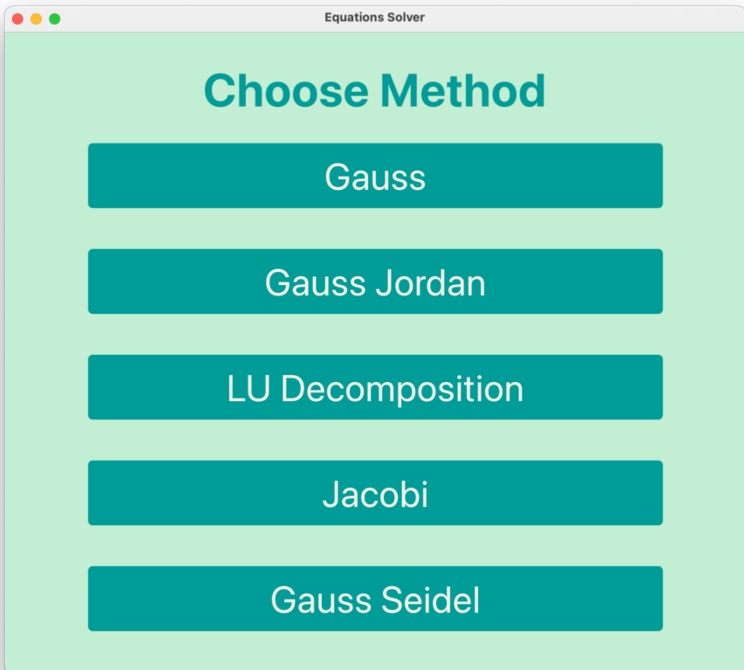
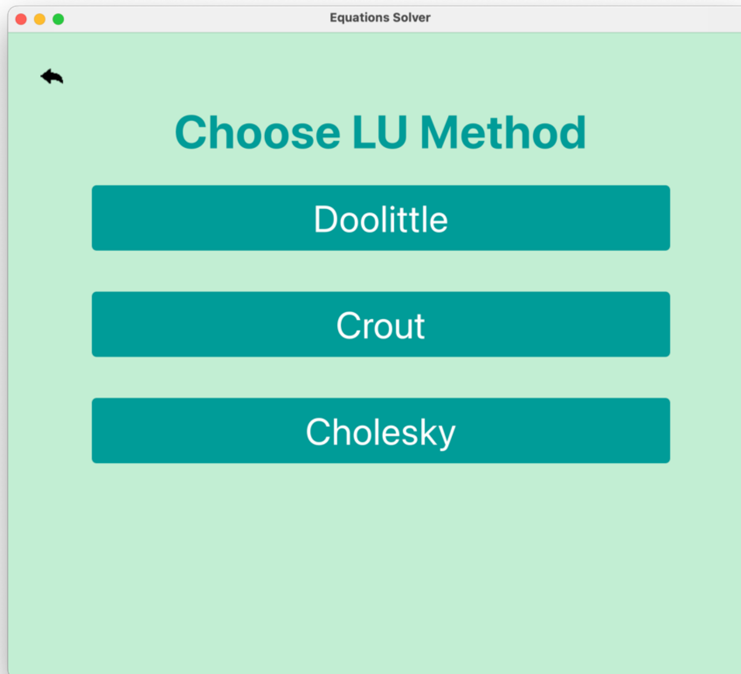
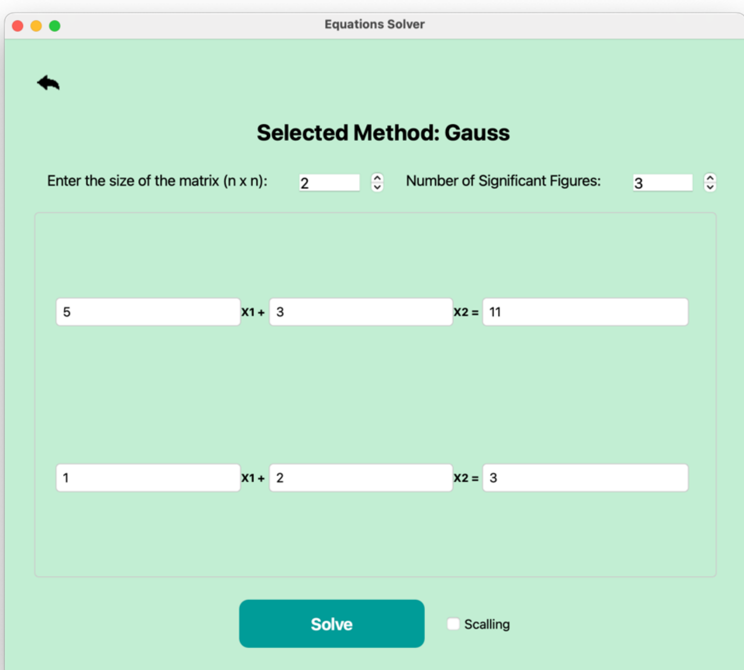
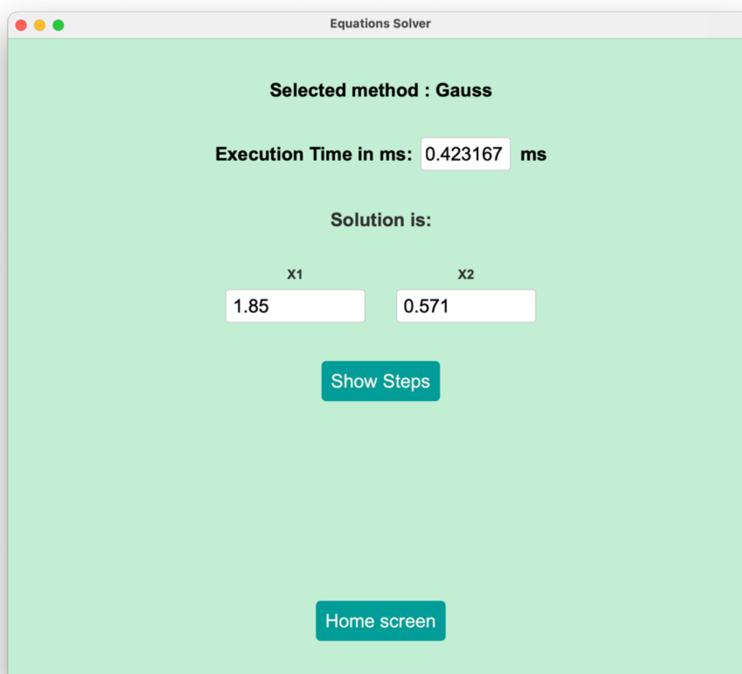
Project Report -Phase 1

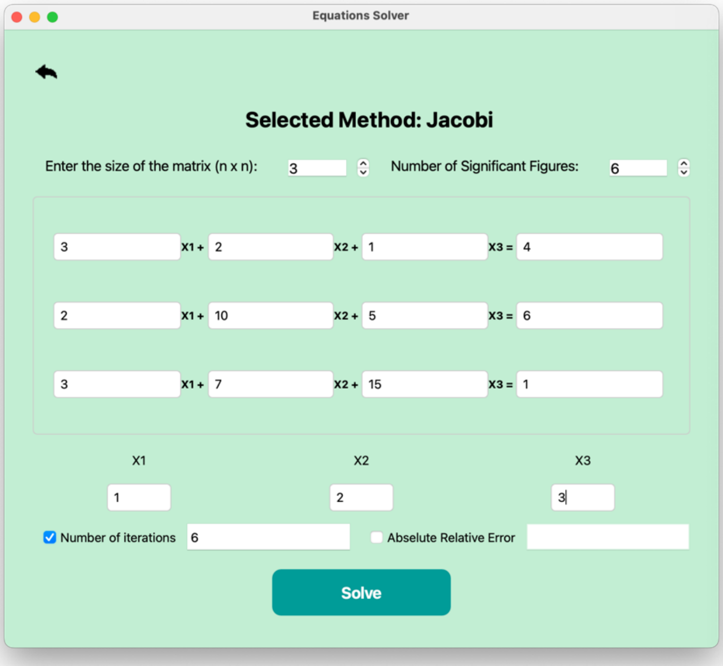
- **ScreenShots from GUI**

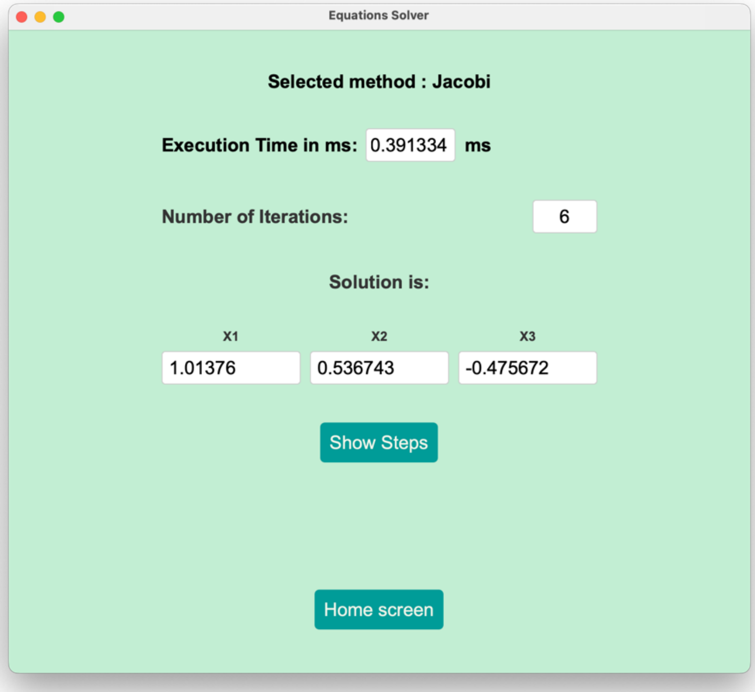


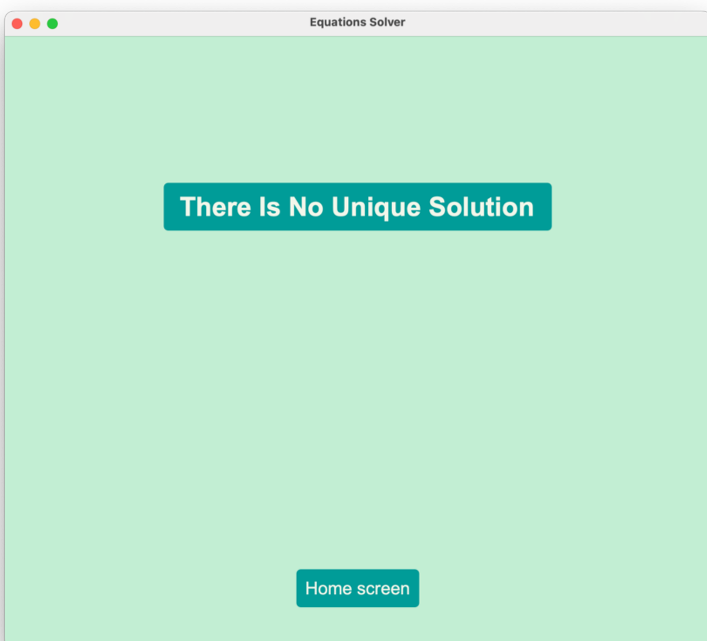


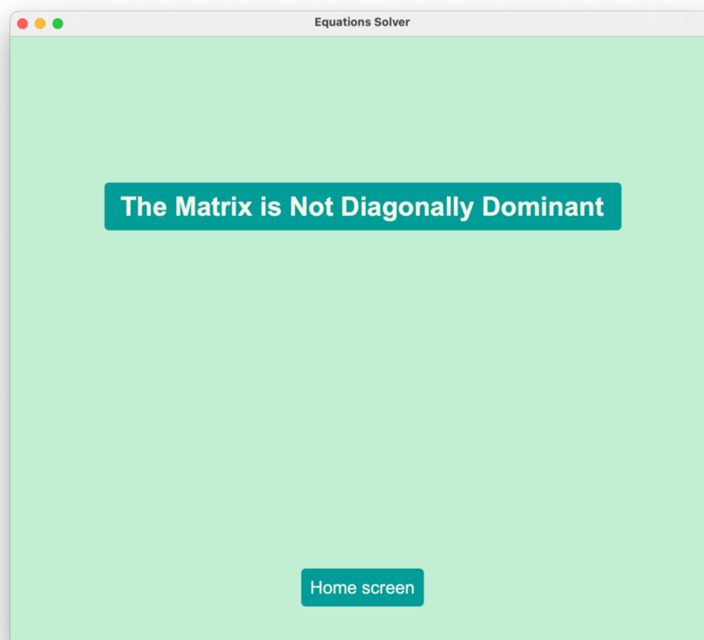





Equations Solver
Selected Method: Jacobi
Enter the size of the matrix (n x n):
3
Number of Significant Figures:
6
3 X1 + 2 X2 + 1 X3 = 4
2 X1 + 10 X2 + 5 X3 = 6
3 X1 + 7 X2 + 15 X3 = 1
X1
X2
X3
1
2
3
Number of iterations
6
Abselute Relative Error
Solve


Equations Solver
Selected method : Jacobi
Execution Time in ms: 0.391334 ms
Number of Iterations: 6
Solution is:
X1
X2
X3
1.01376
0.536743
-0.475672
Show Steps
Home screen


Equations Solver
There Is No Unique Solution
Home screen


Equations Solver
The Matrix is Not Diagonally Dominant
Home screen

- ## Pseudo-code for methods

  - ### Gauss

Assumption the system is augmented matrix

Forward Elimination with Partial Pivoting:

For i=0 to n : (Loop through pivot columns)

Find the Pivot Row:

Find the row index j with the largest absolute value in the i-th column

If j != i ,swap row j with row i

Eliminate Below Pivot:

For k=i+1 to n : (Loop through rows below the pivot row)

Compute the elimination factor: aug[k][i] / aug[i][i]

Update row k : aug[k][j] -= factor*aug[i][j]

Back Substitution:

For i=n−1 to 0 : (Loop through x's)

$$x[i] = \frac{\mathrm{aug}[i,n] - \sum_{j=i+1}^{n-1} \mathrm{aug}[i,j] \cdot x[j]}{\mathrm{aug}[i,i]}$$

Return the solution vector x

- ## Gauss-Jordan:

Assumption the system is augmented matrix

Forward Elimination with Partial Pivoting:

For i=0 to n : (Loop through pivot columns)

    Find the Pivot Row:

    Find the row index j with the largest absolute value in the i-th column

    If j != i ,swap row j with row i

    Eliminate Below Pivot:

    For k=i+1 to n : (Loop through rows below the pivot row)

        Compute the elimination factor: aug[k][i] / aug[i][i]

        Update row k : aug[k][j] -= factor*aug[i][j]

Eliminate Above Pivot:

For i=n-1 to 0 : (Loop through pivot columns)

    For j=i-1 to 0 : (Loop through rows above the pivot row)

        Compute the elimination factor: aug[j][i] / aug[i][i]

        Update row j : aug[j][i] -= factor*aug[i][i]

Find the solution vector x:

For i=0 to n:

    x[i] = aug[i][n] / aug[i][i]

Return the solution vector x

- ## LU [DoLittle]:

Method Decomposition:

```
For k from 0 to n-1:
    For j from k to n-1:
        Initialize sum_var to 0
        For m from 0 to k-1:
            sum_var += L[k, m] * U[m, j]
        Set U[k, j] = A[k, j] - sum_var
    For i from k+1 to n-1:
        Initialize sum_var to 0
        For m from 0 to k-1:
            sum_var += L[i, m] * U[m, k]
        Set L[i, k] = (A[i, k] - sum_var) / U[k, k]
```

Method solve:

```
Call decompose() to perform LU decomposition and calculate L and U matrices
For i from 0 to n-1:
    Initialize sum_var to 0
    For j from 0 to i-1:
        sum_var += L[i, j] * y[j]
    Set y[i] = b[i] - sum_var
For i from n-1 down to 0:
    Initialize sum_var to 0
    For j from i+1 to n-1:
        sum_var += U[i, j] * x[j]
    Set x[i] = (y[i] - sum_var) / U[i, i]
```

- ## LU [Crout]:

Method Decomposition:

```
For i from 0 to n-1:
    For j from 0 to n-1:
        If j == 0:
            Set L[i, j] = matrix[i, j]
        Else If i == 0:
            Set U[i, j] = matrix[i, j] / L[0, 0]
        Else If i > j:
            Initialize sumVar to 0
            For z from 0 to j-1:
                sumVar += L[i, z] * U[z, j]
            Set L[i, j] = matrix[i, j] - sumVar
        Else If i < j:
            Initialize sumVar to 0
            For z from 0 to i-1:
                sumVar += L[i, z] * U[z, j]
            Set U[i, j] = (matrix[i, j] - sumVar) / L[i, i]
        Else If i == j:
            Initialize sumVar to 0
            For z from 0 to i-1:
                sumVar += L[i, z] * U[z, j]
            Set L[i, j] = matrix[i, j] - sumVar
            Set U[i, j] = 1
```

Method solve:

```
Call decompose() method to perform LU decomposition
For i from 0 to n-1:
    Initialize sumVar to 0
    For j from 0 to i-1:
        sumVar += L[i, j] * y[j]
    Set y[i] = (b[i] - sumVar) / L[i, i]
For i from n-1 down to 0:
    Initialize sumVar to 0
    For j from i+1 to n-1:
        sumVar += U[i, j] * x[j]
    Set x[i] = (y[i] - sumVar)
```

[Cholesky]:

Method Decomposition:

```
For i from 0 to n-1:
  For j from 0 to i:
    Initialize sumVar to 0
    If i == j:
      For z from 0 to i-1:
        sumVar += L[i, z] ^ 2
      Set L[i, i] = sqrt(matrix[i, i] - sumVar)
    Else:
      For z from 0 to j-1:
        sumVar += L[j, z] * L[i, z]
      Set L[i, j] = (matrix[i, j] - sumVar) / L[j, j]
```

Method solve:

```
Call decompose() method to perform Cholesky decomposition
For i from 0 to n-1:
  Initialize sumVar to 0
  For j from 0 to i-1:
    sumVar += L[i, j] * y[j]
  Set y[i] = (b[i] - sumVar) / L[i, i]
For i from n-1 down to 0:
  Initialize sumVar to 0
  For j from i+1 to n-1:
    sumVar += L[j, i] * x[j]
  Set x[i] = (y[i] - sumVar) / L[i, i]
```

- **Jacobi:**

```
#solve with iterations
for t in range(iterations):
    for i in range(len(matrixB)):
        new[i] = matrixB[i]
        for j in range(len(matrixB)):
            if i != j:
        new[i] /= SFCalc(matrixA[i][i], SignificantFigures)
    old =new[:]
res = new
```

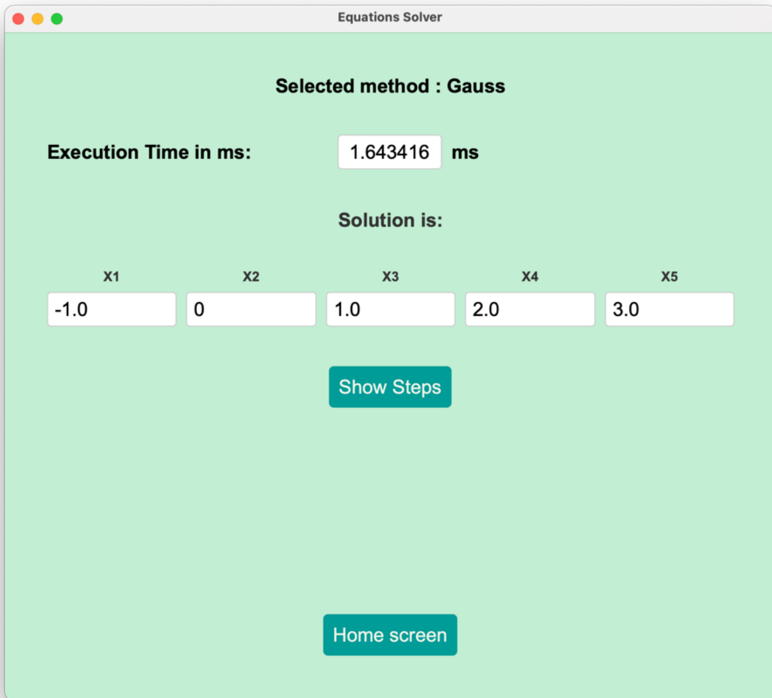
```
#solve with tolerance
tolerance /= 100
flag = False //to know when to end iteration
iteration = 0
while not flag:
    valid = flag
    iteration ++
    for i in range(len(matrixB)):
        new[i] = matrixB[i]
        for j in range(len(matrixB)):
            if i != j:
        new[i] -= SFCalc((matrixA[i][j]) * (old[j]),SignificantFigures)
        relative_error = abs((new[i]) - (old[i])) /abs(new[i])
        if relative_error > tolerance
            flag = False
    old = new[:]
```

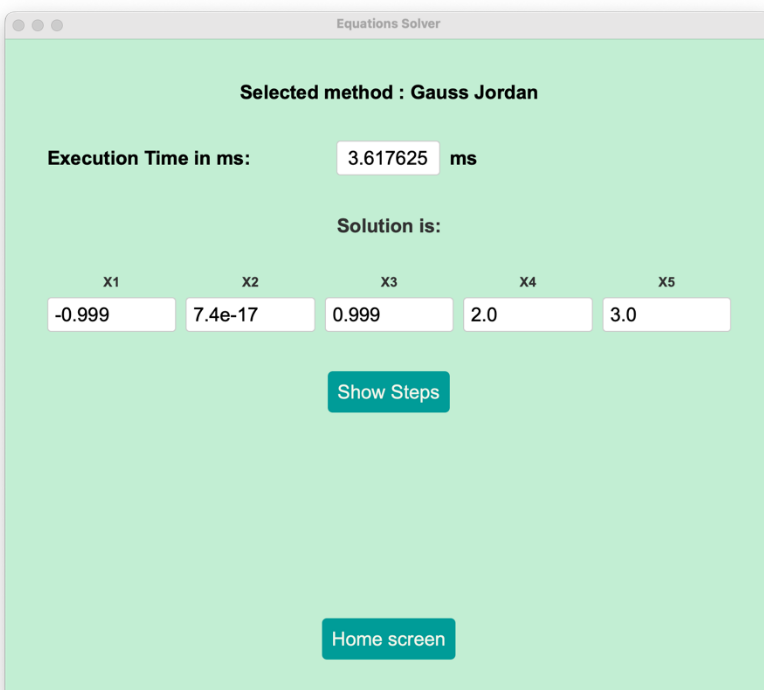
- **Gauss Seidel:**

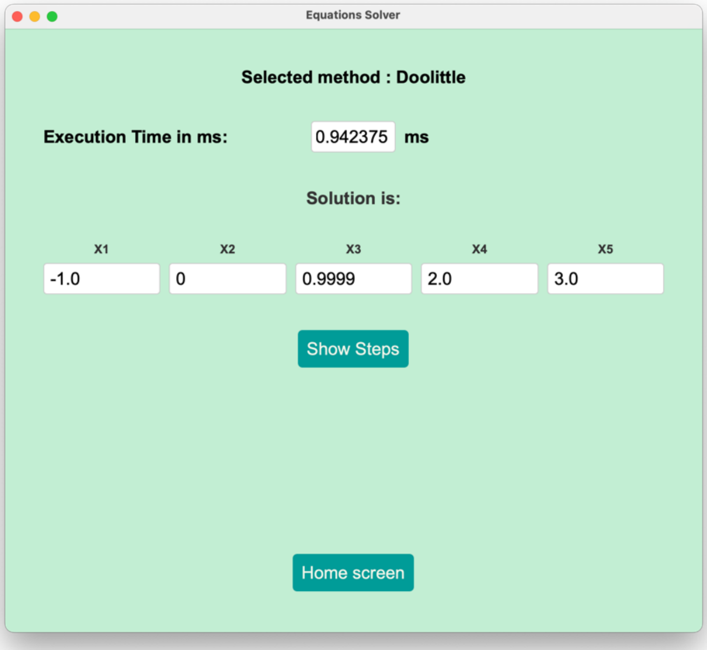
```
#Solve with iterations
for t = 1 to iterations :
    for i = 1 to Length(matrixB) :
        sum = matrixB[i]
        for j = 1 to Length(matrixB) :
            if i ≠ j :
                sum -= SFCalc(matrixA[i][j] * new[j],significant_figures)
            solution[i] = SFCalc(sum / matrixA[i][i],significant_figures)
    result = new
```

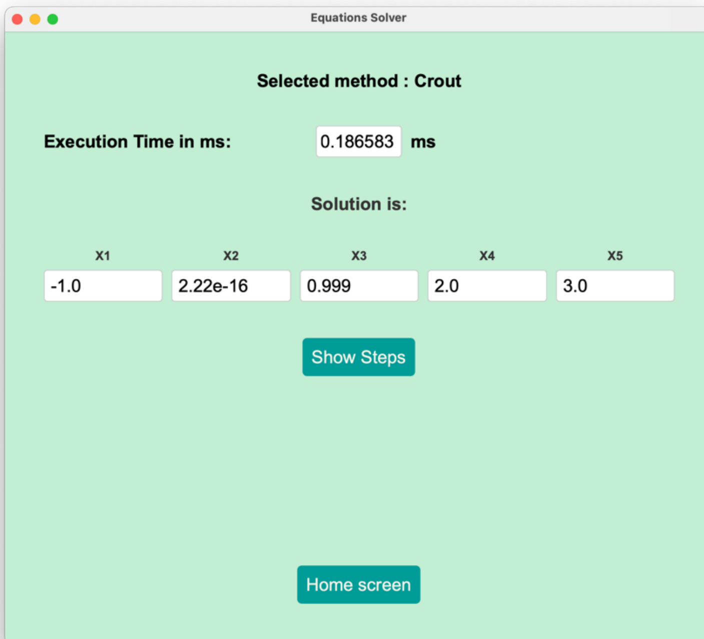
```
#Solve with specified tolerance
tolerance = tolerance / 100
valid = False
iteration = 0
old = initial_guess
while not valid :
    valid = True
    iteration_count ++1
    for i = 1 to Length(matrixB) :
        sum = matrixB[i]
        for j = 1 to Length(matrixB) :
            if i != j :
                sum -= SFCalc(matrixA[i][j] * new[j],significant_figures)
            new[i] = SFCalc(sum / matrixA[i][i],significant_figures)
            relative_error = l(new[i] - old[i]) / new[i]l
            if relative_error > tolerance :
                valid = False
        old = new
    result =solution
```

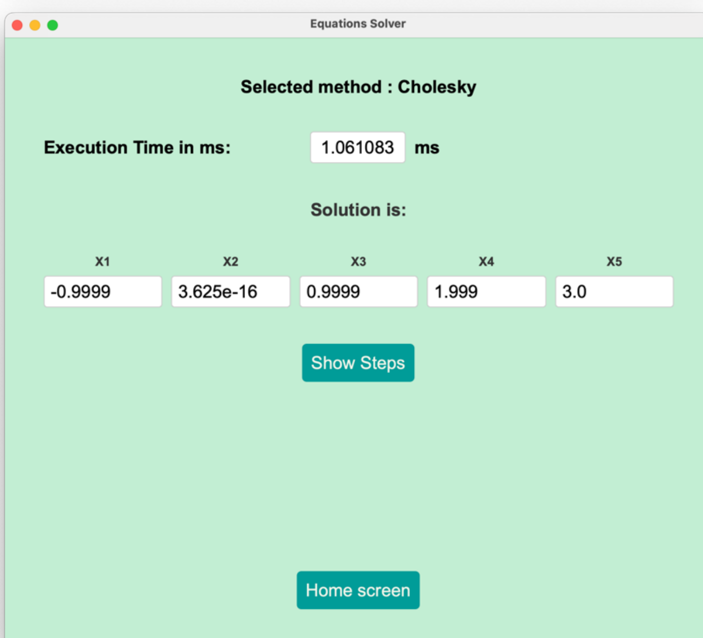
- # Test Cases
  - ## Test Case 1





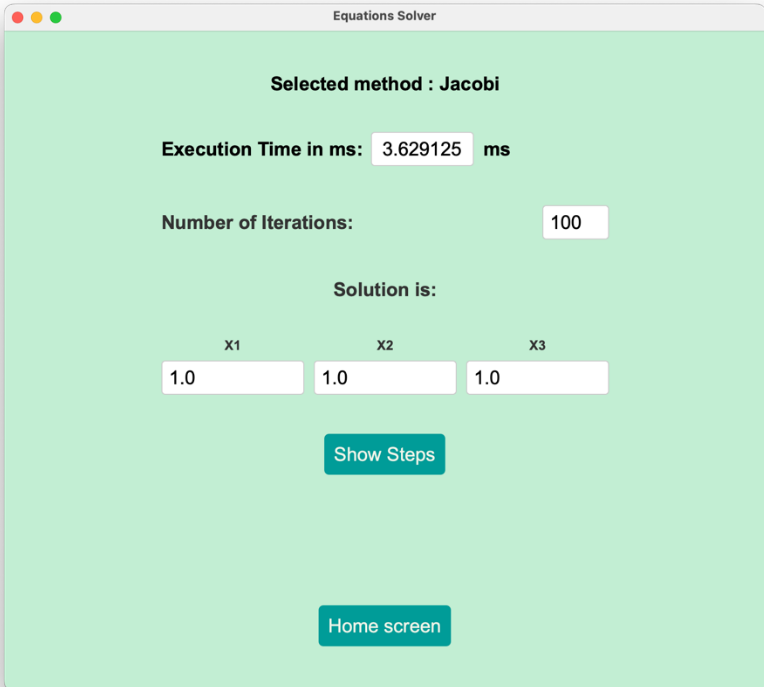

Equations Solver
Selected method : Doolittle
Execution Time in ms:
0.942375
ms
Solution is:
X1
X2
X3
X4
X5
-1.0
0
0.9999
2.0
3.0
Show Steps
Home screen


Equations Solver
Selected method : Crout
Execution Time in ms:
0.186583
ms
Solution is:
X1
X2
X3
X4
X5
-1.0
2.22e-16
0.999
2.0
3.0
Show Steps
Home screen

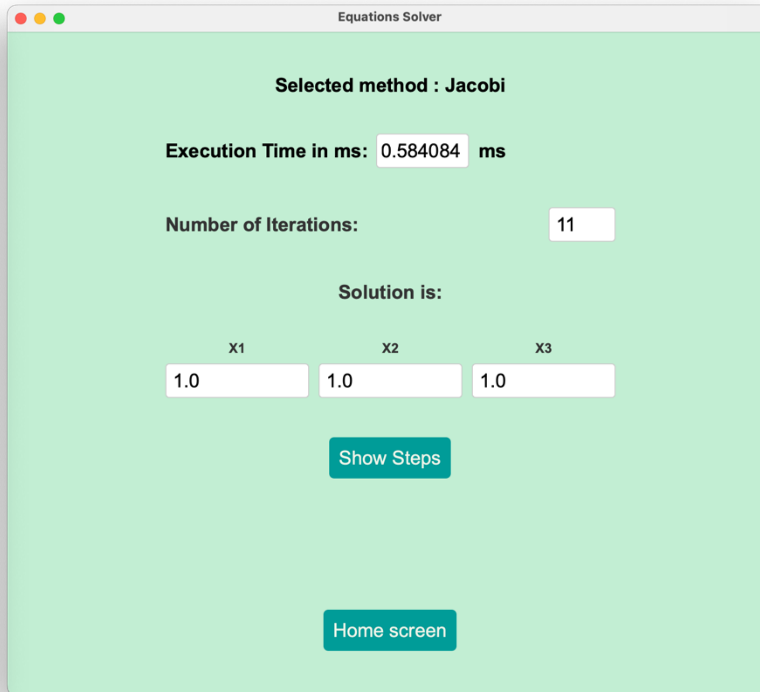

Equations Solver
Selected method : Cholesky
Execution Time in ms:
1.061083
ms
Solution is:
X1
X2
X3
X4
X5
-0.9999
3.625e-16
0.9999
1.999
3.0
Show Steps
Home screen

- ## Test Case 2

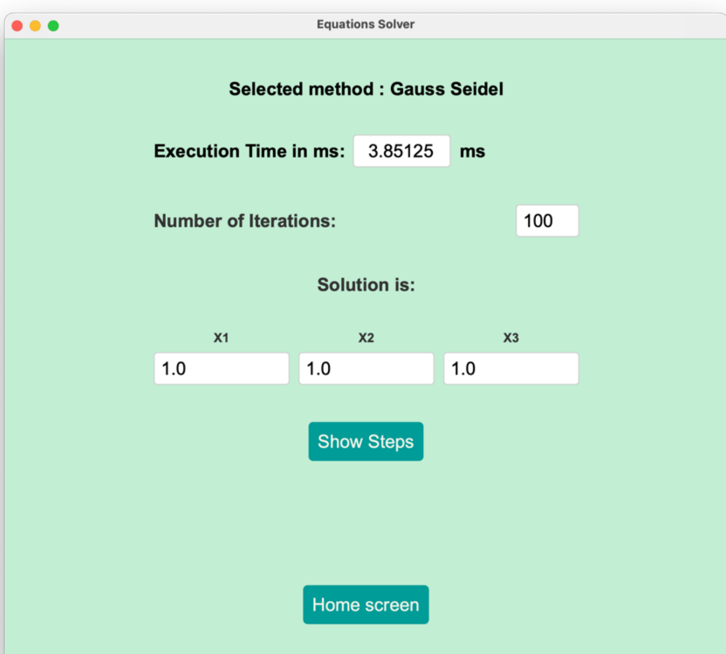
For 100 iterations



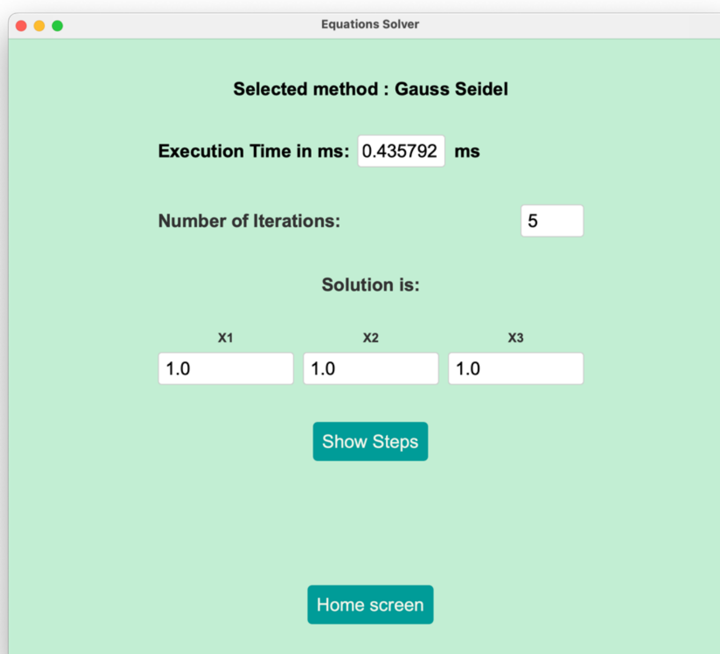
For 0.00001 relative error



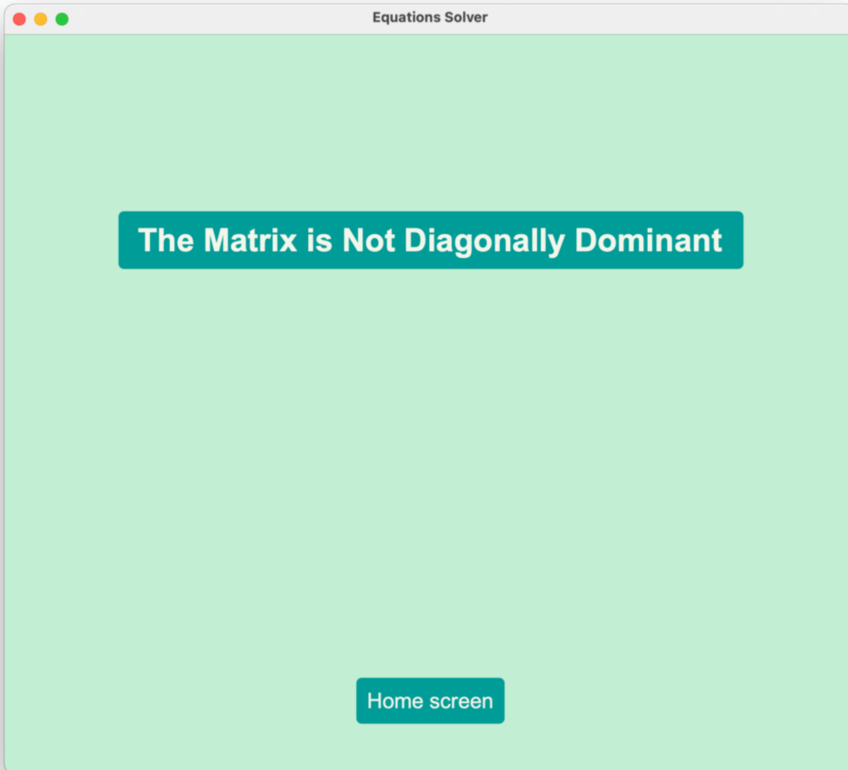
For 100 iterations



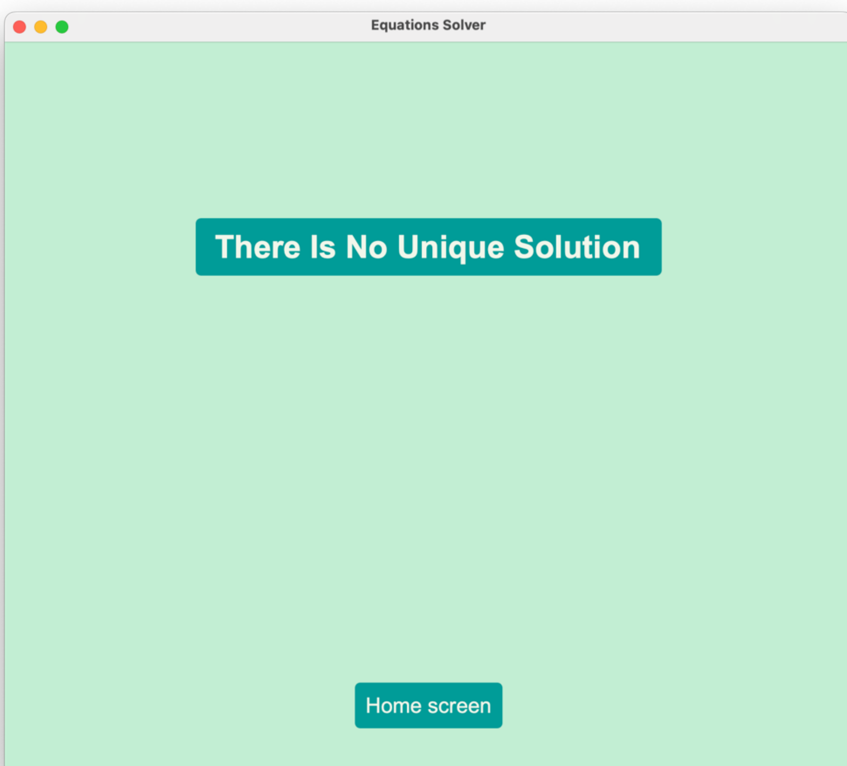
For 0.00001 relative error



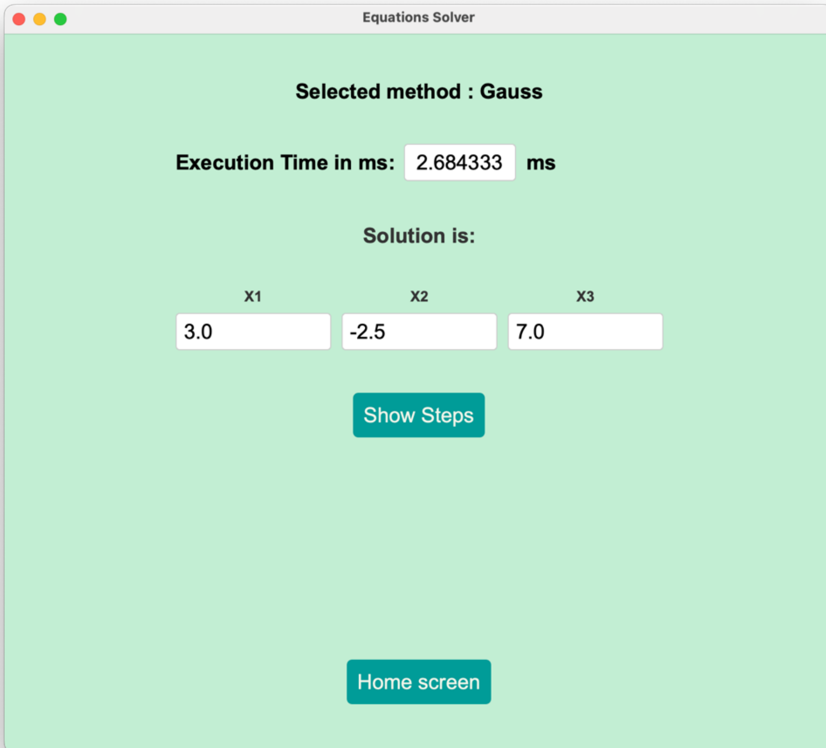
- **Test Case 3**



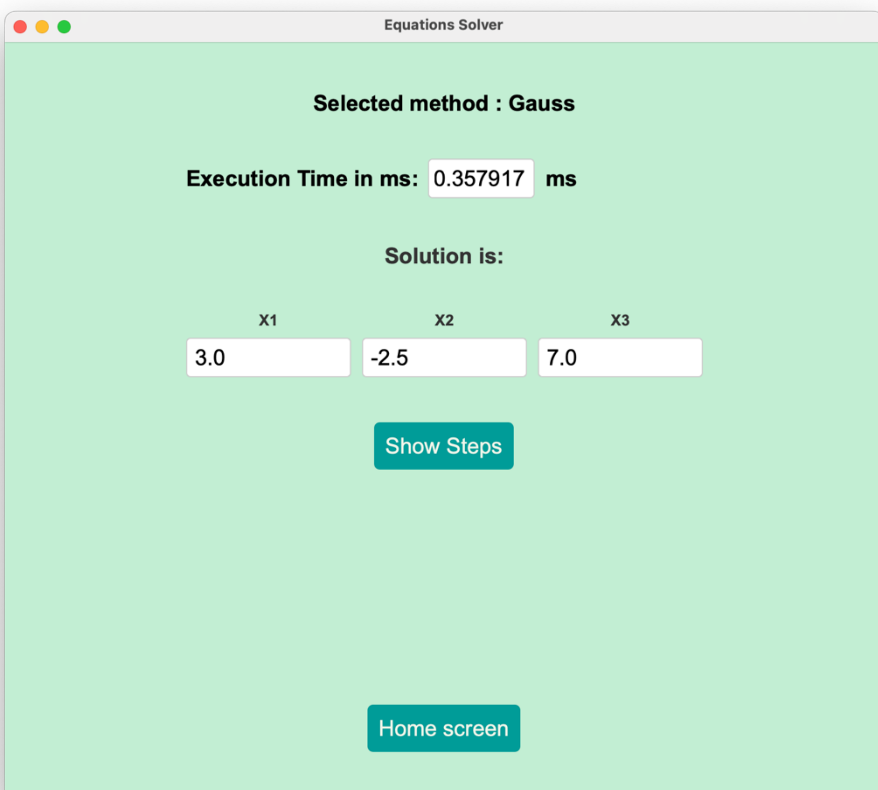
- **Test Case 4**



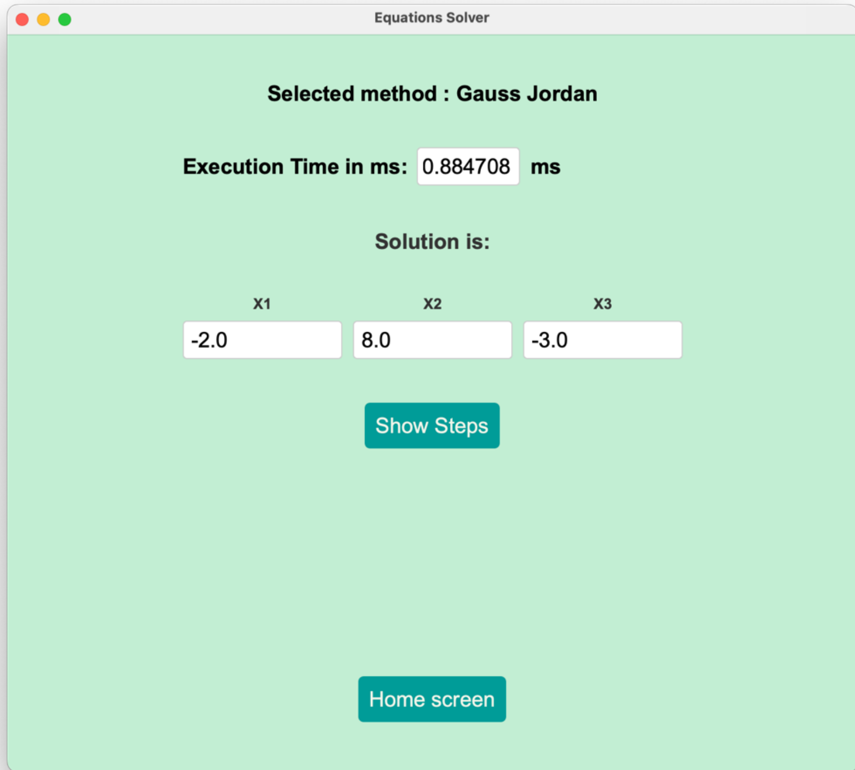
- **Test Case 5**
  For Precision = 6

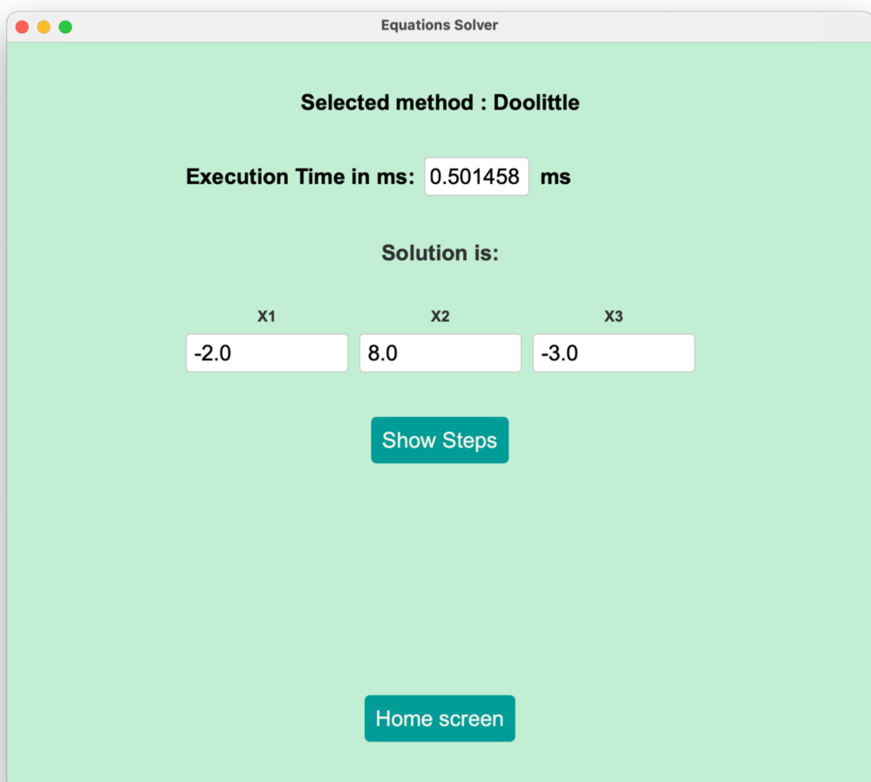


For Precision=3

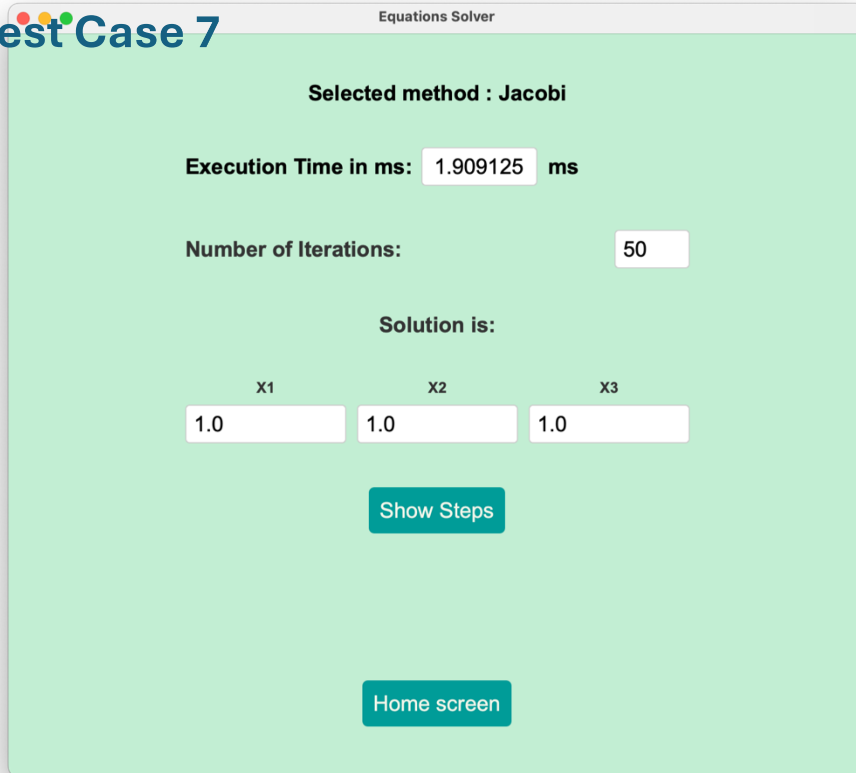


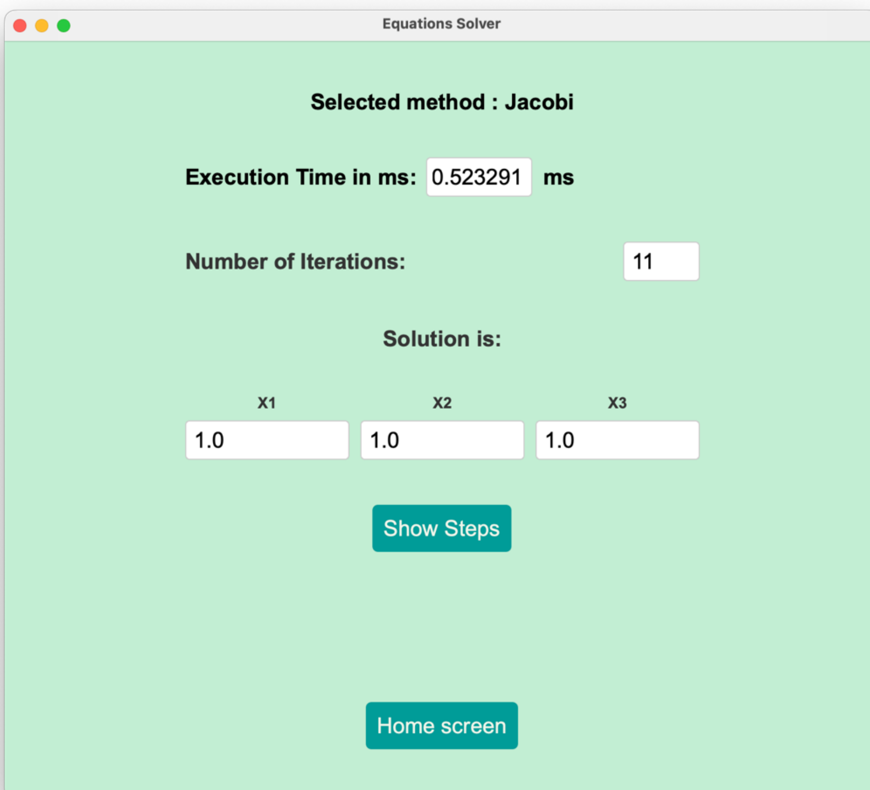
- ## Test Case 6


Equations Solver
Selected method : Gauss Jordan
Execution Time in ms: 0.884708 ms
Solution is:
X1
X2
X3
-2.0
8.0
-3.0
Show Steps
Home screen


Equations Solver
Selected method : Doolittle
Execution Time in ms: 0.501458 ms
Solution is:
X1
X2
X3
-2.0
8.0
-3.0
Show Steps
Home screen

- ## Test Case 7


Equations Solver
Selected method : Jacobi
Execution Time in ms: 1.909125 ms
Number of Iterations: 50
Solution is:
X1
X2
X3
1.0
1.0
1.0
Show Steps
Home screen

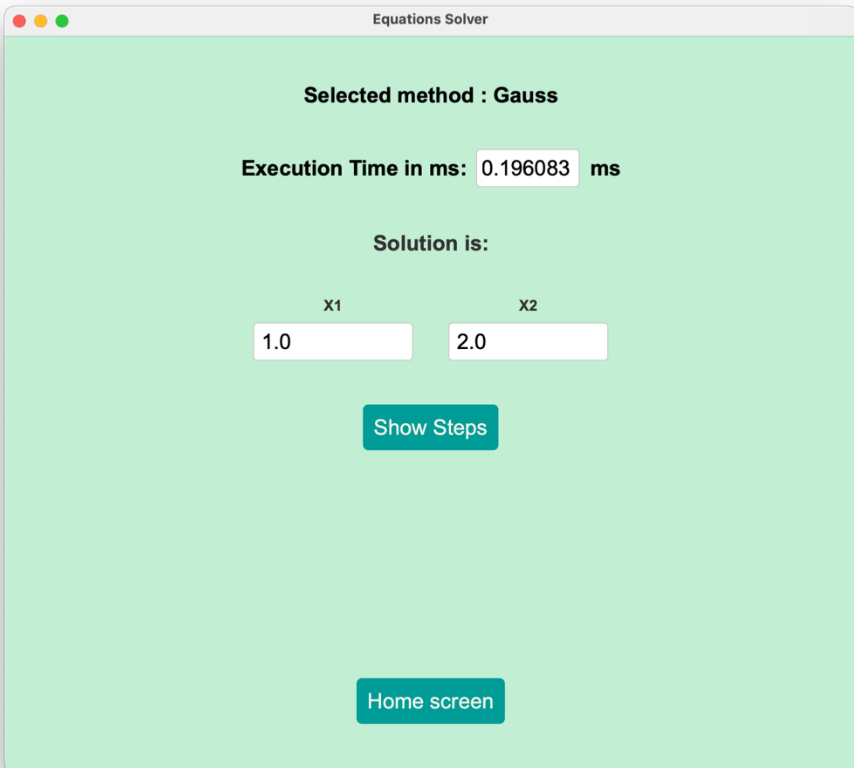

Equations Solver
Selected method : Jacobi
Execution Time in ms: 0.523291 ms
Number of Iterations: 11
Solution is:
X1
X2
X3
1.0
1.0
1.0
Show Steps
Home screen

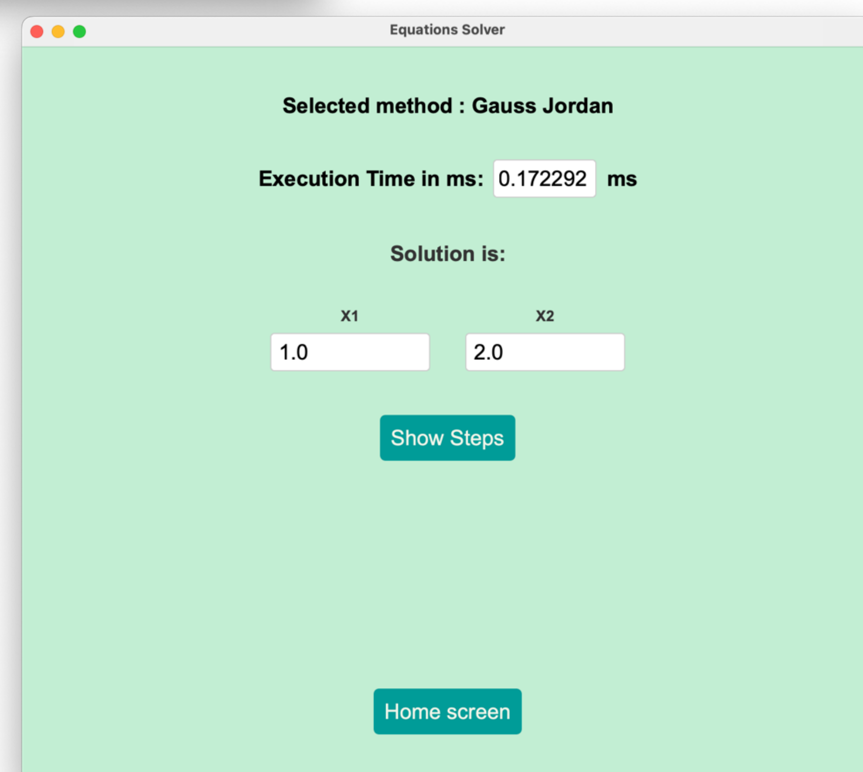
- **Comparison Between Methods**

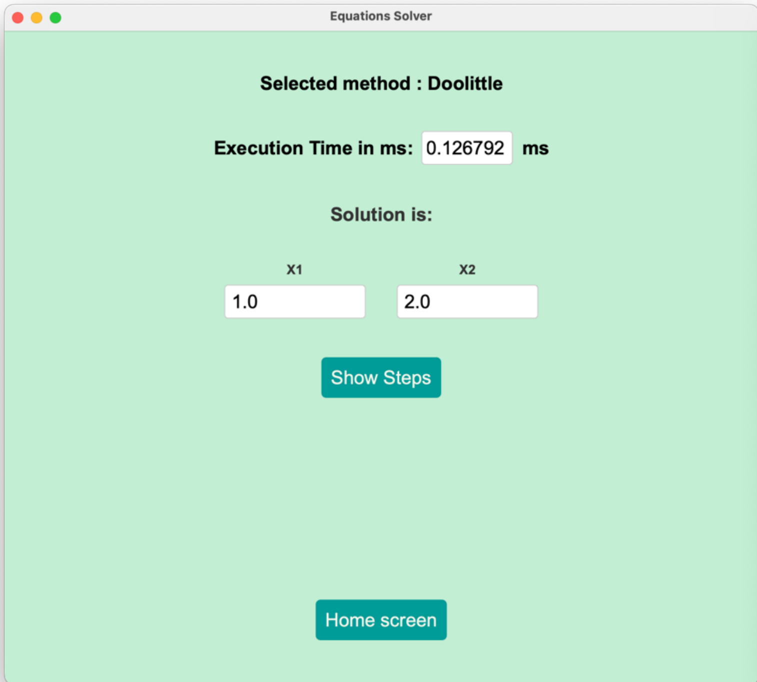
For this system

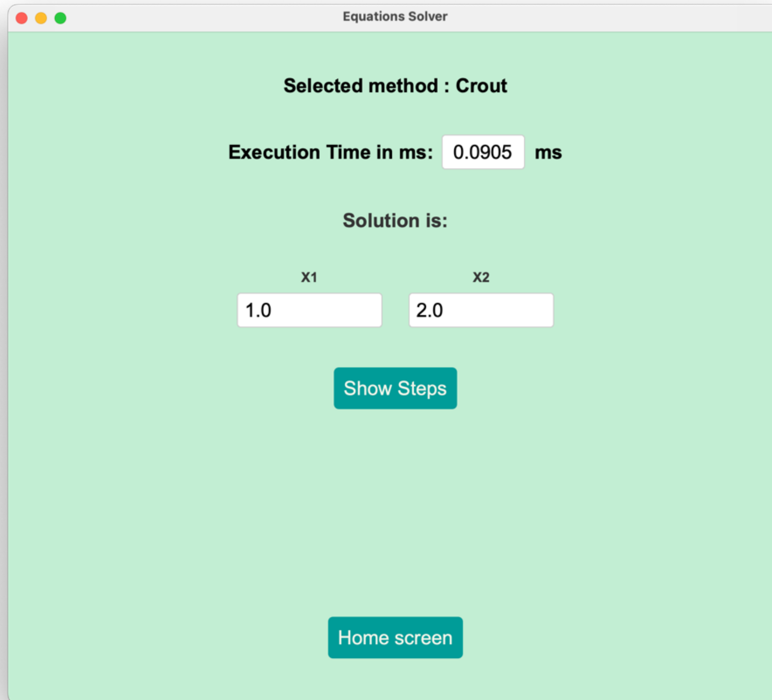
5X3 + 3X2 =11

X1 + X2 = 5



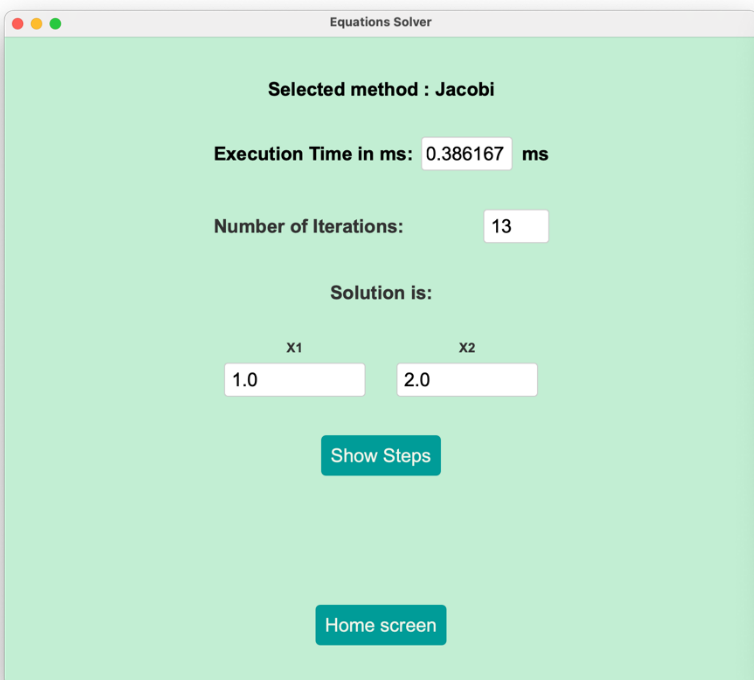


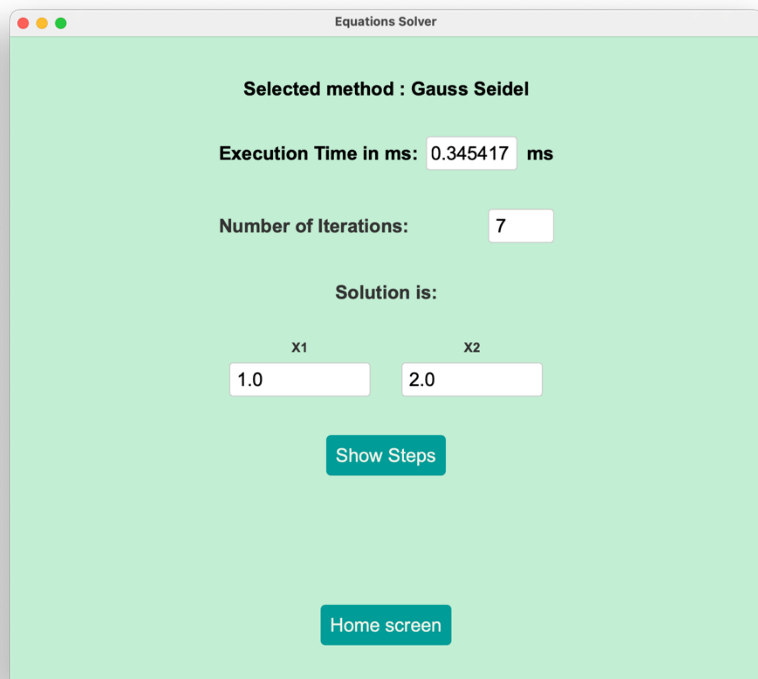




For relative erorr 0.1

For relative erorr 0.1





Used Data Stractures

- **Stack to GUI screens**
- **Array to Store the matrix**