Проект:

Калькулятор матриц с интерфейсом

Студент:  
 \_\_\_\_\_\_\_\_\_\_\_\_Марченко Л. А.

Студент:

\_\_\_\_\_\_\_\_\_\_\_\_Донник И. П.

Содержание

1. Обзорная………………………………..……………………………3

1.1 Библиотеки…………………………………………………………3

1.2. Цели и задачи проекта…………………………………………….4

1.3. Актуальность………………………………………………………4

1.4. Анализ продуктов………………………………………………….5

1.5.Что было изучено………………………………………………….6

2. Реализация…………………………………………………………....6

2.1. Как работает код…………………………………………..………6

2.2. windows.py…………………………………………………………7

2.3. functions.py…………………………………………………………7

2.4. matrix\_calc.py………………………………………………………………..12

Вывод……………………………………………………………..………………14

Список используемой литературы……………………………………………...15

**1. Обзорная**

**1.1. Библиотеки**

В проекте (калькулятор матриц) мы используем библиотеки numpy для работы с матрицами, и библиотеку PySimpleGUI для создания кнопок и полей ввода значений.

Numpy – достаточно быстро работает с многомерными массивами. Вообще, интерпретируемые языки производят вычисления медленнее компилируемых, а Python как раз язык интерпретируемый. NumPy же сделана так, чтобы эффективно работать с наборами чисел любого размера в Python.

Массивы в NumPy отличаются от обычных списков и кортежей в Python тем, что они должны состоять только из элементов одного типа. Такое ограничение позволяет увеличить скорость вычислений в 50 раз, а также избежать ненужных ошибок.

PySimpleGUI это пакет Python, который позволяет программистам Python всех уровней создавать графические интерфейсы. Указываем тип окна GUI, используя «макет», который содержит виджеты (они называются «Элементами» в PySimpleGUI). Макет используется для создания окна с использованием одной из 4 поддерживаемых платформ. Поддерживаемые платформы: tkinter, Qt, WxPython и Remi.

Код на PySimpleGUI будет проще и короче, чем написанный непосредственно с использованием базовой платформы, потому что PySimpleGUI реализует большую часть «шаблонного кода» за нас.

**1.2. Цели и задачи проекта**

Калькулятор матриц поможет упростить выполнение сложных математических операций, таких как умножение, сложение, вычитание матриц.

Использование калькулятора матриц может помочь студентам или учащимся лучше разобраться в операциях с матрицами, а также научиться использовать их в решении задач.

В научных, инженерных или экономических областях калькулятор матриц может быть полезен для проведения анализа данных, решения систем линейных уравнений, других профессиональных задач.

Создание калькулятора матриц может быть частью разработки математических алгоритмов и методов для решения сложных задач в области науки, техники и технологий.

Использование калькулятора матриц может помочь улучшить процессы обработки данных, оптимизировать вычисления и улучшить эффективность работы с матрицами в общем.

**1.3. Актуальность**

Калькулятор матриц является актуальным инструментом для решения различных задач в математике, физике, инженерии, компьютерных науках, экономике и других областях. Он позволяет умножать, складывать, вычитать матрицы, находить определитель, обратную матрицу и многое другое.

Калькулятор матриц актуален для студентов, ученых, инженеров и специалистов, работающих с линейной алгеброй и матричными операциями. Он облегчает выполнение вычислений, ускоряет процесс решения задач и помогает избежать ошибок при манипуляциях с матрицами.

Благодаря развитию вычислительных технологий, онлайн-калькуляторы матриц становятся все более популярными и доступными, что делает их актуальными для использования на практике.

**1.4. Анализ продуктов**

Анализ продуктов калькуляторов матриц может включать в себя следующие характеристики:

1. Функциональность: качественный калькулятор матриц должен предлагать широкий спектр операций с матрицами, включая умножение, сложение, вычитание, нахождение определителя и обратной матрицы. Также желательно наличие дополнительных функций как нахождение собственных значений и собственных векторов.

2. Интерфейс: удобный и интуитивно понятный интерфейс калькулятора матриц позволит пользователям легко взаимодействовать с программой. Это может включать в себя удобное расположение кнопок, легкость ввода матриц и наглядное отображение результатов.

3. Точность: продукт должен обеспечивать высокую точность вычислений, особенно при выполнении сложных математических операций. Это особенно важно при работе с большими матрицами или матрицами с высокой точностью.

4. Доступность: калькулятор матриц должен быть доступен для использования на различных устройствах и операционных системах, включая веб-приложения, мобильные приложения и настольные программы.

5. Дополнительные возможности: некоторые продукты могут предлагать дополнительные функции, такие как графическое представление матриц, возможность сохранения и загрузки матриц, а также интеграцию с другими математическими инструментами.

При выборе калькулятора матриц важно учитывать вышеуказанные характеристики, чтобы обеспечить удовлетворение потребностей пользователя в удобном, функциональном и точном инструменте для работы с матрицами.

**1.5. Что было изучено**

В ходе создания проекта было необходимо изучить следующее:

- создание графических пользовательских интерфейсов (GUI) с помощью PySimpleGUI.

- интерактивное взаимодействие с элементами управления, такими как кнопки, поля ввода.

- обработка пользовательского ввода данных.

- как находить ранг, определитель, сложение вычитание и т.д. с использование numpy

**2. Реализация**

**2.1. Как работает код**

Чтобы код заработал (я использую Visual Studio), нужно установить numpy и PySimpleGUI следующими командами pip install numpy и pip install PySimpleGUI, далее положить все файлы в одну папку и запустить файл matrix\_calc.py чтобы код начал работать.

**2.2. windows.py**

matrix\_a\_layout - это список списков, представляющих собой поля для ввода значений матриц. Каждый список представляет собой одну строку.

Для каждого элемента матрицы создается текстовое поле, в котором пользователь может вводить значение

matrix\_a\_layout = [[sg.Text('Матрица A:')],

       [sg.InputText(key='-1.1-', size=2, default\_text='0', justification='centre'),

создает текстовое поле для ввода элемента матрицы в первой строке и первом столбце, с начальным значением '0' и выравниванием текста по центру.

matrix\_operations\_layout тут создается список списков в которых лежат кнопки с действиями ( +, -, \*) и ключами кнопок

matrix\_operations\_layout = [

        [sg.Button('\*', key='-\*-', size=1)],

        [sg.Button('+', key='-+-', size=1)],

operations тут создаются кнопки с ключами и размером самих кнопок

operations = [

        [sg.Button('Найти определитель матрицы A', key='-DET-', size=(30, 1)),

         sg.Button('Найти обратную матрицы A', key='-INV-', size=(30, 1))],

**2.3. functions.py**  
- функция det рассчитывает определитель матрицы.

def det(matrix):                        # расчет определителя матрицы

    matrix = triangle\_matrix(matrix)

    det\_matrix = 1

    for diagonal in range(len(matrix)):

        det\_matrix \*= matrix[diagonal, diagonal]

    det\_matrix \*= (-1) \*\* count

    print(f'Определитель матрицы A равен: {det\_matrix:g}')

    return det\_matrix

Функция det(matrix) принимает на вход матрицу и вычисляет её определитель. Вначале матрица приводится к треугольному виду с помощью функции triangle\_matrix. Затем определитель вычисляется путем перемножения элементов на главной диагонали полученной треугольной матрицы.

Переменная det\_matrix инициализируется единицей, а затем в цикле для каждой диагональной ячейки матрицы умножается на текущее значение det\_matrix. После этого значение det\_matrix корректируется, умножая на (-1) в степени count. Результат выводится на экран с использованием функции print и возвращается как результат выполнения функции.

- функция triangle\_matrix

def triangle\_matrix(matrix):       # треугольник

    global count

    col = 0

    count = 0

    matrix = matrix.copy()

    for i in range(len(matrix) - 1):

        for row in range(col + 1, len(matrix)):

            if matrix[col, col] == 0:

                matrix[:, [col, -1]] = matrix[:, [-1, col]]

                count += 1

            matrix[row] = matrix[row] - matrix[col] \* (matrix[row, col] / matrix[col, col])

        col += 1

    return matrix

Функция triangle\_matrix принимает на вход матрицу matrix и преобразует ее в треугольную матрицу путем выполнения элементарных преобразований строк. Эта функция использует метод Гаусса для приведения матрицы к треугольному виду.

Внутри функции устанавливается глобальная переменная count, которая инициализируется значением 0. Затем переменная col устанавливается в 0, а копия входной матрицы сохраняется в переменной matrix.

Затем происходит проход по порядку range(len(matrix) - 1), чтобы пройти по каждой строке матрицы, и далее по каждой из следующих строк. Внутренний цикл использует переменную row для движения по строкам матрицы начиная с col + 1. Внутри этого цикла выполняются вычисления для приведения матрицы к треугольному виду.

Если элемент в позиции matrix[col, col] равен 0, то происходит перестановка столбцов в матрице. После этого пересчитывается значение переменной count. Далее выполняется элементарное преобразование строк, в результате чего значения в некоторых строках матрицы вычитаются из других строк. После завершения внутреннего цикла увеличивается значение переменной col, и эти операции повторяются до тех пор, пока не будет достигнута верхняя треугольная форма.

В конце функция возвращает полученную треугольную матрицу.

- функция inverted\_matrix рассчитывает обратную матрицу

def inverted\_matrix(matrix):        # обратная матрицу

    matrix = matrix.copy()

    if det(matrix) == 0:

        print('Обратная матрица не существует, так как определитель равен 0')

    else:

        col = 0

        col\_r = len(matrix) - 1

        e = np.random.randint(0, 1, size=(len(matrix), len(matrix[0])))

        for diagonal in range(len(matrix)):

            e[diagonal, diagonal] += 1

        matrix = np.append(matrix, e, axis=1)

        for i in range(len(matrix) - 1):

            for row in range(col + 1, len(matrix)):

                matrix[row] = matrix[row] - matrix[col] \* (matrix[row, col] / matrix[col, col])

            col += 1

        for i in range(len(matrix) - 1):

            for row in range(col\_r - 1, -1, -1):

                matrix[row] = matrix[row] - matrix[col\_r] \* (matrix[row, col\_r] / matrix[col\_r, col\_r])

            col\_r -= 1

        for diagonal in range(len(matrix)):

            matrix[diagonal] \*= 1 / matrix[diagonal, diagonal]

        print(f'Обратная матрице A: \n{np.round(matrix[:, len(matrix):], 1)}')

        return matrix[:, len(matrix):]

Функция inverted\_matrix(matrix) принимает матрицу matrix и возвращает ее обратную матрицу. Если определитель матрицы равен нулю, то функция выводит сообщение о том, что обратная матрица не существует.

Для вычисления обратной матрицы используется метод Гаусса-Жордана. Сначала к матрице добавляется единичная матрица того же размера, чтобы получить расширенную матрицу. Затем происходит прямой ход метода Гаусса, при котором из каждой строки вычитается предыдущая, умноженная на коэффициент, чтобы получить треугольную матрицу. Затем происходит обратный ход метода Гаусса, при котором из каждой строки вычитается следующая, умноженная на коэффициент. После этого на главной диагонали матрицы остаются единицы, а в остальных ячейках - обратные элементы.

В конце функция выводит обратную матрицу и возвращает ее в виде массива.

- функция transpose транспонирует матрицу

def transpose(matrix):                  # транспонирует

    matrix = matrix.copy()

    matrix = np.array(list(map(list, zip(\*matrix))))

    print(f'Транспонированная матрица A: \n{matrix.astype(int)}')

    return matrix

Функция transpose(matrix) принимает матрицу matrix и транспонирует ее, то есть меняет местами строки и столбцы. Для этого сначала создается копия матрицы, чтобы не изменять исходную матрицу. Затем матрица преобразуется в массив numpy, чтобы можно было использовать метод транспонирования. Для транспонирования используется функция zip, которая объединяет элементы списков по позициям, а затем функция map, которая применяет функцию list к каждому элементу полученной последовательности. Результатом является список списков, который снова преобразуется в массив numpy.

После транспонирования функция выводит полученную матрицу в виде строки с помощью функции print, используя метод astype(int) для преобразования элементов массива в целочисленный тип данных. Затем функция возвращает транспонированную матрицу в виде двумерного массива.

- функция rank определяет ранг матрицы

def rank(matrix):

    matrix = matrix.copy()

    matrix = triangle\_matrix(matrix)

    rank\_count = 0

    for row in range(len(matrix)):

        if matrix[row, -1] != 0:

            rank\_count += 1

    print(f'Ранг матрицы A равен: {rank\_count}')

    return rank\_count

Эта функция rank(matrix) принимает матрицу в качестве аргумента и вычисляет её ранг.

1. В начале происходит копирование входной матрицы.

2. Затем вызывается функция triangle\_matrix(matrix), которая преобразует исходную матрицу в треугольную матрицу. С помощью функции triangle\_matrix приводим матрицу в треугольную форму.

3. После преобразований происходит подсчет количества ненулевых строк в последнем столбце матрицы, и это количество присваивается переменной rank\_count.

4. Результат выводится в виде сообщения о том, что ранг матрицы A равен определенному значению.

5. Функция возвращает вычисленное значение ранга матрицы.

- функция matrix prob умножает матрицу А на В если это возможно

def matrix\_prod(matrix\_a, matrix\_b):

    matrix\_a = matrix\_a.copy()

    matrix\_b = matrix\_b.copy()

    matrix\_c = np.random.randint(0, 1, size=(len(matrix\_a), len(matrix\_b[0]))).astype('float')

    if len(matrix\_a[0]) != len(matrix\_b):

        return 'Умножение невозможно'

    row = 0

    for i in range(len(matrix\_a)):

        for col in range(len(matrix\_b[0])):

            matrix\_c[row, col] = sum(matrix\_a[row] \* matrix\_b[:, col])

        row += 1

    print(f'Результат умножения матрицы A на B: \n{np.round(matrix\_c, 1)}')

    return matrix\_c

Функция matrix\_prod(matrix\_a, matrix\_b) принимает две матрицы (matrix\_a и matrix\_b) и выполняет их умножение.

1. В начале создаются копии входных матриц (matrix\_a и matrix\_b).

2. Затем создается новая матрица matrix\_c размером len(matrix\_a) x len(matrix\_b[0]), заполненная случайными целыми числами от 0 до 1 в виде массива с плавающей запятой.

3. Выполняется проверка на возможность умножения матриц: если количество столбцов в первой матрице не соответствует количеству строк во второй матрице, функция возвращает сообщение "Умножение невозможно" и завершает выполнение.

4. Происходит проход по элементам результирующей матрицы matrix\_c и вычисляются их значения с использованием вложенных циклов и операции суммирования.

5. Результат умножения выводится в виде сообщения и округляется до одного знака после запятой с помощью метода np.round().

6. Функция возвращает результирующую матрицу matrix\_c.

Таким образом, функция matrix\_prod выполняет умножение матриц matrix\_a на matrix\_b и возвращает результат этого умножения.

**2.4. matrix\_calc.py**

matrix\_calc.py является файлом запуска всего проекта.

запускаем функцию matrix\_enter из файла(windows)

запускается цикл while

event, matrix = window.read() отслеживает нажатия

 matrix\_a\_list = [[matrix['-1.1-'], matrix['-1.2-'], matrix['-1.3-']],

                         [matrix['-2.1-'], matrix['-2.2-'], matrix['-2.3-']],

                         [matrix['-3.1-'], matrix['-3.2-'], matrix['-3.3-']]]

Хранит данные матрицы а

matrix\_a\_created сохраняет в себя массив данных из matrix\_a\_list с плавающей точкой

Далее проверяем что нажмет пользователь и какую функцию нужно выполнить по ключу

Цикл while завершится когда пользователь закроет окно

if event == sg.WIN\_CLOSED:

            break

**Вывод**

В ходе проекта было реализован калькулятор матрицы с возможностью найти определитель, обратную, ранг, транспонированную, умноженную матрицу на какое-то число, возвести матрицу в какую-то степень.

На данный момент в проекте не реализован вывод матрицы в окно приложения, чтобы не использовать терминал

В будущем можно будет добавить возможность менять число строк и столбцов в матрице, добавлять новые матрицы и создать строку где можно будет составить свое уравнение и найти его конечный результат.

**Список используемой литературы**

1. <https://www.pysimplegui.org/en/latest/#user-interfaces-for-humans-transforms-tkinter-qt-remi-wxpython-into-portable-people-friendly-pythonic-interfaces>

2. <https://pypi.org/project/PySimpleGUI/>

3. Семинар по numpy  
4. <https://pythonworld.ru/numpy/1.html>