Numerical Final Project

Team Members:

Fares Mohamed Elsayed	6537
Aly Ahmed Aboelnasr	6511
Maged Magdy Zearban	6395

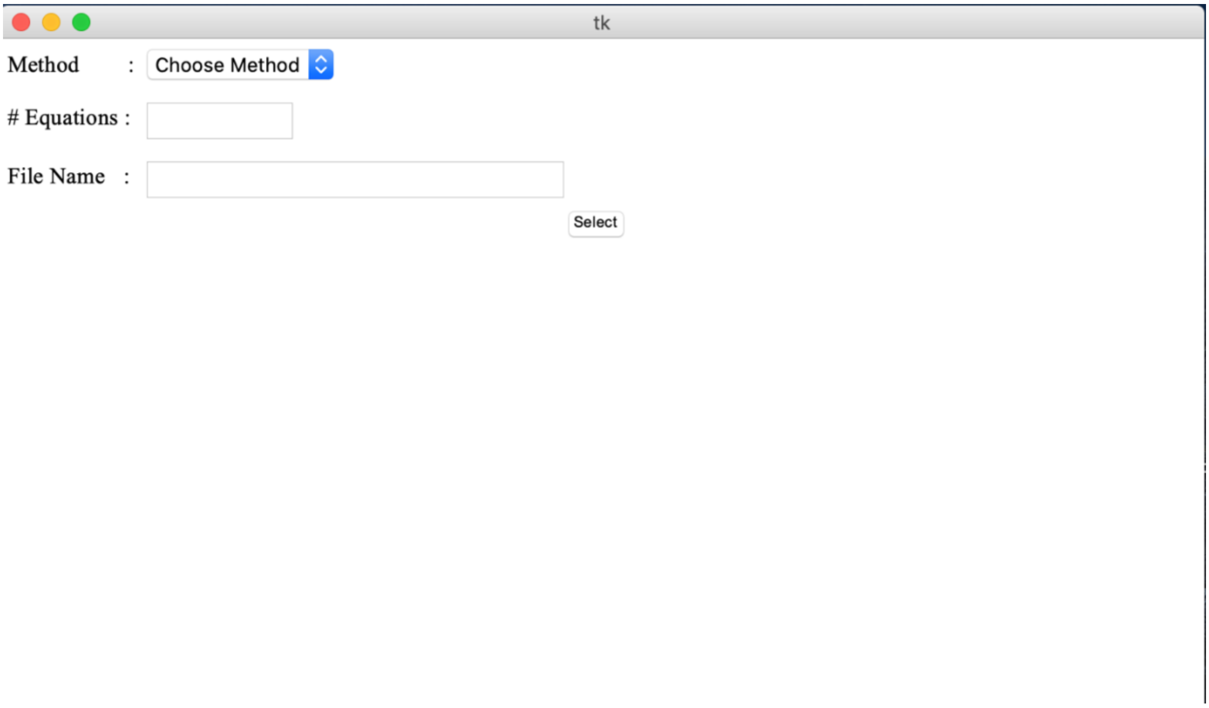
GUI is created with Tkinter using python to calculate Gaussian Elimination, Jordan, Seidel and LU decomposition.

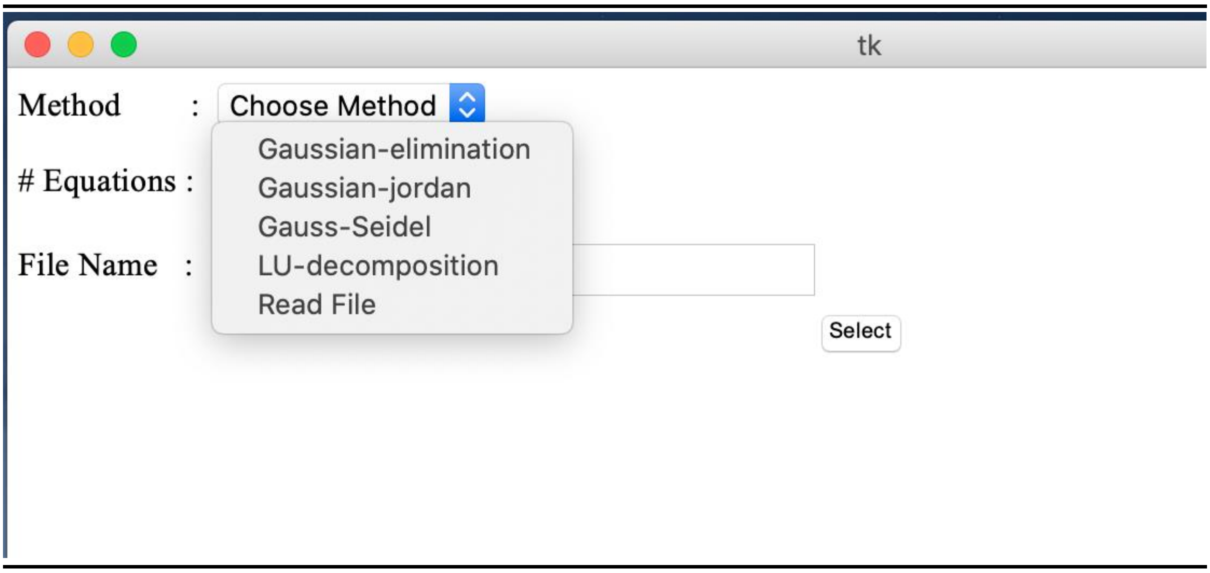
Firstly we choose method of calculation (one of the 4 functions) then enter number of equations, or we can just take the input as a text file to be read by just choosing Read File and enter the file name for example (Jordan.txt)
Final Results & error calculations are displayed in the GUI screen.
Steps of each iteration along with the final result are printed in the output file.

Four functions are implemented and their codes are shown in the screenshots below.

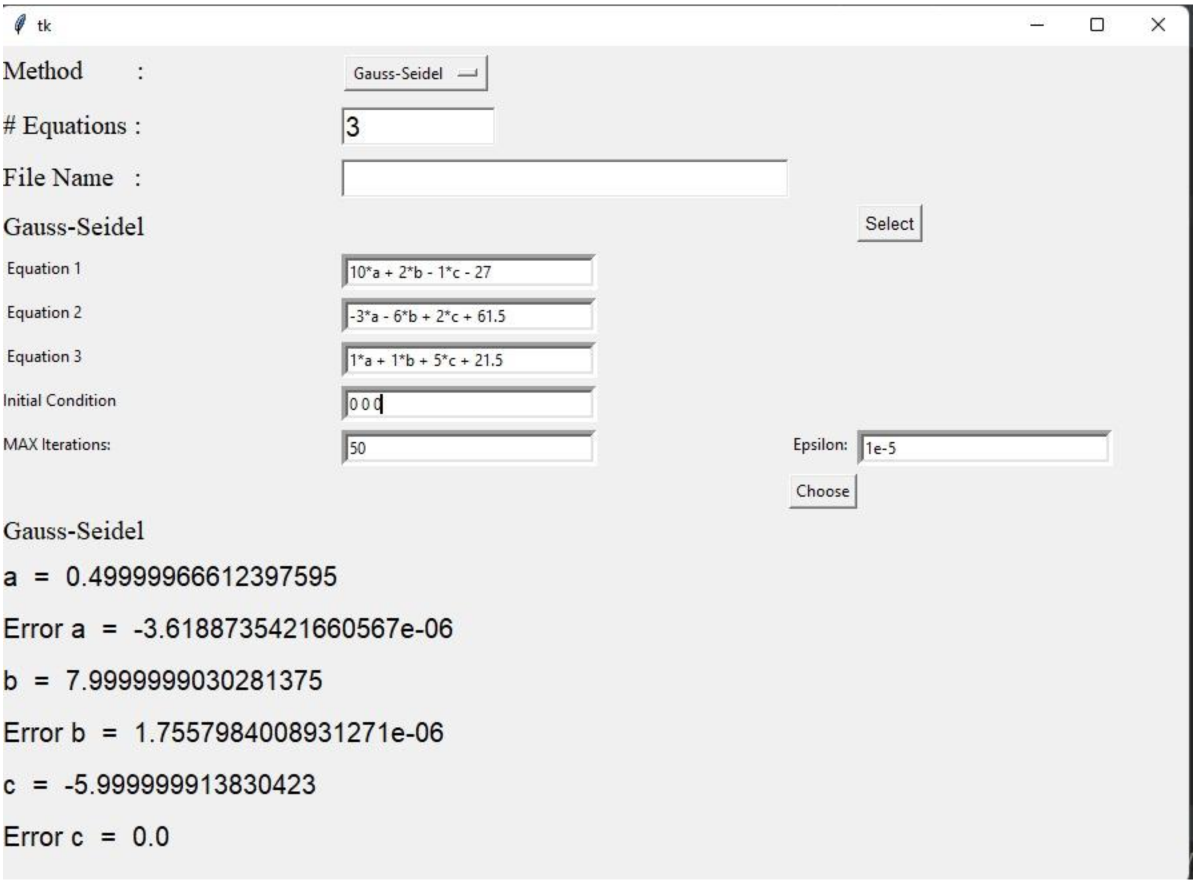
In the Video we tried to cover all the cases by showing one test for each method either by entering the equations manually or just read a text file containing the equation with the method type.

GUI Implementation :





Gaussian Seidel:



Method : Gauss-Seidel

Equations : 3

File Name :

Gauss-Seidel

Equation 1 10*a + 2*b - 1*c - 27

Equation 2 -3*a - 6*b + 2*c + 61.5

Equation 3 1*a + 1*b + 5*c + 21.5

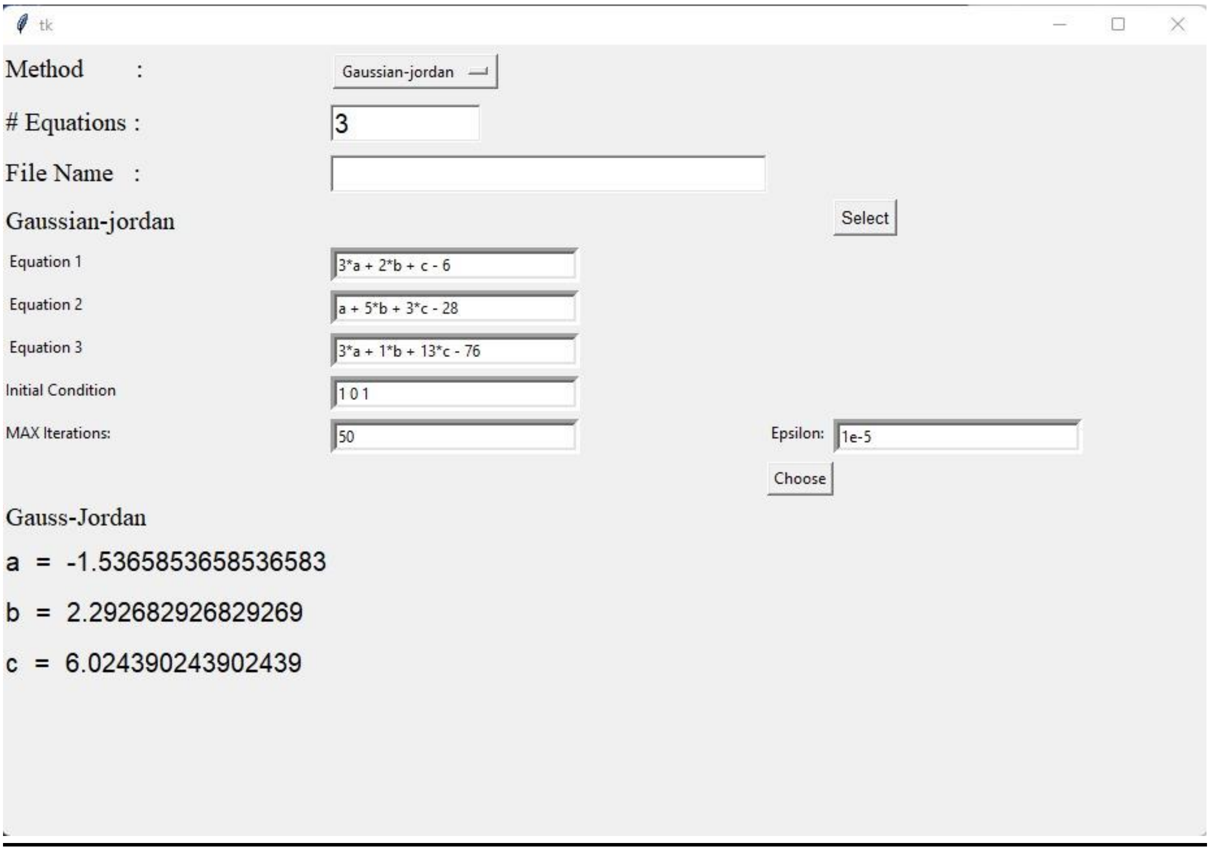
Initial Condition 0 0 0

MAX Iterations: 50

Epsilon: 1e-5

```
Gauss-Seidel
a = 0.49999966612397595
Error a = -3.6188735421660567e-06
b = 7.9999999030281375
Error b = 1.7557984008931271e-06
c = -5.999999913830423
Error c = 0.0
```

Gaussian Jordan:



Method : Gaussian-jordan

Equations : 3

File Name :

Gaussian-jordan

Equation 1 3*a + 2*b + c - 6

Equation 2 a + 5*b + 3*c - 28

Equation 3 3*a + 1*b + 13*c - 76

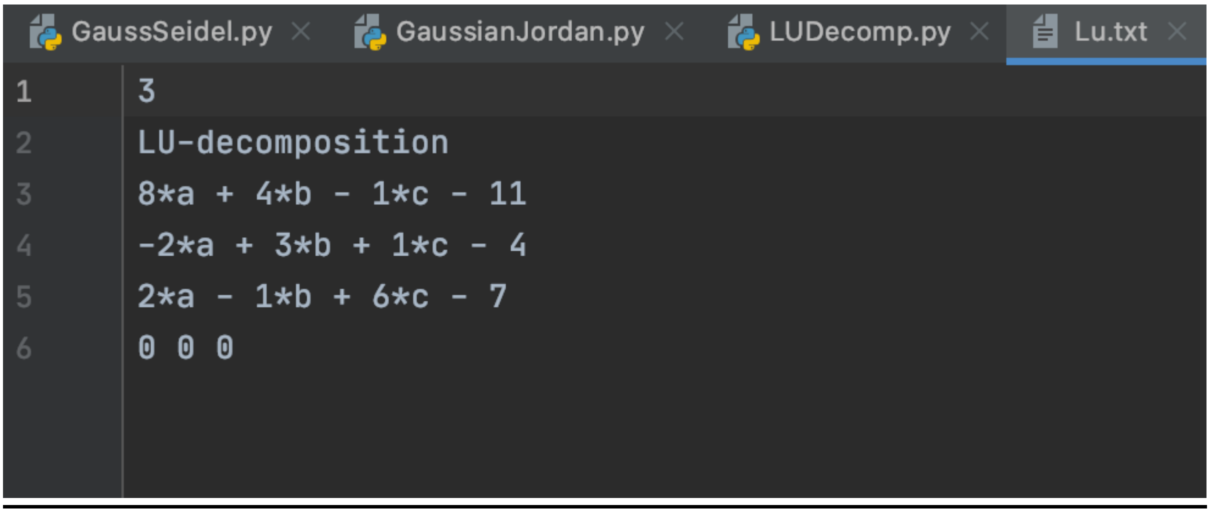
Initial Condition 1 0 1

MAX Iterations: 50

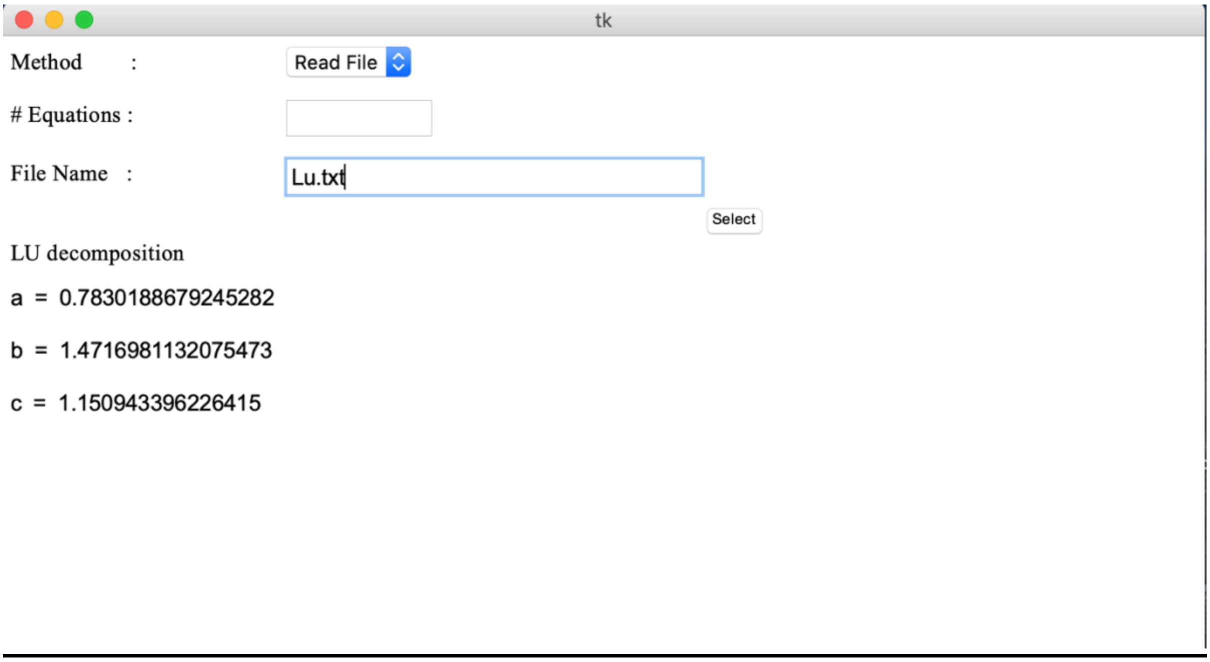
Epsilon: 1e-5

```
Gauss-Jordan
a  =  -1.5365853658536583
b  =  2.292682926829269
c  =  6.024390243902439
```

Reading LU equations from File and showing Output:



```
3
LU-decomposition
8*a + 4*b - 1*c - 11
-2*a + 3*b + 1*c - 4
2*a - 1*b + 6*c - 7
0 0 0
```



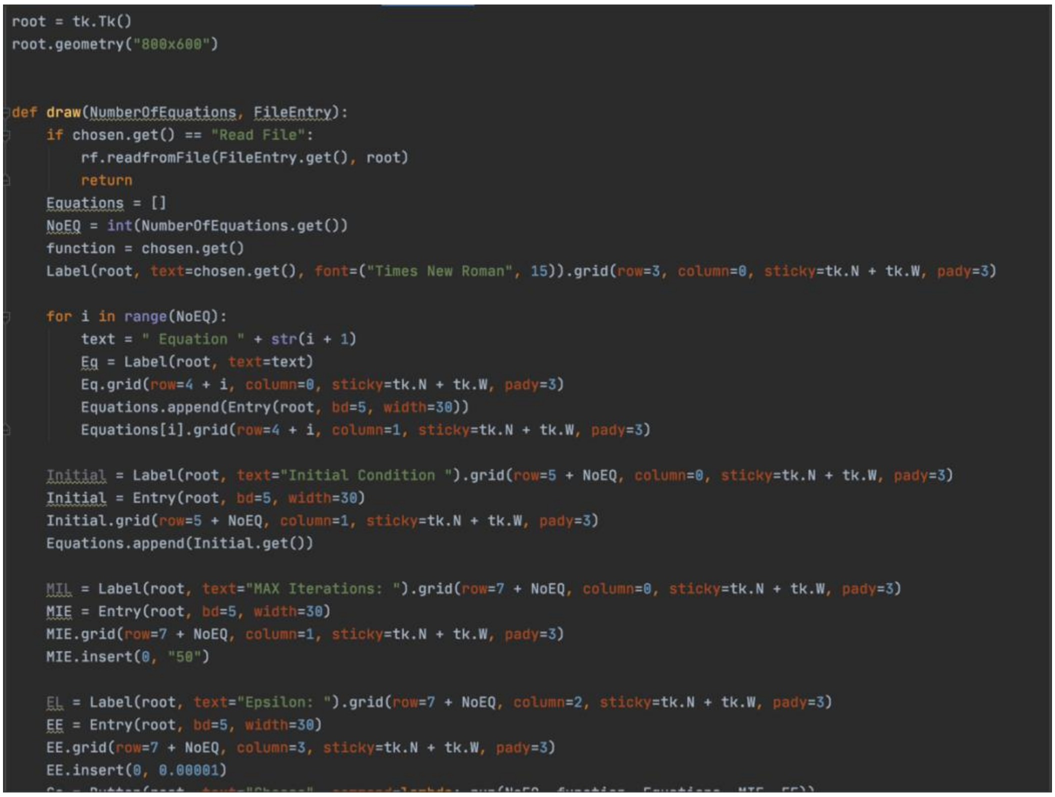
Method : Read File

Equations :

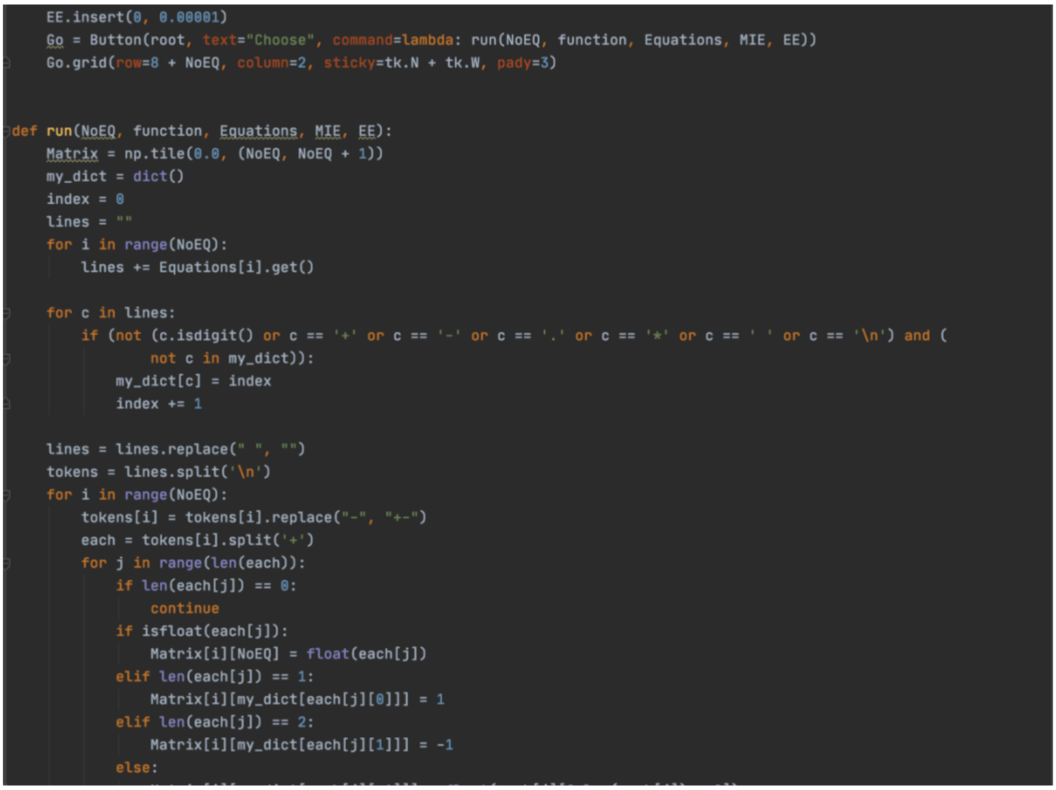
File Name : Lu.txt

```
LU decomposition
a = 0.7830188679245282
b = 1.4716981132075473
c = 1.150943396226415
```

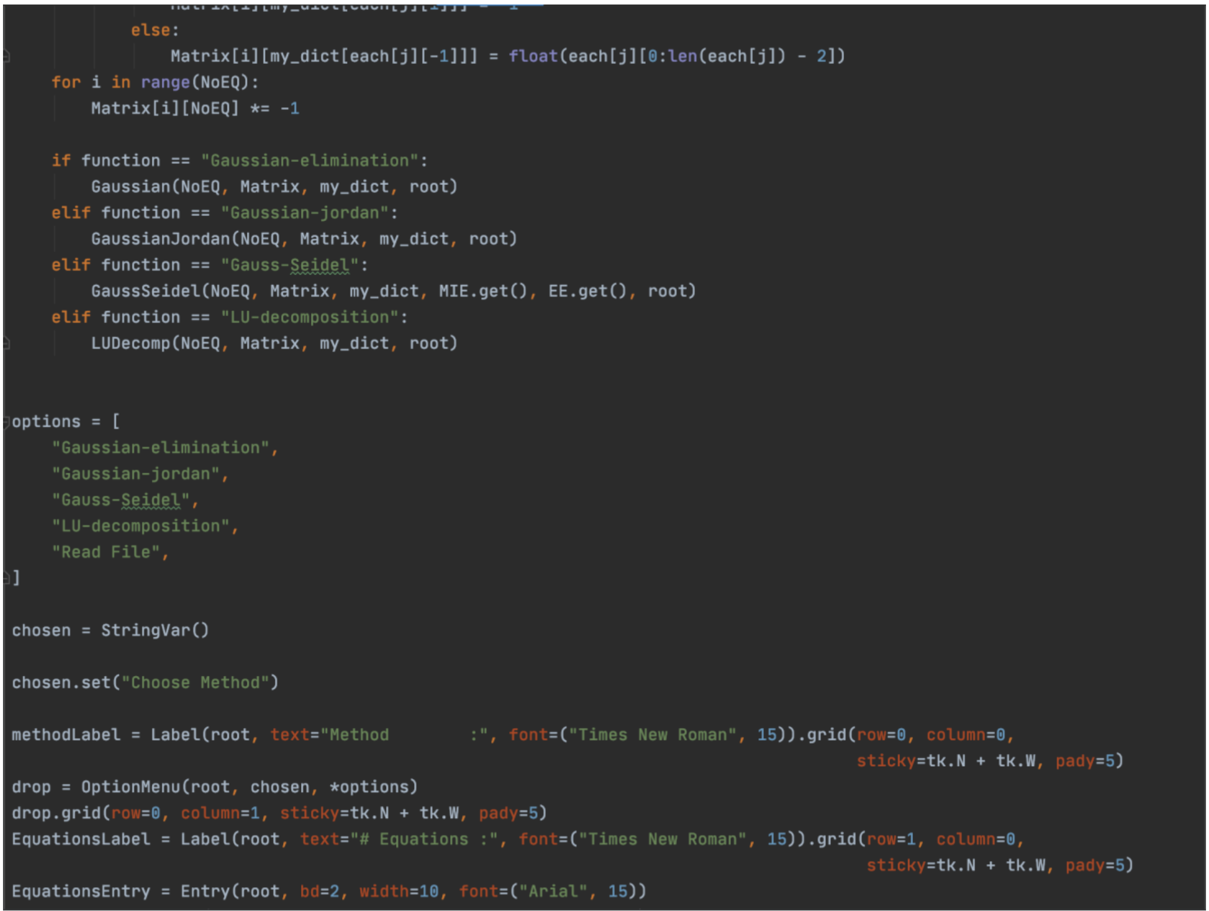
GUI Python Code:



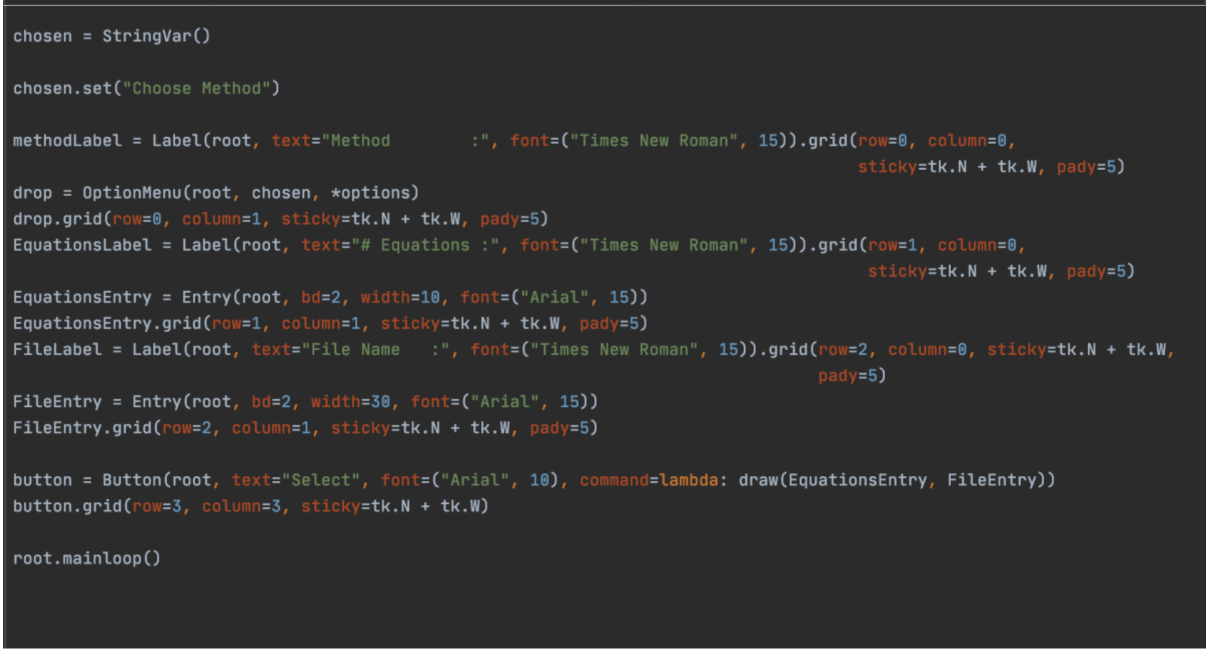
```
root = tk.Tk()
root.geometry("800x600")


def draw(NumberOfEquations, FileEntry):
    if chosen.get() == "Read File":
        rf.readfromFile(FileEntry.get(), root)
        return
    Equations = []
    NoEQ = int(NumberOfEquations.get())
    function = chosen.get()
    Label(root, text=chosen.get(), font=("Times New Roman", 15)).grid(row=3, column=0, sticky=tk.N + tk.W, pady=3)

    for i in range(NoEQ):
        text = " Equation " + str(i + 1)
        Eq = Label(root, text=text)
        Eq.grid(row=4 + i, column=0, sticky=tk.N + tk.W, pady=3)
        Equations.append(Entry(root, bd=5, width=30))
        Equations[i].grid(row=4 + i, column=1, sticky=tk.N + tk.W, pady=3)

    Initial = Label(root, text="Initial Condition ").grid(row=5 + NoEQ, column=0, sticky=tk.N + tk.W, pady=3)
    Initial = Entry(root, bd=5, width=30)
    Initial.grid(row=5 + NoEQ, column=1, sticky=tk.N + tk.W, pady=3)
    Equations.append(Initial.get())

    MIL = Label(root, text="MAX Iterations: ").grid(row=7 + NoEQ, column=0, sticky=tk.N + tk.W, pady=3)
    MIE = Entry(root, bd=5, width=30)
    MIE.grid(row=7 + NoEQ, column=1, sticky=tk.N + tk.W, pady=3)
    MIE.insert(0, "50")

    EL = Label(root, text="Epsilon: ").grid(row=7 + NoEQ, column=2, sticky=tk.N + tk.W, pady=3)
    EE = Entry(root, bd=5, width=30)
    EE.grid(row=7 + NoEQ, column=3, sticky=tk.N + tk.W, pady=3)
    EE.insert(0, 0.00001)
```



```
    EE.insert(0, 0.00001)
    Go = Button(root, text="Choose", command=lambda: run(NoEQ, function, Equations, MIE, EE))
    Go.grid(row=8 + NoEQ, column=2, sticky=tk.N + tk.W, pady=3)


def run(NoEQ, function, Equations, MIE, EE):
    Matrix = np.tile(0.0, (NoEQ, NoEQ + 1))
    my_dict = dict()
    index = 0
    lines = ""
    for i in range(NoEQ):
        lines += Equations[i].get()

    for c in lines:
        if (not (c.isdigit() or c == '+' or c == '-' or c == '.' or c == '*' or c == ' ' or c == '\n') and (
                not c in my_dict)):
            my_dict[c] = index
            index += 1

    lines = lines.replace(" ", "")
    tokens = lines.split('\n')
    for i in range(NoEQ):
        tokens[i] = tokens[i].replace("-", "+-")
        each = tokens[i].split('+')
        for j in range(len(each)):
            if len(each[j]) == 0:
                continue
            if isfloat(each[j]):
                Matrix[i][NoEQ] = float(each[j])
            elif len(each[j]) == 1:
                Matrix[i][my_dict[each[j][0]]] = 1
            elif len(each[j]) == 2:
                Matrix[i][my_dict[each[j][1]]] = -1
            else:
```

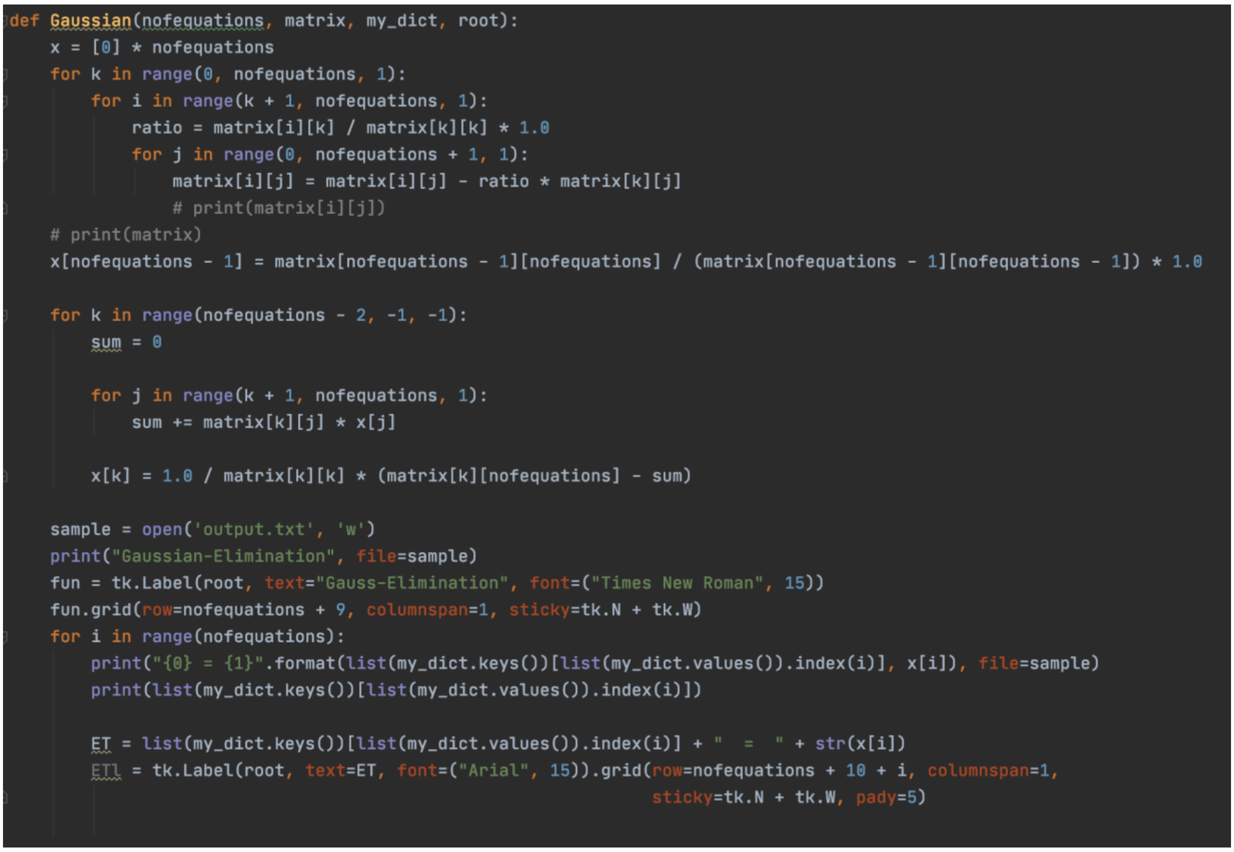


```
            else:
                Matrix[i][my_dict[each[j][-1]]] = float(each[j][0:len(each[j]) - 2])
    for i in range(NoEQ):
        Matrix[i][NoEQ] *= -1

    if function == "Gaussian-elimination":
        Gaussian(NoEQ, Matrix, my_dict, root)
    elif function == "Gaussian-jordan":
        GaussianJordan(NoEQ, Matrix, my_dict, root)
    elif function == "Gauss-Seidel":
        GaussSeidel(NoEQ, Matrix, my_dict, MIE.get(), EE.get(), root)
    elif function == "LU-decomposition":
        LUDecomp(NoEQ, Matrix, my_dict, root)


options = [
    "Gaussian-elimination",
    "Gaussian-jordan",
    "Gauss-Seidel",
    "LU-decomposition",
    "Read File",
]

chosen = StringVar()

chosen.set("Choose Method")

methodLabel = Label(root, text="Method        :", font=("Times New Roman", 15)).grid(row=0, column=0,
                                                                                     sticky=tk.N + tk.W, pady=5)
drop = OptionMenu(root, chosen, *options)
drop.grid(row=0, column=1, sticky=tk.N + tk.W, pady=5)
EquationsLabel = Label(root, text="# Equations :", font=("Times New Roman", 15)).grid(row=1, column=0,
                                                                                      sticky=tk.N + tk.W, pady=5)
EquationsEntry = Entry(root, bd=2, width=10, font=("Arial", 15))
```



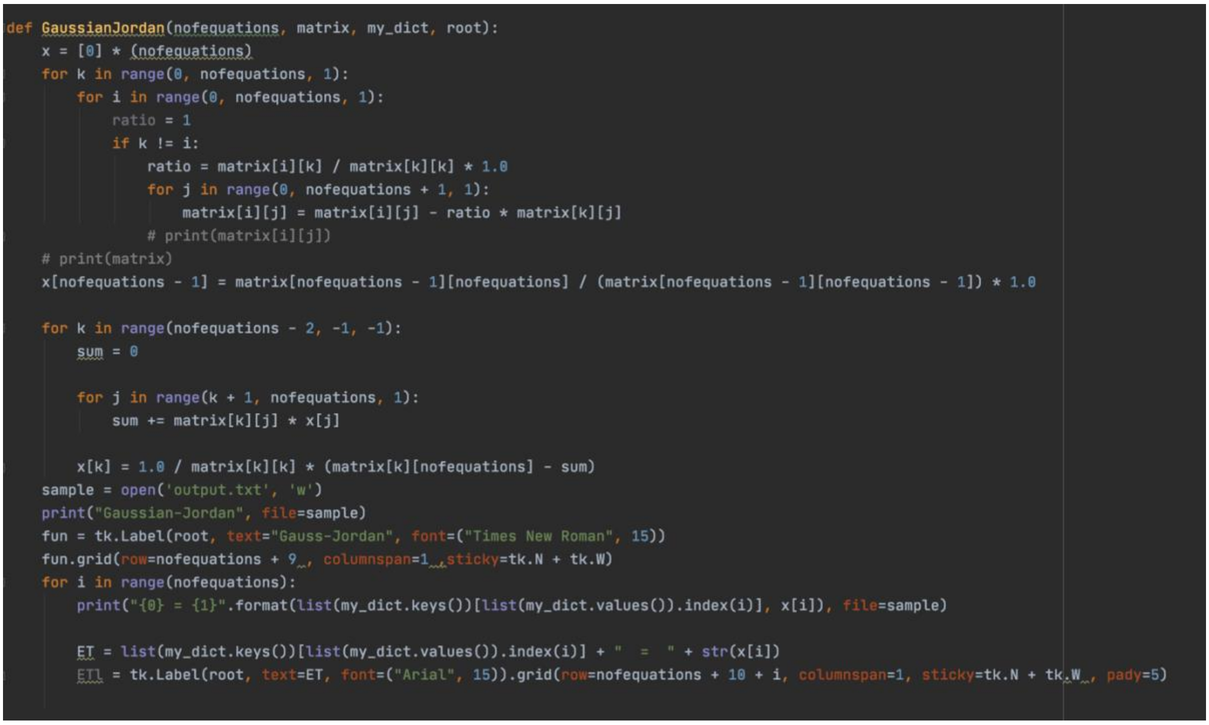
```
chosen = StringVar()

chosen.set("Choose Method")

methodLabel = Label(root, text="Method        :", font=("Times New Roman", 15)).grid(row=0, column=0,
                                                                                     sticky=tk.N + tk.W, pady=5)
drop = OptionMenu(root, chosen, *options)
drop.grid(row=0, column=1, sticky=tk.N + tk.W, pady=5)
EquationsLabel = Label(root, text="# Equations :", font=("Times New Roman", 15)).grid(row=1, column=0,
                                                                                      sticky=tk.N + tk.W, pady=5)
EquationsEntry = Entry(root, bd=2, width=10, font=("Arial", 15))
EquationsEntry.grid(row=1, column=1, sticky=tk.N + tk.W, pady=5)
FileLabel = Label(root, text="File Name   :", font=("Times New Roman", 15)).grid(row=2, column=0, sticky=tk.N + tk.W,
                                                                                 pady=5)
FileEntry = Entry(root, bd=2, width=30, font=("Arial", 15))
FileEntry.grid(row=2, column=1, sticky=tk.N + tk.W, pady=5)

button = Button(root, text="Select", font=("Arial", 10), command=lambda: draw(EquationsEntry, FileEntry))
button.grid(row=3, column=3, sticky=tk.N + tk.W)

root.mainloop()
```

Gaussian Elimination:



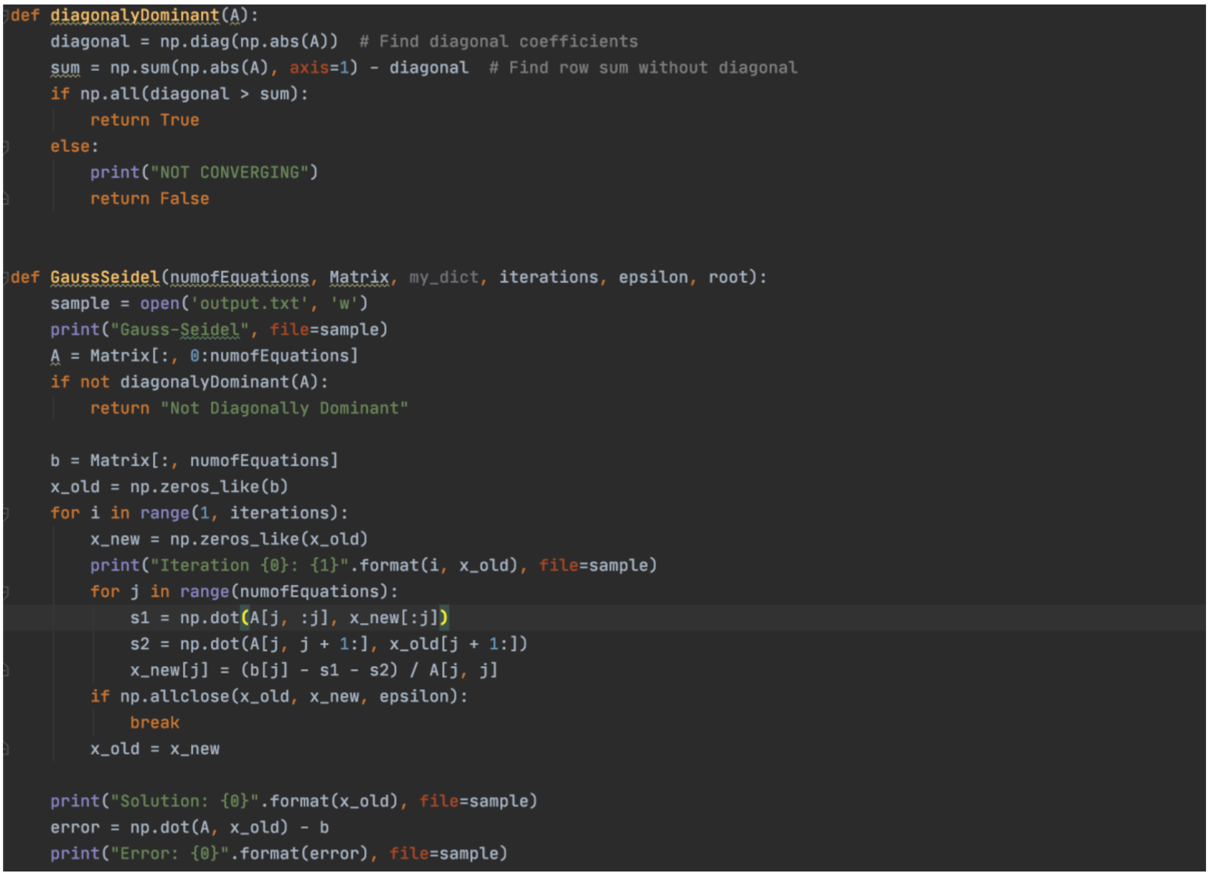
```python
def Gaussian(nofequations, matrix, my_dict, root):
    x = [0] * nofequations
    for k in range(0, nofequations, 1):
        for i in range(k + 1, nofequations, 1):
            ratio = matrix[i][k] / matrix[k][k] * 1.0
            for j in range(0, nofequations + 1, 1):
                matrix[i][j] = matrix[i][j] - ratio * matrix[k][j]
                # print(matrix[i][j])
    # print(matrix)
    x[nofequations - 1] = matrix[nofequations - 1][nofequations] / (matrix[nofequations - 1][nofequations - 1]) * 1.0

    for k in range(nofequations - 2, -1, -1):
        sum = 0

        for j in range(k + 1, nofequations, 1):
            sum += matrix[k][j] * x[j]

        x[k] = 1.0 / matrix[k][k] * (matrix[k][nofequations] - sum)

    sample = open('output.txt', 'w')
    print("Gaussian-Elimination", file=sample)
    fun = tk.Label(root, text="Gauss-Elimination", font=("Times New Roman", 15))
    fun.grid(row=nofequations + 9, columnspan=1, sticky=tk.N + tk.W)
    for i in range(nofequations):
        print("{0} = {1}".format(list(my_dict.keys())[list(my_dict.values()).index(i)], x[i]), file=sample)
        print(list(my_dict.keys())[list(my_dict.values()).index(i)])

        ET = list(my_dict.keys())[list(my_dict.values()).index(i)] + "  =  " + str(x[i])
        ETl = tk.Label(root, text=ET, font=("Arial", 15)).grid(row=nofequations + 10 + i, columnspan=1,
                                                                sticky=tk.N + tk.W, pady=5)
```

Gaussian Jordan:

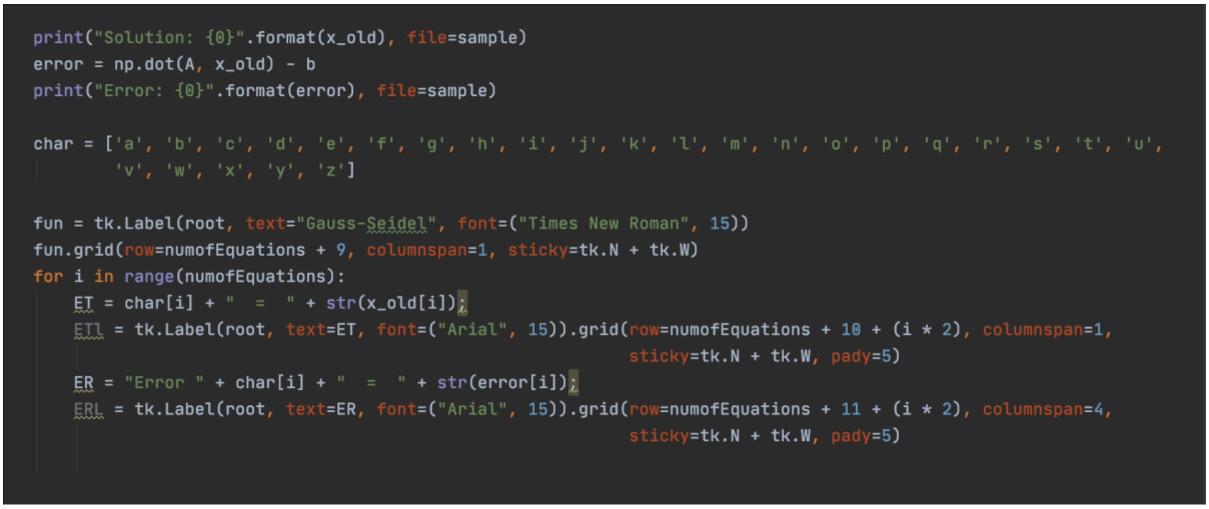


```python
def GaussianJordan(nofequations, matrix, my_dict, root):
    x = [0] * (nofequations)
    for k in range(0, nofequations, 1):
        for i in range(0, nofequations, 1):
            ratio = 1
            if k != i:
                ratio = matrix[i][k] / matrix[k][k] * 1.0
                for j in range(0, nofequations + 1, 1):
                    matrix[i][j] = matrix[i][j] - ratio * matrix[k][j]
                # print(matrix[i][j])
    # print(matrix)
    x[nofequations - 1] = matrix[nofequations - 1][nofequations] / (matrix[nofequations - 1][nofequations - 1]) * 1.0

    for k in range(nofequations - 2, -1, -1):
        sum = 0

        for j in range(k + 1, nofequations, 1):
            sum += matrix[k][j] * x[j]

        x[k] = 1.0 / matrix[k][k] * (matrix[k][nofequations] - sum)
    sample = open('output.txt', 'w')
    print("Gaussian-Jordan", file=sample)
    fun = tk.Label(root, text="Gauss-Jordan", font=("Times New Roman", 15))
    fun.grid(row=nofequations + 9 , columnspan=1 ,sticky=tk.N + tk.W)
    for i in range(nofequations):
        print("{0} = {1}".format(list(my_dict.keys())[list(my_dict.values()).index(i)], x[i]), file=sample)

        ET = list(my_dict.keys())[list(my_dict.values()).index(i)] + "  =  " + str(x[i])
        ET1 = tk.Label(root, text=ET, font=("Arial", 15)).grid(row=nofequations + 10 + i, columnspan=1, sticky=tk.N + tk.W , pady=5)
```

Gaussian Seidel (Also we check whether it converges or not):

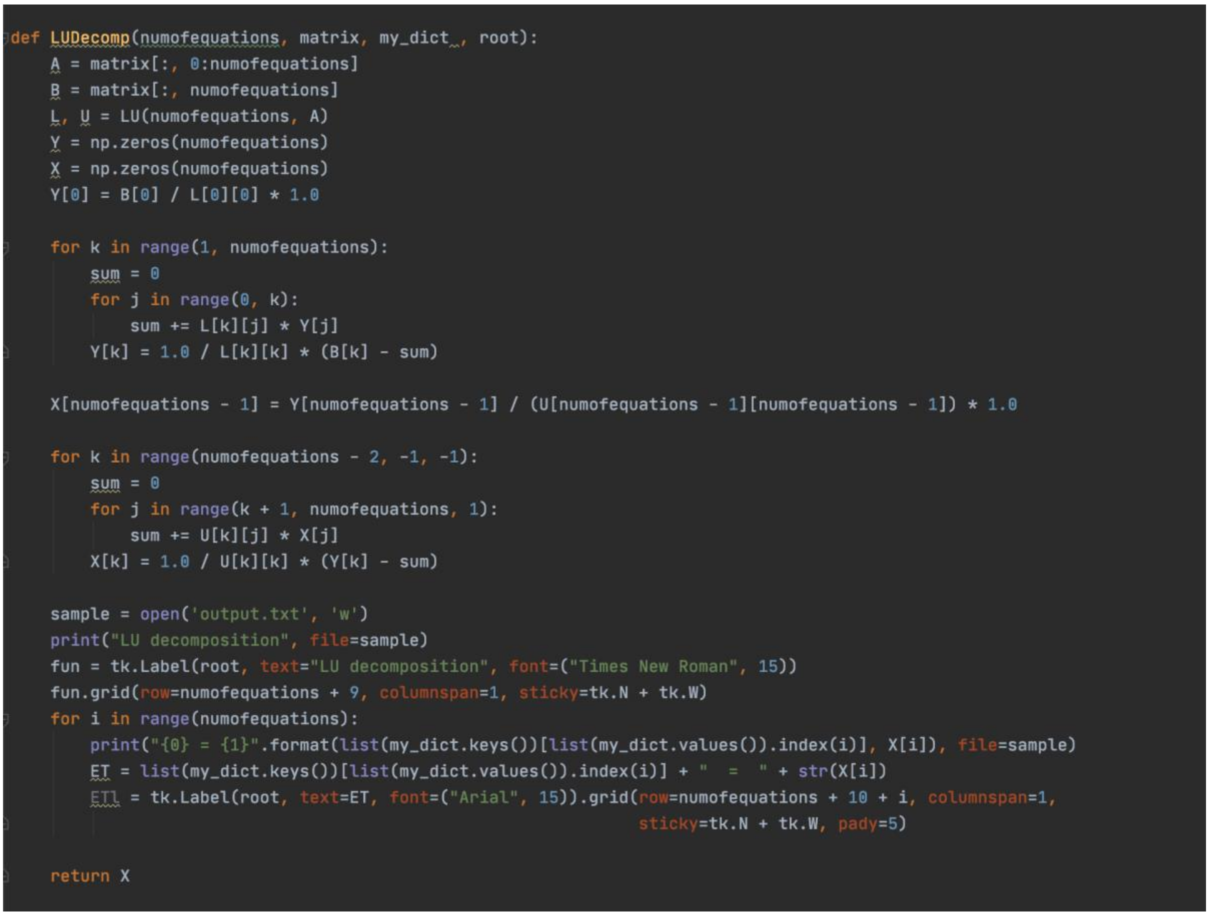


```
def diagonalyDominant(A):
    diagonal = np.diag(np.abs(A))  # Find diagonal coefficients
    sum = np.sum(np.abs(A), axis=1) - diagonal  # Find row sum without diagonal
    if np.all(diagonal > sum):
        return True
    else:
        print("NOT CONVERGING")
        return False


def GaussSeidel(numofEquations, Matrix, my_dict, iterations, epsilon, root):
    sample = open('output.txt', 'w')
    print("Gauss-Seidel", file=sample)
    A = Matrix[:, 0:numofEquations]
    if not diagonalyDominant(A):
        return "Not Diagonally Dominant"

    b = Matrix[:, numofEquations]
    x_old = np.zeros_like(b)
    for i in range(1, iterations):
        x_new = np.zeros_like(x_old)
        print("Iteration {0}: {1}".format(i, x_old), file=sample)
        for j in range(numofEquations):
            s1 = np.dot(A[j, :j], x_new[:j])
            s2 = np.dot(A[j, j + 1:], x_old[j + 1:])
            x_new[j] = (b[j] - s1 - s2) / A[j, j]
        if np.allclose(x_old, x_new, epsilon):
            break
        x_old = x_new

    print("Solution: {0}".format(x_old), file=sample)
    error = np.dot(A, x_old) - b
    print("Error: {0}".format(error), file=sample)
```

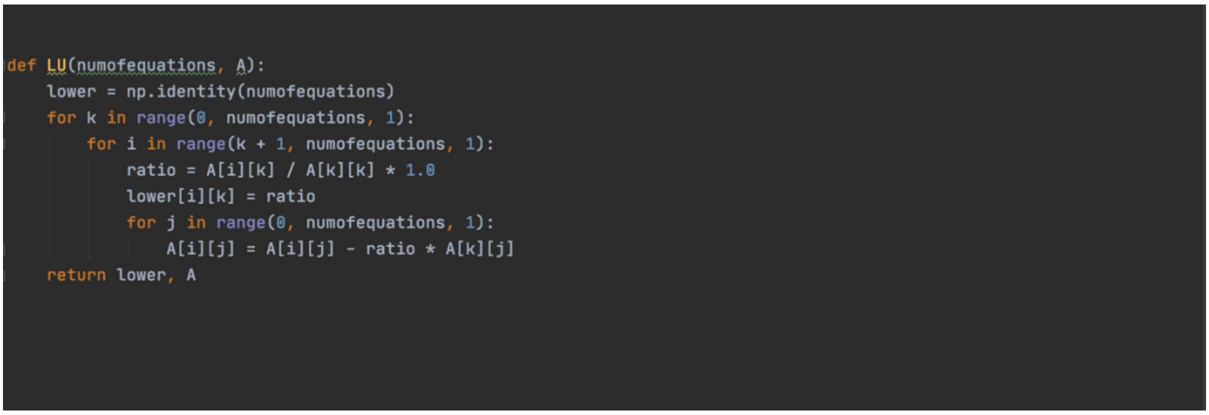


```
print("Solution: {0}".format(x_old), file=sample)
error = np.dot(A, x_old) - b
print("Error: {0}".format(error), file=sample)

char = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
        'v', 'w', 'x', 'y', 'z']

fun = tk.Label(root, text="Gauss-Seidel", font=("Times New Roman", 15))
fun.grid(row=numofEquations + 9, columnspan=1, sticky=tk.N + tk.W)
for i in range(numofEquations):
    ET = char[i] + "  =  " + str(x_old[i]);
    ETL = tk.Label(root, text=ET, font=("Arial", 15)).grid(row=numofEquations + 10 + (i * 2), columnspan=1,
                                                             sticky=tk.N + tk.W, pady=5)
    ER = "Error " + char[i] + "  =  " + str(error[i]);
    ERL = tk.Label(root, text=ER, font=("Arial", 15)).grid(row=numofEquations + 11 + (i * 2), columnspan=4,
                                                             sticky=tk.N + tk.W, pady=5)
```

LU-Decomposition:

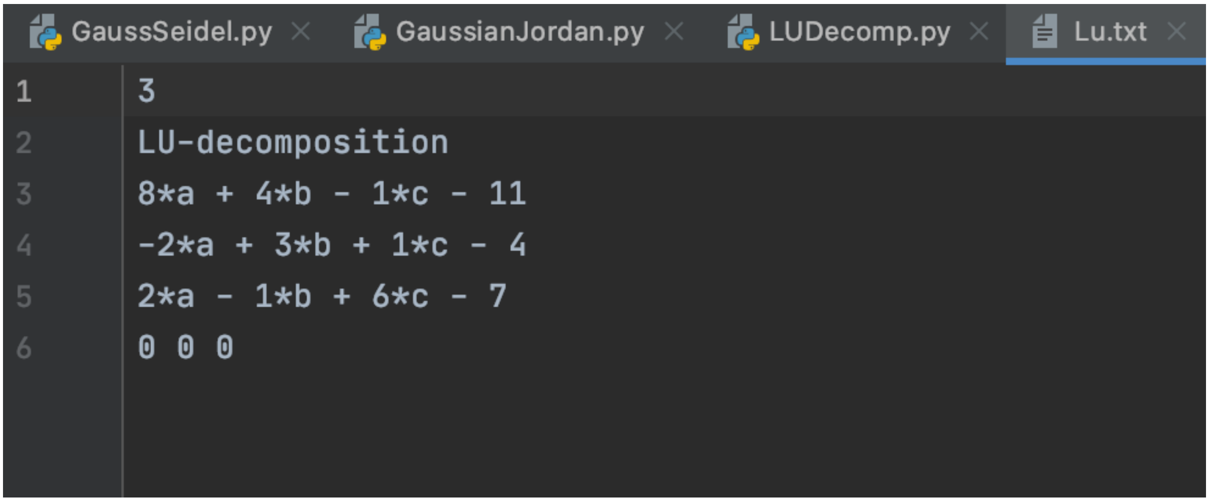


```python
def LUDecomp(numofequations, matrix, my_dict , root):
    A = matrix[:, 0:numofequations]
    B = matrix[:, numofequations]
    L, U = LU(numofequations, A)
    Y = np.zeros(numofequations)
    X = np.zeros(numofequations)
    Y[0] = B[0] / L[0][0] * 1.0

    for k in range(1, numofequations):
        sum = 0
        for j in range(0, k):
            sum += L[k][j] * Y[j]
        Y[k] = 1.0 / L[k][k] * (B[k] - sum)

    X[numofequations - 1] = Y[numofequations - 1] / (U[numofequations - 1][numofequations - 1]) * 1.0

    for k in range(numofequations - 2, -1, -1):
        sum = 0
        for j in range(k + 1, numofequations, 1):
            sum += U[k][j] * X[j]
        X[k] = 1.0 / U[k][k] * (Y[k] - sum)

    sample = open('output.txt', 'w')
    print("LU decomposition", file=sample)
    fun = tk.Label(root, text="LU decomposition", font=("Times New Roman", 15))
    fun.grid(row=numofequations + 9, columnspan=1, sticky=tk.N + tk.W)
    for i in range(numofequations):
        print("{0} = {1}".format(list(my_dict.keys())[list(my_dict.values()).index(i)], X[i]), file=sample)
        ET = list(my_dict.keys())[list(my_dict.values()).index(i)] + "  =  " + str(X[i])
        ETl = tk.Label(root, text=ET, font=("Arial", 15)).grid(row=numofequations + 10 + i, columnspan=1,
                                                               sticky=tk.N + tk.W, pady=5)

    return X
```



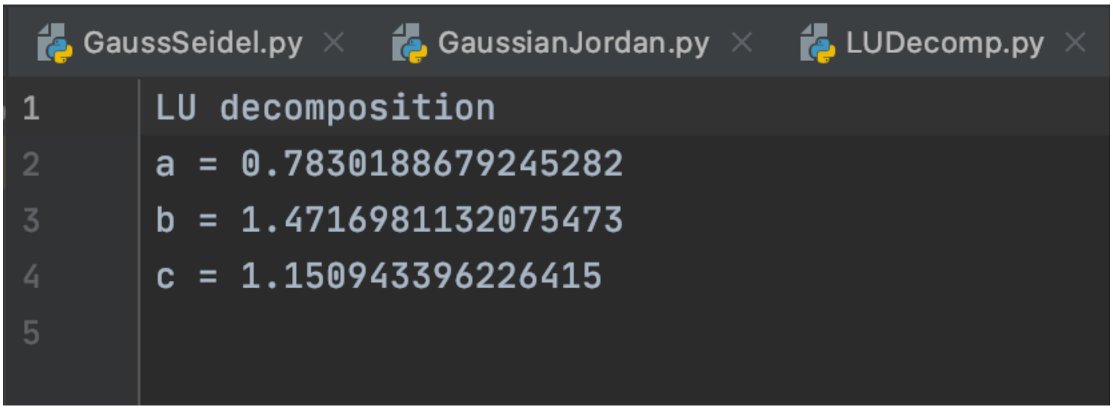
```python
def LU(numofequations, A):
    lower = np.identity(numofequations)
    for k in range(0, numofequations, 1):
        for i in range(k + 1, numofequations, 1):
            ratio = A[i][k] / A[k][k] * 1.0
            lower[i][k] = ratio
            for j in range(0, numofequations, 1):
                A[i][j] = A[i][j] - ratio * A[k][j]
    return lower, A
```

Text File (Reading equation from text file):



```
3
LU-decomposition
8*a + 4*b - 1*c - 11
-2*a + 3*b + 1*c - 4
2*a - 1*b + 6*c - 7
0 0 0
```

Output File :



```
LU decomposition
a = 0.7830188679245282
b = 1.4716981132075473
c = 1.150943396226415
```