Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий и систем связи

Лабораторная работа №1

Вариант №1

Выполнил(и:)

Алексеев Т.Ю.

Проверил

Мусаев А.А.

Санкт-Петербург,

2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы являлось знакомство с языком программирования Python, обучение работы с матрицами и знакомство с библиотеками NumPy и timeit.

Для достижения данной цели необходимо было выполнить следующие задания:

1. Задание 2: создание программы на языке Python, обладающей возможностями транспонирования матрицы, умножения матриц и определения ранга матриц.
2. Задание 3: выполнить Задание 2 с помощью NumPy и оценить достоинства и недостатки данной библиотеки.
3. Задание 4: написать программу для возведения матрицы в -1 степень и сравнить время её работы с аналогом из библиотеки NumPy.

Решения данных задач будут находиться на GitHub по ссылке: https://github.com/NorthPole0499/Algoritms_task_1

1. ЗАДАНИЕ 2

1.1 Описание функции транспонирования матрицы

Алгоритм транспонирования матрицы мною был реализован через создание новой матрицы. Во вложенных циклах for в список elem_of_new_matr вносились значения столбцов матрицы. После завершения внесения в него очередного столбца список добавлялся в новую матрицу как строка. Таким образом, за счёт перевода столбцов старой матрицы в строки новой выполняется транспонирование матрицы.

```
def transp(data):
    new_matrix = []
    elem_of_new_matr = []
    for i in range(len(data[0])):
        for j in range(len(data)):
            elem_of_new_matr.append(data[j][i])
        new_matrix.append(elem_of_new_matr)
        elem_of_new_matr = []
    return new_matrix
```

Приложение 1 – Содержание функции transp

1.2 Описание функции для умножения матриц

Так как операция умножения матриц бинарная, то в начале функции представлено считывание второй матрицы. Также, для более простого перебора была использована функция transp, описанная ранее. В переменную value записывается сумма произведений соответствующих элементов в i-ой строке из первой матрицы и в j-ой строке из второй матрицы. После этого переменная добавляется в elem_of_new_matrix – i-ю строку в результирующей матрице. Таким образом, функция возвращает результат умножения двух матриц.

```
def multiplication(first_matrix):
    print("Введите матрицу, на которую хотите умножить, построчно. "
          "Когда вы закончите вводить данные, нажмите дважды клавишу Enter")
    second_matrix = []
    while True:
        elem = input()
        if elem == '':
            break
        else:
            second_matrix.append(list(map(int, elem.split())))
    second_matrix = transp(second_matrix)

    new_matrix = []
    elem_of_new_matrix = []
    value = 0
    for i in range(len(first_matrix)):
        for j in range(len(second_matrix)):
            for k in range(len(first_matrix[i])):
                value += first_matrix[i][k] * second_matrix[j][k]
            elem_of_new_matrix.append(value)
            value = 0
        new_matrix.append(elem_of_new_matrix)
        elem_of_new_matrix = []
    return new_matrix
```

Приложение 2 – Содержание функции multiplication

1.3 Описание функции определения ранга матрицы

На вход данной функции подаётся матрица размером 3x3. Для определения ранга матрицы был выбран способ перебора всех миноров матрица 2 и 3 порядков. Сначала считаются все миноры 2 порядка, а потом считается единственный минор 3 порядка. Если все миноры 2 порядка равны 0, то ранг будет равен 1. Если максимальный минор 2 порядка больше минора 3 порядка, то ранг будет равен 2. В противном случае ранг матрицы равен 3.

```
def rank(matrix):
    minor_data = []
    a1, a2, a3 = matrix[0][0], matrix[1][0], matrix[2][0]
    b1, b2, b3 = matrix[0][1], matrix[1][1], matrix[2][1]
    c1, c2, c3 = matrix[0][2], matrix[1][2], matrix[2][2]
    new_a1 = c2 * b3 - b2 * c3
    new_a2 = c1 * b3 - b1 * c3
    new_a3 = c1 * b2 - b1 * c2
    new_b1 = c2 * a3 - a2 * c3
    new_b2 = c1 * a3 - a1 * c3
    new_b3 = c1 * a2 - a1 * c2
    new_c1 = b2 * a3 - a2 * b3
    new_c2 = b1 * a3 - a1 * b3
    new_c3 = b1 * a2 - a1 * b2
    main_minor = a1 * b2 * c3 + a2 * b3 * c1 + a3 * b1 * c2 - a3 * b2 * c1 - a2 * b1 * c3 - a1 * b3 * c2
    minor_data.append((new_a1, new_b1, new_c1, new_a2, new_b2, new_c2, new_a3, new_b3, new_c3))
    if max(minor_data[0]) == 0:
        max_minor = 1
    elif max(minor_data[0]) > main_minor:
        max_minor = 2
    else:
        max_minor = 3
    return max_minor
```

Приложение 3 – Содержание функции rank

2. ЗАДАНИЕ 3

2.1 Описание функции транспонирования матриц при помощи NumPy

Библиотека NumPy помогает выполнять сложные операции с матрицами буквально в одну строку. Но переменная matrix перед этим должна быть преобразована с помощью numpy.array(matrix) в массив типа numpy. А само транспонирование выполняется с помощью функции numpy.transpose(matrix).

```
def transp_numpy(matrix):
    tr_matrix = numpy.transpose(matrix)
    return tr_matrix
```

Приложение 4 – Содержание функции transp_numpy

2.2 Описание функции умножения матриц при помощи NumPy

Так как операция умножения матриц бинарная, то опять начинаем со считывания второй матрицы. Её также необходимо привести к массиву вида numpy с помощью numpy.array(second_matrix). Умножение выполняется с помощью функции numpy.dot(first_matrix, second_matrix).

```
def multiplication_numpy(first_matrix):
    second_matrix = []
    print("Введите матрицу построчно. Когда вы закончите вводить данные, "
          "нажмите дважды клавишу Enter")
    while True:
        elem = input()
        if elem == '':
            break
        else:
            second_matrix.append(list(map(int, elem.split())))
    second_matrix = numpy.array(second_matrix)
    answer = numpy.dot(first_matrix, second_matrix)
    return answer
```

Приложение 5 – Содержание функции multiplication_numpy

2.3 Описание функции определения ранга матрицы при помощи Numpy

На вход функции подаётся переменная matrix – массив типа numpy. Сама операция определения ранга матрицы выполняется с помощью функции numpy.linalg.matrix_rank(matrix).

```
def rank_numpy(matrix):
    martix_rank = numpy.linalg.matrix_rank(matrix)
    return martix_rank
```

Приложение 6 – Содержание функции rank_numpy

2.4 Достоинства и недостатки библиотеки NumPy

Достоинства:

1. Возможность выполнить сложные математические операции с матрицами буквально одной строчкой кода.
2. В библиотеке представлены максимально оптимизированные алгоритмы, прошедшие множество проверок и правок сообщества программистов.

Недостатки:

1. Нет общего понимания алгоритма и нет возможности вносить свои правки в алгоритм выполнения операции, так как он закрыта и представляет собой лишь работающую функцию.
2. NumPy работает с матрицами, которые преобразованы в массивы типа numpy. Это может создать проблемы при дальнейшей самостоятельной работе с ними.

3. ЗАДАНИЕ 4

3.1 Описание функции возведение матрицы в -1 степень

Основная функция reverse_matrix начинается с того, что нам нужно узнать определитель исходной матрицы. Он определяется с помощью функции opredelitel(matrix), подставляя значения в формулу определителя для матрицы 3x3. Если определитель равен 0, то для исходной матрицы не существует обратной, возвращаем None и завершаем работу функции.

```
def opredelitel(matrix):
    a1, a2, a3 = matrix[0][0], matrix[1][0], matrix[2][0]
    b1, b2, b3 = matrix[0][1], matrix[1][1], matrix[2][1]
    c1, c2, c3 = matrix[0][2], matrix[1][2], matrix[2][2]
    opred = a1 * b2 * c3 + a3 * b1 * c2 + a2 * b3 * c1 - a3 * b2 * c1 - a1 * b3 * c2 - a2 * b1 * c3
    return opred


def reverse_matrix(matrix):
    opred = opredelitel(matrix)
    if opred == 0:
        return None
    else:
        matrix_alg_dop = minor(matrix)
        tr_matrix_alg_dop = transp(matrix_alg_dop)
        reverse_opred = 1 / opred
        answer = tr_matrix_alg_dop
        for i in range(len(tr_matrix_alg_dop)):
            for j in range(len(tr_matrix_alg_dop[i])):
                answer[i][j] = reverse_opred * tr_matrix_alg_dop[i][j]
        return answer
```

Приложение 7 – Содержание функций reverse_matrix и opredelitel

Далее с помощью функции minor(matrix) находим матрицу миноров и матрицу алгебраических дополнений matrix_alg_dop. После этого с помощью функции transp(matrix), описанной ранее, получаем транспонированную матрицу алгебраических дополнений tr_matrix_alg_dop. Чтобы получить ответ, нам необходимо умножить каждый элемент матрицы на число, обратное определителю. В итоге мы получаем матрицу в -1 степени.

```python
def minor(matrix):
    minor_matrix = []
    a1, a2, a3 = matrix[0][0], matrix[1][0], matrix[2][0]
    b1, b2, b3 = matrix[0][1], matrix[1][1], matrix[2][1]
    c1, c2, c3 = matrix[0][2], matrix[1][2], matrix[2][2]
    new_a1 = b2 * c3 - c2 * b3
    new_a2 = -1 * (b1 * c3 - c1 * b3)
    new_a3 = b1 * c2 - c1 * b2
    new_b1 = -1 * (a2 * c3 - c2 * a3)
    new_b2 = a1 * c3 - c1 * a3
    new_b3 = -1 * (a1 * c2 - c1 * a2)
    new_c1 = a2 * b3 - b2 * a3
    new_c2 = -1 * (a1 * b3 - b1 * a3)
    new_c3 = a1 * b2 - b1 * a2
    minor_matrix.append([new_a1, new_b1, new_c1])
    minor_matrix.append([new_a2, new_b2, new_c2])
    minor_matrix.append([new_a3, new_b3, new_c3])
    return minor_matrix
```

Приложение 8 – Содержание функции minor

3.2 Описание функции возведения матрицы в -1 степень при помощи NumPy

На вход функции подаётся матрица, которая в первой строке функции становится массивом типа numpy. Получение матрицы в -1 степени происходит с помощью функции numpy.linalg.inv(matrix).

```
def reverse_matrix_numpy(matrix):
    num_matrix = numpy.array(matrix)
    answer = numpy.linalg.inv(num_matrix)
    return answer
```

Приложение 9 – Содержание функции reverse_matrix_numpy

3.3 Сравнение времени работы самостоятельно написанного алгоритма возведения матрицы в -1 степень и алгоритма из библиотеки NumPy

Проводить сравнения времени мы будем с помощью библиотеки timeit. Проведём 4 теста и попробуем вычислить закономерность. Ниже представлена таблица с замерами.

№ теста	Собственный алгоритм	Алгоритм NumPy
1	0.0002716999999998748	0.0000306000000001028
2	0.0002501999999999782	0.0000296999999997993
3	0.00031060000000060484	0.0000239000000004097
4	0.0003543999999990888	0.0000245999999997082

Таблица 1 – Результаты замеров времени выполнения

Выполнив замеры времени выполнения, можно сказать, что в среднем самостоятельно написанный алгоритм в 11, 2 раза медленнее алгоритма из библиотеки NumPy. Это достаточно значительный перевес, который показывает превосходство алгоритма из библиотеки NumPy.

СПИСОК ЛИТЕРАТУРЫ

1) NumPy: матрицы и операции над ними // URL: http://cs.mipt.ru/advanced_python/lessons/lab16.html#section-14 (дата обращения: 05.10.2022)
2) Как найти обратную матрицу? // URL: http://mathprofi.ru/kak_naiti_obratnuyu_matricu.html (дата обращения: 05.10.2022)
3) Модуль timeit в Python: как работает с примерами // URL: https://pythonim.ru/moduli/timeit-python (дата обращения: 05.10.2022)
4) Нахождение ранга матрицы: методы и примеры нахождения // URL: https://zaochnik.com/spravochnik/matematika/matritsy/rang-matritsy/ (дата обращения: 06.10.2022)