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 5

Выполнил:
Беков Шамиль Расулович
2 курс, группа ИВТ-б-о-23-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии 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.

Порядок выполнения работы:

Ссылка на GitHub: https://github.com/ShamilBekov/Laba-2

1. Выполнили операцию сложения над двумя массивами, векторизацию.

1. Создание массива:

```
[18]: import numpy as np
      a = np.array([1, 2, 3])
      b = np.array([4, 5, 6])
      result = a + b
      print(result)
```

```
[5 7 9]
```

Рисунок 1. Операция сложения над двумя массивами

2. Использовали статистические функции.

2.2. Статистические функции:

```
[26]: data = np.array([1, 2, 3, 4, 5])
      mean = np.mean(data)
      std_dev = np.std(data)
      print(mean, "\n", std_dev)
```

```
3.0
 1.4142135623730951
```

Рисунок 2. Статистические функции в NumPy

3. Выполнили различные операции над матрицей.

4.1. Доступ к элементам массива

[34]:
```
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]]
```

4.2. Элемент матрицы с заданными координатами.

[37]:
```
print(m[1, 0])
```

```
5
```

4.2. Строка массива.

[40]:
```
print(m[1,:])
```

```
[[5 6 7 8]]
```

4.3. Столбец массива.

[43]:
```
print(m[:, 2])
```

```
[[3]
 [7]
 [5]]
```

Рисунок 3. Выполнение различных операций над матрицей

4.4. Часть строки массива.

```
[46]: print(m[1, 2:])
```

```
[[7 8]]
```

4.5. Часть столбца массива.

```
[32]: print(m[0:2,1])
```

```
[[2]
 [6]]
```

4.6. Непрерывная часть массива.

```
[35]: print(m[0:2,1:3])
```

```
[[2 3]
 [6 7]]
```

4.7. Произвольные стольцы/строки массива.

```
[38]: cols=[0,1,3]
print(m[:,cols])
```

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

Рисунок 4. Выполнение различных операций над матрицей

4. Произвели расчет статистик массива.

5. Расчет статистик по данным в массиве.

[41]:
```
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]]
```

6. Размерность массива.

[44]:
```
print(m.shape)
```

```
(3, 4)
```

7. Вызов функции расчета статистик.

[47]:
```
print(np.max(m))
```

```
9
```

8. Расчет статистик по строкам или столбцам массива.

[50]:
```
print(m.max())
```

```
9
```

Рисунок 5. Функции расчета статистик в NumPy

9. Поиск максимального элемента в каждой строке.

```
[53]: print(m.max(axis=1))
```

```
[[4]
 [8]
 [9]]
```

10. Вычисление статистик по столбцам.

```
[56]: print(m.max(axis=0))
```

```
[[9 6 7 8]]
```

11. Функции (методы) для расчета статистик в Numpy.

```
[59]: print(m.shape)
      print(m.max())
      print(np.max(m))
```

```
(3, 4)
9
9
```

Рисунок 6. Функции расчета статистик в NumPy

5. Использовали boolean массив.

12. Использование boolean массива для доступа к ndarray.

12.1. Создание массивов.

[63]:
```
nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
letters = np.array(['a', 'b', 'c', 'd', 'a', 'e', 'b'])
print(nums)
print(letters)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
['a' 'b' 'c' 'd' 'a' 'e' 'b']
```

[65]:
```
a
```

[65]:
```
array([1, 2, 3])
```

[67]:
```
b=5>7
print(b)
```

```
False
```

12.2. Построим на базе массивов ndarray массивы с элементами типа boolean.

[70]:
```
less_then_5 = nums < 5
print(less_then_5)
pos_a = letters == 'a'
print(pos_a)
```

```
[ True  True  True  True False False False False False False]
[ True False False False  True False False]
```

Рисунок 7. Операции над boolean массивом

6. Построили логическую матрицу с условием.

[80]:
```
mod_m = np.logical_and(m>=3, m<=7)
print(mod_m)
print(m[mod_m])
```

```
[[False False  True  True]
 [ True  True  True False]
 [False False  True  True]]
[[3 4 5 6 7 5 7]]
```

Рисунок 8. Логическая матрица

7. Модифицировали массив с помощью boolean.

[87]:
```
m[m > 7] = 25
print(m)
```

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

Рисунок 9. Модифицирование массива с помощью boolean

8. Воспользовались функцией arange().

13. Дополнительные функции.

13.1. Создание вектора из целых чисел от 0 до stop.

```
[91]: print(np.arange(10))
```

```
[0 1 2 3 4 5 6 7 8 9]
```

13.2. Задаем интервал.

```
[94]: print(np.arange(5, 12))
```

```
[ 5  6  7  8  9 10 11]
```

13.2. Определение интервала чисел и шаг.

```
[97]: print(np.arange(1, 5, 0.5))
```

```
[1.  1.5 2.  2.5 3.  3.5 4.  4.5]
```

Рисунок 10. Функция arange()

9. Использовали функцию matrix() для задания матрицы.

14. Задаем матрицу с помощью matrix.

```
[100]: a = [[1, 2], [3, 4]]
print(np.matrix(a))
```

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

14.1. Вариант с массивом Numpy.

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

```
[[5 6]
 [7 8]]
```

Рисунок 11. Функция matrix()

10. Использовали функции zeros() и eye() для создания нулевой матрицы и единичной квадратной матрицы соответственно.

15.1. Создание нулевой матрицы.

```
[110]: print(np.zeros((3, 4)))
```

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

15.2. Создание единичной матрицы.

```
[113]: print(np.eye(3))
```

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

Рисунок 12. Функции zeros() и eye()

11. Использовали функцию ravel() для преобразования матрицы в одномерный вектор. Применили параметр order.

16. Преобразование матрицы в одномерный вектор.

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

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

16.1. Примение функцию ravel() - фукнция, которая отвечает за порядок построения одномерного массива.

```
[119]: print(np.ravel(A))
```

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

16.2. Если у order есть 'C', то массив будет собираться из строк исходной матрицы.

```
[122]: print(np.ravel(A, order='C'))
```

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

16.3. Если у order есть 'F', то в качестве элементов для сборки будут выступать столбы матрицы.

```
[125]: print(np.ravel(A, order='F'))
```

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

Рисунок 13. Функция ravel()

12.Использовали функции random.rand() и where().

17. Возвращение один из двух заданных элементов в зависимости от условия.

[130]:
```
a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print(np.where(a % 2 == 0, a * 10, a / 10))
```

```
[ 0.   0.1 20.   0.3 40.   0.5 60.   0.7 80.   0.9]
```

[141]:
```
a = np.random.rand(10)
print(a)
print(np.where(a > 0.5, True, False))
print(np.where(a > 0.5, 1, -1))
printx = np.linspace(0, 1, 5)
print(x)
y = np.linspace(0, 2, 5)
print(y)
```

```
[0.70305281 0.75022111 0.26820493 0.14875246 0.01528184 0.86726913
 0.92124504 0.78030073 0.02289251 0.76382316]
[ True  True False False False  True  True  True False  True]
[ 1  1 -1 -1 -1  1  1  1 -1  1]
[0.   0.25 0.5  0.75 1.  ]
[0.  0.5 1.  1.5 2. ]
```

Рисунок 14. Функции random и where

13. Использовали функцию meshgrid для создания матрицы координат.

18. Функция np.meshgrid()- позваляет получить матрицу координат из координатных векторов.

18.1. Создадим два вектора.

[136]:
```
x = np.linspace(0, 1, 5)
print(x)
y = np.linspace(0, 2, 5)
print(y)
```

```
[0.   0.25 0.5  0.75 1.  ]
[0.  0.5 1.  1.5 2. ]
```

18.2. Построим матрицу координат с помощью meshgrid.

[139]:
```
xg, yg = np.meshgrid(x,y)
print(xg)
print(yg)
```

```
[[0.   0.25 0.5  0.75 1.  ]
 [0.   0.25 0.5  0.75 1.  ]
 [0.   0.25 0.5  0.75 1.  ]
 [0.   0.25 0.5  0.75 1.  ]
 [0.   0.25 0.5  0.75 1.  ]]
[[0.  0.  0.  0.  0. ]
 [0.5 0.5 0.5 0.5 0.5]
 [1.  1.  1.  1.  1. ]
 [1.5 1.5 1.5 1.5 1.5]
 [2.  2.  2.  2.  2. ]]
```

Рисунок 15. Функция meshgrid()

14. Визуализировали полученные матрицы.

```
[956]: import matplotlib.pyplot as plt
       %matplotlib inline
       plt.plot(xg, yg, color="r", marker="*", linestyle="none")
       plt.show()
```

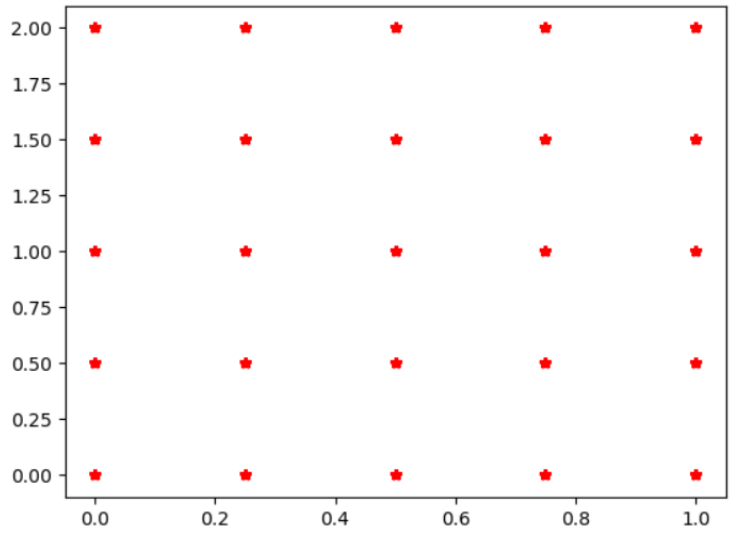

Рисунок 16. Визуализация данных в виде графика

15. Использовали функцию random.permutation для генерации натуральных чисел и перемешали переданные данные.

```
)43]: print(np.random.permutation(7))
      a = ['a', 'b', 'c', 'd', 'e']
      print(np.random.permutation(a))
```

```
[0 5 6 1 4 2 3]
['d' 'e' 'a' 'c' 'b']
```

Рисунок 17. Функция meshgrid

16. Создали несколько различных векторов-строк.

21. Вектор-строка.

[431]:
```
v_hor_np = np.array([1, 2])
print(v_hor_np )
```

```
[1 2]
```

21.1. Создадим нулевую вектор-строку с размером 5.

[434]:
```
v_hor_zeros_v1=np.zeros((5,))
print(v_hor_zeros_v1)
```

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

21.2. Построим единичную вектор-строку.

[436]:
```
v_hor_one_v1 = np.ones((5,))
print(v_hor_one_v1)
v_hor_one_v2 = np.ones((1, 5))
print(v_hor_one_v2)
```

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

Рисунок 18. Способы задать векторы-строки

17. Создали различные векторы-столбцы.

22. Вектор столбец.

[438]:
```
v_vert_np = np.array([[1], [2]])
print(v_vert_np)
```

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

22.1. Нулевой вектор столбец.

[442]:
```
v_vert_zeros = np.zeros((5, 1))
print(v_vert_zeros)
```

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

22.2. Единичный вектор столбец.

[444]:
```
v_vert_ones = np.ones((5, 1))
print(v_vert_ones)
```

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

Рисунок 19. Способы задать вектор-столбец

18. Использовали различные методы для создания квадратных матриц.

23. Создание квадратной матрицы с помощью метода array().

```
[446]: 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]]
```

24. Создание списка.

```
[448]: 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]]
```

25. Построение обекта типа matrix с помощью одноименного метода.

```
[450]: 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 20. Способы задать квадратную матрицу

19. Создали диагональную матрицу различными методами.

26. Диагональная матрица.

[453]:
```
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]]
```

27. Создадим матрицу размера 3 3.

[455]:
```
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]]
```

27.1. Извлечем ее главную диагональ.

[457]:
```
diag = np.diag(m_sqr_mx)
print(diag)
```

```
[1 5 9]
```

27.1. Построим диагональную матрицу на базе полученной диагонали.

[459]:
```
m_diag_np = np.diag(np.diag(m_sqr_mx))
print(m_diag_np)
```

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

Рисунок 21. Способы задать диагональную матрицу

20. Создали единичную матрицу различными методами.

28. Единичная матрица.

28.1. Создание единичной мтарицы на базе списка.

```
[462]: 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]]
```

28.2. Использование функции eye() для построения единичной матрицы.

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

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

28.3. Использование функции identity() для построения единичной матрицы.

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

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

Рисунок 22. Создание единичной матрицы

21. Создали нулевую матрицу.

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

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

Рисунок 23. Нулевая матрица

22. Выполнили процесс транспонирования матрицы различными методами.

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

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

31.2. Транспонируем матрицу с помощью метода transpose():

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

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

Рисунок 24. Транспонирование матриц

23. Умножили матрицу на число.

33.1. Умножение матрицы на число.

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

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

Рисунок 25. Умножение матрицы на число

24. Выполнили сложение матриц.

33.2. Сложение матриц.

```
[514]: 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]]
```

Рисунок 26. Сложение матриц

25. Выполнили умножение матриц.

33.3. Умножение матриц.

[523]:
```
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]]
```

Рисунок 27. Умножение матриц

26. Посчитали определитель матрицы.

34. Определитель матрицы.

34.1. Создадим матрицу 3x3.

[537]:
```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
```

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

34.2. Вычислим определитель с помощью функции det() и пакета linalg.

[539]:
```
print(np.linalg.det(A))
```

```
-14.000000000000009
```

Рисунок 28. Определитель матрицы

27. Посчитали обратную матрицу.

35. Обратная матрица

35.1. Для получения обратной матрицы матрицы используем функцию inv().

[561]:
```
A = np.matrix('1 -3; 2 5')
A_inv = np.linalg.inv(A)
print(A_inv)
```

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

Рисунок 29. Обратная матрица

28. Посчитали ранг матрицы.

36. Ранг матрицы.

36.1. Создадим единичную матрицу:

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

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

36.2. Ранг матрицы равен количеству ее столбцов (или строк), в нашем случае ранг будет равен четырем для его вычисления на Python воспользуемся функцией matrix_rank():

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

```
4
```

Рисунок 30. Ранг матрицы

29. Создайте массив NumPy размером 3×3, содержащий числа от 1 до 9. Умножьте все элементы массива на 2, а затем замените все элементы больше 10 на 0. Выведите итоговый массив.

```
[3]: import numpy as np
     A = np.array([[1,2,3],
                   [4,5,6],
                   [7,8,9]])
     A=A*2
     for i in range(3):
         for j in range(3):
             if (A[i][j]>10):
                 A[i][j]=0
     A
```

```
[3]: array([[ 2,  4,  6],
            [ 8, 10,  0],
            [ 0,  0,  0]])
```

Рисунок 31. Результат задания 1

30. Создайте массив NumPy из 20 случайных целых чисел от 1 до 100. Найдите и выведите все элементы, которые делятся на 5 без остатка. Затем замените их на -1 и выведите обновленный массив.

```
[190]: A = np.random.randint(1, 101, size=(20))
print("Исходный массив: ",A)
print("Числа, которые делятся на 5 без остатка:")
for i in range(20):
    if (A[i]%5==0):
        print(A[i])
        A[i]=-1
print("Массив, в котором элементы, которые делятся на 5 без остатка, заменены на -1: ",A)
```

```
Исходный массив:  [18  6 22 88  3 50 42 15 94 47 67 75 19 40 72 78 59 91 32 80]
Числа, которые делятся на 5 без остатка:
50
15
75
40
80
Массив, в котором элементы, которые делятся на 5 без остатка, заменены на -1:  [18  6 22 88  3 -1 42 -1 94 47 67 -1 19 -1 72 78 59 91 32 -1]
```

Рисунок 32. Результат задания 2

31. Создайте два массива NumPy размером 1×5, заполненные случайными числами от 0 до 50:

- Объедините эти массивы в один двумерный массив (по строкам).
- Разделите полученный массив на два массива, каждый из которых содержит 5 элементов.

```
[7]: A = np.random.randint(0,51)
B = np.random.randint(0,51)
print("Массивы 1x5, заполненные случайными числами от 0 до 50:")
print("A =",A);print("B =", B)
matrix = np.vstack((A,B))
print("Массивы A и B, обьединенные в один двумерный массив: ")
print(matrix)
A_new, B_new = np.vsplit(matrix, 2)
print("Двумерный массив, который был разделен обратно на два массива: ")
print("A =",A_new);print("B =", B_new)
```

```
Массивы 1x5, заполненные случайными числами от 0 до 50:
A = 28
B = 16
Массивы A и B, обьединенные в один двумерный массив:
[[28]
 [16]]
Двумерный массив, который был разделен обратно на два массива:
A = [[28]]
B = [[16]]
```

Рисунок 33. Результат задания 3

32. Создайте массив из 50 чисел, равномерно распределенных от -10 до 10. Вычислите сумму всех элементов, сумму положительных элементов и сумму отрицательных элементов. Выведите результаты.

[25]:
```
A = np.random.uniform(-10, 10, 50)
C = np.sum(A)
print("Массив из 50 чисел, равномерно распределенных от -10 до 10:\n", A)
print("Сумма всех элементов массива: ", C)
B = np.sum(A[A>0])
print("Сумма всех положительных элементов массива: ", B)
D = np.sum(A[A<0])
print("Сумма всех отрицательных элементов массива: ", D)
```

```
Массив из 50 чисел, равномерно распределенных от -10 до 10:
 [-3.04808618  5.77730068  0.73807768 -8.05731552  1.60705552  5.79984385
 -5.92565126  9.61903352  5.26405655 -4.950066    4.72879139 -1.44237931
  0.80228986  9.57082972  8.06686139  5.49478119  2.69195975 -8.6959409
 -9.3968371  -4.50666585 -1.94557818 -9.56312302  9.96547389 -4.73382486
  4.73850507  3.51775598 -7.92803748 -7.59597257 -0.97320504 -9.02104322
  1.79743826  2.32388716  8.85228084 -9.57510468 -2.56073903 -6.21598143
  2.41916251  3.55261564 -2.34903775  3.13052564  5.42738325 -4.55233852
 -3.12811804 -7.55588191  5.19800276  7.79417337  2.88497311 -2.07516092
  5.50483474  3.59402245]
Сумма всех элементов массива:  5.0658269970045
Сумма всех положительных элементов массива:  130.86191576180937
Сумма всех отрицательных элементов массива:  -125.7960887648049
```

Рисунок 34. Результат задания 4

33. Создайте указанные матрицы и найдите сумму всех элементов каждой из этих матриц. Сравните результаты.

- Единичную матрицу размером 4×4.
- Диагональную матрицу размером 4×4 с диагональными элементами [5, 10, 15, 20] (не использовать циклы).

[314]:
```
A = np.eye(4)
print("Единичная матрица размером 4 порядка:\n",A)
B = np.diag([5, 10, 15, 20])
print("Диагональная матрица 4 порядка с диагональными элементами (5,10,15,20):\n", B)
k = np.sum(A)
k1 = np.sum(B)
print("Сумма всех элементов единичной матрицы:",k)
print("Сумма всех элементов диагональной матрицы:",k1)
```

```
Единичная матрица размером 4 порядка:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Диагональная матрица 4 порядка с диагональными элементами (5,10,15,20):
 [[ 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 35. Результат задания 5

34. Создайте две квадратные матрицы NumPy размером 3×3, заполненные случайными целыми числами от 1 до 20. Вычислите и выведите:

- Их сумму.
- Их разность.
- Их поэлементное произведение.

[356]:
```
A = np.random.randint(1,21, size=(3,3))
B = np.random.randint(1,21, size=(3,3))
print("A = \n",A)
print("B = \n",B)
C = A + B
print("A + B =\n",C)
D = A - B
print("A - B =\n",D)
E = A*B
print("A * B (поэлементное произведение) =\n",E)
```

```
A =
 [[ 8 15 16]
 [ 6 12  9]
 [16 17 14]]
B =
 [[ 1 13 14]
 [ 2  8 11]
 [19 18 17]]
A + B =
 [[ 9 28 30]
 [ 8 20 20]
 [35 35 31]]
A - B =
 [[ 7  2  2]
 [ 4  4 -2]
 [-3 -1 -3]]
A * B (поэлементное произведение) =
 [[  8 195 224]
 [ 12  96  99]
 [304 306 238]]
```

Рисунок 36. Результат задания 6

35. Создайте две матрицы NumPy и выполните матричное умножение (@ или np.dot). Выведите результат:

- Первую размером 2×3, заполненную случайными числами от 1 до 10.
- Вторую размером 3×2, заполненную случайными числами от 1 до 10.

[33]:
```
A = np.random.randint(1,11, size=(2,3))
B = np.random.randint(1,11, size=(3,2))
print("A = \n",A)
print("B = \n",B)
C = np.dot(A,B)
G = A@B
print("Произведение матрицы A на матрицу B с помощью dot:\n",C)
print("Произведение матрицы A на матрицу B с помощью @:\n",G)
```

```
A =
 [[ 3  1 10]
 [ 4  5  8]]
B =
 [[ 4  3]
 [ 2 10]
 [ 3  8]]
Произведение матрицы A на матрицу B с помощью dot:
 [[ 44  99]
 [ 50 126]]
Произведение матрицы A на матрицу B с помощью @:
 [[ 44  99]
 [ 50 126]]
```

Рисунок 37. Результат задания 7

36. Создайте случайную квадратную матрицу 3×3. Найдите и выведите:

- Определитель этой матрицы.
- Обратную матрицу (если существует, иначе выведите сообщение, что матрица вырождена).

[469]:
```
A = np.random.rand(3, 3)
print("Матрица A =\n",A)
C = np.linalg.det(A)
print("det A = ", C)
D = np.linalg.inv(A)
if (C != 0):
    print("A⁻¹ = \n", D)
else:
    print("det A = 0, обратной матрицы не существует!")
```

```
Матрица A =
 [[0.50650186 0.62886753 0.07887878]
 [0.20213945 0.65439707 0.82874358]
 [0.77868256 0.71258994 0.17992485]]
det A =  0.11464172628121103
A⁻¹ =
 [[-4.12425788 -0.49668365  4.09582006]
 [ 5.31183787  0.25916168 -3.52241429]
 [-3.18841194  1.12315165  1.78237372]]
```

Рисунок 38. Результат задания 8

37. Создайте матрицу NumPy размером 4×4, содержащую случайные целые числа от 1 до 50. Выведите:

- Исходную матрицу.
- Транспонированную матрицу.
- След матрицы (сумму элементов на главной диагонали).

```
36]: A = np.random.randint(1,51, size=(4,4))
print("Исходная матрица: \n",A)
print("Транспонированная матрица: \n", np.transpose(A))
print("След матрицы: ", np.trace(A))
```

```
Исходная матрица:
 [[47  5  3  6]
 [ 9 50 32 30]
 [41 46  4  7]
 [29 20 15 44]]
Транспонированная матрица:
 [[47  9 41 29]
 [ 5 50 46 20]
 [ 3 32  4 15]
 [ 6 30  7 44]]
След матрицы:  145
```

Рисунок 39. Результат задания 9

38. Решите систему линейных уравнений вида:

$$\begin{cases} 2x + 3y - z = 5 \\ 4x - y + 2z = 6 \\ -3x + 5y + 4z = -2 \end{cases}$$

Используйте матричное представление Ax = B , где A – матрица коэффициентов, x – вектор неизвестных, B – вектор правой части. Решите систему с помощью np.linalg.solve и выведите результат.

```
[525]: A = np.array([[2,3,-1],
                    [4,-1,2],
                    [-3,5,4]])
B = np.array([[5],[6],[-2]])
print("Ответ:\n",np.linalg.solve(A,B))
```

```
Ответ:
 [[1.63963964]
 [0.57657658]
 [0.00900901]]
```

Рисунок 40. Результат задания 10

39. Решите индивидуальное задание согласно варианту. Каждое задание предусматривает построение системы линейных уравнений. Решите полученную систему уравнений с использованием библиотеки NumPy. Для решения системы используйте метод Крамера и матричный метод. Сравните полученные результаты, с результатами, полученными с помощью np.linalg.solve.

Условие для варианта 5: Распределение нагрузки на работников. Три сотрудника должны выполнить 100 задач. Второй сотрудник должен выполнить на 10 задач больше, чем первый, а третий — в два раза больше, чем второй. Сколько задач получит каждый сотрудник?

```
A = np.array([
    [1, 1, 1],    # x + y + z = 100
    [-1, 1, 0],    # -x + y = 10
    [0, -2, 1]     # -2y + z = 0
])
B = np.array([100, 10, 0])

n = np.linalg.solve(A, B)
print("Решение СЛУ с помощью np.linalg.solve:")
print(f"x = {n[0]:.0f}, y = {n[1]:.0f}, z = {n[2]:.0f}.")

G = np.linalg.inv(A)
m = G @ B
print("Решение СЛУ с помощью матричного метода:")
print(f"x = {m[0]:.0f}, y = {m[1]:.0f}, z = {m[2]:.0f}.")

det_A = np.linalg.det(A)

A_x = A.copy()
A_x[:, 0] = B
x = np.linalg.det(A_x) / det_A

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

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

k = np.array([x, y, z])
print("Решение СЛУ с помощью метода Крамера:")
print(f"x = {k[0]:.0f}, y = {k[1]:.0f}, z = {k[2]:.0f}.")
```

Рисунок 41. Выполнение индивидуального задания

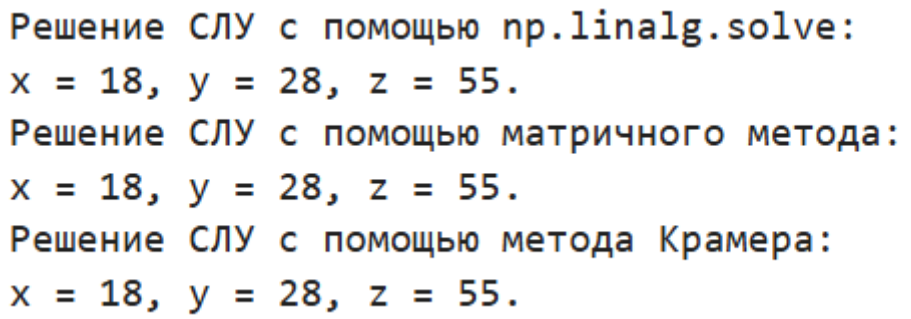

Рисунок 39. Результат выполнения индивидуального задания

Ответы на контрольные вопросы:

1. Каково назначение библиотеки NumPy?

Назначение библиотеки NumPy – выполнение эффективных операций с многомерными массивами, линейной алгебры, статистики и обработки данных.

2. Что такое массивы ndarray?

Массивы ndarray – это основной объект NumPy, представляющий собой многомерные массивы с однородными элементами и быстрыми математическими операциями.

3. Как осуществляется доступ к частям многомерного массива?

Доступ к частям многомерного массива – через индексацию (array[i, j]), срезы (array[:, 1]), логические маски (array[array > 5]) или итерации (for row in array).

4. Как осуществляется расчет статистик по данным?

Расчет статистик – с помощью встроенных функций, например np.mean(), np.median(), np.std(), np.sum(), np.min(), np.max().

5. Как выполняется выборка данных из массивов ndarray?

Выборка данных – через индексацию (arr[1]), срезы (arr[1:4]), логические условия (arr[arr > 10]) или fancy indexing (arr[[0, 2, 4]]).

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

Основные виды матриц и векторов и их создание в Python (NumPy)

- Нулевые матрицы: np.zeros((m, n))

- Единичные матрицы: np.eye(n) или np.identity(n)
- Диагональные матрицы: np.diag([a, b, c])
- Случайные матрицы: np.random.randint(1, 10, (m, n))
- Векторы (1D массивы): np.array([x, y, z])

7. Как выполняется транспонирование матриц?

- Для стандартных матриц используем A.T или np.transpose(A).
- Для многомерных массивов используем np.transpose() или np.swapaxes().

8. Приведите свойства операции транспонирования матриц.

Свойства операции транспонирования матриц

- Дважды транспонированная матрица совпадает с исходной: $(A^T)^T=A$
- Транспонирование суммы: $(A+B)^T=A^T+B^T$
- Транспонирование произведения: $(AB)^T=B^TA^T$
- Транспонирование обратной матрицы: $(A^{-1})^T=(A^T)^{-1}$

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

Средства в NumPy для транспонирования матриц

- .T (быстрый метод): A.T
- np.transpose(A) (более гибкий)
- np.swapaxes(A, 0, 1) (перестановка осей)

10. Какие существуют основные действия над матрицами?

Основные действия над матрицами

- Сложение: A + B
- Вычитание: A - B
- Умножение поэлементное: A * B
- Матричное умножение: A @ B или np.dot(A, B)
- Обратная матрица: np.linalg.inv(A)
- Определитель: np.linalg.det(A)
- След матрицы: np.trace(A)

11. Как осуществляется умножение матрицы на число?

Умножение матрицы на число выполняется поэлементно: каждый элемент матрицы умножается на заданное число.

```
import numpy as np

A = np.array([[1, 2, 3],
              [4, 5, 6]])

k = 3  # Число, на которое умножаем

result = A * k

print(result)
```

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

12. Какие свойства операции умножения матрицы на число?

- Ассоциативность по скалярам: $(a \cdot b) \cdot A = a \cdot (b \cdot A)$
- Дистрибутивность относительно сложения чисел: $(a+b) \cdot A = a \cdot A + b \cdot A$
- Дистрибутивность относительно сложения матриц: $a \cdot (A+B) = a \cdot A + a \cdot B$
- Сохранение нулевой матрицы: $0 \cdot A = 0$
- Умножение на 1 не изменяет матрицу: $1 \cdot A = A$

13. Как осуществляется операции сложения и вычитания матриц?

Сложение и вычитание матриц в NumPy выполняются поэлементно, но матрицы должны быть одинакового размера.

- Используем np.add() для сложения
- Используем np.subtract() для вычитания

```python
import numpy as np

A = np.array([[1, 2, 3],
              [4, 5, 6]])

B = np.array([[6, 5, 4],
              [3, 2, 1]])

sum_matrix = np.add(A, B)

diff_matrix = np.subtract(A, B)

print("Сумма через np.add():\n", sum_matrix)
print("Разность через np.subtract():\n", diff_matrix)
```

```
Сумма через np.add():
 [[7 7 7]
 [7 7 7]]
Разность через np.subtract():
 [[-5 -3 -1]
 [ 1  3  5]]
```

14. Каковы свойства операций сложения и вычитания матриц?

Свойства операций сложения и вычитания матриц

- Коммутативность (для сложения): A+B=B+A
- Ассоциативность (для сложения): (A+B)+C=A+(B+C)
- Существование нулевой матрицы: A+0=A
- Существование противоположной матрицы: A+(−A)=0
- Не коммутативность для вычитания: A−B≠B−A

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

- np.add(A, B) # Сложение
- np.subtract(A, B) # Вычитание

16. Как осуществляется операция умножения матриц?

Матричное умножение выполняется по правилам линейной алгебры, а не поэлементно.

Элемент в позиции (i,j) итоговой матрицы равен скалярному произведению строки из первой матрицы и столбца из второй.

C = A @ B Или np.dot(A, B)

17. Каковы свойства операции умножения матриц?

Свойства операции умножения матриц:

- Не коммутативность: $AB \neq BA$
- Ассоциативность: $(AB)C=A(BC)$
- Дистрибутивность: $A(B+C)=AB+AC$
- Существование единичной матрицы III: $AI = IA =A$
- Перемножение диагональных матриц: $D_1D_2=D_3$ (где D — диагональная матрица).

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

- Оператор @
- C = A @ B
- Функция np.dot()
- C = np.dot(A, B)
- Функция np.matmul() (аналог @, но для многомерных массивов)
- C = np.matmul(A, B)

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

Определитель матрицы — это число, которое характеризует матрицу и её свойства.

Для квадратной матрицы AAA определитель обозначается как det(A) или $|A|$.

Основные свойства определителя:

- Определитель единичной матрицы равен 1: $\det(I)=1$
- Если у матрицы есть строка (или столбец) из нулей, то её определитель равен 0.
- Если две строки (или два столбца) матрицы равны, то её определитель равен 0.
- Если поменять две строки (или два столбца) местами, знак определителя меняется.
- Определитель произведения матриц: $\det(AB)=\det(A)\cdot\det(B)$

- Определитель обратной матрицы: $\det(A^{-1})=1/\det(A)$ (если A невырожденная, т.е. $\det(A)\neq 0$).

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

В NumPy определитель вычисляется через np.linalg.det().

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

Обратная матрица A^{-1} — это такая матрица, которая при умножении на исходную матрицу даёт единичную матрицу:

$AA^{-1}=A^{-1}A=I$

Обратная матрица существует, только если определитель матрицы $\det(A)\neq 0$

Алгоритм нахождения обратной матрицы:

1. Проверяем, что матрица квадратная (n × n).
2. Вычисляем определитель det(A).

- Если $\det(A)=0$, матрица вырождена, обратной матрицы не существует.

3. Находим матрицу алгебраических дополнений.
4. Транспонируем матрицу алгебраических дополнений (находим присоединённую матрицу A^{*}).
5. Вычисляем обратную матрицу: $A^{-1}=1/\det(A)*A^{*}$

22. Каковы свойства обратной матрицы?

Свойства обратной матрицы

- Если матрица обратима, то её определитель ненулевой: $\det(A)\neq 0$
- Обратная от обратной — это исходная матрица: $(A^{-1})^{-1}=A$
- Обратная произведения — произведение обратных в обратном порядке: $(AB)^{-1}=B^{-1}A^{-1}$
- Обратная транспонированной матрицы — транспонированная обратной: $(A^{T})^{-1}=(A^{-1})^{T}$

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

Для нахождения обратной матрицы используется np.linalg.inv(A)

Если матрица вырождена (det(A)=0 вызовет ошибку.

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

```python
import numpy as np

A = np.array([[2, -1, 3],
              [1,  1, 1],
              [3, -3, 2]])

# Вектор правой части
B = np.array([5, 6, 2])

# Вычисляем определитель
det_A = np.linalg.det(A)

# Проверяем, существует ли единственное решение
if np.isclose(det_A, 0):
    print("Система не имеет единственного решения")
else:
    # Вычисляем детерминанты A_i (заменяя столбцы)
    A1, A2, A3 = A.copy(), A.copy(), A.copy()
    A1[:, 0], A2[:, 1], A3[:, 2] = B, B, B

    det_A1, det_A2, det_A3 = np.linalg.det(A1), np.linalg.det(A2), np.linalg.det(A3)

    # Вычисляем значения x, y, z
    x = det_A1 / det_A
    y = det_A2 / det_A
    z = det_A3 / det_A

    print("Решение методом Крамера:", [x, y, z])
```

```
Решение методом Крамера: [2.7777777777777803, 2.555555555555557, 0.6666666666666672]
```

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

```python
import numpy as np

A = np.array([[2, -1, 3],
              [1,  1, 1],
              [3, -3, 2]])

# Вектор правой части
B = np.array([5, 6, 2])

# Проверяем, можно ли найти обратную матрицу
if np.linalg.det(A) == 0:
    print("Система не имеет единственного решения")
else:
    # Вычисляем обратную матрицу
    A_inv = np.linalg.inv(A)

    # Находим решение
    X = A_inv @ B

    print("Решение матричным методом:", X)
```

```
Решение матричным методом: [2.77777778 2.55555556 0.66666667]
```

Вывод: в ходе лабораторной работы были исследованы базовые возможности библиотеки NumPy языка программирования Python.