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 ПРАКТИЧЕСКОЙ РАБОТЕ №2
дисциплины
«Искусственный интеллект и машинное обучение»
Вариант 4

Выполнила:
Баранник Софья Евгеньевна
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 репозиторий GitHub: https://github.com/S-o-n-y-a/AI-ML.git

1. Выполнили операцию сложения над двумя массивами, векторизацию.

```
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.

```
data = np.array([1,2,3,4,5])
mean = np.mean(data) #Среднее значение
std_dev = np.std(data) #Стандратное отклонение
print(mean, std_dev)
```

```
3.0 1.4142135623730951
```

Рисунок 2. Статистические функции в NumPy

3. Выполнили различные операции над матрицей.

```
m = np.matrix("1 2 3 4; 5 6 7 8; 9 1 5 7") # Создание матрицы
print("Созданная матрица:\n", m)
# Все значения начинаются с 0
print("Вывод элемента 1 строки и 0 столбца:", m[1,0])
print("Вывод 1 строки матрицы:", m[1, :])
print("Вывод 2 столбца матрицы:", m[:, 2])
print("Вывод части строки матрицы:", m[1,2:])
print("Вывод части столбца матрицы:", m[0:2, 1])
print("Извлечение подмассива:", m[0:2, 1:3])
cols = [0, 1, 3]
print("Извлечение столбцов и строк:", m[:, cols])
```

```
Созданная матрица:
 [[1 2 3 4]
 [5 6 7 8]
 [9 1 5 7]]
Вывод элемента 1 строки и 0 столбца: 5
Вывод 1 строки матрицы: [[5 6 7 8]]
Вывод 2 столбца матрицы: [[3]
 [7]
 [5]]
Вывод части строки матрицы: [[7 8]]
Вывод части столбца матрицы: [[2]
 [6]]
Извлечение подмассива: [[2 3]
 [6 7]]
Извлечение столбцов и строк: [[1 2 4]
 [5 6 8]
 [9 1 7]]
```

Рисунок 3. Выполнение различных операций над матрицей

4. Произвели расчет статистик массива.

```
print("Размерность массива:",m.shape)
print("Расчет статистики, максимальное значение в матрице:", m.max())
print("Расчет статистики, максимальные значения по строкам:", m.max(axis=1))
print("Расчет статистики, максимальные значения по столбцам:", m.max(axis=0))
```

```
Размерность массива: (3, 4)
Расчет статистики, максимальное значение в матрице: 9
Расчет статистики, максимальные значения по строкам: [[4]
 [8]
 [9]]
Расчет статистики, максимальные значения по столбцам: [[9 6 7 8]]
```

Рисунок 4. Функции расчета статистик в NumPy

5. Использовали boolean массив.

```
b = 5 > 7
print(b)
nums = np.array([1,2,3,4,5,6,7,8,9,10])
less_then_7 = nums < 7
less_then_7
```

```
False
array([ True,  True,  True,  True,  True,  True, False, False, False,
       False])
```

Рисунок 5. Операции над boolean массивом

6. Построили логическую матрицу с условием.

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

```
matrix([[3, 4, 5, 6, 7, 5, 7]])
```

Рисунок 6. Логическая матрица

7. Модифицировали массив с помощью boolean.

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

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

Рисунок 7. Модифицирование массива с помощью boolean

8. Воспользовались функцией arange().

```
np.arange(10)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.arange(5,12)
```

```
array([ 5,  6,  7,  8,  9, 10, 11])
```

```
np.arange(1,5,0.5)
```

```
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

Рисунок 8. Функция arange()

9. Использовали функцию matrix() для задания матрицы.

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

```
matrix([[1, 2],
        [3, 4]])
```

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

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

Рисунок 9. Функция matrix()

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

```
np.zeros((3,4))
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
np.eye(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Рисунок 10. Функции zeros() и eye()

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

```
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(A)
print(np.ravel(A))
print("order = C:", np.ravel(A, order = "C")) # Массив собирается из строк
print("order = F:", np.ravel(A, order = "F")) # Массив мобирается из стольцов
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[1 2 3 4 5 6 7 8 9]
order = C: [1 2 3 4 5 6 7 8 9]
order = F: [1 4 7 2 5 8 3 6 9]
```

Рисунок 11. Функция ravel()

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

```
a = np.random.rand(10)
print(a)
print(np.where(a > 0.5, True, False))
print(np.where(a > 0.5, 1, -1))
```

```
[6.60638502e-04 9.35641026e-01 9.09375164e-01 4.53880180e-01
 1.99402923e-01 9.44270588e-01 8.76601061e-01 9.28278048e-01
 9.92061643e-01 5.03936474e-01]
[False  True  True False False  True  True  True  True  True]
[-1  1  1 -1 -1  1  1  1  1  1]
```

Рисунок 12. Функции random и where

13. Использовали функцию meshgrid для создания матрицы координат.

```
x = np.linspace(0,1,5)
print("x = ", x)
y = np.linspace(0,2,5)
print("y = ", y)
xg, yg = np.meshgrid(x, y)
print("xg = ", xg)
print("yg = ", yg)
```

```
x =  [0.   0.25 0.5  0.75 1.  ]
y =  [0.  0.5 1.  1.5 2. ]
xg =  [[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.  ]]
yg =  [[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 13. Функция meshgrid()

14. Визуализировали полученные матрицы.

```
import matplotlib.pyplot as plt
plt.plot(xg,yg, color = "r", marker = "*", linestyle = "none")
```

```
[<matplotlib.lines.Line2D at 0x1fa8766c680>,
 <matplotlib.lines.Line2D at 0x1fa8766c740>,
 <matplotlib.lines.Line2D at 0x1fa8766c7d0>,
 <matplotlib.lines.Line2D at 0x1fa8766c890>,
 <matplotlib.lines.Line2D at 0x1fa8766c350>]
```

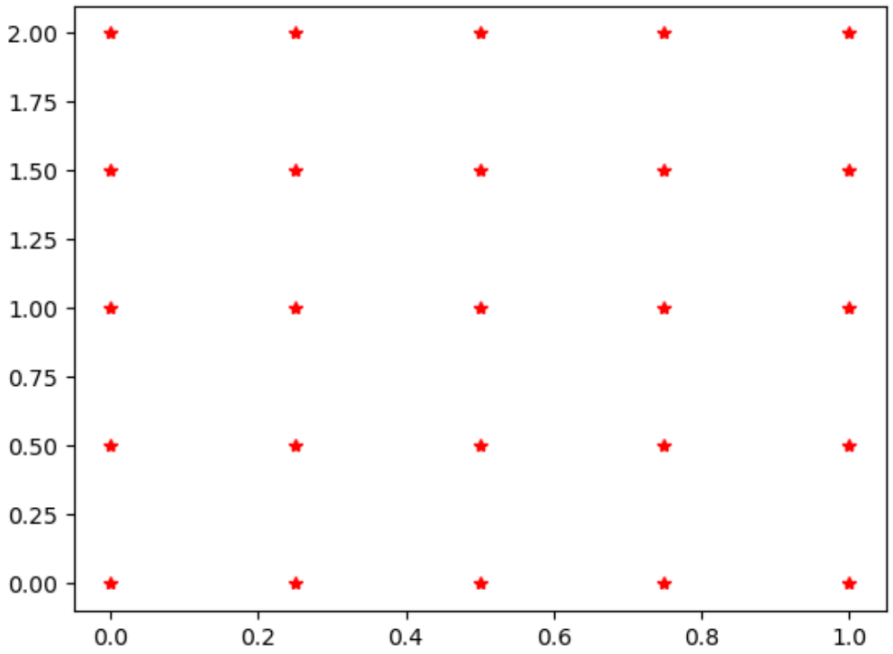

Рисунок 14. Визуализация данных в виде графика

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

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

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

Рисунок 15. Функция meshgrid

16. Создали несколько различных векторов-строк.

```
v_hor_np = np.array([1,2])
print("Вектор-строка: ", v_hor_np)
v_hor_zeros = np.zeros((5,))
print("Нулевой вектор: ", v_hor_zeros)
v_hor_one = np.ones((5,))
print("Единичный вектор: ", v_hor_one)
```

```
Вектор-строка:  [1 2]
Нулевой вектор:  [0. 0. 0. 0. 0.]
Единичный вектор:  [1. 1. 1. 1. 1.]
```

Рисунок 16. Способы задать векторы-строки

17. Создали различные векторы-столбцы.

```
v_vert = np.array([[1],[2]])
print("Вектор-столбец: ", v_vert)
v_vert_zeros = np.zeros((5,1))
print("Нулевой вектор-столбец: ", v_vert_zeros)
v_vert_ones = np.ones((5,1))
print("Единичный вектор-столбец: ", v_vert_ones)
```

```
Вектор-столбец:  [[1]
 [2]]
Нулевой вектор-столбец:  [[0.]
 [0.]
 [0.]
 [0.]
 [0.]]
Единичный вектор-столбец:  [[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
```

Рисунок 17. Способы задать вектор-столбец

18. Использовали различные методы для создания квадратных матриц.

```
m_sqr_arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("С помощью array:\n", m_sqr_arr)
m_sqr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
m_sqr_arr2 = np.array(m_sqr)
print("С передачей в функцию array:\n", m_sqr_arr2)
m_sqr_matr = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("С помощью matrix:\n", m_sqr_matr)
m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
print("matrix со строчными значениями:\n", m_sqr_mx)
```

```
С помощью array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
С передачей в функцию array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
С помощью matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
matrix со строчными значениями:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Рисунок 18. Способы задать квадратную матрицу

19. Создали диагональную матрицу различными методами.

```
m_diag = [[1, 0, 0], [0, 5, 0], [0, 0, 9]]
m_diag_np = np.matrix(m_diag)
print("Диагональная матрица, заполненная вручную:\n", m_diag_np)
m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
diag = np.diag(m_sqr_mx)
print("Извлеченная диагональ из матрицы:\n", diag)
new_diag = np.diag(np.diag(m_sqr_mx))
print("Матрица, построенная из извлеченной диагонали:\n", new_diag)
```

```
Диагональная матрица, заполненная вручную:
 [[1 0 0]
 [0 5 0]
 [0 0 9]]
Извлеченная диагональ из матрицы:
 [1 5 9]
Матрица, построенная из извлеченной диагонали:
 [[1 0 0]
 [0 5 0]
 [0 0 9]]
```

Рисунок 19. Способы задать диагональную матрицу

20. Создали единичную матрицу различными методами.

```
m_e = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
m_e_np = np.matrix(m_e)
print("Диагональная матрица, заполненная вручную:\n", m_diag_np)
m_eye = np.eye(3)
print("Функция eye:\n", m_eye)
m_idnt = np.identity(3)
print("Функция identity:\n", m_idnt)
```

```
Диагональная матрица, заполненная вручную:
 [[1 0 0]
 [0 5 0]
 [0 0 9]]
Функция eye:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Функция identity:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Рисунок 20. Создание единичной матрицы

21. Создали нулевую матрицу.

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

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

Рисунок 21. Нулевая матрица

22. Выполнили процесс транспонирования матрицы различными методами.

```
A = np.matrix('1 2 3; 4 5 6')
print("Созданная матрица:\n", A)
A_t = A.transpose()
print("Фунция transpose:\n", A_t)
print("Сокращенный вариант:\n", A.T)
```

```
Созданная матрица:
 [[1 2 3]
 [4 5 6]]
Фунция transpose:
 [[1 4]
 [2 5]
 [3 6]]
Сокращенный вариант:
 [[1 4]
 [2 5]
 [3 6]]
```

Рисунок 22. Транспонирование матриц

23. Умножили матрицу на число.

```
A = np.matrix('1 2 3; 4 5 6')
print(A)
C = 3 * A
print("3 * A\n", C)
```

```
[[1 2 3]
 [4 5 6]]
3 * A
 [[ 3  6  9]
 [12 15 18]]
```

Рисунок 23. Умножение матрицы на число

24. Выполнили сложение 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]]
```

Рисунок 24. Сложение матриц

25. Выполнили умножение матриц.

```
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 25. Умножение матриц

26. Посчитали определитель матрицы.

```
A = np.matrix('-4 -1 2; 10 4 -1; 8 3 1')
print(A)
np.linalg.det(A)
```

```
[[-4 -1  2]
 [10  4 -1]
 [ 8  3  1]]
[71]:

-14.000000000000009
```

Рисунок 26. Определитель матрицы

27. Посчитали обратную матрицу.

```
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 27. Обратная матрица

28. Посчитали ранг матрицы.

```
m_eye = np.eye(4)
print(m_eye)
rank = np.linalg.matrix_rank(m_eye)
print(rank)
```

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

Рисунок 28. 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.

```
import numpy as np
matrix = np.matrix("1,5,7;2,6,9;4,8,3")
multiple = matrix * 2
print(multiple)
multiple[multiple > 10] = 0
print(multiple)
```

```
[[ 2 10 14]
 [ 4 12 18]
 [ 8 16  6]]
[[ 2 10  0]
 [ 4  0  0]
 [ 8  0  6]]
```

Рисунок 29. 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.

```
arr = np.random.randint(1, 101, 20)
print(arr)
div_5 = arr[arr % 5 == 0]
print("элементы, которые делятся на 5 без остатка:\n", div_5)
arr[arr % 5 == 0] = -1
print("Обновлённый массив:\n", arr)
```

```
[60 28  5 27 41 77 74 46 27 86 18 93 69 45 96 99 40 35  2 27]
элементы, которые делятся на 5 без остатка:
 [60  5 45 40 35]
Обновлённый массив:
 [-1 28 -1 27 41 77 74 46 27 86 18 93 69 -1 96 99 -1 -1  2 27]
```

Рисунок 30. 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.

```
a = np.random.randint(0, 51, 5)
print("Первый массив:\n", a)
b = np.random.randint(0, 51, 5)
print("Второй массив:\n", b)
ab = np.vstack((a,b))
print("Объединенный массив:\n", ab)
a1, b1 = np.vsplit(ab, 2)
print("Разделенные массивы:\nПервый:", a1, "\nВторой:", b1)
```

```
Первый массив:
 [11 46 14 45 48]
Второй массив:
 [ 5 33 22 40 45]
Объединенный массив:
 [[11 46 14 45 48]
 [ 5 33 22 40 45]]
Разделенные массивы:
Первый: [[11 46 14 45 48]]
Второй: [[ 5 33 22 40 45]]
```

Рисунок 31. 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.

```
a = np.random.randint(-10, 11, 50)
print("Созданный массив: ", a)
summ_all = np.sum(a)
print("Сумма всех элементов: ", summ_all)
pos_summ = np.sum(a[a > 0])
print("Сумма положительных элементов: ", pos_summ)
neg_summ = np.sum(a[a < 0])
print("Сумма отрицательных элементов: ", neg_summ)
```

```
Созданный массив:  [  0  -5  -2  -1 -10   6  -1  -7   4 -10   4   5   2   6   5   2  -8  -8
  -4   5  -5   5  -4  -3  -3  -8   7  -8   1   2  -4  -7  10   1  -5  -9
   1  -2   5  -3 -10   0   1   5   0   2   6   6   8   4]
Сумма всех элементов:  -24
Сумма положительных элементов:  103
Сумма отрицательных элементов:  -127
```

Рисунок 32. 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).

```
edm = np.identity(4)
print("Единичная матрица: \n", edm)
diagm = np.diag([5, 10, 15, 20])
print("Диагональная матрица:\n", diagm)
edm_summ = np.sum(edm)
print("Сумма элементов единичной матрицы:\n", edm_summ)
diagm_summ = np.sum(diagm)
print("Сумма элементов диагональной матрицы:\n", diagm_summ)
print("По результатам сумма элементов диагональной матрицы оказалась больше, чем единичной.")
```

```
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 элементов единичной матрицы:
 4.0
Сумма элементов диагональной матрицы:
 50
По результатам сумма элементов диагональной матрицы оказалась больше, чем единичной.
```

Рисунок 33. 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.

```
a = np.random.randint(1, 21, (3, 3))
print("Первая матрица:\n", a)
b = np.random.randint(1, 21, (3, 3))
print("Вторая матрица:\n", b)
summ = a + b
print("Сумма: ", summ)
razn = a - b
print("Разница: ", razn)
proz = a * b
print("Произведение: ", proz)
```

```
Первая матрица:
 [[ 7  6  1]
 [ 1 12  9]
 [ 9 20  7]]
Вторая матрица:
 [[ 4 16 15]
 [17  7 16]
 [10  8  2]]
Сумма:  [[11 22 16]
 [18 19 25]
 [19 28  9]]
Разница:  [[  3 -10 -14]
 [-16   5  -7]
 [ -1  12   5]]
Произведение:  [[ 28  96  15]
 [ 17  84 144]
 [ 90 160  14]]
```

Рисунок 34. 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.

```
a = np.random.randint(1, 11, (2, 3))
print("Первая матрица:\n", a)
b = np.random.randint(1, 11, (3, 2))
print("Вторая матрица:\n", b)
mult = a @ b
print("Матричное умножение:\n", mult)
```

```
Первая матрица:
 [[ 7  7 10]
 [ 9 10  3]]
Вторая матрица:
 [[4 3]
 [5 1]
 [8 1]]
Матричное умножение:
 [[143  38]
 [110  40]]
```

Рисунок 35. 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).

```
a = np.random.randint(1, 21, (3, 3))
print("Матрица:\n", a)
det = np.linalg.det(a)
print("Определитель:\n", det)
if np.isclose(det, 0):
    inv = "Матрица вырождена"
else:
    inv = np.linalg.inv(a)
print("Обратная матрица:\n", inv)
```

```
Матрица:
 [[20 12 18]
 [15 14 17]
 [ 6 14  1]]
Определитель:
 -1168.0000000000002
Обратная матрица:
 [[ 0.19178082 -0.20547945  0.04109589]
 [-0.0744863   0.07534247  0.05993151]
 [-0.10787671  0.17808219 -0.08561644]]
```

Рисунок 36. 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).

```
a = np.random.randint(1, 51, (4, 4))
print("Матрица:\n", a)
transp = a.T
print("Транспонированная матрица:\n", transp)
value = np.trace(a)
print("След матрицы:\n", value)
```

```
Матрица:
 [[15 47 12  8]
 [31 32 26  3]
 [45 16 28 13]
 [36 20 33 37]]
Транспонированная матрица:
 [[15 31 45 36]
 [47 32 16 20]
 [12 26 28 33]
 [ 8  3 13 37]]
След матрицы:
 112
```

Рисунок 37. 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.

```
A = np.array([[2, 3, -1],
              [4, -1, 2],
              [-3, 5, 4]])
B = np.array([5, 6, -2])
res = np.linalg.solve(A, B)
print("Решение системы: ", res)
```

```
Решение системы:  [1.63963964 0.57657658 0.00900901]
```

Рисунок 38. 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 4: В диете необходимо включить три продукта: мясо, рыбу и овощи. Они содержат белки, жиры и углеводы в разном количестве. В 100 г мяса содержится 20 г белков, 10 г жиров и 5 г углеводов, в 100 г рыбы — 15 г белков, 5 г жиров и 10 г углеводов, в 100 г овощей — 5 г белков, 2 г жиров и 20 г углеводов. Суточная норма: 100 г белков, 50 г жиров и 150 г углеводов. Сколько граммов каждого продукта нужно съесть?

```python
import numpy as np

# Матрица содержания питательных веществ в продуктах (белки, жиры, углеводы)
A = np.array([[20, 15, 5],   # Белки
              [10, 5, 2],    # Жиры
              [5, 10, 20]])  # Углеводы

# Вектор суточной нормы питательных веществ
B = np.array([100, 50, 150])  # Белки, жиры, углеводы

# Решение системы уравнений методом Крамера
det_A = np.linalg.det(A)
if det_A != 0:
    kram1 = np.linalg.det(np.column_stack([B, A[:, 1], A[:, 2]])) / det_A
    kram2 = np.linalg.det(np.column_stack([A[:, 0], B, A[:, 2]])) / det_A
    kram3 = np.linalg.det(np.column_stack([A[:, 0], A[:, 1], B])) / det_A
    kramer = np.array([kram1, kram2, kram3])
else:
    kramer = "не применим"

# Решение методом матричного уравнения
if det_A == 0:
    matrix = "не применим"
else:
    matrix = np.linalg.inv(A).dot(B)

# Решение системы уравнений средствами библиотеки NumPy
try:
    numpy_solution = np.linalg.solve(A, B)
except np.linalg.LinAlgError:
    numpy_solution = "не применим"

print("Метод Крамера: ", kramer)
print("Метод матричного уравнения: ", matrix)
print("Библиотека NumPy: ", numpy_solution)
```

```
Метод Крамера:  [ 4.28571429 -1.42857143  7.14285714]
Метод матричного уравнения:  [ 4.28571429 -1.42857143  7.14285714]
Библиотека NumPy:  [ 4.28571429 -1.42857143  7.14285714]
```

Рисунок 39. Результат задания 11

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.

```python
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: $\det(A-1)=1$ (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 ($n \times n$).

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 практической работы были исследованы базовые возможности библиотеки NumPy языка программирования Python.