

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 9

Выполнил:
Кравчук Мирослав Витальевич
2 курс, группа ИТС-б-о-23-1,
11.03.02 «Инфокоммуникационные
технологии и системы связи», очная
форма обучения

(подпись)

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

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2025 г.

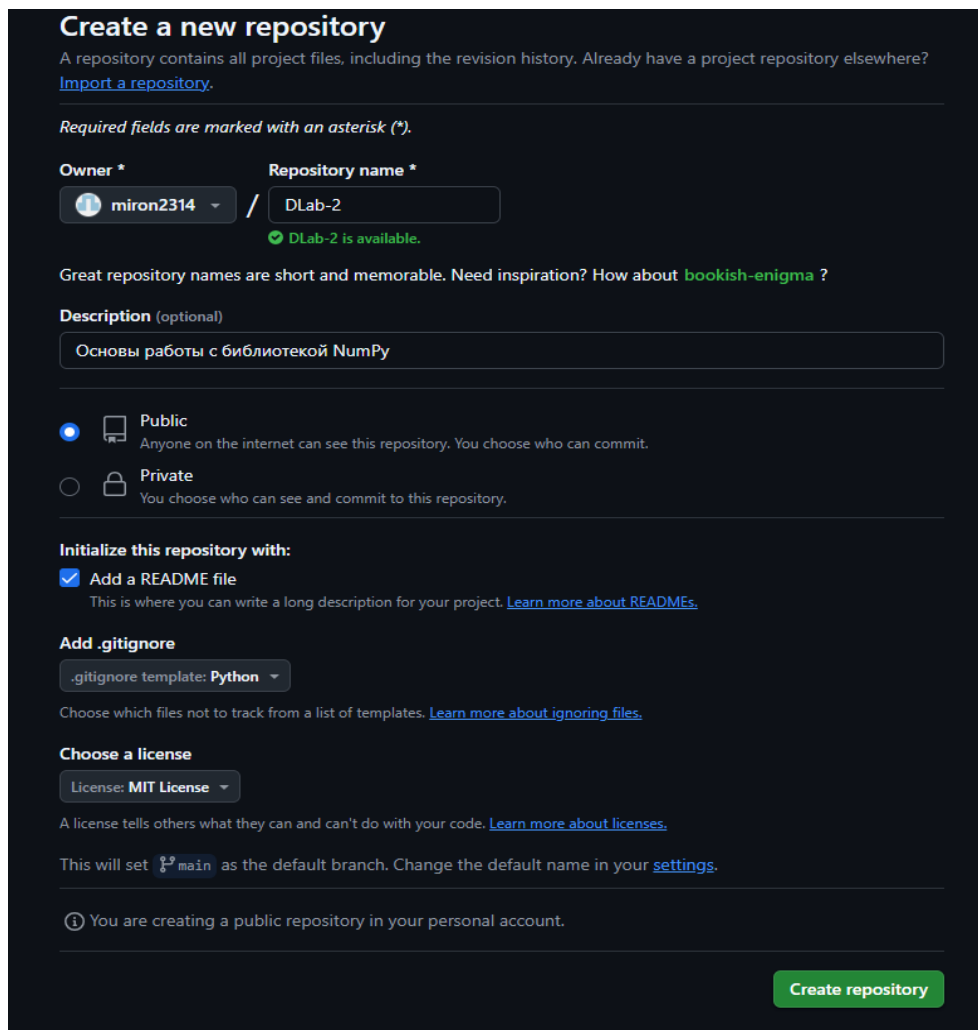
Тема: Основы работы с библиотекой Numpy

Цель: исследовать базовые возможности библиотеки NumPy языка программирования Python

Ссылка на GitHub: <https://github.com/miron2314/DLab-2.git>

Порядок выполнения работы:

1. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и выбранный язык программирования.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner * miron2314 / **Repository name *** DLab-2
✔ DLab-2 is available.

Great repository names are short and memorable. Need inspiration? How about **bookish-enigma** ?

Description (optional)
Основы работы с библиотекой NumPy

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: Python

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license
License: MIT License

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

ⓘ You are creating a public repository in your personal account.

Create repository

Рисунок 1. Репозиторий

3. Выполнил клонирование репозитория.

```

PS C:\Users\USER> git clone https://github.com/miron2314/DLab-2.git
Cloning into 'DLab-2'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), done.
PS C:\Users\USER>

```

Рисунок 2. Клонирование

4.Проработал примеры работы.

```

[ ] #Векторизация
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = a + b #быстрое сложение

[ ] #Создание массивов
import numpy as np
arr = np.array([1, 2, 3, 4])

[ ] #Операции с массивами
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
result = arr1 + arr2 # [5, 7, 9]

[ ] #Матричные операции
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])
result = np.dot(matrix1, matrix2) # Матричное умножение

[ ] #Статические функции
data = np.array([1, 2, 3, 4, 5])
mean = np.mean(data) # Среднее значение
std_dev = np.std(data) # Стандартное отклонение

[ ] #Поддержка векторизации
import numpy as np
arr = np.array([1, 2, 3, 4])
result = arr * 2 # Быстрое поэлементное умножение

[ ] #Создание матрицы
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]]

```

Рисунок 3. Проработка примеров

```
[ ] #Элемент матрицы с заданным координатами
import numpy as np
m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
print(m[1,0])
```

⇒ 5

```
▶ #Строка массива
import numpy as np
m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
print(m[1,:])
```

⇒ [[5 6 7 8]]

```
[ ] #Столбец массива
import numpy as np
m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
print(m[:,3])
```

⇒ [[4]
[8]
[7]]

```
[ ] #Часть строки массива
import numpy as np
m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
print(m[1,2:])
```

⇒ [[7 8]]

```
[ ] #Часть столбца массива
import numpy as np
m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
print(m[0:2,1])
```

⇒ [[2]
[6]]

Рисунок 4. Проработка примеров

```
[ ] #Непрерывная часть массива
import numpy as np
m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
print(m[0:2,1:3])
```

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

```
[ ] #Произвольные столбцы и строки массива
import numpy as np
m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
cols=[0,1,3]
print(m[:,cols])
```

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

```
#Расчет статистик по данным в массиве
m = np.matrix('1 2 3 4; 5 6 7 8; 9 1 5 7')
m.shape
```

```
(3, 4)
```

```
[ ] #Использование boolean массива для доступа к ndarray
nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
letters = np.array(['a', 'b', 'c', 'd', 'a', 'e', 'b'])
a = True
b = 5 > 7
print(b)
```

```
False
```

```
[ ] #Использование boolean массива для доступа к ndarray
nums = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
letters = np.array(['a', 'b', 'c', 'd', 'a', 'e', 'b'])
less_than_5 = nums < 5
less_than_5
```

```
array([ True,  True,  True,  True, False, False, False, False, False,
```

Рисунок 5. Проработка примеров

```
[ ] #np.arange()
    np.arange(1, 5, 0.5)

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

[ ] np.arange(10)

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

[ ] np.arange(6,20)

⇒ array([ 6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])

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

⇒ matrix([[1, 2],
          [3, 4]])

▶ #np.zeros(), np.eye()
  np.zeros((3, 4))

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

[ ] np.eye(4)

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

Рисунок 6. Проработка примеров

```
[ ] #np.ravel()
np.ravel(A, order='F')

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

[ ] #np.random()
a = np.random.rand(10)
a

array([0.3590862 , 0.57599732, 0.37594434, 0.42317349, 0.91038402,
       0.92238252, 0.54318537, 0.18100887, 0.26194715, 0.58904987])

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

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

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

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

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

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

[ ] #np.meshgrid()
x = np.linspace(0, 1, 5)
y = np.linspace(0, 2, 5)
xg, yg = np.meshgrid(x, y)
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(xg, yg, color="r", marker="*", linestyle="none")

[<matplotlib.lines.Line2D at 0x7dd98e734e50>,
```

Рисунок 7. Проработка примеров

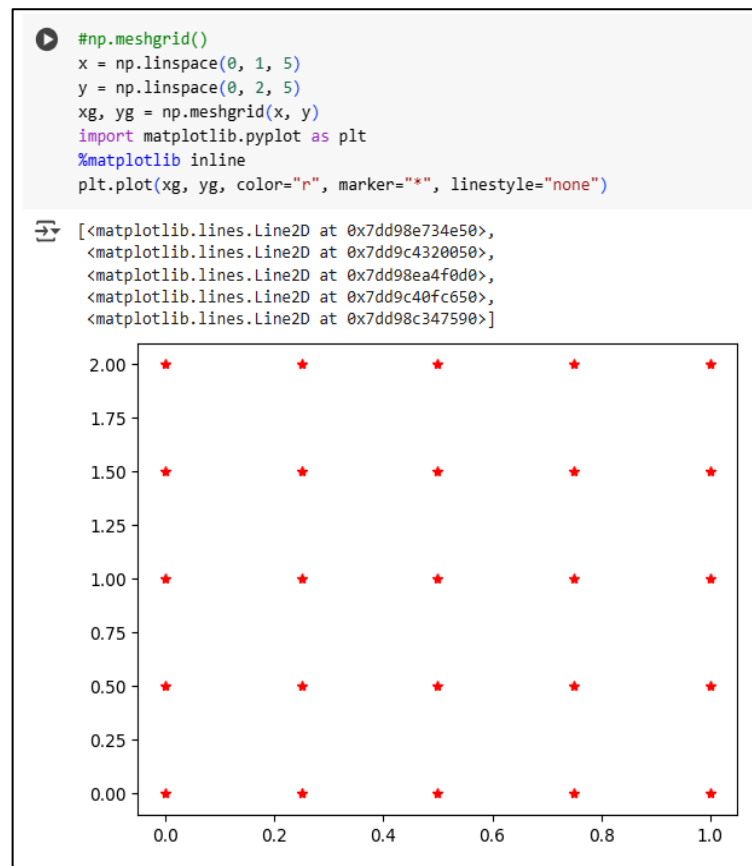


Рисунок 8. Проработка примеров

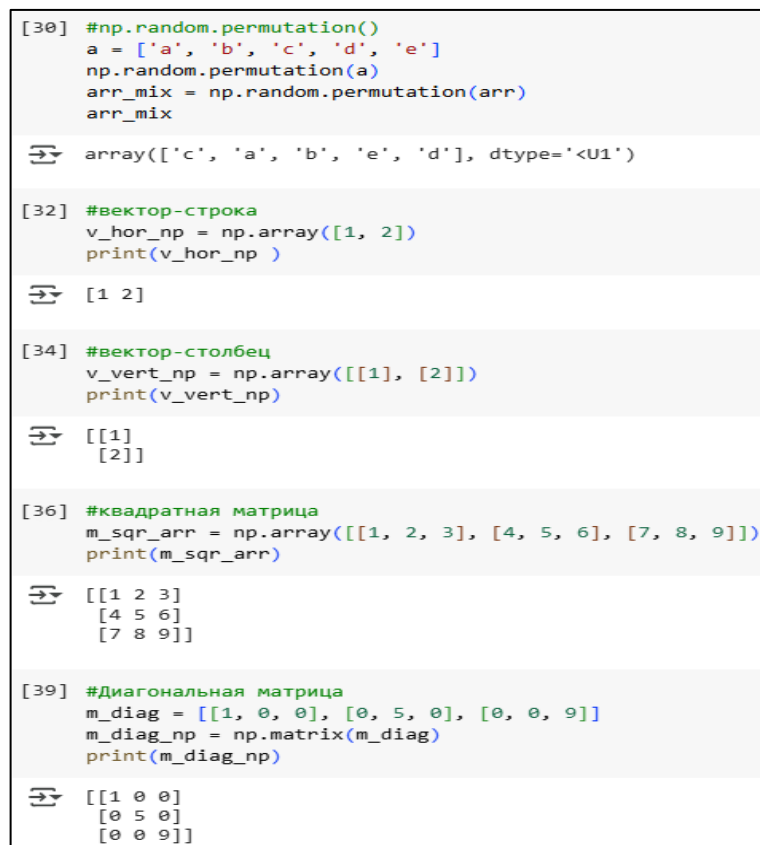


Рисунок 9. Проработка примеров


```
[41] #Единичная матрица
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]]

[43] #Нулевая матрица
m_zeros = np.zeros((3, 3))
print(m_zeros)

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

[45] #Задание матрицы в общем виде
m_mx = np.matrix('1 2 3; 4 5 6')
print(m_mx)

⇌ [[1 2 3]
    [4 5 6]]

[47] #Транспонирование матрицы
A = np.matrix('1 2 3; 4 5 6')
print(A)

⇌ [[1 2 3]
    [4 5 6]]

[51] #Транспонирование суммы матриц равно сумме транспонированных матриц
A = np.matrix('1 2 3; 4 5 6')
B = np.matrix('7 8 9; 0 7 5')
L = (A + B).T
R = A.T + B.T
print(L)
print(R)

⇌ [[ 8  4]
    [10 12]
    [12 11]]
```

Рисунок 10. Проработка примеров

```
[51] #Транспонирование суммы матриц равно сумме транспонированных матриц
A = np.matrix('1 2 3; 4 5 6')
B = np.matrix('7 8 9; 0 7 5')
L = (A + B).T
R = A.T + B.T
print(L)
print(R)

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

[53] #Умножение матрицы на число
A = np.matrix('1 2 3; 4 5 6')
C = 3 * A
print(C)

⇌ [[ 3  6  9]
    [12 15 18]]

▶ #Произведение заданной матрицы на единичную равно исходной матрице
A = np.matrix('1 2; 3 4')
E = np.matrix('1 0; 0 1')
L = E.dot(A)
R = A.dot(E)
print(L)
print(R)
print(A)

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

Рисунок 11. Проработка примеров

```
#Обратная матрица
A = np.matrix('1 -3; 2 5')
A_inv = np.linalg.inv(A)
print(A_inv)

[[ 0.45454545  0.27272727]
 [-0.18181818  0.09090909]]

[59] #Ранг матрицы
m_eye = np.eye(4)
print(m_eye)

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

Рисунок 12. Проработка примеров

5.Выполнил практические задания.

Задание 1. Создание и изменение массива

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.

Листинг кода:

```
import numpy as np
arr = np.arange(1, 10).reshape((3, 3))
print("Исходный массив:")
print(arr)
arr *= 2
print("\nУмноженный массив:")
print(arr)
arr[arr > 10] = 0
print("\nИтоговый массив:")
print(arr)
```

```
#Задание 1
import numpy as np
arr = np.arange(1, 10).reshape((3, 3))
print("Исходный массив:")
print(arr)
arr *= 2
print("\nУмноженный массив:")
print(arr)
arr[arr > 10] = 0
print("\nИтоговый массив:")
print(arr)
```

Исходный массив:

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

Умноженный массив:

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

Итоговый массив:

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

Рисунок 13. Выполнение задания 1

Задание 2. Работа с булевыми масками

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.

Листинг кода:

```
#Задание 2
import numpy as np
a = np.random.randint(1, 101, size=20)
print("Полученный массив\n", a)
delenie = a[a % 5 == 0]
print("Числа, делящиеся на 5\n", delenie)
a[a % 5 == 0] = -1
print("Обновленный массив\n", a)
```

```
#Задание 2
import numpy as np
a = np.random.randint(1, 101, size=20)
print("Полученный массив\n", a)
delenie = a[a % 5 == 0]
print("Числа,делящиеся на 5\n",delenie)
a[a % 5 == 0] = -1
print("Обновленный массив\n",a)
```

Полученный массив
[24 23 77 59 32 77 35 10 6 38 23 70 56 2 100 36 80 23
11 10]
Числа,делящиеся на 5
[35 10 70 100 80 10]
Обновленный массив
[24 23 77 59 32 77 -1 -1 6 38 23 -1 56 2 -1 36 -1 23 11 -1]

Рисунок 14. Практическое задание 2

Задание 3. Объединение и разбиение массивов

Создайте два массива NumPy размером 1×5 , заполненные случайными числами от 0 до 50. Объедините эти массивы в один двумерный массив. Разделите полученный массив на два массива, каждый из которых содержит 5 элементов. Выведите все промежуточные и итоговые результаты.

Листинг кода:

```
#Задание 3
import numpy as np
a = np.random.randint(0, 51, size=(1, 5))
a1 = np.random.randint(0, 51, size=(1, 5))
summa = np.concatenate((a, a1))
pazd = np.split(summa, 2)
print("Первый массив:", a)
print("Второй массив:", a1)
print("Объединённый массив:\n", summa)
print("Разделённые массивы:")
for i, arr in enumerate(pazd):
    print(f"Массив {i+1}:", a)
```

```

#Задание 3
import numpy as np
a = np.random.randint(0, 51, size=(1, 5))
a1 = np.random.randint(0, 51, size=(1, 5))
summa = np.concatenate((a, a1))
pazd = np.split(summa, 2)
print("Первый массив:", a)
print("Второй массив:", a1)
print("Объединённый массив:\n", summa)
print("Разделённые массивы:")
for i, arr in enumerate(pazd):
    print(f"Массив {i+1}:", a)

Первый массив: [[29 38 46  8 32]]
Второй массив: [[43 10 27 10  2]]
Объединённый массив:
[[29 38 46  8 32]
 [43 10 27 10  2]]
Разделённые массивы:
Массив 1: [[29 38 46  8 32]]
Массив 2: [[29 38 46  8 32]]

```

Рисунок 15. Практическое задание 3

Задание 4. Генерация и работа с линейными последовательностями

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

Листинг кода:

```

#Задание 4
import numpy as np
a = np.linspace(-10, 10, 50)
print('Полученный массив:\n',a)
sum_all = a.sum()
positive_elements = a[a > 0]
negative_elements = a[a < 0]
sum_positive = positive_elements.sum()
sum_negative = negative_elements.sum()

print("Сумма всех элементов:", sum_all)
print("Сумма положительных элементов:", sum_positive)
print("Сумма отрицательных элементов:", sum_negative)

```

```
#Задание 4
import numpy as np
a = np.linspace(-10, 10, 50)
print('Полученный массив:\n',a)
sum_all = a.sum()
positive_elements = a[a > 0]
negative_elements = a[a < 0]
sum_positive = positive_elements.sum()
sum_negative = negative_elements.sum()

print("Сумма всех элементов:", sum_all)
print("Сумма положительных элементов:", sum_positive)
print("Сумма отрицательных элементов:", sum_negative)
```

Полученный 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 16. Практическое задание 4

Задание 5. Работа с диагональными и единичными матрицами

Создайте:

- 1.Единичную матрицу размером 4×4.
- 2.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.

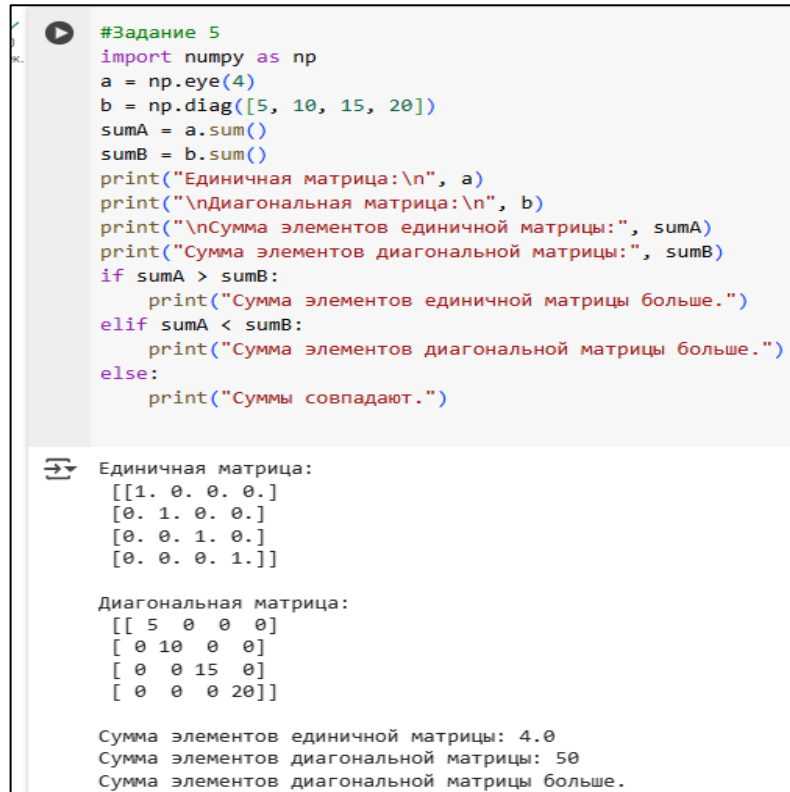
Листинг кода:

```
#Задание 5
import numpy as np
a = np.eye(4)
b = np.diag([5, 10, 15, 20])
sumA = a.sum()
sumB = b.sum()
print("Единичная матрица:\n", a)
print("\nДиагональная матрица:\n", b)
print("\nСумма элементов единичной матрицы:", sumA)
print("Сумма элементов диагональной матрицы:", sumB)
```

```

if sumA > sumB:
    print("Сумма элементов единичной матрицы больше.")
elif sumA < sumB:
    print("Сумма элементов диагональной матрицы больше.")
else:
    print("Суммы совпадают.")

```



```

#Задание 5
import numpy as np
a = np.eye(4)
b = np.diag([5, 10, 15, 20])
sumA = a.sum()
sumB = b.sum()
print("Единичная матрица:\n", a)
print("\nДиагональная матрица:\n", b)
print("\nСумма элементов единичной матрицы:", sumA)
print("Сумма элементов диагональной матрицы:", sumB)
if sumA > sumB:
    print("Сумма элементов единичной матрицы больше.")
elif sumA < sumB:
    print("Сумма элементов диагональной матрицы больше.")
else:
    print("Суммы совпадают.")

```

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
Сумма элементов диагональной матрицы больше.

Рисунок 17. Практическое задание 5

Задание 6. Создание и базовые операции с матрицами

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

1. Их сумму
2. Их разность
3. Их поэлементное произведение

Листинг кода:

```

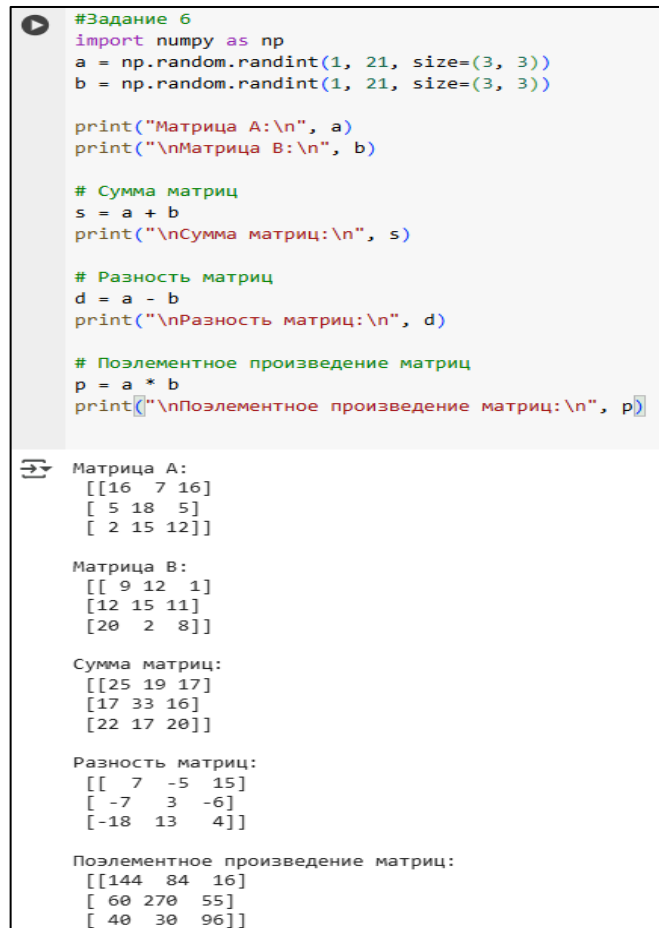
#Задание 6
import numpy as np
a = np.random.randint(1, 21, size=(3, 3))
b = np.random.randint(1, 21, size=(3, 3))

```

```

print("Матрица A:\n", a)
print("\nМатрица B:\n", b)
s = a + b
print("\nСумма матриц:\n", s)
d = a - b
print("\nРазность матриц:\n", d)
p = a * b
print("\nПоэлементное произведение матриц:\n", p)

```



```

#Задание 6
import numpy as np
a = np.random.randint(1, 21, size=(3, 3))
b = np.random.randint(1, 21, size=(3, 3))

print("Матрица A:\n", a)
print("\nМатрица B:\n", b)

# Сумма матриц
s = a + b
print("\nСумма матриц:\n", s)

# Разность матриц
d = a - b
print("\nРазность матриц:\n", d)

# Поэлементное произведение матриц
p = a * b
print("\nПоэлементное произведение матриц:\n", p)

```

Матрица A:

```
[[16  7 16]
 [ 5 18  5]
 [ 2 15 12]]
```

Матрица B:

```
[[ 9 12  1]
 [12 15 11]
 [20  2  8]]
```

Сумма матриц:

```
[[25 19 17]
 [17 33 16]
 [22 17 20]]
```

Разность матриц:

```
[[ 7 -5 15]
 [-7  3 -6]
 [-18 13  4]]
```

Поэлементное произведение матриц:

```
[[144  84 16]
 [ 60 270 55]
 [ 40  30 96]]
```

Рисунок 18. Практическое задание 6

Задание 7. Умножение матриц

Создайте две матрицы NumPy:

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

Выполните матричное умножение (@ или np.dot) и выведите результат.

Листинг кода:

```

#Задание 7
import numpy as np

```



```

A = np.random.randint(1, 11, size=(2, 3))
B = np.random.randint(1, 11, size=(3, 2))
result = np.dot(A, B)
print("Первая матрица (2x3):\n", A)
print("\nВторая матрица (3x2):\n", B)
print("\nРезультат матричного умножения (2x2):\n", result)

```

```

#Задание 7
import numpy as np
A = np.random.randint(1, 11, size=(2, 3))
B = np.random.randint(1, 11, size=(3, 2))
result = np.dot(A, B)
print("Первая матрица (2x3):\n", A)
print("\nВторая матрица (3x2):\n", B)
print("\nРезультат матричного умножения (2x2):\n", result)

Первая матрица (2x3):
[[ 7 10  5]
 [10  1  1]]

Вторая матрица (3x2):
[[5 8]
 [1 7]
 [6 8]]

Результат матричного умножения (2x2):
[[ 75 166]
 [ 57  95]]

```

Рисунок 19. Практическое задание 7

Задание 8. Определитель и обратная матрица

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

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

Используйте функции `np.linalg.det` и `np.linalg.inv`.

Листинг кода:

```

#Задание 8
import numpy as np
a = np.random.randint(1, 11, size=(3, 3))
det = np.linalg.det(a)
print("Исходная матрица:")

```

```

print(a)
print(f"\nОпределитель матрицы: {det:.2f}")
if abs(det) > 1e-10:
    i = np.linalg.inv(a)
    print("\nОбратная матрица:")
    print(i)
else:
    print("\nМатрица вырождена, обратной матрицы не существует.")

```

```

#Задание 8
import numpy as np
a = np.random.randint(1, 11, size=(3, 3))
det = np.linalg.det(a)
print("Исходная матрица:")
print(a)
print(f"\nОпределитель матрицы: {det:.2f}")
if abs(det) > 1e-10:
    i = np.linalg.inv(a)
    print("\nОбратная матрица:")
    print(i)
else:
    print("\nМатрица вырождена, обратной матрицы не существует.")

```

Исходная матрица:

```

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

```

Определитель матрицы: -202.00

Обратная матрица:

```

[[-0.16831683 -0.01980198  0.23762376]
 [-0.06930693  0.16831683 -0.01980198]
 [ 0.24752475 -0.02970297 -0.14356436]]

```

Рисунок 20. Практическое задание 8

Задание 9. Транспонирование и след матрицы

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

1. Исходную матрицу
2. Транспонированную матрицу
3. След матрицы (сумму элементов на главной диагонали)

Используйте `np.trace` для нахождения следа.

Листинг кода:

```

#Задание 9
import numpy as np
matrix = np.random.randint(1, 51, size=(4, 4))

```

```

print("Исходная матрица:")
print(matrix)
transposed_matrix = matrix.T
print("\nТранспонированная матрица:")
print(transposed_matrix)
trace = np.trace(matrix)
print(f"\nСлед матрицы: {trace}")

```

```

#Задание 9
import numpy as np
matrix = np.random.randint(1, 51, size=(4, 4))
print("Исходная матрица:")
print(matrix)
transposed_matrix = matrix.T
print("\nТранспонированная матрица:")
print(transposed_matrix)
trace = np.trace(matrix)
print(f"\nСлед матрицы: {trace}")

```

Исходная матрица:

```

[[41 33 44 41]
 [24  3 13 28]
 [21  4 28  9]
 [19 22 16 25]]

```

Транспонированная матрица:

```

[[41 24 21 19]
 [33  3  4 22]
 [44 13 28 16]
 [41 28  9 25]]

```

След матрицы: 97

Рисунок 21. Практическое задание 9

Задание 10. Системы линейных уравнений

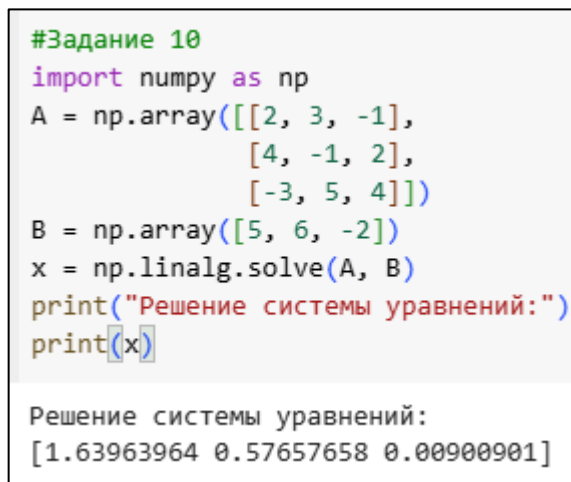
Решите систему линейных уравнений вида:

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

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.

Листинг кода:

```
#Задание 10
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("Решение системы уравнений:")
print(x)
```



```
#Задание 10
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("Решение системы уравнений:")
print(x)
```

Решение системы уравнений:
[1.63963964 0.57657658 0.00900901]

Рисунок 22. Практическое задание 10

6. Выполнил индивидуальное задание.

Задание 11. Индивидуальное задание

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`

Вариант 9

Оптимальный выбор топлива. Автомобиль использует три вида топлива в разных пропорциях. Бензин даёт 30 МДж энергии на литр, дизель — 35 МДж, газ — 25 МДж. Для поездки требуется 100 МДж энергии. Водитель

хочет использовать бензина в два раза больше, чем газа, а количество дизеля равно количеству газа. Сколько литров каждого топлива нужно залить?

Листинг кода:

```
#Индивидуальное задание Вариант 10
import numpy as np
A = np.array([
    [30, 35, 25],
    [1, 0, -2],
    [0, 1, -1]
])
B = np.array([100, 0, 0])
solution = np.linalg.solve(A, B)
print("Количество литров бензина:", solution[0])
print("Количество литров дизеля:", solution[1])
print("Количество литров газа:", solution[2])
```

```
#Индивидуальное задание Вариант 10
import numpy as np
A = np.array([
    [30, 35, 25],
    [1, 0, -2],
    [0, 1, -1]
])
B = np.array([100, 0, 0])
solution = np.linalg.solve(A, B)
print("Количество литров бензина:", solution[0])
print("Количество литров дизеля:", solution[1])
print("Количество литров газа:", solution[2])

Количество литров бензина: 1.6666666666666663
Количество литров дизеля: 0.8333333333333334
Количество литров газа: 0.8333333333333334
```

Рисунок 23. Индивидуальное задание

7. Зафиксированы изменения на репозитории и отправлены на сервер GitHub.

```
[main 9814a3f] Jupiter
1 file changed, 1627 insertions(+)
create mode 100644 "\\320\\233\\320\\240_2_(1).ipynb"
PS C:\Users\USER\DLab-2> git pull
Already up to date.
PS C:\Users\USER\DLab-2> git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 17.83 KiB | 8.91 MiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/miron2314/DLab-2.git
   b5aacd9..9814a3f  main -> main
PS C:\Users\USER\DLab-2> █
```

Рисунок 24. Отправка на сервер GitHub

Ответы на контрольные вопросы:

1. Каково назначение библиотеки NumPy?

Библиотека NumPy (Numerical Python) — это фундаментальная библиотека для научных вычислений в Python. Ее основное назначение заключается в предоставлении мощных инструментов для работы с многомерными массивами и матрицами, а также для выполнения математических операций над этими структурами данных.

2. Что такое массивы ndarray?

ndarray (n-dimensional array) — это основной объект в библиотеке NumPy, представляющий собой многомерный массив однотипных элементов. По сути, это таблица данных, где все элементы имеют один и тот же тип.

3. Как осуществляется доступ к частям многомерного массива?

Доступ к частям многомерного массива осуществляется с помощью индексов, которые указывают на конкретные элементы или срезы массива. В зависимости от языка программирования синтаксис может немного различаться, но общая концепция остается одинаковой.

4. Как осуществляется расчет статистик по данным?

1. Сбор данных Источники данных: Данные могут поступать из различных источников, таких как базы данных, CSV-файлы, API и т.д.

Формат данных: убедитесь, что данные находятся в удобном формате для анализа.

2. Предварительная обработка данных

Очистка данных: Удаление или исправление пропусков, дубликатов и аномалий.

Трансформация данных: Приведение данных к единому формату, например, преобразование типов (числовые, категориальные), нормализация и стандартизация. Фильтрация: Удаление ненужных или нерелевантных данных.

3. Исследовательский анализ данных (EDA) Визуализация: Построение графиков и диаграмм (гистограммы, диаграммы рассеяния, боксплоты) для понимания распределения и взаимосвязей между переменными. Статистические описания: Вычисление основных статистических показателей.

4. Расчет статистик

Центральные тенденции:

Среднее (Mean): Сумма всех значений, деленная на количество значений.

Медиана (Median): Среднее значение, которое делит набор данных на две равные части.

Мода (Mode): наиболее часто встречающееся значение в наборе данных.

5. Как выполняется выборка данных из массивов ndarray? Выборка данных из ndarray выполняется через:

Индексация: Доступ к отдельным элементам по их координатам (например, `arr[0, 1]`).

Срезы (slicing): Извлечение подмассивов по диапазонам индексов (`arr[1:3, :]`). Срезы создают представления (views), а не копии.

Булевая индексация: Выбор элементов на основе логических условий (`arr[arr > 5]`).

Fancy indexing: Выбор элементов с использованием массивов индексов.

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

Основные виды матриц и векторов: Вектор-строка: 1 строка, n столбцов. Вектор-столбец: n строк, 1 столбец.

Квадратная матрица: Количество строк равно количеству столбцов (n x n).

Нулевая матрица: Все элементы равны нулю.

Единичная матрица: Квадратная матрица с единицами на главной диагонали и нулями в остальных местах.

Диагональная матрица: Квадратная матрица, все элементы вне главной диагонали равны нулю.

Треугольная матрица (верхняя/нижняя): Квадратная матрица, у которой все элементы ниже/выше главной диагонали равны нулю.

7. Как выполняется транспонирование матриц? Транспонирование матрицы — это операция, при которой строки и столбцы матрицы меняются местами. Строки становятся столбцами, а столбцы становятся строками. Если исходная матрица имеет размерность $m \times n$, то транспонированная матрица будет иметь размерность $n \times m$. В NumPy для транспонирования матрицы используется метод `T` или функция `np.transpose()`.

8. Приведите свойства операции транспонирования матриц.

1. $(A.T).T == A$ (Транспонирование дважды возвращает исходную матрицу)

2. $(A + B).T == A.T + B.T$ (Транспонирование суммы равно сумме транспонированных)

3. $(c \times A).T == c \times A.T$ (Транспонирование произведения на скаляр равно произведению скаляра на транспонированную)

4. $(A @ B).T == B.T @ A.T$ (Транспонирование произведения равно произведению транспонированных в обратном порядке). `@` - оператор матричного умножения в NumPy (эквивалентно `np.dot(A, B)`).

9. Какие имеются средства в библиотеке NumPy для выполнения транспонирования матриц? Для транспонирования матриц в NumPy существует несколько удобных методов:

1. `numpy.transpose()` — основной метод для транспонирования матрицы. Он меняет строки и столбцы местами

2. Метод `.T` — альтернатива `transpose()`, который делает то же самое, но короче и удобнее в записи.

10. Какие существуют основные действия над матрицами? Основные операции над матрицами в Python с использованием библиотеки NumPy включают:

1. Создание матрицы: Используются функции `np.array()` или `np.matrix()`.

2. Транспонирование: Можно использовать функцию `np.transpose()` или метод `.T`.

3. Умножение матриц: Для умножения используется оператор `@` или функция `np.dot()`.

4. Сложение/вычитание матриц: применяются операторы `+` и `-`.

5. Обращение матрицы: Используется функция `np.linalg.inv()`

6. Определение детерминанта: Функция `np.linalg.det()`

7. Вычисление собственных значений и векторов: С помощью функций `np.linalg.eigvals()` и `np.linalg.eig()`.

8. Нахождение ранга матрицы: Определяется функцией `np.linalg.matrix_rank()`.

11. Как осуществляется умножение матрицы на число? В Python с использованием библиотеки NumPy умножение матрицы на число осуществляется просто с помощью оператора `*`.

12. Какие свойства операции умножения матрицы на число?

- Ассоциативность
- Дистрибутивность относительно сложения матриц
- Дистрибутивность относительно сложения скаляров
- Умножение на единицу

- Умножение на ноль
- Коммутативность

13. Как осуществляется операции сложения и вычитания матриц?

В NumPy сложение и вычитание матриц выполняется поэлементно, с использованием операторов $+$ и $-$. Матрицы должны иметь одинаковый размер

14. Каковы свойства операций сложения и вычитания матриц?

- Коммутативность
- Ассоциативность
- Дистрибутивность
- Существование нулевой матрицы
- Существование противоположной матрицы
- Размерность

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

- Сложение: $+$ или `np.add()`
- Вычитание: $-$ или `np.subtract()`

16. Как осуществляется операция умножения матриц?

- $*$ или `np.multiply()` - поэлементное умножение.
- $@$ или `np.matmul()` - матричное умножение (стандартное умножение матриц).
- `np.dot()` - универсальное умножение (выбирает тип умножения в зависимости от размерности массивов).

17. Каковы свойства операции умножения матриц?

- Некоммутативность
- Ассоциативность
- Дистрибутивность
- Умножение на скаляр
- Умножение на единичную матрицу
- Транспонирование

- Размерности
- Нельзя делить

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

- $C = A @ B$
- `numpy.matmul(A, B)`
- `numpy.dot(A, B)`

19. Что такое определитель матрицы? Каковы свойства определителя матрицы? Определитель — это скалярное значение, которое, может быть, вычислено только для квадратных матриц

Свойства:

- Определитель единичной матрицы
- Умножение на скаляр
- Транспонирование
- Перестановка строк/столбцов
- Умножение матриц
- Нулевая строка/столбец
- Линейная зависимость

20. Какие имеются средства в библиотеке NumPy для нахождения значения определителя матрицы? Для нахождения определителя матрицы в NumPy используется функция `numpy.linalg.det()`

21. Что такое обратная матрица? Какой алгоритм нахождения обратной матрицы? Обратная матрица, обозначаемая A^{-1} , — это такая матрица, которая при умножении на исходную матрицу A дает единичную матрицу. Алгоритм: NumPy использует LU-разложение.

22. Каковы свойства обратной матрицы?

Существование: Обратная матрица существует только для квадратных матриц, которые являются невырожденными (т.е. имеют ненулевой определитель).

Обозначение: Обратная матрица для матрицы A обозначается как A^{-1} .

Не единственность: если матрица A имеет обратную, то её обратная матрица уникальна.

23. Какие имеются средства в библиотеке NumPy для нахождения обратной матрицы? Для нахождения обратной матрицы используется функция `numpy.linalg.inv(A)`

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

```
import numpy as np

def cramer(A, b):
    det_A = np.linalg.det(A)
    if np.isclose(det_A, 0): return None # No solution/infinite solutions
    x = [np.linalg.det(np.column_stack([b if j == i else A[:,j] for j in
range(A.shape[0])])) / det_A for i in range(A.shape[0])]
    return np. array(x)
```

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

```
import numpy as np

def matrix_solve(A, b):
    """Решает Ax = b матричным методом."""
    try:
        A_inv = np.linalg.inv(A)
        x = A_inv @ b
        return x
    except np.linalg.LinAlgError:
        return None # Нет решения (матрица вырождена)
```

Вывод: в ходе лабораторной работы были приобретены навыки работы с базовыми возможностями библиотеки NumPy.