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 «Анализ данных»

Выполнил студент группы ИВТ-б-о-21-1

Лысенко И.А. « »_____20__г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2023

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

Ход работы:

1. Создал репозиторий на GitHub: https://github.com/IsSveshuD/lab_3.3.git .

2. Проработал примеры:

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

```
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)
```

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

```
In [6]: v_hor_one_v2 = np.ones((1, 5))
        print(v_hor_one_v2)
```

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

```
In [7]: v_vert_np = np.array([[1], [2]])
        print(v_vert_np)
```

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

```
In [8]: v_vert_zeros = np.zeros((5, 1))
        print(v_vert_zeros)
```

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

In [9]:
```
v_vert_ones = np.ones((5, 1))
print(v_vert_ones)
```

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

In [10]:
```
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 [11]:
```
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 [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]]
```

In [13]:
```
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 [14]:
```
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 [15]:
```
m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
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]]
```

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 [20]:
```
m_eye = np.eye(3)
print(m_eye)
```

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

In [21]:
```
m_idnt = np.identity(3)
print(m_idnt)
```

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

In [22]:
```
m_zeros = np.zeros((3, 3))
print(m_zeros)
```

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

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

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

In [24]:
```
m_var = np.zeros((2, 5))
print(m_var)
```

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

In [25]:
```
A = np.matrix('1 2 3; 4 5 6')
print(A)
```

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

In [26]:
```
A_t = A.transpose()
print(A_t)
```

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

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

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

In [28]:
```
R = (A.T).T
print(R)
```

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

In [29]:
```
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 [30]:
```
print(R)
```

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

In [31]:
```
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]]
```

In [32]:
```
print(R)
```

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

In [33]:
```
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]]
```

In [34]:
```
print(R)
```

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

In [36]:
```
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
```

In [37]:
```
print(format(A_T_det, '.9g'))
```

```
-2
```

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

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

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

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

In [5]:
```
print(R)
```

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

In [2]:
```
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]]
```

In [3]:
```
print(R)
```

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

In [4]:
```
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]]
```

In [5]:
```
print(R)
```

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

In [6]:
```
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]]
```

In [7]:
```
print(R)
```

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

In [8]:
```
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 [9]:
```
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]]
```

In [10]:
```
print(R)
```

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

In [11]:
```
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]]
```

In [12]:
```
print(R)
```

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

In [13]:
```
A = np.matrix('1 2; 3 4')
Z = np.matrix('0 0; 0 0')
L = A + (-1)*A
print(L)
```

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

In [14]:
```
print(Z)
```

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

In [15]:
```
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 [16]:
```
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]]
```

In [17]:
```
print(R)
```

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

In [18]:
```
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]]
```

In [19]:
```
print(R)
```

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

In [20]:
```
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]]
```

In [21]:
```
print(R)
```

```
[[23 34]
 [31 46]]
```

In [22]:
```
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]]
```

In [23]:
```
print(R)
```

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

In [24]:
```
print(A)
```

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

In [25]:
```
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]]
```

In [26]:
```
print(R)
```

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

In [27]:
```
print(Z)
```

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

In [28]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
```

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

In [29]:
```
np.linalg.det(A)
```

Out[29]:
```
-14.000000000000009
```

In [30]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
```

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

In [31]:
```
print(A.T)
```

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

In [32]:
```
det_A = round(np.linalg.det(A), 3)
det_A_t = round(np.linalg.det(A.T), 3)
print(det_A)
```

```
-14.0
```

In [33]:
```
print(det_A_t)
```

```
-14.0
```

In [34]:
```
A = np.matrix('-4 -1 2; 0 0 0; 8 3 1')
print(A)
```

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

In [35]:
```
np.linalg.det(A)
```

Out[35]:
```
0.0
```

In [36]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
```

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

In [37]:
```
B = np.matrix('10 4 -1; -4 -1 2; 8 3 1')
print(B)
```

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

In [38]:
```
round(np.linalg.det(A), 3)
```

Out[38]:
```
-14.0
```

In [39]:
```
round(np.linalg.det(B), 3)
```

Out[39]:
```
14.0
```

In [40]:
```
A = np.matrix('-4 -1 2; -4 -1 2; 8 3 1')
print(A)
```

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

In [41]:
```
np.linalg.det(A)
```

Out[41]:
```
0.0
```

In [42]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
```

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

In [43]:
```
k = 2
B = A.copy()
B[2, :] = k * B[2, :]
print(B)
```

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

In [44]:
```
det_A = round(np.linalg.det(A), 3)
det_B = round(np.linalg.det(B), 3)
det_A * k
```

Out[44]: -28.0

In [45]:
```
det_B
```

Out[45]: -28.0

In [46]:
```
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)
```

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

In [47]:
```
print(A)
```

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

In [48]:
```
print(B)
```

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

```
In [49]: round(np.linalg.det(C), 3)

Out[49]: 4.0

In [50]: round(np.linalg.det(A), 3) + round(np.linalg.det(B), 3)

Out[50]: 4.0

In [51]: 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)

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

In [52]: print(B)

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

In [53]: round(np.linalg.det(A), 3)

Out[53]: -14.0

In [54]: round(np.linalg.det(B), 3)
```

Рисунок 1 – Примеры

3. Выполнил индивидуальное задание 1.

Свойства транспонированных матриц

Свойство 1. Дважды транспонированная матрица равна исходной матрице:

In [10]:
```
A = np.matrix('3 1 7; 8 9 10')
print(" Изначальная матрица\n",A)
R = (A.T).T
print("\nДважды транспортированная матрица\n", R)
```

```
 Изначальная матрица
 [[ 3  1  7]
 [ 8  9 10]]

Дважды транспортированная матрица
 [[ 3  1  7]
 [ 8  9 10]]
```

Свойство 2. Транспонирование суммы матриц равно сумме транспонированных матриц:

In [11]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
B = np.matrix('4 8 7; 0 7 5; 3 3 6')
L = (A + B).T
R = A.T + B.T
print("Транспортированная сумма матриц\n", L)
print("\nСумма транспортированных матриц\n", R)
```

```
Транспортированная сумма матриц
 [[ 7  4  4]
 [10 12  5]
 [ 8 11  9]]

Сумма транспортированных матриц
 [[ 7  4  4]
 [10 12  5]
 [ 8 11  9]]
```

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

In [12]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
B = np.matrix('4 8 7; 0 7 5; 3 3 6')
L = (A.dot(B)).T
R = (B.T).dot(A.T)
print("Транспортирование произведения матриц\n", L)
print("\nПроизведение транспортированных матриц в обратном порядке\n", R)
```

```
Транспортирование произведения матриц
 [[15 34 13]
 [41 85 31]
 [37 89 35]]

Произведение транспортированных матриц в обратном порядке
 [[15 34 13]
 [41 85 31]
 [37 89 35]]
```

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

In [13]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
k = 5
L = (k * A).T
R = k * (A.T)
print("Транспортирование произведения матрицы на число\n", L)
print("\nПроизведение числа на транспортированную матрицу\n", R)
```

```
Транспортирование произведения матрицы на число
 [[15 20  5]
 [10 25 10]
 [ 5 30 15]]

Произведение числа на транспортированную матрицу
 [[15 20  5]
 [10 25 10]
 [ 5 30 15]]
```

Свойство 5. Определители исходной и транспонированной матрицы совпадают:

In [29]:
```
B = np.matrix('3 2 1; 4 5 1; 2 3 4')
B_det = np.linalg.det(B)
B_T_det = np.linalg.det(B.T)
print("Определитель исходной матрицы:", format(B_det, '.9g'))
print("\nОпределитель транспортированной матрицы:", format(B_T_det, '.9g'))
```

```
Определитель исходной матрицы: 25

Определитель транспортированной матрицы: 25
```

Свойства операции умножения матрицы

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

In [31]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
L = 1 * A
R = A
print("Исходная матрица\n", A)
print("\nПроизведение матрицы на единицу\n", L)
```

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

Произведение матрицы на единицу
 [[3 2 1]
 [4 5 6]
 [1 2 3]]
```

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

In [32]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
Z = np.matrix('0 0 0; 0 0 0; 0 0 0')
L = 0 * A
print("Нулевая равная размерностью исходной матрица\n", Z)
print("\nПроизведение матрицы на ноль\n", L)
```

```
Нулевая равная размерностью исходной матрица
 [[0 0 0]
 [0 0 0]
 [0 0 0]]

Произведение матрицы на ноль
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
```

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

In [34]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
p = 5
q = 4
L = (p + q) * A
R = p * A + q * A
print("Произведение матрицы на сумму чисел\n", L)
print("\nСумма произведний матрицы на числа\n", R)
```

```
Произведение матрицы на сумму чисел
 [[27 18  9]
 [36 45 54]
 [ 9 18 27]]

Сумма произведний матрицы на числа
 [[27 18  9]
 [36 45 54]
 [ 9 18 27]]
```

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

In [35]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
p = 5
q = 4
L = (p * q) * A
R = p * (q * A)
print("Произведение матрицы на произведение чисел\n", L)
print("\nПроизведение второго числа и заданной матрицы, умноженное на первое число\n", R)
```

```
Произведение матрицы на произведение чисел
 [[ 60  40  20]
 [ 80 100 120]
 [ 20  40  60]]

Произведение второго числа и заданной матрицы, умноженное на первое число
 [[ 60  40  20]
 [ 80 100 120]
 [ 20  40  60]]
```

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

In [36]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
B = np.matrix('4 8 7; 0 7 5; 3 3 6')
k = 4
L = k * (A + B)
R = k * A + k * B
print("Произведение суммы матриц на число\n", L)
print("\nСумма произведений матриц на число\n", R)
```

```
Произведение суммы матриц на число
 [[28 40 32]
 [16 48 44]
 [16 20 36]]

Сумма произведений матриц на число
 [[28 40 32]
 [16 48 44]
 [16 20 36]]
```

Свойства сложения матриц ¶

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

In [37]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
B = np.matrix('4 8 7; 0 7 5; 3 3 6')
L = A + B
R = B + A
print("Сумма матриц A + B\n", L)
print("\nСумма матриц B + A\n", R)
```

```
Сумма матриц A + B
 [[ 7 10  8]
 [ 4 12 11]
 [ 4  5  9]]

Сумма матриц B + A
 [[ 7 10  8]
 [ 4 12 11]
 [ 4  5  9]]
```

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

In [2]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
B = np.matrix('4 8 7; 0 7 5; 3 3 6')
C = np.matrix('5 4 3; 3 5 7; 9 2 5')
L = A + (B + C)
R = (A + B) + C
print("Сумма матриц A + (B + C)\n", L)
print("\nСумма матриц (A + B) + C\n", R)
```

```
Сумма матриц A + (B + C)
 [[12 14 11]
 [ 7 17 18]
 [13  7 14]]

Сумма матриц (A + B) + C
 [[12 14 11]
 [ 7 17 18]
 [13  7 14]]
```

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

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

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

Свойства произведения матриц

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

In [4]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
B = np.matrix('4 8 7; 0 7 5; 3 3 6')
C = np.matrix('5 4 3; 3 5 7; 9 2 5')
L = A.dot(B.dot(C))
R = (A.dot(B)).dot(C)
print("Произведение матриц A * (B * C)\n", L)
print("\nПроизведение матриц (A * B) * C\n", R)
```

```
Произведение матриц A * (B * C)
 [[ 531  339  517]
 [1226  739 1142]
 [ 473  277  431]]

Произведение матриц (A * B) * C
 [[ 531  339  517]
 [1226  739 1142]
 [ 473  277  431]]
```

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

In [5]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
B = np.matrix('4 8 7; 0 7 5; 3 3 6')
C = np.matrix('5 4 3; 3 5 7; 9 2 5')
L = A.dot(B + C)
R = A.dot(B) + A.dot(C)
print("Произведение матрицы на сумму матриц\n", L)
print("\nСумма произведений матриц\n", R)
```

```
Произведение матрицы на сумму матриц
 [[ 45  65  65]
 [123 138 166]
 [ 51  51  67]]

Сумма произведений матриц
 [[ 45  65  65]
 [123 138 166]
 [ 51  51  67]]
```

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

In [6]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
B = np.matrix('4 8 7; 0 7 5; 3 3 6')
L = A.dot(B)
R = B.dot(A)
print("Произведение матриц A * B\n", L)
print("\nПроизведение матриц B * A\n", R)
```

```
Произведение матриц A * B
 [[15 41 37]
 [34 85 89]
 [13 31 35]]

Произведение матриц B * A
 [[51 62 73]
 [33 45 57]
 [27 33 39]]
```

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

In [7]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
E = np.matrix('1 0 0; 0 1 0; 0 0 1')
L = E.dot(A)
R = A.dot(E)
print("Исходная матрица\n", L)
print("\nМатрица, умноженная на единичную матрицу\n", R)
```

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

Матрица, умноженная на единичную матрицу
 [[3 2 1]
 [4 5 6]
 [1 2 3]]
```

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

In [9]:
```
A = np.matrix('3 2 1; 4 5 6; 1 2 3')
E = np.matrix('0 0 0; 0 0 0; 0 0 0')
R = A.dot(E)
print("Исходная матрица\n", A)
print("\nМатрица, умноженная на нулевую матрицу\n", R)
```

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

Матрица, умноженная на нулевую матрицу
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
```

Свойства определителя матрицы

Свойство 1. Определитель матрицы остается неизменным при ее транспонировании:

In [10]:
```
A = np.matrix('3 -2 1; 4 5 -6; 1 -2 3')
det_A = round(np.linalg.det(A), 3)
det_A_t = round(np.linalg.det(A.T), 3)
print("Определитель исходной матрицы\n", det_A)
print("\nОпределитель транспортированной матрицы\n", det_A_t)
```

```
Определитель исходной матрицы
 32.0

Определитель транспортированной матрицы
 32.0
```

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

In [11]:
```
A = np.matrix('3 -2 1; 4 5 -6; 0 0 0')
det_A = round(np.linalg.det(A), 3)
print("Исходная матрица\n", A)
print("\nОпределитель матрицы\n", det_A)
```

```
Исходная матрица
 [[ 3 -2  1]
 [ 4  5 -6]
 [ 0  0  0]]

Определитель матрицы
 0.0
```

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

In [12]:
```
A = np.matrix('3 -2 1; 4 5 -6; 1 -2 3')
B = np.matrix('4 5 -6; 3 -2 1; 1 -2 3')
det_A = round(np.linalg.det(A), 3)
det_B = round(np.linalg.det(B), 3)
print("Исходная матрица\n", A)
print("\nОпределитель матрицы\n", det_A)
print("Исходная матрица с переставленными строками\n", B)
print("\nОпределитель матрицы\n", det_B)
```

```
Исходная матрица
 [[ 3 -2  1]
 [ 4  5 -6]
 [ 1 -2  3]]

Определитель матрицы
 32.0
Исходная матрица с переставленными строками
 [[ 4  5 -6]
 [ 3 -2  1]
 [ 1 -2  3]]

Определитель матрицы
 -32.0
```

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

In [13]:
```
A = np.matrix('3 -2 1; 3 -2 1; 1 -2 3')
det_A = round(np.linalg.det(A), 3)
print("Исходная матрица\n", A)
print("\nОпределитель матрицы\n", det_A)
```

```
Исходная матрица
 [[ 3 -2  1]
 [ 3 -2  1]
 [ 1 -2  3]]

Определитель матрицы
 0.0
```

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

In [14]:
```
A = np.matrix('3 -2 1; 4 5 -6; 1 -2 3')
k = 4
B = A.copy()
B[1, :] = k * B[1, :]
det_A = round(np.linalg.det(A), 3)
det_B = round(np.linalg.det(B), 3)
print("Определитель матрицы умноженной на число\n", det_A * k)
print("\nОпределитель матрицы, где все элементы строки умножены на k\n", det_B)
```

```
Определитель матрицы умноженной на число
 128.0

Определитель матрицы, где все элементы строки умножены на k
 128.0
```

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 [35]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
B = np.matrix('-4 -1 2; 8 3 2; 8 3 1')
C = A.copy()
C[1, :] += B[1, :]
round(np.linalg.det(C), 3)
round(np.linalg.det(A), 3) + round(np.linalg.det(B), 3)
print("Определитель матрицы из элементов суммы двух слагаемых\n", round(np.linalg.det(C), 3))
print("\nСумма определителей двух соответствующих матриц\n", round(np.linalg.det(A), 3) + round(np.linalg.det(B), 3))
```

```
Определитель матрицы из элементов суммы двух слагаемых
 -10.0

Сумма определителей двух соответствующих матриц
 -10.0
```

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

In [49]:
```
A = np.matrix('3 -2 1; 4 5 -6; 1 -2 3')
k = 4
B = A.copy()
B[1, :] = B[1, :] + k * B[0, :]
print("Определитель исходной матрица\n", round(np.linalg.det(A), 3))
print("\nОпределитель матрицы, у которой к элементам одной строки прибавлены элементы другой строки, \
умноженные на одно и тоже число\n", round(np.linalg.det(B), 3))
```

```
Определитель исходной матрица
 32.0

Определитель матрицы, у которой к элементам одной строки прибавлены элементы другой строки, умноженные на одно и тоже число
 32.0
```

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

In [50]:
```
A = np.matrix('3 -2 1; 4 5 -6; 1 -2 3')
k = 2
A[1, :] = A[0, :] + k * A[2, :]

print("Определитель полученной матрицы", round(np.linalg.det(A), 3))
```

```
Определитель полученной матрицы -0.0
```

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

In [52]:
```
k = 2
A[1, :] = k * A[0, :]
print("Определитель, полученной матрицы\n",round(np.linalg.det(A), 3))
```

```
Определитель, полученной матрицы
 0.0
```

Обратная матрица

Свойство 1. Обратная матрица обратной матрицы есть исходная матрица:

In [54]:
```
A = np.matrix('2. -1.; 4. 5.')
A_inv = np.linalg.inv(A)
A_inv_inv = np.linalg.inv(A_inv)
print("Исходная матрица\n", A)
print("\nОбратная матрица от обратной матрицы\n", A_inv_inv)
```

```
Исходная матрица
 [[ 2. -1.]
 [ 4.  5.]]

Обратная матрица от обратной матрицы
 [[ 2. -1.]
 [ 4.  5.]]
```

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

In [55]:
```
A = np.matrix('2. -1.; 4. 5.')
L = np.linalg.inv(A.T)
R = (np.linalg.inv(A)).T
print("Обратная матрица транспонированной матрицы\n", L)
print("\nТранспонированная матрица от обратной матрицы\n", R)
```

```
Обратная матрица транспонированной матрицы
 [[ 0.35714286 -0.28571429]
 [ 0.07142857  0.14285714]]

Транспонированная матрица от обратной матрицы
 [[ 0.35714286 -0.28571429]
 [ 0.07142857  0.14285714]]
```

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

In [56]:
```
A = np.matrix('2. -1.; 4. 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("Обратная матрица произведения матриц\n", L)
print("\nПроизведение обратных матриц\n", R)
```

```
Обратная матрица произведения матриц
 [[ 0.09142857 -0.00571429]
 [-0.04714286  0.01857143]]

Произведение обратных матриц
 [[ 0.09142857 -0.00571429]
 [-0.04714286  0.01857143]]
```

Рисунок 2 – Индивидуальное задание

4. Выполнил индивидуальное задание 2:

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

Метод Крамера

In [5]:
```
import numpy as np
```

Создадим и заполним матрицу рандомными значениями от 0 до 10

In [6]:
```
matrix = np.random.randint (0, 10, (4, 4))
print(matrix)
```

```
[[0 8 5 9]
 [3 6 5 9]
 [9 2 9 5]
 [9 9 4 8]]
```

Найдём определитель

In [11]:
```
opred_1 = np.linalg.det(matrix)
print(opred_1)
```

```
1368.000000000001
```

Заполним рандомными значенями столбец свободных членов

In [13]:
```
sv = np.random.randint (0, 10, (4, 1))
print(sv)
x = np.ones((4, 1))
```

```
[[4]
 [5]
 [9]
 [5]]
```

Решим матрицу методом Крамера. Скопируем значения в дополнительную матрицу и подставим в неё столбец свободных членов. Затем найдём определитель

In [23]:
```
t = [0, 1, 2, 3]
if opred_1 != 0:
    for i in t:
        mat_dop = matrix.copy()
        mat_dop[:, i] = sv[:, 0]
        x[i,0] = np.linalg.det(mat_dop) / opred_1
    print(x)
else:
    print("Невозможно решить")
```

```
[[-0.37214363]
 [ 1.40152339]
 [ 1.95429815]
 [ 0.30903156]]
```

Матричный метод:

Пусть дана система n линейных уравнений с n переменными $(n \times n)$.
Если основная матрица не вырождена, т.е. $|A| \neq 0$, тогда для матрицы A существует A^{-1}. Умножив матричное уравнение на A^{-1} получим:

$$X = A^{-1} * B$$

Создадим и заполним матрицу:

In [24]:
```
matrix = np.random.randint (0, 10, (4, 4))
print(matrix)
```

```
[[7 6 9 1]
 [0 8 6 5]
 [6 0 6 7]
 [6 4 5 8]]
```

Найдём обратную матрицу

In [25]:
```
mat_inv = np.linalg.inv(matrix)
print(mat_inv)
```

```
[[ 0.07933194 -0.15970772 -0.1263048   0.20041754]
 [ 0.03444676  0.01617954 -0.20615866  0.16597077]
 [ 0.03757829  0.1085595   0.22964509 -0.27348643]
 [-0.10020877  0.04384134  0.05427975  0.06263048]]
```

Найдём определитель

In [26]:
```
opred_1 = np.linalg.det(matrix)
print(opred_1)
```

```
1915.9999999999998
```

Заполним столбец свободных членов

In [27]:
```
sv = np.random.randint (0, 20, (4, 1))
print(sv)
```

```
[[ 6]
 [ 1]
 [17]
 [ 4]]
```

Найдём решения СЛАУ, умножив нашу обратную матрицу на столбец свободных членов:

In [28]:
```
if opred_1 != 0:
    x = mat_inv.dot(sv)
    print(x)
else:
    print("Невозможно решить")
```

```
[[-1.02922756]
 [-2.61795407]
 [ 3.1440501 ]
 [ 0.61586639]]
```

Рисунок 3 – Индивидуальное задание 2.

Ответы на вопросы:

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

Вектор-столбец:

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

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

print(v_vert_np)

[[1]

[2]]

Нулевой вектор-столбец.

v_vert_zeros = np.zeros((5, 1))

print(v_vert_zeros)

[[0.]

[0.]

[0.]

[0.]

[0.]]

Единичный вектор-столбец.

v_vert_ones = np.ones((5, 1))

print(v_vert_ones)

[[1.]

[1.]

[1.]

[1.]

[1.]]

Квадратная матрица:

Довольно часто, на практике, приходится работать с квадратными мат рицами. Квадратной называется матрица, у которой количество столбцов и строк совпадает.

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]]

Но в Numpy есть еще одни способ создания матриц – это построение объекта типа matrix с помощью одноименного метода. Задать матрицу можно в виде списка.

```
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]]

Также доступен стиль Matlab, когда между элементами ставятся пробелы, а строки разделяются точкой с запятой, при этом такое описание должно быть передано в виде строки.

```
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:

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

Создадим матрицу:

m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')

Извлекаем её диагональ:

```
diag = np.diag(m_sqr_mx)
print(diag)
```

[1 5 9]

Построим диагональную матрицу на базе полученной диагонали.

```
m_diag_np = np.diag(np.diag(m_sqr_mx))
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.

Такой способ не очень удобен, к счастью для нас, для построения такого типа матриц в библиотеке Numpy есть специальная функция – eye().

```
m_eye = np.eye(3)
print(m_eye)
[[ 1. 0. 0.]
[ 0. 1. 0.]
[ 0. 0. 1.]]
```

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)
print(m_idnt)
[[ 1. 0. 0.]
[ 0. 1. 0.]
[ 0. 0. 1.]]
```

Нулевая матрица:

У нулевой матрицы все элементы равны нулю. Пример того, как создать такую матрицу с использованием списков, мы приводить не будем, он дела ется по аналогии с предыдущим разделом. Что касается Numpy, то в

составе этой библиотеки есть функция zeros(), которая создает нужную нам матрицу.

```
m_zeros = np.zeros((3, 3))
print(m_zeros)
[[ 0. 0. 0.]
[ 0. 0. 0.]
[ 0. 0. 0.]]
```

В качестве параметра функции zeros() передается размерность требуемой матрицы в виде кортежа из двух элементов, первый из которых – число строк, второй – столбцов. Если функции zeros() передать в качестве аргумента число, то будет построен нулевой вектор-строка, это мы делали в параграфе, посвященном векторам.

Прямоугольная матрица.

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

2. Как выполняется транспонирование матриц? С помощью функ ции .transpone() и сокращённый вариант .T

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

Для исходной матрицы:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}.$$

Транспонированная будет выглядеть так:

$$A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}.$$

```
>>> 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]]
```

Существует сокращенный вариант получения транспонированной матрицы, он очень удобен в практическом применении:

```
>>> print(A.T)
[[1 4]
[2 5]
[3 6]]
```

3. Приведите свойства операции транспонирования матриц.

Свойство 1. Дважды транспонированная матрица равна исходной мат рице.

Свойство 2. Транспонирование суммы матриц равно сумме транспони рованных матриц.

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

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

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?

С помощью функции .transpone() и сокращённый вариант .T.

5. Какие существуют основные действия над матрицами?

Сложения, вычитания, умножение, умножение на число матрицы

Транспонирование матрицы, вынесение минуса из матрицы(внесение), нахождение обратной матрицы.

6. Как осуществляется умножение матрицы на число?

Необходимо каждый элемент матрицы умножить 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 мат рице, 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?

Складывать можно только матрицы одинаковой размерности — то есть матрицы, у которых совпадает количество столбцов и строк.

Матрицы складываются поэлементно. Например: 1,1 с элементом 1,1; 1,2 с 1,2 и т. д..

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. A + (-A) = Z (нулевая матрица)

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

+ -

11. Как осуществляется операция умножения матриц?

Умножение матриц это уже более сложная операция, по сравнению с рассмотренными выше. Умножать можно только матрицы, отвечающие

следующему требованию: количество столбцов первой матрицы должно быть равно числу строк второй матрицы.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \; B = \begin{pmatrix} 7 & 8 \\ 9 & 1 \\ 2 & 3 \end{pmatrix},$$

$$C = A \times B,$$

$$C = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \times \begin{pmatrix} 7 & 8 \\ 9 & 1 \\ 2 & 3 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 \cdot 7 + 2 \cdot 9 + 3 \cdot 2 & 1 \cdot 8 + 2 \cdot 1 + 3 \cdot 3 \\ 4 \cdot 7 + 5 \cdot 9 + 6 \cdot 2 & 4 \cdot 8 + 5 \cdot 1 + 6 \cdot 3 \end{pmatrix} = \begin{pmatrix} 31 & 19 \\ 85 & 55 \end{pmatrix}.$$

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?

функция dot(), Например:

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]]
```

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

Определитель матрицы размера (n-го порядка) является одной из ее чис ленных характеристик. Определитель матрицы A обозначается как |A| или det(A), его также называют детерминантом.

Перед тем, как привести методику расчета определителя в общем виде, введем понятие минора элемента определителя. *Минор элемента определителя* – это определитель, полученный из данного, путем вычеркивания всех элементов строки и столбца, на пересечении которых стоит данный элемент. Для матрицы 3×3 следующего вида:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

Минор *M23* будет выглядеть так:

$$M_{23} = \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix}.$$

Введем еще одно понятие – *алгебраическое дополнение элемента определителя* – это минор этого элемента, взятый со знаком *плюс* или *минус*:

$$A_{ij} = (-1)^{i+j} M_{ij}.$$

В общем виде вычислить определитель матрицы можно через разложение определителя по элементам строки или столбца. Суть в том, что определитель равен сумме произведений элементов любой строки или столбца на их алгебраические дополнения. Для матрицы 3×3 это правило будет выполняться следующим образом:

$$|A| = \begin{vmatrix} -4 & -1 & 2 \\ 10 & 4 & -1 \\ 8 & 3 & 1 \end{vmatrix} = (-4) \cdot A_{11} + (-1) \cdot A_{12} + 2 \cdot A_{13} =$$

$$= (-4) \cdot (-1)^{1+1} M_{11} + (-1) \cdot (-1)^{1+2} M_{12} + 2 \cdot (-1)^{1+3} M_{13} =$$

$$= (-4) \cdot \begin{vmatrix} 4 & -1 \\ 3 & 1 \end{vmatrix} - (-1) \cdot \begin{vmatrix} 10 & -1 \\ 8 & 1 \end{vmatrix} + 2 \cdot \begin{vmatrix} 10 & 4 \\ 8 & 3 \end{vmatrix} =$$

$$= (-4) \cdot (4+3) + (10+8) + 2 \cdot (30-32) =$$

$$= -28 + 18 - 4 = -14.$$

Свойства определителя матрицы.

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?

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

np.linalg.det(A)

-14.000000000000009

16. Что такое обратная матрица? Какой алгоритм нахождения об ратной 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 мат

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

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?

Для получения обратной матрицы будем используем 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 ли нейных уравнений. Приведите алгоритм решения системы линейных уравнений методом Крамера средствами библиотеки NumPy.

Метод Крамера

In [5]:
```
import numpy as np
```

Создадим и заполним матрицу рандомными значениями от 0 до 10

In [6]:
```
matrix = np.random.randint (0, 10, (4, 4))
print(matrix)
```

```
[[0 8 5 9]
 [3 6 5 9]
 [9 2 9 5]
 [9 9 4 8]]
```

Найдём определитель

In [11]:
```
opred_1 = np.linalg.det(matrix)
print(opred_1)
```

```
1368.000000000001
```

Заполним рандомными значенями столбец свободных членов

In [13]:
```
sv = np.random.randint (0, 10, (4, 1))
print(sv)
x = np.ones((4, 1))
```

```
[[4]
 [5]
 [9]
 [5]]
```

Решим матрицу методом Крамера. Скопируем значения в дополнительную матрицу и подставим в неё столбец свободных членов. Затем найдём определитель

In [23]:
```
t = [0, 1, 2, 3]
if opred_1 != 0:
    for i in t:
        mat_dop = matrix.copy()
        mat_dop[:, i] = sv[:, 0]
        x[i,0] = np.linalg.det(mat_dop) / opred_1
    print(x)
else:
    print("Невозможно решить")
```

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

Матричный метод:

Пусть дана система n линейных уравнений с n переменными $(n \times n)$.
Если основная матрица не вырождена, т.е. $|A| \neq 0$, тогда для матрицы A существует A^{-1}. Умножив матричное уравнение на A^{-1} получим:

$$X = A^{-1} * B$$

Создадим и заполним матрицу:

In [24]:
```
matrix = np.random.randint (0, 10, (4, 4))
print(matrix)
```

```
[[7 6 9 1]
 [0 8 6 5]
 [6 0 6 7]
 [6 4 5 8]]
```

Найдём обратную матрицу

In [25]:
```
mat_inv = np.linalg.inv(matrix)
print(mat_inv)
```

```
[[ 0.07933194 -0.15970772 -0.1263048   0.20041754]
 [ 0.03444676  0.01617954 -0.20615866  0.16597077]
 [ 0.03757829  0.1085595   0.22964509 -0.27348643]
 [-0.10020877  0.04384134  0.05427975  0.06263048]]
```

Найдём определитель

In [26]:
```
opred_1 = np.linalg.det(matrix)
print(opred_1)
```

```
1915.9999999999998
```

Заполним столбец свободных членов

In [27]:
```
sv = np.random.randint (0, 20, (4, 1))
print(sv)
```

```
[[ 6]
 [ 1]
 [17]
 [ 4]]
```

Найдём решения СЛАУ, умножив нашу обратную матрицу на столбец свободных членов:

Найдём решения СЛАУ, умножив нашу обратную матрицу на столбец свободных членов:

In [28]:
```
if opred_1 != 0:
    x = mat_inv.dot(sv)
    print(x)
else:
    print("Невозможно решить")
```

```
[[-1.02922756]
 [-2.61795407]
 [ 3.1440501 ]
 [ 0.61586639]]
```

Вывод: в результате выполнения лабораторной работы были получены практические знания и теоретические сведения о методах работы с матрицами и векторами с помощью библиотеки NumPy языка программирования Python.