Numerical Analysis

Project II - Solving systems of linear equations

Made By

1- Muhamad Hassan El-Hassan Oraby 6155

2- Muhamd Said Hassan 6162

3- Abdul-Rahman Muhamad Attia 6262

Codes:

Gauss Elimination Method:

## 1.1- Pseudocode for Gauss Elimination :

1. Start2. Input the Augmented Coefficients Matrix (A):For i = 1 to nFor j = 1 to n+1Read Ai,jNext jNext i3. Apply Gauss Elimination on Matrix A:For i = 1 to n-1If Ai,i = 0Print "Mathematical Error!"StopEnd IfFor j = i+1 to nRatio = Aj,i/Ai,iFor k = 1 to n+1Aj,k = Aj,k - Ratio \* Ai,kNext kNext jNext i4. Obtaining Solution by Back Substitution:Xn = An,n+1/An,nFor i = n-1 to 1 (Step: -1)Xi = Ai,n+1For j = i+1 to nXi = Xi - Ai,j \* XjNext jXi = Xi/Ai,iNext i5. Display Solution:For i = 1 to nPrint XiNext i6. Stop

## 1.2- Code for Gauss Elimination :

##########################################################

def gauss\_elimination(self):

# forward elimination

# for every unknown ( a , b ,c ...)

for k in range(0, self.order):

# for each equation

for i in range(k + 1, self.order + 1):

mul\_factor = self.equations\_matrix[i, k] / self.equations\_matrix[k, k]

# for every element in this equation

for j in range(k + 1, self.order + 2):

# start with j = 0 in case you want to get an upper triangular matrix

self.equations\_matrix[i, j] = self.equations\_matrix[i, j] - (

mul\_factor \* self.equations\_matrix[k, j])

# backward elimination

n = self.order

self.solutions[n] = self.equations\_matrix[n, n + 1] / self.equations\_matrix[n, n]

# for each equation

for i in range(n - 1, -1, -1):

sum\_a = self.equations\_matrix[i, n + 1] # b of i

for j in range(i + 1, n + 1):

sum\_a = sum\_a - (self.equations\_matrix[i, j] \* self.solutions[j])

self.solutions[i] = sum\_a / self.equations\_matrix[i, i]

##########################################################

## Gauss Jordan Method:

## 1.1- Pseudocode for Gauss Jorden:

1. Start2. Input the Augmented Coefficients Matrix (A):For i = 1 to nFor j = 1 to n+1Read Ai,jNext jNext i3. Apply Gauss Jordan Elimination on Matrix A:For i = 1 to nIf Ai,i = 0Print "Mathematical Error!"StopEnd IfFor j = 1 to nIf i ≠ j Ratio = Aj,i/Ai,iFor k = 1 to n+1Aj,k = Aj,k - Ratio \* Ai,kNext kEnd IfNext jNext i4. Obtaining Solution:For i = 1 to n Xi = Ai,n+1/Ai,iNext i5. Display Solution:For i = 1 to nPrint XiNext i6. Stop

## 1.2- Code for Gauss Jorden :

##########################################################

def gauss\_elimination(self):

# forward elimination

# for every unknown ( a , b ,c ...)

for k in range(0, self.order):

# for each equation

for i in range(k + 1, self.order + 1):

mul\_factor = self.equations\_matrix[i, k] / self.equations\_matrix[k, k]

# for every element in this equation

for j in range(k + 1, self.order + 2):

# start with j = 0 in case you want to get an upper triangular matrix

self.equations\_matrix[i, j] = self.equations\_matrix[i, j] - (

mul\_factor \* self.equations\_matrix[k, j])

# backward elimination

n = self.order

self.solutions[n] = self.equations\_matrix[n, n + 1] / self.equations\_matrix[n, n]

# for each equation

for i in range(n - 1, -1, -1):

sum\_a = self.equations\_matrix[i, n + 1] # b of i

for j in range(i + 1, n + 1):

sum\_a = sum\_a - (self.equations\_matrix[i, j] \* self.solutions[j])

self.solutions[i] = sum\_a / self.equations\_matrix[i, i]

##########################################################

## LU Decomposition Method:

## 1.1- Pseudocode for LU Decomposition:

// Assume arrays start with index 1 instead of 0.

// a: Coef. of matrix A; 2-D array. Upon successful

//

completion, it contains the coefficients of

//

both L and U.

// b: Coef. of vector b; 1-D array

// n: Dimension of the system of equations

// x: Coef. of vector x (to store the solution)

// tol: Tolerance; smallest possible scaled

//

pivot allowed.

// er: Pass back -1 if matrix is singular.

//

(Reference var.)

LUDecomp(a, b, n, x, tol, er) {

Declare s[n] // An n-element array for storing scaling factors

Declare o[n] // Use as indexes to pivot rows.

// o i or o(i) stores row number of the i th pivot row.

er = 0

Decompose(a, n, tol, o, s, er)

if (er != -1)

Substitute(a, o, n, b, x)

}

## 1.2- Code for LU Decomposition:

##########################################################

def lu\_decomposition(self):

# decompose

# for every unknown ( a , b ,c ...)

for k in range(0, self.order):

# for each equation

for i in range(k + 1, self.order + 1):

mul\_factor = self.A[i, k] / self.A[k, k]

self.A[i, k] = mul\_factor # L is the lower triangular matrix in A (instead of zeroes to save space)

# for every element in this equation

for j in range(k + 1, self.order + 1):

self.A[i, j] = self.A[i, j] - (mul\_factor \* self.A[k, j])

# forward elimination

# Convert B into D

n = self.order

for i in range(1, n + 1):

sum\_a = self.B[i]

for j in range(0, i):

sum\_a = sum\_a - (self.A[i, j] \* self.B[j])

self.B[i] = sum\_a

# backward elimination

self.solutions[n] = self.B[n] / self.A[n, n]

# for each equation

for i in range(n - 1, -1, -1):

sum\_a = 0

for j in range(i + 1, n + 1):

sum\_a = sum\_a + (self.A[i, j] \* self.solutions[j])

self.solutions[i] = (self.B[i] - sum\_a) / self.A[i, i]

##########################################################

## Guess Seidl Method:

## 1.1- Pseudocode for Guess Seidl:

1. Start2. Arrange given system of linear equations in diagonally dominant form3. Read tolerable error (e)4. Convert the first equation in terms of first variable, second equation in terms of second variable and so on. 5. Set initial guesses for x0, y0, z0 and so on6. Substitute value of y0, z0 ... from step 5 in first equation obtained from step 4 to calculate new value of x1. Use x1, z0, u0 .... in second equation obtained from step 4 to caluclate new value of y1. Similarly, use x1, y1, u0... to find new z1 and so on. 7. If| x0 - x1| > e and | y0 - y1| > e and | z0 - z1| > e and so on then goto step 98. Set x0=x1, y0=y1, z0=z1 and so on and goto step 69. Print value of x1, y1, z1 and so on10. Stop

## 1.2- Code for Guess Seidl:

##########################################################

def gauss\_seidel(self):

for iteration in range(1, self.max\_iterations + 1):

# for every unknown (x1 , x2 , x3 ...)

for i in range(0, self.order + 1):

# sum of the terms

sum\_x = self.B[i]

for j in range(0, self.order + 1):

if i != j:

sum\_x = sum\_x - (self.A[i, j] \* self.X[j])

self.X[i] = sum\_x / self.A[i, i]

##########################################################

Analysis:

After analysis we found that Guess Jorden is the fastest method,

then Guess Elimination is the second one,

then LU Decomposition is the least one.

Iterative method – Guess Seidl depends on no of Iterations

Problematic functions:

-In Guess Elimination & Guess Jorden problems have come out were:

Division by Zero

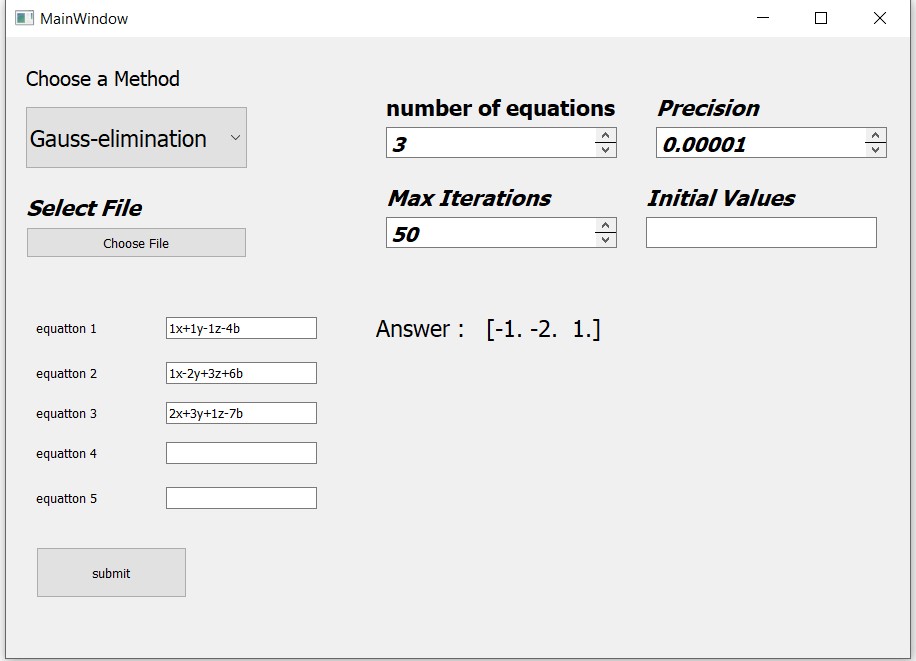
Round-off Errors

And we solved them using partial pivoting

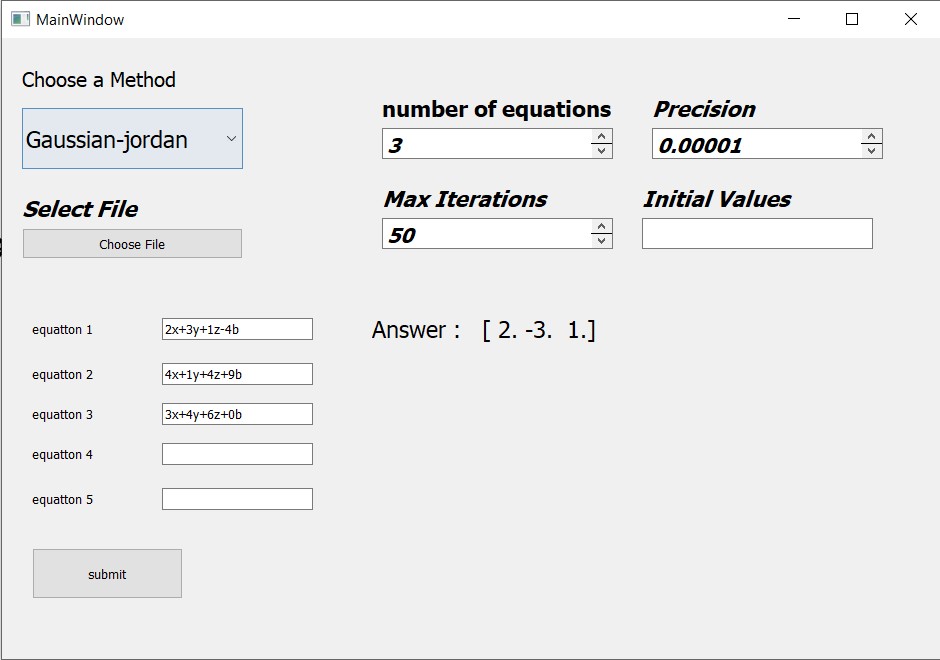
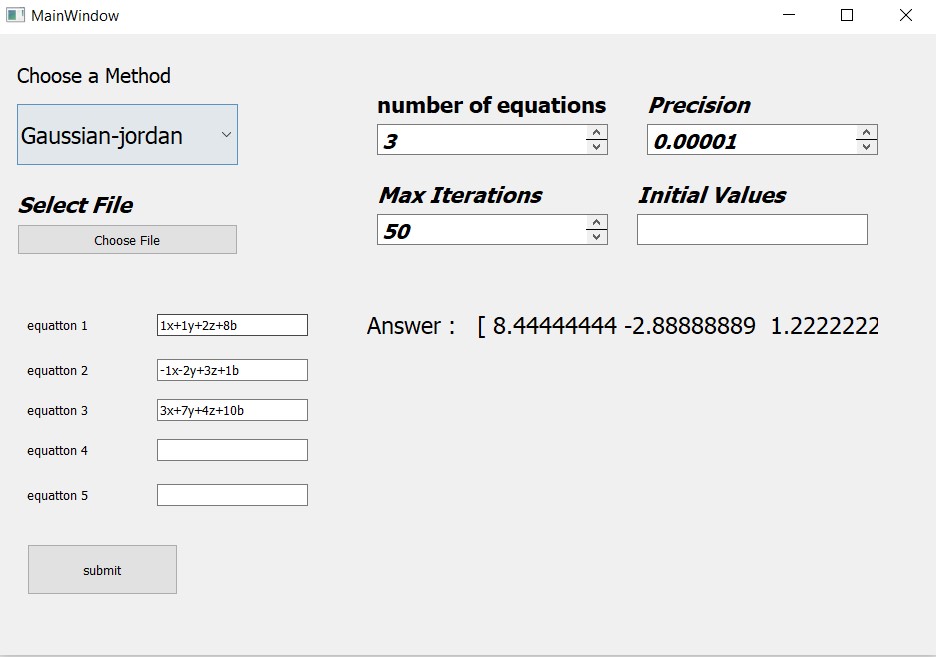
TEST CASES

ELIMINATIONGraphical user interface, application

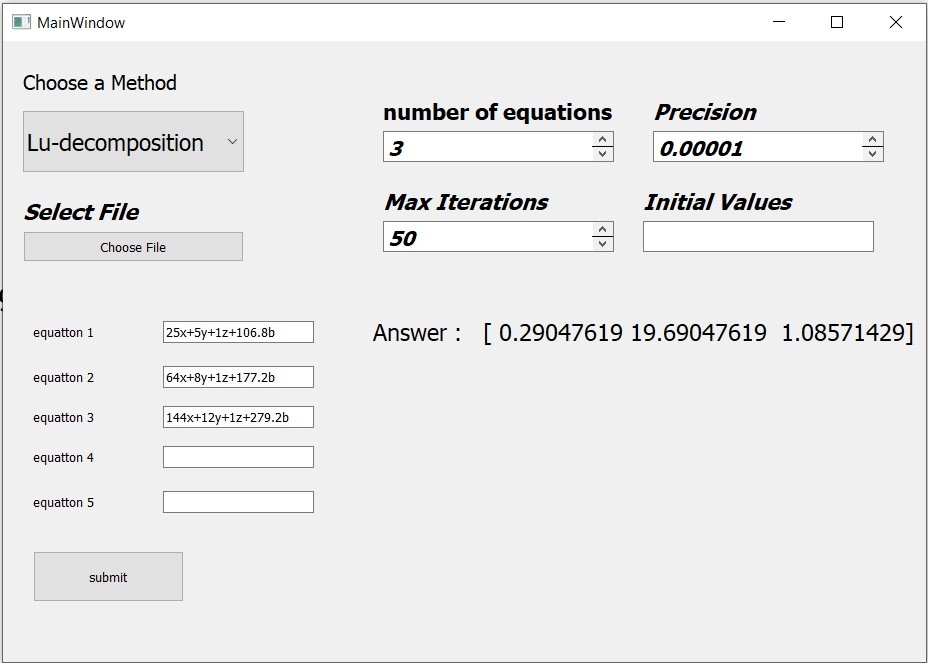
Description automatically generated

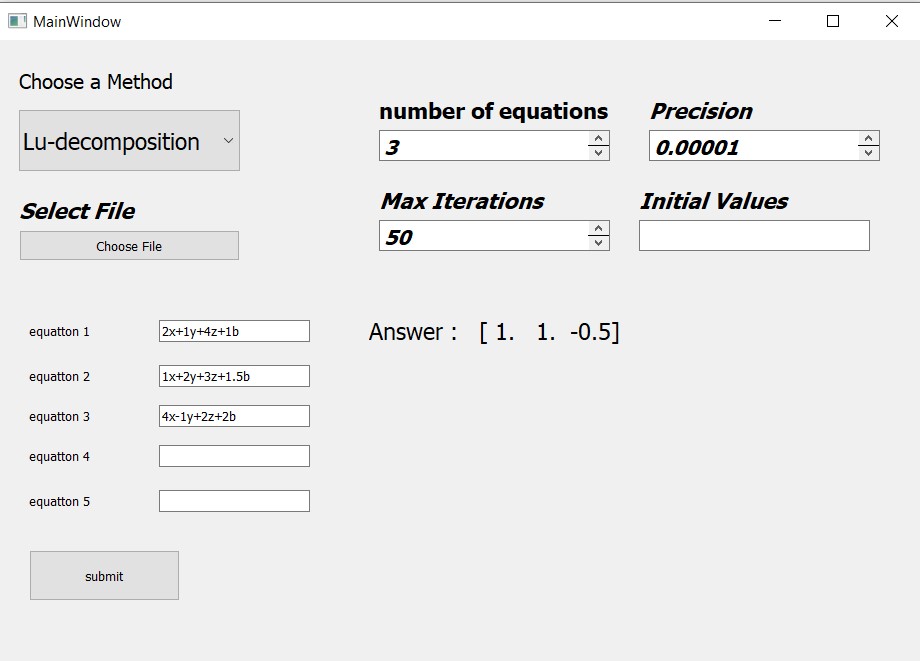


JORDAN

LU





SEIDEL