

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

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

ОТЧЕТ
по лабораторной работе №3
дисциплины
«Технологии распознавания образов»

Выполнил:
Сотников Андрей Александрович
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

Проработка примеров из лабораторной работы:

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

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

[1 2]

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]]
```

Рисунок 1 – проработка примеров

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

```
In [16]: m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
```

```
In [17]: diag = np.diag(m_sqr_mx)  
print(diag)  
  
[1 5 9]
```

```
In [18]: m_diag_np = np.diag(np.diag(m_sqr_mx))  
print(m_diag_np)  
  
[[1 0 0]  
 [0 5 0]  
 [0 0 9]]
```

```
In [19]: 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]]
```

Рисунок 2 – проработка примеров

```

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 [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()
          print(R)

[[19 43]
 [22 50]]

[[19 43]
 [22 50]]

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()
          print(format(A_T_det, '.9g'))

-2
-2

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

In [35]: print(L)

[[1 2]
 [3 4]]

```

Рисунок 3 – проработка примеров

```
In [38]: 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]]
```

```
In [39]: print(R)
```

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

```
In [40]: 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 [41]: 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 [42]: print(R)
```

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

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

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

```
In [46]: print(R)
```

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

```
In [47]: 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 [48]: print(R)
```

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

Рисунок 4 – проработка примеров

```
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
```

```
In [51]: print(A.T)
```

```
[[-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)
```

```
In [53]: print(det_A)
```

```
-14.0
```

```
In [54]: print(det_A_t)
```

```
-14.0
```

```
In [56]: A = np.matrix('1 -3; 2 5')
A_inv = np.linalg.inv(A)
print(A)
```

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

```
In [57]: print(A_inv)
```

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

```
In [58]: A = np.matrix('1. -3.; 2. 5.')
L = np.linalg.inv(A.T)
R = (np.linalg.inv(A)).T
print(L)
```

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

```
In [59]: print(R)
```

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

```
In [60]: 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)
```

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

```
In [61]: print(R)
```

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

```
In [63]: m_eye = np.eye(4)
print(m_eye)
```

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

Рисунок 5 – проработка примеров

```
[ 0.00000000  0.00000000]

In [63]: m_eye = np.eye(4)
print(m_eye)

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

In [64]: rank = np.linalg.matrix_rank(m_eye)
print(rank)

4

In [65]: m_eye[3][3] = 0
print(m_eye)

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

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

3

In [ ]:
```

Рисунок 6 – проработка примеров

Задание №1: Создать ноутбук, в котором будут приведены собственные примеры на языке Python для каждого из представленных свойств матричных вычислений.

Рисунок 7 – Проработка свойств

```
In [10]: import numpy as np
```

Транспонирование матрицы

Свойство 1. Дважды транспонированная матрица равна исходной матрице.

```
In [12]: x = np.matrix('18 0 -9; 5 1 12; 10 1 -9')
t_mrx = (x.T).T
print(t_mrx)

[[18  0 -9]
 [ 5  1 12]
 [10  1 -9]]
```

Свойство 2. Транспонирование суммы матриц равно сумме транспонированных матриц

```
In [13]: x = np.matrix('18 0 -9; 5 1 12; 10 1 -9')
y = np.matrix('9 -5 10; 7 1 0; 9 2 -5')
first = (x + y).T
second = x.T + y.T
print(first)
print(second)

[[ 27  12  19]
 [ -5   2   3]
 [  1  12 -14]]
[[ 27  12  19]
 [ -5   2   3]
 [  1  12 -14]]
```

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

```
In [16]: x = np.matrix('18 5 -9; 5 1 12; 10 1 -9')
y = np.matrix('9 -5 10; 7 1 0; 9 2 -5')
first_mrx = (x.dot(y)).T
sec_mrx = (y.T).dot(x.T)
print(first_mrx)
print(sec_mrx)

[[ 116  160   16]
 [-103    0  -67]
 [ 225  -10  145]]
[[ 116  160   16]
 [-103    0  -67]
 [ 225  -10  145]]
```

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

```
In [17]: x = np.matrix('18 5 -9; 5 1 12; 10 1 -9')
g = 5
first_mrx = g * (x.T)
sec_mrx = (g * x).T
print(first_mrx)
print(sec_mrx)

[[ 90  25  50]
 [ 25   5   5]
 [-45  60 -45]]
[[ 90  25  50]
 [ 25   5   5]
 [-45  60 -45]]
```

Свойство 5. Определители исходной и транспонированной матрицы совпадают:

Рисунок 8 – Проработка свойств


```
In [18]: x = np.matrix('18 5 -9; 5 1 12; 10 1 -9')
x_det = np.linalg.det(x)
x_T_det = np.linalg.det(x.T)
print(format(x_det, '.9g'))
print(format(x_T_det, '.9g'))
```

```
492
492
```

Умножение матрицы на число

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

```
In [25]: A = np.matrix('18 5 -9; 5 1 12; 10 1 -9')
L = 1 * A
R = A
print(L)
print(R)
```

```
[[18  5 -9]
 [ 5  1 12]
 [10  1 -9]]
[[18  5 -9]
 [ 5  1 12]
 [10  1 -9]]
```

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

```
In [26]: A = np.matrix('18 5 -9; 5 1 12; 10 1 -9')
Z = np.matrix('0 0 0; 0 0 0; 0 0 0')
L = 0 * A
print(L)
print(Z)
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
[[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 [28]: A = np.matrix('15 9 -5; -3 10 11; 2 1 4')
p = 21
q = 12
L = (p + q) * A
R = p * A + q * A
print(L)
print(R)
```

```
[[ 495  297 -165]
 [ -99  330  363]
 [  66   33  132]]
[[ 495  297 -165]
 [ -99  330  363]
 [  66   33  132]]
```

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

```
In [29]: A = np.matrix('1 2; 3 4')
p = 5
q = 6
L = (p * q) * A
R = p * (q * A)
print(L)
print(R)
```

```
[[ 30  60]
```

Рисунок 9 – Проработка свойств

Свойство 3. Транспонирование суммы матриц на число равно сумме транспонированных матриц на заданное число.

```
In [30]: 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

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

```
In [31]: A = np.matrix('12 7; 8 11')
B = np.matrix('5 5; 9 3')
L = A + B
R = B + A
print(L)
print(R)
```

```
[[17 12]
 [17 14]]
[[17 12]
 [17 14]]
```

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

```
In [32]: A = np.matrix('12 7; 8 11')
B = np.matrix('5 5; 9 3')
C = np.matrix('11 2; 8 9')
L = A + (B + C)
R = (A + B) + C
print(L)
print(R)
```

```
[[28 14]
 [25 23]]
[[28 14]
 [25 23]]
```

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

```
In [33]: A = np.matrix('12 7; 8 11')
Z = np.matrix('0 0; 0 0')
L = A + (-1)*A
print(L)
```

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

Рисунок 10 – Проработка свойств

Умножение матриц

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

```
In [34]: A = np.matrix('12 7; 8 11')
B = np.matrix('5 5; 9 3')
C = np.matrix('11 2; 8 9')
L = A.dot(B.dot(C))
R = (A.dot(B)).dot(C)
print(L)
print(R)
```

```
[[2001  975]
 [2113  935]]
[[2001  975]
 [2113  935]]
```

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

```
In [35]: A = np.matrix('12 7; 8 11')
B = np.matrix('5 5; 9 3')
C = np.matrix('11 2; 8 9')
L = A.dot(B + C)
R = A.dot(B) + A.dot(C)
print(L)
print(R)
```

```
[[311 168]
 [315 188]]
[[311 168]
 [315 188]]
```

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

```
In [36]: A = np.matrix('12 7; 8 11')
B = np.matrix('5 5; 9 3')
L = A.dot(B)
R = B.dot(A)
print(L)
print(R)
```

```
[[123  81]
 [139  73]]
[[100  90]
 [132  96]]
```

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

```
In [37]: A = np.matrix('5 9; 12 7')
E = np.matrix('1 0; 0 1')
L = E.dot(A)
R = A.dot(E)
print(L)
print(R)
print(A)
```

```
[[ 5  9]
 [12  7]]
[[ 5  9]
 [12  7]]
[[ 5  9]
 [12  7]]
```

Рисунок 11 – Проработка свойств

Определитель матрицы

Свойство 1. Определитель матрицы остается неизменным при ее транспонировании:

```
In [39]: A = np.matrix('7 1 -8; 15 42 16; 7 5 6')
print(A)
print(A.T)
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)

[[ 7  1 -8]
 [15 42 16]
 [ 7  5  6]]
[[ 7 15  7]
 [ 1 42  5]
 [-8 16  6]]
2978.0
2978.0
```

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

```
In [40]: A = np.matrix('7 1 -8; 15 42 16; 7 5 6')
print(A)
np.linalg.det(A)

[[ 7  1 -8]
 [15 42 16]
 [ 7  5  6]]
```

Out[40]: 2977.9999999999986

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

```
In [41]: A = np.matrix('7 1 -8; 15 42 16; 7 5 6')
print(A)
B = np.matrix('18 4 -12; -1 -2 3; 5 1 5')
print(B)
round(np.linalg.det(A), 3)
round(np.linalg.det(B), 3)

[[ 7  1 -8]
 [15 42 16]
 [ 7  5  6]]
[[ 18  4 -12]
 [-1 -2  3]
 [ 5  1  5]]
```

Out[41]: -262.0

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

```
In [43]: A = np.matrix('-5 5 1; 0 12 3; -5 5 1')
print(A)
np.linalg.det(A)

[[-5  5  1]
 [ 0 12  3]
 [-5  5  1]]
```

Out[43]: 0.0

Рисунок 12 – Проработка свойств

```
In [45]: 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[45]: -28.0

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 [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)
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[46]: 4.0

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

```
In [50]: A = np.matrix('-5 5 1; 0 12 3; -5 5 2')
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)
```

```
[[ -5  5  1]
 [  0 12  3]
 [ -5  5  2]]
[[ -5  5  1]
 [-10 22  5]
 [ -5  5  2]]
```

Out[50]: -60.0

Рисунок 13 – Проработка свойств

```
In [54]: A = np.matrix('5 5 1; 1 12 3; -5 5 3')
print(A)
k = 2
A[1, :] = A[0, :] + k * A[2, :]
round(np.linalg.det(A), 3)
```

```
[[ 5  5  1]
 [ 1 12  3]
 [-5  5  3]]
```

Out[54]: 0.0

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

```
In [55]: 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[55]: 0.0

Рисунок 14 – Проработка свойств

Обратная матрица

```
In [56]: 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.]]
```

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

```
In [57]: 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]]
```

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

```
In [58]: 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]]
```

```
In [ ]:
```

Рисунок 15 – Проработка свойств

Задание №2: Создать ноутбук, в котором будут приведены собственные примеры решения систем линейных уравнений матричным методом и методом Крамера.

Решение матричным методом

```
In [21]: M = np.matrix('1 5 6; 2 5 1; -3 2 7')
V = np.matrix ('3; -4; 1')
```

Теперь необходимо проверить является ли матрица вырожденной (определитель равен 0) от этого будет зависеть сможем ли вообще использовать матричный метод для решения данного уравнения

```
In [22]: opr = np.linalg.det(M)
print(opr)
```

```
62.00000000000001
```

Определитель не равен 0

Найдем матрицу, обратную матрице коэффициентов при неизвестных

```
In [23]: m_inv = np.linalg.inv(M)
print(m_inv)

[[ 0.53225806 -0.37096774 -0.40322581]
 [-0.27419355  0.40322581  0.17741935]
 [ 0.30645161 -0.27419355 -0.08064516]]
```

Найдем матрицу не известных по формуле $X = M^{-1} * V$

```
In [25]: m_inv.dot(V)
```

```
Out[25]: matrix([[ 2.67741935],
                 [-2.25806452],
                 [ 1.93548387]])
```

В библиотеке numpy для решения систем уравнений матричным методом используется функция solve из linalg, проверим правильность решения с помощью этой функции

```
In [26]: np.linalg.solve(M,V)
```

```
Out[26]: matrix([[ 2.67741935],
                 [-2.25806452],
                 [ 1.93548387]])
```

Значения совпадают

Рисунок 16 – Решение матричным методом

Решение методом Крамера

Решим выше указанное уравнение методом Крамера

Для начала объявим необходимые матрицы, а именно матрицу коэффициентов при неизвестных, матрицу свободных членов и матрицу которая будет возвращать первоначальные значения изменённому столбцу. Так же зададим список для определителей полученных в ходе подстановки значений из матрицы свободных членов в матрицу коэффициентов при неизвестных.

```
In [36]: M = np.matrix('1 5 6; 2 5 1; -3 2 7')
V = np.matrix ('3; -4; 1')
check_matrix = np.matrix('0;0;0')
dets = []
```

Посчитаем изначальный определитель матрицы коэффициентов при неизвестных, его так же называют главным определителем

```
In [37]: main_det = np.linalg.det(M)
```

```
In [38]: i = 0
while i < len(M):
    check_matrix[:, 0] = M[:, i]
    M[:, i] = V[:, 0]
    temp_det = np.linalg.det(M)
    dets.append(temp_det)
    M[:, i] = check_matrix[:, 0]
    i += 1
```

Теперь осталось только сделать из полученного списка питру массив и разделить каждый элемент этого массива на главный определитель

```
In [39]: dets = np.array(dets)
result = dets // main_det
print(result)

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

Полученная матрица сходится с результатами предыдущего метода, из чего можно сделать вывод, что алгоритм работает корректно.

Рисунок 17 – Решение методом Крамера

Контрольные вопросы

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

Вектор-строка

Вектор-строка имеет следующую математическую запись.

$$v = (1 \ 2) \quad (3)$$

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.

$$v = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad (4)$$

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.

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

Создадим следующую матрицу.

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

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.

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

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.

$$E = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}. \quad (8)$$

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:

$$\hat{A} + (\hat{B} + \hat{C}) = (\hat{A} + \hat{B}) + \hat{C}.$$

➤ 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:

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

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

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:

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

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 \(al-shell.ru\)](http://al-shell.ru)

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

[Решение систем линейных уравнений в Python \(xn--80ahcjeib4ac4d.xn--p1ai\)](http://xn--80ahcjeib4ac4d.xn--p1ai)