СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 2](#_Toc148975742)

[1 Постановка задачи 3](#_Toc148975743)

[1.1 Метод Гаусса. 4](#_Toc148975744)

[1.2 Системные требования, предъявляемые к программе 6](#_Toc148975745)

[2 Структура программы 8](#_Toc148975746)

[3 Схемы алгоритма программы 11](#_Toc148975747)

[4 Отладка программы 12](#_Toc148975748)

[4.1 Синтаксические ошибки, обнаруженные в программе. 12](#_Toc148975749)

[4.2 Ошибки компоновки, обнаруженные в программе 12](#_Toc148975750)

[5 Оптимизация программы 13](#_Toc148975751)

[6 Тестирование программы 17](#_Toc148975752)

[6.1 Тестирование в нормальных условиях 17](#_Toc148975753)

[6.2 Тестирование в экстремальных условиях 19](#_Toc148975754)

[6.3 Тестирование в исключительных ситуациях 20](#_Toc148975755)

[7 Руководство пользователя 23](#_Toc148975756)

[ЗАКЛЮЧЕНИЕ 25](#_Toc148975757)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 26](#_Toc148975758)

[ПРИЛОЖЕНИЕ А 27](#_Toc148975759)

[ПРИЛОЖЕНИЕ Б 33](#_Toc148975760)

ВВЕДЕНИЕ

Линейная алгебра является одной из фундаментальных областей математики, которая находит широкое применение в различных сферах науки и техники. Квадратные матрицы, в частности, играют важную роль в решении систем линейных уравнений, определении собственных значений и векторов, а также в различных алгоритмах и методах обработки данных.

Одним из ключевых методов, используемых для обращения квадратных матриц, является схема Гаусса. Этот метод позволяет преобразовать исходную матрицу в единичную матрицу или нахождение её обратной, что является важной задачей в решении систем линейных уравнений, в вычислениях с матрицами и в других приложениях.

В данной курсовой работе будем рассматривать процесс обращения квадратной матрицы с использованием схемы Гаусса на языке программирования Python. Python является мощным и популярным языком программирования, обладающим богатой экосистемой библиотек для работы с матрицами и численными методами. Углубимся в теоретические основы схемы Гаусса, реализуем соответствующий алгоритм на Python.

Программа, созданная в процессе курсового проекта, должна обращать квадратную матрицу с помощью метода Гаусса.

1 Постановка задачи

Целью данного курсового проекта является разработка программного решения на языке программирования Python для обращения квадратных матриц с использованием метода Гаусса.

Для достижения этой цели необходимо выполнить следующие задачи:

1. Изучение теоретических основ схемы Гаусса:

* Понять принцип работы метода Гаусса для обращения квадратных матриц.
* Изучить основные этапы алгоритма, включая приведение матрицы к треугольному виду и обратный ход.

1. Реализация алгоритма на Python:

* Написать программный код, который будет выполнять обращение квадратной матрицы с использованием схемы Гаусса.
* Обеспечить возможность ввода данных для матрицы.

1. Проведение численных экспериментов:

* Сгенерировать несколько тестовых матриц разных размеров и структур.
* Применить разработанный алгоритм к каждой из тестовых матриц.
* Оценить точность полученных результатов.

1. Написание заключения:

* Сформулировать основные выводы и результаты работы.
* Подытожить достижения и ограничения разработанного решения.

Выполнение данных задач позволит достичь поставленной цели и создать полноценное программное решение для обращения квадратных матриц с использованием схемы Гаусса на Python, а также провести его анализ и оценку эффективности.

1.1 Метод Гаусса.

Метод Гаусса, также известный как метод исключения Гаусса или метод приведения к треугольному виду, представляет собой численный метод решения систем линейных уравнений и обращения квадратных матриц. Он основан на преобразованиях элементов матрицы с целью привести её к верхнетреугольному или уменьшенному верхнетреугольному виду, что облегчает нахождение решений систем уравнений или обратной матрицы.

Описание метода Гаусса:

1. Прямой ход (forward elimination): На данном этапе матрица приводится к верхнетреугольному виду путем применения элементарных операций над строками (умножение строки на число и прибавление одной строки к другой). Цель - получить нули под главной диагональю матрицы.
2. Обратный ход (backward substitution): На этом этапе начинается с последней строки верхнетреугольной матрицы и последовательно вычисляются значения переменных, начиная с последней итерации и двигаясь к первой. Это позволяет найти решения системы уравнений.

Формулы для прямого хода метода Гаусса:

Пусть у нас есть система линейных уравнений:

На каждом этапе прямого хода выбираем главный элемент (pivot) и используем его для обнуления элементов под ним. Главный элемент обычно выбирается как наибольший элемент в текущем столбце (чтобы избежать деления на ноль).

Этот процесс можно записать следующим образом:

1. Выберите главный элемент (pivot) в текущем столбце и строке.
2. Поделите всю текущую строку на значение главного элемента, чтобы сделать его равным 1 (процесс нормализации).
3. Для каждой строки ниже текущей строки вычитайте умноженную на определенный коэффициент текущую строку так, чтобы обнулить соответствующий элемент под главным элементом.

Этот процесс продолжается для каждого столбца, двигаясь сверху вниз, пока не получим верхнетреугольную матрицу.

Формулы для обратного хода метода Гаусса:

После прямого хода и получения верхнетреугольной матрицы, можем найти решения системы уравнений методом обратной подстановки.

Для i-ой строки уравнения (последняя строка матрицы) можно записать следующее уравнение:

Отсюда находим :

Затем, двигаясь вверх по строкам, находим остальные переменные.

Таким образом, метод Гаусса позволяет решать системы линейных уравнений и находить обратные матрицы квадратных матриц, приводя их к верхнетреугольному виду и используя обратную подстановку для нахождения значений переменных.

1.2 Системные требования, предъявляемые к программе

Программа должна быть разработана для компьютеров со следующей конфигурацией:

1. Операционная система:

Выполнение программы поддерживается на устройствах с операционными системами семейства Windows и Linux.

1. Аппаратное обеспечение:

* Процессор, совместимый с аппаратной архитектурой x86 или x64.
* Рекомендуется наличие не менее 1 ГБ оперативной памяти (RAM).
* Наличие от 25 МБ свободного дискового пространства.

1. Программное обеспечение:

* Python 3.x (поддержка Python 3.6 и выше).
* Библиотеки Python: tkinter, numpy.

1. Разрешение экрана:

Программа должна корректно работать на экранах с разрешением 1024x768 и выше.

Разработка программы на Python имеет несколько преимуществ, особенно для задачи, подобной созданию программы для вычисления обратной матрицы с графическим интерфейсом:

1. Простота и читаемость кода: Python известен своей читаемостью и простотой написания кода. Это позволяет разработчикам быстро создавать и отлаживать программы.
2. Богатая библиотека сторонних модулей: Python имеет богатую экосистему библиотек и модулей, включая библиотеку NumPy для работы с матрицами и числовыми данными. Это упрощает выполнение математических операций.
3. Поддержка многих платформ: Python является переносимым и поддерживается на различных операционных системах, что позволяет разрабатывать приложения, работающие на Windows и Linux.
4. Большое сообщество и документация: Python имеет большое сообщество разработчиков, что означает наличие обширной документации, руководств и форумов для получения помощи и поддержки.
5. Быстрый прототипирование: В Python легко создавать прототипы программы благодаря выразительности языка и наличию множества инструментов для разработки.
6. Графический интерфейс с Tkinter: Tkinter — это стандартная библиотека Python для создания графических интерфейсов, что делает разработку GUI-приложений относительно простой и позволяет быстро создавать пользовательские интерфейсы.
7. Открытый исходный код: Python — это язык с открытым исходным кодом, что означает, что вы можете свободно использовать, изменять и распространять свой код без ограничений.
8. Быстрое выполнение кода: Несмотря на интерпретируемую природу, Python имеет оптимизированные библиотеки, такие как NumPy, которые обеспечивают высокую производительность для вычислений.
9. Большой набор инструментов: Python предоставляет множество инструментов для тестирования, управления зависимостями, развертывания и других аспектов разработки.
10. Интеграция с другими языками и системами: Python легко интегрируется с другими языками программирования и системами, что позволяет использовать его как часть многоязычных решений.

Все эти факторы делают Python привлекательным языком для разработки приложений, включая программы с графическим интерфейсом, как в вашем случае.

2 Структура программы

На рисунке 2.1 представлена структура программы



Рисунок 2.1 – Структура программы

Подпрограммы, используемые в основной программе, приведены в таблице 2.1.

Таблица 2.1

|  |  |
| --- | --- |
| Подпрограмма | Назначение |
| gauss\_elimination | Выполняет метод Гаусса для вычисления обратной матрицы на основе входной матрицы. |
| is\_valid\_number | Проверяет валидность вводимых чисел в полях ввода матрицы, включая ограничения на нули и другие символы. |
| create\_matrix\_window | Создает окно для ввода матрицы и настраивает виджеты для ввода данных. |
| hide\_matrix\_window | Скрывает окно для ввода матрицы. |
| calculate\_inverse | Извлекает данные из полей ввода матрицы, выполняет проверку и вызывает метод gauss\_elimination для вычисления обратной матрицы. |
| show\_inverse\_matrix | Отображает результат вычисления обратной матрицы в новом окне. |
| MatrixCalculator (класс) | Основной класс, управляющий GUI и взаимодействием с пользователем. Создает главное окно и настраивает его виджеты. |

Переменные, используемые в основной программе, приведены в таблице 2.2

Таблица 2.2

|  |  |
| --- | --- |
| Переменная | Назначение |
| root | Основное окно приложения (главное окно tkinter). |
| main\_frame | Фрейм внутри главного окна для размещения виджетов. |
| dimension\_var | Переменная для хранения выбранной размерности матрицы. |
| dimension\_combobox | Поле выбора размерности матрицы (Combobox). |
| create\_matrix\_button | Кнопка для создания окна ввода матрицы. |
| matrix\_window | Окно для ввода матрицы. |
| gauss\_elimination | Функция для вычисления обратной матрицы методом Гаусса. |
| is\_valid\_number | Функция для проверки валидности введенных чисел. |
| MatrixCalculator | Основной класс, управляющий GUI и взаимодействием с пользователем. |
| calculate\_inverse | Функция для извлечения данных из полей ввода матрицы и вызова метода gauss\_elimination. |
| show\_inverse\_matrix | Функция для отображения результатов вычисления обратной матрицы. |

3 Схемы алгоритма программы



4 Отладка программы

В процессе разработки программы были обнаружены синтаксические ошибки, ошибки компоновки и ошибки выполнения.

4.1 Синтаксические ошибки, обнаруженные в программе.

При компилировании программы компилятор указывал на ошибки, допущенные при кодировании, а также на их место возникновения, что значительно сократило время на устранения выявленных проблем.

Ошибка в импорте библиотеки NumPy:

import numPy as np

Ошибка: Название библиотеки numpy написано с большой буквы, и несоответствие регистра вызовет ошибку.

Ошибка в передаче аргумента в метод self.create\_matrix\_window:

self.create\_matrix\_window(matrix\_entries, dimension)

Ошибка: Метод create\_matrix\_window не ожидает аргументов, но здесь передаются matrix\_entries и dimension.

4.2 Ошибки компоновки, обнаруженные в программе

Ошибка компановки для кнопки calculate\_button:

calculate\_button.grid(row=dimension, columnspan=dimension, padx=5, pady=10, sticky="w")

Ошибка: Используется sticky="w", но columnspan задает ширину кнопки по всей строке.

5 Оптимизация программы

Оптимизация программного кода является важной частью разработки программ и приложений. Её целью является улучшение производительности, эффективности использования ресурсов и уменьшение времени выполнения программы. Причины, по которым необходимо оптимизировать свои программы, включают в себя следующее:

1. Увеличение производительности: Оптимизация позволяет ускорить выполнение программы, что особенно важно для приложений с высокими требованиями к скорости.
2. Экономия ресурсов: Оптимизация может сократить использование оперативной памяти и процессорных ресурсов, что актуально для мобильных устройств и серверных приложений.
3. Уменьшение нагрузки на сеть и хранилища: Оптимизация может сократить объём передаваемых данных по сети и уменьшить нагрузку на базы данных и хранилища.
4. Увеличение отзывчивости интерфейса: Для приложений с графическим интерфейсом оптимизация может сделать интерфейс более отзывчивым и приятным для пользователя.
5. Эффективное использование энергии: Для мобильных устройств и ноутбуков оптимизация может увеличить время автономной работы за счёт уменьшения нагрузки на процессор.

В нашем проекте были приняты следующие методы оптимизации:

1. Разделение на функции и классы: Код был разделен на функции и классы, что улучшает читаемость, обслуживаемость и переиспользуемость кода.
2. Использование библиотек: Вы использовали библиотеку NumPy для выполнения математических операций, что улучшает производительность при работе с матрицами.
3. Проверки ввода: Вы добавили проверки ввода данных пользователем, что помогает предотвратить ошибки и улучшить интерактивность приложения.
4. Локализация текстов: Тексты в приложении были локализованы, что облегчает адаптацию программы для разных языков.
5. Использование объектно-ориентированного подхода: Вы создали класс `MatrixCalculator`, что помогло упорядочить код и логику программы.
6. Управление окнами: Программа управляет окнами таким образом, чтобы не создавать более одного окна для ввода матрицы и поднимать его на передний план, если оно уже существует.
7. Оптимизация проверки ввода: Добавлена оптимизация в проверке ввода, чтобы не позволять вводить более одного нуля перед запятой.

Оптимизация куска кода, связанного с проверкой ввода данных, позволяет предотвратить ввод некорректных данных и улучшить пользовательский опыт. Она также снижает вероятность возникновения ошибок в программе. Плюсы оптимизации этого куска кода:

1. Повышение точности: Предотвращение ввода некорректных данных гарантирует более точные результаты.
2. Улучшенная интерактивность: Пользователи получают более информативные сообщения об ошибках и ограничениях, что помогает им лучше понимать, что требуется для корректного ввода.
3. Более надежное приложение: За счет оптимизации кода, связанного с проверкой ввода, вероятность возникновения ошибок и сбоев в работе программы снижается.
4. Повышенная удовлетворенность пользователей: Четкие ограничения и сообщения об ошибках помогают пользователям легче использовать программу и избегать нежелательных ситуаций.

Программа, разрабатываемая в рамках курсового проекта, была оптимизирована как по памяти, так и по времени. Для оптимизации по памяти программа была разделена на модули и организованна таким образом, чтобы исключалось дублирование исходных данных и структурных типов. Соответствующие фрагменты программы приведены ниже:

На рисунке 5.1 представлен фрагмент кода до оптимизации (слева) и после оптимизации (справа). Таким образом был исключен ввод пользователем в процессе выбора размерности матрицы. Также в код была добавлена проверка при вводе значений в матрицу.

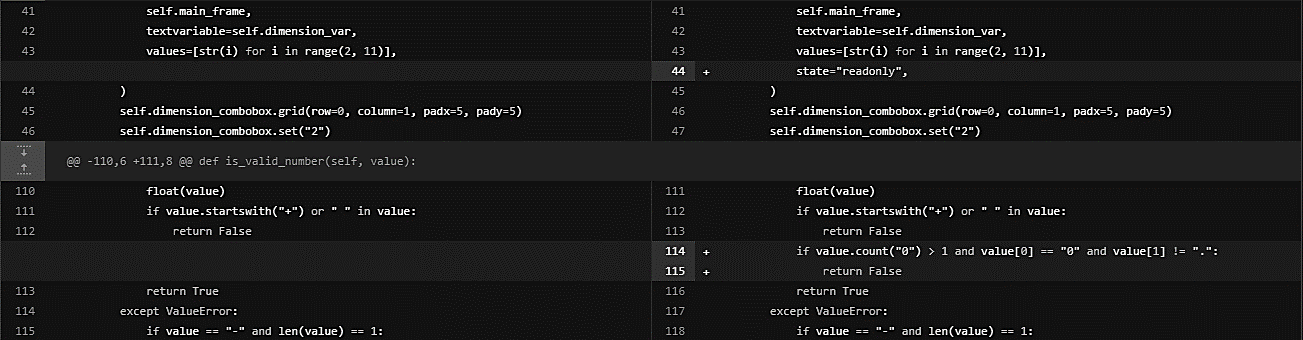


Рисунок 5.1 – Оптимизация кода

На рисунках 5.2-5.5 представлены изменения кода для достижения наиболее красивого и удобно интерфейса. За отрисовку интерфейса была взята библиотека «CustomTkinter».

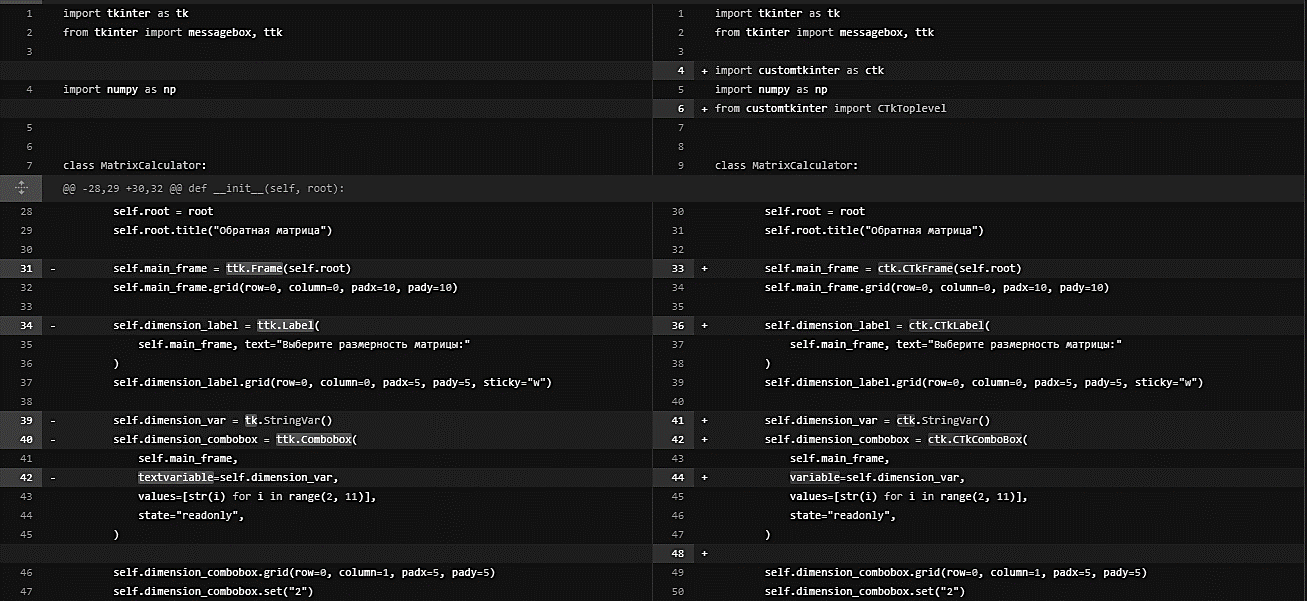


Рисунок 5.2 – Оптимизация интерфейса программы

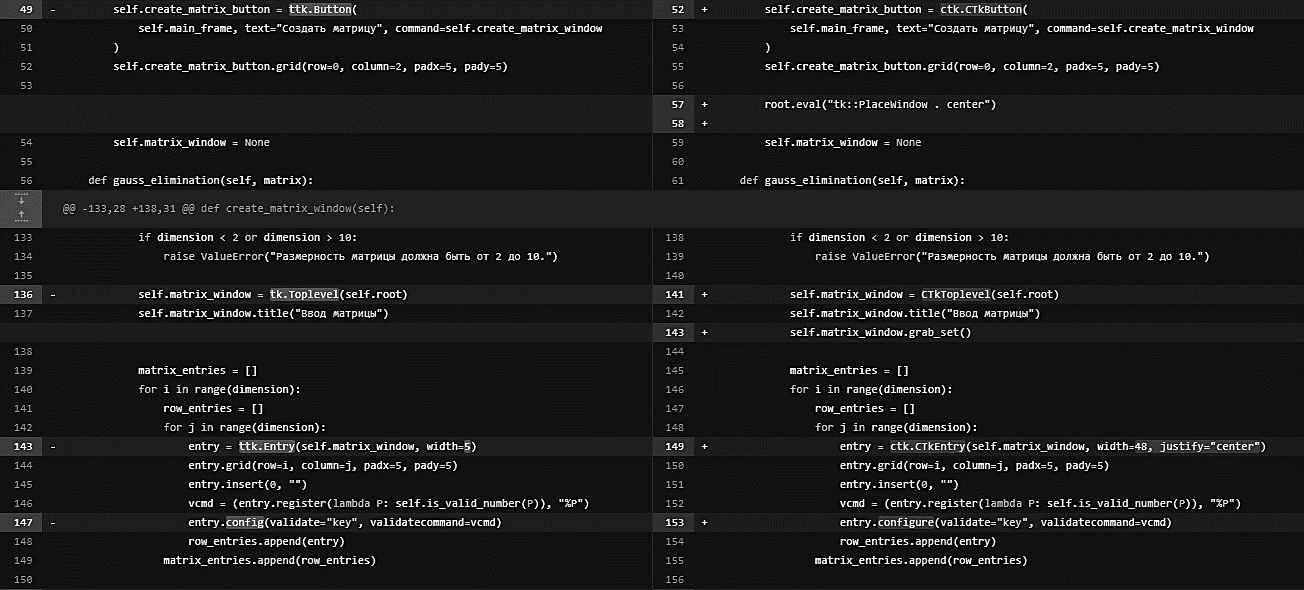


Рисунок 5.3 – Оптимизация интерфейса программы

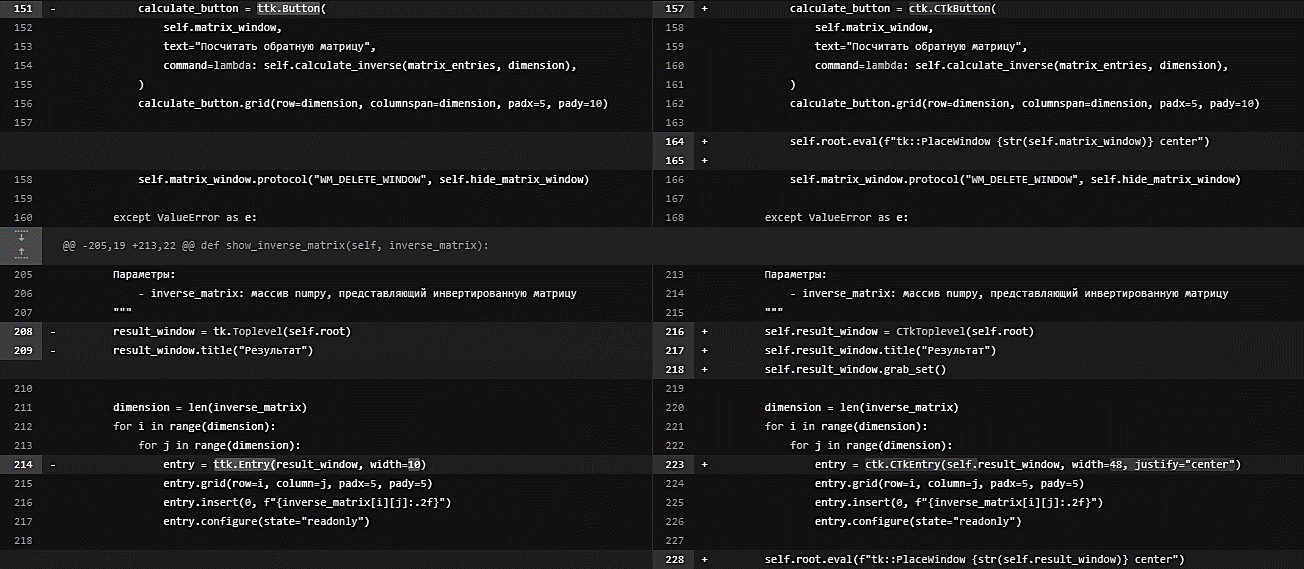


Рисунок 5.4 – Оптимизация интерфейса программы

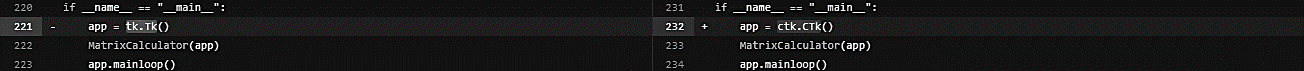


Рисунок 5.5 – Оптимизация интерфейса программы

6 Тестирование программы

Тестирование – это набор процедур и действий, предназначенных для демонстрации правильности работы программы в заданных режимах и внешних условиях. Цель тестирования – выявить наличие ошибок или убедительно продемонстрировать их отсутствие.

Процесс тестирования проходит в три этапа:

1. Проверка поведения программы в нормальных условиях.
2. Проверка поведения программы в экстремальных условиях.
3. Проверка поведения программы в исключительных ситуациях.

Каждый из этапов предполагает задание определенного, характерного для данного этапа набора входных данных.

6.1 Тестирование в нормальных условиях

Для тестирования программы в нормальных условиях достаточно сформировать стартовые значения, при которых подразумевается безотказное выполнение программы. На рисунках 6.1, 6.2, 6.3 продемонстрировано тестирование в нормальных условиях.



Рисунок 6.1 – Выбор размерности матрицы

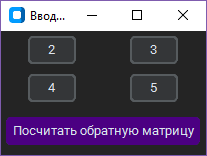


Рисунок 6.2 – Заполнение матрицы значениями

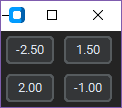


Рисунок 6.3 – Обратная матрица

**Ручной расчет** к результатам, отображенным на рисунке 6.3:

Запишем систему в виде:

Последовательно будем выбирать разрешающий элемент РЭ, который лежит на главной диагонали матрицы.

Разрешающий элемент равен 2. На месте разрешающего элемента получаем 1, а в самом столбце записываем нули. Все остальные элементы матрицы, включая элементы столбца B, определяются по правилу прямоугольника. Для этого выбираем четыре числа, которые расположены в вершинах прямоугольника и всегда включают разрешающий элемент РЭ.

НЭ = СЭ - (А∙В)/РЭ

РЭ - разрешающий элемент (2), A и В - элементы матрицы, образующие прямоугольник с элементами СТЭ и РЭ.

Представим расчет каждого элемента в виде таблицы:

|  |  |  |  |
| --- | --- | --- | --- |
| x1 | x2 | x3 | x4 |
| 2 / 2 = 1 | 3 / 2 = 1.5 | 1 / 2 = 0.5 | 0 / 2 = 0 |
|  |  |  |  |

Разрешающий элемент равен -1. На месте разрешающего элемента получаем 1, а в самом столбце записываем нули. Все остальные элементы матрицы, включая элементы столбца B, определяются по правилу прямоугольника. Для этого выбираем четыре числа, которые расположены в вершинах прямоугольника и всегда включают разрешающий элемент РЭ.

Представим расчет каждого элемента в виде таблицы:

|  |  |  |  |
| --- | --- | --- | --- |
| x1 | x2 | x3 | x4 |
|  |  |  |  |
| 0 / -1 = 0 | -1 / -1 = 1 | -2 / -1 = 2 | 1 / -1 = -1 |

Обратная матрица A-1:

6.2 Тестирование в экстремальных условиях

При тестировании в экстремальных условиях совокупность исходных данных – это данные, когда у неё есть хотя бы одно нулевое собственное значение.

Это вытекает из уравнения, которому удовлетворяют все собственные значения матрицы: (где E — единичная матрица), а также из того факта, что определитель матрицы равен произведению её собственных значений. На рисунках 6.4, 6.5 и 6.6 иллюстрируется тестирование программы в экстремальных условиях с использованием начальных значений и нулевых примеров.



Рисунок 6.4 – Выбор размерности матрицы

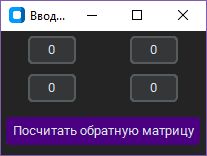


Рисунок 6.5 – Ввод экстремальных значений

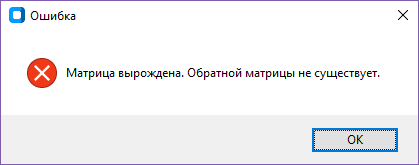


Рисунок 6.6 – Вывод сообщения об ошибке.

По определению, обратная матрица существует только для квадратных матриц, у которых определитель не равен нулю. Если вся матрица состоит из нулей, то её определитель равен нулю, и обратной матрицы не существует.

Обратная матрица — это такая матрица, которая, умноженная на исходную матрицу, дает единичную матрицу. Если исходная матрица состоит из нулей, то умножение на неё ничего не изменит, и вы не получите единичную матрицу.

Иными словами, для обратной матрицы необходимо, чтобы исходная матрица была невырожденной, что означает, что её определитель не равен нулю. Если матрица состоит только из нулей, она всегда будет вырожденной и не будет иметь обратной матрицы.

6.3 Тестирование в исключительных ситуациях

Тестирование в исключительных ситуациях предполагает формирование набора данных, значения которых лежат за пределами допустимой области изменения. Программа, тестируемая в исключительных ситуациях, должна отвергать любые значения, которые она не в состоянии обрабатывать правильно.

Рисунки 6.7, 6.8, 6.9 иллюстрируют поведение программы при вводе данных, выходящих за пределы области изменения, а также при вводе некорректных данных. Видно, что программа отбрасывает бессмысленные данные, и, либо прекращает свою работу, либо предлагает ввести данные повторно.



Рисунок 6.7 – Выбор размерности матрицы

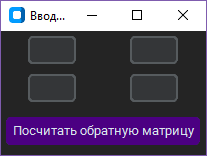


Рисунок 6.8 – Попытка оставить поля пустыми

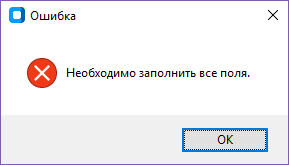


Рисунок 6.9 – Вывод сообщения об ошибке

**Вывод:** просмотр результатов тестирования в исключительных ситуациях показал, что программа устойчива к бессмысленным и неправильным данным. Она отвергает такие данные и корректно продолжает свою работу.

**Вывод по тестированию:** программа успешно прошла тестирование в нормальных и экстремальных условиях, а также в исключительных ситуациях, следовательно, она работает правильно.

7 Руководство пользователя

Данное руководство поможет вам начать использовать программу "Matrix Calculator" для вычисления обратной матрицы методом Гаусса.

Шаг 1: Запуск программы

1. После установки найдите ярлык "Matrix Calculator" на рабочем столе и запустите программу.
2. После запуска, вы увидите главное окно программы.

Шаг 2: Вычисление обратной матрицы

1. В главном окне программы выберите размерность матрицы. Используйте выпадающий список, чтобы выбрать число от 2 до 10. Это определит размерность матрицы, с которой вы будете работать.
2. Нажмите кнопку "Создать матрицу". Это откроет окно для ввода значений матрицы.
3. В окне ввода матрицы введите числовые значения в ячейки. Вы можете вводить как целые, так и десятичные числа. Обратите внимание на следующие правила:

* Запятая (`,`) используется для разделения дробной части числа.
* Ноль (`0`) может вводиться только один раз перед запятой.
* Отрицательные числа вводятся с использованием минуса (`-`) перед числом.

4. После ввода всех значений нажмите кнопку "Посчитать обратную матрицу". Программа выполнит вычисления и отобразит обратную матрицу в новом окне.

Шаг 3: Просмотр результата

1. В новом окне, отображающем результат, вы увидите вычисленную обратную матрицу.

2. Значения в этой матрице представлены с округлением до двух знаков после запятой.

3. Результаты представлены в виде только для чтения. Вы не можете редактировать значения в этой матрице.

Шаг 4: Завершение работы с программой

1. После завершения работы с программой, закройте окна, нажав на кнопку закрытия (крестик) в верхнем правом углу окон.

Примечание:

- Если вы допустили ошибку при вводе, программа предоставит информативные сообщения об ошибках, и вы сможете исправить их.

- Если матрица вырождена (нет обратной матрицы), программа также выдаст соответствующее сообщение.

ЗАКЛЮЧЕНИЕ

Целью данного курсового проекта являлось написание программы по теме: «Разработка программы обращения квадратной матрицы с помощью схемы Гаусса», которая была достигнута в ходе выполненной работы.

В ходе написания курсового проекта были получены практические навыки в области проектирования программы. Программа разрабатывалась с использованием стратегии анализа сообщений, реализующей метод нисходящей разработки. Анализ основывался на изучении потоков данных, обрабатываемых программой. Опыт, полученный при написании работы, подтвердил необходимость в первую очередь правильной выработки требований к программе и грамотного подхода к её проектированию перед началом написания кода.

Так же были получены практические навыки при оптимизации и тестировании программы. Появилось понимание важности и сложности отладки, её затратности по времени относительно всего процесса разработки, необходимости знания аспектов из разных областей IT-технологий, как, например, знание аппаратной части, операционных систем, самих реализуемых процессов, природы и специфики различных ошибок и т.д.

Результатом проведенной работы является правильно спроектированное приложение, а также работающий, эффективный, отлаженный и корректно отвечающий тестам соответствующих условий тестирования проект.

Программа имеет надлежащую структуру, схемы алгоритма и необходимые комментарии.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Официальная документация Python и библиотеки Tkinter: https://docs.python.org/3/library/tkinter.html
2. Официальная документация библиотеки NumPy: https://numpy.org/doc/stable/
3. Учебники и курсы по Python и Tkinter, такие как "Python GUI Programming with Tkinter" на Real Python: https://realpython.com/tkinter-python-gui-tutorial/
4. Учебники и курсы по библиотеке NumPy, такие как "NumPy Quickstart Tutorial" на официальном сайте NumPy: https://numpy.org/doc/stable/user/quickstart.html
5. Руководства по разработке программ с графическим интерфейсом на Python и Tkinter на GitHub и Stack Overflow.
6. Письменный Д. Т. Конспект лекций по высшей математике. «Айрис-Пресс», 2017.
7. Киркинский А.С. Математический анализ. «Академический Проект», 2006.
8. Книги о разработке программ с графическим интерфейсом на Python, включая "Python GUI Development with Tkinter" (Alan D. Moore).
9. Примеры и учебные проекты на GitHub и других репозиториях для разработки программ на Python и Tkinter.
10. Форумы и сообщества разработчиков Python и Tkinter, такие как Stack Overflow и Reddit.
11. Официальная документация и ресурсы по использованию библиотеки customtkinter.
12. Документация и руководства по использованию библиотеки NumPy для операций с матрицами и массивами.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы

import tkinter as tk

from tkinter import messagebox, ttk

import customtkinter as ctk

import numpy as np

from customtkinter import CTkToplevel

class MatrixCalculator:

    """

    Класс для вычисления обратной матрицы методом исключения Гаусса.

    Атрибуты:

        - root: корневой объект ctk

        - main\_frame: основной фрейм графического интерфейса

        - dimension\_label: метка для выбора размерности матрицы

        - dimension\_var: строковый переменный для хранения выбранной размерности

        - dimension\_combobox: Комбобокс для выбора размерности матрицы

        - create\_matrix\_button: кнопка для создания окна ввода матрицы

        - matrix\_window: Двухуровневое окно для ввода матрицы

    """

    def \_\_init\_\_(self, root):

        """

        Инициализирует объект MatrixCalculator.

        Параметры:

            - root: корневой объект ctk

        """

        self.root = root

        root.title("Калькулятор обратной матрицы")

        self.main\_frame = ctk.CTkFrame(root)

        self.main\_frame.grid(row=0, column=0, padx=10, pady=10)

        self.dimension\_label = ctk.CTkLabel(

            self.main\_frame, text="Выберите размерность матрицы:"

        )

        self.dimension\_label.grid(row=0, column=0, padx=5, pady=5, sticky="w")

        self.dimension\_var = ctk.StringVar()

        self.dimension\_combobox = ctk.CTkComboBox(

            self.main\_frame,

            variable=self.dimension\_var,

            values=[str(i) for i in range(2, 11)],

            state="readonly",

        )

        self.dimension\_combobox.grid(row=0, column=1, padx=5, pady=5)

        self.dimension\_combobox.set("2")

        self.create\_matrix\_button = ctk.CTkButton(

            self.main\_frame,

            text="Создать матрицу",

            command=self.create\_matrix\_window,

            fg\_color="#4B0082",

            hover\_color="#5f02a6",

        )

        self.create\_matrix\_button.grid(row=0, column=2, padx=5, pady=5)

        root.eval("tk::PlaceWindow . center")

        self.matrix\_window = None

    def gauss\_elimination(self, matrix):

        """

        Вычисляет обратную матрицу, используя гауссово исключение.

        Параметры:

            - matrix: массив numpy, представляющий матрицу для инвертирования

        Возвращает:

            - inverse\_matrix: массив numpy, представляющий инвертированную матрицу, или None, если матрица сингулярна.

        """

        n = len(matrix)

        augmented\_matrix = np.hstack((matrix, np.identity(n)))

        for col in range(n):

            pivot\_row = col

            for i in range(col + 1, n):

                if abs(augmented\_matrix[i][col]) > abs(

                    augmented\_matrix[pivot\_row][col]

                ):

                    pivot\_row = i

            if abs(augmented\_matrix[pivot\_row][col]) < 1e-10:

                messagebox.showerror(

                    "Ошибка", "Матрица вырождена. Обратной матрицы не существует."

                )

                return None

            augmented\_matrix[[col, pivot\_row]] = augmented\_matrix[[pivot\_row, col]]

            for i in range(col + 1, n):

                factor = augmented\_matrix[i][col] / augmented\_matrix[col][col]

                augmented\_matrix[i, col:] -= factor \* augmented\_matrix[col, col:]

        for col in range(n - 1, -1, -1):

            augmented\_matrix[col] /= augmented\_matrix[col, col]

            for i in range(col - 1, -1, -1):

                augmented\_matrix[i] -= augmented\_matrix[i, col] \* augmented\_matrix[col]

        inverse\_matrix = augmented\_matrix[:, n:]

        return inverse\_matrix

    def is\_valid\_number(self, value):

        """

        Проверяет, является ли строка действительным числом.

        Параметры:

            - value: строка, представляющая проверяемое значение

        Возвращает:

            - True, если значение является действительным числом, False в противном случае

        """

        if value == "":

            return True

        try:

            float(value)

            if value.startswith("+") or " " in value:

                return False

            if value.count("0") > 1 and value[0] == "0" and value[1] != ".":

                return False

            return True

        except ValueError:

            if value == "-" and len(value) == 1:

                return True

            return False

    def create\_matrix\_window(self):

        """

        Создает окно Toplevel для ввода матрицы.

        """

        if self.matrix\_window is not None:

            self.matrix\_window.deiconify()

            self.matrix\_window.lift()

            return

        try:

            dimension = int(self.dimension\_var.get())

            if dimension < 2 or dimension > 10:

                raise ValueError("Размерность матрицы должна быть от 2 до 10.")

            self.matrix\_window = CTkToplevel(self.root)

            self.matrix\_window.title("Ввод матрицы")

            self.matrix\_window.grab\_set()

            matrix\_entries = []

            for i in range(dimension):

                row\_entries = []

                for j in range(dimension):

                    entry = ctk.CTkEntry(self.matrix\_window, width=48, justify="center")

                    entry.grid(row=i, column=j, padx=5, pady=5)

                    entry.insert(0, "")

                    vcmd = (entry.register(lambda P: self.is\_valid\_number(P)), "%P")

                    entry.configure(validate="key", validatecommand=vcmd)

                    row\_entries.append(entry)

                matrix\_entries.append(row\_entries)

            calculate\_button = ctk.CTkButton(

                self.matrix\_window,

                text="Посчитать обратную матрицу",

                command=lambda: self.calculate\_inverse(matrix\_entries, dimension),

                fg\_color="#4B0082",

                hover\_color="#5f02a6",

            )

            calculate\_button.grid(row=dimension, columnspan=dimension, padx=5, pady=10)

            self.root.eval(f"tk::PlaceWindow {str(self.matrix\_window)} center")

            self.matrix\_window.protocol("WM\_DELETE\_WINDOW", self.hide\_matrix\_window)

        except ValueError as e:

            messagebox.showerror("Ошибка", str(e))

    def hide\_matrix\_window(self):

        """

        Скрывает окно ввода матрицы.

        """

        if self.matrix\_window is not None:

            self.matrix\_window.withdraw()

            self.matrix\_window = None

            self.main\_frame.grab\_set()

    def calculate\_inverse(self, matrix\_entries, dimension):

        """

        Вычисляет обратную величину матрицы, введенной в окно ввода матрицы.

        Параметры:

            - matrix\_entries: список списков объектов ttk.Entry, представляющих матрицу

            - dimension: целое число, представляющее размерность матрицы

        """

        try:

            matrix = []

            for i in range(dimension):

                row = []

                for j in range(dimension):

                    entry\_value = matrix\_entries[i][j].get()

                    if entry\_value == "":

                        messagebox.showerror("Ошибка", "Необходимо заполнить все поля.")

                        return

                    row.append(float(entry\_value))

                matrix.append(row)

            inverse\_matrix = self.gauss\_elimination(np.array(matrix))

            if inverse\_matrix is not None:

                self.show\_inverse\_matrix(inverse\_matrix)

        except ValueError as e:

            messagebox.showerror("Ошибка", str(e))

        except Exception as e:

            messagebox.showerror("Ошибка", f"Произошла ошибка: {e}")

    def show\_inverse\_matrix(self, inverse\_matrix):

        """

        Отображает инвертированную матрицу в новом окне Toplevel.

        Параметры:

            - inverse\_matrix: массив numpy, представляющий инвертированную матрицу

        """

        self.result\_window = CTkToplevel(self.root)

        self.result\_window.title("Результат")

        self.result\_window.grab\_set()

        dimension = len(inverse\_matrix)

        for i in range(dimension):

            for j in range(dimension):

                entry = ctk.CTkEntry(self.result\_window, width=48, justify="center")

                entry.grid(row=i, column=j, padx=5, pady=5)

                entry.insert(0, f"{inverse\_matrix[i][j]:.2f}")

                entry.configure(state="readonly")

        self.root.eval(f"tk::PlaceWindow {str(self.result\_window)} center")

if \_\_name\_\_ == "\_\_main\_\_":

    app = ctk.CTk()

    MatrixCalculator(app)

    app.mainloop()

ПРИЛОЖЕНИЕ Б

(обязательное)

Результаты выполнения программы

На рисунках Б.1, Б.2, Б.3, Б.4, Б.5, Б.6 продемонстрированы результаты нахождения обратной матрицы методом Жордано-Гаусса.

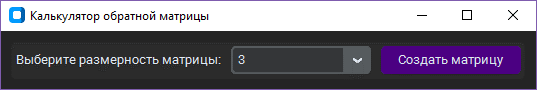


Рисунок Б.1 – Выбор размерности матрицы.

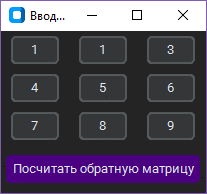


Рисунок Б.2 – Задаём параметры матрицы 3x3

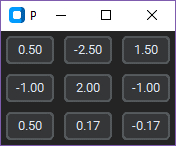


Рисунок Б.3 – Результат нахождения обратной матрицы