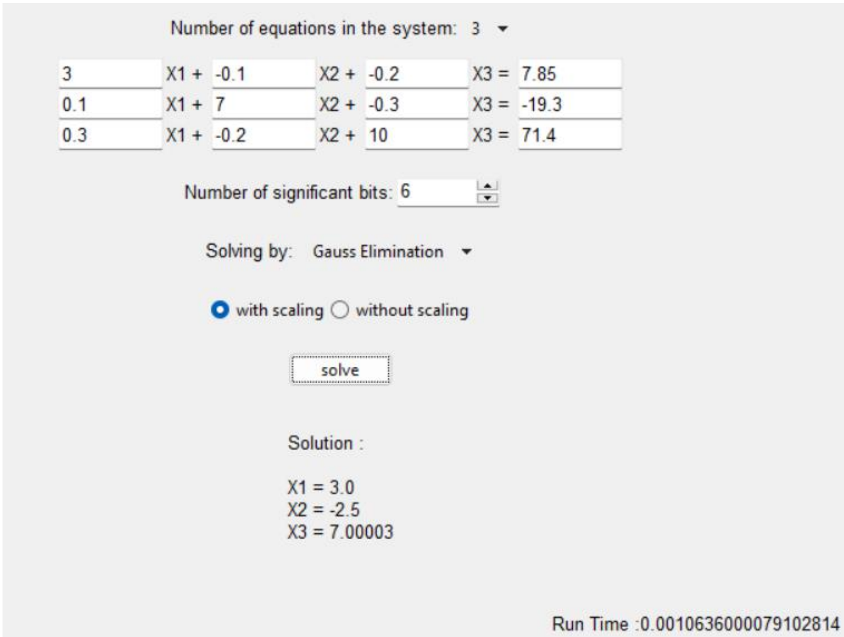
3.

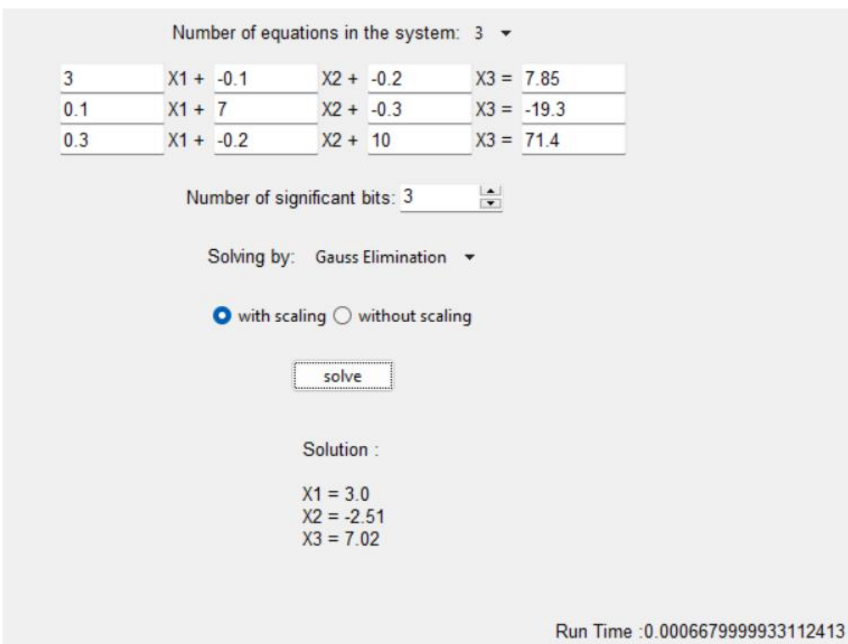

Number of equations in the system: 7
2 X1 + 3 X2 + -1 X3 + 4 X4 + -1 X5 + 5 X6 + 6 X7 = 10
1 X1 + 2 X2 + 3 X3 + -1 X4 + 4 X5 + 1 X6 + -5 X7 = 5
3 X1 + 1 X2 + -2 X3 + 3 X4 + -4 X5 + 2 X6 + -1 X7 = 3
4 X1 + 3 X2 + 1 X3 + 2 X4 + 0 X5 + 3 X6 + -6 X7 = 9
1 X1 + -2 X2 + 3 X3 + 1 X4 + 2 X5 + -3 X6 + 4 X7 = -2
2 X1 + -1 X2 + 4 X3 + 2 X4 + -1 X5 + 3 X6 + -2 X7 = 6
3 X1 + 2 X2 + 2 X3 + -3 X4 + 4 X5 + -5 X6 + 6 X7 = -1
Number of significant bits: 5
Solving by: Gauss Sediel
The initial guess :
X1 0
X2 0
X3 0
X4 0
X5 0
X6 0
X7 0
The Stopping Conditions :
MAX Number of Iterations 100
Absolute Relative Error 0.0005
solve
Solution :
system is not converging
Run Time :0.005320800002664328

4.

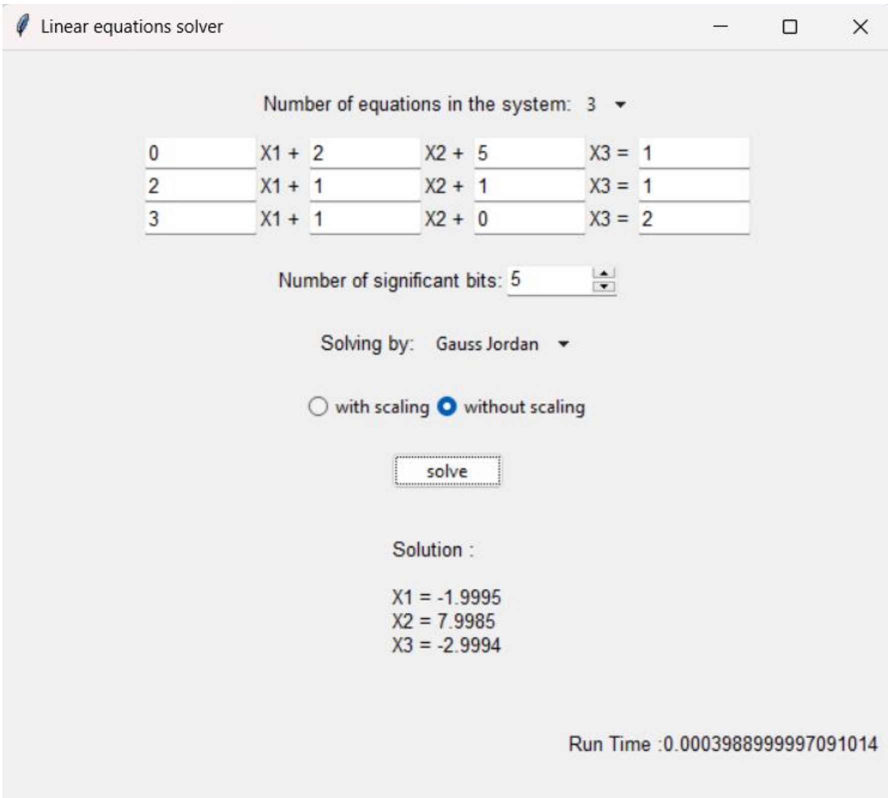

Linear equations solver
Number of equations in the system: 5
2 X1 + 3 X2 + -1 X3 + 4 X4 + -1 X5 = 10
1 X1 + 2 X2 + 3 X3 + -1 X4 + 4 X5 = 5
3 X1 + 1 X2 + -2 X3 + 3 X4 + -4 X5 = 3
4 X1 + 3 X2 + 1 X3 + 2 X4 + 0 X5 = 8
1 X1 + -2 X2 + 3 X3 + 1 X4 + 2 X5 = -2
Number of significant bits: 5
Solving by: Gauss Elimination
solve
Solution :
This linear system of equations has infinite number of solutions.
Run Time :0.0006317000002127315

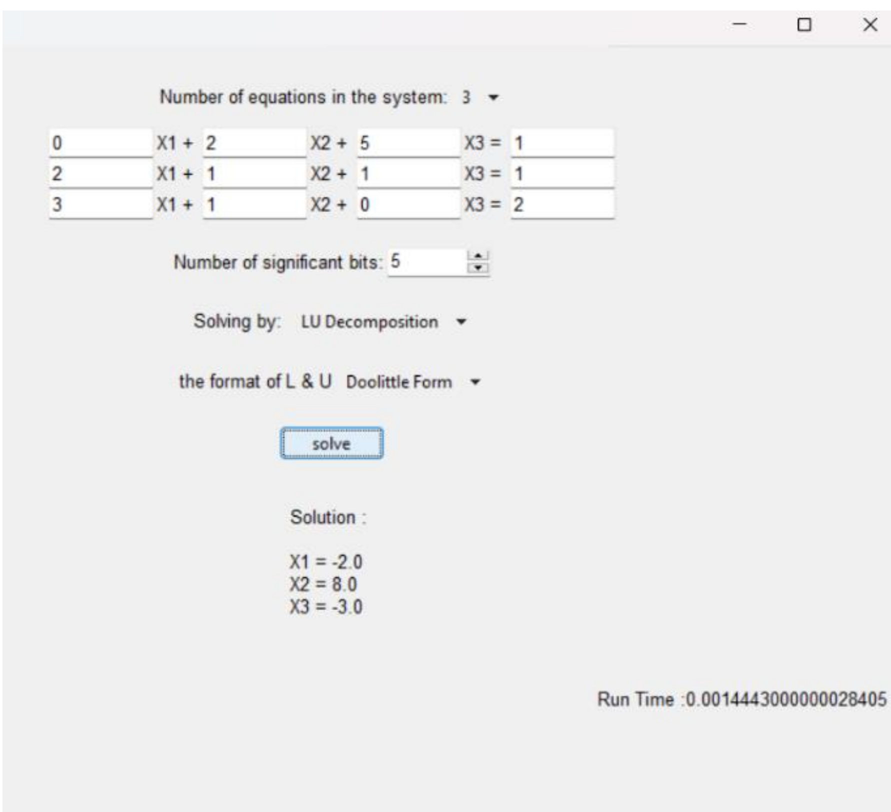
5.


Number of equations in the system: 3
3 X1 + -0.1 X2 + -0.2 X3 = 7.85
0.1 X1 + 7 X2 + -0.3 X3 = -19.3
0.3 X1 + -0.2 X2 + 10 X3 = 71.4
Number of significant bits: 6
Solving by: Gauss Elimination
with scaling
without scaling
solve
Solution :
X1 = 3.0
X2 = -2.5
X3 = 7.00003
Run Time :0.0010636000079102814

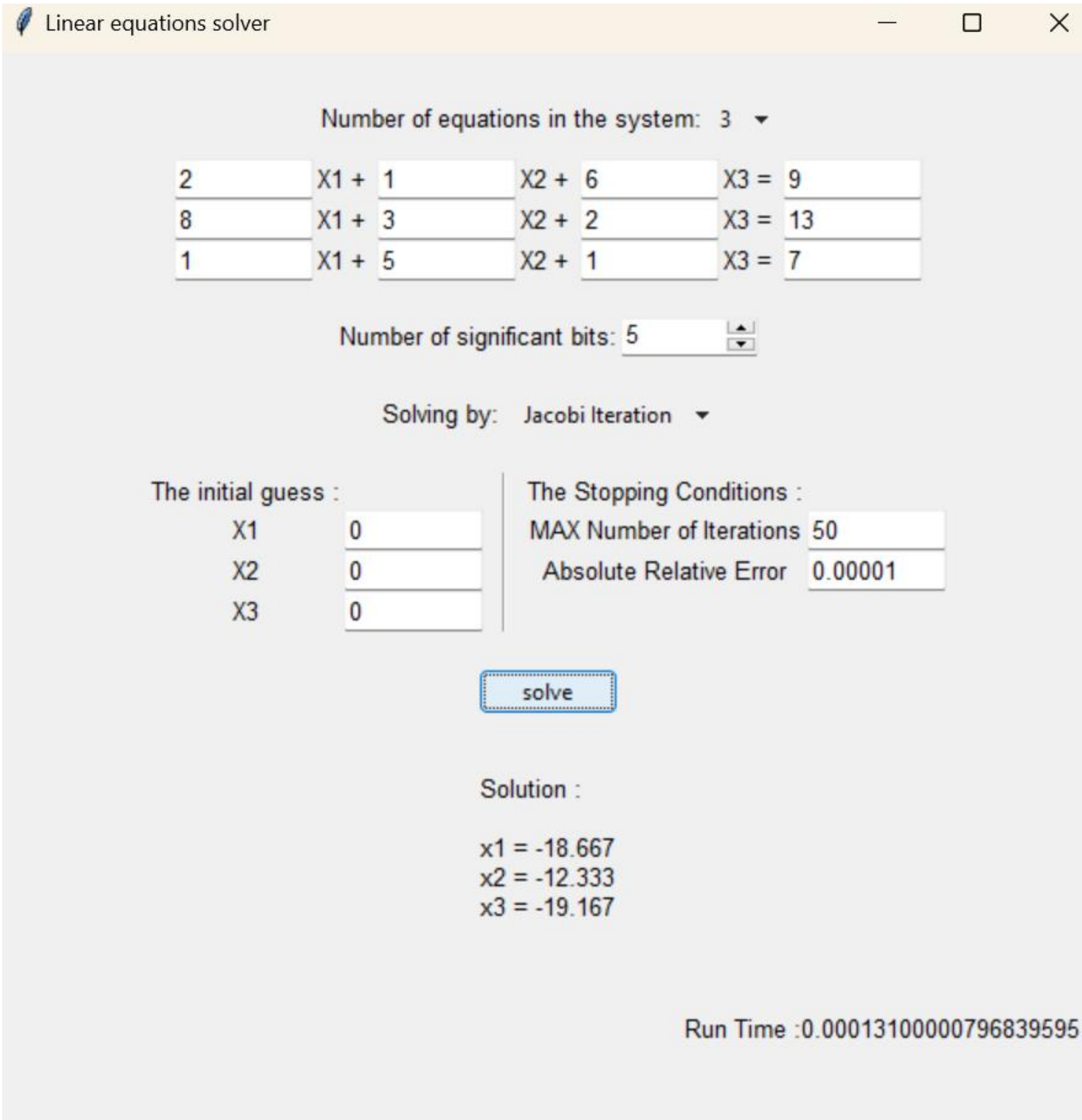

Number of equations in the system: 3
3 X1 + -0.1 X2 + -0.2 X3 = 7.85
0.1 X1 + 7 X2 + -0.3 X3 = -19.3
0.3 X1 + -0.2 X2 + 10 X3 = 71.4
Number of significant bits: 3
Solving by: Gauss Elimination
with scaling
without scaling
solve
Solution :
X1 = 3.0
X2 = -2.51
X3 = 7.02
Run Time :0.0006679999933112413

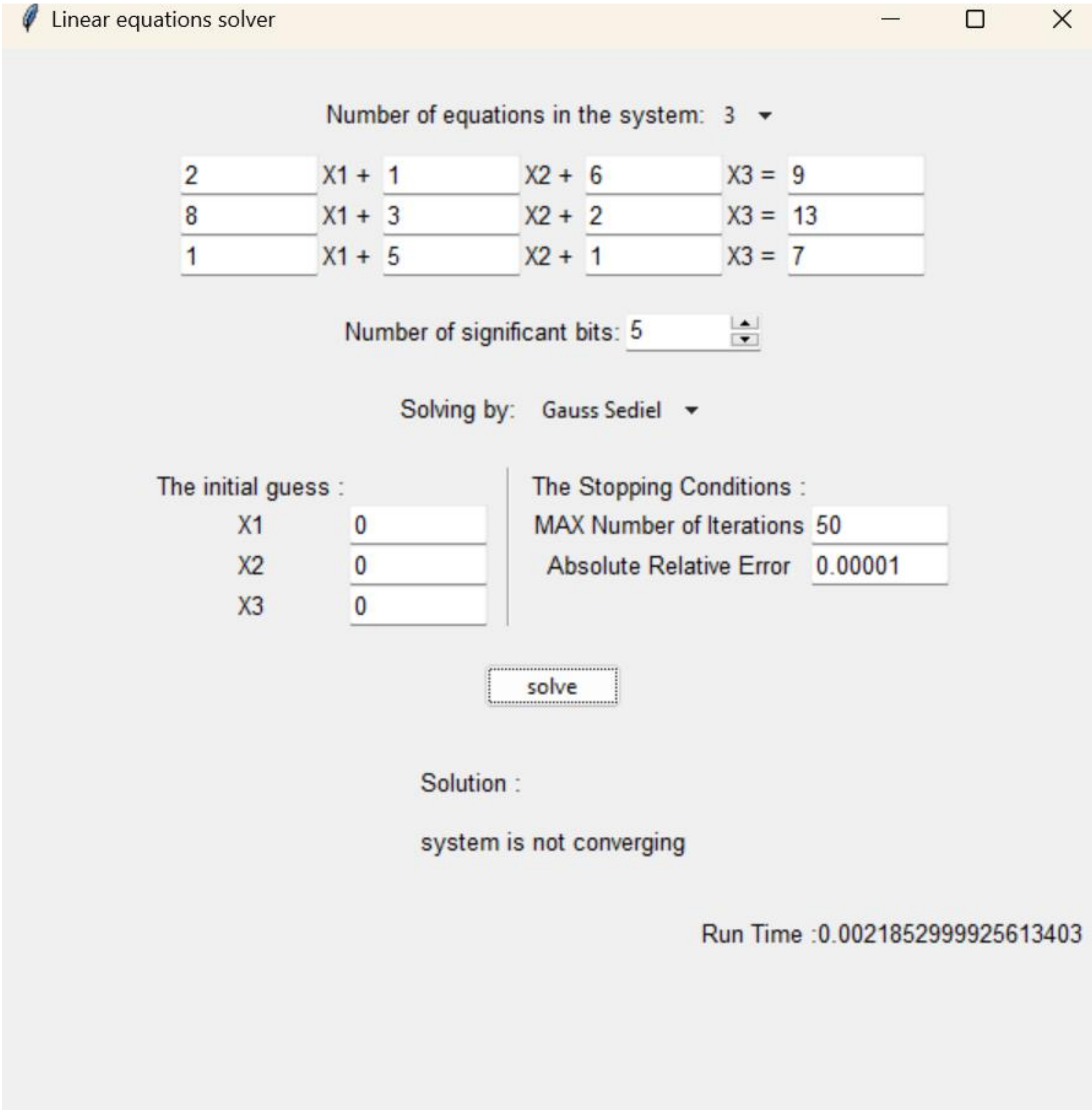
6.


Linear equations solver
Number of equations in the system: 3
0 X1 + 2 X2 + 5 X3 = 1
2 X1 + 1 X2 + 1 X3 = 1
3 X1 + 1 X2 + 0 X3 = 2
Number of significant bits: 5
Solving by: Gauss Jordan
with scaling
without scaling
solve
Solution :
X1 = -1.9995
X2 = 7.9985
X3 = -2.9994
Run Time :0.0003988999997091014


Number of equations in the system: 3
0 X1 + 2 X2 + 5 X3 = 1
2 X1 + 1 X2 + 1 X3 = 1
3 X1 + 1 X2 + 0 X3 = 2
Number of significant bits: 5
Solving by: LU Decomposition
the format of L & U Doolittle Form
solve
Solution :
X1 = -2.0
X2 = 8.0
X3 = -3.0
Run Time :0.0014443000000028405

7.


Linear equations solver
Number of equations in the system: 3
2 X1 + 1 X2 + 6 X3 = 9
8 X1 + 3 X2 + 2 X3 = 13
1 X1 + 5 X2 + 1 X3 = 7
Number of significant bits: 5
Solving by: Jacobi Iteration
The initial guess :
X1 0
X2 0
X3 0
The Stopping Conditions :
MAX Number of Iterations 50
Absolute Relative Error 0.00001
solve
Solution :
x1 = -18.667
x2 = -12.333
x3 = -19.167
Run Time :0.00013100000796839595



The system isn't diagonally dominant, so convergence isn't guaranteed.

As we can see the system didn't converge using Gauss-Seidel, while it converged with Jacobi; so, we conclude that the system is divergent

Bonus:

Scaling