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 14

Выполнил:
Степанов Артем Сергеевич
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 Роман Александрович

_____
(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

1. Выполнил задания из практической работы.

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. Реши задание в jupyter notebook формате

[4]:
```
import numpy as np


arraynp = np.arange(1, 10).reshape(3, 3)
arraynp = arraynp * 2
arraynp[arraynp > 10] = 0
print(arraynp)
```

```
[[ 2  4  6]
 [ 8 10  0]
 [ 0  0  0]]
```

Рисунок 1 – Решение задания №1

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.

[10]:
```
import numpy as np


array = np.random.randint(1, 101, 20)
print("Исходный массив:")
print(array)
divisible_by_5 = array[array % 5 == 0]
print("Элементы, которые делятся на 5:")
print(*divisible_by_5)
array[array % 5 == 0] = -1
print("Обновленный массив:")
print(array)
```

```
Исходный массив:
[ 13  92  79  20  33   9  50  21  37  23  50  31  33  29   4  15 100  57
  15  78]
Элементы, которые делятся на 5:
20 50 50 15 100 15
Обновленный массив:
[13 92 79 -1 33  9 -1 21 37 23 -1 31 33 29  4 -1 -1 57 -1 78]
```

Рисунок 2 – Решение задания №2

```
Элементы, которые делятся на 5:
20 50 50 15 100 15
Обновленный массив:
[13 92 79 -1 33  9 -1 21 37 23 -1 31 33 29  4 -1 -1 57 -1 78]
```

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.Выведите все промежуточные и итоговые результаты.

[17]:
```
import numpy as np

array1 = np.random.randint(0, 51, (1, 5))
array2 = np.random.randint(0, 51, (1, 5))

print("Исходный массив 1:")
print(array1)
print("Исходный массив 2:")
print(array2)
combined_array = np.vstack((array1, array2))
print("Объединенный массив (по строкам):")
print(combined_array)

split_array1, split_array2 = np.vsplit(combined_array, 2)

print("Разделенный массив 1:")
print(split_array1)
print("Разделенный массив 2:")
print(split_array2)
```

```
Исходный массив 1:
[[ 5 14 29  6  5]]
Исходный массив 2:
[[34 16  2 22 31]]
Объединенный массив (по строкам):
[[ 5 14 29  6  5]
 [34 16  2 22 31]]
Разделенный массив 1:
[[ 5 14 29  6  5]]
Разделенный массив 2:
[[34 16  2 22 31]]
```


Рисунок 3 – Решение задания №3

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

[28]:
```
import numpy as np


array = np.linspace(-10, 10, 50)
print("Массив:")
print(array)
total_sum = np.sum(array)
positive_sum = np.sum(array[array > 0])
negative_sum = np.sum(array[array < 0])
print("Сумма всех элементов:", total_sum)
print("Сумма положительных элементов:", positive_sum)
print("Сумма отрицательных элементов:", negative_sum)
# прикольный метод linspace, мне понравился)) но логически сумма же должна быть 0, если они равномерно распределены, тут хочу уточнить момент
```

```
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 4 – Решение задания №4

Создайте: **Единичную матрицу размером 4×4. Диагональную матрицу размером 4×4 с диагональными элементами [5, 10, 15, 20] (не использовать циклы)** *Найдите сумму всех элементов каждой из этих матриц и сравните результаты*

[33]:
```
import numpy as np


edin_mat = np.eye(4)
print("Единичная матрица:")
print(edin_mat)
diag_mat = np.diag([5, 10, 15, 20])
print("Диагональная матрица:")
print(diag_mat)
summa_edin = np.sum(edin_mat)
sum_diagonal = np.sum(diag_mat)
print("Сумма всех элементов единичной матрицы:", summa_edin)
print("Сумма всех элементов диагональной матрицы:", sum_diagonal)
if summa_edin > sum_diagonal:
    print("Сумма элементов единичной матрицы больше.")
elif summa_edin < sum_diagonal:
    print("Сумма элементов диагональной матрицы больше.")
else:
    print("Суммы элементов матриц равны.")
```

```
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
Сумма элементов диагональной матрицы больше.
```

Рисунок 5 – Решение задания №5

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

[36]:
```
import numpy as np


matrix1 = np.random.randint(1, 21, (3, 3))
matrix2 = np.random.randint(1, 21, (3, 3))
print("Матрица 1:")
print(matrix1)
print("Матрица 2:")
print(matrix2)
matrix_sum = matrix1 + matrix2
print("Сумма матриц:")
print(matrix_sum)
matrix_razn = matrix1 - matrix2
print("Разность матриц:")
print(matrix_razn)
matrix_proiz = matrix1 * matrix2
print("Поэлементное произведение матриц:")
print(matrix_proiz)
```

```
Матрица 1:
[[19  9  1]
 [13 12  1]
 [14 10  2]]
Матрица 2:
[[ 3  5  2]
 [ 9 11  5]
 [19 16 10]]
Сумма матриц:
[[22 14  3]
 [22 23  6]
 [33 26 12]]
Разность матриц:
[[16  4 -1]
 [ 4  1 -4]
 [-5 -6 -8]]
Поэлементное произведение матриц:
[[ 57  45   2]
 [117 132   5]
 [266 160  20]]
```

Рисунок 6 – Решение задания №6

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

[39]:
```
import numpy as np


matrix_A = np.random.randint(1, 11, (2, 3))
matrix_B = np.random.randint(1, 11, (3, 2))

print("Матрица A:")
print(matrix_A)
print("Матрица B:")
print(matrix_B)
matrix_proiz = np.dot(matrix_A, matrix_B)
print("Результат матричного умножения A @ B:")
print(matrix_proiz)
```

```
Матрица A:
[[9 1 1]
 [3 1 7]]
Матрица B:
[[5 4]
 [8 6]
 [4 5]]
Результат матричного умножения A @ B:
[[57 47]
 [51 53]]
```

Рисунок 7 – Решение задания №7

Создайте случайную квадратную матрицу 3×3. Найдите и выведите:Определитель этой матрицы Обратную матрицу (если существует, иначе выведите сообщение, что матрица вырождена)Используйте функции np.linalg.det и np.linalg.inv

[45]:
```
import numpy as np


matrix_rand = np.random.randint(1, 100, (3, 3)) # я сделал от 1 до 100, не знаю какую надо
print("Случайная матрица:")
print(matrix_rand)

D = np.linalg.det(matrix_rand)
print("Определитель матрицы:")
print(D)
if D != 0:
    matrix_obr = np.linalg.inv(matrix_rand)
    print("Обратная матрица:")
    print(matrix_obr)
else:
    print("Матрица вырождена(нет обратной)")
```

```
Случайная матрица:
[[66 21 67]
 [40 37 63]
 [31 79 29]]
Определитель матрицы:
-106139.99999999985
Обратная матрица:
[[ 0.03678161 -0.04413039  0.01089128]
 [-0.00747126  0.00153571  0.013925  ]
 [-0.01896552  0.04299039 -0.01509327]]
```

Рисунок 8 – Решение задания №8

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). Используйте np.trace для нахождения следа.

48]:
```
import numpy as np

matrix = np.random.randint(1, 51, (4, 4))
print("Исходная матрица:")
print(matrix)
matrix_tran = matrix.T
print("Транспонированная матрица:")
print(matrix_tran)
sled = np.trace(matrix)
print("След матрицы:")
print(sled)
```

```
Исходная матрица:
[[38 28  7 50]
 [34 27 46 29]
 [27 27 43  7]
 [16 32 41 36]]
Транспонированная матрица:
[[38 34 27 16]
 [28 27 27 32]
 [ 7 46 43 41]
 [50 29  7 36]]
След матрицы:
144
```

Рисунок 9 – Решение задания №9

Решите систему линейных уравнений вида:(система уравнений) 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.

51]:
```
import numpy as np

A = np.array([
    [2, 3, -1],
    [4, -1, 2],
    [-3, 5, 4]
])
B = np.array([5, 6, -2])
x = np.linalg.solve(A, B)
print(f"x = {x[0]}")
print(f"y = {x[1]}")
print(f"z = {x[2]}")
```

```
x = 1.6396396396396398
y = 0.5765765765765767
z = 0.009009009009008978
```

Рисунок 10 – Решение задания №10

2. Индивидуальное задание

[1]:
```
import numpy as np
A = np.array([
    [2, 1, 1],
    [1, 1, 1],
    [1, 1, 1]
])
B = np.array([500, 500, 500])
# Определитель матрицы A
det_A = np.linalg.det(A)
print(det_A) # это я уже проверял из-за чего ошибка, в итоге пришел к выводу, что матрица вырождена и решение невозможно
# Матрицы для метода Крамера
A_copy_Kramer = A.copy()
A_copy_Kramer[:, 0] = B
det_A1 = np.linalg.det(A_copy_Kramer)

A_copy_Kramer2 = A.copy()
A_copy_Kramer2[:, 1] = B
det_A2 = np.linalg.det(A_copy_Kramer2)

A_copy_Kramer3 = A.copy()
A_copy_Kramer3[:, 2] = B
det_A3 = np.linalg.det(A_copy_Kramer3)

if det_A == 0:
    if det_A1 == 0 and det_A2 == 0 and det_A3 == 0:
        print("Решение имеет бесконечное количество решений")
    else:
        print("Решений нет")
else:
    # Решение системы(суть в делении каждого определителя,
    # подставляя значения из B вместо столбца, стоящего на i-ой позиции, на общий определитель)
    x1 = det_A1 / det_A
    x2 = det_A2 / det_A
    x3 = det_A3 / det_A
    print("Метод Крамера:")
    print(f"x1 = {x1}, x2 = {x2}, x3 = {x3}")
```

```
0.0
Решение имеет бесконечное количество решений
```

Рисунок 11 – Неработающий метод Крамера

Метод Матричный

[60]:
```
import numpy as np


A = np.array([
    [2, 1, 1],
    [1, 1, 1],
    [1, 1, 1]
])
B = np.array([500, 500, 500])

A_obr = np.linalg.inv(A)

# Решение системы
x = A_obr @ B

print("\nМатричный метод:")
print(f"x1 = {x[0]}, x2 = {x[1]}, x3 = {x[2]}")
# Тоже не работает естественно из-за определителя=0
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[60], line 11
      4 A = np.array([
      5     [2, 1, 1],
      6     [1, 1, 1],
      7     [1, 1, 1]
      8 ])
      9 B = np.array([500, 500, 500])
---> 11 A_obr = np.linalg.inv(A)
     13 # Решение системы
     14 x = A_obr @ B

File D:\Anaconda\Lib\site-packages\numpy\linalg\linalg.py:561, in inv(a)
    559 signature = 'D->D' if isComplexType(t) else 'd->d'
    560 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 561 ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    562 return wrap(ainv.astype(result_t, copy=False))

File D:\Anaconda\Lib\site-packages\numpy\linalg\linalg.py:112, in _raise_linalgerror_singular(err, flag)
    111 def _raise_linalgerror_singular(err, flag):
--> 112     raise LinAlgError("Singular matrix")
```

Рисунок 12 – Неработающий Матричный метод

Использование функции тоже не дало результата

[63]:
```
import numpy as np


A = np.array([
    [2, 1, 1],
    [1, 1, 1],
    [1, 1, 1]
])
B = np.array([500, 500, 500])

# Решение системы
x = np.linalg.solve(A, B)

print("Метод np.linalg.solve:")
print(f"x1 = {x[0]}, x2 = {x[1]}, x3 = {x[2]}")
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[63], line 12
      9 B = np.array([500, 500, 500])
     11 # Решение системы
---> 12 x = np.linalg.solve(A, B)
     14 print("Метод np.linalg.solve:")
     15 print(f"x1 = {x[0]}, x2 = {x[1]}, x3 = {x[2]}")

File D:\Anaconda\Lib\site-packages\numpy\linalg\linalg.py:409, in solve(a, b)
    407 signature = 'DD->D' if isComplexType(t) else 'dd->d'
    408 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 409 r = gufunc(a, b, signature=signature, extobj=extobj)
    411 return wrap(r.astype(result_t, copy=False))

File D:\Anaconda\Lib\site-packages\numpy\linalg\linalg.py:112, in _raise_linalgerror_singular(err, flag)
    111 def _raise_linalgerror_singular(err, flag):
--> 112     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```

Рисунок 13 – Неработающий метод, с использованием np.linalg.solve

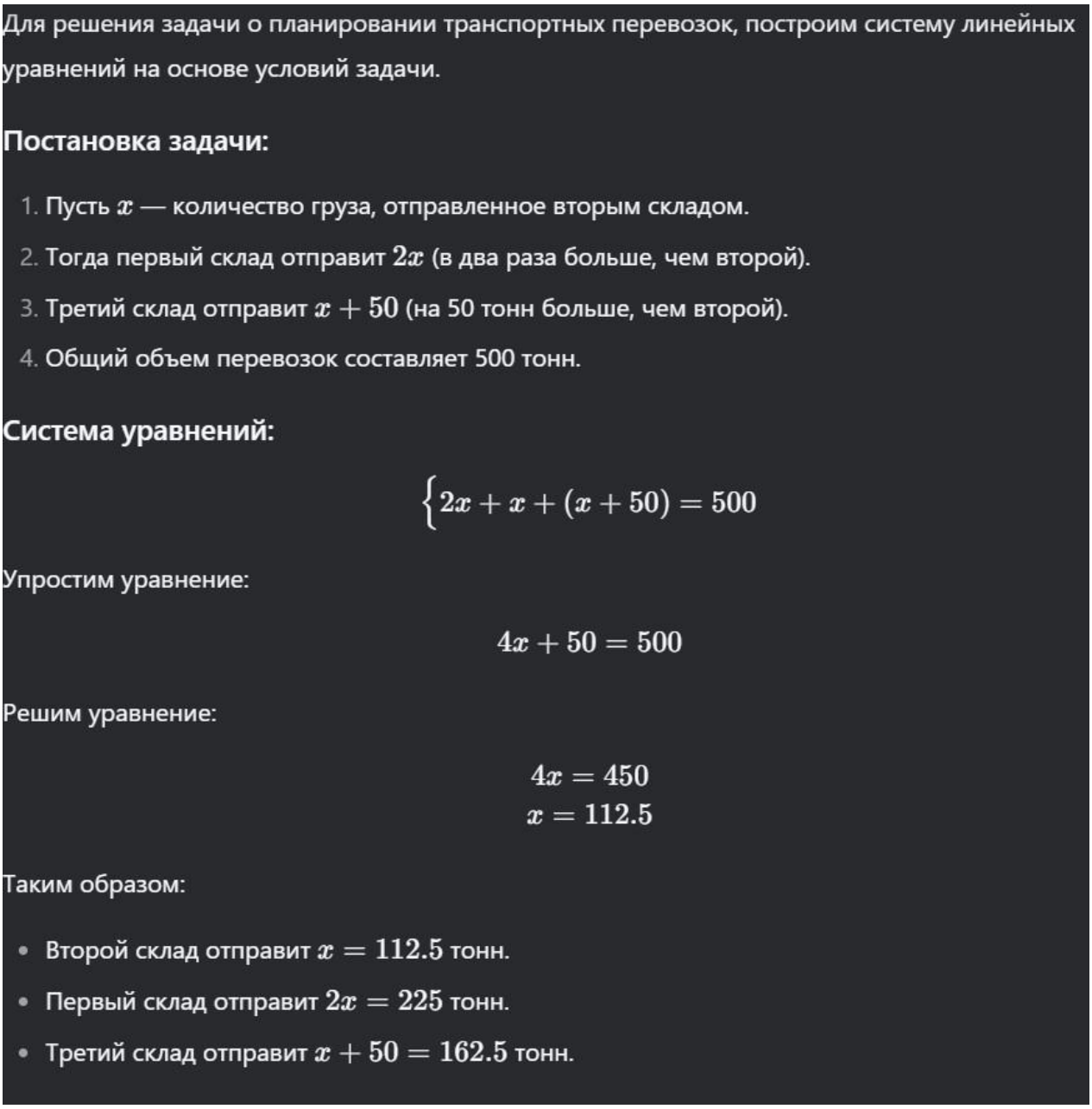

Рисунок 14 – Решенное задание «руками»

Ссылка на репозиторий GitHub:

https://github.com/ArtemStepanovNkey/AI-university/tree/main

Ответы на контрольные вопросы:

1. **Назначение библиотеки NumPy**: NumPy предназначена для работы с многомерными массивами и матрицами, а также для выполнения математических операций над ними. Она обеспечивает высокую производительность и удобство работы с числовыми данными.

2. **Массивы ndarray**: Это многомерные массивы в NumPy, которые позволяют хранить и обрабатывать данные одинакового типа. Они поддерживают операции линейной алгебры, статистики и другие математические функции.

3. **Доступ к частям многомерного массива**: Доступ осуществляется с помощью индексации и срезов. Например, array[i, j] для доступа к элементу или array[:, 1] для доступа ко второму столбцу.

4. **Расчет статистик по данным**: NumPy предоставляет функции для вычисления статистик, такие как np.mean(), np.median(), np.std(), np.sum() и другие.

5. **Выборка данных из массивов ndarray**: Используется индексация и срезы, например, array[array > 5] для выборки элементов, больших 5.

6. **Основные виды матриц и векторов**: Векторы (одномерные массивы) и матрицы (двумерные массивы). Создаются с помощью np.array(), np.zeros(), np.ones(), np.eye() и других функций.

7. **Транспонирование матриц**: Выполняется с помощью атрибута .T или функции np.transpose().

8. **Свойства операции транспонирования**: (AT)T=A(*AT*)*T*=*A*, (A+B)T=AT+BT, (kA) T=kAT, (AB)T=BTAT.

9. **Средства для транспонирования в NumPy**: Атрибут .T и функция np.transpose().

10. **Основные действия над матрицами**: Сложение, вычитание, умножение на число, умножение матриц, транспонирование, нахождение определителя и обратной матрицы.

11. **Умножение матрицы на число**: Выполняется поэлементно, например, A * 2.

12. **Свойства умножения матрицы на число**: k(A+B)=kA+kB, (k+m)A=kA+mA, k(mA) =(km)A.

13. **Сложение и вычитание матриц**: Выполняется поэлементно, если матрицы одинакового размера, например, A + B.

14. **Свойства сложения и вычитания матриц**: Коммутативность A+B=B+A, ассоциативность (A+B)+C=A+(B+C).

15. **Средства для сложения и вычитания в NumPy**: Операторы + и -

16. **Умножение матриц**: Выполняется с помощью функции np.dot() или оператора @, например, A @ B.

17. **Свойства умножения матриц**: Ассоциативность (AB)C=A(BC), дистрибутивность A(B+C)=AB+AC.

18. **Средства для умножения матриц в NumPy**: Функция np.dot() и оператор @.

19. **Определитель матрицы**: Это скалярное значение, которое характеризует свойства матрицы. Свойства:

det(AB)=det(A)det(B)det(*AB*)=det(*A*)det(*B*), det(AT)=det(A)det(*AT*)=det(*A*).

20. **Средства для нахождения определителя в NumPy**:

Функция np.linalg.det().

21. **Обратная матрица**: Это матрица, которая при умножении на исходную дает единичную матрицу. Находится с помощью np.linalg.inv().

22. **Свойства обратной матрицы**:

(A−1)−1=A, (AB)−1=B−1A−1.

23. **Средства для нахождения обратной матрицы в NumPy**:

Функция np.linalg.inv().

24. **Метод Крамера**: Решение системы линейных уравнений через определители. В NumPy можно реализовать через вычисление определителей подматриц.

25. **Матричный метод**: Решение системы Ax=B через x=A−1B. В NumPy используется np.linalg.solve() или np.linalg.inv() @ B.