Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Рязанский государственный радиотехнический университет имени
В. Ф. Уткина»

Кафедра вычислительной и прикладной математики

Отчет
о лабораторной работе № 3
по дисциплине
«Вычислительные алгоритмы»
на тему
“Решение систем линейных алгебраических уравнений”

Выполнила: ст. гр. 343 Гаджиева А.В
Проверила:
доц. Проказникова Е.Н.
ас. Щенева Ю.Б.

Рязань 2025

Дата выполнения лабораторной работы: 21.03.2025

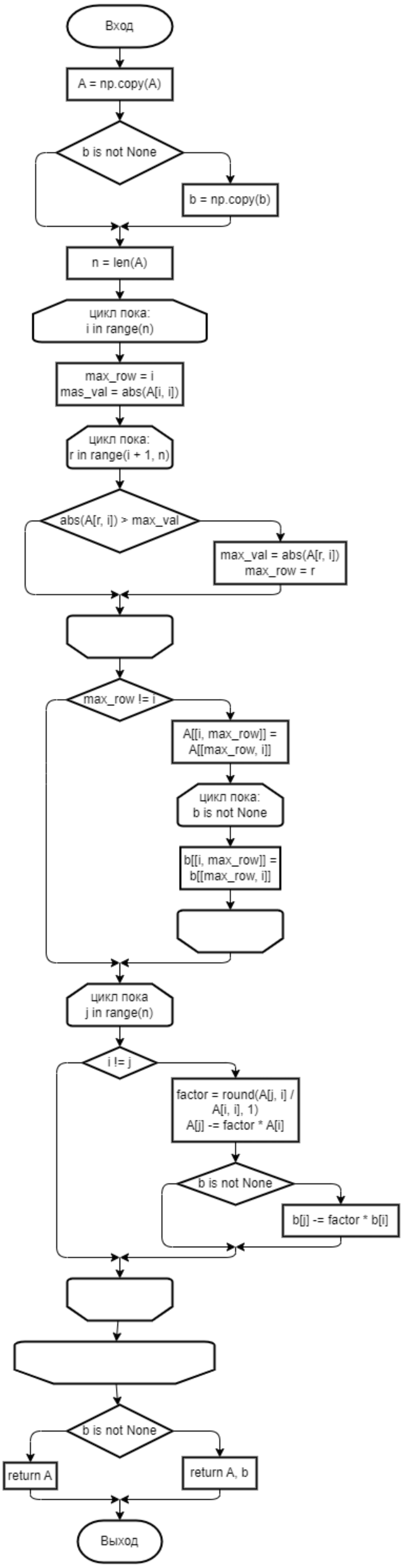
Задание

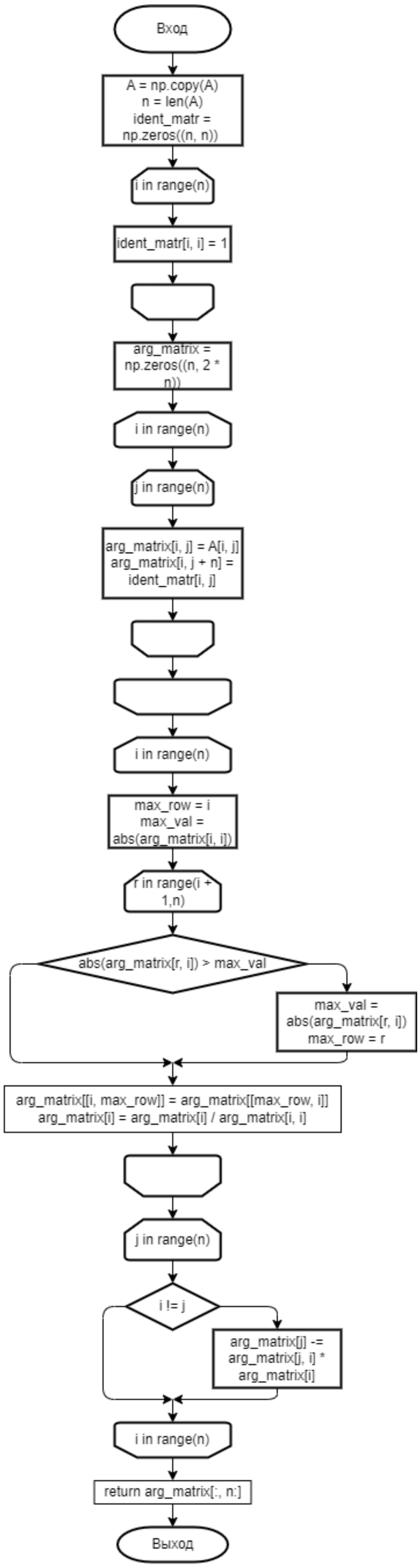
Для заданного варианта решить методом итераций систему уравнений A * X = B. Для остановки процесса последовательных приближений использовать условие: сумма модулей приращений элементов вектора X на последнем шаге итераций меньше ε=0.001.

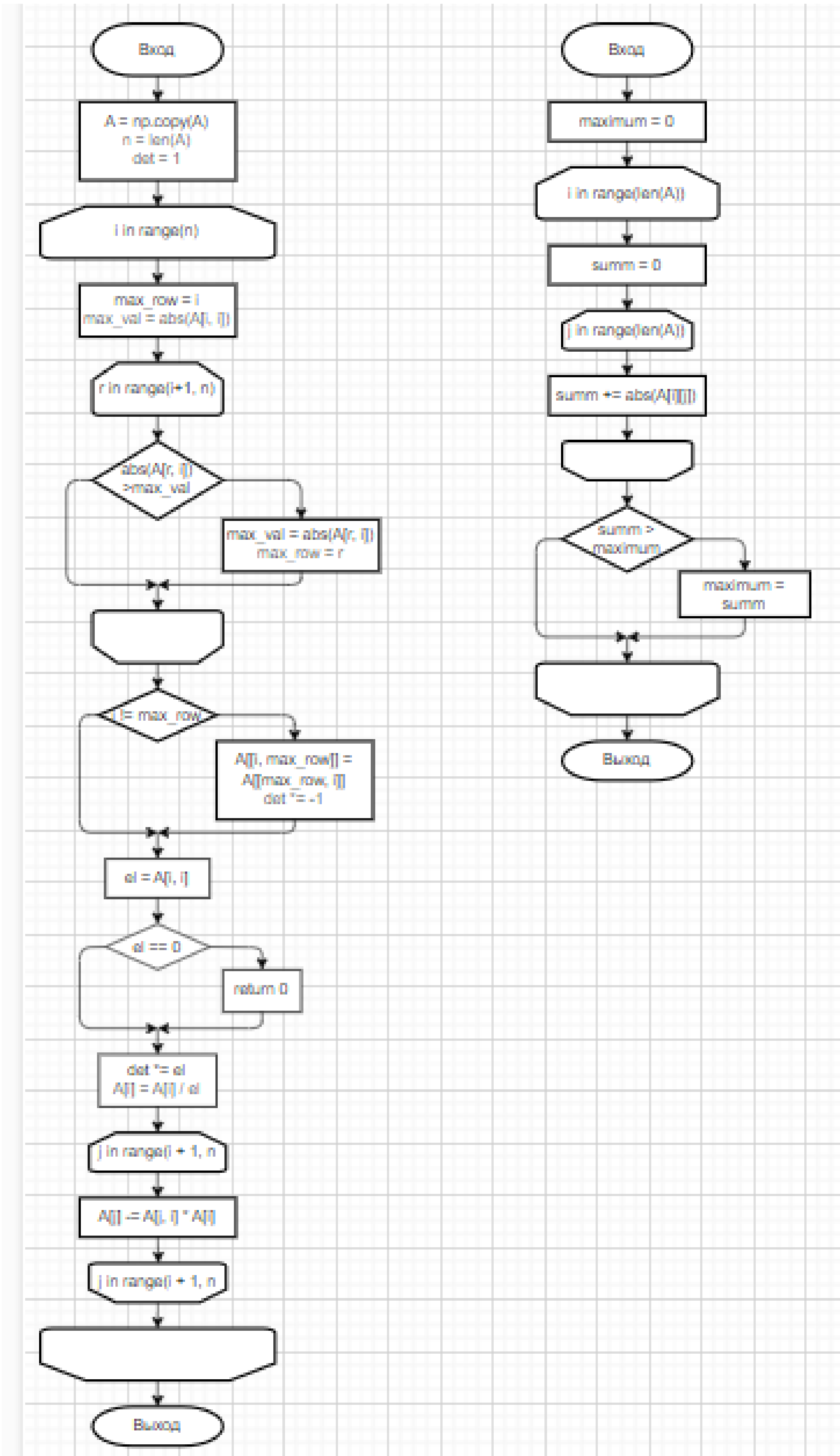
Используя метод Гаусса, вычислить определитель и число обусловленности матрицы A.

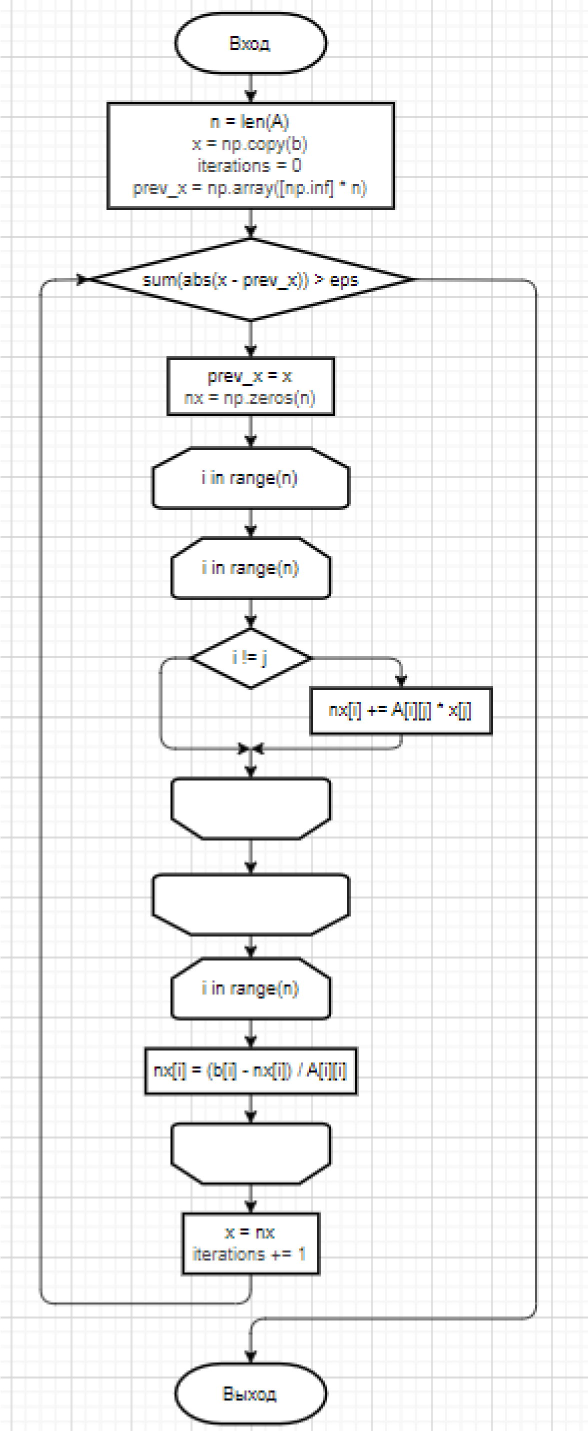
$$\begin{cases} 6{,}1x_1 + 6{,}2x_2 - 6{,}3x_3 + 6{,}4x_4 = 6{,}5, \\ 1{,}1x_1 - 1{,}5x_2 + 2{,}2x_3 - 3{,}8x_4 = 4{,}2, \\ 5{,}1x_1 - 5{,}0x_2 + 4{,}9x_3 - 4{,}8x_4 = 4{,}7, \\ 1{,}8x_1 + 1{,}9x_2 + 2{,}0x_3 - 2{,}1x_4 = 2{,}2. \end{cases}$$

Схемы алгоритмов программы






Вход
A = np.copy(A)
n = len(A)
det = 1
i in range(n)
max_row = i
max_val = abs(A[i, i])
r in range(i+1, n)
abs(A[r, i])
>max_val
max_val = abs(A[r, i])
max_row = r
i != max_row
A[[i, max_row]] =
A[[max_row, i]]
det *= -1
el = A[i, i]
el == 0
return 0
det *= el
A[i] = A[i] / el
j in range(i + 1, n
A[j] -= A[j, i] * A[i]
j in range(i + 1, n
Выход
Вход
maximum = 0
i in range(len(A))
summ = 0
j in range(len(A))
summ += abs(A[i][j])
summ >
maximum
maximum =
summ
Выход


Вход
n = len(A)
x = np.copy(b)
iterations = 0
prev_x = np.array([np.inf] * n)
sum(abs(x - prev_x)) > eps
prev_x = x
nx = np.zeros(n)
i in range(n)
i in range(n)
i != j
nx[i] += A[i][j] * x[j]
i in range(n)
nx[i] = (b[i] - nx[i]) / A[i][i]
x = nx
iterations += 1
Выход

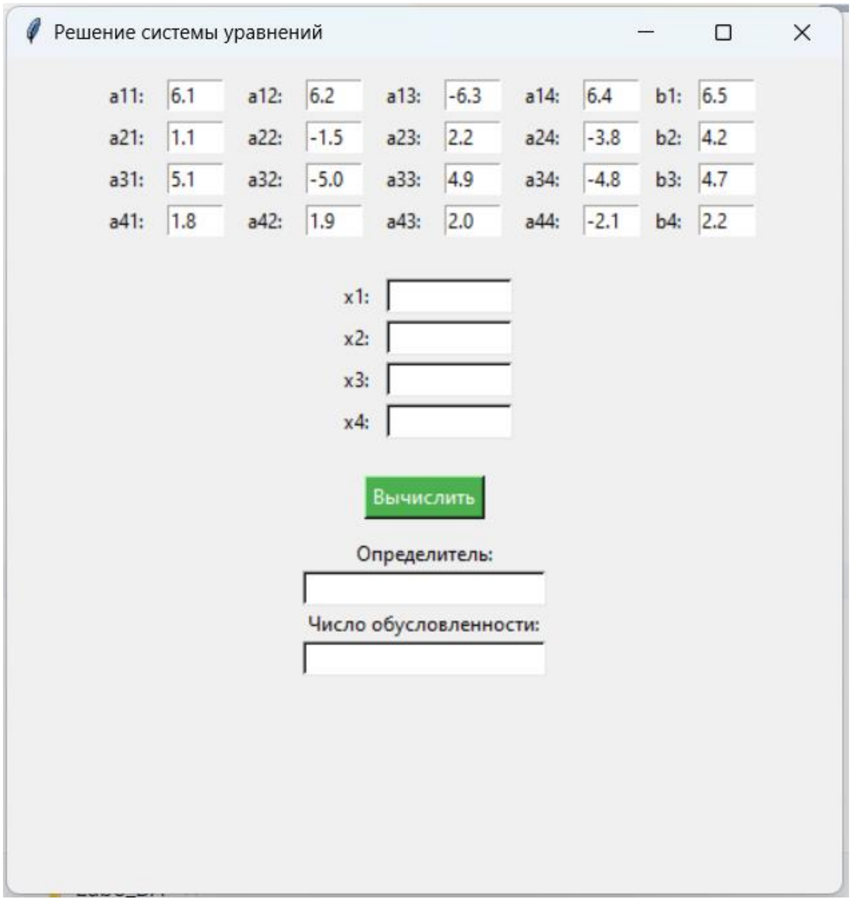
Листинг кода основных методов

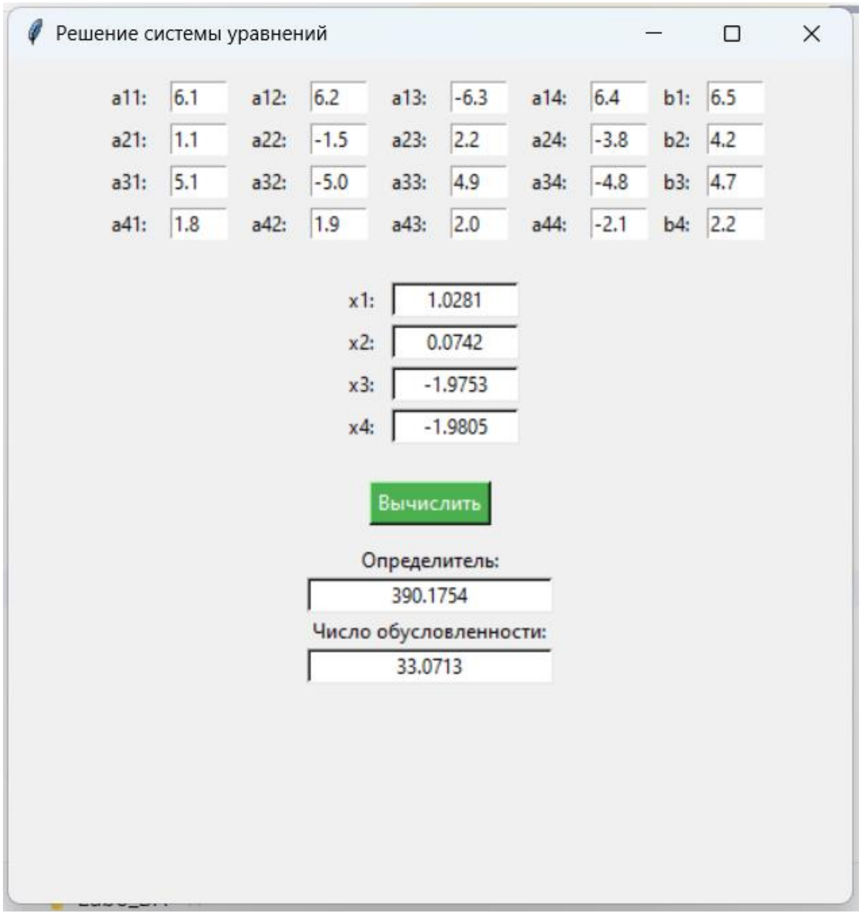
```
import tkinter as tk
import tkinter.messagebox as messagebox
import numpy as np

def make_diagonally_dominant(A, b=None):
    A = np.copy(A)
    if b is not None:
        b = np.copy(b)

    n = len(A)

    for i in range(n):
        max_row = i
        max_val = abs(A[i, i])
        for r in range(i + 1, n):
            if abs(A[r, i]) > max_val:
                max_val = abs(A[r, i])
                max_row = r

        if max_row != i:
            A[[i, max_row]] = A[[max_row, i]]
            if b is not None:
                b[[i, max_row]] = b[[max_row, i]]

        for j in range(n):
            if i != j:
                factor = round(A[j, i] / A[i, i], 1)
                A[j] -= factor * A[i]
                if b is not None:
                    b[j] -= factor * b[i]

    if b is not None:
        return A, b
    else:
        return A

def inverse_matrix(A):
    A = np.copy(A)
    n = len(A)

    ident_matr = np.zeros((n, n))
    for i in range(n):
        ident_matr[i, i] = 1

    arg_matrix = np.zeros((n, 2 * n))
    for i in range(n):
        for j in range(n):
            arg_matrix[i, j] = A[i, j]
            arg_matrix[i, j + n] = ident_matr[i, j]

    for i in range(n):
        max_row = i
        max_val = abs(arg_matrix[i, i])
        for r in range(i + 1, n):
            if abs(arg_matrix[r, i]) > max_val:
                max_val = abs(arg_matrix[r, i])
                max_row = r

        arg_matrix[[i, max_row]] = arg_matrix[[max_row, i]]
        arg_matrix[i] = arg_matrix[i] / arg_matrix[i, i]
```

```python
        for j in range(n):
            if i != j:
                arg_matrix[j] -= arg_matrix[j, i] * arg_matrix[i]

    return arg_matrix[:, n:]

def determinant_matrix(A):
    A = np.copy(A)
    n = len(A)
    det = 1

    for i in range(n):
        max_row = i
        max_val = abs(A[i, i])
        for r in range(i + 1, n):
            if abs(A[r, i]) > max_val:
                max_val = abs(A[r, i])
                max_row = r

        if i != max_row:
            A[[i, max_row]] = A[[max_row, i]]
            det *= -1
        el = A[i, i]
        if el == 0:
            return 0
        det *= el
        A[i] = A[i] / el

        for j in range(i + 1, n):
            A[j] -= A[j, i] * A[i]

    return det

def norm(A):
    maximum = 0

    for i in range(len(A)):
        summ = 0

        for j in range(len(A)):
            summ += abs(A[i][j])

        if summ > maximum:
            maximum = summ

    return maximum

def solve(A, b, eps):
    n = len(A)
    x = np.copy(b)
    iterations = 0
    prev_x = np.array([np.inf] * n)

    while sum(abs(x - prev_x)) > eps:
        prev_x = x
        nx = np.zeros(n)

        for i in range(n):
            for j in range(n):
                if i != j:
                    nx[i] += A[i][j] * x[j]

        for i in range(n):
```

```
            nx[i] = (b[i] - nx[i]) / A[i][i]
        x = nx
        iterations += 1

    return x, iterations

def calculate():
    try:
        # Read matrix A from input fields
        a_values = []
        for i in range(matrix_size):
            row_values = []
            for j in range(matrix_size):
                value = float(a_entries[i][j].get())
                row_values.append(value)
            a_values.append(row_values)
        A = np.array(a_values)

        # Read vector b from input fields
        b_values = []
        for i in range(matrix_size):
            value = float(b_entries[i].get())
            b_values.append(value)
        b = np.array(b_values)

        eps = 1e-4 # Default epsilon value

        # Perform calculations
        new_A, new_b = make_diagonally_dominant(A, b)
        x, iterations = solve(new_A, new_b, eps)
        det_A = determinant_matrix(A)
        norma_a = norm(A)
        inv_A = inverse_matrix(A)
        norma_inv_A = norm(inv_A)
        condition_number = norma_a * norma_inv_A


        # Update result labels
        for i in range(matrix_size):
            x_labels[i].config(text=f"{x[i]:.4f}")
        determinant_label.config(text=f"{det_A:.4f}")
        condition_number_label.config(text=f"{condition_number:.4f}")


    except ValueError:
        messagebox.showerror("Ошибка", "Пожалуйста, введите корректные
числовые значения.")
    except np.linalg.LinAlgError:
        messagebox.showerror("Ошибка", "Невозможно вычислить обратную матрицу
(матрица вырождена).")
    except Exception as e:
         messagebox.showerror("Ошибка", f"Произошла ошибка: {e}")


default_A = np.array([
    [6.1, 6.2, -6.3, 6.4],
    [1.1, -1.5, 2.2, -3.8],
    [5.1, -5.0, 4.9, -4.8],
    [1.8, 1.9, 2.0, -2.1]
])

default_b = np.array([6.5, 4.2, 4.7, 2.2])
```

Графический интерфейс программы





Проверка правильности работы программы

Решение системы

Запишем систему в виде расширенной матрицы:

6.1	6.2	-6.3	6.4	6.5
1.1	-1.5	2.2	-3.8	4.2
5.1	-5.0	4.9	-4.8	4.7
1.8	1.9	2.0	-2.1	2.2

Работаем со столбцом №1.

Умножим 1-ю строку на (-0.18). Добавим 2-ю строку к 1-й:

0	-2.62	3.34	-4.95	3.03
1.1	-1.5	2.2	-3.8	4.2
5.1	-5.0	4.9	-4.8	4.7
1.8	1.9	2.0	-2.1	2.2

Умножим 2-ю строку на (-4.636). Добавим 3-ю строку к 2-й:

0	-2.62	3.34	-4.95	3.03
0	1.95	-5.3	12.82	-14.77
5.1	-5.0	4.9	-4.8	4.7
1.8	1.9	2.0	-2.1	2.2

Умножим 3-ю строку на (-0.353). Добавим 4-ю строку к 3-й:

0	-2.62	3.34	-4.95	3.03
0	1.95	-5.3	12.82	-14.77
0	3.66	0.27	-0.41	0.54
1.8	1.9	2.0	-2.1	2.2

Работаем со столбцом №2.

Умножим 1-ю строку на (0.747). Добавим 2-ю строку к 1-й:

0	0	-2.81	9.12	-12.51
0	1.95	-5.3	12.82	-14.77
0	3.66	0.27	-0.41	0.54
1.8	1.9	2.0	-2.1	2.2

Умножим 2-ю строку на (-1.875). Добавим 3-ю строку к 2-й:

0	0	-2.81	9.12	-12.51
0	0	10.21	-24.44	28.24
0	3.66	0.27	-0.41	0.54
1.8	1.9	2.0	-2.1	2.2

Работаем со столбцом №3.

Умножим 1-ю строку на (3.633). Добавим 2-ю строку к 1-й:

0	0	0	8.7	-17.22
0	0	10.21	-24.44	28.24
0	3.66	0.27	-0.41	0.54
1.8	1.9	2.0	-2.1	2.2

В итоге получаем:

0	0	0	8.7	-17.22
0	0	10.21	-24.44	28.24
0	3.66	0.27	-0.41	0.54
1.8	1.9	2.0	-2.1	2.2

Теперь исходную систему можно записать так:

x_4 = -17.223/8.696

x_3 = [28.24-(- 24.44x_4)]/10.208

x_2 = [0.541-(0.27x_3 - 0.41x_4)]/3.665

x_1 = [2.2-(1.9x_2 + 2x_3 - 2.1x_4)]/1.8

Из 1-й строки выражаем x_4

$$x_4 = \frac{-17.223}{8.696} = -1.981$$

Из 2-й строки выражаем x_3

$$x_3 = \frac{28.24 - (-24.44)\cdot(-1.98)}{10.208} = \frac{-20.163}{10.208} = -1.975$$

Из 3-й строки выражаем x_2

$$x_2 = \frac{0.541 - 0.27\cdot(-1.98) - (-0.41)\cdot(-1.98)}{3.665} = \frac{0.272}{3.665} = 0.0742$$

Из 4-й строки выражаем x_1

$$x_1 = \frac{2.2 - 1.9\cdot 0.074 - 2.0\cdot(-1.98) - (-2.1)\cdot(-1.98)}{1.8} = \frac{1.851}{1.8} = 1.028$$

X(1.028;0.0742;-1.975;-1.981)

Определитель матрицы

$$\det \mathbf{A} = \begin{vmatrix} 6.1 & 6.2 & -6.3 & 6.4 \\ 1.1 & -1.5 & 2.2 & -3.8 \\ 5.1 & -5 & 4.9 & -4.8 \\ 1.8 & 1.9 & 2 & -2.1 \end{vmatrix} =$$

от 2 строки отнимаем 1 строку, умноженную на $\frac{11}{61}$; от 3 строки отнимаем 1 строку, умноженную на $\frac{51}{61}$; от 4 строки отнимаем 1 строку, умноженную на $\frac{18}{61}$

$$= \begin{vmatrix} 6.1 & 6.2 & -6.3 & 6.4 \\ 0 & -2\frac{377}{610} & 3\frac{41}{122} & -4\frac{291}{305} \\ 0 & -10\frac{56}{305} & 10\frac{51}{305} & -10\frac{46}{305} \\ 0 & \frac{43}{610} & 3\frac{262}{305} & -3\frac{603}{610} \end{vmatrix} =$$

от 3 строки отнимаем 2 строку, умноженную на $3\frac{1421}{1597}$; к 4 строке добавляем 2 строку, умноженную на $\frac{43}{1597}$

$$= \begin{vmatrix} 6.1 & 6.2 & -6.3 & 6.4 \\ 0 & -2\frac{377}{610} & 3\frac{41}{122} & -4\frac{291}{305} \\ 0 & 0 & -2\frac{6463}{7985} & 9\frac{191}{1597} \\ 0 & 0 & 3\frac{15153}{15970} & -4\frac{1947}{15970} \end{vmatrix} =$$

к 4 строке добавляем 3 строку, умноженную на $1\frac{18197}{44866}$

$$= \begin{vmatrix} 6.1 & 6.2 & -6.3 & 6.4 \\ 0 & -2\frac{377}{610} & 3\frac{41}{122} & -4\frac{291}{305} \\ 0 & 0 & -2\frac{6463}{7985} & 9\frac{191}{1597} \\ 0 & 0 & 0 & 8\frac{156237}{224330} \end{vmatrix} = 6.1\cdot(-2\frac{377}{610})\cdot(-2\frac{6463}{7985})\cdot 8\frac{156237}{224330} = 390.1754$$

Число обусловленности

Начальная запись

$$\begin{pmatrix} 6.1 & 6.2 & -6.3 & 6.4 \\ 1.1 & -1.5 & 2.2 & -3.8 \\ 5.1 & -5 & 4.9 & -4.8 \\ 1.8 & 1.9 & 2 & -2.1 \end{pmatrix}$$

Ответ

33.0713