

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

**ОТЧЕТ ПО  
ЛАБОРАТОРНОЙ РАБОТЕ №1**

Выполнил:  
Абдулов И.А.

Проверил:  
Мусаев А.А.

Санкт-Петербург  
2022

## СОДЕРЖАНИЕ

### СОДЕРЖАНИЕ 1

***ВВЕДЕНИЕ***      **2**

***Задание 2***    **3**

***Ввод матрицы***    **3**

**Транспонирование матрицы**    **3**

**Умножение матриц**    **4**

**Ранг матрицы**    **4**

**Вывод данных**    **6**

**Пример работы**      **7**

***Задание 2***    **8**

***Задание 3***    **10**

**Обратная матрица**    **10**

**Решение с питру и сравнение времени работы**    **11**

***Заключение***      **13**

***Список использованных источников***      **14**

## **ВВЕДЕНИЕ**

Данная лабораторная работа направлена на работу с матрицами в языке программирования Python.

В рамках выполнения данной лабораторной работы были поставлены данные задачи:

1. Задание 2: создать программу на языке Python, которая будет выполнять транспонирование, умножение, определение ранга матриц.
2. Задание 3: создать вышеописанную программу, используя библиотеку Numpy. Также необходимо проанализировать достоинства и недостатки использования Numpy.
3. Задание 4: Создать функцию для возведения матрицы в -1 степень и сравнить время выполнения с аналогичной из библиотеки Numpy.

## ЗАДАНИЕ 2

Задания: создать программу на языке Python, которая будет выполнять транспонирование, умножение, определение ранга матриц.

### ВВОД МАТРИЦЫ

Для ввода матрицы с клавиатуры реализуем генератор списков.

```
54 print('Введите матрицу 3 x 3: ')
55 array = [[i for i in map(int, input().split())] for j in range(n)]
```

Рисунок 1 — Генератор списков в Python

### Транспонирование матрицы

Создадим функцию транспонирования матрицы `transp()`, которая принимает в качестве аргументов матрицу `array` и ее размерности `n` и `m`. В функции создается нулевая матрица `transp_array`. После запускаются два цикла `for`, по строкам и столбцам, в которых элементу `j`-ой строки, `i`-ого столбца матрицы `transp_array` присваивается элемент `i`-ой строки, `j`-ого столбца матрицы `array`. Функция возвращает транспонированную матрицу `transp_array`.

```
10 def transp(array, n, m):
11     transp_array = [[0 for i in range(n)] for j in range(m)]
12     for i in range(n):
13         for j in range(m):
14             transp_array[j][i] = array[i][j]
15     return transp_array
16
```

Рисунок 2 — Транспонирование матрицы

## Умножение матриц

Следующая функция `mult()` позволяет перемножить заданные ей в качестве аргументов матрицы `arr1` и `arr2`. При умножении матриц количество столбцов матрицы `arr1` должно совпадать с количеством строк матрицы `arr2`, поэтому в начале мы делаем проверку этого условия умножения матриц. Если матрицы подходят, то создаем нулевую матрицу `mult_array`. После делаем вложенные циклы, в которых для элемента `mult_array[i][j]` конечной матрицы, вычислим сумму произведений элементов строки `i` первой матрицы `arr1` и элементов столбца `j` второй матрицы `arr2`, по строке и столбцу идет цикл с переменной `k`. После завершения работы циклов возвращаем полученную матрицу `mult_array`.

```
17
18 def mult(arr1, arr2):
19     h1, w1, h2, w2 = len(arr1), len(arr1[0]), len(arr2), len(arr2[0])
20     if w1 != h2: return None
21     mult_array = [[0 for i in range(w2)] for j in range(h1)]
22     for i in range(h1):
23         for j in range(w2):
24             for k in range(min(w1, h2)):
25                 mult_array[i][j] += arr1[i][k]*arr2[k][j]
26     return mult_array
27
```

Рисунок 3 — Умножение матриц

## Ранг матрицы

Для нахождения ранга матрицы будем использовать метод окаймляющих миноров.

Функция `array_rang` принимает матрицу `arr`. В начале создается нулевая матрица  $3 \times 3$  из алгебраических дополнений элементов матрицы `arr`. Далее выполняются два цикла, которые для элемента матрицы `arr[x][y]` определяют его дополнительный минор 2-го порядка, элементы минора записываются в

матрицу  $ar$ . Имея элементы определителя, вычислим его. Детерминант матрицы 2-го порядка считается по формуле:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}.$$

Рисунок 4 — Формула вычисления определителя матрицы 2 x 2

Далее считаем алгебраическое дополнение элемента  $arr[x][y]$  по формуле:

$$A_{x,y} = (-1)^{x+y} * M_{x,y}$$

Рисунок 5 — Вычисления алгебраического дополнения элемента

```

28
29 def array_rang(arr):
30     alg_dop = [[0 for i in range(n)] for j in range(n)]
31     rang = 0
32     for x in range(n):
33         for y in range(n):
34             ar = [[0 for i in range(n-1)] for j in range(n-1)]
35             for i in range(n):
36                 for j in range(n):
37                     if i<x and j<y: ar[i][j] = arr[i][j]
38                     elif i<x and j>y: ar[i][j-1] = arr[i][j]
39                     elif i>x and j<y: ar[i-1][j] = arr[i][j]
40                     elif i>x and j>y: ar[i-1][j-1] = arr[i][j]
41             det = ar[0][0]*ar[1][1]-ar[0][1]*ar[1][0]
42             if arr[x][y] != 0: rang=max(rang,1)
43             if det != 0: rang=max(rang, 2)
44             alg_dop[x][y] = (-1)**(x+y)*det
45     det3 = arr[0][0]*alg_dop[0][0]\
46         -arr[0][1]*alg_dop[0][1]\
47         +arr[0][2]*alg_dop[0][2]
48     if det3 != 0: rang = max(rang, 3)
49     return rang
50

```

Рисунок 6 — Функция вычисления ранга матрицы

Вычислим определитель всей матрицы. Если он не равен 0, то ранг матрицы, который находится в переменной `rang`, считаем равным 3. Иначе, в предыдущем цикле мы проверяли определители 1-го и 2-го порядков, тем самым увеличивая значение переменной `rang`.

Функция `array_rang` возвращает переменную `rang`, которая равна максимальному порядку минора матрицы, который не равен 0.

### Вывод данных

Для вывода матрицы в приемлемом виде была создана функция, которая выводит строки в столбик.

```
2 def array_print(array):
3     n, m = len(array), len(array[0])
4     for i in range(n):
5         for j in range(m):
6             print(f'{array[i][j]:3d}', end=' ')
7         print()
8
```

Рисунок 7 — Функция вывода матрицы

## Пример работы

```
Введите матрицу 3 x 3:  
1 2 3  
4 5 6  
7 8 9  
Исходная матрица:  
1 2 3  
4 5 6  
7 8 9  
Транспонированная матрица:  
1 4 7  
2 5 8  
3 6 9  
Произведение матриц:  
14 32 50  
32 77 122  
50 122 194  
Ранг матрицы = 2
```

Рисунок 8 — Пример 1.1



## ЗАДАНИЕ 2

Задание: выполнить задание 1 с использованием библиотеки `numpy`.  
Проанализировать достоинства и недостатки использования `numpy`.

В начале импортируем `numpy`. Реализуем три метода библиотеки, которые транспонируют, умножают матрицы и ищут их ранг.

```
1  import numpy as np
2  print('Введите матрицу 3 x 3: ')
3  N = 3
4  array = [[i for i in map(int, input().split())] for j in range(N)]
5  array = np.array(array)
6  print('матрица:\n', array)
7  print('транспонированная матрица:\n', array.T)
8  print('умножение матриц:\n', array.dot(array.T))
9  print('ранг матрицы =', np.linalg.matrix_rank(array))
10
```

Рисунок 9 — Использование `numpy`

```
Введите матрицу 3 x 3:
1 2 3
4 5 6
7 8 9
матрица:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
транспонированная матрица:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
умножение матриц:
[[ 14  32  50]
 [ 32  77 122]
 [ 50 122 194]]
ранг матрицы = 2
```

## Рисунок 10 — Пример 2.1

### **Достоинства и недостатки numpy:**

+ Огромный код, на который тратится много времени и сил можно реализовать одним методом numpy. При этом скорость работы остается высокой.

- Сильно зависит от функций и библиотек, которые не относятся к Python.

### ЗАДАНИЕ 3

Задание: реализовать свою функцию для возведения матрицы  $A$  размерности  $3 \times 3$  в степень  $-1$ , где входными переменными функции будут элементы матрицы  $A$ , вводимые вручную. С помощью библиотеки `timeit` сравнить быстродействие функции с ее аналогом из библиотеки `numpy`.

#### Обратная матрица

Для вычисления обратной матрицы я использую следующую формулу:

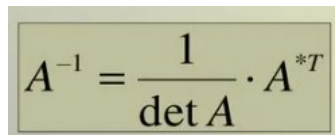

$$A^{-1} = \frac{1}{\det A} \cdot A^{*T}$$

Рисунок 11 — Формула обратной матрицы

Реализуем функцию обратной матрицы `array_inv()`, которая принимает уже транспонированную матрицу. Сначала она считает определитель матрицы 3-его порядка. Проверяет, чтобы определитель не равнялся 0. Переменная `arr_adj` является союзной матрицей для исходной. Ее мы вычислили, используя определение:

Алгебраические дополнения равны минорам умноженным на  $(-1)$  в степени суммы номера строки и столбца элемента матрицы.

Для простоты можно использовать приведенную ниже схему знаков миноров

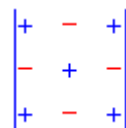

$$\begin{vmatrix} + & - & + \\ - & + & - \\ + & - & + \end{vmatrix}$$

Рисунок 12 — Схема знаков миноров

```

1 def array_inv(a11, a12, a13, a21, a22, a23, a31, a32, a33):
2     det = a11*a22*a33+a12*a23*a31+a21*a32*a13-a13*a22*a31-a11*a23*a32-a33*a12*a21
3     if det==0:
4         print("Обратной матрицы нету! Определитель равен 0!")
5         return None
6     arr_adj = [[a22*a33-a23*a32, -(a21*a33-a23*a31), a21*a32-a22*a31],
7               [-(a12*a33-a13*a32), a11*a33-a13*a31, -(a11*a32-a12*a31)],
8               [a12*a23-a13*a22, -(a11*a23-a13*a21), a11*a22-a12*a21]]
9     arr_inv = [[0 for i in range(3)] for j in range(3)]
10    for i in range(3):
11        for j in range(3):
12            arr_inv[i][j] = 1/det*arr_adj[i][j]
13    return arr_inv
14

```

Рисунок 13 — Функция нахождения обратной матрицы

В конце работы функция возвращает обратную матрицу, вычисленную по формуле с Рисунка 11.

### Решение с numru и сравнение времени работы

Теперь решим задачу с помощью numru. Для этой задачи в numru есть метод `np.linalg.inv(A)`, которая считает обратную матрицу.

Теперь засечем время. Используем библиотеку `timeit` и введем переменные, в которых будет время старта и финиша работы. `start` принимает время начала отсчета, `end` — время конца отчета. После находим их разность и получаем результат в долях секунды. Выводим результаты работы программы и метода библиотеки numru.

```

22 end1 = timeit.default_timer()
23 start2 = timeit.default_timer()
24 arr_inv = np.linalg.inv(A)
25 end2 = timeit.default_timer()
26 print('Время работы моего кода: ', '{:.10f}'.format((end1 - start1)*10**(-3)), 'секунд')
27 print('Время работы библиотеки numru: ', '{:.10f}'.format((end2 - start2)*10**(-3)), 'секунд')

```

Рисунок 14 — Вычисление времени работы

Сравним время работы.

Ниже представлено два примера. Если проанализировать их, то можно понять, что время работы моего кода лучше на матрице 3 x 3, в среднем быстрее в 4-5 раз.

```
Введите матрицу A:
```

```
7 9 10
```

```
8 3 1
```

```
4 3 5
```

```
Время работы моего кода: 0.0000000726 секунд
```

```
Время работы библиотеки numpy: 0.0000005637 секунд
```

Рисунок 15 — Пример 3.1(Разница в 7 раз)

```
Введите матрицу A:
```

```
103 109 110
```

```
123 104 108
```

```
194 120 204
```

```
Время работы моего кода: 0.0000000674 секунд
```

```
Время работы библиотеки numpy: 0.0000004769 секунд
```

Рисунок 16 — Пример 3.2 (Разница в 8 раз)

## **ЗАКЛЮЧЕНИЕ**

В ходе работы была изучена работа в Python, в частности с матрицами. Также я ознакомился с библиотекой `numpy`. Я написал функции для транспонирования умножения матриц, для нахождения их ранга и обратной матрицы. Также выполнил те же задания с использованием `numpy` и сравнил два кода. Скажу, что `numpy` определенно выигрывает и в удобстве, и в скорости.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Pythonist [Интернет-статья] – URL: <https://pythonist.ru/kak-transponirovat-matriczu-v-python/> (Дата обращения 06.10.2022).
2. GitHub [Электронный ресурс] – URL: <https://github.com/estle/itmo-uni/tree/main/sem1/ADS/lab1> (Дата обращения 29.12.2022)