

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 № 13

Выполнил:
Новиков Евгений Александрович
2 курс, группа ИВТ-б-о-23-2,
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 В.И.

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2025 г.

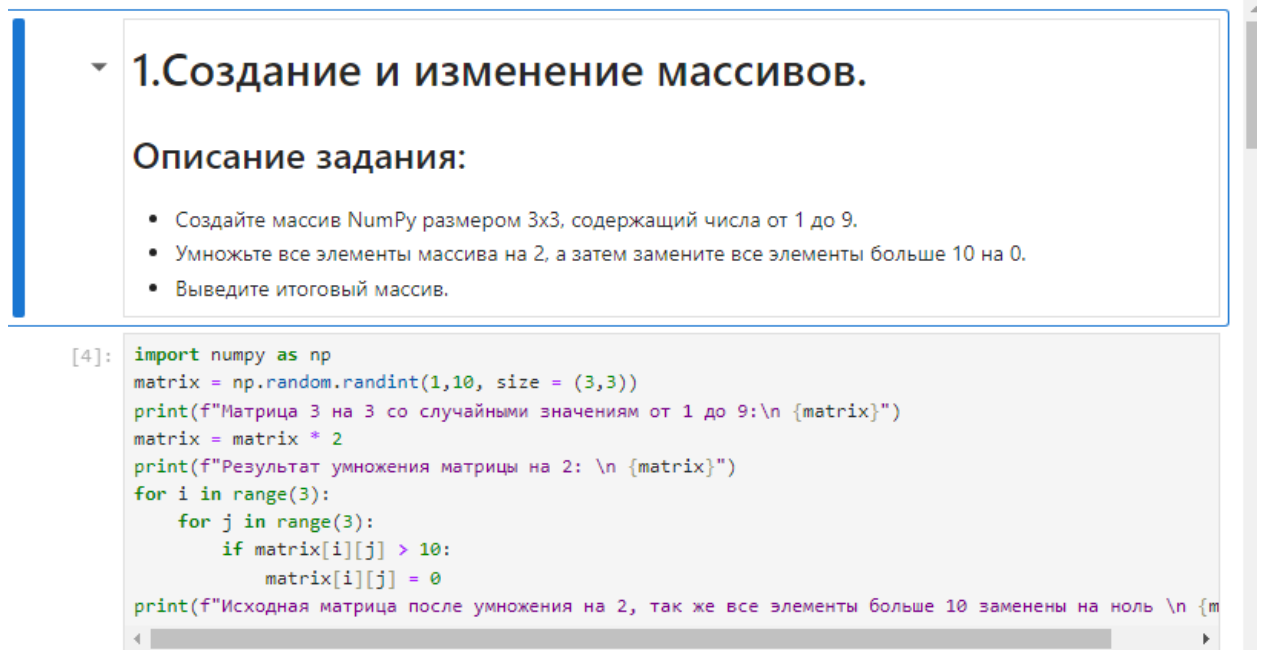
Тема работы: «Основы работы с библиотекой NumPy».

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

Порядок выполнения работы:

Ссылка на git репозиторий: https://github.com/Zhenya0123456789/ml_2.git

1. Выполнил 1 задание из методического материала:



▼ **1.Создание и изменение массивов.**

Описание задания:

- Создайте массив NumPy размером 3x3, содержащий числа от 1 до 9.
- Умножьте все элементы массива на 2, а затем замените все элементы больше 10 на 0.
- Выведите итоговый массив.

```
[4]: import numpy as np
matrix = np.random.randint(1,10, size = (3,3))
print(f"Матрица 3 на 3 со случайными значениям от 1 до 9:\n {matrix}")
matrix = matrix * 2
print(f"Результат умножения матрицы на 2: \n {matrix}")
for i in range(3):
    for j in range(3):
        if matrix[i][j] > 10:
            matrix[i][j] = 0
print(f"Исходная матрица после умножения на 2, так же все элементы больше 10 заменены на ноль \n {m
```

Рисунок 1 – Программа к заданию №1

Матрица 3 на 3 со случайными значениям от 1 до 9:

```
[[9 6 6]
 [9 7 7]
 [9 3 7]]
```

Результат умножения матрицы на 2:

```
[[18 12 12]
 [18 14 14]
 [18 6 14]]
```

Исходная матрица после умножения на 2, так же все элементы больше 10 заменены на ноль

```
[[0 0 0]
 [0 0 0]
 [0 6 0]]
```

Рисунок 2 – Результат работы программы к заданию №1

2. Выполнил задание №2 из методического материала:

2.Работа с булевыми масками.

Описание задания:

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

```
[19]: array = np.random.randint(1,101,size=20)
print(f"Массив из 20 случайных чисел от 1 до 100: \n {array}")
print(f"Массив, в котором только элементы кратные 5: \n {array[array%5==0]}")
array[array%5==0]=-1
print(f"Массив, в котором все элементы кратные 5 заменены на -1: \n {array}")
```

Рисунок 3 – Программа к заданию №2

```
Массив из 20 случайных чисел от 1 до 100:
[84  5 28 90  6 12  4 92 59 78  2 28 35 46 93 23 64 47 81 42]
Массив, в котором только элементы кратные 5:
[ 5 90 35]
Массив, в котором все элементы кратные 5 заменены на -1:
[84 -1 28 -1  6 12  4 92 59 78  2 28 -1 46 93 23 64 47 81 42]
```

Рисунок 4 – Результат работы программы к заданию №2

3. Выполнил задание №3 из методического материала:

3.Объединение и разбиение массивов.

Описание задания:

- Создайте два массива NumPy размером 1x5, 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.
- Выведите все промежуточные и итоговые результаты.

```
[17]: array1 = np.random.randint(0,51, size = (1,5))
array2 = np.random.randint(0,51, size = (1,5))
print(f"Первый массив: \n {array1}")
print(f"Второй массив: \n {array2}")
array3 = np.vstack((array1,array2))
print(f"Массив, полученный путем объединения двух других: \n {array3}")
array4 = np.vsplit(array3,2)
print(f"Разделенные массивы: \n {array4}")
```

Рисунок 5 – Программа к заданию №3

```

Первый массив:
[[20  5 39 12 22]]
Второй массив:
[[36 41 40 27 20]]
Массив, полученный путем объединения двух других:
[[20  5 39 12 22]
 [36 41 40 27 20]]
Разделенные массивы:
[array([[20,  5, 39, 12, 22]]), array([[36, 41, 40, 27, 20]])]

```

Рисунок 6 – Результат работы программы к заданию №3

4. Выполнил задание №4 из методического материала:

▼ 4.Генерация и работа с линейными последовательностями.

Описание задания:

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

```

[7]: array = np.linspace(-10,10,num=50)
print(f"Исходный массив с равномерно распределенными числами от -10 до 10:\n {array}")
array_sum = np.sum(array)
print(f"Сумма всех элементов массива: \n {array_sum}")
positive_elements = array[array>0]
sum_positive = np.sum(positive_elements)
negative_elements = array[array < 0]
sum_negative = np.sum(negative_elements)
print(f"Сумма всех положительный элементов: \n {sum_positive}")
print(f"Сумма всех отрицательных элементов: \n {sum_negative}")

```

Рисунок 7 – Программа к заданию №4

```

Исходный массив с равномерно распределенными числами от -10 до 10:
[-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 массива:
7.105427357601002e-15
Сумма всех положительный элементов:
127.55102040816328
Сумма всех отрицательных элементов:
-127.55102040816327

```

Рисунок 8 – Результат работы программы к заданию №4

5. Выполнил задание №5 из методического материала:

5.Работа с диагональными и единичными матрицами.

Описание задания:

- Создайте единичную матрицу размером 4x4.
- Создайте диагональную матрицу размером 4x4 с диагональными элементами [5,10,15,20] (не использовать циклы).

```
[9]: matrix = np.eye(4)
      diagonal_matrix = np.diag([5,10,15,20])
      print(f"Единичная матрица 4x4: \n {matrix}")
      print(f"Диагональная матрица с заранее известными элементами: \n {diagonal_matrix}")
```

Рисунок 9 – Программа к заданию №5

```
Единичная матрица 4x4:
[[1.  0.  0.  0.]
 [0.  1.  0.  0.]
 [0.  0.  1.  0.]
 [0.  0.  0.  1.]]
Диагональная матрица с заранее известными элементами:
[[ 5  0  0  0]
 [ 0 10  0  0]
 [ 0  0 15  0]
 [ 0  0  0 20]]
```

Рисунок 10 – Результат работы программы к заданию №5

6. Выполнил задание №6 из методического материала:

6.Создание и базовые операции с матрицами.

Описание задания:

- Создайте две квадратные матрицы NumPy размером 3x3, заполненные случайными числами от 1 до 20. Вычислите и выведите: их сумму, разность, поэлементное произведение.

```
[13]: matrix_1 = np.random.randint(1,21,(3,3))
matrix_2 = np.random.randint(1,21,(3,3))
print(f"Первая матрица: \n {matrix_1}")
print(f"Вторая матрица: \n {matrix_2}")
matrix_sum = np.add(matrix_1,matrix_2)
print(f"Сумма матриц: \n {matrix_sum}")
matrix_1_razn = np.subtract(matrix_1,matrix_2)
matrix_2_razn = np.subtract(matrix_2,matrix_1)
print(f"Разность между первой и второй матрицами \n {matrix_1_razn}")
print(f"Разность между второй и первой матрицами \n {matrix_2_razn}")
multiply_matrix = np.multiply(matrix_1,matrix_2)
print(f"Поэлементное произведение матриц: \n {multiply_matrix}")
```

Рисунок 11 – Программа к заданию №6

```
Первая матрица:
[[ 2 12  4]
 [10 18 14]
 [19 10 11]]
Вторая матрица:
[[ 4  1 13]
 [14 10 13]
 [14  7 20]]
Сумма матриц:
[[ 6 13 17]
 [24 28 27]
 [33 17 31]]
Разность между первой и второй матрицами
[[-2 11 -9]
 [-4  8  1]
 [ 5  3 -9]]
Разность между второй и первой матрицами
[[ 2 -11  9]
 [ 4 -8 -1]
 [-5 -3  9]]
Поэлементное произведение матриц:
[[ 8 12 52]
 [140 180 182]
 [266  70 220]]
```

Рисунок 12 – Результат работы программы к заданию №6

7. Выполнил задание №7 из методического материала:

▼ 7. Умножение матриц.

Описание задания:

- Создайте две матрицы NumPy:
 - Первую размером 2x3, заполненную случайными числами от 1 до 10.
 - Вторую размером 3x2, заполненную случайными числами от 1 до 10.
- Выполните матричное умножение (@ или np.dot) и выведите результат.

```
[15]: matrix_1 = np.random.randint(1,11,(2,3))
matrix_2 = np.random.randint(1,11,(3,2))
print(f"Наша первая матрица: \n {matrix_1}")
print(f"Наша вторая матрица: \n {matrix_2}")
multiply_1_2 = np.dot(matrix_1,matrix_2)
multiply_2_1 = matrix_2 @ matrix_1
print(f"Результат умножения первой матрицы на вторую: \n {multiply_1_2}")
print(f"Результат умножения второй матрицы на первую: \n {multiply_2_1}")
```

Рисунок 13 – Программа к заданию №7

```
Наша первая матрица:
[[7 4 4]
 [8 6 5]]
Наша вторая матрица:
[[ 1  5]
 [ 3 10]
 [ 6  1]]
Результат умножения первой матрицы на вторую:
[[ 43  79]
 [ 56 105]]
Результат умножения второй матрицы на первую:
[[ 47  34  29]
 [101  72  62]
 [ 50  30  29]]
```

Рисунок 14 – Результат работы программы к заданию №7

8. Выполнил задание №8 из методического материала:

8.Определитель и обратная матрица.

Описание:

- Создайте случайную квадратную матрицу 3x3, найдите и выведите:
 - Определитель этой матрицы.
 - Обратную матрицу(если существует, иначе выведите сообщение, что матрица вырождена).

```
[47]: nul_matrix = np.array([[1,2,3],[4,5,6],[1,2,3]])
      matrix = np.random.randint(-10,10,(3,3))
      print(f"Случайная матрица 3x3: \n {matrix}")
      det_matrix = np.linalg.det(matrix)
      print(f"Определитель матрицы: \n {det_matrix}")
      obr_matrix = np.linalg.inv(matrix)
      if det_matrix == 0:
          print("Детерминант матрицы равен 0, значит матрица вырождена и не имеет обратной")
      else:
          print(f"Обратная матрица: \n {obr_matrix}")
      print(f"Пример вырожденной матрицы: \n {nul_matrix}")
      print(f"Определитель этой матрицы: \n {np.linalg.det(nul_matrix)}")
```

Рисунок 15 – Программа к заданию №8

```
Случайная матрица 3x3:
[[ 8 -5  5]
 [-4  9  1]
 [ 0 -9 -3]]
Определитель матрицы:
95.99999999999999
Обратная матрица:
[[-0.1875    -0.625    -0.52083333]
 [-0.125     -0.25     -0.29166667]
 [ 0.375     0.75      0.54166667]]
Пример вырожденной матрицы:
[[1 2 3]
 [4 5 6]
 [1 2 3]]
Определитель этой матрицы:
0.0
```

Рисунок 16 – Результат работы программы к заданию №8

9. Выполнил задание №9 из методического материала:

9. Транспонирование и след матрицы.

Описание задания:

- Создайте матрицу NumPy размером 4x4, содержащую случайные целые числа от 1 до 50, выведите:
 - Исходную матрицу.
 - Транспонированную матрицу.
 - След матрицы (сумму элементов главной диагонали).

```
[50]: matrix = np.random.randint(1,51,(4,4))
print(f"Исходная матрица: \n {matrix}")
trans_matrix = np.transpose(matrix)
print(f"Транспонированная матрица: \n {trans_matrix}")
trace = 0
for i in range(4):
    trace += matrix[i][i]
print(f"След матрицы: \n {trace}")
```

Рисунок 17 – Программа к заданию №9

```
Исходная матрица:
[[29  4 27 23]
 [ 2  2 47 18]
 [ 1  8  2 21]
 [34  7  1 49]]
Транспонированная матрица:
[[29  2  1 34]
 [ 4  2  8  7]
 [27 47  2  1]
 [23 18 21 49]]
След матрицы:
82
```

Рисунок 18 – Результат работы программы к заданию №9

10. Выполнил задание №10 из методических материалов

10. Системы линейных уравнений.

Описание задания:

- Решите систему линейных уравнений вида:

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

```
57]: A = np.array([[2,3,-1],[4,-1,2],[-3,5,4]])
      B = np.array([[5],[6],[-2]])
      print(f"Матрица коэффициентов: \n {A}")
      print(f"Вектор правой части: \n {B}")
      resh = np.linalg.solve(A,B)
      print(f"Решение системы: \n {resh}")
```

Матрица коэффициентов:

Рисунок 19 – Программа к заданию №10

```
Матрица коэффициентов:
[[ 2  3 -1]
 [ 4 -1  2]
 [-3  5  4]]
Вектор правой части:
[[ 5]
 [ 6]
 [-2]]
Решение системы:
[[1.63963964]
 [0.57657658]
 [0.00900901]]
```

Рисунок 20 – Результат работы программы к заданию №10

11. Выполнил индивидуальное задание:

11. Индивидуальное задание (13 вариант).

Решите индивидуальное задание согласно варианта. 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`.

13. **Оптимизация рабочего времени.** Три сотрудника должны выполнить работу за минимальное время. Первый выполняет задачу за 5 часов, второй — за 3 часа, а третий — за 2 часа. Как распределить работу между ними, если всего 20 задач?

Матричный метод ($A^{-1} \cdot B$):

- Наша система линейных уравнений:

$$\begin{cases} x_1 + x_2 + x_3 = 20, \\ 5x_1 - 3x_2 = 0, \\ 3x_2 - 2x_3 = 0. \end{cases}$$

```
[20]: A = np.array([[1,1,1],[5,-3,0],[0,3,-2]])
      B = np.array([[20],[0],[0]])
      print(f"Матрица коэффициентов: \n {A}")
      print(f"Вектор правой части: \n {B}")
      C = np.linalg.inv(A) @ B
      proverka = np.linalg.solve(A,B)
      print(f"Решение системы: \n x1 = {C[0]} \n x2 = {C[1]} \n x3 = {C[2]}")
      print(f"Решение сисетмы через np.linalg.solve: \n {proverka}")
```

Рисунок 21 – Программа для индивидуального задания

```
Матрица коэффициентов:
[[ 1  1  1]
 [ 5 -3  0]
 [ 0  3 -2]]
Вектор правой части:
[[20]
 [ 0]
 [ 0]]
Решение системы:
x1 = [3.87096774]
x2 = [6.4516129]
x3 = [9.67741935]
Решение сисетмы через np.linalg.solve:
[[3.87096774]
 [6.4516129 ]
 [9.67741935]]
```

Рисунок 22 –Результат работы программы для индивидуального задания

Ответы на контрольные вопросы:

1. Назначение библиотеки NumPy

— это библиотека для работы с многомерными массивами, матрицами и числами в Python. Она предоставляет высокопроизводительные структуры данных и операции для численных вычислений, включая функции для математических, логических, статистических и алгебраических операций.

2. Массивы ndarray Массивы ndarray (N-dimensional array)

— это основная структура данных библиотеки NumPy. Они представляют собой многомерные таблицы, где каждый элемент имеет одинаковый тип данных. Массивы могут быть одномерными, двумерными и многомерными.

3. Доступ к частям многомерного массива

Доступ к частям массива осуществляется с помощью индексации. Можно использовать:

- Индексацию с помощью числовых индексов для одномерных и многомерных массивов.
- Срезы (например, `arr[1:3, 2:4]` для двумерных массивов).
- Логическую индексацию или маски.

4. Расчет статистик по данным

NumPy предоставляет функции для расчета различных статистик, таких как:

- Среднее значение: `np.mean()`
- Медиана: `np.median()`
- Стандартное отклонение: `np.std()`
- Минимум и максимум: `np.min()`, `np.max()`
- Квантиль: `np.percentile()`
- Корреляция: `np.corrcoef()`

5. Выборка данных из массивов ndarray

Выборка данных из массива осуществляется через индексацию, срезы или логическую маску. Например, чтобы выбрать все элементы, которые больше 5: `arr[arr > 5]`

6. Основные виды матриц и векторов

- Вектор: одномерный массив. Создается с помощью `np.array([1, 2, 3])` или `np.arange()`.
- Матрица: двумерный массив. Создается с помощью `np.array([[1, 2], [3, 4]])` или `np.zeros((2, 2))`.
- Единичная матрица: `np.eye(n)`
- Нулевая матрица: `np.zeros((n, m))`
- Матрица с произвольными числами: `np.random.rand(n, m)`

7. Транспонирование матриц

Транспонирование матрицы меняет строки на столбцы. В NumPy это делается через метод `.T`: `A.T`

8. Свойства операции транспонирования матриц

- Транспонирование единичной матрицы дает единичную матрицу.
- Транспонирование транспонированной матрицы возвращает исходную матрицу: $(A.T).T = A$.
- Транспонирование произведения матриц: $(A * B).T = B.T * A.T$.

9. Средства NumPy для транспонирования матриц

Для транспонирования в NumPy можно использовать: • `.T` • `np.transpose(A)`

10. Основные действия над матрицами

Основные действия: • Сложение и вычитание: $A + B$, $A - B$ • Умножение: $A * B$, или `np.dot(A, B)` для матричного умножения • Транспонирование: `A.T` • Определитель: `np.linalg.det(A)` • Обратная матрица: `np.linalg.inv(A)`

11. Умножение матрицы на число

Для умножения матрицы на число используется стандартная операция умножения: $A * c$

12. Свойства операции умножения матрицы на число

- Это дистрибутивная операция: $(c * A) + (c * B) = c * (A + B)$.
- Умножение на 1 оставляет матрицу неизменной: $1 * A = A$.
- Умножение на 0 дает нулевую матрицу: $0 * A = 0$.

13. Сложение и вычитание матриц

Сложение и вычитание матриц выполняется через стандартные операторы: $A + B$ $A - B$

14. Свойства операций сложения и вычитания матриц

- Операции ассоциативны и коммутативны: $A + B = B + A$, $(A + B) + C = A + (B + C)$.
- Вычитание матриц не является коммутативным: $A - B \neq B - A$.

15. Средства в NumPy для сложения и вычитания матриц

Для выполнения этих операций в NumPy можно использовать обычные операторы: $A + B$ $A - B$

16. Операция умножения матриц

Для матричного умножения в NumPy используется функция `np.dot()` или оператор `@`: `np.dot(A, B)` $A @ B$

17. Свойства операции умножения матриц

- Умножение не является коммутативным: $A * B \neq B * A$.
- Ассоциативность: $(A * B) * C = A * (B * C)$.
- Дистрибутивность: $A * (B + C) = A * B + A * C$.

18. Средства NumPy для умножения матриц

Для умножения матриц можно использовать:

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

19. Определитель матрицы

Определитель матрицы — это скаляр, связанный с матрицей, который даёт информацию о ее обратимости. Если определитель равен нулю, матрица необратима.

20. Средства NumPy для нахождения определителя

Для нахождения определителя матрицы используется функция `np.linalg.det()`: `np.linalg.det(A)`

21. Обратная матрица

Обратная матрица — это матрица, которая при умножении на исходную дает единичную матрицу. Для нахождения обратной матрицы используется метод `np.linalg.inv()`.

22. Свойства обратной матрицы

- Обратная матрица для произведения: $(A * B)^{-1} = B^{-1} * A^{-1}$.
- Обратная матрица для транспонированной матрицы: $(A^T)^{-1} = (A^{-1})^T$.

23. Средства NumPy для нахождения обратной матрицы

Для нахождения обратной матрицы в NumPy используется:
`np.linalg.inv(A)`

24. Метод Крамера для решения систем линейных уравнений

Алгоритм метода Крамера заключается в вычислении определителей матрицы коэффициентов и матриц, полученных заменой столбцов на столбец свободных членов. В NumPy можно вычислить определители и решить систему с помощью этого метода.

```
import numpy as np
A = np.array([[2, 1], [1, 3]])
b = np.array([1, 2])
det_A = np.linalg.det(A)
x1 = np.linalg.det(np.column_stack((b, A[:, 1]))) / det_A
x2 = np.linalg.det(np.column_stack((A[:, 0], b))) / det_A
```

25. Матричный метод для решения систем линейных уравнений

Матричный метод решения системы уравнений $Ax=b$ заключается в нахождении вектора x как $x=A^{-1}b$.

```
import numpy as np
A = np.array([[2, 1], [1, 3]])
b = np.array([1, 2])
x = np.linalg.inv(A).dot(b)
```

Вывод: исследовал базовые возможности библиотеки NumPy языка программирования Python.