Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Кафедра Систем Управления и Информатики

**Лабораторная работа №1**

**Вариант №1**

Выполнил(и:)

Гусев Я.А.

Проверил

Мусаев А.А.

Санкт-Петербург,

2022

**Введение**

В этой работе я напишу несколько функций, позволяющих производить некоторые манипуляции над матрицами, а также проанализирую рациональность использования библиотеки numpy в Python для реализации этих же функций.

**Задание 1**

Цель – создать программу на языке Python, которая будет содержать следующие функции: умножение матриц, транспонирование матрицы.

Начнём с ввода произвольной матрицы. Считываем её высоту (n) и ширину(m), далее создаём 2д-список (далее – матрица) из нулей с заданными размерами и считываем в неё элементы построчно, используя цикл for (Рисунок 1).

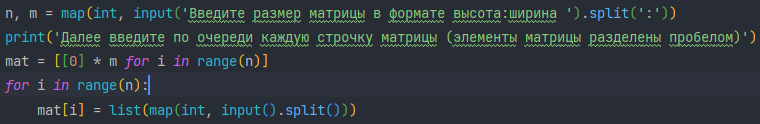


Рисунок 1 – Ввод

**Транспонирование**

Создадим функцию transp (Рисунок 2), которая будет принимать на вход нашу матрицу, а возвращать транспонированную матрицу.

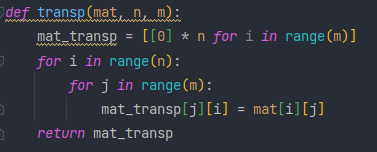


Рисунок 2 – Функция transp

В качестве аргументов наша функция принимает матрицу и её размеры. Внутри самой функции создаём новую матрицу, у которой высота равна ширине нашей введённой матрицы, а ширина соответственно равна высоте нашей матрицы. Далее с помощью вложенного цикла проходимся по нашему списку mat и присваиваем в список mat\_transp элементы с противоположными индексами (столбец становится строчкой, строчка – столбцом). Таким образом у нас получается транспонированная матрица, которую мы благополучно возвращаем из функции.

**Интерфейс**

Так как для умножения матриц нам потребуется считать у пользователя вторую матрицу, создадим простенький интерфейс для нашей программы (Рисунок 3). Для начала попросим выбрать действие: транспонирование или умножение. Если пользователь вводит «Т», инициируется выполнение вышеописанной функции транспонирования с присваиванием её результата в переменную mat\_transp и дальнейшим выводом результата в удобном для чтения виде (построчно).

В случае если пользователь вводит «У», мы считываем вторую матрицу аналогично первой, присваиваем результат функции умножения (она будет описана далее) в переменную u и выводим его построчно



Рисунок 3 – Интерфейс

**Умножение**

Создаём функцию умножения матриц (Рисунок 4), принимающую 2 матрицы в качестве аргументов.

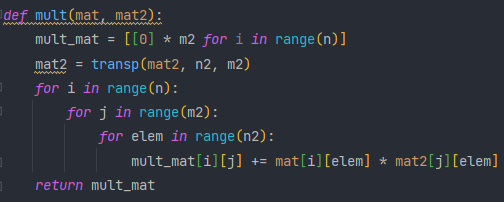


Рисунок 4 – Функция умножения

Внутри функции создаём матрицу, состоящую из нулей, шириной второй матрицы, высотой первой матрицы. Транспонируем матрицу 2 с помощью нашей первой функции. Далее проходимся по строкам первой матрицы, по столбцам второй и по элементам в них (элементов одинаковое количество, т.к. высота первой матрицы должна быть равна ширине второй, иначе их нельзя было бы умножить). К элементу mult\_mat, имеющему координаты (строка первой матрицы; столбец второй матрицы) прибавляем соответственные произведения элементов этих строки и столбца (Рисунок 5).

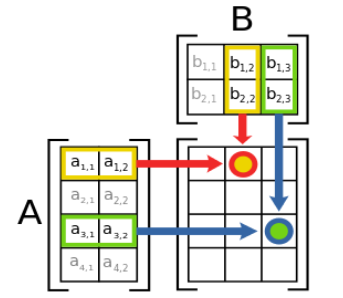


Рисунок 5 – Принцип умножения матриц

Таким образом, мы получили программу, способную транспонировать и умножать матрицы.

**Задание 2**

Цель – реализовать программу, созданную нами в задании 1, при помощи библиотеки numpy.

Импортируем библиотеку numpy (Рисунок 6).



Рисунок 6 – Импорт библиотеки numpy в Python

Создаём 2д-список с нулями и принимаем в него значения от пользователя аналогично первому заданию. Используем функцию array() библиотеки numpy, для того чтобы превратить наш список в матрицу, с которой мы будем производить дальнейшие манипуляции (Рисунок 7).

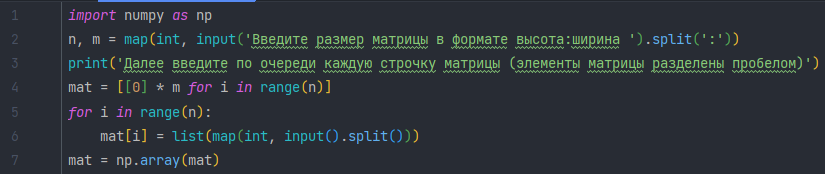


Рисунок 7 – Ввод матрицы

**Транспонирование**

Позаимствуем интерфейс из первого задания и реализуем транспонирование матрицы при помощи функции transpose() (Рисунок 8).

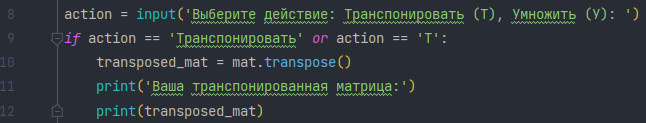


Рисунок 8 – Транспонирование матрицы с помощью numpy

Выводим результат.

У**множение**

Считываем вторую матрицу. Реализуем умножение при помощи функции matmul() (Рисунок 9).

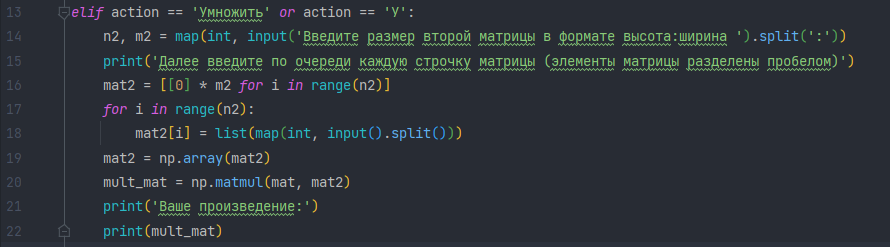


Рисунок 9 – Умножение матриц с помощью numpy

Выводим результат.

**Преимущества и недостатки numpy**

Очевидным преимуществом библиотеки numpy по сравнению с созданием функций вручную является упрощение кода в несколько десятков раз с сохранением скорости

Недостаток – numpy сильно зависит от библиотек C++.

**Задание 3**

Цель – создание функции для возведения матрицы A размерности 3x3 в степень -1 и сравнение быстродействия нашей функции с её аналогом из numpy.

Считываем матрицу 3 на 3 (Рисунок 10).

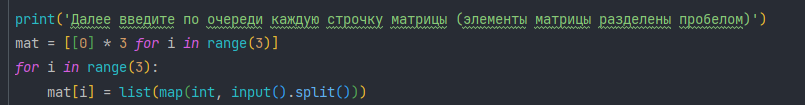


Рисунок 10 – Считывание матрицы

Создаём функцию reverse\_mat. Позаимствуем функцию транспонирования из задания 1 (Рисунок 11).

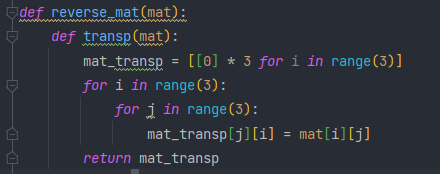


Рисунок 11 – Начало функции

Находим детерминант матрицы методом треугольников (Рисунок 12).

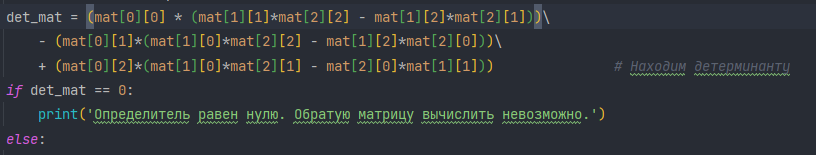


Рисунок 12 – Нахождение детерминанта

Если детерминант не равен нулю, продолжаем выполнение функции.

**Алгоритм**

Создаём пустую матрицу 3 на 3. Транспонируем нашу изначальную матрицу (Рисунок 13).

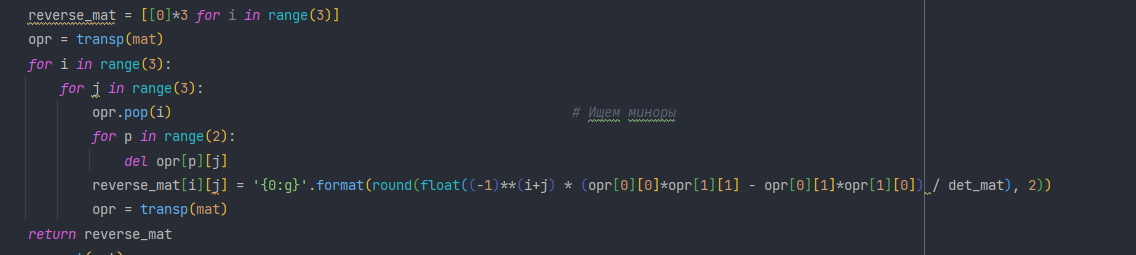


Рисунок 13 – Главный алгоритм

Далее нам требуется найти минор для каждого элемента матрицы. Для этого мы проходимся по матрице с помощью вложенного цикла, удаляем из нашей транспонированной матрицы строчку с индексом i, при помощи метода pop(). Проходимся по оставшимся строчкам и из каждой удаляем элемент с индексом j (таким образом убираем j-ный столбец). У нас получился минор opr. Определяем его детерминант и алгебраическое дополнение матрицы. Делим на детерминант нашей входной матрицы и присваиваем это значение элементу reverse\_mat[i][j]. Проделываем данный алгоритм для каждого из 9 элементов матрицы и получаем обратную матрицу opr.

**Реализация в numpy**

Та же функция с помощью numpy (Рисунок 14).

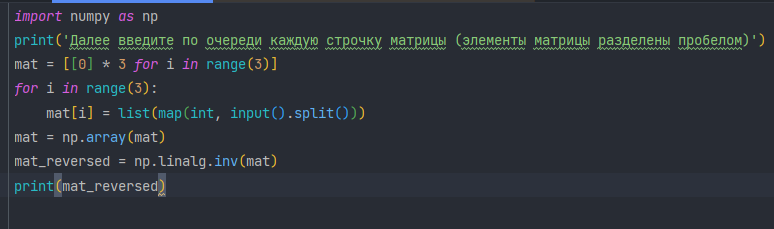


Рисунок 14 – Нахождение обратной матрицы с помощью numpy

**Сравнение быстродействия**

Импортируем библиотеки time и timeit для измерения быстродействия кода. Сначала замерим скорость функции в numpy (Рисунок 15).

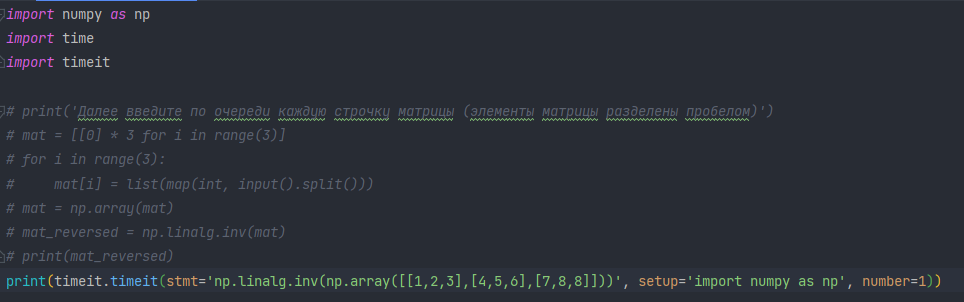


Рисунок 15 – Измерение быстродействия функции с numpy

В консоль вывелось 0.0001915. Это и есть наша скорость. Далее замерим скорость самописной функции (Рисунок 16).

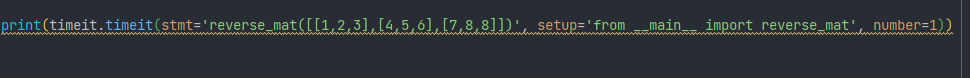


Рисунок 16 – Измерение быстродействия самописной функци

В консоль вывелось 0,00432999. Мы видим, что использование numpy повышает быстродействие функции в 40 раз.

**Вывод**

Я написал функции, позволяющие транспонировать, умножать и находить обратные матрицы. Также я понял, что библиотека numpy – это лучшее, что придумало человечество, так как она позволяет сохранить кучу времени, а также не только не замедляет код, но и ускоряет его в десятки раз.

**Список литературы.**

1. Wikipedia - <https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F> (Дата последнего обращения 28.09.2022)