

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

Институт перспективной инженерии  
Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**  
**дисциплины**  
**«Искусственный интеллект и машинное обучение»**  
**Вариант 3**

Выполнил:  
Борцов Богдан Михайлович  
2 курс, группа ИТС-б-о-23-1,  
11.03.02 «Инфокоммуникационные  
технологии и системы связи»,  
направленность (профиль)  
«Инфокоммуникационные системы и  
сети», очная форма обучения

---

(подпись)

Проверил:

Доцент департамента цифровых,  
робототехнических систем и  
электроники Воронкин Р. А.

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2025 г.

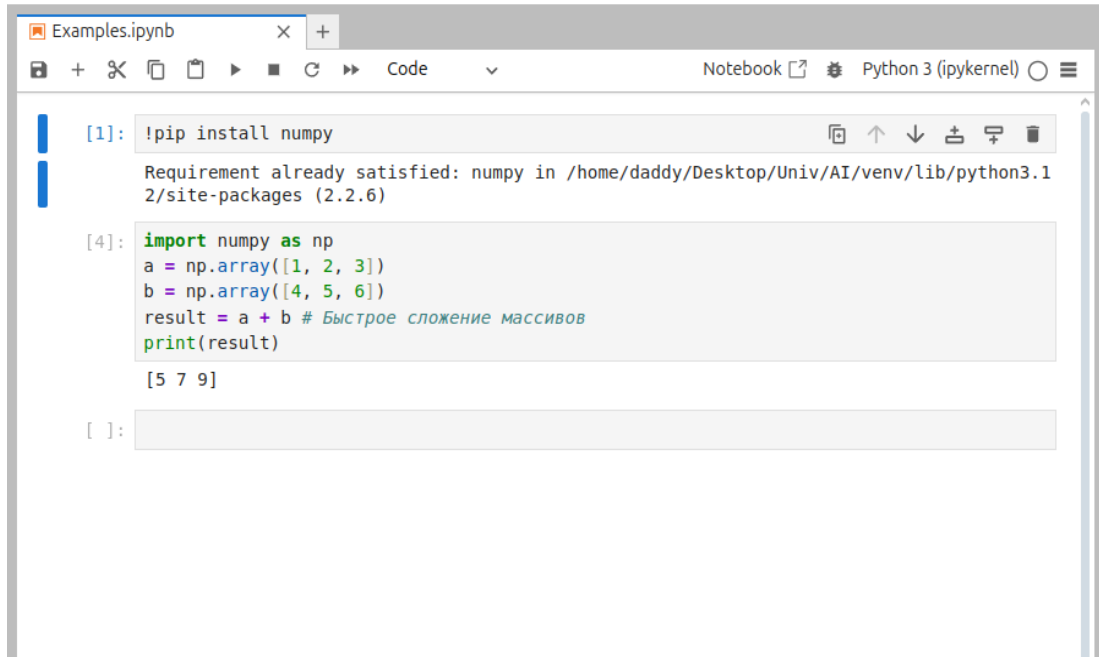
**Тема:** Основы работы с библиотекой NumPy

**Цель:** исследовать базовые возможности библиотеки NumPy языка программирования Python.

Ссылка на репозиторий: <https://github.com/REPONCFU/ai-jlab>

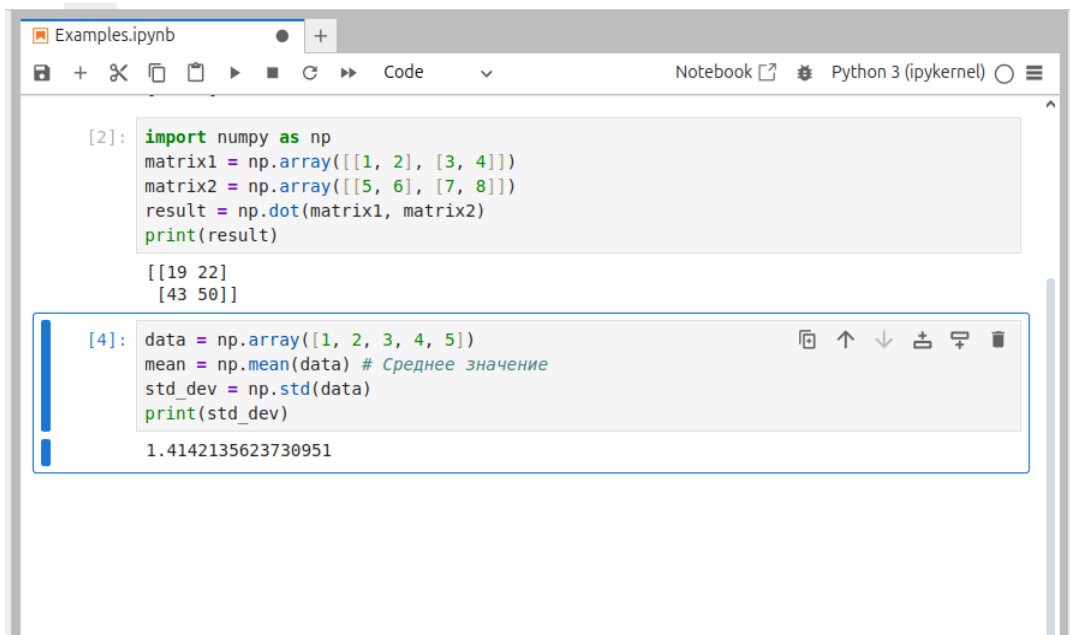
**Порядок выполнения работы:**

1. Ознакомление с методическими указаниями
2. Выполнение методических указаний



The screenshot shows a Jupyter Notebook window titled 'Examples.ipynb'. The first code cell contains the command `!pip install numpy`, which outputs 'Requirement already satisfied: numpy in /home/daddy/Desktop/Univ/AI/venv/lib/python3.12/site-packages (2.2.6)'. The second code cell contains the following Python code: `import numpy as np`, `a = np.array([1, 2, 3])`, `b = np.array([4, 5, 6])`, `result = a + b # Быстрое сложение массивов`, and `print(result)`. The output of this cell is `[5 7 9]`. The third code cell is empty, showing `[ ]:`.

Рисунок 1. Векторизация NumPy



The screenshot shows a Jupyter Notebook window titled 'Examples.ipynb'. The first code cell contains the following Python code: `import numpy as np`, `matrix1 = np.array([[1, 2], [3, 4]])`, `matrix2 = np.array([[5, 6], [7, 8]])`, `result = np.dot(matrix1, matrix2)`, and `print(result)`. The output of this cell is `[[19 22]` and `[43 50]]`. The second code cell contains the following Python code: `data = np.array([1, 2, 3, 4, 5])`, `mean = np.mean(data) # Среднее значение`, `std_dev = np.std(data)`, and `print(std_dev)`. The output of this cell is `1.4142135623730951`.

Рисунок 2. Матричные операции и статистические операции

```
Examples.ipynb
[5]: import numpy as np

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

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

[9]: m[1,0]

[9]: np.int64(5)

[8]: m[1,:]

[8]: matrix([[5, 6, 7, 8]])

[10]: m[:, 2]

[10]: matrix([[3],
             [7],
             [5]])

[11]: m[1, 2:]

[11]: matrix([[7, 8]])

[12]: m[0:2, 1]

[12]: matrix([[2],
             [6]])

[13]: m[0:2, 1:3]

[13]: matrix([[2, 3],
             [6, 7]])

[14]: cols = [0, 1, 3]
      m[:, cols]

[14]: matrix([[1, 2, 4],
             [5, 6, 8],
             [9, 1, 7]])
```

Рисунок 3. Достаем из матрицы необходимые данные по координатам

```
Examples.ipynb
#Доступ к элементам массива

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

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

[17]: type(m)

[17]: numpy.matrix

[18]: m = np.array(m)
      type(m)

[18]: numpy.ndarray

[19]: m.shape

[19]: (3, 4)

[20]: m.max()

[20]: np.int64(9)

[21]: np.max(m)

[21]: np.int64(9)

[22]: m.max()

[22]: np.int64(9)

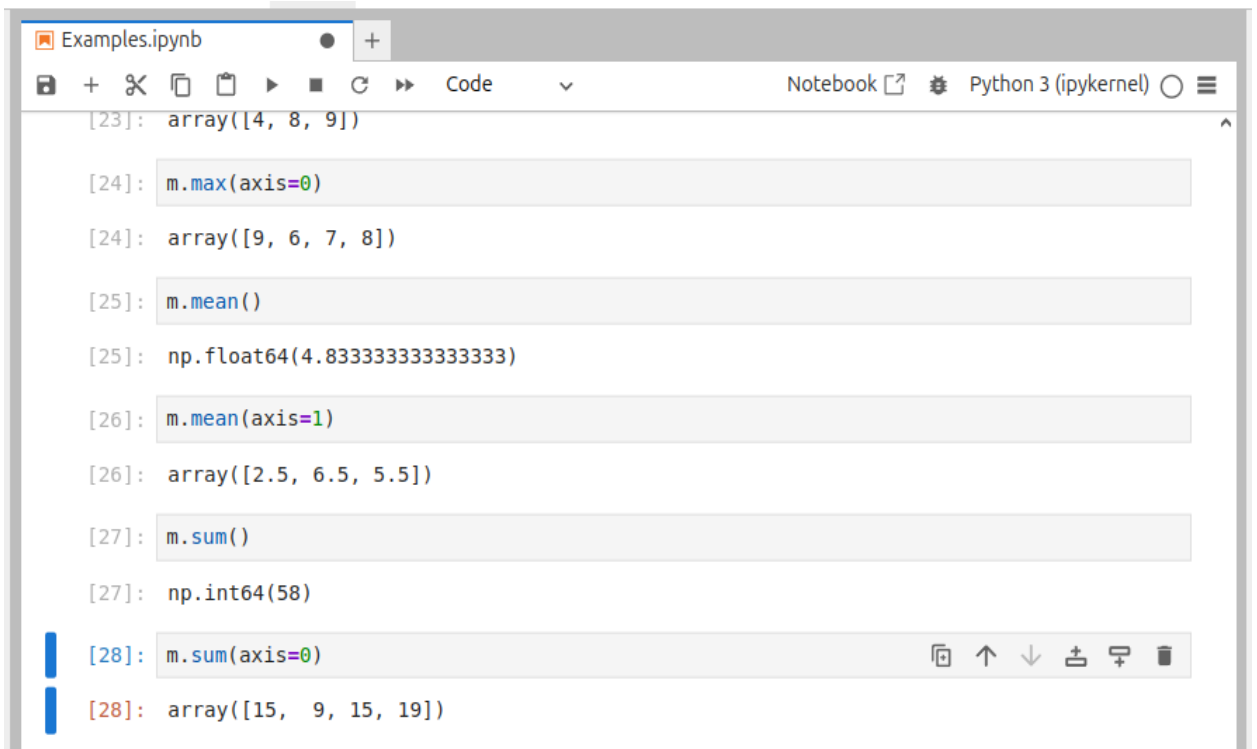
[23]: m.max(axis=1)

[23]: array([4, 8, 9])

[24]: m.max(axis=0)

[24]: array([9, 6, 7, 8])
```

Рисунок 4. Расчет статистик по данным в массиве



The image shows a Jupyter Notebook interface with a single cell containing the following code and output:

```
[23]: array([4, 8, 9])

[24]: m.max(axis=0)

[24]: array([9, 6, 7, 8])

[25]: m.mean()

[25]: np.float64(4.833333333333333)

[26]: m.mean(axis=1)

[26]: array([2.5, 6.5, 5.5])

[27]: m.sum()

[27]: np.int64(58)

[28]: m.sum(axis=0)

[28]: array([15, 9, 15, 19])
```

The notebook has a title bar 'Examples.ipynb' and a toolbar with icons for saving, adding, deleting, and running code. The kernel is 'Python 3 (ipykernel)'.

Рисунок 5. Методы расчета статистик

```
#Использование boolean массива для доступа к ndarray

[29]: nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
      letters = np.array(['a', 'b', 'c', 'd', 'a', 'e', 'b'])
      a = True
      b = 5 > 7
      print(b)
      False

[30]: less_than_5 = nums < 5
      less_than_5

[30]: array([ True,  True,  True,  True, False, False, False, False, False,
         False])

[31]: pos_a = letters == 'a'
      pos_a

[31]: array([ True, False, False, False,  True, False, False])

[32]: less_than_5 = nums < 5
      less_than_5

[32]: array([ True,  True,  True,  True, False, False, False, False, False,
         False])

[33]: nums[less_than_5]

[33]: array([1, 2, 3, 4])

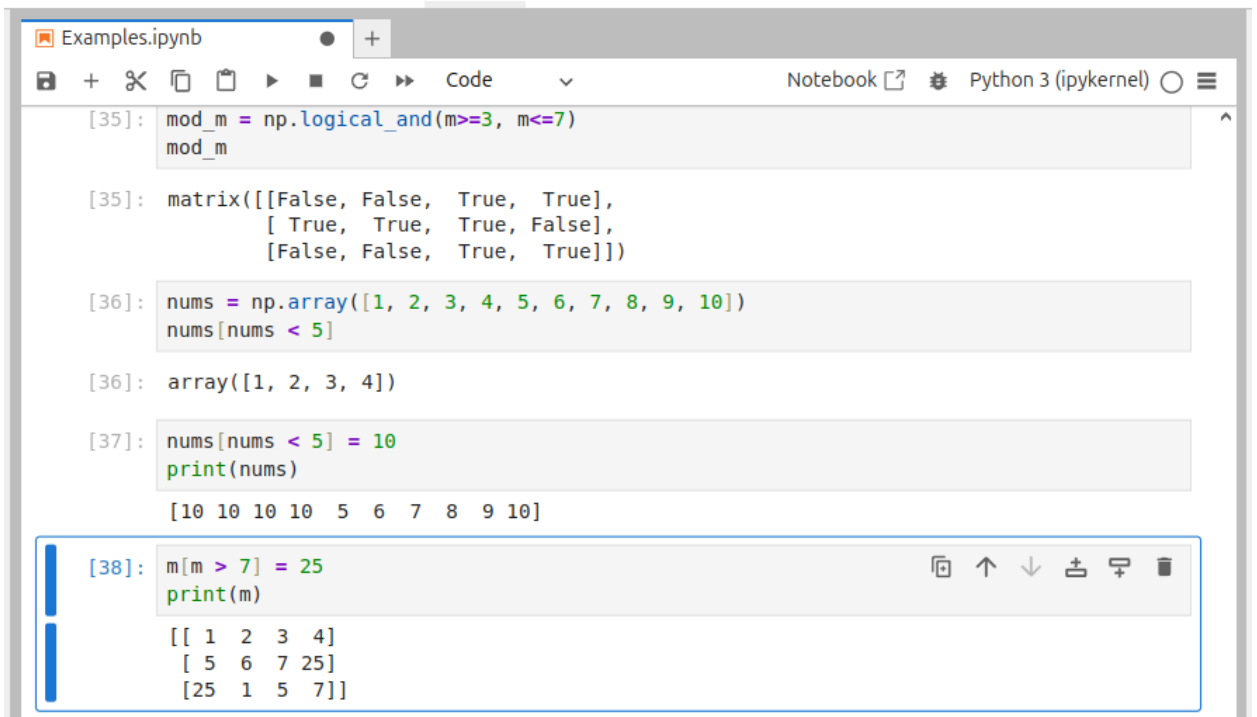
[34]: m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
      print(m)

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

[35]: mod_m = np.logical_and(m>=3, m<=7)
      mod_m

[35]: matrix([[False, False,  True,  True],
             [ True,  True,  True, False],
             [ True,  True,  True,  True]])
```

Рисунок 6. Использование boolean массива для доступа к ndarray



```
Examples.ipynb
+ ✂ 📄 ▶ ■ ↺ ⏩ Code Python 3 (ipykernel)

[35]: mod_m = np.logical_and(m>=3, m<=7)
      mod_m

[35]: matrix([[False, False,  True,  True],
             [ True,  True,  True, False],
             [False, False,  True,  True]])

[36]: nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
      nums[nums < 5]

[36]: array([1, 2, 3, 4])

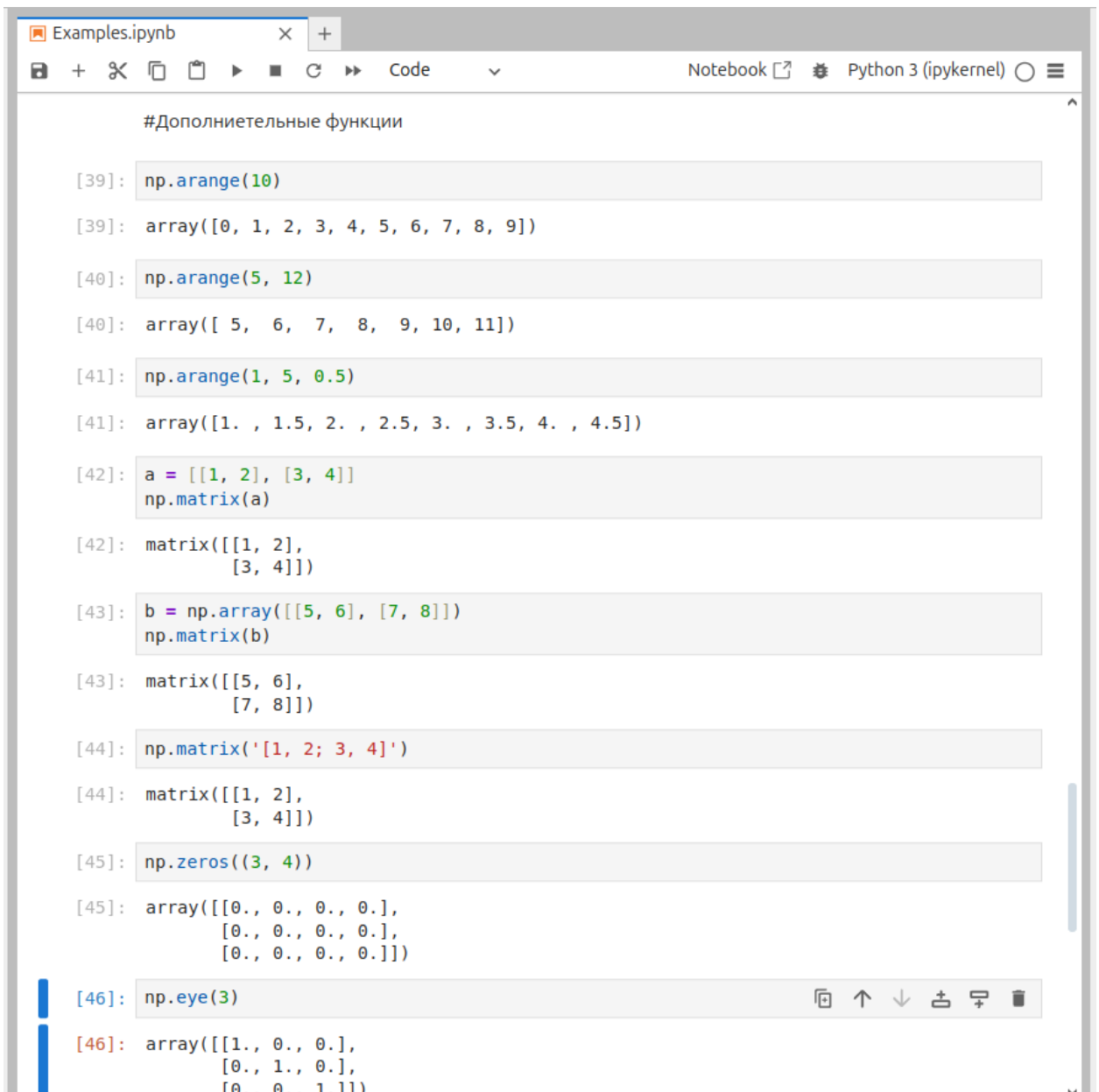
[37]: nums[nums < 5] = 10
      print(nums)

[10 10 10 10  5  6  7  8  9 10]

[38]: m[m > 7] = 25
      print(m)

[[ 1  2  3  4]
 [ 5  6  7 25]
 [25  1  5  7]]
```

Рисунок 7. Дополнительные юзкейсы boolean



```
#Дополнительные функции

[39]: np.arange(10)
[39]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[40]: np.arange(5, 12)
[40]: array([ 5, 6, 7, 8, 9, 10, 11])

[41]: np.arange(1, 5, 0.5)
[41]: array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])

[42]: a = [[1, 2], [3, 4]]
      np.matrix(a)
[42]: matrix([[1, 2],
              [3, 4]])

[43]: b = np.array([[5, 6], [7, 8]])
      np.matrix(b)
[43]: matrix([[5, 6],
              [7, 8]])

[44]: np.matrix('1, 2; 3, 4')
[44]: matrix([[1, 2],
              [3, 4]])

[45]: np.zeros((3, 4))
[45]: array([[0., 0., 0., 0.],
            [0., 0., 0., 0.],
            [0., 0., 0., 0.]])

[46]: np.eye(3)
[46]: array([[1., 0., 0.],
            [0., 1., 0.],
            [0., 0., 1.]])
```

Рисунок 8. Дополнительные функции

```
[47]: A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
      np.ravel(A)

[47]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

[48]: np.ravel(A, order='C')

[48]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

[49]: np.ravel(A, order='F')

[49]: array([1, 4, 7, 2, 5, 8, 3, 6, 9])

[50]: a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
      np.where(a % 2 == 0, a * 10, a / 10)

[50]: array([ 0. ,  0.1, 20. ,  0.3, 40. ,  0.5, 60. ,  0.7, 80. ,  0.9])

[52]: a = np.random.rand(10)
      a

[52]: array([0.04884413, 0.63455721, 0.12160657, 0.2701945 , 0.64257944,
          0.80165535, 0.20607685, 0.231942 , 0.63105829, 0.02610174])

[53]: np.where(a > 0.5, True, False)

[53]: array([False,  True, False, False,  True,  True, False, False,  True,
          False])

[54]: np.where(a > 0.5, 1, -1)

[54]: array([-1,  1, -1, -1,  1,  1, -1, -1,  1, -1])

[56]: x = np.linspace(0, 1, 5)
      x

[56]: array([0. , 0.25, 0.5 , 0.75, 1.  ])

[57]: y = np.linspace(0, 2, 5)
      y

[57]: array([0. , 0.5, 1. , 1.5, 2.  ])
```

Рисунок 9. Дополнительные функции



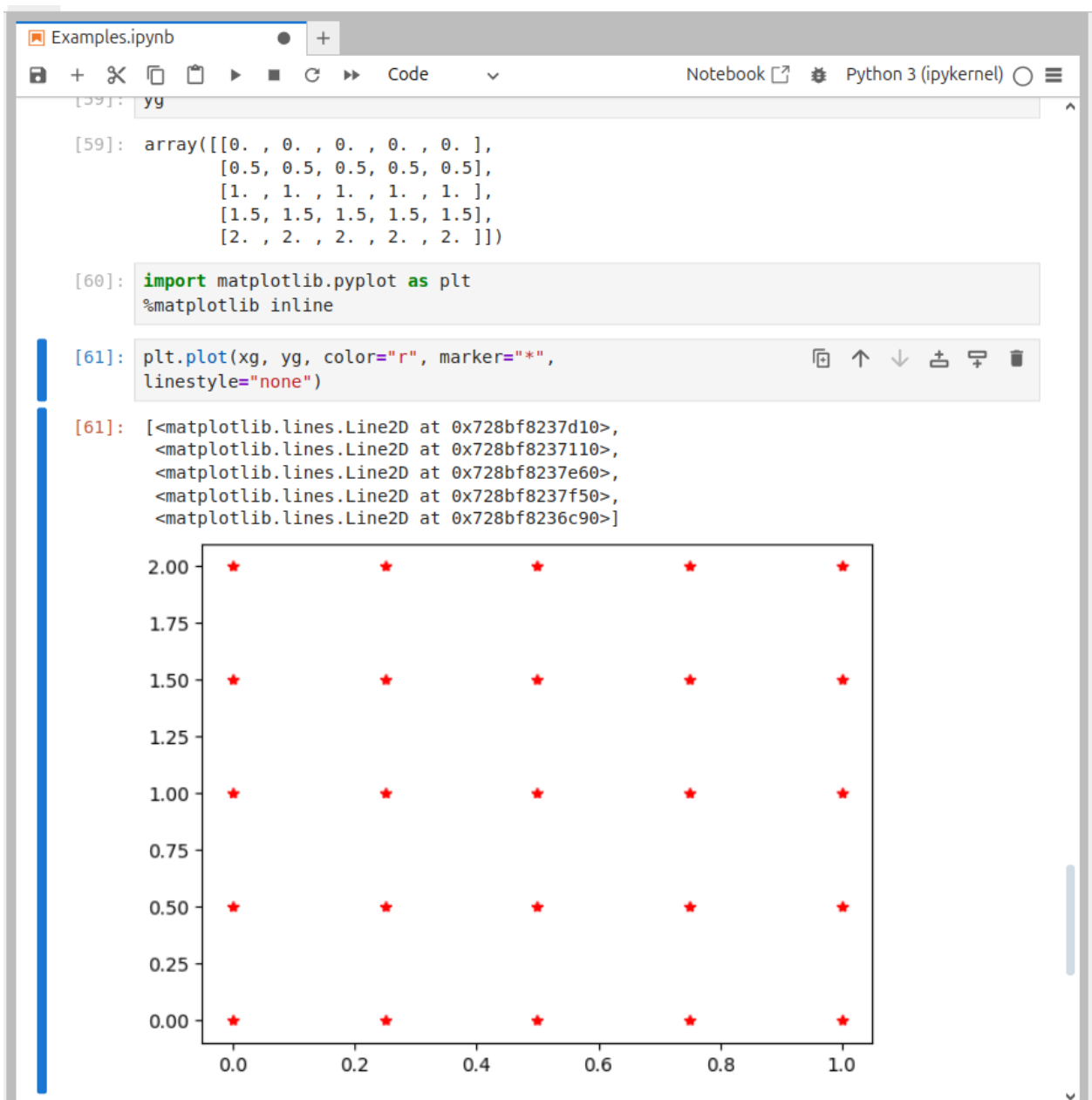
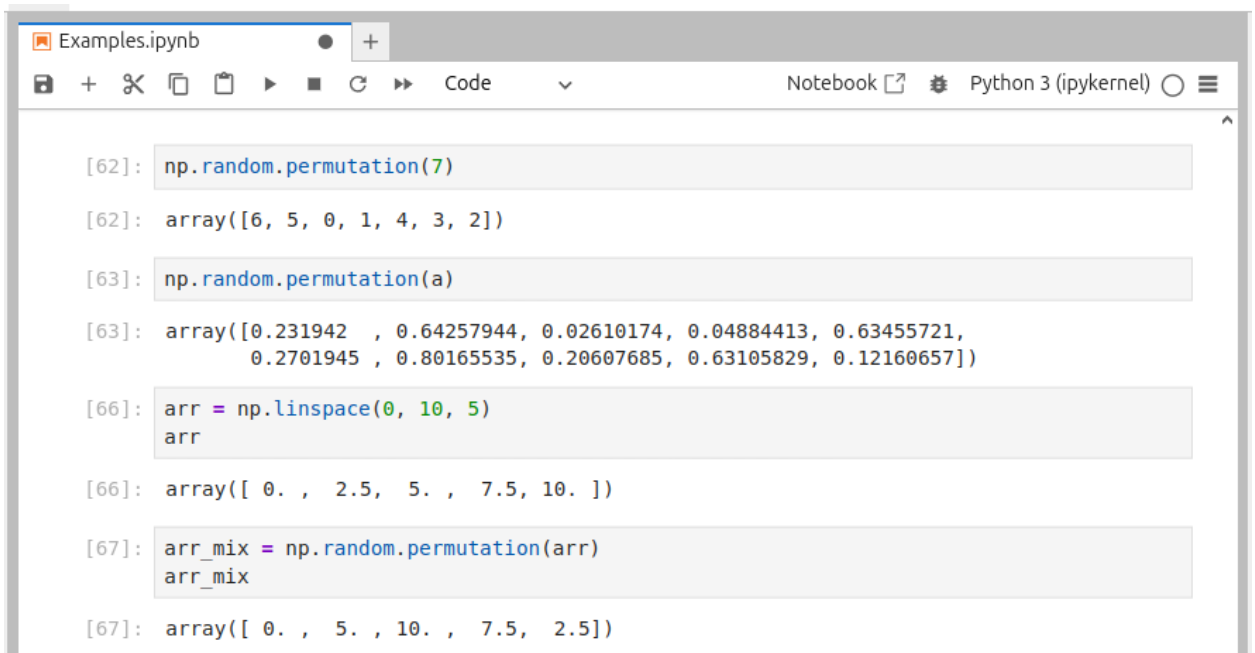


Рисунок 10. Построение графика matplotlib



The image shows a Jupyter Notebook interface with a single cell containing several lines of Python code. The code demonstrates the use of `np.random.permutation` and `np.linspace` to generate random permutations and a linear space, and then combines them. The output of each code block is displayed below the code.

```
[62]: np.random.permutation(7)
[62]: array([6, 5, 0, 1, 4, 3, 2])

[63]: np.random.permutation(a)
[63]: array([0.231942, 0.64257944, 0.02610174, 0.04884413, 0.63455721,
          0.2701945, 0.80165535, 0.20607685, 0.63105829, 0.12160657])

[66]: arr = np.linspace(0, 10, 5)
      arr
[66]: array([ 0. ,  2.5,  5. ,  7.5, 10. ])

[67]: arr_mix = np.random.permutation(arr)
      arr_mix
[67]: array([ 0. ,  5. , 10. ,  7.5,  2.5])
```

Рисунок 11. Работа с рандомом и миксингом

```
#Матрицы и векторы

[68]: import numpy as np

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

      [1 2]

[70]: v_hor_zeros_v1 = np.zeros((5,))
      print(v_hor_zeros_v1 )

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

[71]: v_hor_zeros_v2 = np.zeros((1, 5))
      print(v_hor_zeros_v2 )

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

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

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

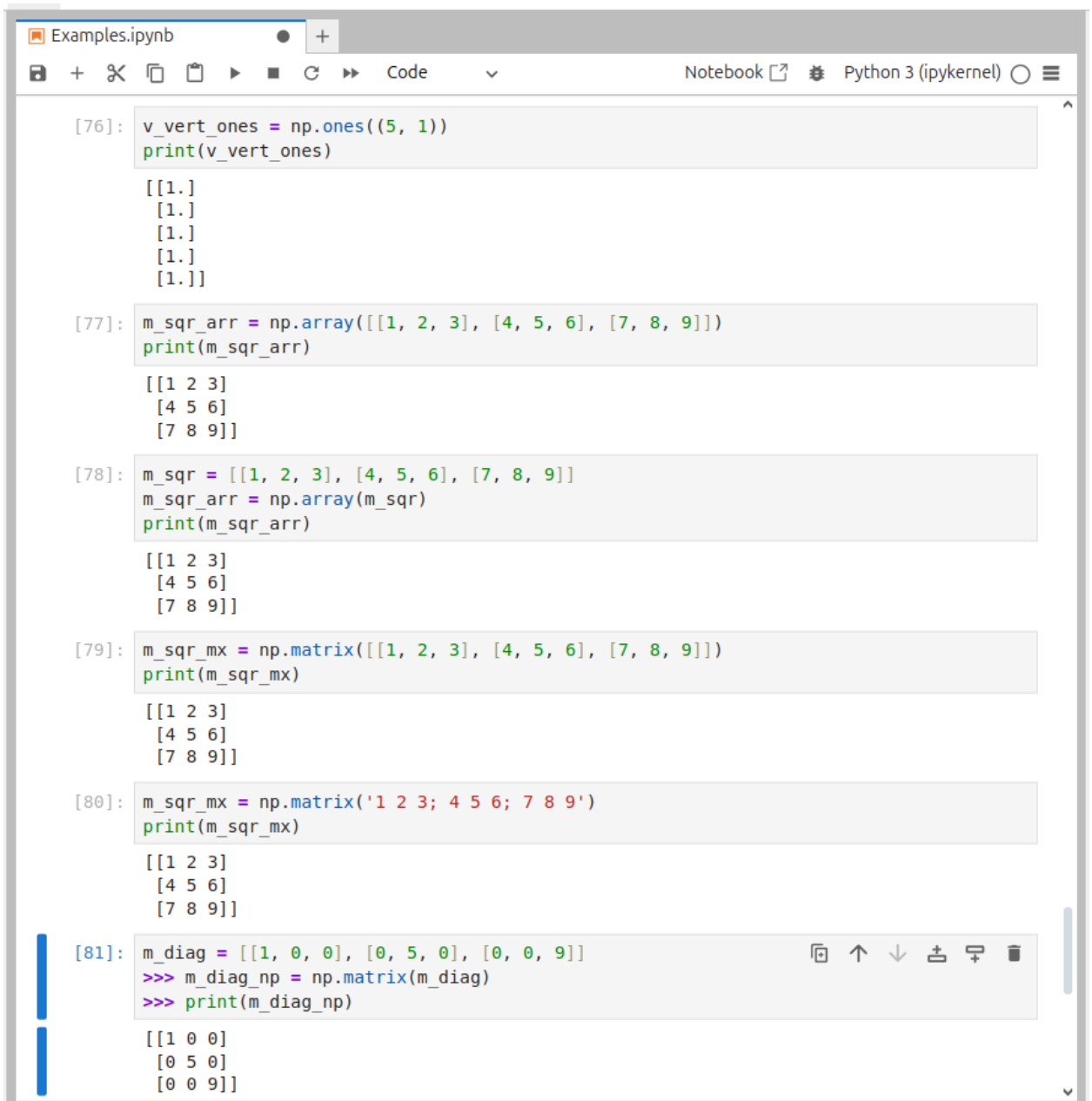
[74]: v_vert_np = np.array([[1], [2]])
      print(v_vert_np)

      [[1]
       [2]]

[75]: v_vert_zeros = np.zeros((5, 1))
      print(v_vert_zeros)

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

Рисунок 12. Работа с векторами



The image shows a Jupyter Notebook interface with the following code cells:

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

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

[77]: 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]]

[78]: 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]]

[79]: 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]]

[80]: 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]]

[81]: 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]]
```

Рисунок 13. Работа с матрицами

The image shows a Jupyter Notebook interface with a tab titled 'Examples.ipynb'. The notebook contains five code cells, each with a prompt number in brackets. The code in each cell demonstrates different ways to create and manipulate matrices in NumPy. The output of each cell is displayed below the code. The first cell creates a 3x3 matrix from a string and prints its diagonal. The second cell creates a 3x3 diagonal matrix from the diagonal of the first matrix. The third cell creates a 3x3 matrix from a list of lists and prints it. The fourth cell creates a 3x3 identity matrix and prints it. The fifth cell creates a 3x3 identity matrix and prints it.

```
[82]: 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]

[83]: m_diag_np = np.diag(np.diag(m_sqr_mx))
      print(m_diag_np)

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

[84]: 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]]

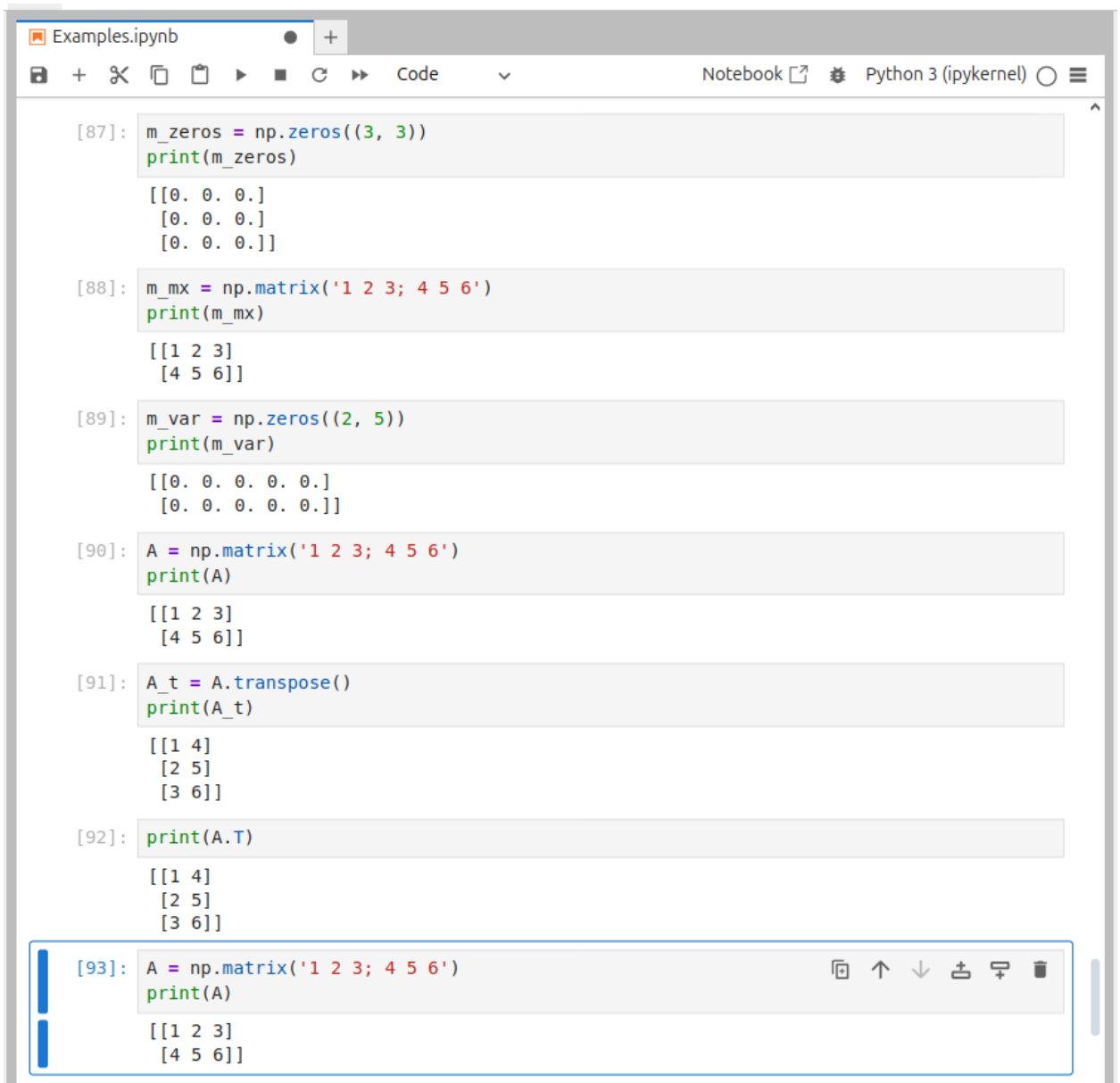
[85]: m_eye = np.eye(3)
      print(m_eye)

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

[86]: m_idnt = np.identity(3)
      print(m_idnt)

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

Рисунок 14. Единичные матрицы



The image shows a Jupyter Notebook interface with a tab titled 'Examples.ipynb'. The notebook contains several code cells demonstrating matrix operations using NumPy. The cells are numbered [87] through [93]. The operations include creating zero matrices, creating matrices from strings, transposing a matrix, and printing the transpose of a matrix. The output of each cell is displayed below the code.

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

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

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

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

[89]: m_var = np.zeros((2, 5))
      print(m_var)

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

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

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

[91]: A_t = A.transpose()
      print(A_t)

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

[92]: print(A.T)

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

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

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

Рисунок 15. Операции над матрицами

```
Examples.ipynb
+
Notebook Python 3 (ipykernel)

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

[94]: 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)
      print(R)

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

[96]: 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]]

[97]: 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]]
```

Рисунок 16. Транспонирование матриц

The image shows a Jupyter Notebook interface with three code cells. The first cell (index 99) defines a 2x3 matrix A and multiplies it by the scalar 3. The second cell (index 100) defines a 2x2 matrix A and multiplies it by the scalar 1. The third cell (index 101) defines a 2x2 matrix A and multiplies it by the zero matrix Z. A fourth cell (index 103) defines a 2x2 matrix A and multiplies it by the sum of scalars p and q.

```
Examples.ipynb
```

Python 3 (ipykernel)

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

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

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

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

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

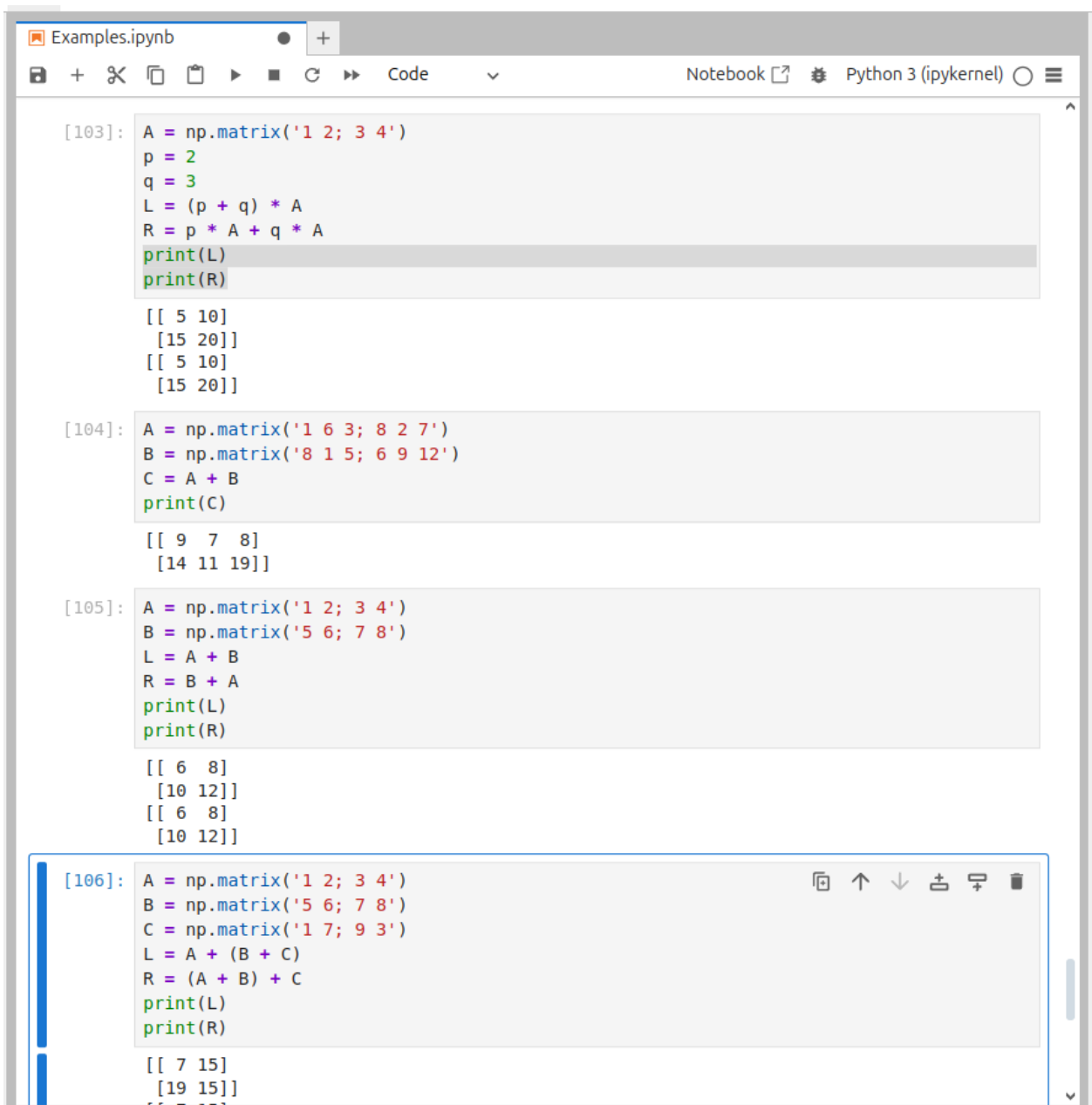
`[[0 0]  
 [0 0]]  
 [[0 0]  
 [0 0]]`

[103]: `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]]`

Рисунок 17. Свойства умножения матриц





```
[103]: 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]]

[104]: 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]]

[105]: 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]]

[106]: 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]]
```

Рисунок 18. Свойства сложения матриц

The screenshot shows a Jupyter Notebook interface with the following content:

- Cell [436 572]:

```
[436 572]]
```
- Cell [109]:

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

Output:  
`[[1. 0. 0. 0.]`  
`[0. 1. 0. 0.]`  
`[0. 0. 1. 0.]`  
`[0. 0. 0. 1.]]`
- Cell [110]:

```
[110]: rank = np.linalg.matrix_rank(m_eye)
print(rank)
```

Output:  
`4`
- Cell [111]:

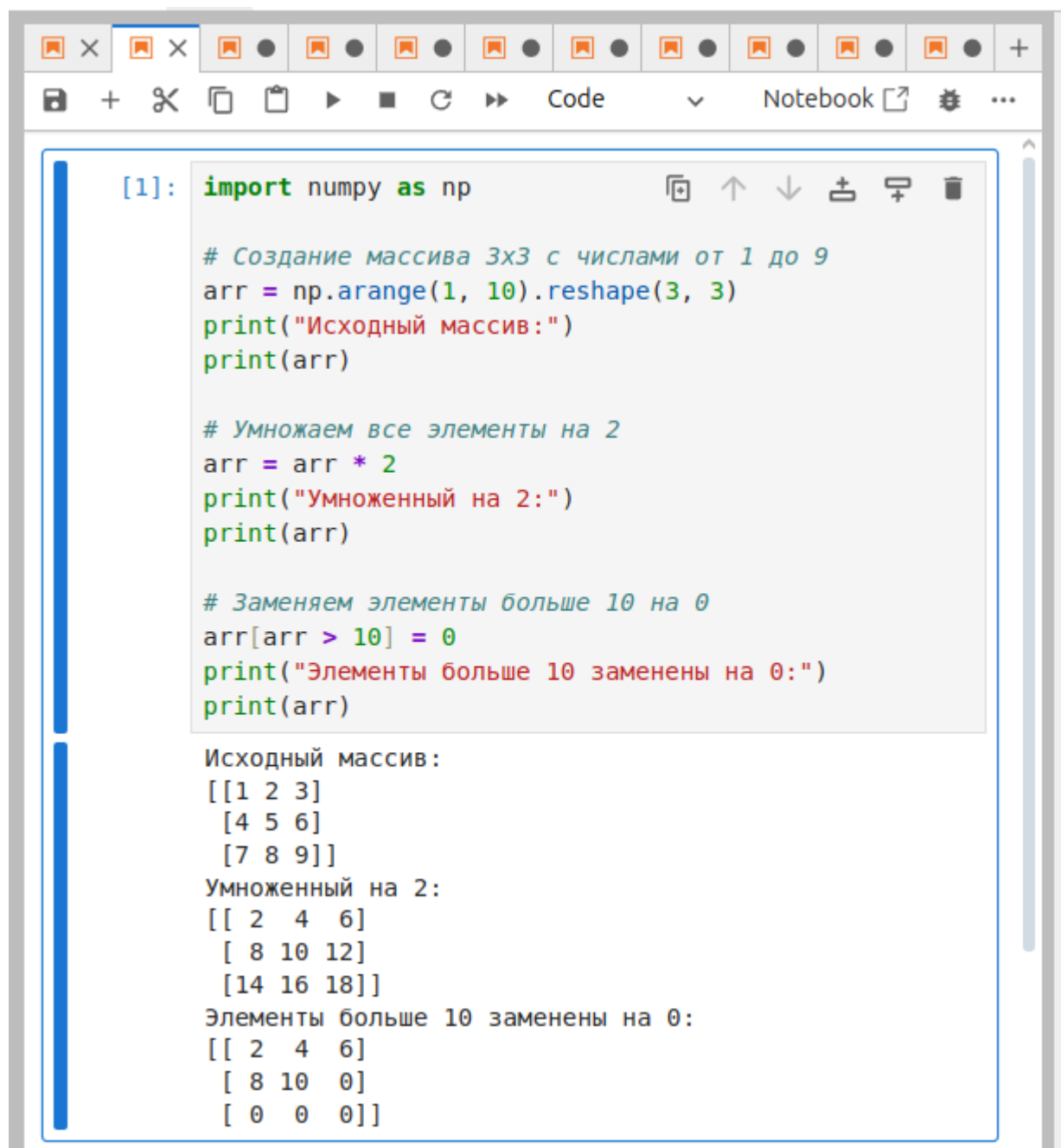
```
[111]: m_eye[3][3] = 0
print(m_eye)
```

Output:  
`[[1. 0. 0. 0.]`  
`[0. 1. 0. 0.]`  
`[0. 0. 1. 0.]`  
`[0. 0. 0. 0.]]`
- Cell [112]:

```
[112]: rank = np.linalg.matrix_rank(m_eye)
print(rank)
```

Output:  
`3`

Рисунок 19. Ранг матрицы



The image shows a Jupyter Notebook window with a toolbar at the top containing icons for file operations, code execution, and view toggling. The code cell is labeled [1]: and contains the following Python code:

```
[1]: import numpy as np

# Создание массива 3x3 с числами от 1 до 9
arr = np.arange(1, 10).reshape(3, 3)
print("Исходный массив:")
print(arr)

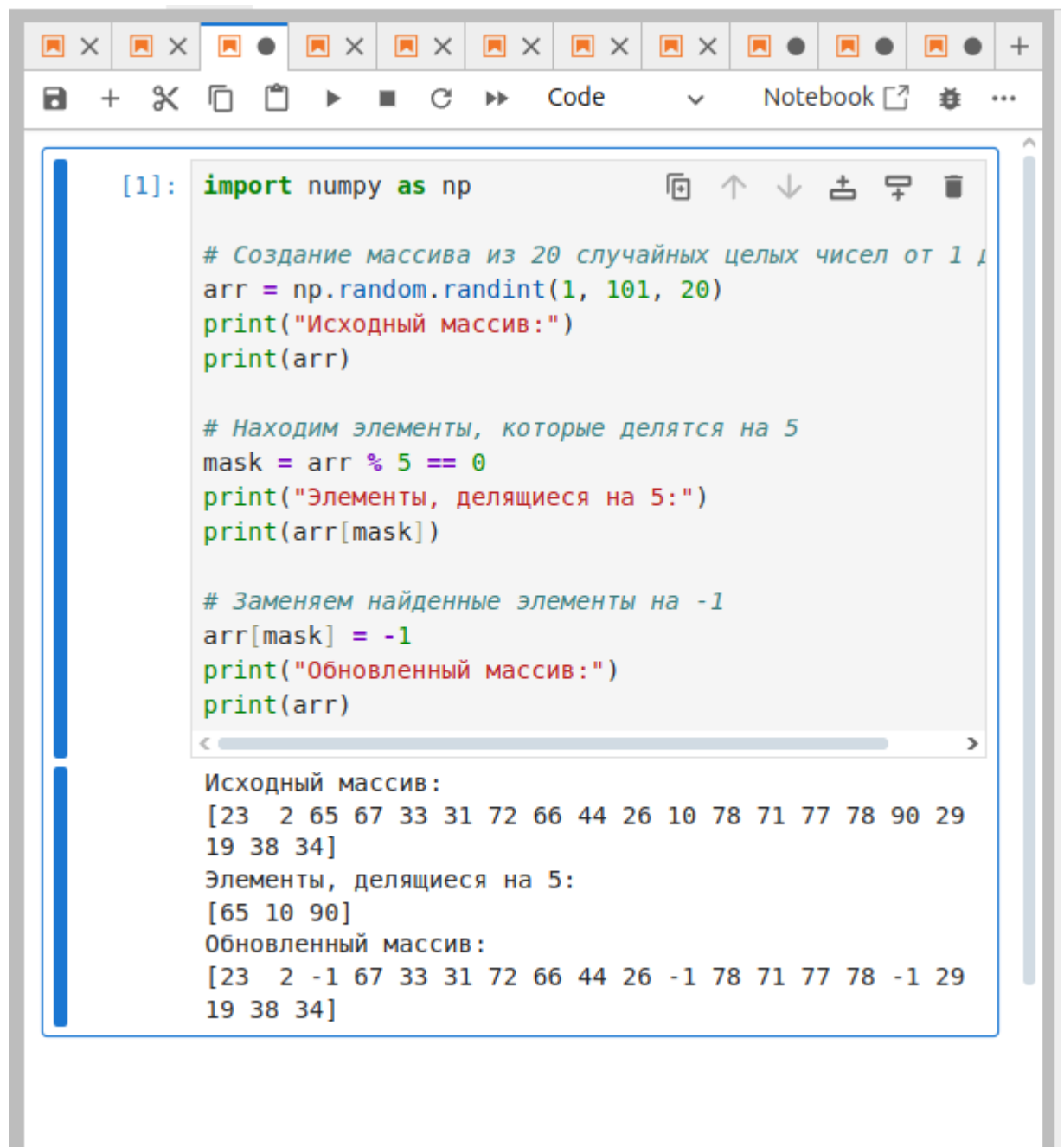
# Умножаем все элементы на 2
arr = arr * 2
print("Умноженный на 2:")
print(arr)

# Заменяем элементы больше 10 на 0
arr[arr > 10] = 0
print("Элементы больше 10 заменены на 0:")
print(arr)
```

The output of the code is displayed below the code cell:

```
Исходный массив:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Умноженный на 2:
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
Элементы больше 10 заменены на 0:
[[ 2  4  6]
 [ 8 10  0]
 [ 0  0  0]]
```

Рисунок 20. Практическое задание 1



The image shows a Jupyter Notebook window with a toolbar at the top containing icons for file operations, code execution, and help. The main area contains a code cell with the following Python code:

```
[1]: import numpy as np

# Создание массива из 20 случайных целых чисел от 1 до 100
arr = np.random.randint(1, 101, 20)
print("Исходный массив:")
print(arr)

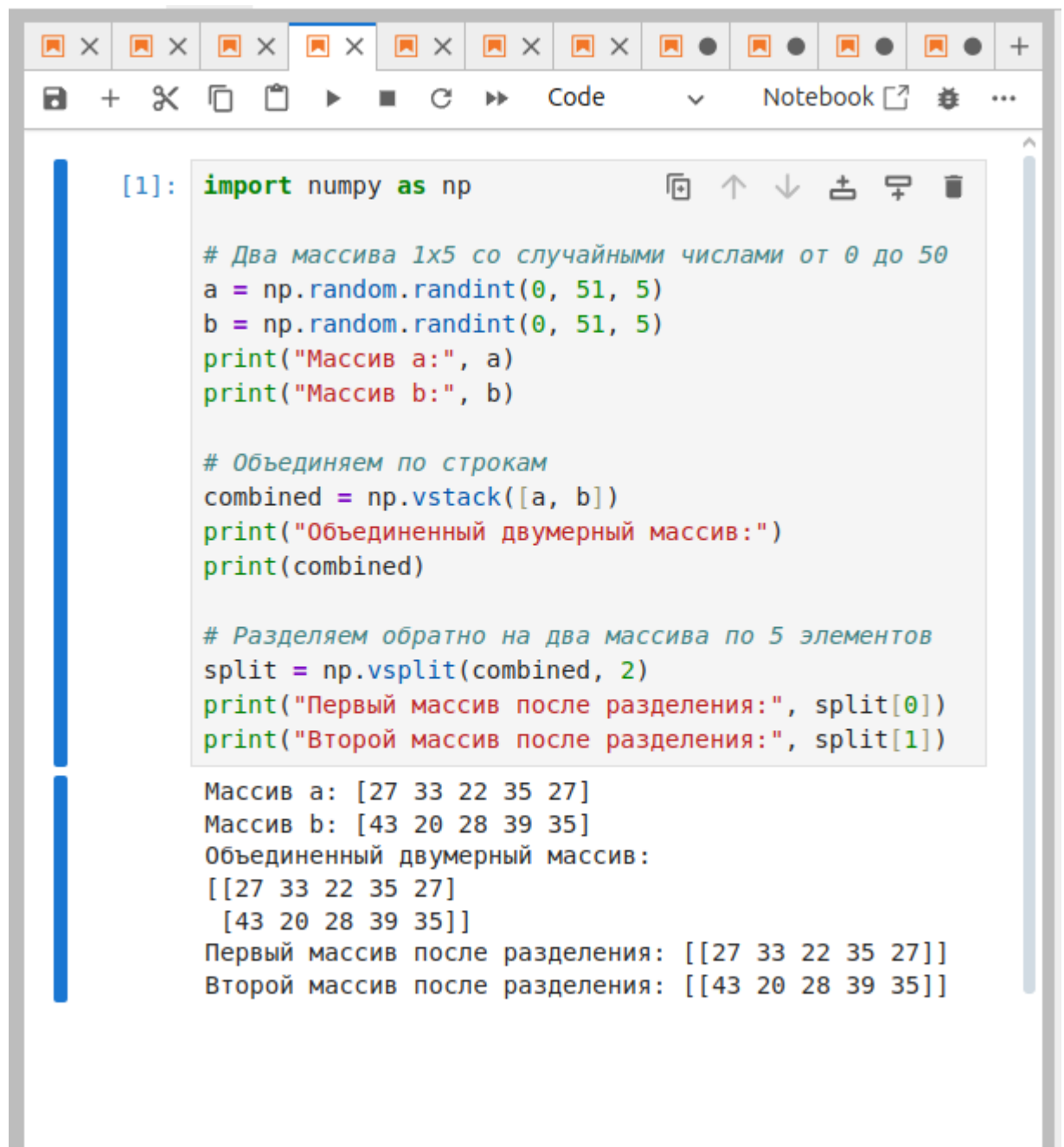
# Находим элементы, которые делятся на 5
mask = arr % 5 == 0
print("Элементы, делящиеся на 5:")
print(arr[mask])

# Заменяем найденные элементы на -1
arr[mask] = -1
print("Обновленный массив:")
print(arr)
```

The output of the code cell is displayed below the code:

```
Исходный массив:
[23  2 65 67 33 31 72 66 44 26 10 78 71 77 78 90 29
 19 38 34]
Элементы, делящиеся на 5:
[65 10 90]
Обновленный массив:
[23  2 -1 67 33 31 72 66 44 26 -1 78 71 77 78 -1 29
 19 38 34]
```

Рисунок 21. Практическое задание 2



The image shows a Jupyter Notebook window with a toolbar at the top containing icons for file operations, execution, and help. The main area displays a code cell with the following Python code:

```
[1]: import numpy as np

# Два массива 1x5 со случайными числами от 0 до 50
a = np.random.randint(0, 51, 5)
b = np.random.randint(0, 51, 5)
print("Массив a:", a)
print("Массив b:", b)

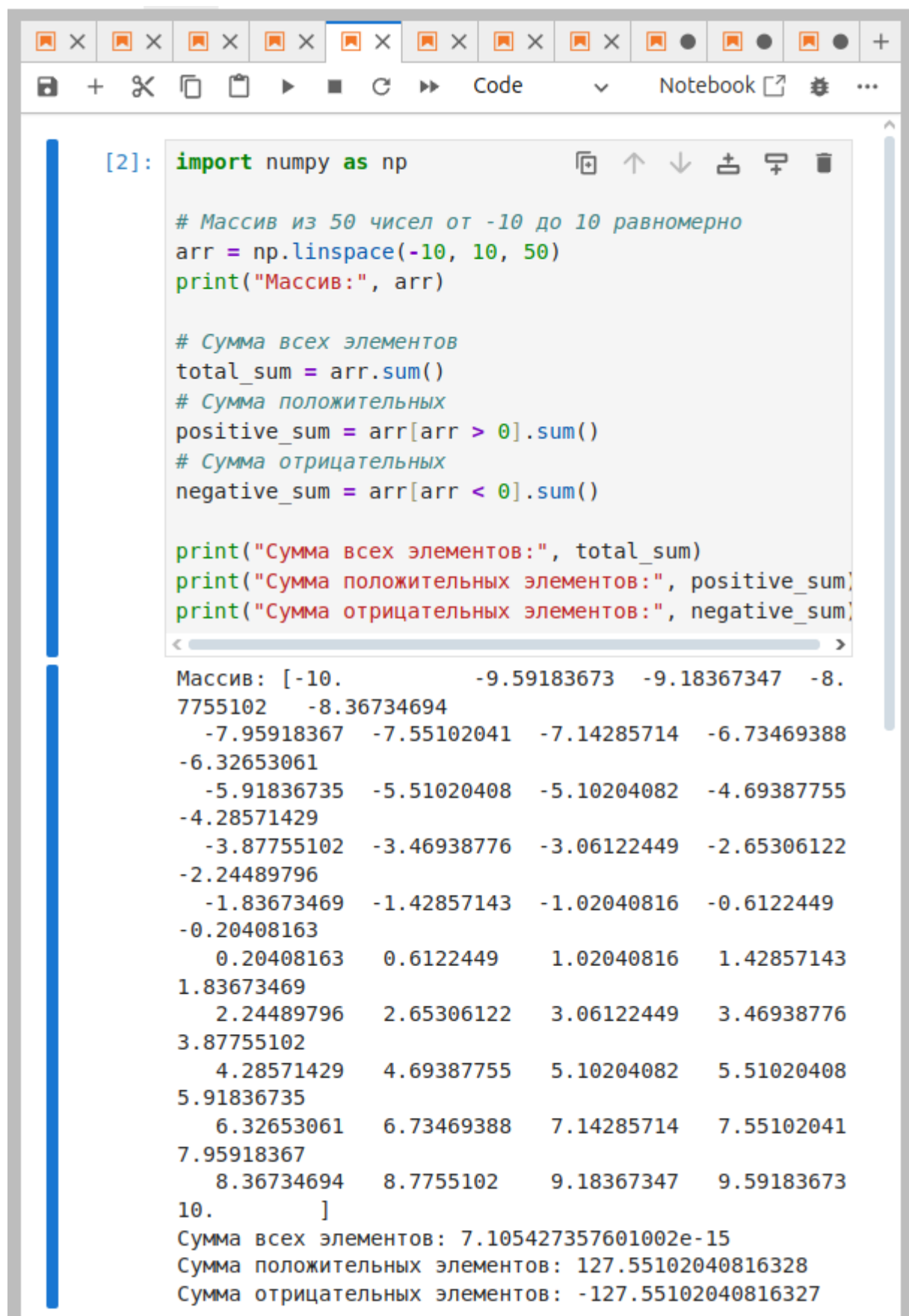
# Объединяем по строкам
combined = np.vstack([a, b])
print("Объединенный двумерный массив:")
print(combined)

# Разделяем обратно на два массива по 5 элементов
split = np.vsplit(combined, 2)
print("Первый массив после разделения:", split[0])
print("Второй массив после разделения:", split[1])
```

The output of the code cell is as follows:

```
Массив a: [27 33 22 35 27]
Массив b: [43 20 28 39 35]
Объединенный двумерный массив:
[[27 33 22 35 27]
 [43 20 28 39 35]]
Первый массив после разделения: [[27 33 22 35 27]]
Второй массив после разделения: [[43 20 28 39 35]]
```

Рисунок 22. Практическое задание 3



The image shows a Jupyter Notebook window with a toolbar at the top containing icons for file operations, execution, and view toggling. The code cell is labeled [2]: and contains the following Python code:

```
[2]: import numpy as np

# Массив из 50 чисел от -10 до 10 равномерно
arr = np.linspace(-10, 10, 50)
print("Массив:", arr)

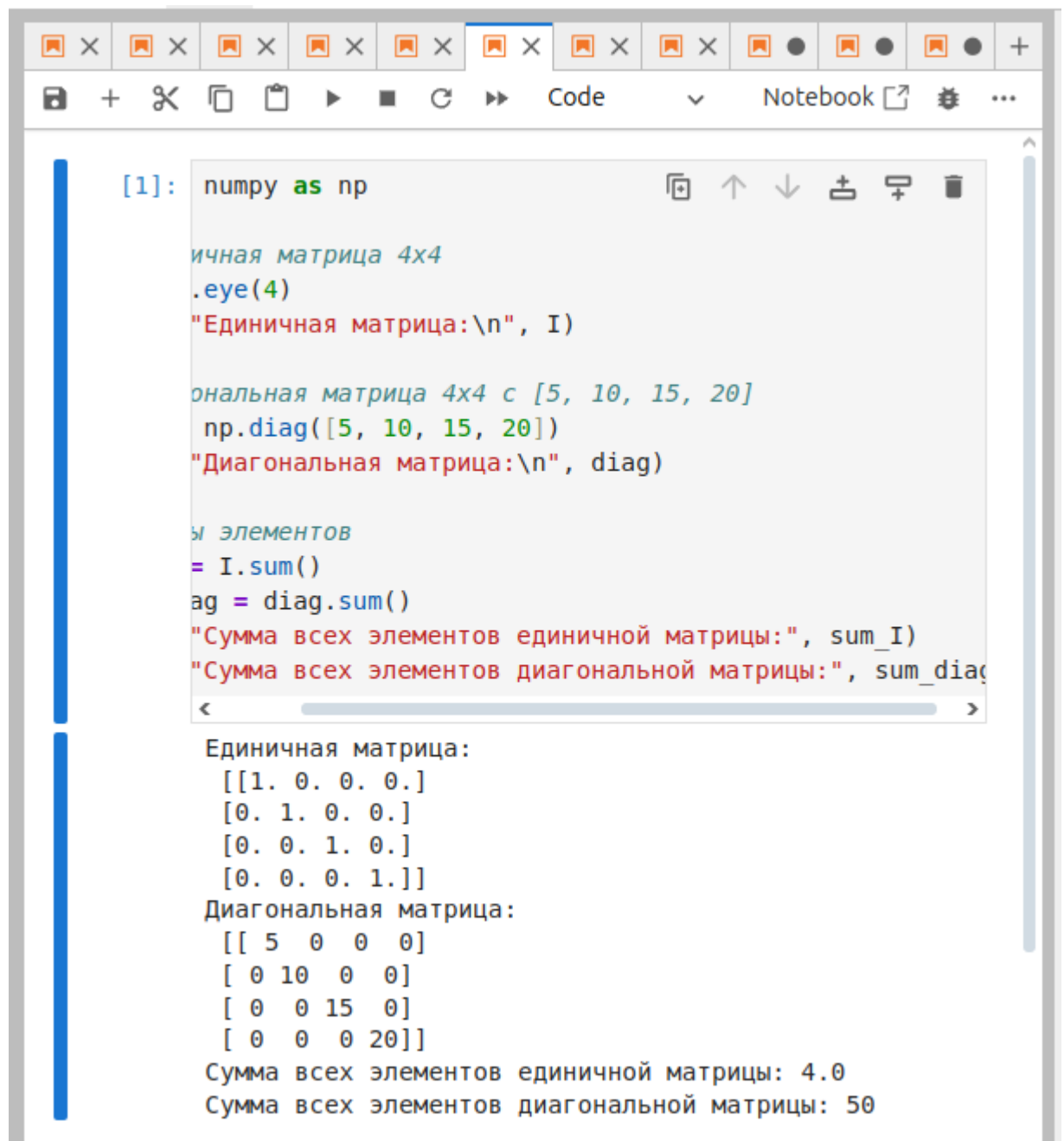
# Сумма всех элементов
total_sum = arr.sum()
# Сумма положительных
positive_sum = arr[arr > 0].sum()
# Сумма отрицательных
negative_sum = arr[arr < 0].sum()

print("Сумма всех элементов:", total_sum)
print("Сумма положительных элементов:", positive_sum)
print("Сумма отрицательных элементов:", negative_sum)
```

The output of the code is displayed below the cell:

```
Массив: [-10.          -9.59183673 -9.18367347 -8.7755102
 -8.36734694
 -7.95918367 -7.55102041 -7.14285714 -6.73469388
 -6.32653061
 -5.91836735 -5.51020408 -5.10204082 -4.69387755
 -4.28571429
 -3.87755102 -3.46938776 -3.06122449 -2.65306122
 -2.24489796
 -1.83673469 -1.42857143 -1.02040816 -0.6122449
 -0.20408163
  0.20408163  0.6122449   1.02040816  1.42857143
  1.83673469
  2.24489796  2.65306122  3.06122449  3.46938776
  3.87755102
  4.28571429  4.69387755  5.10204082  5.51020408
  5.91836735
  6.32653061  6.73469388  7.14285714  7.55102041
  7.95918367
  8.36734694  8.7755102   9.18367347  9.59183673
 10.         ]
Сумма всех элементов: 7.105427357601002e-15
Сумма положительных элементов: 127.55102040816328
Сумма отрицательных элементов: -127.55102040816327
```

Рисунок 23. Практическое задание 4



The image shows a Jupyter Notebook window with a toolbar at the top containing icons for file operations, code execution, and viewing options. The code cell is labeled [1]: and contains the following Python code:

```
import numpy as np

# Единичная матрица 4x4
I = np.eye(4)
print("Единичная матрица:\n", I)

# Диагональная матрица 4x4 с [5, 10, 15, 20]
diag = np.diag([5, 10, 15, 20])
print("Диагональная матрица:\n", diag)

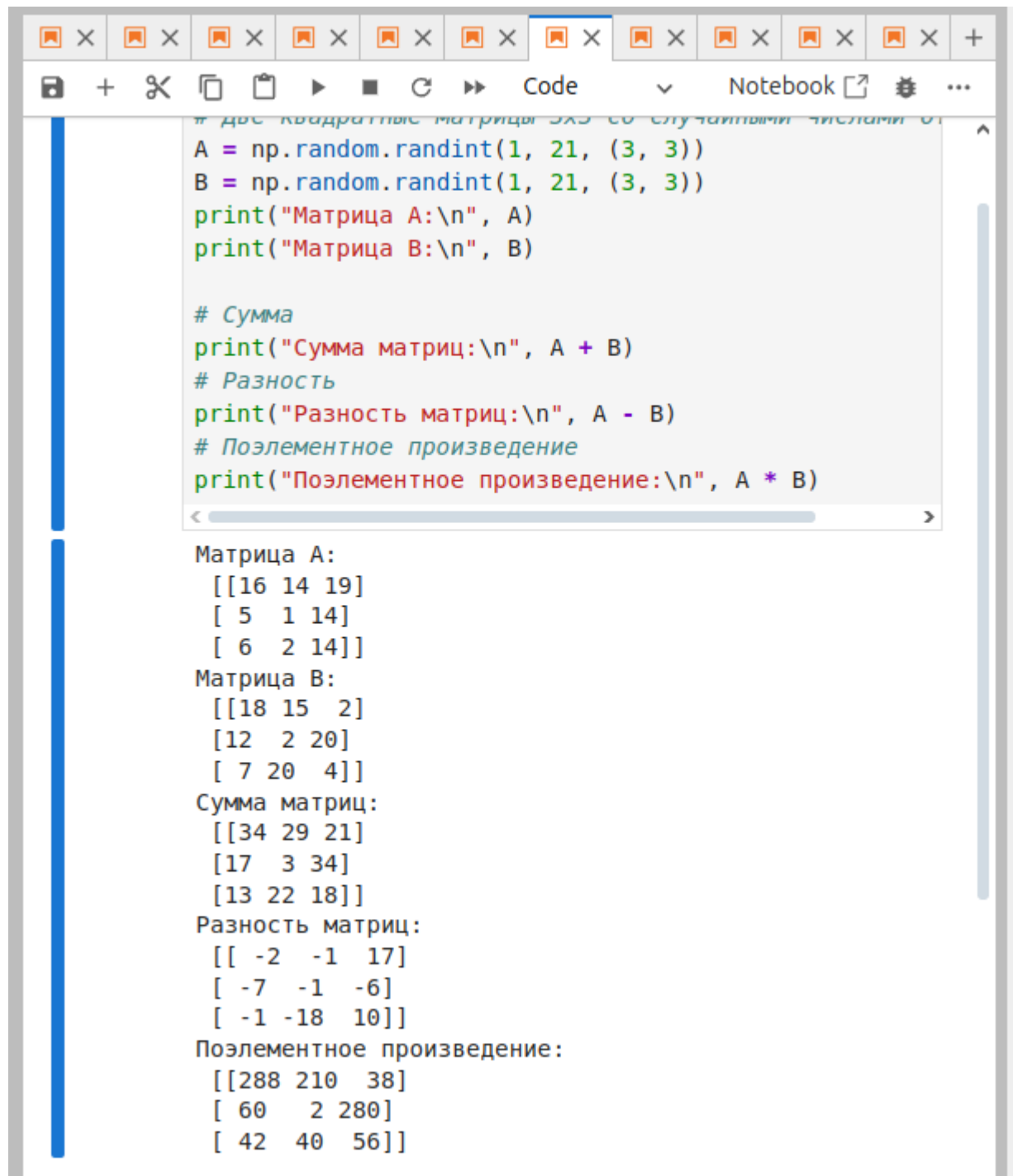
# Сумма элементов
sum_I = I.sum()
sum_diag = diag.sum()

print("Сумма всех элементов единичной матрицы:", sum_I)
print("Сумма всех элементов диагональной матрицы:", sum_diag)
```

The output of the code cell is displayed below the code:

```
Единичная матрица:
[[1.  0.  0.  0.]
 [0.  1.  0.  0.]
 [0.  0.  1.  0.]
 [0.  0.  0.  1.]]
Диагональная матрица:
[[ 5  0  0  0]
 [ 0 10  0  0]
 [ 0  0 15  0]
 [ 0  0  0 20]]
Сумма всех элементов единичной матрицы: 4.0
Сумма всех элементов диагональной матрицы: 50
```

Рисунок 24. Практическое задание 5



The image shows a Jupyter Notebook interface with a code editor and an output area. The code defines two 3x3 matrices A and B using NumPy's random.randint function. It then calculates and prints the sum, difference, and element-wise product of these matrices. The output area displays the resulting 3x3 arrays for each operation.

```
# две квадратные матрицы 3x3 со случайными числами от 0 до 20
A = np.random.randint(1, 21, (3, 3))
B = np.random.randint(1, 21, (3, 3))
print("Матрица A:\n", A)
print("Матрица B:\n", B)

# Сумма
print("Сумма матриц:\n", A + B)
# Разность
print("Разность матриц:\n", A - B)
# Поэлементное произведение
print("Поэлементное произведение:\n", A * B)
```

Матрица A:

```
[[16 14 19]
 [ 5  1 14]
 [ 6  2 14]]
```

Матрица B:

```
[[18 15  2]
 [12  2 20]
 [ 7 20  4]]
```

Сумма матриц:

```
[[34 29 21]
 [17  3 34]
 [13 22 18]]
```

Разность матриц:

```
[[ -2  -1 17]
 [ -7  -1 -6]
 [ -1 -18 10]]
```

Поэлементное произведение:

```
[[288 210  38]
 [ 60   2 280]
 [ 42  40  56]]
```

Рисунок 25. Практическое задание 6



The image shows a Jupyter Notebook window with a toolbar at the top containing icons for file operations, code execution, and help. The code cell contains the following Python code:

```
[1]: import numpy as np

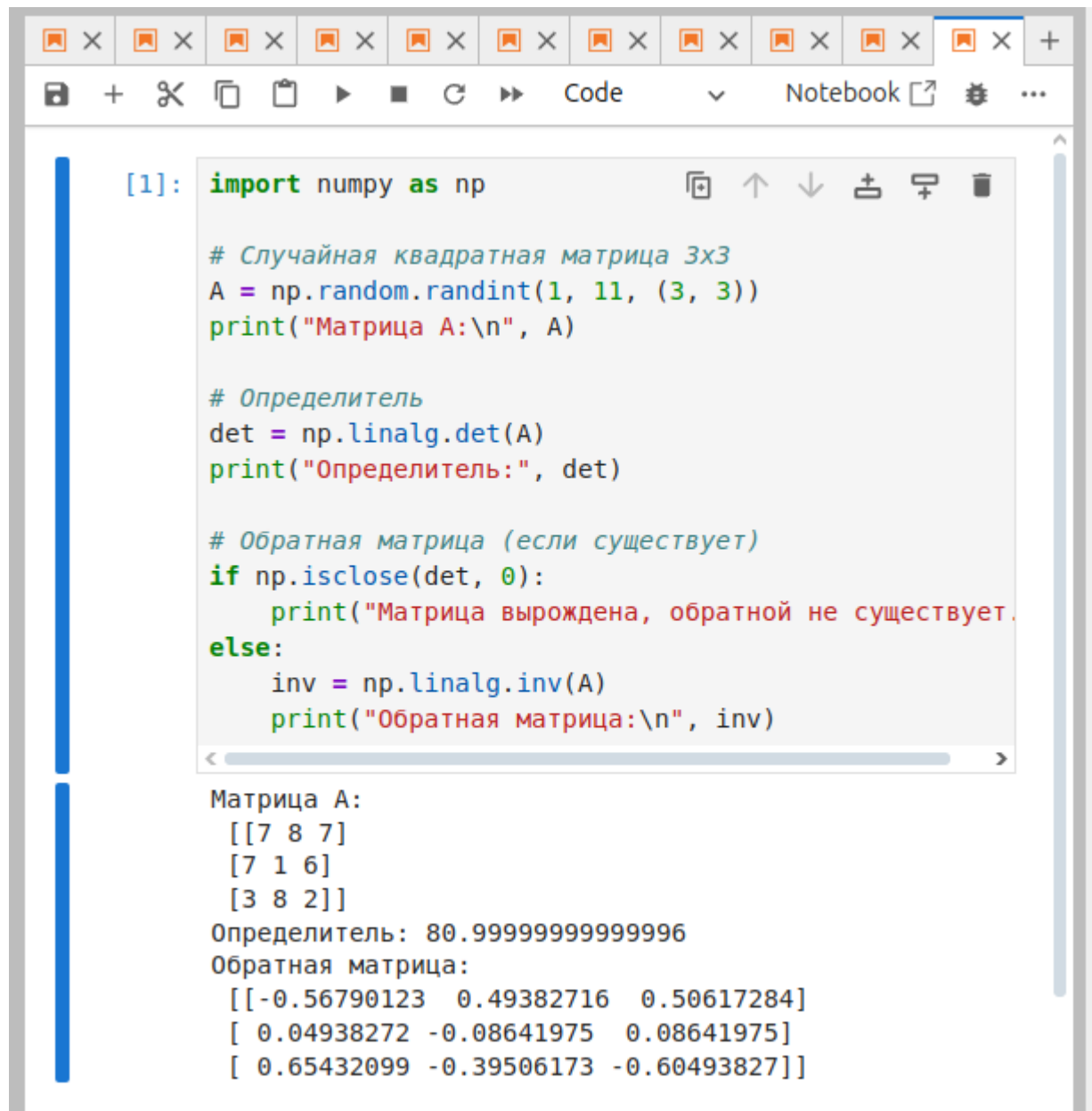
# Первая матрица 2x3
A = np.random.randint(1, 11, (2, 3))
# Вторая матрица 3x2
B = np.random.randint(1, 11, (3, 2))
print("Матрица A:\n", A)
print("Матрица B:\n", B)

# Матричное умножение
product = A @ B # или np.dot(A, B)
print("Результат умножения:\n", product)
```

The output of the code cell is as follows:

```
Матрица A:
[[8 6 8]
 [4 7 9]]
Матрица B:
[[9 2]
 [7 3]
 [7 8]]
Результат умножения:
[[170 98]
 [148 101]]
```

Рисунок 26. Практическое задание 7



The image shows a Jupyter Notebook window with a toolbar at the top containing icons for file operations, execution, and navigation. The code cell is labeled [1]: and contains the following Python code:

```
[1]: import numpy as np

# Случайная квадратная матрица 3x3
A = np.random.randint(1, 11, (3, 3))
print("Матрица A:\n", A)

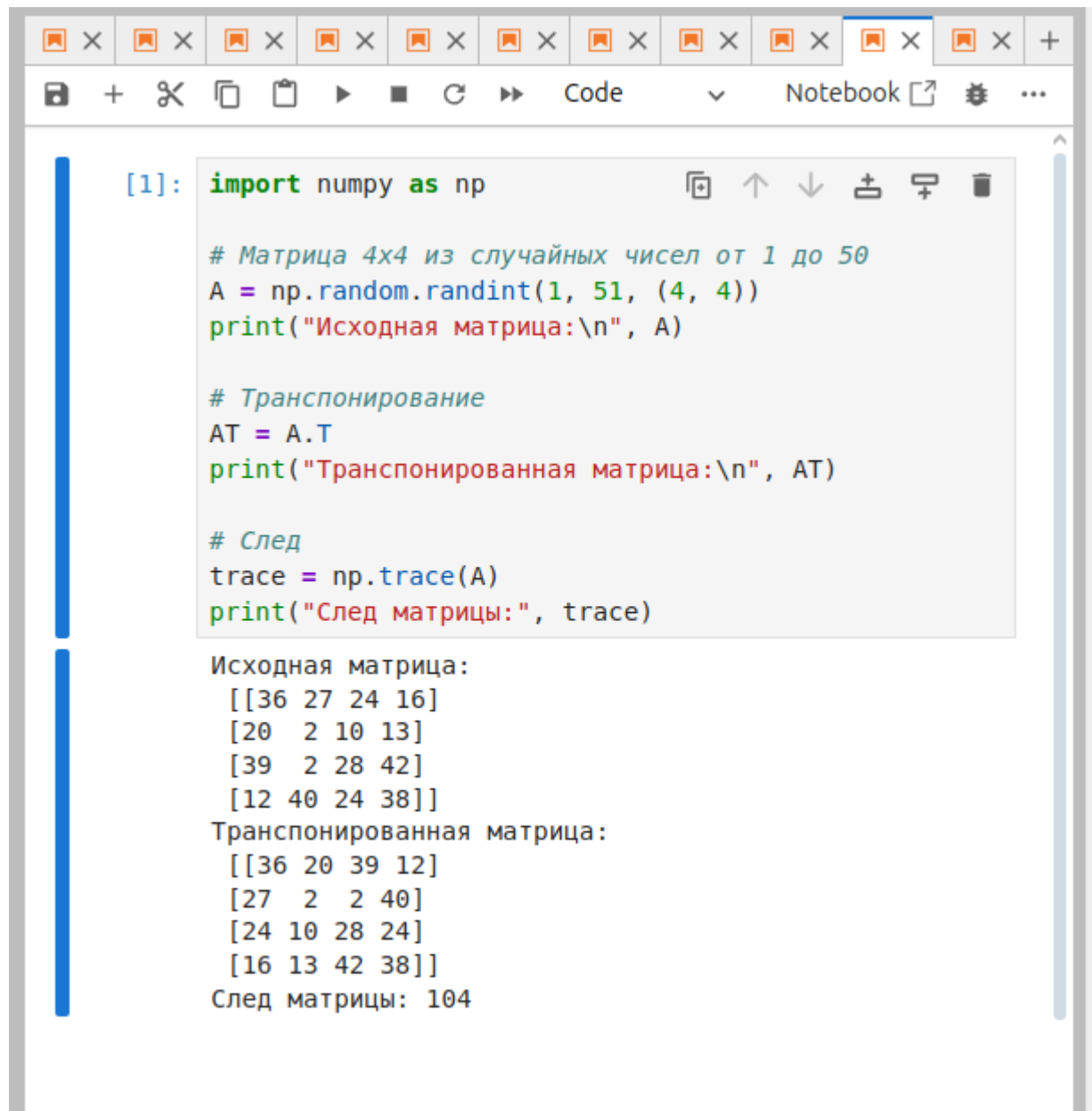
# Определитель
det = np.linalg.det(A)
print("Определитель:", det)

# Обратная матрица (если существует)
if np.isclose(det, 0):
    print("Матрица вырождена, обратной не существует.")
else:
    inv = np.linalg.inv(A)
    print("Обратная матрица:\n", inv)
```

The output of the code cell is displayed below the code:

```
Матрица A:
[[7 8 7]
 [7 1 6]
 [3 8 2]]
Определитель: 80.99999999999996
Обратная матрица:
[[-0.56790123  0.49382716  0.50617284]
 [ 0.04938272 -0.08641975  0.08641975]
 [ 0.65432099 -0.39506173 -0.60493827]]
```

Рисунок 27. Практическое задание 8



The image shows a Jupyter Notebook window with a toolbar at the top containing icons for file operations, code execution, and viewing options. The code cell is labeled [1]: and contains the following Python code:

```
[1]: import numpy as np

# Матрица 4x4 из случайных чисел от 1 до 50
A = np.random.randint(1, 51, (4, 4))
print("Исходная матрица:\n", A)

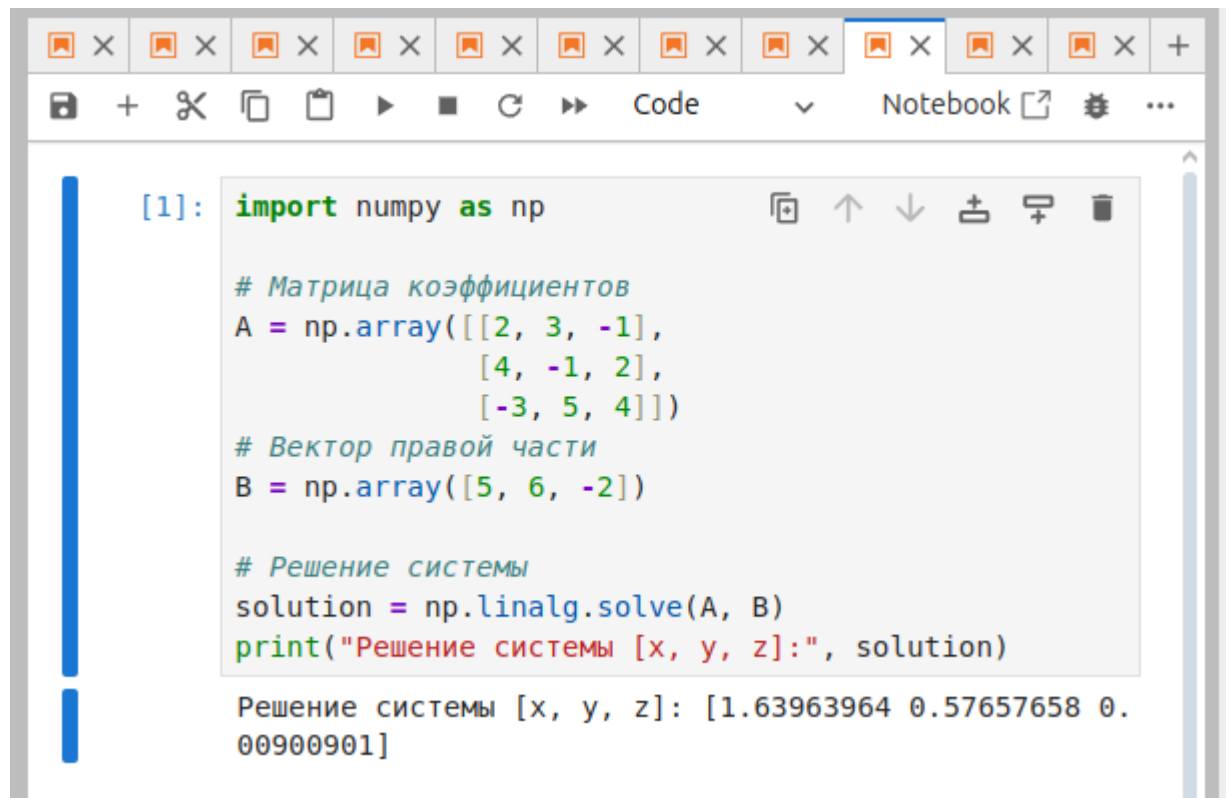
# Транспонирование
AT = A.T
print("Транспонированная матрица:\n", AT)

# След
trace = np.trace(A)
print("След матрицы:", trace)
```

The output of the code cell is displayed below the code:

```
Исходная матрица:
[[36 27 24 16]
 [20  2 10 13]
 [39  2 28 42]
 [12 40 24 38]]
Транспонированная матрица:
[[36 20 39 12]
 [27  2  2 40]
 [24 10 28 24]
 [16 13 42 38]]
След матрицы: 104
```

Рисунок 28. Практическое задание 9



The image shows a Jupyter Notebook window with a toolbar at the top containing icons for file operations, code execution, and help. The main area displays a code cell with the following Python code:

```
[1]: import numpy as np

# Матрица коэффициентов
A = np.array([[2, 3, -1],
              [4, -1, 2],
              [-3, 5, 4]])
# Вектор правой части
B = np.array([5, 6, -2])

# Решение системы
solution = np.linalg.solve(A, B)
print("Решение системы [x, y, z]:", solution)
```

Below the code, the output of the execution is displayed:

```
Решение системы [x, y, z]: [1.63963964 0.57657658 0.00900901]
```

Рисунок 29. Практическое задание 10

```
# Метод Крамера
det_A = np.linalg.det(A)
A_x = A.copy()
A_x[:, 0] = B
det_x = np.linalg.det(A_x)

A_y = A.copy()
A_y[:, 1] = B
det_y = np.linalg.det(A_y)

A_z = A.copy()
A_z[:, 2] = B
det_z = np.linalg.det(A_z)

x_kramer = np.array([det_x/det_A, det_y/det_A, det_z/det_A])

# np.linalg.solve
x_solve = np.linalg.solve(A, B)

print("Матричный метод:", x_mat)
print("Метод Крамера: ", x_kramer)
print("np.linalg.solve:", x_solve)
```

Матричный метод: [128.57142857 178.57142857 192.85714286]  
Метод Крамера: [128.57142857 178.57142857 192.85714286]  
np.linalg.solve: [128.57142857 178.57142857 192.85714286]

Рисунок 30. Индивидуальное задание

### Ответы на контрольные вопросы:

#### 1. Каково назначение библиотеки NumPy?

NumPy — это основная библиотека для научных вычислений в Python. Она предоставляет удобные структуры данных (главным образом массивы), высокоэффективные математические операции, средства для линейной

алгебры, статистики, работы с большими наборами данных и интеграции с другими научными пакетами.

## 2. Что такое массивы `ndarray`?

`ndarray` — основной тип данных в NumPy, многомерный массив, который хранит элементы одного типа (например, только `float` или только `int`). Он похож на обычные списки Python, но поддерживает операции над всеми элементами сразу, быстрые вычисления, работу с матрицами и т.д.

## 3. Как осуществляется доступ к частям многомерного массива?

Доступ осуществляется с помощью индексов и срезов. Для двумерного массива: `arr[0, 1]` — элемент первой строки, второго столбца. Для выделения целого столбца или строки: `arr[0, :]` (вся первая строка), `arr[:, 1]` (весь второй столбец).

## 4. Как осуществляется расчет статистик по данным?

NumPy содержит встроенные методы для подсчета статистик:

- `arr.sum()` — сумма,
- `arr.mean()` — среднее,
- `arr.std()` — стандартное отклонение,
- `arr.min()/arr.max()` — минимум/максимум,
- и др.

Часто есть параметр `axis`, указывающий направление подсчета.

## 5. Как выполняется выборка данных из массивов `ndarray`?

С помощью индексов, срезов (`arr[1:5]`, `arr[:, 0]`), булевых масок (`arr[arr > 5]`), списка индексов (`arr[[0, 2, 3]]`).

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

- **Вектор:** одномерный массив (`np.array([1, 2, 3])`)
- **Матрица:** двумерный массив (`np.array([[1, 2], [3, 4]])`)
- **Нулевая матрица:** `np.zeros((m, n))`
- **Единичная матрица:** `np.eye(n)`
- **Диагональная:** `np.diag([1, 2, 3])`
- **Матрица с одинаковыми элементами:** `np.full((m, n), val)`

### 7. Как выполняется транспонирование матриц?

С помощью свойства `.T`:

```
A = np.array([[1, 2], [3, 4]])  
A_T = A.T
```

### 8. Приведите свойства операции транспонирования матриц.

- Транспонирование транспонированной матрицы возвращает исходную:  
 $(A.T).T = A$
- $(A + B).T = A.T + B.T$
- $(kA).T = kA.T$
- $(AB).T = B.T @ A.T$

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

- `A.T`
- `np.transpose(A)`

### 10. Какие существуют основные действия над матрицами?

- Сложение, вычитание
- Умножение на число
- Поэлементное умножение
- Матричное умножение
- Транспонирование
- Определитель, обратная матрица

### 11. Как осуществляется умножение матрицы на число?

Обычное умножение:

```
B = 3 * A
```

### 12. Какие свойства операции умножения матрицы на число?

- $k(A + B) = kA + kB$
- $(k + l)A = kA + lA$
- $k(lA) = (kl)A$

### 13. Как осуществляется операции сложения и вычитания матриц?

Элемент к элементу, если одинаковый размер:

$$C = A + B$$

$$D = A - B$$

### 14. Каковы свойства операций сложения и вычитания матриц?

- Переместительность:  $A + B = B + A$
- Ассоциативность:  $(A + B) + C = A + (B + C)$
- Нейтральность:  $A + 0 = A$

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

- Операторы  $+$ ,  $-$
- Функции `np.add(A, B)`, `np.subtract(A, B)`

### 16. Как осуществляется операция умножения матриц?

С помощью `@` или `np.dot(A, B)` для матричного (не поэлементного!) произведения.

### 17. Каковы свойства операции умножения матриц?

- Не переместительно (обычно):  $AB \neq BA$
- Ассоциативно:  $(AB)C = A(BC)$
- Дистрибутивно относительно сложения:  $A(B + C) = AB + AC$

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

- `A @ B`
- `np.dot(A, B)`
- `np.matmul(A, B)`

### 19. Что такое определитель матрицы? Каковы свойства определителя матрицы?

Определитель — числовая характеристика квадратной матрицы, определяющая её обратимость, объём преобразования, решение СЛУ. Свойства:

- $\det(A \cdot B) = \det(A) \cdot \det(B)$
- $\det(A.T) = \det(A)$



- $\det(\lambda A) = \lambda^n \det(A)$ , где  $n$  — размер
- Матрица обратима  $\Leftrightarrow \det \neq 0$

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

- `np.linalg.det(A)`

**21. Что такое обратная матрица? Какой алгоритм нахождения обратной матрицы?**

Обратная матрица  $A^{-1}$  — такая, что  $A \cdot A^{-1} = I$ .

Стандартный алгоритм: через элементарные преобразования или через присоединённую матрицу/определитель.

**22. Каковы свойства обратной матрицы?**

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

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

- `np.linalg.inv(A)`

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

**Метод Крамера:**

Для  $Ax = b$ , где  $A$  — квадратная матрица,

- `det_A = np.linalg.det(A)`
- Для каждого  $x_i$ : заменить  $i$ -й столбец  $A$  на  $b$ , вычислить `det_i = np.linalg.det(A_i)`,  $x_i = \text{det\_i} / \text{det\_A}$

**Алгоритм на Python:**

```
import numpy as np
def solve_kramer(A, b):
    n = len(b)
    det_A = np.linalg.det(A)
    if np.isclose(det_A, 0):
        raise ValueError("Матрица вырождена")
    result = []
    for i in range(n):
        Ai = A.copy()
```

```
    Ai[:, i] = b
    result.append(np.linalg.det(Ai) / det_A)
return np.array(result)
```

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

**Матричный метод:**

$x = A^{-1} \cdot b$  (если определитель  $\neq 0$ )

**На Python:**

```
import numpy as np
A = np.array([[...]]) # матрица коэффициентов
b = np.array([...])   # вектор правой части
x = np.linalg.inv(A) @ b
```

**Вывод:** в результате практики освоены ключевые приёмы работы с массивами и матрицами в NumPy: создание, изменение, объединение, статистика, индексация, маски. Реализованы основные матричные операции — транспонирование, вычисление определителя, получение обратной матрицы, поэлементные и матричные произведения. Проведено сравнение решений систем линейных уравнений разными методами: матричным, Крамера и через стандартную функцию NumPy, все методы дали одинаковый результат. NumPy позволил быстро и точно решить задачи, продемонстрировав высокую эффективность для анализа и вычислений в Python. Освоенные инструменты являются прочной базой для дальнейшего развития в программном анализе и моделировании данных.