МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

«Исследование методов работы с матрицами и векторами с помощью библиотеки NumPy»

Отчет по лабораторной работе № 3 по дисциплине «Технологии распознавания образов»

Выполнил студент группы ПИЖ-б-о-21-1

Гребенкин Е. А. « » 2022г.

Подпись студента____ _____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____ (подпись)

Цель работы: Цель работы: исследовать методы работы с матрицами и векторами с помощью библиотеки NumPy языка программирования Python.

Выполнение работы:

Проработка примеров:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия
    1. MIT и выбранный Вами язык программирования (выбор языка программирования будет доступен после установки флажка Add .gitignore).
    2. Выполните клонирование созданного репозитория на рабочий компьютер.
    3. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.
    4. Дополните файл .gitignore необходимыми правилами для выбранного языка программирования, интерактивной оболочки Jupyter notebook и интегрированной среды разработки.
    5. Проработать примеры лабораторной работы.
    6. Создать ноутбук, в котором будут приведены собственные примеры на языке Python для каждого из представленных свойств матричных вычислений.
    7. Создать ноутбук, в котором будут приведены собственные примеры решения систем линейных уравнений матричным методом и методом Крамера.
    8. Зафиксируйте сделанные изменения в репозитории.
    9. Выполните слияние ветки для разработки с веткой main (master).
    10. Отправьте сделанные изменения на сервер GitHub.

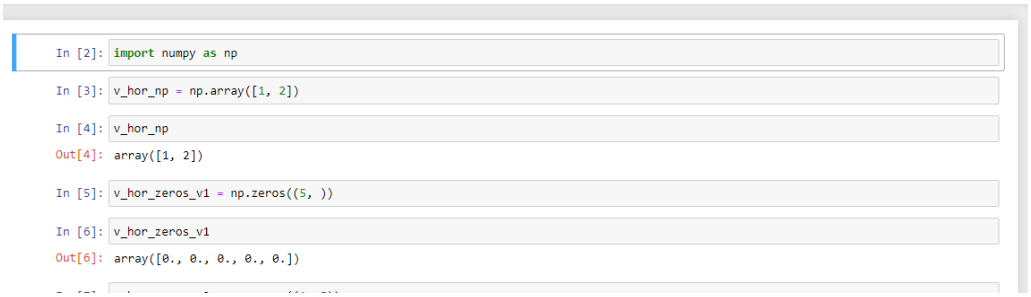

Рисунок 1 – Пример 1

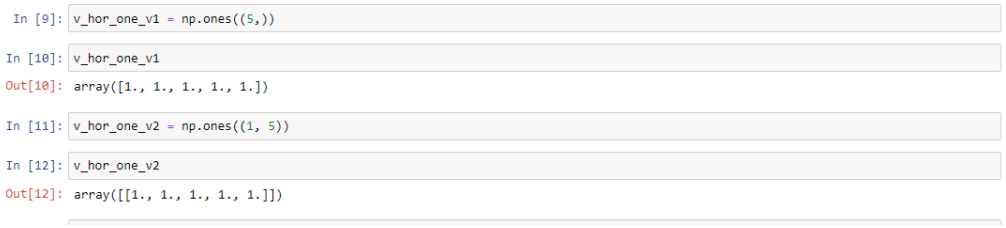

Рисунок 2 – Пример 2

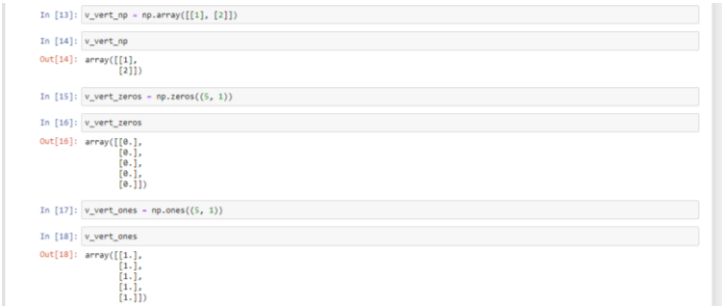

Рисунок 3 – Пример 3

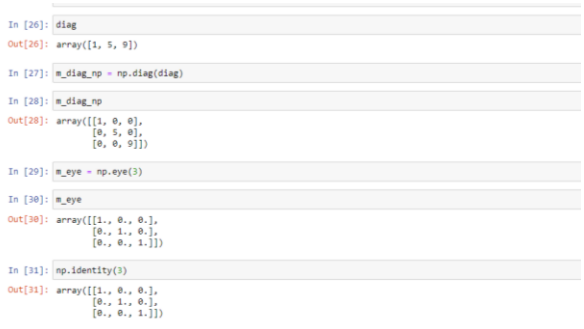

Рисунок 4 – Пример 4

```
In [37]: A.T.T
Out[37]: matrix([[1, 2, 3],
                 [4, 5, 6]])

In [38]: A = np.matrix('1 2 3; 4 5 6')
         B = np.matrix('7 8 9; 0 7 5')

In [39]: L = (A + B).T
         L
Out[39]: matrix([[ 8,  4],
                 [10, 12],
                 [12, 11]])

In [40]: R = A.T + B.T
         R
```

Рисунок 5 – Пример 5

```
In [47]: A = np.matrix('1 2; 3 4')

In [48]: A_det = np.linalg.det(A)
         format(A_det, '.9g')
Out[48]: '-2'

In [49]: A_T_det = np.linalg.det(A.T)
         format(A_T_det, '.9g')
Out[49]: '-2'
```

Рисунок 6 – Пример 6

```
In [53]: A * 1
Out[53]: matrix([[1, 2],
                 [3, 4]])

In [54]: A = np.matrix("1 2; 3 4")
         A
Out[54]: matrix([[1, 2],
                 [3, 4]])

In [55]: Z = np.matrix("0 0; 0 0")
         Z
Out[55]: matrix([[0, 0],
                 [0, 0]])
```

Рисунок 7 – Пример 7

```
                 [36, 50]])

In [66]: A = np.matrix('1 6 3; 8 2 7')
         B = np.matrix('8 1 5; 6 9 12')

In [67]: C = A + B
         C
Out[67]: matrix([[ 9,  7,  8],
                 [14, 11, 19]])

In [68]: A = np.matrix('1 2; 3 4')
         B = np.matrix('5 6; 7 8')

In [69]: A + B
Out[69]: matrix([[ 6,  8],
                 [10, 12]])

In [70]: B + A
Out[70]: matrix([[ 6,  8],
                 [10, 12]])
```

Рисунок 8 – Пример 8

Индивидуальное задания:

$$\begin{cases} 2x_1 - x_2 + 3x_3 = 1 \\ -2x_2 + 2x_3 = 2 \\ 3x_1 + x_2 + x_3 = 0 \end{cases}$$

Матричный метод

```
In [1]: import numpy as np

In [2]: B = np.matrix("1; 2; 0")

In [3]: A = np.matrix("2 -1 3; 0 -2 2; 3 1 1")

In [4]: A_inv = np.linalg.inv(A)
        A_inv

Out[4]: matrix([[-1.  ,  1.  ,  1.  ],
                [ 1.5 , -1.75, -1.  ],
                [ 1.5 , -1.25, -1.  ]])

In [7]: X = A_inv.dot(B)
        X

Out[7]: matrix([[ 1.],
                [-2.],
                [-1.]])
```

Ответ: x1 = 1, x2 = -2, x3 = -1

Рисунок 9 – Решение задач

Метод Крамера

```
In [42]: A = np.matrix("2 -1 3; 0 -2 2; 3 1 1")
         AB = np.matrix("2 -1 3 1; 0 -2 2 2; 3 1 1 0")

In [43]: A_main_det = round(np.linalg.det(A), 3)
         A_main_det

Out[43]: 4.0

In [44]: A_x1_det = round(np.linalg.det(AB[:, 1:]), 3)
         A_x1_det

Out[44]: 4.0

In [51]: A_x2_det = -round(np.linalg.det(AB[:, [0, 2, 3]]), 3)
         A_x2_det

Out[51]: -8.0

In [52]: A_x3_det = round(np.linalg.det(AB[:, [0, 1, 3]]), 3)
         A_x3_det

Out[52]: -4.0

In [53]: x1 = A_x1_det / A_main_det
         x1

Out[53]: 1.0

In [54]: x2 = A_x2_det / A_main_det
         x2

Out[54]: -2.0

In [55]: x3 = A_x3_det / A_main_det
         x3

Out[55]: -1.0
```

Ответ: x1 = 1, x2 = 2, x3 = 3

Рисунок 10 – Решение задач

Контрольные вопросы

1. Приведите основные виды матриц и векторов. Опишите способы их создания в языке Python.

Вектор-строка

Вектор-строка имеет следующую математическую запись.

$$v = (1\ 2) \tag{3}$$

Такой вектор в *Python* можно задать следующим образом.

```
>>> v_hor_np = np.array([1, 2])
>>> print(v_hor_np )
[1 2]
```

Если необходимо создать **нулевой** или **единичный вектор**, то есть вектор, у которого все элементы нули либо единицы, то можно использовать специальные функции из библиотеки *Numpy*.

Создадим нулевую вектор-строку размера *5*.

```
>>> v_hor_zeros_v1 = np.zeros((5,))
>>> print(v_hor_zeros_v1 )
[0. 0. 0. 0. 0.]
```

Вектор-столбец

Вектор-столбец имеет следующую математическую запись.

$$v = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{4}$$

В общем виде вектор столбец можно задать следующим образом.

```
>>> v_vert_np = np.array([[1], [2]])
>>> print(v_vert_np)
[[1]
[2]]
```

Квадратная матрица

Довольно часто, на практике, приходится работать с **квадратными матрицами**. Квадратной называется матрица, у которой количество столбцов и строк совпадает. В общем виде они выглядят так.

$$M_{sqr} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}. \tag{5}$$

Создадим следующую матрицу.

$$M_{sqr} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}. \tag{6}$$

В *Numpy* можно создать квадратную матрицу с помощью метода *array()*.

```
>>> m_sqr_arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> print(m_sqr_arr)
[[1 2 3]
[4 5 6]
[7 8 9]]
```

Диагональная матрица

Особым видом квадратной матрицы является **диагональная** – это такая матрица, у которой все элементы, кроме тех, что расположены на главной диагонали, равны нулю.

$$M_{diag} = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix} \tag{7}$$

Диагональную матрицу можно построить вручную, задав только значения элементам на главной диагонали.

```
>>> m_diag = [[1, 0, 0], [0, 5, 0], [0, 0, 9]]
>>> m_diag_np = np.matrix(m_diag)
>>> print(m_diag_np)
[[1 0 0]
[0 5 0]
[0 0 9]]
```

Единичная матрица

Единичной матрицей называют такую квадратную матрицу, у которой элементы главной диагонали равны единицы, а все остальные нулю.

$$E = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}. \tag{8}$$

Создадим единичную матрицу на базе списка, который передадим в качестве аргумента функции *matrix()*.

```
>>> m_e = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
>>> m_e_np = np.matrix(m_e)
>>> print(m_e_np)
[[1 0 0]
[0 1 0]
[0 0 1]]
```

2. Как выполняется транспонирование матриц?

Транспонируем матрицу с помощью метода ***transpose()***:

```
>>> A_t = A.transpose()
>>> print(A_t)
[[1 4]
[2 5]
[3 6]]
```

3. Приведите свойства операции транспонирования матриц.

Свойство 1. Дважды транспонированная матрица равна исходной матрице:

$$(A^T)^T = A.$$

➢ *Численный пример*

$$\left(\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}^T\right)^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}^T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> print(A)
[[1 2 3]
[4 5 6]]
>>> R = (A.T).T
>>> print(R)
[[1 2 3]
[4 5 6]]
```

Свойство 2. Транспонирование суммы матриц равно сумме транспонированных матриц:

$$(A+B)^T = A^T + B^T.$$

➢ *Численный пример*

$$\left(\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix} + \begin{pmatrix}7 & 8 & 9\\ 0 & 7 & 5\end{pmatrix}\right)^T = \begin{pmatrix}8 & 10 & 12\\ 4 & 12 & 11\end{pmatrix}^T = \begin{pmatrix}8 & 4\\ 10 & 12\\ 12 & 11\end{pmatrix}.$$

$$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}^T + \begin{pmatrix}7 & 8 & 9\\ 0 & 7 & 5\end{pmatrix}^T = \begin{pmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{pmatrix} + \begin{pmatrix}7 & 0\\ 8 & 7\\ 9 & 5\end{pmatrix} = \begin{pmatrix}8 & 4\\ 10 & 12\\ 12 & 11\end{pmatrix}.$$

➤*Пример на Python*

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> B = np.matrix('7 8 9; 0 7 5')
>>> L = (A + B).T
>>> R = A.T + B.T
>>> print(L)
[[ 8  4]
[10 12]
[12 11]]
>>> print(R)
[[ 8  4]
[10 12]
[12 11]]
```

Свойство 3. Транспонирование произведения матриц равно произведению транспонированных матриц расставленных в обратном порядке:

$$(AB)^T = B^T A^T.$$

➢ *Численный пример*

$$\left(\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} \times \begin{pmatrix}5 & 6\\7 & 8\end{pmatrix}\right)^T = \begin{pmatrix}19 & 22\\43 & 50\end{pmatrix}^T = \begin{pmatrix}19 & 43\\22 & 50\end{pmatrix}.$$

$$\begin{pmatrix}5 & 6\\7 & 8\end{pmatrix}^T \times \begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}^T = \begin{pmatrix}5 & 7\\6 & 8\end{pmatrix} \times \begin{pmatrix}1 & 3\\2 & 4\end{pmatrix} = \begin{pmatrix}19 & 43\\22 & 50\end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> L = (A.dot(B)).T
>>> R = (B.T).dot(A.T)
>>> print(L)
[[19 43]
[22 50]]
>>> print(R)
[[19 43]
[22 50]]
```

В данном примере, для умножения матриц, использовалась функция ***dot()*** из библиотеки *Numpy*.

Свойство 4. Транспонирование произведения матрицы на число равно произведению этого числа на транспонированную матрицу:

$$(\lambda A)^T = \lambda A^T.$$

➢ *Численный пример*

$$\left(3 \cdot \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}\right)^T = \begin{pmatrix} 3 & 6 & 9 \\ 12 & 15 & 18 \end{pmatrix}^T = \begin{pmatrix} 3 & 12 \\ 6 & 15 \\ 9 & 18 \end{pmatrix},$$

$$3 \cdot \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}^T = 3 \cdot \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} = \begin{pmatrix} 3 & 12 \\ 6 & 15 \\ 9 & 18 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> k = 3
>>> L = (k * A).T
>>> R = k * (A.T)
>>> print(L)
[[ 3 12]
[ 6 15]
[ 9 18]]
\>>> print(R)
[[ 3 12]
[ 6 15]
[ 9 18]]
```

Свойство 5. Определители исходной и транспонированной матрицы совпадают:

$$|A| = |A^T|.$$

➢ *Численный пример*

$$det\left(\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}\right) = 4 - 6 = -2.$$

$$det\left(\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}^T\right) = det\left(\begin{pmatrix}1 & 3\\ 2 & 4\end{pmatrix}\right) = 4 - 6 = -2.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> A_det = np.linalg.det(A)
>>> A_T_det = np.linalg.det(A.T)
>>> print(format(A_det, '.9g'))
-2
>>> print(format(A_T_det, '.9g'))
-2
```

Ввиду особенностей *Python* при работе с числами с плавающей точкой, в данном примере вычисления определителя рассматриваются только первые девять значащих цифр после запятой (за это отвечает параметр `'.9g'`).

4. Какие имеются средства в библиотеке NumPy для выполнения транспонирования матриц?

Решим задачу транспонирования матрицы на *Python*. Создадим матрицу *A*:

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> print(A)
[[1 2 3]
[4 5 6]]
```

Транспонируем матрицу с помощью метода ***transpose()***:

```
>>> A_t = A.transpose()
>>> print(A_t)
[[1 4]
[2 5]
[3 6]]
```

5. Какие существуют основные действия над матрицами?

Умножение матрицы на число

Сложение матриц

Умножение матриц

Определитель матрицы

Транспонирование матрицы

6. Как осуществляется умножение матрицы на число?

► Пример на Python

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> C = 3 * A
>>> print(C)
[[ 3  6 9]
[12 15 18]]
```

7. Какие свойства операции умножения матрицы на число?

Свойство 1. Произведение единицы и любой заданной матрицы равно заданной матрице:

$$1 \cdot A = A.$$

➢ *Численный пример*

$$1 \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> L = 1 * A
>>> R = A
>>> print(L)
[[1 2]
[3 4]]
>>> print(R)
[[1 2]
[3 4]]
```

Свойство 2. Произведение нуля и любой матрицы равно нулевой матрице, размерность которой равна исходной матрицы:

$$0 \cdot A = Z.$$

➢ *Численный пример*

$$0 \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> Z = np.matrix('0 0; 0 0')
>>> L = 0 * A
>>> R = Z
>>> print(L)
[[0 0]
[0 0]]
>>> print(R)
[[0 0]
[0 0]]
```

Свойство 3. Произведение матрицы на сумму чисел равно сумме произведений матрицы на каждое из этих чисел:

$$(\alpha+\beta)\cdot A=\alpha\cdot A+\beta\cdot A.$$

➢ *Численный пример*

$$(2+3)\cdot\begin{pmatrix}1&2\\3&4\end{pmatrix}=2\cdot\begin{pmatrix}1&2\\3&4\end{pmatrix}+3\cdot\begin{pmatrix}1&2\\3&4\end{pmatrix}=\begin{pmatrix}2&4\\6&8\end{pmatrix}+\begin{pmatrix}3&6\\9&12\end{pmatrix}=\begin{pmatrix}5&10\\15&20\end{pmatrix},$$

$$5\cdot\begin{pmatrix}1&2\\3&4\end{pmatrix}=\begin{pmatrix}5&10\\15&20\end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> p = 2
>>> q = 3
>>> L = (p + q) * A
>>> R = p * A + q * A
>>> print(L
[[ 5 10]
[15 20]]
>>> print(R)
[[ 5 10]
[15 20]]
```

Свойство 4. Произведение матрицы на произведение двух чисел равно произведению второго числа и заданной матрицы, умноженному на первое число:

$$(\alpha \cdot \beta) \cdot A = \alpha \cdot (\beta \cdot A).$$

➢ *Численный пример*

$$(2 \cdot 3) \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = 2 \cdot \left(3 \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \right) = 2 \cdot \begin{pmatrix} 3 & 6 \\ 9 & 12 \end{pmatrix} = \begin{pmatrix} 6 & 12 \\ 18 & 24 \end{pmatrix},$$

$$(2 \cdot 3) \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = 6 \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 6 & 12 \\ 18 & 24 \end{pmatrix}.$$

➤*Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> p = 2
>>> q = 3
>>> L = (p * q) * A
>>> R = p * (q * A)
>>> print(L)
[[ 6 12]
[18 24]]
>>> print(R)
[[ 6 12]
[18 24]]
```

Свойство 5. Произведение суммы матриц на число равно сумме произведений этих матриц на заданное число:

$$\lambda \cdot (A + B) = \lambda \cdot A + \lambda \cdot B.$$

➢ *Численный пример*

$$3 \cdot \left(\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \right) = 3 \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + 3 \cdot \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 18 & 24 \\ 30 & 36 \end{pmatrix},$$

$$3 \cdot \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix} = \begin{pmatrix} 18 & 24 \\ 30 & 36 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> k = 3
>>> L = k * (A + B)
>>> R = k * A + k * B
>>> print(L)
[[18 24]
[30 36]]
>>> print(R)
[[18 24]
[30 36]]
```

8. Как осуществляется операции сложения и вычитания матриц?

➤ *Пример на Python*

```
>>> A = np.matrix('1 6 3; 8 2 7')
>>> B = np.matrix('8 1 5; 6 9 12')
>>> C = A + B
>>> print(C)
[[ 9  7 8]
[14 11 19]]
```

9. Каковы свойства операций сложения и вычитания матриц?

Свойство 1. Коммутативность сложения. От перестановки матриц их сумма не изменяется:

$$A + B = B + A.$$

➢ *Численный пример*

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} + \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> L = A + B
>>> R = B + A
>>> print(L)
[[ 6  8]
[10 12]]
>>> print(R)
[[ 6  8]
[10 12]]
```

Свойство 2. Ассоциативность сложения. Результат сложения трех и более матриц не зависит от порядка, в котором эта операция будет выполняться:

$$A + (B + C) = (A + B) + C.$$

➢ *Численный пример*

$$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} + \left(\begin{pmatrix}5 & 6\\7 & 8\end{pmatrix} + \begin{pmatrix}1 & 7\\9 & 3\end{pmatrix}\right) = \begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} + \begin{pmatrix}6 & 13\\16 & 11\end{pmatrix} = \begin{pmatrix}7 & 15\\19 & 15\end{pmatrix},$$

$$\left(\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} + \begin{pmatrix}5 & 6\\7 & 8\end{pmatrix}\right) + \begin{pmatrix}1 & 7\\9 & 3\end{pmatrix} = \begin{pmatrix}6 & 8\\10 & 12\end{pmatrix} + \begin{pmatrix}1 & 7\\9 & 3\end{pmatrix} = \begin{pmatrix}7 & 15\\19 & 15\end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> C = np.matrix('1 7; 9 3')
>>> L = A + (B + C)
>>> R = (A + B) + C
>>> print(L)
[[ 7 15]
[19 15]]
>>> print(R)
[[ 7 15]
[19 15]]
```

Свойство 3. Для любой матрицы существует противоположная ей , такая, что их сумма является нулевой матрицей :

$$A + (-A) = Z.$$

➢ *Численный пример*

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + (-1) \cdot \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} -1 & -2 \\ -3 & -4 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> Z = np.matrix('0 0; 0 0')
>>> L = A + (-1)*A
>>> print(L)
[[0 0]
[0 0]]
>>> print(Z)
[[0 0]
[0 0]]
```

10. Какие имеются средства в библиотеке NumPy для выполнения операций сложения и вычитания матриц?

```
>>> A = np.matrix('1 6 3; 8 2 7')
>>> B = np.matrix('8 1 5; 6 9 12')
>>> C = A + B
>>> print(C)
[[ 9  7 8]
[14 11 19]]
```

11. Как осуществляется операция умножения матриц?

Решим задачу умножения матриц на языке *Python*. Для этого будем использовать функцию **dot()** из библиотеки *Numpy*:

```
>>> A = np.matrix('1 2 3; 4 5 6')
>>> B = np.matrix('7 8; 9 1; 2 3')
>>> C = A.dot(B)
>>> print(C)
[[31 19]
[85 55]]
```

12. Каковы свойства операции умножения матриц?

Свойство 1. Ассоциативность умножения. Результат умножения матриц не зависит от порядка, в котором будет выполняться эта операция:

$$A \times (B \times C) = (A \times B) \times C.$$

➢ *Численный пример*

$$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} \times \left(\begin{pmatrix}5 & 6\\7 & 8\end{pmatrix} \times \begin{pmatrix}2 & 4\\7 & 8\end{pmatrix}\right) = \begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} \times \begin{pmatrix}52 & 68\\70 & 92\end{pmatrix} = \begin{pmatrix}192 & 252\\436 & 572\end{pmatrix},$$

$$\left(\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix} \times \begin{pmatrix}5 & 6\\7 & 8\end{pmatrix}\right) \times \begin{pmatrix}2 & 4\\7 & 8\end{pmatrix} = \begin{pmatrix}19 & 22\\43 & 50\end{pmatrix} \times \begin{pmatrix}2 & 4\\7 & 8\end{pmatrix} = \begin{pmatrix}192 & 252\\436 & 572\end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> C = np.matrix('2 4; 7 8')
>>> L = A.dot(B.dot(C))
>>> R = (A.dot(B)).dot(C)
>>> print(L)
[[192 252]
[436 572]]
>>> print(R)
[[192 252]
[436 572]]
```

Свойство 2. Дистрибутивность умножения. Произведение матрицы на сумму матриц равно сумме произведений матриц:

$$A \times (B + C) = A \times B + A \times C.$$

➢ *Численный пример*

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \left(\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} + \begin{pmatrix} 2 & 4 \\ 7 & 8 \end{pmatrix} \right) = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 7 & 10 \\ 14 & 16 \end{pmatrix} = \begin{pmatrix} 35 & 42 \\ 77 & 94 \end{pmatrix},$$

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} + \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 2 & 4 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix} + \begin{pmatrix} 16 & 20 \\ 34 & 44 \end{pmatrix} = \begin{pmatrix} 35 & 42 \\ 77 & 94 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> C = np.matrix('2 4; 7 8')
>>> L = A.dot(B + C)
>>> R = A.dot(B) + A.dot(C)
>>> print(L)
[[35 42]
[77 94]]
>>> print(R)
[[35 42]
[77 94]]
```

Свойство 3. Умножение матриц в общем виде не коммутативно. Это означает, что для матриц не выполняется правило независимости произведения от перестановки множителей:

$$A \times B \neq B \times A.$$

➢ *Численный пример*

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix},$$

$$\begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 23 & 34 \\ 31 & 46 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> B = np.matrix('5 6; 7 8')
>>> L = A.dot(B)
>>> R = B.dot(A)
>>> print(L)
[[19 22]
[43 50]]
>>> print(R)
[[23 34]
[31 46]]
```

Свойство 4. Произведение заданной матрицы на единичную равно исходной матрице:

$$E \times A = A \times E = A.$$

➢ *Численный пример*

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix},$$

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> E = np.matrix('1 0; 0 1')
>>> L = E.dot(A)
>>> R = A.dot(E)
>>> print(L)
[[1 2]
[3 4]]
>>> print(R)
[[1 2]
[3 4]]
>>> print(A)
[[1 2]
[3 4]]
```

Свойство 5. Произведение заданной матрицы на нулевую матрицу равно нулевой матрице:

$$Z \times A = A \times Z = Z.$$

➢ *Численный пример*

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

➤ *Пример на Python*

```
>>> A = np.matrix('1 2; 3 4')
>>> Z = np.matrix('0 0; 0 0')
>>> L = Z.dot(A)
>>> R = A.dot(Z)
>>> print(L)
[[0 0]
[0 0]]
>>> print(R)
[[0 0]
[0 0]]
>>> print(Z)
[[0 0]
[0 0]]
```

13. Какие имеются средства в библиотеке NumPy для выполнения операции умножения матриц?

Есть три основных способа выполнить умножение матрицы NumPy:

- np.dot(array a, array b): возвращает скалярное произведение или скалярное произведение двух массивов
- np.matmul(array a, array b): возвращает матричное произведение двух массивов
- np.multiply(array a, array b): возвращает поэлементное матричное умножение двух массивов

14. Что такое определитель матрицы? Каковы свойства определителя матрицы?

Определитель матрицы

Определитель матрицы размера (*n*-го порядка) является одной из ее численных характеристик. Определитель матрицы *A* обозначается как $|A|$ или *det(A)*, его также называют детерминантом. Рассмотрим квадратную матрицу 2×2 в общем виде:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}.$$

Определитель такой матрицы вычисляется следующим образом:

$$|A| = det(A) = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \times a_{22} - a_{12} \times a_{21}.$$

Свойство 1. Определитель матрицы остается неизменным при ее транспонировании:

$$det(A) = det(A^T).$$

Свойство 2. Если у матрицы есть строка или столбец, состоящие из нулей, то определитель такой матрицы равен нулю:

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = 0.$$

Свойство 3. При перестановке строк матрицы знак ее определителя меняется на противоположный:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}; \; A' = \begin{pmatrix} a_{21} & a_{22} & \dots & a_{2n} \\ a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix},$$

$$det(A) = -det(A').$$

Свойство 4. Если у матрицы есть две одинаковые строки, то ее определитель равен нулю:

$$\begin{vmatrix} a_1 & a_2 & \dots & a_n \\ a_1 & a_2 & \dots & a_n \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = 0.$$

Свойство 5. Если все элементы строки или столбца матрицы умножить на какое-то число, то и определитель будет умножен на это число:

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \lambda \cdot a_{21} & \lambda \cdot a_{22} & \dots & \lambda \cdot a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \lambda \cdot det(A).$$

Свойство 6. Если все элементы строки или столбца можно представить как сумму двух слагаемых, то определитель такой матрицы равен сумме определителей двух соответствующих матриц:

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2n}+b_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}.$$

Свойство 7. Если к элементам одной строки прибавить элементы другой строки, умноженные на одно и тоже число, то определитель матрицы не изменится:

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} + \beta \cdot a_{11} & a_{22} + \beta \cdot a_{12} & \dots & a_{2n} + \beta \cdot a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}.$$

Свойство 8. Если строка или столбец матрицы является линейной комбинацией других строк (столбцов), то определитель такой матрицы равен нулю:

$$a_{2i} = \alpha \cdot a_{1i} + \beta \cdot a_{3i}; \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = 0.$$

15. Какие имеются средства в библиотеке NumPy для нахождения значения определителя матрицы?

```
>>> A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
>>> print(A)
[[-4 -1 2]
[10 4 -1]
[ 8 3 1]]
```

Для вычисления определителя этой матрицы воспользуемся функцией *det()* из пакета *linalg*.

```
>>> np.linalg.det(A)
-14.000000000000009
```

16. Что такое обратная матрица? Какой алгоритм нахождения обратной матрицы?

Обратной матрицей A^{-1} матрицы A называют матрицу, удовлетворяющую следующему равенству:

$$A \times A^{-1} = A^{-1} \times A = E,$$

где – E это единичная матрица.

Для того, чтобы у квадратной матрицы A была обратная матрица необходимо и достаточно чтобы определитель $|A|$ был не равен нулю. Введем понятие **союзной матрицы**. Союзная матрица A строится на базе исходной A путем замены всех элементов матрицы A на их алгебраические дополнения.

Исходная матрица:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}.$$

Союзная ей матрица A:

$$A^* = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \dots & \dots & \dots & \dots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{pmatrix}.$$

Транспонируя матрицу A, мы получим так называемую присоединенную матрицу A^T:

$$A^{*T} = \begin{pmatrix} A_{11} & A_{21} & \dots & A_{n1} \\ A_{12} & A_{22} & \dots & A_{n2} \\ \dots & \dots & \dots & \dots \\ A_{1n} & A_{2n} & \dots & A_{nn} \end{pmatrix}.$$

Теперь, зная как вычислять определитель и присоединенную матрицу, мы можем определить матрицу A^{-1}, обратную матрице A:

$$A^{-1} = \frac{1}{det(A)} \times A^{*T}.$$

17. Каковы свойства обратной матрицы?

Свойство 1. Обратная матрица обратной матрицы есть исходная матрица:

$$(A^{-1})^{-1} = A.$$

Свойство 2. Обратная матрица транспонированной матрицы равна транспонированной матрице от обратной матрицы:

$$(A^T)^{-1} = (A^{-1})^T.$$

Свойство 3. Обратная матрица произведения матриц равна произведению обратных матриц:

$$(A_1 \times A_2)^{-1} = A_2^{-1} \times A_1^{-1}.$$

18. Какие имеются средства в библиотеке NumPy для нахождения обратной матрицы?

Решим задачу определения обратной матрицы на *Python*. Для получения обратной матрицы будем использовать функцию ***inv()***:

```
>>> A = np.matrix('1 -3; 2 5')
>>> A_inv = np.linalg.inv(A)
>>> print(A_inv)
[[ 0.45454545 0.27272727]
 [-0.18181818 0.09090909]]
```

19. Самостоятельно изучите метод Крамера для решения систем линейных уравнений. Приведите алгоритм решения системы линейных уравнений методом Крамера средствами библиотеки NumPy.

Решение систем линейных уравнений методом крамера python (al-shell.ru)

20. Самостоятельно изучите матричный метод для решения системлинейных уравнений. Приведите алгоритм решения системы линейных уравнений матричным методом средствами библиотеки

NumPy

Решение систем линейных уравнений в Python (xn--80ahcjeib4ac4d.xn--p1ai)