Numerical Analysis

Project Name: Numerical Methods Calculator Using python tkinter

Team Members:

1)Esraa Mahmoud Mohamed Mahmoud G2 CS

2)Esraa Khaled Abdelatay Yasin G2 CS

Supervised By:

Prof. Rania Ahmed

T.A.: Dina Ahmed

T.A.: Nagham yahya

T.A.: Ahmed Rashad

T.A.: Salma Elawady

Table of Contents

Modules & Library

1-tkinter library:

- tkinter: This is the standard GUI (Graphical User Interface) toolkit in Python. It provides a fast and easy way to create desktop applications.
- Tkinter is based on the Tk GUI toolkit, which is used for creating GUIs in Tcl (Tool Command Language).
- Tkinter provides various widgets like buttons, labels, text boxes, etc., which you can use to build your application's interface.

2-subprocess:

- This module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.
- It's useful for running system commands from within Python scripts.
- You can use it to execute external commands, such as running other programs or scripts, and interacting with them from your Python code.

3-os:

- This module provides a way of using operating system-dependent functionality.
- It allows you to interact with the operating system in a platform-independent way.
- You can perform tasks such as navigating the file system, creating and deleting directories, and running system commands.

4-messagebox (from tkinter):

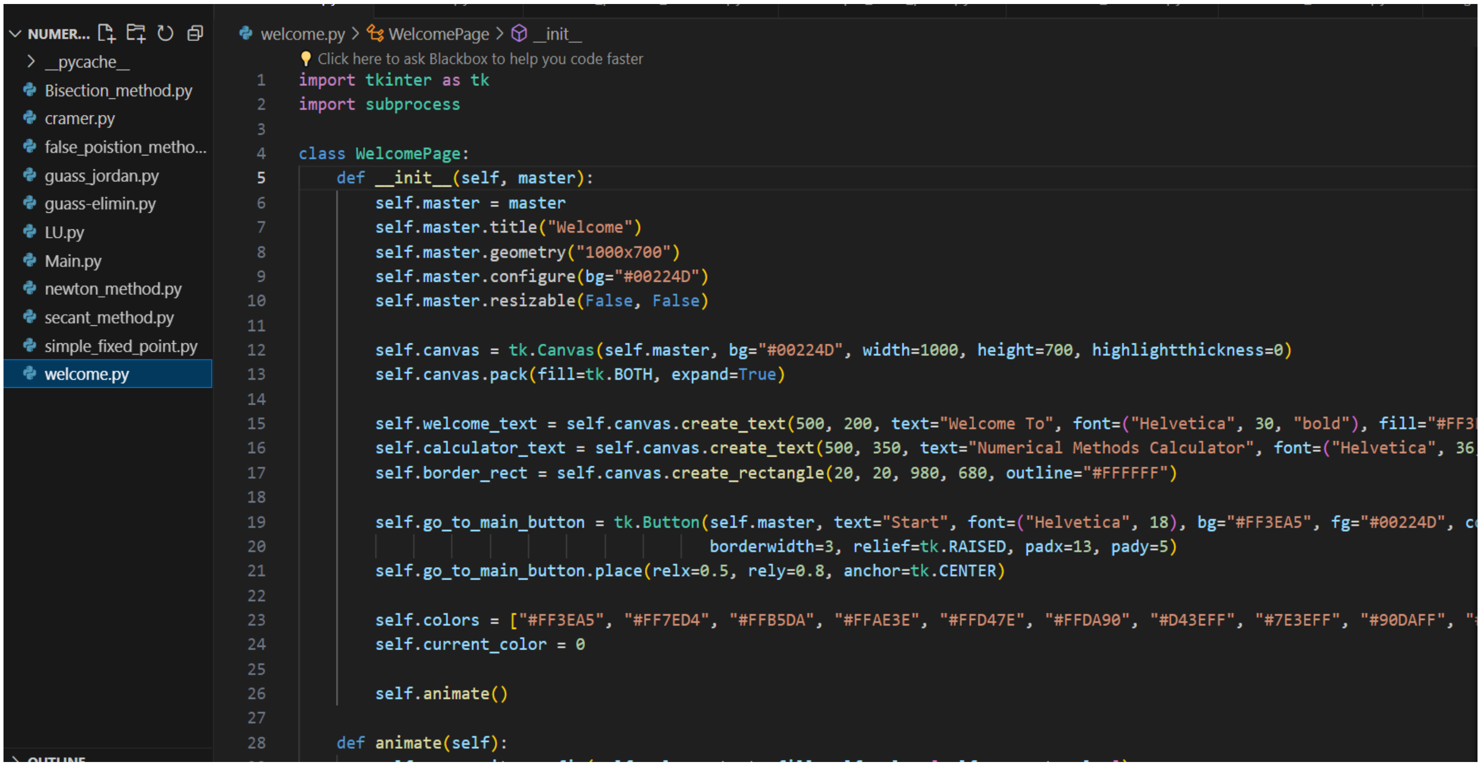
- This is a submodule of the tkinter library that provides a way to create message boxes, including alert boxes, confirmation dialogs, and prompt dialogs.
- Message boxes are useful for displaying information to the user or getting input from the user in a graphical way.

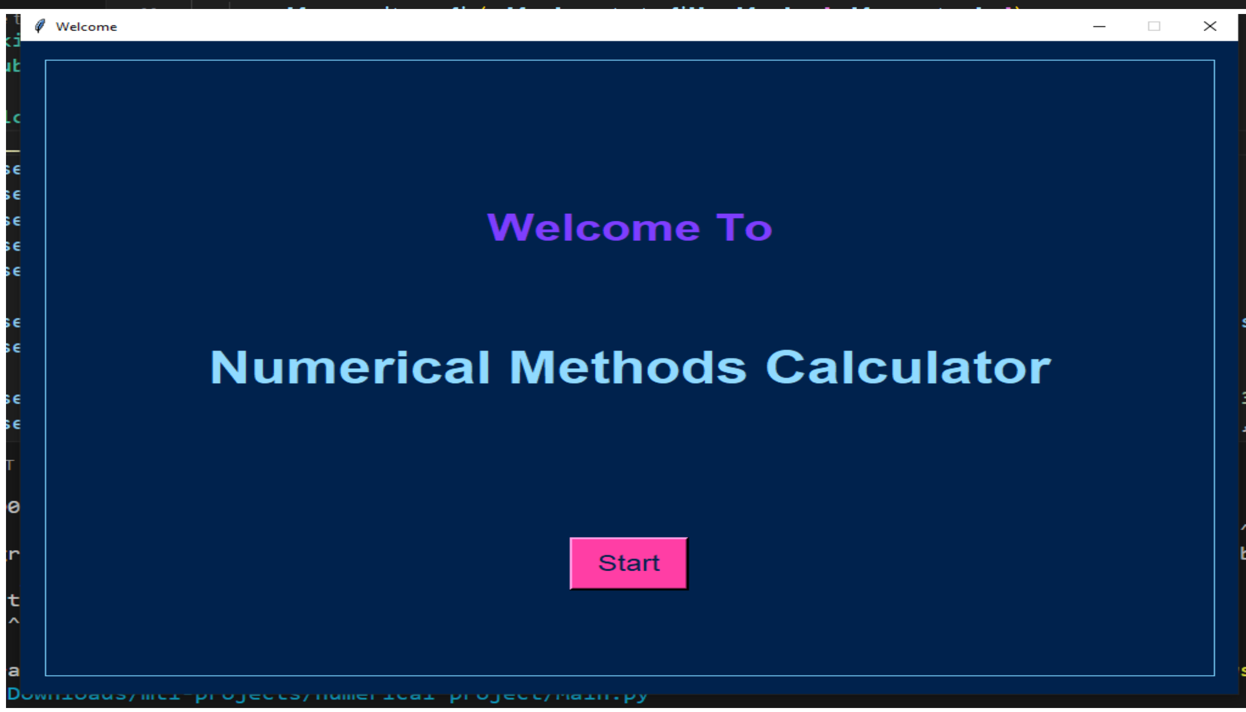
5-ttk (from tkinter):

- This is the themed tkinter module, which provides access to themed widgets with a modern appearance and additional features compared to classic tkinter widgets. It enhances the visual appeal and user experience of your GUI applications.

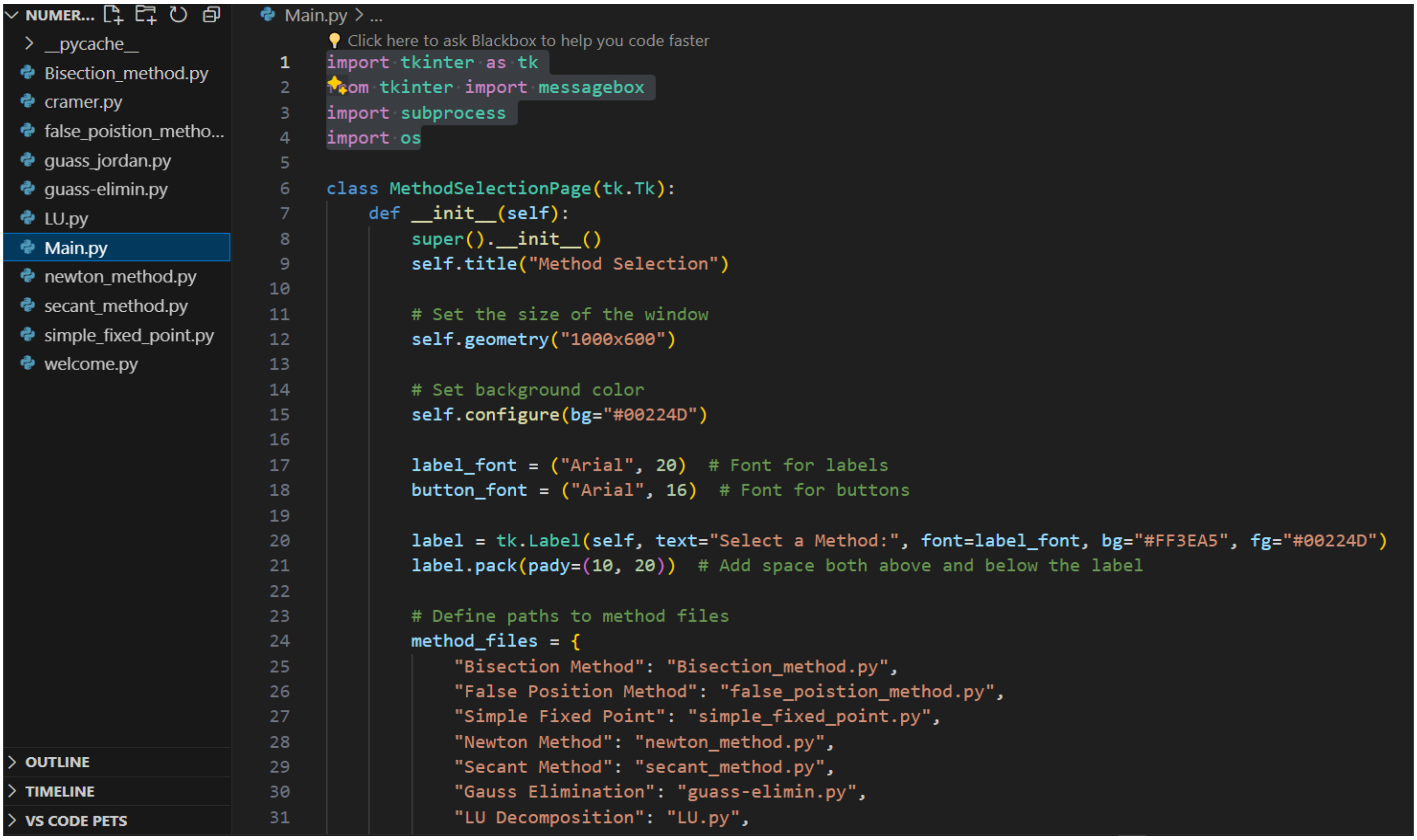
Code implementation & Outputs

Welcome page

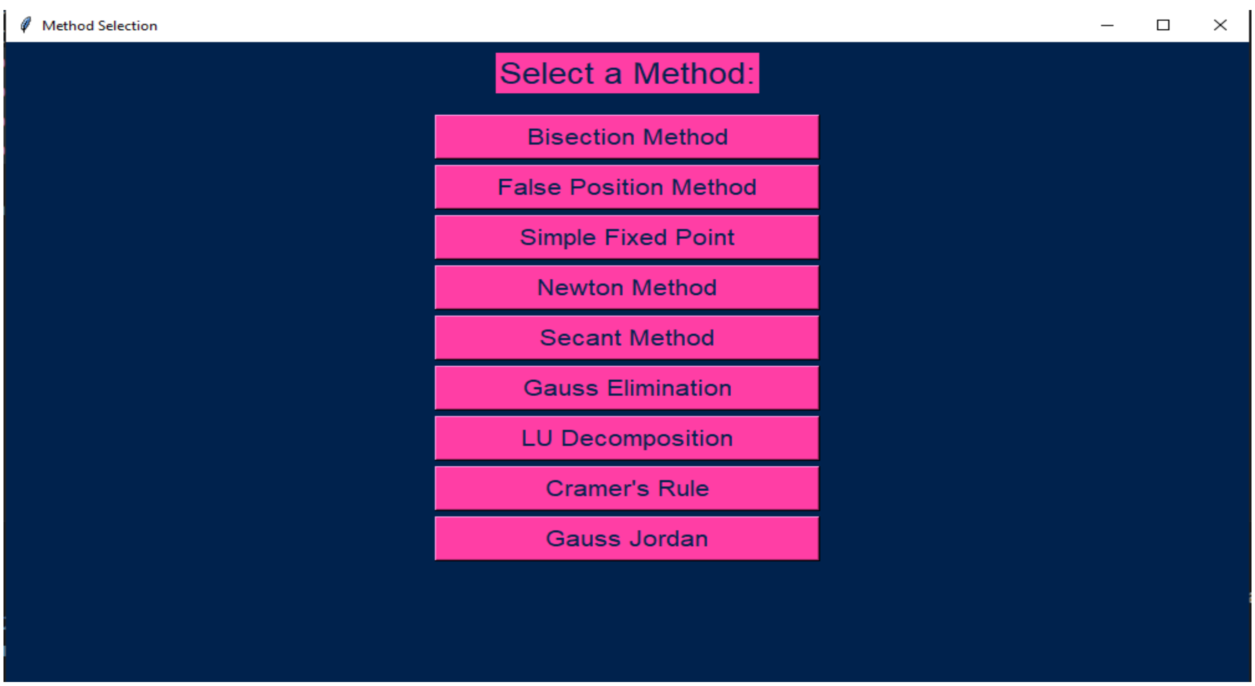




Main page

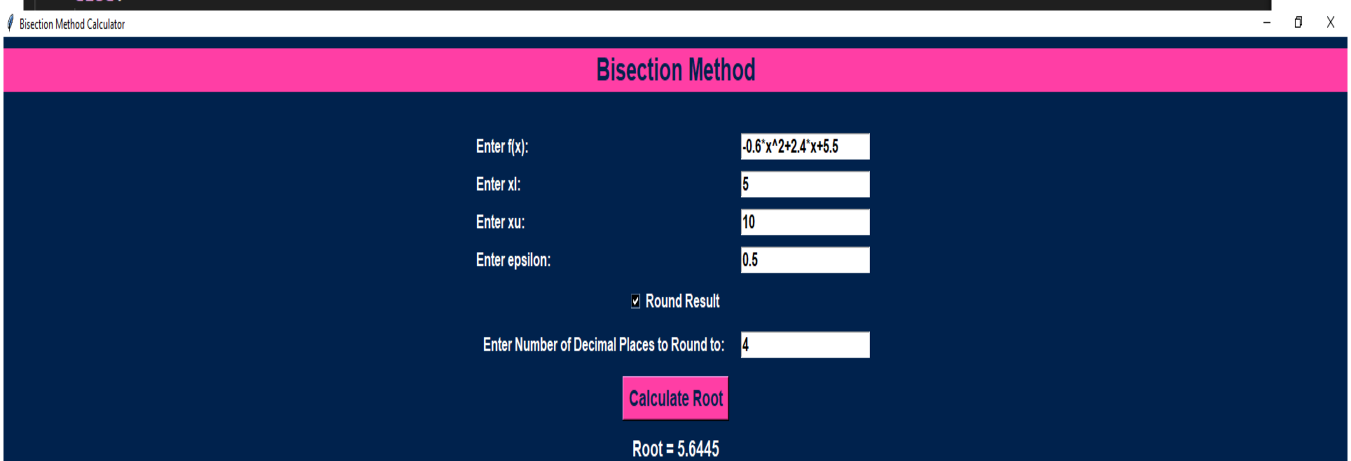



```python
import tkinter as tk
from tkinter import messagebox
import subprocess
import os

class MethodSelectionPage(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Method Selection")

        # Set the size of the window
        self.geometry("1000x600")

        # Set background color
        self.configure(bg="#00224D")

        label_font = ("Arial", 20)  # Font for labels
        button_font = ("Arial", 16)  # Font for buttons

        label = tk.Label(self, text="Select a Method:", font=label_font, bg="#FF3EA5", fg="#00224D")
        label.pack(pady=(10, 20))  # Add space both above and below the label

        # Define paths to method files
        method_files = {
            "Bisection Method": "Bisection_method.py",
            "False Position Method": "false_poistion_method.py",
            "Simple Fixed Point": "simple_fixed_point.py",
            "Newton Method": "newton_method.py",
            "Secant Method": "secant_method.py",
            "Gauss Elimination": "guass-elimin.py",
            "LU Decomposition": "LU.py",
```



Bisection method page

```
import tkinter as tk
from tkinter import ttk
from math import pow
# Function to calculate f(x)
def f(x):
    return eval(entry_f.get().replace("^", "**"))  # Allowing '^' as exponentiation operator

# Bisection method
def bisect(xl, xu, xr, xr_old, error, iter, results):
    xr_old = xr
    xr = (xl + xu) / 2
    error = abs((xr - xr_old) / xr) * 100

    if round_check_var.get():
        if iter == 0:
            results.append((iter, round(xl, round_num), round(f(xl), round_num), round(xu, round_num), round(f(xu), round_n
        else:
            results.append((iter, round(xl, round_num), round(f(xl), round_num), round(xu, round_num), round(f(xu), round_n
    else:
        if iter == 0:
            results.append((iter, xl, f(xl), xu, f(xu), xr, f(xr), ""))
        else:
            results.append((iter, xl, f(xl), xu, f(xu), xr, f(xr), error ))

    m = f(xl) * f(xr)
    if m > 0:
        xl = xr
    elif m == 0:
        return xr
    else:
```



Iteration	xl	f(xl)	xu	f(xu)	xr	f(xr)	Error %
0	5.0	2.5	10.0	-30.5	7.5	-10.25	
1	5.0	2.5	7.5	-10.25	6.25	-2.9375	20.0
2	5.0	2.5	6.25	-2.9375	5.625	0.0156	11.1111
3	5.625	0.0156	6.25	-2.9375	5.9375	-1.4023	5.2632
4	5.625	0.0156	5.9375	-1.4023	5.7812	-0.6787	2.7027
5	5.625	0.0156	5.7812	-0.6787	5.7031	-0.3279	1.3699
6	5.625	0.0156	5.7031	-0.3279	5.6641	-0.1552	0.6897
7	5.625	0.0156	5.6641	-0.1552	5.6445	-0.0696	0.346

False position method page

```
Click here to ask Blackbox to help you code faster
import tkinter as tk
from tkinter import ttk
from math import pow

# Function to calculate f(x)
def f(x):
    return eval(entry_f.get().replace("^", "**"))  # Allowing '^' as exponentiation operator

# False Position method
def false_position(xl, xu, xr, xr_old, error, iter, results):
    xr_old = xr
    xr = xu - (f(xu) * (xl - xu)) / (f(xl) - f(xu))
    error = abs((xr - xr_old) / xr) * 100

    if round_check_var.get():
        if iter == 0:
            results.append((iter, round(xl, round_num), round(f(xl), round_num), round(xu, round_num), round(f(xu), round_n
        else:
            results.append((iter, round(xl, round_num), round(f(xl), round_num), round(xu, round_num), round(f(xu), round_n
    else:
        if iter == 0:
            results.append((iter, xl, f(xl), xu, f(xu), xr, f(xr), ""))
        else:
            results.append((iter, xl, f(xl), xu, f(xu), xr, f(xr), error))

    m = f(xl) * f(xr)
    if m > 0:
        xl = xr
    elif m == 0:
```

False Position Method Calculator

False Position Method

Enter f(x): x^3+1.2*x^2-4*x-4.8

Enter xl: -1.5

Enter xu: -1

Enter epsilon: 0.1

☑ Round Result

Enter Number of Decimal Places to Round to: 4

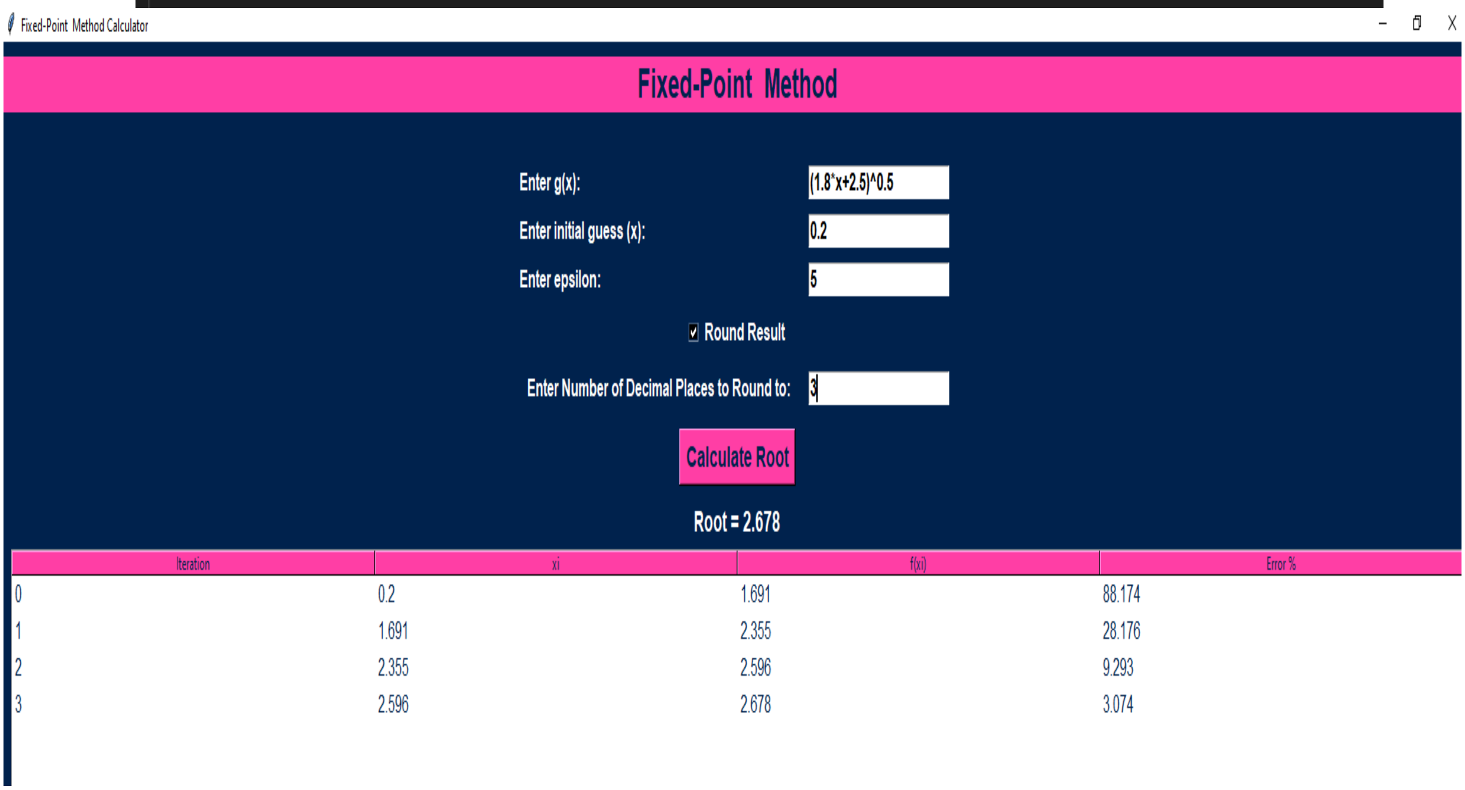
Calculate Root

Root = -1.2000

Iteration	xl	f(xl)	xu	f(xu)	xr	f(xr)	Error %
0	-1.5	0.525	-1.0	-0.6	-1.2667	0.1597	
1	-1.2667	0.1597	-1.0	-0.6	-1.2106	0.0269	4.6306
2	-1.2106	0.0269	-1.0	-0.6	-1.2016	0.004	0.7517
3	-1.2016	0.004	-1.0	-0.6	-1.2002	0.0006	0.112
4	-1.2002	0.0006	-1.0	-0.6	-1.2	0.0001	0.0165

Simple fixed point method page

```python
import tkinter as tk
from tkinter import ttk

# Function to calculate g(x) for the Fixed-Point Iteration method
def g(x):
    return eval(entry_g.get().replace("^", "**"))  # Allowing '^' as exponentiation operator

# Fixed-Point Iteration method
def fixed_point_iteration(x, error, iter, results):
    xi = x
    xi_plus_1 = 0
    while True:
        xi_plus_1 = g(xi)
        error = abs((xi_plus_1 - xi) / xi_plus_1) * 100

        if round_check_var.get():
            results.append((iter, round(xi, round_num), round(xi_plus_1, round_num), round(error, round_num)))
        else:
            results.append((iter, xi, xi_plus_1, error))

        xi = xi_plus_1
        iter += 1

        if error <= eps:
            break

    root_result.set(f"Root = {xi_plus_1:.{round_num}f}" if round_check_var.get() else f"Root = {xi_plus_1}")  # Set root r
    final_result_label.config(text=root_result.get())  # Update final result label
```

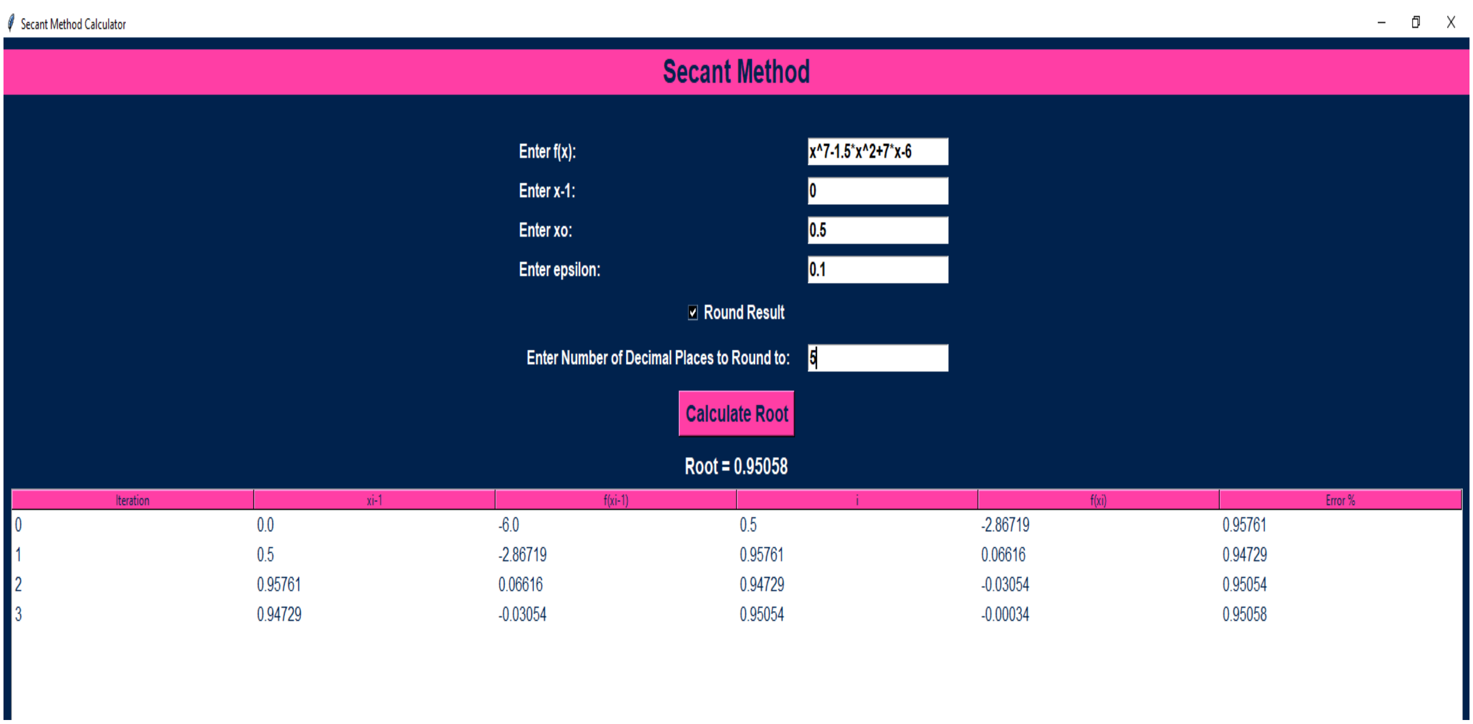


Newton method page

```
import tkinter as tk
from tkinter import ttk
from math import pow

# Function to calculate f(x)
def f(x):
    return eval(entry_f.get().replace("^", "**"))  # Allowing '^' as exponentiation operator

# Function to calculate f'(x)
def f_dash(x):
    h = 1e-10  # Small value for calculating derivative numerically
    return (f(x + h) - f(x)) / h

# Newton-Raphson method
def newton(x, error, iter, results):
    xi = x
    xi_plus_1 = 0
    while True:
        xi_plus_1 = xi - (f(xi) / f_dash(xi))
        error = abs((xi_plus_1 - xi) / xi_plus_1) * 100

        if round_check_var.get():
            results.append((iter, round(xi, round_num), round(f(xi), round_num), round(f_dash(xi), round_num), round(error,
        else:
            results.append((iter, xi, f(xi), f_dash(xi), error))

        xi = xi_plus_1
        iter += 1
```

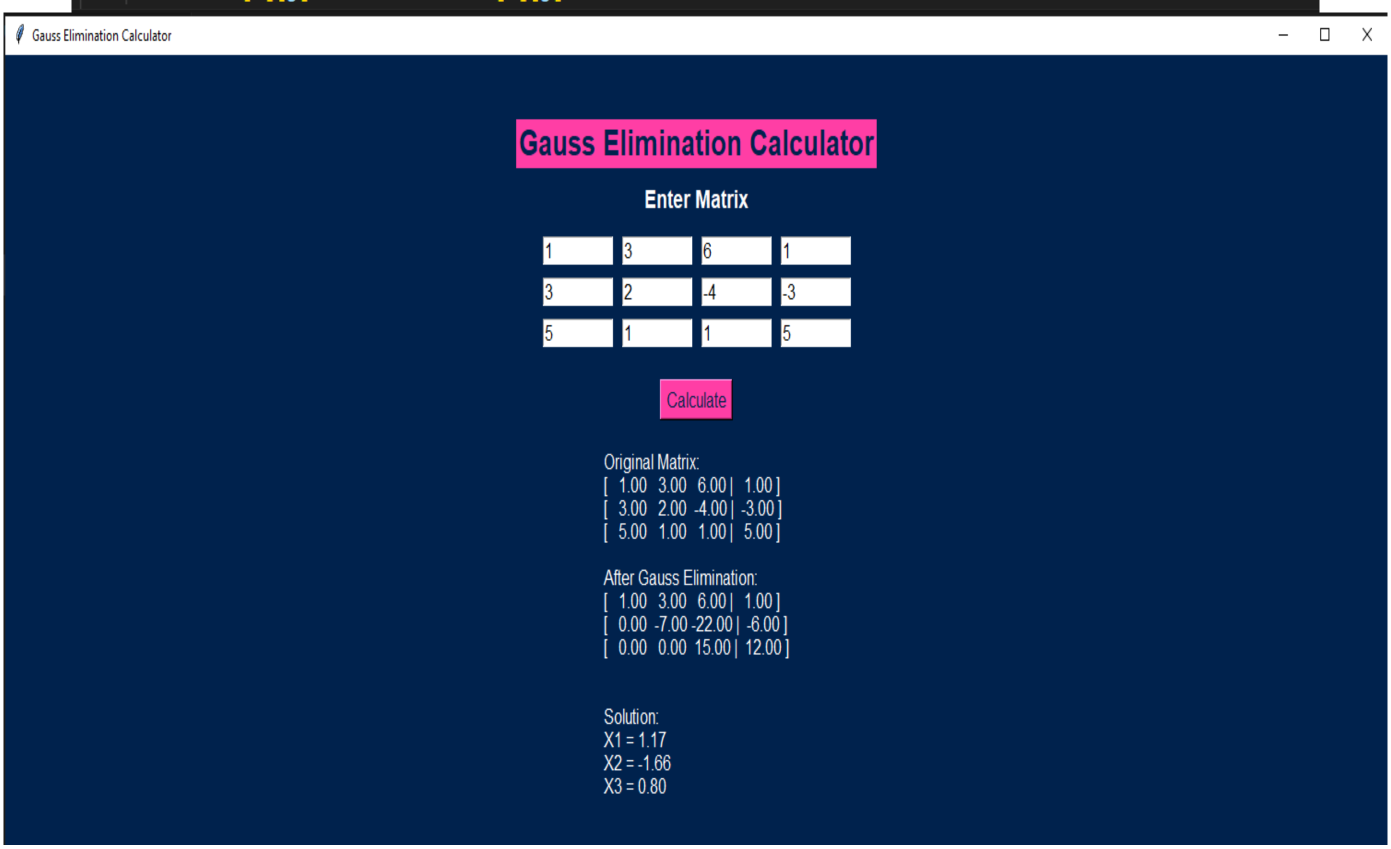
Secant method page

```
import tkinter as tk
from tkinter import ttk

# Function to calculate f(x)
def f(x):
    return eval(entry_f.get().replace("^", "**"))  # Allowing '^' as exponentiation operator

# Secant method
def secant(x0, x1, eps, iter, results):
    while True:
        x2 = x1 - (f(x1) * (x1 - x0)) / (f(x1) - f(x0))
        error = abs((x2 - x1) / x2) * 100

        if round_check_var.get():
            results.append((iter, round(x0, round_num), round(f(x0), round_num), round(x1, round_num), round(f(x1), round_n
        else:
            results.append((iter, x0, f(x0), x1, f(x1), x2, f(x2), error))

        if error < eps:
            root_result.set(f"Root = {x2:.{round_num}f}" if round_check_var.get() else f"Root = {x2}")
            final_result_label.config(text=root_result.get())

            for item in result_tree.get_children():
                result_tree.delete(item)

            for result in results:
                result_tree.insert('', 'end', values=result)

            return
```



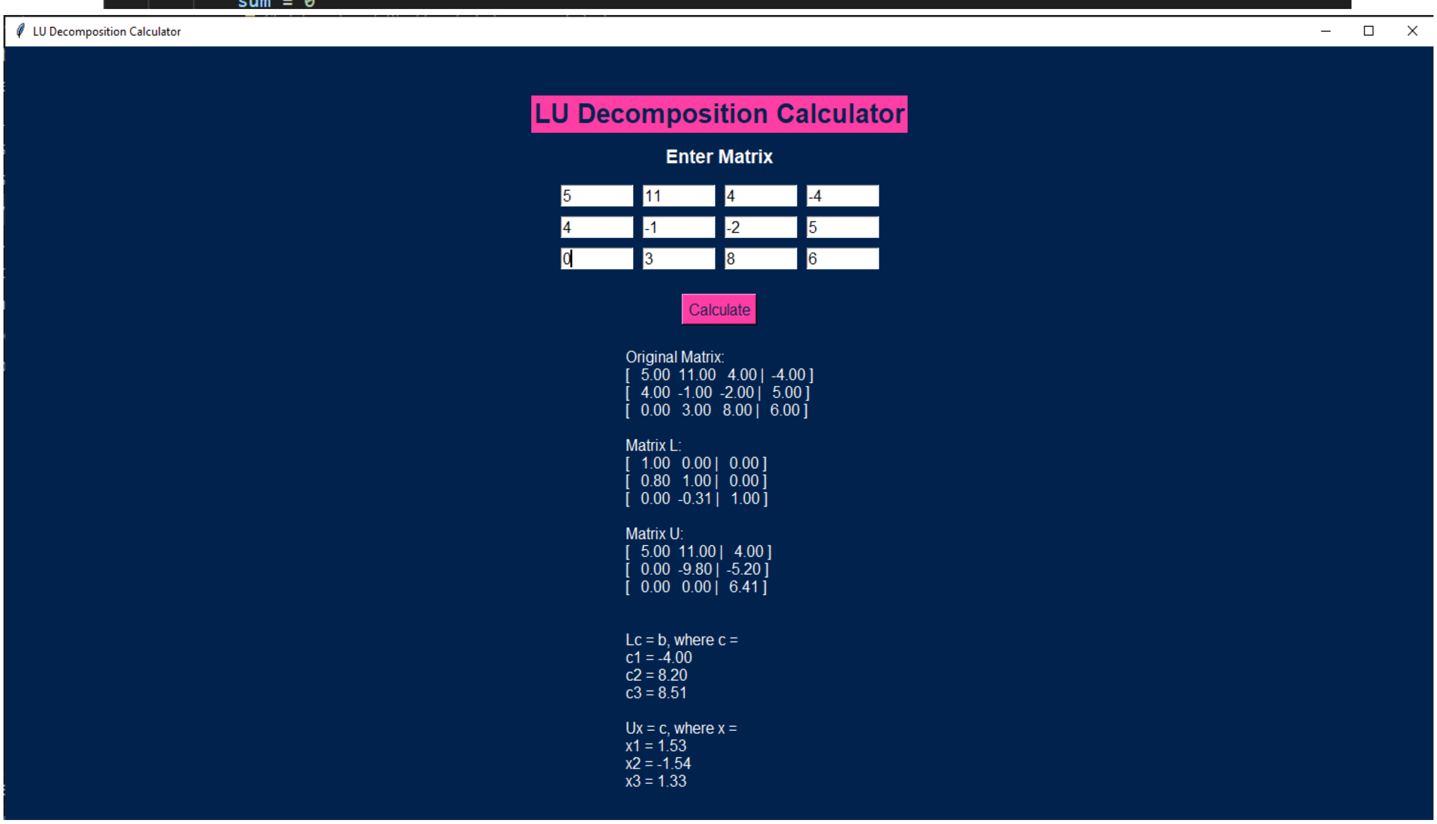
Guass eliminnation page

```python
import tkinter as tk
from tkinter import messagebox

def display_matrix(matrix):
    matrix_str = ""
    for i in range(3):
        matrix_str += "[ "
        for j in range(4):
            if j == 3:
                matrix_str += "| "
            matrix_str += "{:>6.2f} ".format(matrix[i][j])  # Adjusted format for larger matrix
        matrix_str += "]\n"
    return matrix_str

def gje(matrix):
    m21 = matrix[1][0] / matrix[0][0]
    m31 = matrix[2][0] / matrix[0][0]

    # Rule E2-(m21)E1 = E2
    for j in range(4):
        matrix[1][j] -= m21 * matrix[0][j]

    # Rule E3-(m31)E1 = E3
    for j in range(4):
        matrix[2][j] -= m31 * matrix[0][j]

    m32 = matrix[2][1] / matrix[1][1]

    # Rule E3-(m32)E2 = E3
    for j in range(4):
        matrix[2][j] -= m32 * matrix[1][j]
```



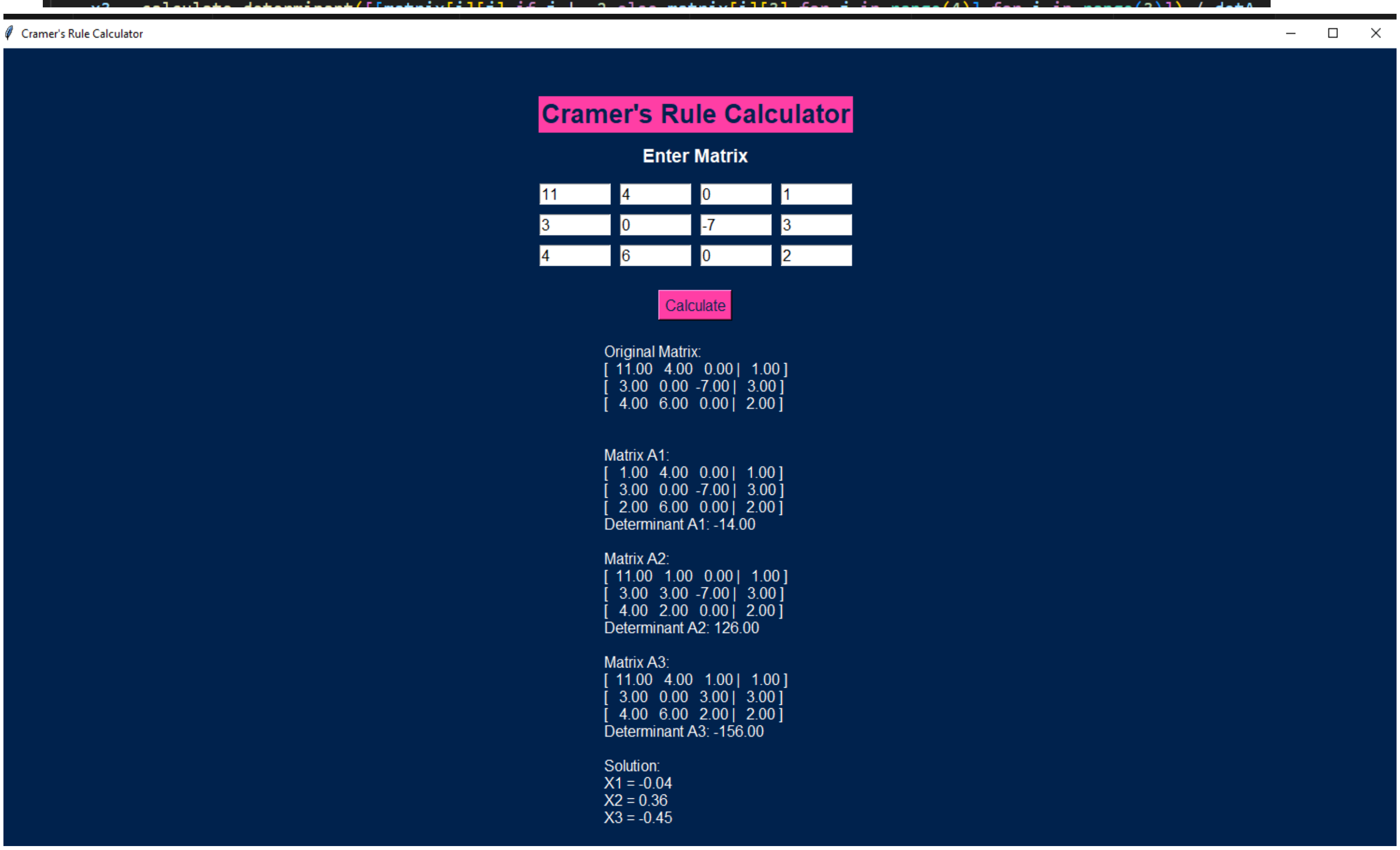
LU page

```python
import tkinter as tk
from tkinter import messagebox

def display_matrix(matrix):
    matrix_str = ""
    for i in range(len(matrix)):
        matrix_str += "[ "
        for j in range(len(matrix[i])):
            if j == len(matrix[i]) - 1:
                matrix_str += "| "
            matrix_str += "{:>6.2f} ".format(matrix[i][j])
        matrix_str += "]\n"
    return matrix_str

def lu_decomposition(matrix):
    n = len(matrix)
    lower = [[0.0] * n for _ in range(n)]
    upper = [[0.0] * n for _ in range(n)]

    for i in range(n):
        lower[i][i] = 1.0

    for i in range(n):
        for j in range(i, n):
            sum = 0
            for k in range(i):
                sum += (lower[i][k] * upper[k][j])
            upper[i][j] = matrix[i][j] - sum

        for j in range(i, n):
            sum = 0
```



Cramer page

```python
import tkinter as tk
from tkinter import messagebox

def display_matrix(matrix):
    matrix_str = ""
    for i in range(3):
        matrix_str += "[ "
        for j in range(4):
            if j == 3:
                matrix_str += "| "
            matrix_str += "{:>6.2f} ".format(matrix[i][j])  # Adjusted format for larger matrix
        matrix_str += "]\n"
    return matrix_str

def calculate_determinant(matrix):
    return (matrix[0][0] * matrix[1][1] * matrix[2][2] +
            matrix[0][1] * matrix[1][2] * matrix[2][0] +
            matrix[0][2] * matrix[1][0] * matrix[2][1] -
            matrix[0][2] * matrix[1][1] * matrix[2][0] -
            matrix[0][1] * matrix[1][0] * matrix[2][2] -
            matrix[0][0] * matrix[1][2] * matrix[2][1])

def cramer(matrix):
    detA = calculate_determinant([row[:3] for row in matrix])

    if detA == 0:
        return None  # No unique solution

    x1 = calculate_determinant([[matrix[i][3] if j == 0 else matrix[i][j] for j in range(4)] for i in range(3)]) / detA
    x2 = calculate_determinant([[matrix[i][j] if j != 1 else matrix[i][3] for j in range(4)] for i in range(3)]) / detA
    x3 = calculate_determinant([[matrix[i][j] if j != 2 else matrix[i][3] for j in range(4)] for i in range(3)]) / detA
```



Guass Jordan page

Click here to ask Blackbox to help you code faster

```
import tkinter as tk
from tkinter import messagebox

def print_matrix(matrix, message="Matrix"):
    print(message)
    for row in matrix:
        print(row)

def gauss_jordan(matrix, result_label):
    n = len(matrix)

    # Forward Elimination
    for i in range(n):
        # Partial pivoting
        max_row = i
        for j in range(i + 1, n):
            if abs(matrix[j][i]) > abs(matrix[max_row][i]):
                max_row = j
        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        # Make the diagonal elements 1
        pivot = matrix[i][i]
        for j in range(i, n + 1):
            matrix[i][j] /= pivot

        # Make the other elements in the column 0
        for j in range(n):
            if j != i:
                factor = matrix[j][i]
```