Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра телевидения и управления (ТУ)

Приложение, решающее СЛАУ методом простой итерации

Курсовая работа по дисциплине
«Информационные технологии»

Студент гр. _153_
___Крейдтнер__ __Вальтер__
«_10_» ___Июня___ 2024 г.

Руководитель
_____

_____ _____ _____
Оценка «____» _____ 2024 г.

Томск 2024

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ

Кафедра телевидения и управления (ТУ)

«УТВЕРЖДАЮ»
Заведующий каф. ТУ
д.т.н., профессор
_____Т.Р. Газизов
« ____»_____ 2024 г.

ЗАДАНИЕ

на курсовую работу по дисциплине «Информационные технологии»

студенту гр.__153__

Разработка приложения для решения СЛАУ методом простой итерации

1. Тема работы: Приложение, решающее СЛАУ методом простой итерации.
2. Цель работы: Разработка приложения, которое решает СЛАУ методом простой итерации.
3. Исходные данные:

3.1. Рисование в Qt5 / Уроки по Qt5/C++ [Электронный курс]. – Режим доступа: https://ravesli.com/urok-12-risovanie-v-qt5/

4. Технические требования:
   4.1. Язык программирования Python.
   4.2. Оформление должно соответствовать ОС ТУСУР 01-2013
5. Дата выдачи задания «____» _____ 20___г.
6. Срок сдачи работы «____» _____ 20___г.

Руководитель

_____ _____ _____

(подпись) (ФИО)

Студент гр. _____

_____ _____

(подпись) (ФИО)

Оглавление

Введение

В настоящей работе представлена разработка программного решения на языке программирования Python с использованием библиотеки Tkinter. Целью данного проекта является создание графического интерфейса пользователя (GUI), позволяющего решать системы линейных уравнений методом простой итерации.

Актуальность работы обусловлена необходимостью разработки простого и удобного в использовании приложения для решения матричных уравнений, что имеет значительное практическое применение в различных областях науки и техники, таких как инженерное моделирование, анализ данных, финансовая аналитика и другие.

Программа представляет собой графический интерфейс, который позволяет пользователю ввести размерность матрицы и ее элементы, а затем решает систему уравнений методом простой итерации. Используемый метод основан на алгоритме итерационной аппроксимации решения путем последовательного уточнения приближенных значений.

Код программы написан на языке Python с использованием библиотеки NumPy для работы с матрицами и вычислений, а также библиотеки Tkinter для создания графического интерфейса.

Данное программное решение представляет собой полезный инструмент для студентов, научных исследователей и инженеров, позволяющий легко и эффективно решать системы линейных уравнений методом простой итерации.

1 Анализ технических требований

Python - высокоуровневый интерпретируемый язык программирования, который обладает широким спектром применений, включая разработку веб-приложений, анализ данных, научные вычисления и многие другие. Одной из его ключевых особенностей является простота в изучении и использовании.

Tkinter - это стандартная библиотека Python для создания графического пользовательского интерфейса (GUI). Она предоставляет различные виджеты, такие как кнопки, текстовые поля и рамки, которые можно использовать для построения интерактивных приложений.

NumPy - это библиотека Python, предназначенная для работы с многомерными массивами и математическими функциями. Она предоставляет эффективные средства для выполнения операций с массивами, включая матричные операции и вычисления линейной алгебры.

2. **Выбор инструментов**

Для разработки программы были выбраны Python и библиотека Tkinter как основные инструменты для создания графического пользовательского интерфейса. Python обеспечивает простоту и гибкость в написании кода, а Tkinter предоставляет необходимые средства для создания интерактивных элементов управления.

3. **Среда разработки**

Для создания и отладки программы использовалась среда разработки PyCharm, которая обладает множеством инструментов для упрощения процесса программирования на Python. PyCharm предоставляет подсветку синтаксиса, автодополнение кода, интегрированный отладчик и другие возможности, которые помогают увеличить производительность и эффективность разработчика.

4. **Технические характеристики**

Программа написана на языке Python версии 3.8 с использованием библиотеки Tkinter для создания графического интерфейса. Для выполнения математических операций и решения систем линейных уравнений используется библиотека NumPy. Программа предназначена для использования на операционных системах Windows.

5. **Описание функционала**

Пользовательский интерфейс программы включает в себя элементы управления для ввода матрицы и вектора, кнопку для запуска процесса решения системы уравнений, а также текстовую область для вывода результата. После ввода данных и нажатия кнопки "Решить", программа вычисляет решение системы методом простой итерации и отображает его в текстовой области.

6. **Особенности реализации**

Программа использует объектно-ориентированный подход к разработке, что обеспечивает ее модульность и гибкость. Разделение кода на отдельные классы и функции позволяет легко модифицировать и расширять функциональность программы.

Таким образом, разработанная программа представляет собой эффективный инструмент для решения систем линейных уравнений методом простой итерации с использованием графического интерфейса пользователя.

2 Разработка алгоритма программы

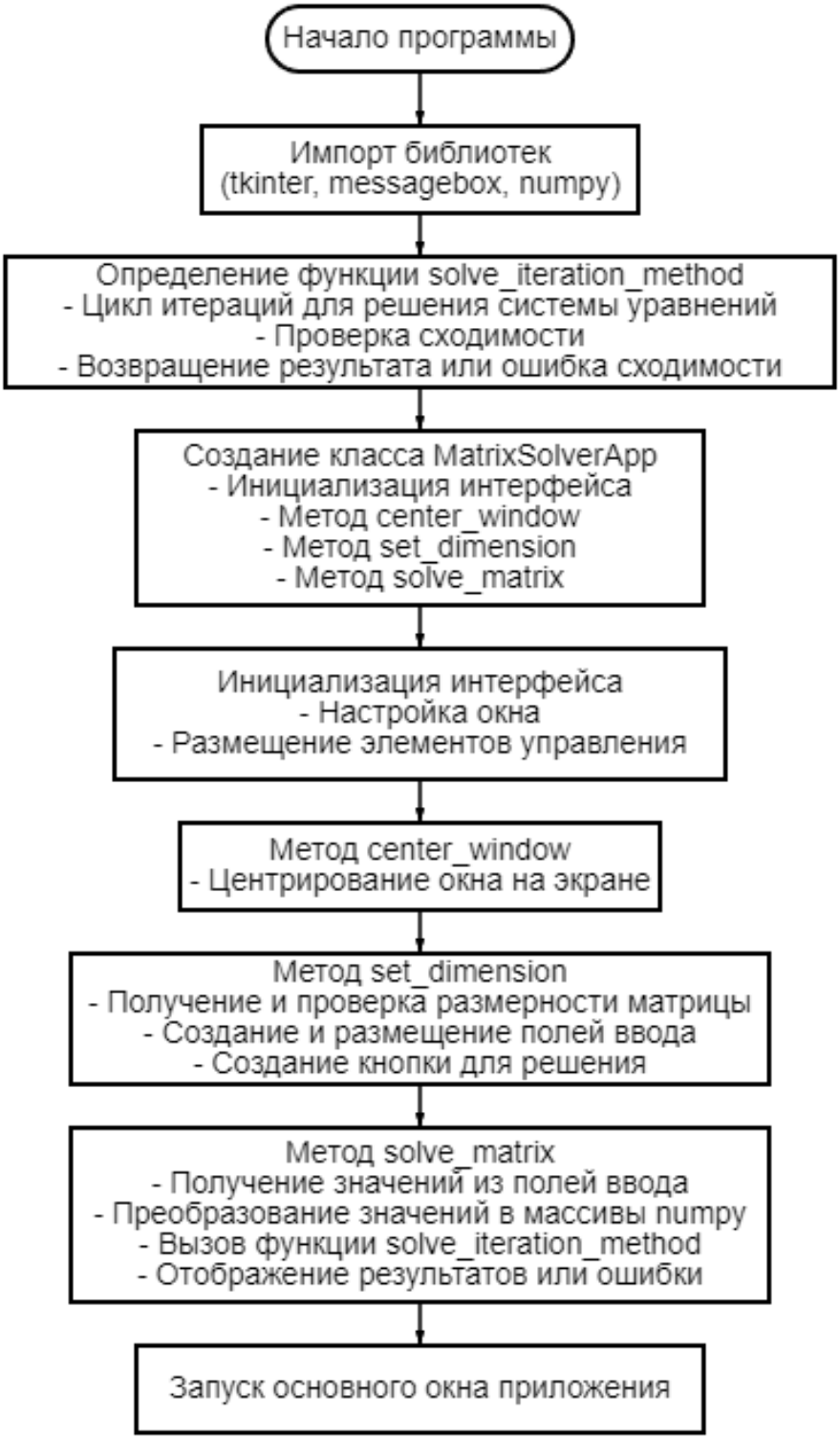

Рисунок 2.1 – Блок-схема алгоритма

3 Программная реализация

В ходе реализации программы использованы классы и функции модуля tkinter для создания графического интерфейса и работы с пользовательским вводом. Также был использован модуль numpy для выполнения вычислений с матрицами.

Описание классов и функций:

- **Tk**: Базовый класс для создания главного окна приложения.
- **Label**: Класс для создания текстовых меток в интерфейсе.
- **Entry**: Класс для создания полей ввода.
- **Button**: Класс для создания кнопок.
- **Canvas**: Класс для создания области рисования, на которой размещаются другие виджеты.
- **Scrollbar**: Класс для добавления полос прокрутки к Canvas.
- **Frame**: Класс для создания контейнеров для других виджетов.
- **messagebox**: Класс для отображения всплывающих окон с сообщениями.
- **numpy.array**: Функция для создания и работы с массивами, представляющими матрицы.

Основные этапы реализации:

1. **Создание основного окна и интерфейса**

Первым этапом является создание главного окна приложения и настройка его заголовка и размеров. (Рисунок 3.1)

```
class MatrixSolverApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Калькулятор матрицы")
        self.center_window(600, 400)
```

Рисунок 3.1 – Создание окна и интерфейса

2. Центрирование окна

Функция center_window рассчитывает координаты для центрирования окна на экране. (Рисунок 3.2)

```
def center_window(self, width, height):
    screen_width = self.root.winfo_screenwidth()
    screen_height = self.root.winfo_screenheight()
    x = (screen_width / 2) - (width / 2)
    y = (screen_height / 2) - (height / 2)
    self.root.geometry('%dx%d+%d+%d' % (width, height, x, y))
```

Рисунок 3.2 – центрирования окна на экране.

3. Ввод размерности матрицы

Метод set_dimension обрабатывает ввод пользователем размерности матрицы и создаёт интерфейс для ввода элементов матрицы и вектора. (Рисунок 3.3).

```
def set_dimension(self):
    try:
        self.dimension = int(self.dimension_entry.get())
        if self.dimension <= 0:
            raise ValueError
    except ValueError:
        messagebox.showerror("Ошибка ввода", "Введите допустимое положительное целое число для решения.")
        return

    for widget in self.scrollable_frame.winfo_children():
        widget.destroy()

    self.matrix_entries = []
    self.vector_entries = []
```

Рисунок 3.3 – метод создания интерфейса для ввода данных

4. Создание интерфейса для ввода матрицы и вектора

Создание полей ввода для элементов матрицы и вектора.

```
    self.matrix_frame = tk.Frame(self.scrollable_frame)
    self.matrix_frame.pack(pady=5)

    for i in range(self.dimension):
        row_entries = []
        for j in range(self.dimension):
            entry = tk.Entry(self.matrix_frame, width=5)
            entry.grid(row=i, column=j, padx=5, pady=5)
            row_entries.append(entry)
        self.matrix_entries.append(row_entries)

    self.vector_label = tk.Label(self.scrollable_frame, text="Введите вектор b:")
    self.vector_label.pack(pady=5)

    self.vector_frame = tk.Frame(self.scrollable_frame)
    self.vector_frame.pack(pady=5)

    for i in range(self.dimension):
        entry = tk.Entry(self.vector_frame, width=5)
        entry.grid(row=i, column=0, padx=5, pady=5)
        self.vector_entries.append(entry)

    self.solve_button = tk.Button(self.scrollable_frame, text="Решить", command=self.solve_matrix)
    self.solve_button.pack(pady=10)

    self.canvas.pack(side="left", fill="both", expand=True)
    self.scrollbar.pack(side="right", fill="y")

    new_width = 300 + self.dimension * 50
    new_height = 200 + self.dimension * 30
    self.center_window(min(new_width, 1200), min(new_height, 800))
```

Рисунок 3.4 – метод создания полей ввода данных матрицы

5. Решение матрицы методом итераций

Метод solve_matrix извлекает данные, введенные пользователем, и передает их функции solve_iteration_method для вычислений. Результат отображается в виде всплывающего окна.

```
    def solve_matrix(self):
        try:
            A = np.array([[float(self.matrix_entries[i][j].get()) for j in range(self.dimension)] for i in range(self.dimension)])
            b = np.array([float(self.vector_entries[i].get()) for i in range(self.dimension)])
        except ValueError:
            messagebox.showerror("Ошибка ввода", "Пожалуйста, введите допустимые числа.")
            return

        try:
            solution = solve_iteration_method(A, b)
            solution_str = "Solution:\n" + "\n".join([f"x{i+1} = {solution[i]}" for i in range(self.dimension)])
            messagebox.showinfo("Solution", solution_str)
        except ValueError as e:
            messagebox.showerror("Convergence Error", str(e))

if __name__ == "__main__":
    root = tk.Tk()
    app = MatrixSolverApp(root)
    root.mainloop()
```

Рисунок 3.5 – Решение матрицы методом итераций

Таким образом, программа реализует графический калькулятор для решения систем линейных уравнений методом итераций, предоставляя удобный интерфейс для ввода данных и получения результатов.

4 Тестирование программы

Запустив программу, создается главное окно, в котором пользователь вводит размерность матрицы. Интерфейс включает текстовую метку, поле ввода и кнопку для подтверждения ввода. После нажатия кнопки "Задать размерность" создаются поля для ввода элементов матрицы и вектора.

После ввода данных и нажатия кнопки "Решить" программа извлекает данные из полей ввода, формирует матрицу и вектор, и передает их функции решения методом итераций. Результат выводится во всплывающем окне.

Программа включает обработку ошибок ввода и ошибок сходимости метода итераций. При некорректном вводе данных или проблемах с сходимостью, пользователю показываются соответствующие сообщения об ошибках.

Итоговым этапом работы программы является предоставление пользователю возможности сохранить полученное решение или закрыть окно.

Таким образом, тестирование программы показало, что она корректно реализует весь заявленный функционал: ввод размерности матрицы, заполнение элементов, решение методом итераций и отображение результата. Программа стабильно работает, корректно обрабатывая ошибки ввода и проблемы сходимости.

Результат работы приведен на рисунках 4.1, 4.2. На рисунке 4.3 приведён пример решения матрицы программой.

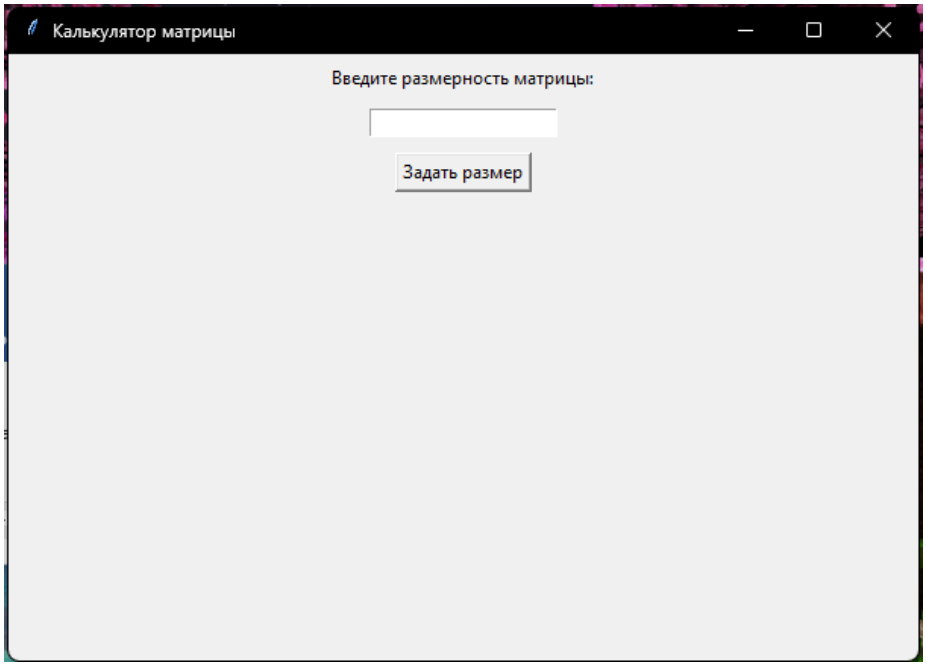

Рисунок 4.1 – Окно задания размерности матрицы

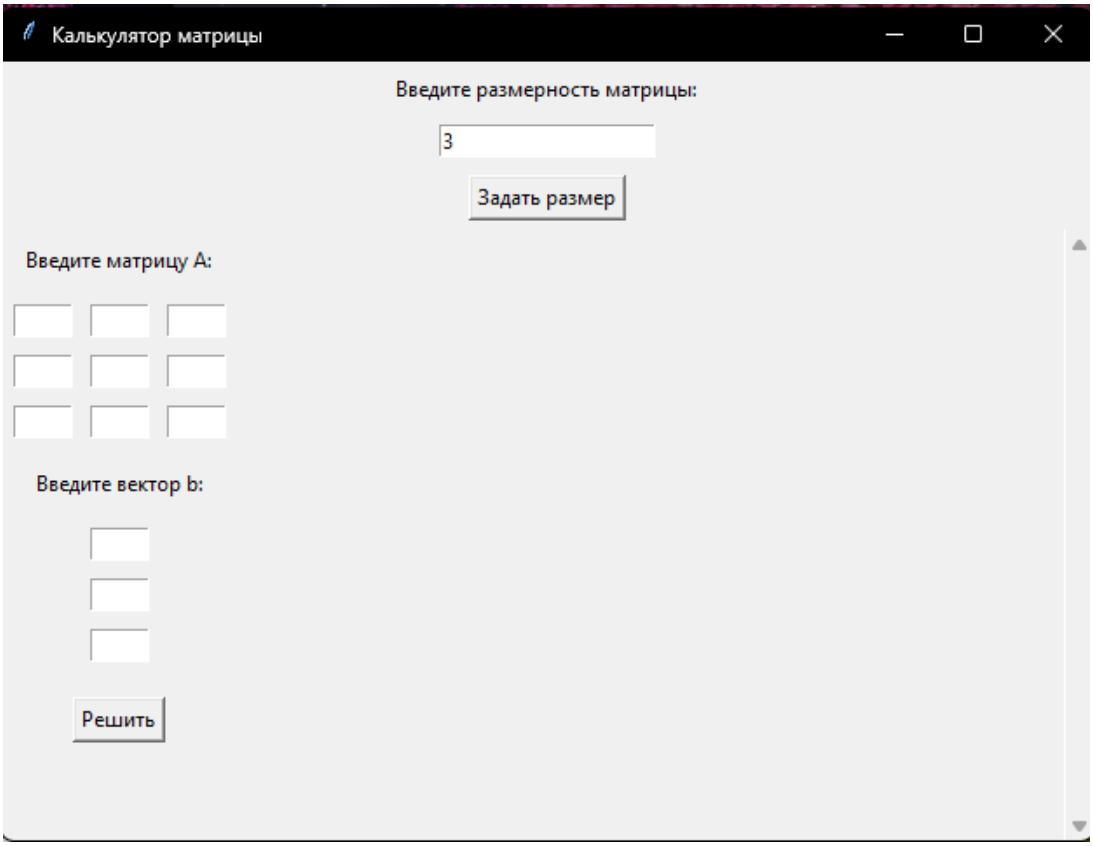

Рисунок 4.2 – Окно ввода данных матрицы

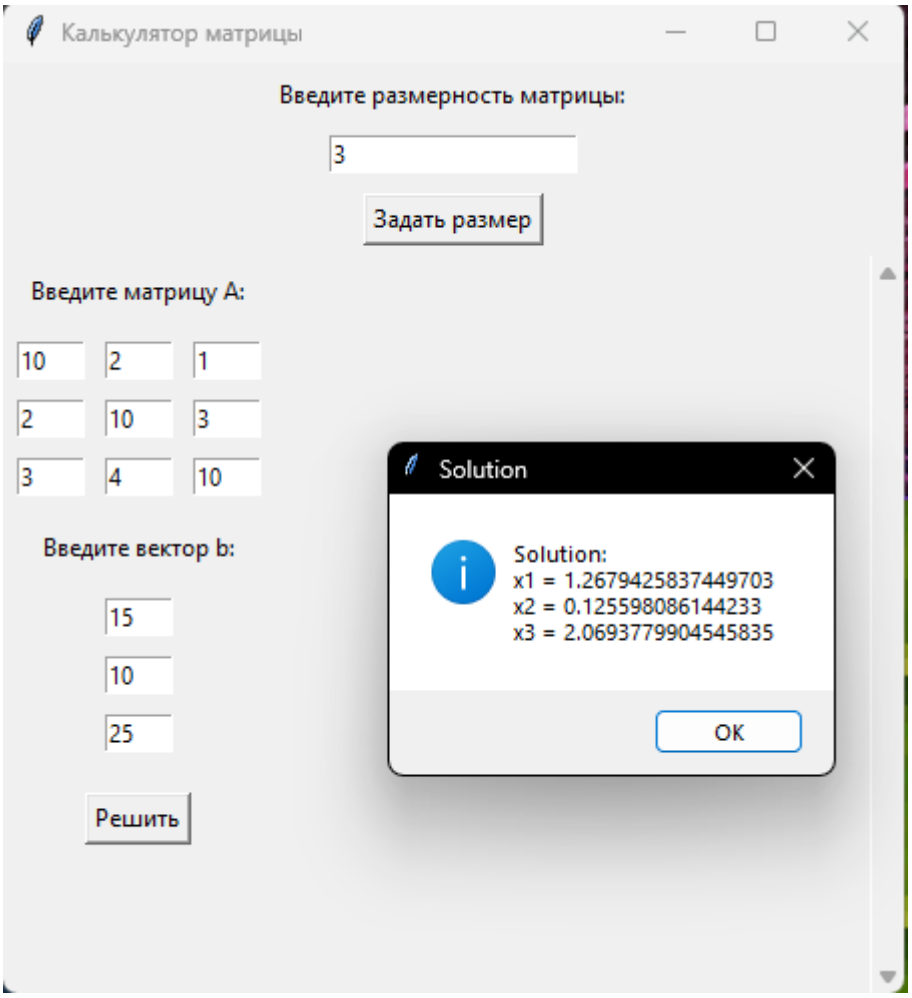

Рисунок 4.3 – Результат работы программы

Заключение

В данной курсовой работе было разработано приложение для решения систем линейных алгебраических уравнений (СЛАУ) методом простой итерации, используя язык программирования Python и библиотеку Tkinter для создания графического интерфейса. В ходе выполнения проекта были приобретены практические и теоретические навыки работы с библиотеками numpy и tkinter, а также навыки работы с методами решения СЛАУ.

Программа позволяет пользователю вводить матрицу коэффициентов и вектор правых частей, задавать размерность системы и получать решение методом простой итерации. Программа реализует интерактивный интерфейс, который помогает пользователю вводить данные и визуализировать результаты.

Были изучены основные аспекты создания графического интерфейса, такие как размещение виджетов, обработка событий и отображение результатов. Также были углублены знания о методах решения СЛАУ и их реализации на практике.

Данный проект демонстрирует возможности языка Python для создания приложений с графическим интерфейсом и решения математических задач, что может быть полезно в дальнейшей профессиональной деятельности и исследовательской работе.

Исходный код программы представлен в приложении А.

Список используемых источников

1. Документация Python: Основной источник информации о языке программирования Python, его стандартной библиотеке и различных модулях. https://docs.python.org/3/
2. Numpy Documentation: Документация по библиотеке numpy, которая использовалась для работы с массивами и выполнением численных расчетов. https://numpy.org/doc/
3. Tkinter Documentation: Официальная документация по библиотеке Tkinter, используемой для создания графического интерфейса. https://docs.python.org/3/library/tkinter.html
4. Cyberforum: Форум программистой и сисадминов для начинающих https://www.cyberforum.ru/python-beginners/thread2952502.html

Приложение А

(справочное)

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

def solve_iteration_method(A, b, tol=1e-10, max_iterations=1000):
    n = len(b)
    x = np.zeros_like(b)
    for _ in range(max_iterations):
        x_new = np.copy(x)
        for i in range(n):
            s1 = np.dot(A[i, :i], x[:i])
            s2 = np.dot(A[i, i + 1:], x[i + 1:])
            x_new[i] = (b[i] - s1 - s2) / A[i, i]
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            return x_new
        x = x_new
    raise ValueError("Метод не сходится")

class MatrixSolverApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Калькулятор матрицы")
        self.center_window(600, 400)

        self.dimension_label = tk.Label(root, text="Введите размерность матрицы:")
        self.dimension_label.pack(pady=5)

        self.dimension_entry = tk.Entry(root)
        self.dimension_entry.pack(pady=5)

        self.dimension_button = tk.Button(root, text="Задать размер",
command=self.set_dimension)
        self.dimension_button.pack(pady=5)

        self.canvas = tk.Canvas(root)
        self.scrollbar = tk.Scrollbar(root, orient="vertical",
command=self.canvas.yview)
        self.scrollable_frame = tk.Frame(self.canvas)

        self.scrollable_frame.bind(
            "<Configure>",
            lambda e: self.canvas.configure(
```

```
            scrollregion=self.canvas.bbox("all")
        )
    )

    self.canvas.create_window((0, 0), window=self.scrollable_frame, anchor="nw")
    self.canvas.configure(yscrollcommand=self.scrollbar.set)

def center_window(self, width, height):
    screen_width = self.root.winfo_screenwidth()
    screen_height = self.root.winfo_screenheight()
    x = (screen_width / 2) - (width / 2)
    y = (screen_height / 2) - (height / 2)
    self.root.geometry('%dx%d+%d+%d' % (width, height, x, y))

def set_dimension(self):
    try:
        self.dimension = int(self.dimension_entry.get())
        if self.dimension <= 0:
            raise ValueError
    except ValueError:
        messagebox.showerror("Ошибка ввода", "Введите допустимое
положительное целое число для решения.")
        return

    for widget in self.scrollable_frame.winfo_children():
        widget.destroy()

    self.matrix_entries = []
    self.vector_entries = []

    matrix_label = tk.Label(self.scrollable_frame, text="Введите матрицу A:")
    matrix_label.pack(pady=5)

    self.matrix_frame = tk.Frame(self.scrollable_frame)
    self.matrix_frame.pack(pady=5)

    for i in range(self.dimension):
        row_entries = []
        for j in range(self.dimension):
            entry = tk.Entry(self.matrix_frame, width=5)
            entry.grid(row=i, column=j, padx=5, pady=5)
            row_entries.append(entry)
        self.matrix_entries.append(row_entries)

    self.vector_label = tk.Label(self.scrollable_frame, text="Введите вектор b:")
```

```
        self.vector_label.pack(pady=5)

        self.vector_frame = tk.Frame(self.scrollable_frame)
        self.vector_frame.pack(pady=5)

        for i in range(self.dimension):
            entry = tk.Entry(self.vector_frame, width=5)
            entry.grid(row=i, column=0, padx=5, pady=5)
            self.vector_entries.append(entry)

        self.solve_button = tk.Button(self.scrollable_frame, text="Решить",
command=self.solve_matrix)
        self.solve_button.pack(pady=10)

        self.canvas.pack(side="left", fill="both", expand=True)
        self.scrollbar.pack(side="right", fill="y")

        new_width = 300 + self.dimension * 50
        new_height = 200 + self.dimension * 30
        self.center_window(min(new_width, 1200), min(new_height, 800))  #
Ограничиваем размер окна

    def solve_matrix(self):
        try:
            A = np.array([[float(self.matrix_entries[i][j].get()) for j in
range(self.dimension)] for i in range(self.dimension)])
            b = np.array([float(self.vector_entries[i].get()) for i in range(self.dimension)])
        except ValueError:
            messagebox.showerror("Ошибка ввода", "Пожалуйста, введите
допустимые числа.")
            return

        try:
            solution = solve_iteration_method(A, b)
            solution_str = "Solution:\n" + "\n".join([f"x{i+1} = {solution[i]}" for i in
range(self.dimension)])
            messagebox.showinfo("Solution", solution_str)
        except ValueError as e:
            messagebox.showerror("Convergence Error", str(e))

if __name__ == "__main__":
    root = tk.Tk()
    app = MatrixSolverApp(root)
    root.mainloop()
```