

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	3
1-tkinter library:.....	3
2-subprocess:.....	3
3-os:.....	3
4-messagebox (from tkinter):.....	3
5-ttk (from tkinter):	3
Code implementation & Outputs	4
Welcome page	4
Main page	5
Bisection method page	6
False position method page	7
Simple fixed point method page	8
Newton method page.....	9
Secant method page	10
Guass eliminination page	11
LU page	12
Cramer page	13
Guass Jordan page.....	14

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):

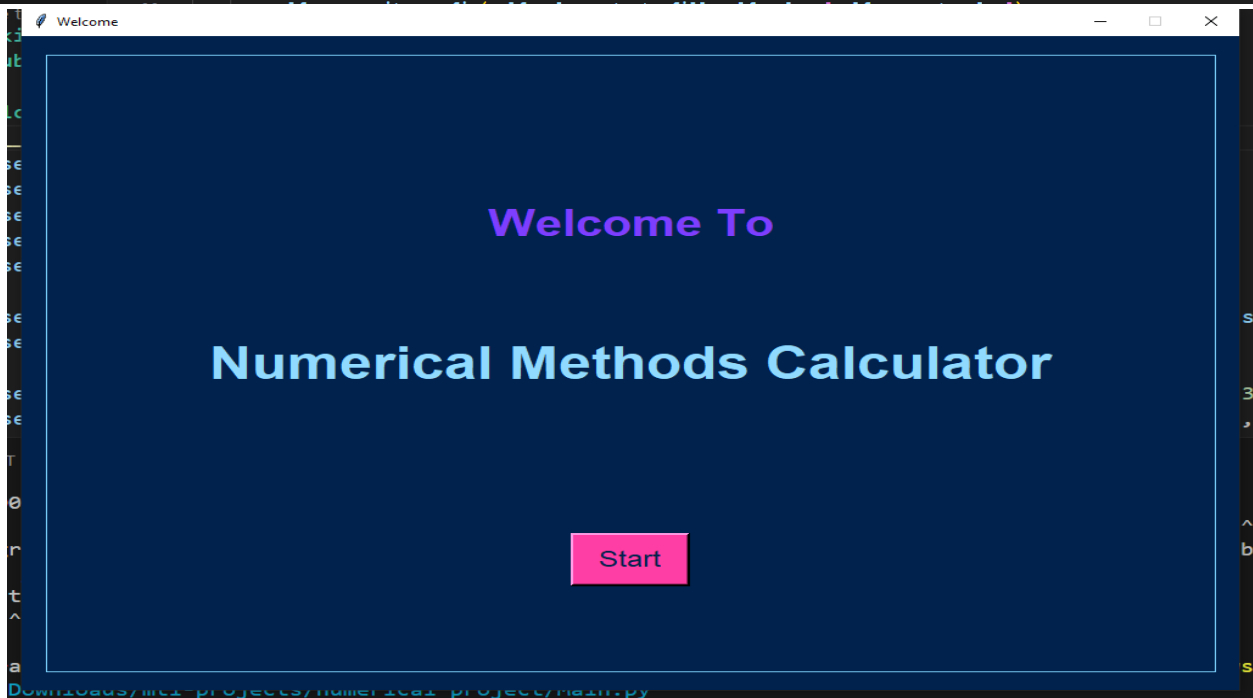
- This is the themed tkinter module, which provides access to themed widgets with a modern appearance and additional features compared to classic tkinter widgets. It enhances the visual appeal and user experience of your GUI applications.

Code implementation & Outputs

Welcome page

```
NUMER...  welcome.py  WelcomePage  _init_
> __pycache__
  Bisection_method.py
  cramer.py
  false_poistion_metho...
  guass_jordan.py
  guass-elimin.py
  LU.py
  Main.py
  newton_method.py
  secant_method.py
  simple_fixed_point.py
  welcome.py

1  import tkinter as tk
2  import subprocess
3
4  class WelcomePage:
5      def __init__(self, master):
6          self.master = master
7          self.master.title("Welcome")
8          self.master.geometry("1000x700")
9          self.master.configure(bg="#00224D")
10         self.master.resizable(False, False)
11
12         self.canvas = tk.Canvas(self.master, bg="#00224D", width=1000, height=700, highlightthickness=0)
13         self.canvas.pack(fill=tk.BOTH, expand=True)
14
15         self.welcome_text = self.canvas.create_text(500, 200, text="Welcome To", font=("Helvetica", 30, "bold"), fill="#FF3EAE")
16         self.calculator_text = self.canvas.create_text(500, 350, text="Numerical Methods Calculator", font=("Helvetica", 36, "bold"), fill="#00224D")
17         self.border_rect = self.canvas.create_rectangle(20, 20, 980, 680, outline="FFFFFF")
18
19         self.go_to_main_button = tk.Button(self.master, text="Start", font=("Helvetica", 18), bg="#FF3EAE", fg="#00224D",
20                                           borderwidth=3, relief=tk.RAISED, padx=13, pady=5)
21         self.go_to_main_button.place(relx=0.5, rely=0.8, anchor=tk.CENTER)
22
23         self.colors = ["#FF3EAE", "#FF7ED4", "#FFB5DA", "#FFAE3E", "#FFD47E", "#FFDA90", "#D43EFF", "#7E3EFF", "#90DAFF", "#00224D"]
24         self.current_color = 0
25
26         self.animate()
27
28     def animate(self):
```

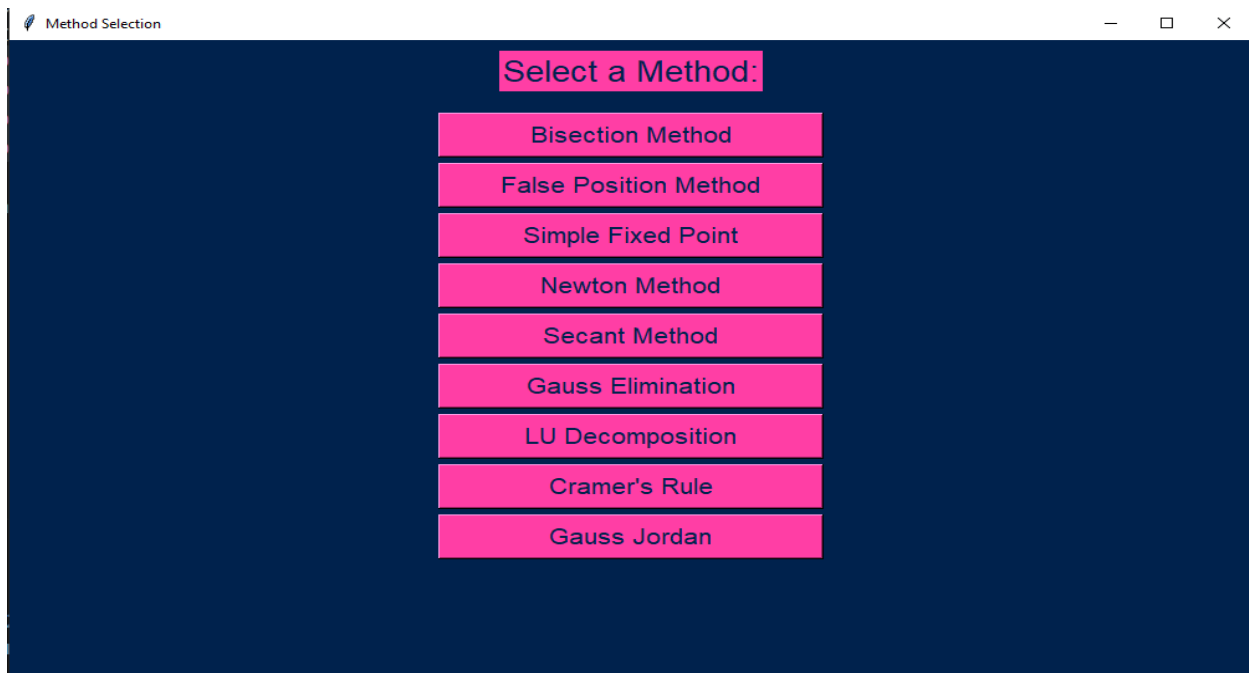


Main page

```
NUMER... [?] [?] [?] [?] [?] [?]
> __pycache__
  Bisecton_method.py
  cramer.py
  false_poistion_metho...
  guass_jordan.py
  guass-elimin.py
  LU.py
  Main.py
  newton_method.py
  secant_method.py
  simple_fixed_point.py
  welcome.py

> OUTLINE
> TIMELINE
> VS CODE PETS

Main.py > ...
  Click here to ask Blackbox to help you code faster
1  import tkinter as tk
2  from tkinter import messagebox
3  import subprocess
4  import os
5
6  class MethodSelectionPage(tk.Tk):
7      def __init__(self):
8          super().__init__()
9          self.title("Method Selection")
10
11         # Set the size of the window
12         self.geometry("1000x600")
13
14         # Set background color
15         self.configure(bg="#00224D")
16
17         label_font = ("Arial", 20) # Font for labels
18         button_font = ("Arial", 16) # Font for buttons
19
20         label = tk.Label(self, text="Select a Method:", font=label_font, bg="#FF3EA5", fg="#00224D")
21         label.pack(pady=(10, 20)) # Add space both above and below the label
22
23         # Define paths to method files
24         method_files = {
25             "Bisection Method": "Bisection_method.py",
26             "False Position Method": "false_poistion_method.py",
27             "Simple Fixed Point": "simple_fixed_point.py",
28             "Newton Method": "newton_method.py",
29             "Secant Method": "secant_method.py",
30             "Gauss Elimination": "guass-elimin.py",
31             "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_num), round(xr, round_num), round(f(xr), round_num), round(error, round_num))))
        else:
            results.append((iter, round(xl, round_num), round(f(xl), round_num), round(xu, round_num), round(f(xu), round_num), round(xr, round_num), round(f(xr), round_num), round(error, round_num))))
    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:
        xu = xr
```

Bisection Method Calculator

Bisection Method

Enter f(x):

Enter xl:

Enter xu:

Enter epsilon:

☒ Round Result

Enter Number of Decimal Places to Round to:

Calculate Root

Root = 5.6445

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_num), round(xr, round_num), round(f(xr), round_num), round(error, round_num)))
        else:
            results.append((iter, round(xl, round_num), round(f(xl), round_num), round(xu, round_num), round(f(xu), round_num), round(xr, round_num), round(f(xr), round_num), round(error, round_num)))
    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:
        xr = xr

```

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

```
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 result
    final_result_label.config(text=root_result.get()) # Update final result label
```

Fixed-Point Method Calculator

- □ X

Fixed-Point Method

Enter g(x): (1.8*x+2.5)^0.5

Enter initial guess (x): 0.2

Enter epsilon: 5

☒ Round Result

Enter Number of Decimal Places to Round to: 3

Calculate Root

Root = 2.678

Iteration	xi	f(xi)	Error %
0	0.2	1.691	88.174
1	1.691	2.355	28.176
2	2.355	2.596	9.293
3	2.596	2.678	3.074

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
```

Newton Method Calculator

- □ X

Newton Method

Enter f(x):

Enter initial guess (x):

Enter epsilon:

☒ Round Result

Enter Number of Decimal Places to Round to:

Calculate Root

Root = 3.5899

Iteration	xi	f(xi)	f'(xi)	Error%
0	5.0	41.0	50.7004	19.294
1	4.1913	10.9099	25.0266	11.6082
2	3.7554	2.3901	14.4415	4.6102

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_num), round(x2, round_num), round(f(x2), round_num), error))
        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
```

Secant Method Calculator

— □ ×

Secant Method

Enter f(x):

Enter x-1:

Enter x0:

Enter epsilon:

☒ Round Result

Enter Number of Decimal Places to Round to:

Calculate Root

Root = 0.95058

Iteration	x_{i-1}	$f(x_{i-1})$	i	$f(x_i)$	Error %
0	0.0	-6.0	0.5	-2.86719	0.95761
1	0.5	-2.86719	0.95761	0.06616	0.94729
2	0.95761	0.06616	0.94729	-0.03054	0.95054
3	0.94729	-0.03054	0.95054	-0.00034	0.95058

Guass eliminination page

```
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]
```

Gauss Elimination Calculator

— □ ×

Gauss Elimination Calculator

Enter Matrix

1	3	6	1
3	2	-4	-3
5	1	1	5

Calculate

Original Matrix:
[1.00 3.00 6.00 | 1.00]
[3.00 2.00 -4.00 | -3.00]
[5.00 1.00 1.00 | 5.00]

After Gauss Elimination:
[1.00 3.00 6.00 | 1.00]
[0.00 -7.00 -22.00 | -6.00]
[0.00 0.00 15.00 | 12.00]

Solution:
X1 = 1.17
X2 = -1.66
X3 = 0.80

LU page

```
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
```

LU Decomposition Calculator

Enter Matrix

5	11	4	-4
4	-1	-2	5
0	3	8	6

Calculate

Original Matrix:

[5.00	11.00	4.00		-4.00]
[4.00	-1.00	-2.00		5.00]
[0.00	3.00	8.00		6.00]

Matrix L:

[1.00	0.00		0.00]
[0.80	1.00		0.00]
[0.00	-0.31		1.00]

Matrix U:

[5.00	11.00		4.00]
[0.00	-9.80		-5.20]
[0.00	0.00		6.41]

$Lc = b$, where $c =$
 $c1 = -4.00$
 $c2 = 8.20$
 $c3 = 8.51$

$Ux = c$, where $x =$
 $x1 = 1.53$
 $x2 = -1.54$
 $x3 = 1.33$

Cramer page

```
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
```

Cramer's Rule Calculator

Cramer's Rule Calculator

Enter Matrix

11	4	0	1
3	0	-7	3
4	6	0	2

Calculate

Original Matrix:

```
[ 11.00  4.00  0.00 |  1.00]
[  3.00  0.00 -7.00 |  3.00]
[  4.00  6.00  0.00 |  2.00]
```

Matrix A1:

```
[  1.00  4.00  0.00 |  1.00]
[  3.00  0.00 -7.00 |  3.00]
[  2.00  6.00  0.00 |  2.00]
```

Determinant A1: -14.00

Matrix A2:

```
[ 11.00  1.00  0.00 |  1.00]
[  3.00  3.00 -7.00 |  3.00]
[  4.00  2.00  0.00 |  2.00]
```

Determinant A2: 126.00

Matrix A3:

```
[ 11.00  4.00  1.00 |  1.00]
[  3.00  0.00  3.00 |  3.00]
[  4.00  6.00  2.00 |  2.00]
```

Determinant A3: -156.00

Solution:

X1 = -0.04

X2 = 0.36

X3 = -0.45

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]
                for k in range(i, n + 1):
                    matrix[j][k] -= factor * matrix[i][k]
```

Gauss Jordan Calculator

Enter Matrix

2	1	1	8
4	1	0	11
-2	2	1	3

Calculate

Original Matrix:

```
[ 2.00  1.00  1.00  8.00 ]
[ 4.00  1.00  0.00 11.00 ]
[ -2.00  2.00  1.00  3.00 ]
```

Intermediate Matrix 1:

```
[ 1.00  0.25  0.00  2.75 ]
[ 0.00  0.50  1.00  2.50 ]
[ 0.00  2.50  1.00  8.50 ]
```

Intermediate Matrix 2:

```
[ 1.00  0.00 -0.10  1.90 ]
[ 0.00  1.00  0.40  3.40 ]
[ 0.00  0.00  0.80  0.80 ]
```

Intermediate Matrix 3:

```
[ 1.00  0.00  0.00  2.00 ]
[ 0.00  1.00  0.00  3.00 ]
[ 0.00  0.00  1.00  1.00 ]
```

Final Solution:

```
[ 1.00  0.00  0.00  2.00 ]
[ 0.00  1.00  0.00  3.00 ]
[ 0.00  0.00  1.00  1.00 ]
```

x1 = 2.0, x2 = 3.0, x3 = 1.0