МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Отчет по лабораторной работе № 3.3

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

по дисциплине «Основы программной инженерии»

Выполнил студент группы
ПИЖ-б-о-21-1
Зиёдуллаев Жавохир
« » марта 2023 г.
Подпись студента_____
Работа защищена
« »_____20__г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь, 2023

Цель работы:

Исследовать базовые возможности библиотеки NumPy языка программирования Python.

Выполнение работы:

Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT, рисунок 1.

Ссылка: https://github.com/javoxir21/tro_3laba.git

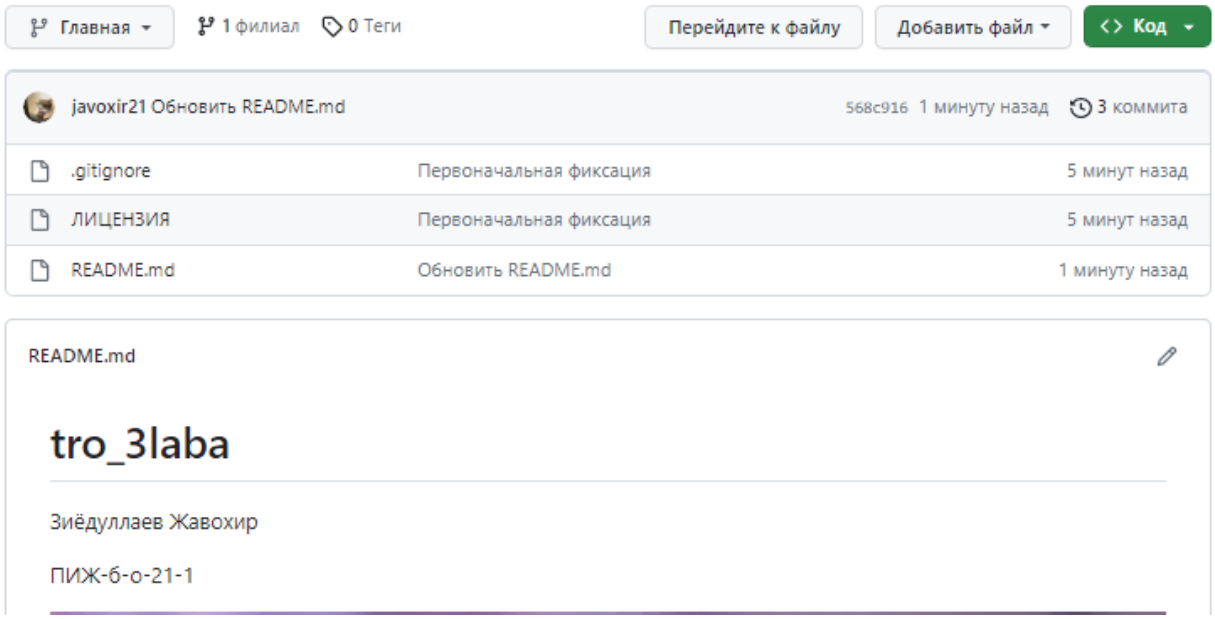

Рисунок 1 – Удаленный репозиторий на GitHub

Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm, рисунок 2.

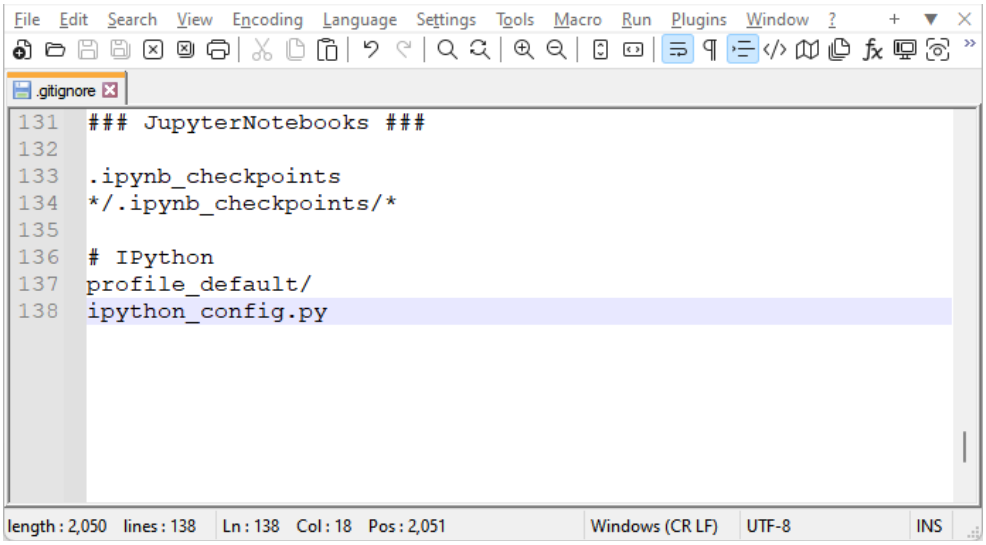

Рисунок 2 – Окно блокнота

Организуйте свой репозиторий в соответствие с моделью ветвления git-flow, рисунок 3.

```
* develop
  main
```

Рисунок 3 – Окно командной строки

Проработать примеры лабораторной работы.

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

Вектор строка

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

```
[1 2]
```

```
In [3]: v_hor_zeros_v1 = np.zeros((5,))
        print(v_hor_zeros_v1 )
```

```
[0. 0. 0. 0. 0.]
```

```
In [4]: v_hor_zeros_v2 = np.zeros((1, 5))
        print(v_hor_zeros_v2 )
```

```
[[0. 0. 0. 0. 0.]]
```

```
In [5]: v_hor_one_v1 = np.ones((5,))
        print(v_hor_one_v1)
        v_hor_one_v2 = np.ones((1, 5))
        print(v_hor_one_v2)
```

```
[1. 1. 1. 1. 1.]
[[1. 1. 1. 1. 1.]]
```

Вектор-столбец

```
In [6]: v_vert_np = np.array([[1], [2]])
        print(v_vert_np)
```

```
[[1]
 [2]]
```

```
In [7]: v_vert_zeros = np.zeros((5, 1))
        print(v_vert_zeros)
```

```
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]]
```

```
In [8]: v_vert_ones = np.ones((5, 1))
        print(v_vert_ones)
```

```
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
```

Рисунок 4 – Пример 1

Квадратная матрица

In [9]:
```
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]]
```

In [10]:
```
m_sqr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
m_sqr_arr = np.array(m_sqr)
print(m_sqr_arr)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [11]:
```
m_sqr_mx = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(m_sqr_mx)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [12]:
```
m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
print(m_sqr_mx)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Диагональная матрица

In [13]:
```
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]]
```

In [14]:
```
m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
```

In [15]:
```
diag = np.diag(m_sqr_mx)
print(diag)
```

```
[1 5 9]
```

In [16]:
```
m_diag_np = np.diag(np.diag(m_sqr_mx))
print(m_diag_np)
```

```
[[1 0 0]
 [0 5 0]
 [0 0 9]]
```

Рисунок 5 – Пример 2

Единичная матрица

In [17]:
```
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]]
```

In [18]:
```
m_eye = np.eye(3)
print(m_eye)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [19]:
```
m_idnt = np.identity(3)
print(m_idnt)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Нулевая матрица

In [20]:
```
m_zeros = np.zeros((3, 3))
print(m_zeros)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Задание матрицы в общем виде

In [21]:
```
m_mx = np.matrix('1 2 3; 4 5 6')
print(m_mx)
```

```
[[1 2 3]
 [4 5 6]]
```

In [22]:
```
m_var = np.zeros((2, 5))
print(m_var)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

Рисунок 6 – Пример 3

Транспонирование матрицы

In [23]:
```
A = np.matrix('1 2 3; 4 5 6')
print(A)
```

```
[[1 2 3]
 [4 5 6]]
```

In [24]:
```
A_t = A.transpose()
print(A_t)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

In [25]:
```
print(A.T)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

In [26]:
```
A = np.matrix('1 2 3; 4 5 6')
print(A)
```

```
[[1 2 3]
 [4 5 6]]
```

In [27]:
```
R = (A.T).T
print(R)
```

```
[[1 2 3]
 [4 5 6]]
```

In [28]:
```
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]]
```

In [29]:
```
print(R)
```

```
[[ 8  4]
 [10 12]
 [12 11]]
```

In [30]:
```
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)
print(R)
```

```
[[19 43]
 [22 50]]
[[19 43]
 [22 50]]
```

Рисунок 7 – Пример 4

In [31]:
```
A = np.matrix('1 2 3; 4 5 6')
k = 3
L = (k * A).T
R = k * (A.T)
print(L)
print(R)
```

```
[[ 3 12]
 [ 6 15]
 [ 9 18]]
[[ 3 12]
 [ 6 15]
 [ 9 18]]
```

In [32]:
```
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'))
print(format(A_T_det, '.9g'))
```

```
-2
-2
```

Действия над матрицами

Умножение матрицы на число

In [33]:
```
A = np.matrix('1 2 3; 4 5 6')
C = 3 * A
print(C)
```

```
[[ 3  6  9]
 [12 15 18]]
```

In [34]:
```
A = np.matrix('1 2; 3 4')
L = 1 * A
R = A
print(L)
print(R)
```

```
[[1 2]
 [3 4]]
[[1 2]
 [3 4]]
```

In [35]:
```
A = np.matrix('1 2; 3 4')
Z = np.matrix('0 0; 0 0')
L = 0 * A
R = Z
print(L)
print(R)
```

```
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

Рисунок 8 – Пример 5

In [36]:
```
A = np.matrix('1 2; 3 4')
p = 2
q = 3
L = (p + q) * A
R = p * A + q * A
print(L)
print(R)
```

```
[[ 5 10]
 [15 20]]
[[ 5 10]
 [15 20]]
```

In [37]:
```
A = np.matrix('1 2; 3 4')
p = 2
q = 3
L = (p * q) * A
R = p * (q * A)
print(L)
print(R)
```

```
[[ 6 12]
 [18 24]]
[[ 6 12]
 [18 24]]
```

In [38]:
```
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)
print(R)
```

```
[[18 24]
 [30 36]]
[[18 24]
 [30 36]]
```

Сложение матриц

In [39]:
```
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]]
```

In [40]:
```
A = np.matrix('1 2; 3 4')
B = np.matrix('5 6; 7 8')
L = A + B
R = B + A
print(L)
print(R)
```

```
[[ 6  8]
 [10 12]]
[[ 6  8]
 [10 12]]
```

Рисунок 9 – Пример 6

In [41]:
```
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)
print(R)
```

```
[[ 7 15]
 [19 15]]
[[ 7 15]
 [19 15]]
```

In [42]:
```
A = np.matrix('1 2; 3 4')
Z = np.matrix('0 0; 0 0')
L = A + (-1)*A
print(L)
print(Z)
```

```
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

Умножение матриц

In [43]:
```
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]]
```

In [44]:
```
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)
print(R)
```

```
[[192 252]
 [436 572]]
[[192 252]
 [436 572]]
```

In [45]:
```
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)
print(R)
```

```
[[35 42]
 [77 94]]
[[35 42]
 [77 94]]
```

Рисунок 10 – Пример 7

In [46]:
```
A = np.matrix('1 2; 3 4')
B = np.matrix('5 6; 7 8')
L = A.dot(B)
R = B.dot(A)
print(L)
print(R)
```

```
[[19 22]
 [43 50]]
[[23 34]
 [31 46]]
```

In [47]:
```
A = np.matrix('1 2; 3 4')
E = np.matrix('1 0; 0 1')
L = E.dot(A)
R = A.dot(E)
print(L)
print(R)
print(A)
```

```
[[1 2]
 [3 4]]
[[1 2]
 [3 4]]
[[1 2]
 [3 4]]
```

In [48]:
```
A = np.matrix('1 2; 3 4')
Z = np.matrix('0 0; 0 0')
L = Z.dot(A)
R = A.dot(Z)
print(L)
print(R)
print(Z)
```

```
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

Определитель матрицы

In [49]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
```

```
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
```

In [50]:
```
np.linalg.det(A)
```

Out[50]:
```
-14.000000000000009
```

Рисунок 11 – Пример 8

```
In [51]: A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
         print(A)
         print(A.T)
```

```
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
[[-4 10  8]
 [-1  4  3]
 [ 2 -1  1]]
```

```
In [52]: det_A = round(np.linalg.det(A), 3)
         det_A_t = round(np.linalg.det(A.T), 3)
         print(det_A)
         print(det_A_t)
```

```
-14.0
-14.0
```

```
In [53]: A = np.matrix('-4 -1 2; 0 0 0; 8 3 1')
         print(A)
         np.linalg.det(A)
```

```
[[-4 -1  2]
 [ 0  0  0]
 [ 8  3  1]]
```

```
Out[53]: 0.0
```

```
In [54]: A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
         print(A)
         B = np.matrix('10 4 -1; -4 -1 2; 8 3 1')
         print(B)
         round(np.linalg.det(A), 3)
         round(np.linalg.det(B), 3)
```

```
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
[[10  4 -1]
 [-4 -1  2]
 [ 8  3  1]]
```

```
Out[54]: 14.0
```

```
In [55]: A = np.matrix('-4 -1 2; -4 -1 2; 8 3 1')
         print(A)
         np.linalg.det(A)
```

```
[[-4 -1  2]
 [-4 -1  2]
 [ 8  3  1]]
```

```
Out[55]: 0.0
```

Рисунок 12 – Пример 9

In [56]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
k = 2
B = A.copy()
B[2, :] = k * B[2, :]
print(B)
det_A = round(np.linalg.det(A), 3)
det_B = round(np.linalg.det(B), 3)
det_A * k
det_B
```

```
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
[[-4 -1  2]
 [10  4 -1]
 [16  6  2]]
```

Out[56]: -28.0

In [57]:
```
A = np.matrix('-4 -1 2; -4 -1 2; 8 3 1')
B = np.matrix('-4 -1 2; 8 3 2; 8 3 1')
C = A.copy()
C[1, :] += B[1, :]
print(C)
print(A)
print(B)
round(np.linalg.det(C), 3)
round(np.linalg.det(A), 3) + round(np.linalg.det(B), 3)
```

```
[[-4 -1  2]
 [ 4  2  4]
 [ 8  3  1]]
[[-4 -1  2]
 [-4 -1  2]
 [ 8  3  1]]
[[-4 -1  2]
 [ 8  3  2]
 [ 8  3  1]]
```

Out[57]: 4.0

In [58]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
k = 2
B = A.copy()
B[1, :] = B[1, :] + k * B[0, :]
print(A)
print(B)
round(np.linalg.det(A), 3)
round(np.linalg.det(B), 3)
```

```
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
[[-4 -1  2]
 [ 2  2  3]
 [ 8  3  1]]
```

Out[58]: -14.0

Рисунок 13 – Пример 10

In [59]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
k = 2
A[1, :] = A[0, :] + k * A[2, :]
round(np.linalg.det(A), 3)
```

```
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
```

Out[59]:
```
0.0
```

In [60]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
k = 2
A[1, :] = k * A[0, :]
print(A)
round(np.linalg.det(A), 3)
```

```
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
[[-4 -1  2]
 [-8 -2  4]
 [ 8  3  1]]
```

Out[60]:
```
0.0
```

Обратная матрица

In [61]:
```
A = np.matrix('1 -3; 2 5')
A_inv = np.linalg.inv(A)
print(A_inv)
```

```
[[ 0.45454545  0.27272727]
 [-0.18181818  0.09090909]]
```

In [62]:
```
A = np.matrix('1. -3.; 2. 5.')
A_inv = np.linalg.inv(A)
A_inv_inv = np.linalg.inv(A_inv)
print(A)
print(A_inv_inv)
```

```
[[ 1. -3.]
 [ 2.  5.]]
[[ 1. -3.]
 [ 2.  5.]]
```

In [63]:
```
A = np.matrix('1. -3.; 2. 5.')
L = np.linalg.inv(A.T)
R = (np.linalg.inv(A)).T
print(L)
print(R)
```

```
[[ 0.45454545 -0.18181818]
 [ 0.27272727  0.09090909]]
[[ 0.45454545 -0.18181818]
 [ 0.27272727  0.09090909]]
```

Рисунок 14 – Пример 11

In [64]:
```
A = np.matrix('1. -3.; 2. 5.')
B = np.matrix('7. 6.; 1. 8.')
L = np.linalg.inv(A.dot(B))
R = np.linalg.inv(B).dot(np.linalg.inv(A))
print(L)
print(R)
```

```
[[ 0.09454545  0.03272727]
 [-0.03454545  0.00727273]]
[[ 0.09454545  0.03272727]
 [-0.03454545  0.00727273]]
```

Ранг матрицы

In [65]:
```
m_eye = np.eye(4)
print(m_eye)
rank = np.linalg.matrix_rank(m_eye)
print(rank)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
4
```

In [66]:
```
m_eye[3][3] = 0
print(m_eye)
rank = np.linalg.matrix_rank(m_eye)
print(rank)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 0.]]
3
```

Рисунок 15 – Пример 12

Создать ноутбук, в котором будут приведены собственные примеры на языке Python для каждого из представленных свойств матричных вычислений.

Создать ноутбук, в котором будут приведены собственные примеры на языке Python для каждого из представленных свойств матричных вычислений.

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

Свойство 1. Дважды транспонированная матрица равна исходной матрице:

In [2]:
```
A = np.random.randint(-20, 100, (3, 3))
print(A)
print("Транспонированная матрица: ")
print(A.T)
print("Дважды транспонированная матрица: ")
R = (A.T).T
print(R)
```

```
[[26 35 16]
 [57 29 22]
 [34 -4 -1]]
Транспонированная матрица:
[[26 57 34]
 [35 29 -4]
 [16 22 -1]]
Дважды транспонированная матрица:
[[26 35 16]
 [57 29 22]
 [34 -4 -1]]
```

Свойство 2. Транспонирование суммы матриц равно сумме транспонированных матриц:

In [3]:
```
A = np.random.randint(-20, 100, (3, 3))
B = np.random.randint(-20, 100, (3, 3))
print(f"A\n{A},\nB\n{B}")

print("Транспонирование суммы матриц:")
L = (A + B).T
print(L)

print("Сумма транспонированных матриц:")
R = A.T + B.T
print(R)
```

```
A
[[ 7 76 -4]
 [57 94 92]
 [68 70 -9]],
B
[[ 5 18 66]
 [68 12 85]
 [-2 22 88]]
Транспонирование суммы матриц:
[[ 12 125  66]
 [ 94 106  92]
 [ 62 177  79]]
Сумма транспонированных матриц:
[[ 12 125  66]
 [ 94 106  92]
 [ 62 177  79]]
```

Рисунок 16 – Свойства 1

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

In [4]:
```
A = np.random.randint(0, 10, (3, 3))
B = np.random.randint(0, 10, (3, 3))
print(f"A\n{A},\nB\n{B}")

print("Транспонирование произведения матриц:")
L = (A.dot(B)).T
print(L)

print("Произведение транспонированных матриц:")
R = (B.T).dot(A.T)
print(R)
```

```
A
[[6 6 9]
 [1 5 5]
 [8 5 2]],
B
[[4 9 0]
 [0 3 2]
 [3 4 2]]
Транспонирование произведения матриц:
[[ 51  19  38]
 [108  44  95]
 [ 30  20  14]]
Произведение транспонированных матриц:
[[ 51  19  38]
 [108  44  95]
 [ 30  20  14]]
```

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

In [5]:
```
A = np.random.randint(0, 20, (3, 3))
print(A)
k = 3

print(f"Транспонирование произведения матрицы на число {k}:")
L = (k * A).T
print(L)

print(f"Произведение числа {k} на транспонированную матрицу:")
R = k * (A.T)
print(R)
```

```
[[12  1 16]
 [13 16  3]
 [14  9  8]]
Транспонирование произведения матрицы на число 3:
[[36 39 42]
 [ 3 48 27]
 [48  9 24]]
Произведение числа 3 на транспонированную матрицу:
[[36 39 42]
 [ 3 48 27]
 [48  9 24]]
```

Рисунок 17 – Свойства 2

Свойство 5. Определители исходной и транспонированной матрицы совпадают:

In [6]:
```
A = np.random.randint(0, 10, (3, 3))
print(A)

print("Определитель исходной матрицы:")
A_det = np.linalg.det(A)
print(format(A_det, '.9g'))

print("Определитель транспонированной матрицы:")
A_T_det = np.linalg.det(A.T)
print(format(A_T_det, '.9g'))
```

```
[[2 1 1]
 [7 8 2]
 [9 9 9]]
Определитель исходной матрицы:
54
Определитель транспонированной матрицы:
54
```

Действия над матрицами

Умножение матриц на число

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

In [7]:
```
A = np.random.randint(0, 10, (2, 2))
print(A)

print("Произведение единицы и матрицы:")
L = 1 * A
print(L)
```

```
[[1 9]
 [6 8]]
Произведение единицы и матрицы:
[[1 9]
 [6 8]]
```

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

In [8]:
```
A = np.random.randint(-20, 100, (2, 2))
Z = np.matrix('0 0; 0 0')
L = 0 * A
print("Нулевая матрица:")
print(Z)
print("Произведение матрицы на 0:")
print(L)
```

```
Нулевая матрица:
[[0 0]
 [0 0]]
Произведение матрицы на 0:
[[0 0]
 [0 0]]
```

Рисунок 18 – Свойства 3

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

In [9]:
```
A = np.random.randint(0, 20, (3, 3))
print(A)
p = 2
q = 3
print("Произведение матрицы на сумму чисел:")
L = (p + q) * A
print(L)
print("Произведений матрицы на каждое из этих чисел:")
R = p * A + q * A
print(R)
```

```
[[18 12 17]
 [ 6  7  4]
 [18  4  8]]
Произведение матрицы на сумму чисел:
[[90 60 85]
 [30 35 20]
 [90 20 40]]
Произведений матрицы на каждое из этих чисел:
[[90 60 85]
 [30 35 20]
 [90 20 40]]
```

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

In [10]:
```
A = np.random.randint(0, 10, (3, 3))
print(A)
p = 2
q = 3
print("Произведение матрицы на произведение чисел:")
L = (p * q) * A
print(L)
print("Произведение второго числа и заданной матрицы, умноженное на первое число:")
R = p * (q * A)
print(R)
```

```
[[0 0 0]
 [4 3 1]
 [0 0 0]]
Произведение матрицы на произведение чисел:
[[ 0  0  0]
 [24 18  6]
 [ 0  0  0]]
Произведение второго числа и заданной матрицы, умноженное на первое число:
[[ 0  0  0]
 [24 18  6]
 [ 0  0  0]]
```

Рисунок 19 – Свойства 4

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

In [11]:
```
A = np.random.randint(0, 10, (3, 3))
B = np.random.randint(0, 10, (3, 3))
print(f"A\n{A} \nB\n{B}")
k = 3
print("Произведение суммы матриц на число:")
L = k * (A + B)
print(L)
print("Сумма произведений этих матриц на заданное число:")
R = k * A + k * B
print(R)
```

```
A
[[9 3 2]
 [7 5 3]
 [5 1 3]]
B
[[4 5 2]
 [7 0 1]
 [5 0 1]]
Произведение суммы матриц на число:
[[39 24 12]
 [42 15 12]
 [30  3 12]]
Сумма произведений этих матриц на заданное число:
[[39 24 12]
 [42 15 12]
 [30  3 12]]
```

Сложение матриц

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

In [12]:
```
A = np.random.randint(0, 10, (3, 3))
B = np.random.randint(0, 10, (3, 3))
print(f"A\n{A} \nB\n{B}")

print("A + B")
L = A + B
print(L)
print("B + A")
R = B + A
print(R)
```

```
A
[[6 3 7]
 [5 7 4]
 [8 1 2]]
B
[[7 3 8]
 [2 3 7]
 [2 7 9]]
A + B
[[13  6 15]
 [ 7 10 11]
 [10  8 11]]
B + A
[[13  6 15]
 [ 7 10 11]
 [10  8 11]]
```

Рисунок 20 – Свойства 5

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

In [13]:
```
A = np.random.randint(0, 20, (2, 2))
B = np.random.randint(0, 20, (2, 2))
C = np.random.randint(0, 20, (2, 2))
print(f"A\n{A} \nB\n{B} \nC\n{C}")

print("A + (B + C)")
L = A + (B + C)
print(L)
print("(A + B) + C")
R = (A + B) + C
print(R)
```

```
A
[[15  8]
 [ 9  0]]
B
[[15  4]
 [16 12]]
C
[[18 15]
 [10 17]]
A + (B + C)
[[48 27]
 [35 29]]
(A + B) + C
[[48 27]
 [35 29]]
```

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

In [14]:
```
A = np.random.randint(0, 20, (2, 2))
print(f"A \n{A}")
print(f"-A \n{A * (-1)}")

print(f"Сумма:")
L = A + (-1)*A
print(L)
```

```
A
[[ 8 12]
 [16 13]]
-A
[[ -8 -12]
 [-16 -13]]
Сумма:
[[0 0]
 [0 0]]
```

Рисунок 21 – Свойства 6

Умножение матриц

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

In [15]:
```
A = np.random.randint(0, 10, (2, 2))
B = np.random.randint(0, 10, (2, 2))
C = np.random.randint(0, 10, (2, 2))
print(f"A \n{A},\nB \n{B},\nC \n{C}")

print("A*(B*C)")
L = A.dot(B.dot(C))
print(L)
print("(A*B)*C")
R = (A.dot(B)).dot(C)
print(R)
```

```
A
[[3 3]
 [5 8]],
B
[[8 2]
 [9 0]],
C
[[4 0]
 [1 3]]
A*(B*C)
[[210  18]
 [458  30]]
(A*B)*C
[[210  18]
 [458  30]]
```

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

In [16]:
```
A = np.random.randint(0, 10, (2, 2))
B = np.random.randint(0, 10, (2, 2))
C = np.random.randint(0, 10, (2, 2))
print(f"A \n{A},\nB \n{B},\nC \n{C}")

print("A*(B+C)")
L = A.dot(B + C)
print(L)
print("(A*B + A*C)")
R = A.dot(B) + A.dot(C)
print(R)
```

```
A
[[3 7]
 [9 8]],
B
[[8 8]
 [0 5]],
C
[[5 1]
 [7 2]]
A*(B+C)
[[ 88  76]
 [173 137]]
(A*B + A*C)
[[ 88  76]
 [173 137]]
```

Рисунок 22 – Свойства 7

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

In [17]:
```
A = np.random.randint(0, 10, (2, 2))
B = np.random.randint(0, 10, (2, 2))
print(f"A\n{A} \nB\n{B}")

print("A*B")
L = A.dot(B)
print(L)
print("B*A")
R = B.dot(A)
print(R)
```

```
A
[[0 0]
 [3 6]]
B
[[6 9]
 [5 3]]
A*B
[[ 0  0]
 [48 45]]
B*A
[[27 54]
 [ 9 18]]
```

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

In [18]:
```
A = np.random.randint(0, 10, (3, 3))
E = np.eye(3)
print(f"A \n{A}")

print("E*A")
L = E.dot(A)
print(L)
print("A*E")
R = A.dot(E)
print(R)
```

```
A
[[2 6 5]
 [8 0 0]
 [7 1 5]]
E*A
[[2. 6. 5.]
 [8. 0. 0.]
 [7. 1. 5.]]
A*E
[[2. 6. 5.]
 [8. 0. 0.]
 [7. 1. 5.]]
```

Рисунок 23 – Свойства 8

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

In [19]:
```
A = np.random.randint(0, 10, (3, 3))
Z = np.zeros((3, 3))
print(f"A \n{A}")

print("Z*A")
L = Z.dot(A)
print(L)
print("A*Z")
R = A.dot(Z)
print(R)
```

```
A
[[0 8 8]
 [5 1 8]
 [6 8 4]]
Z*A
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
A*Z
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Определитель матрицы

Свойство 1. Определитель матрицы остается неизменным при ее транспонировании:

In [20]:
```
A = np.random.randint(0, 10, (3, 3))
print(f"A \n{A}")

det_A = round(np.linalg.det(A), 3)
print(f"△A = {det_A}")

print(f"A(T) \n{A.T}")
det_A_t = round(np.linalg.det(A.T), 3)
print(f"△A(T) = {det_A_t}")
```

```
A
[[0 3 2]
 [8 6 6]
 [8 3 2]]
△A = 48.0
A(T)
[[0 8 8]
 [3 6 3]
 [2 6 2]]
△A(T) = 48.0
```

Рисунок 24 – Свойства 9

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

In [21]:
```
A = np.random.randint(0, 10, (3, 3))
print(f"A \n{A}")
det_A = np.linalg.det(A)
print(f"△A = {det_A}")

A = np.matrix('0 5 2; 0 6 2; 0 9 4')
print(f"A \n{A}")
det_A = np.linalg.det(A)
print(f"△A = {det_A}")
```

```
A
[[0 1 6]
 [4 5 5]
 [9 8 0]]
△A = -33.00000000000003
A
[[0 5 2]
 [0 6 2]
 [0 9 4]]
△A = 0.0
```

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

In [22]:
```
A = np.matrix('2 6 3; 5 2 8; 1 2 3')
print(f"A \n{A}")
det_A = round(np.linalg.det(A), 3)
print(f"|A| = {det_A}")

B = np.matrix('2 6 3; 1 2 3; 5 2 8')
print(f"B \n{B}")
det_B = round(np.linalg.det(B), 3)
print(f"△B = {det_B}")
```

```
A
[[2 6 3]
 [5 2 8]
 [1 2 3]]
|A| = -38.0
B
[[2 6 3]
 [1 2 3]
 [5 2 8]]
△B = 38.0
```

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

In [23]:
```
A = np.matrix('2 6 3; 5 2 8; 1 2 3')
print(f"A \n{A}")
det_A = round(np.linalg.det(A), 3)
print(f"△A = {det_A}")

A = np.matrix('2 6 3; 1 2 3; 1 2 3')
print(f"A \n{A}")
det_A = round(np.linalg.det(A), 3)
print(f"△A = {det_A}\n")
```

```
A
[[2 6 3]
 [5 2 8]
 [1 2 3]]
△A = -38.0
A
[[2 6 3]
 [1 2 3]
 [1 2 3]]
△A = 0.0
```

Рисунок 25 – Свойства 10

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

In [24]:
```
A = np.random.randint(-15, 90, (3, 3))
print(f"A \n{A}")
k = 2
B = A.copy()
B[2, :] = k * B[2, :]
print(f"B \n{B}")
det_A = round(np.linalg.det(A), 3)
det_B = round(np.linalg.det(B), 3)
print(f"△A * k = {det_A * k}")
print(f"△B = {det_B}")
```

```
A
[[-8 53 66]
 [ 4 16 17]
 [-7 31 28]]
B
[[ -8  53  66]
 [  4  16  17]
 [-14  62  56]]
△A * k = 7930.0
△B = 7930.0
```

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

In [25]:
```
A = np.matrix('7 4 3; 7 4 3; 2 7 1')
B = np.matrix('7 4 3; 1 8 -4; 2 7 1')
C = A.copy()
C[1, :] += B[1, :]
print(f"C \n{C}")
print(f"A \n{A}")
print(f"B \n{B}")

print("△C")
print(round(np.linalg.det(C), 3))
print("△A + △B")
print(round(np.linalg.det(A), 3) + round(np.linalg.det(B), 3))
```

```
C
[[ 7  4  3]
 [ 8 12 -1]
 [ 2  7  1]]
A
[[7 4 3]
 [7 4 3]
 [2 7 1]]
B
[[ 7  4  3]
 [ 1  8 -4]
 [ 2  7  1]]
△C
189.0
△A + △B
189.0
```

Рисунок 26 – Свойства 11

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

In [26]:
```
A = np.random.randint(1, 10, (3, 3))
k = 2
B = A.copy()
B[1, :] = B[1, :] + k * B[0, :]
print(f"A\n{A} \nB\n{B}")


print("△A")
print(round(np.linalg.det(A), 3))
print("△B")
print(round(np.linalg.det(B), 3))
```

```
A
[[8 3 7]
 [5 7 7]
 [7 8 4]]
B
[[ 8  3  7]
 [21 13 21]
 [ 7  8  4]]
△A
-200.0
△B
-200.0
```

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

In [27]:
```
A = np.matrix('6 2 4; 2 1 9; 4 3 2')
print(f"A \n{A}")
k = 2

A[1, :] = A[0, :] + k * A[2, :]
print("△A")
print(round(np.linalg.det(A), 3))
```

```
A
[[6 2 4]
 [2 1 9]
 [4 3 2]]
△A
0.0
```

Свойство 9. Если матрица содержит пропорциональные строки, то ее определитель равен нулю:

In [28]:
```
A = np.matrix('6 2 4; 2 1 9; 4 3 2')
print(f"A \n{A}")
print(round(np.linalg.det(A), 3))

k = 4
A[1, :] = k * A[0, :]
print(A)
print("△A")
print(round(np.linalg.det(A), 3))
```

```
A
[[6 2 4]
 [2 1 9]
 [4 3 2]]
-78.0
[[ 6  2  4]
 [24  8 16]
 [ 4  3  2]]
△A
0.0
```

Рисунок 27 – Свойства 12

Обратная матрица

Свойство 1. Обратная матрица обратной матрицы есть исходная матрица:

In [29]:
```
A = np.random.uniform(0, 10, (2, 2))
print(A)
A_inv = np.linalg.inv(A)
A_inv_inv = np.linalg.inv(A_inv)
print(A_inv_inv)
```

```
[[0.67122576 5.44947895]
 [9.83382116 5.56579284]]
[[0.67122576 5.44947895]
 [9.83382116 5.56579284]]
```

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

In [30]:
```
A = np.random.uniform(0, 10, (2, 2))
print("Обратная матрица транспонированной матрицы:")
L = np.linalg.inv(A.T)
print(L)

print("Транспонированная матрица от обратной матрицы:")
R = (np.linalg.inv(A)).T
print(R)
```

```
Обратная матрица транспонированной матрицы:
[[ 0.34985134 -0.25989548]
 [-0.20497718  0.32384085]]
Транспонированная матрица от обратной матрицы:
[[ 0.34985134 -0.25989548]
 [-0.20497718  0.32384085]]
```

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

In [31]:
```
A = np.random.uniform(0, 10, (2, 2))
B = np.random.uniform(0, 10, (2, 2))

print("Обратная матрица произведения матриц:")
L = np.linalg.inv(A.dot(B))
print(L)

print("Произведение обратных матриц:")
R = np.linalg.inv(B).dot(np.linalg.inv(A))
print(R)
```

```
Обратная матрица произведения матриц:
[[ 0.06213741 -0.02910627]
 [-0.08959203  0.06849657]]
Произведение обратных матриц:
[[ 0.06213741 -0.02910627]
 [-0.08959203  0.06849657]]
```

Рисунок 28 – Свойства 13

Создать ноутбук, в котором будут приведены собственные примеры решения систем линейных уравнений матричным методом и методом Крамера.

> Создать ноутбук, в котором будут приведены собственные примеры решения систем линейных уравнений матричным методом и методом Крамера.

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

Метод решения СЛАУ матричным методом

Записываем уравнение в виде $AX = B$. Выражаем X: $X = A^{-1} \times B$. Если определитель матрицы A больше 0, то находим обратную матрицу. Затем умножаем обратную матрицу A на матрицу свободных членов B, получаем ответ.

$$X = A^{-1} \cdot B$$

Пусть задана система уравнений:

$$\begin{cases} 2x_1 - 4x_2 + 3x_3 = 1 \\ 2x_1 - 2x_2 + 4x_3 = 3 \\ 3x_1 - x_2 + 5x_3 = 2 \end{cases}$$

Приведем систему к матричному виду:

$$\left(\begin{array}{ccc|c} 2 & -4 & 3 & 1 \\ 2 & -2 & 4 & 3 \\ 3 & -1 & 5 & 2 \end{array}\right)$$

Найдем определитель $\triangle A$, обратную матрицу $A^{-1} = E/A$ и присоединенную матрицу A^*.

In [2]:
```
A = np.array([[2., 2., 3.], [-4., -2., -1], [3, 4, 5]])
B = np.array([1., 3., 2.])

# Определитель
det_A = np.linalg.det(A)

if det_A != 0:
    # Обратная матрица
    inv_A = np.linalg.inv(A)
    # Присоединенная матрица, решения
    result = inv_A.dot(B)
    print(result)
else:
    print("Матричный метод не подходит для решения этой системы!")
```

```
[-1.  0.  1.]
```

Рисунок 29 – Решение СЛАУ матричным методом

Метод решения СЛАУ методом Крамера

Теорема Крамера: Если определитель матрицы $\triangle$ квадратной системы не равен нулю, то система совместна и имеет единственное решение, которое находится по формулам Крамера: $x_1 = \frac{\triangle_1}{\triangle}; x_2 = \frac{\triangle_2}{\triangle}; x_3 = \frac{\triangle_3}{\triangle}$, где $\triangle_i$ - определитель матрицы системы, где вместо i-го столбца стоит столбец правых частей.

В качестве примера возьмем СЛАУ, решенную выше матричным методом:

$$\begin{cases} 2x_1 - 4x_2 + 3x_3 = 1 \\ 2x_1 - 2x_2 + 4x_3 = 3 \\ 3x_1 - x_2 + 5x_3 = 2 \end{cases}$$

Приведем систему к матричному виду:

$$\left(\begin{array}{ccc|c} 2 & -4 & 3 & 1 \\ 2 & -2 & 4 & 3 \\ 3 & -1 & 5 & 2 \end{array}\right)$$

Найдем определитель $\triangle A$, побочные определители для каждого столбца.

n [3]:
```
A = np.array([[2., 2., 3.], [-4., -2., -1], [3, 4, 5]])
B = np.array([1., 3., 2.])

# считаем главный определитель
det_A = np.linalg.det(A)

if (det_A != 0):
    # Побочные определители
    kram_dets = []
    for i in range(len(A)):
        copied_A = np.array(A)
        copied_A[:, i] = B
        kram_dets.append(np.linalg.det(copied_A))
    # Решения
    for det_i in kram_dets:
        print(float(det_i) / det_A)
else:
    print("Матричный метод не подходит для решения этой системы!")
```

```
-1.0000000000000004
-0.0
1.0
```

Результаты решений СЛАУ обоими способами совпали.

Рисунок 30 – Решение СЛАУ методом Крамера

Вывод: В результате выполнения работы были исследованы методы работы с матрицами и векторами с помощью библиотеки NumPy.

Контрольные вопросы:

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

Вектор

Вектором называется матрица, у которой есть только один столбец или одна строка.

Вектор-строка

Вектор-строка имеет следующую математическую запись.

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

Вектор-столбец

Вектор-столбец имеет следующую математическую запись.

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

Квадратная матрица

Квадратной называется матрица, у которой количество столбцов и строк совпадает.

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

m_sqr_mx = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

m_sqr_mx = np.matrix('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 = [[1, 0, 0], [0, 5, 0], [0, 0, 9]]

m_diag_np = np.matrix(m_diag)

diag = np.diag(m_sqr_mx)

Единичная матрица

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

```
m_e = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
m_e_np = np.matrix(m_e)
m_eye = np.eye(3)
```

В качестве аргумента функции передается размерность матрицы, в нашем примере – это матрица 3 3. Тот же результат можно получить с помощью функции identity().

```
m_idnt = np.identity(3)
```

Нулевая матрица

У нулевой матрицы все элементы равны нулю.

```
m_zeros = np.zeros((3, 3))
```

2. Как выполняется транспонирование матриц?

Транспонирование матрицы – это процесс замены строк матрицы на ее столбцы, а столбцов соответственно на строки. Полученная в результате матрица называется транспонированной. Символ операции транспонирования – буква Т.

3. Приведите свойства операции транспонирования матриц.

Свойство 1. Дважды транспонированная матрица равна исходной матрице.

Свойство 2. Транспонирование суммы матриц равно сумме транспонированных матриц.

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

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

Свойство 5. Определители исходной и транспонированной матрицы совпадают.

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

Функция transpose() или A.T.

5. Какие существуют основные действия над матрицами?

Умножение матрицы на число

Сложение матриц

Умножение матриц

Определитель матрицы

Обратная матрица

Ранг матрицы

6. Как осуществляется умножение матрицы на число?

При умножении матрицы на число, все элементы матрицы умножаются на это число.

7. Какие свойства операции умножения матрицы на число?

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

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

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

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

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

8. Как осуществляется операции сложения и вычитания матриц?

$$A = \begin{pmatrix} 1 & 6 & 3 \\ 8 & 2 & 7 \end{pmatrix}, B = \begin{pmatrix} 8 & 1 & 5 \\ 6 & 9 & 12 \end{pmatrix},$$

$$C = A + B,$$

$$C = \begin{pmatrix} 1+8 & 6+1 & 3+5 \\ 8+6 & 2+9 & 7+12 \end{pmatrix} = \begin{pmatrix} 9 & 7 & 8 \\ 14 & 11 & 19 \end{pmatrix}.$$

9. Каковы свойства операций сложения и вычитания матриц?

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

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

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

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

np.add, np.subtract или знаки +, -.

11. Как осуществляется операция умножения матриц?

Каждый элемент c_{ij} новой матрицы является суммой произведений элементов i-ой строки первой матрицы и j-го столбца второй матрицы.

12. Каковы свойства операции умножения матриц?

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

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

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

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

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

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

np.dot(), np.multiply().

14. Что такое определитель матрицы? Каковы свойства определителя 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.

Свойство 1. Определитель матрицы остается неизменным при ее транспонировании.

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

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

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

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

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

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

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

Свойство 9. Если матрица содержит пропорциональные строки, то ее определитель равен нулю.

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

Функция det() из пакета linalg.

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

Обратной матрицей матрицы 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 строится на базе исходной A путем замены всех элементов матрицы A на их алгебраические дополнения.

Транспонируя союзную, мы получим так называемую присоединенную матрицу. $A^{-1} = \frac{1}{det(A)} \times A^{*T}.$

17. Каковы свойства обратной матрицы?

Свойство 1. Обратная матрица обратной матрицы есть исходная матрица

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

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

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

Функция inv().

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

Алгоритм и решение приведено в лабораторной работе.

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

Алгоритм и решение приведено в лабораторной работе.