

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

**«Исследование методов работы с матрицами и векторами с помощью
библиотеки NumPy»**

**Отчет по лабораторной работе № 3.3
по дисциплине «Технологии распознавания образов»**

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2023г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

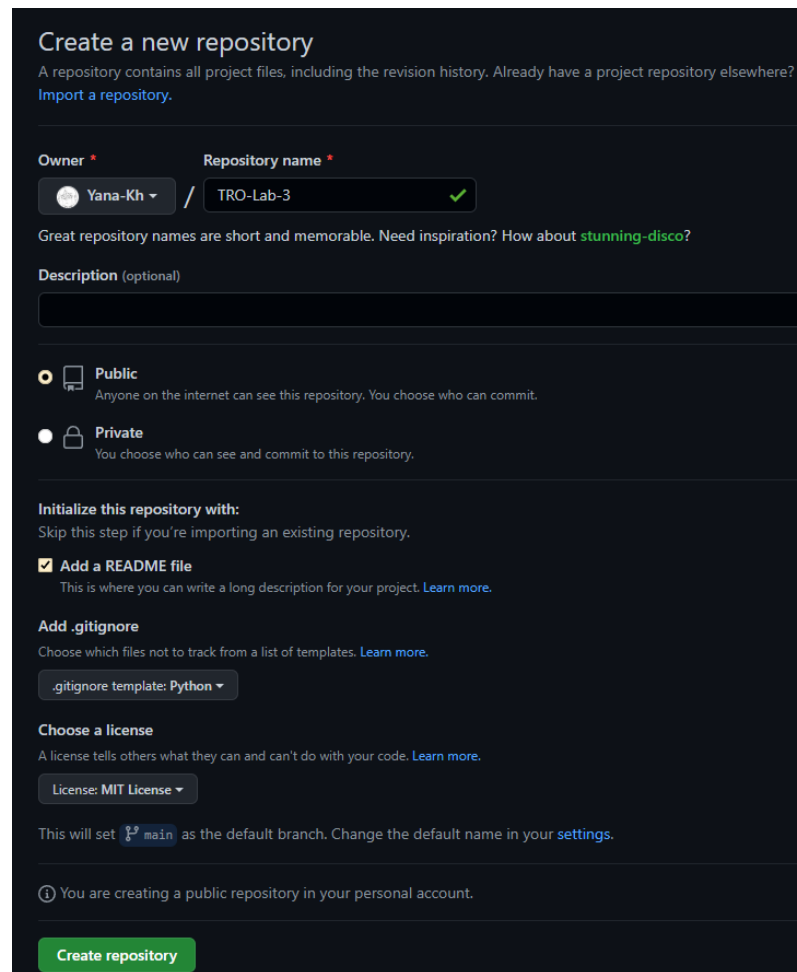
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

Цель работы: исследовать методы работы с матрицами и векторами с помощью библиотеки NumPy языка программирования Python.

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



Create a new repository

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

Owner * Repository name *

Yana-Kh / TRO-Lab-3

Great repository names are short and memorable. Need inspiration? How about [stunning-disco](#)?

Description (optional)

☒ 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:
Skip this step if you're importing an existing repository.

☒ Add a README file
This is where you can write a long description for your project. [Learn more.](#)

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

.gitignore template: Python

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

License: MIT License

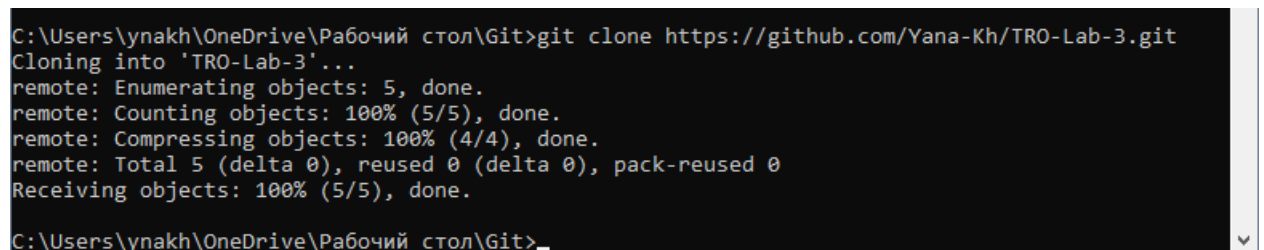
This will set `main` as the default branch. Change the default name in your [settings](#).

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

Create repository

Рисунок 1 – Создание репозитория

3. Выполните клонирование созданного репозитория.



```
C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/Yana-Kh/TRO-Lab-3.git
Cloning into 'TRO-Lab-3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
C:\Users\ynakh\OneDrive\Рабочий стол\Git>
```

Рисунок 2 – Клонирование репозитория

4. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\TRO-Lab-3>git flow init
Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/ynakh/OneDrive/Рабочий стол\Git\TRO-Lab-3/.git/hooks]
C:\Users\ynakh\OneDrive\Рабочий стол\Git\TRO-Lab-3>
```

Рисунок 3 – Организация репозитория в соответствии с моделью git-flow

5. Дополните файл .gitignore необходимыми правилами для выбранного языка программирования, интерактивной оболочки Jupyter notebook и интегрированной среды разработки.

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\TRO-Lab-3>git add .
C:\Users\ynakh\OneDrive\Рабочий стол\Git\TRO-Lab-3>git status
On branch develop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore

C:\Users\ynakh\OneDrive\Рабочий стол\Git\TRO-Lab-3>git commit -m "ignore"
[develop 71c5041] ignore
 1 file changed, 34 insertions(+)
C:\Users\ynakh\OneDrive\Рабочий стол\Git\TRO-Lab-3>
```

Рисунок 4 – Дополнение файла .gitignore

6. Проработать примеры лабораторной работы.

Вектор строка

```
In [4]: v_hor_np = np.array([1, 2])
        print(v_hor_np)

[1 2]

In [7]: v_hor_zeros_v1 = np.zeros((5,))
        print(v_hor_zeros_v1 )
        v_hor_zeros_v2 = np.zeros((1, 5))
        print(v_hor_zeros_v2 )

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

In [8]: v_hor_one_v1 = np.ones((5,))
        print(v_hor_one_v1)
        v_hor_one_v2 = np.ones((1, 5))
        print(v_hor_one_v2)

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

Рисунок 5 – Проработка примеров

Вектор-столбец

```
In [10]: v_vert_np = np.array([[1], [2]])  
print(v_vert_np)
```

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

```
In [22]: v_vert_zeros = np.zeros((5, 1))  
print(v_vert_zeros)  
v_vert_ones = np.ones((5, 1))  
print(v_vert_ones)
```

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

Рисунок 6 – Проработка примеров

Квадратная матрица

```
In [13]: m_sqr_arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(m_sqr_arr)
```

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

```
In [15]: m_sqr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
m_sqr_arr = np.array(m_sqr)  
print(m_sqr_arr)
```

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

```
In [16]: m_sqr_mx = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(m_sqr_mx)
```

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

```
In [28]: #MatLab  
m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')  
print(m_sqr_mx)
```

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

Рисунок 7 – Проработка примеров

Диагональная матрица

```
In [18]: m_diag = [[1, 0, 0], [0, 5, 0], [0, 0, 9]]
m_diag_np = np.matrix(m_diag)
print(m_diag_np)

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

In [21]: m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')
diag = np.diag(m_sqr_mx)
print(diag)
print()

m_diag_np = np.diag(np.diag(m_sqr_mx))
print(m_diag_np)

[1 5 9]

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

Рисунок 8 – Проработка примеров

Единичная матрица

```
In [24]: 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]]

In [26]: m_eye = np.eye(3)
print(m_eye)

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

In [27]: m_idnt = np.identity(3)
print(m_idnt)

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

Рисунок 9 – Проработка примеров

Нулевая матрица

```
In [29]: m_zeros = np.zeros((3, 3))
print(m_zeros)

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

Рисунок 10 – Проработка примеров

Задание матрицы в общем виде

```
In [32]: m_mx = np.matrix('1 2 3; 4 5 6')
print(m_mx)
print()

m_var = np.zeros((2, 5))
print(m_var)
```

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

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

Рисунок 11 – Проработка примеров

Транспонирование матрицы

```
In [33]: A = np.matrix('1 2 3; 4 5 6')
print(A)
print()

A_t = A.transpose()
print(A_t)
```

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

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

Рисунок 12 – Проработка примеров

Умножение матрицы на число

```
In [35]: A = np.matrix('1 2 3; 4 5 6')
C = 3 * A
print(C)
```

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

Рисунок 13 – Проработка примеров

Сложение матриц

```
In [37]: 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 14 – Проработка примеров

Умножение матриц

```
In [38]: 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 15 – Проработка примеров

Определитель матрицы

```
In [40]: 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]]

Out[40]: -14.000000000000009
```

Рисунок 16 – Проработка примеров

Обратная матрица

```
In [41]: 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 17 – Проработка примеров

Ранг матрицы

```
In [42]: 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
```

```
In [43]: m_eye[3][3] = 0
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.  0.]]
3
```

Рисунок 18 – Проработка примеров

7. Создать ноутбук, в котором будут приведены собственные примеры на языке Python для каждого из представленных свойств матричных вычислений.

Свойство 1. Дважды транспонированная матрица равна исходной матрице:

```
In [4]: A = np.matrix('8 9 3 5; 0 9 5 17; 8 4 0 0')
print(A)
print("Транспонированная: ")
print(A.T)
print("Транспонированная второй раз: ")
R = (A.T).T
print(R)

[[ 8  9  3  5]
 [ 0  9  5 17]
 [ 8  4  0  0]]
Транспонированная:
[[ 8  0  8]
 [ 9  9  4]
 [ 3  5  0]
 [ 5 17  0]]
Транспонированная второй раз:
[[ 8  9  3  5]
 [ 0  9  5 17]
 [ 8  4  0  0]]
```

Свойство 2. Транспонирование суммы матриц равно сумме транспонированных матриц:

```
In [8]: A = np.matrix('14 3; 7 8; 0 9')
B = np.matrix('1 9; 10 5; 12 7')
print(f"A:\n{A},\nB:\n{B}")

print("Транспонирование суммы матриц:")
L = (A + B).T
print(L)

print("Сумма транспонированных матриц:")
R = A.T + B.T
print(R)

A:
[[14  3]
 [ 7  8]
 [ 0  9]],
B:
[[ 1  9]
 [10  5]
 [12  7]]
Транспонирование суммы матриц:
[[15 17 12]
 [12 13 16]]
Сумма транспонированных матриц:
[[15 17 12]
 [12 13 16]]
```

Рисунок 19 – Проработка свойств

Свойство 3. Транспонирование произведения матриц равно произведению транспонированных матриц расставленных в обратном порядке

```
In [11]: A = np.matrix('0 2; 1 9; 9 2')
B = np.matrix('23 4 6; 7 1 1')
print(f"A:\n{A},\nB:\n{B}")

print("Транспонирование произведения матриц:")
L = (A.dot(B)).T
print(L)

print("Произведение транспонированных матриц:")
R = (B.T).dot(A.T)
print(R)

A:
[[0 2]
 [1 9]
 [9 2]],
B:
[[23  4  6]
 [ 7  1  1]]
Транспонирование произведения матриц:
[[ 14  86 221]
 [  2  13  38]
 [  2  15  56]]
Произведение транспонированных матриц:
[[ 14  86 221]
 [  2  13  38]
 [  2  15  56]]
```

Рисунок 20 – Проработка свойств

Свойство 4. Транспонирование произведения матрицы на число равно произведению этого числа на транспонированную матрицу

```
In [12]: A = np.matrix('23 4 6; 7 1 1')
k = 3
print(f"A:\n{A}, \nk:{k}")

print("Транспонирование произведения матрицы на число:")
L = (k * A).T
print(L)

print("Произведение числа на транспонированную матрицу:")
R = k * (A.T)
print(R)
```

```
A:
[[23  4  6]
 [ 7  1  1]],
k:3
Транспонирование произведения матрицы на число:
[[69 21]
 [12  3]
 [18  3]]
Произведение числа на транспонированную матрицу:
[[69 21]
 [12  3]
 [18  3]]
```

Свойство 5. Определители исходной и транспонированной матрицы совпадают:

```
In [17]: A = np.matrix('1 0 9; 10 6 5; 12 1 7')
print(f"A:\n{A}")

print("\nОпределитель исходной матрицы:")
A_det = np.linalg.det(A)
print(format(A_det, '.9g'))

print("Определитель транспонированной матрицы:")
A_T_det = np.linalg.det(A.T)
print(format(A_T_det, '.9g'))
```

```
A:
[[ 1  0  9]
 [10  6  5]
 [12  1  7]]
```

Определитель исходной матрицы:

-521

Определитель транспонированной матрицы:

-521

Рисунок 21 – Проработка свойств

Действия над матрицами

Умножение матрицы на число

Свойство 1. Произведение единицы и любой заданной матрицы равно заданной матрице:

```
In [19]: A = np.matrix('1 2; 3 4')
print(f"A:\n{A}")

print("\nПроизведение единицы и матрицы:")
L = 1 * A
print(L)
```

```
A:
[[1 2]
 [3 4]]
```

```
Произведение единицы и матрицы:
[[1 2]
 [3 4]]
```

Свойство 2. Произведение нуля и любой матрицы равно нулевой матрице, размерность которой равна исходной матрице:

```
In [21]: A = np.matrix('14 3; 7 8; 0 9')
Z = np.matrix('0 0; 0 0; 0 0')
L = 0 * A
print("Нулевая матрица:")
print(Z)
print("Произведение матрицы на 0:")
print(L)
```

```
Нулевая матрица:
[[0 0]
 [0 0]
 [0 0]]
```

```
Произведение матрицы на 0:
[[0 0]
 [0 0]
 [0 0]]
```

Свойство 3. Произведение матрицы на сумму чисел равно сумме произведений матрицы на каждое из этих чисел:

```
In [22]: A = np.matrix('23 4 6; 7 1 1')
print(f"A:\n{A}")
print("\nПроизведение матрицы на сумму чисел:")
L = (6 + 2) * A
print(L)
print("Произведений матрицы на каждое из этих чисел:")
R = 6 * A + 2 * A
print(R)
```

```
A:
[[23  4  6]
 [ 7  1  1]]
```

```
Произведение матрицы на сумму чисел:
[[184  32  48]
 [ 56   8   8]]
```

```
Произведений матрицы на каждое из этих чисел:
[[184  32  48]
 [ 56   8   8]]
```

Рисунок 22 – Проработка свойств

Свойство 4. Произведение матрицы на произведение двух чисел равно произведению второго числа и заданной матрицы, умноженному на первое число:

```
In [23]: A = np.matrix('8 1 5; 7 7 1; 0 9 18')
print(f"A:\n{A}")
print("\nПроизведение матрицы на произведение чисел:")
L = (7 * 2) * A
print(L)

print("\nПроизведений второго числа и заданной матрицы, умноженному на первое число:")
R = 7 * (2 * A)
print(R)
```

A:

```
[[ 8  1  5]
 [ 7  7  1]
 [ 0  9 18]]
```

Произведение матрицы на произведение чисел:

```
[[112  14  70]
 [ 98  98  14]
 [  0 126 252]]
```

Произведений второго числа и заданной матрицы, умноженному на первое число:

```
[[112  14  70]
 [ 98  98  14]
 [  0 126 252]]
```

Свойство 5. Произведение суммы матриц на число равно сумме произведений этих матриц на заданное число:

```
In [25]: A = np.matrix('0 9; 3 5')
B = np.matrix('1 8; 11 3')
k = 3
print(f"A:\n{A},\nB:\n{B}")

print("\nПроизведение суммы матриц на число:")
L = k * (A + B)
print(L)

print("\nСумма произведений этих матриц на заданное число:")
R = k * A + k * B
print(R)
```

A:

```
[[0 9]
 [3 5]],
```

B:

```
[[ 1  8]
 [11  3]]
```

Произведение суммы матриц на число:

```
[[ 3 51]
 [42 24]]
```

Сумма произведений этих матриц на заданное число:

```
[[ 3 51]
 [42 24]]
```

Рисунок 23 – Проработка свойств

Сложение матриц

Свойство 1. Коммутативность сложения. От перестановки матриц их сумма не изменяется:

```
In [26]: A = np.matrix('14 3; 7 8; 0 9')
B = np.matrix('1 9; 10 5; 12 7')
print(f"A:\n{A},\nB:\n{B}")

print("\nA + B")
L = A + B
print(L)
print("\nB + A")
R = B + A
print(R)
```

```
A:
[[14  3]
 [ 7  8]
 [ 0  9]],
```

```
B:
[[ 1  9]
 [10  5]
 [12  7]]
```

```
A + B
[[15 12]
 [17 13]
 [12 16]]
```

```
B + A
[[15 12]
 [17 13]
 [12 16]]
```

Рисунок 24 – Проработка свойств

Свойство 2. Ассоциативность сложения. Результат сложения трех и более матриц не зависит от порядка, в котором эта операция будет выполняться:

```
In [30]: A = np.matrix('0 2 1; 9 9 2')
B = np.matrix('23 4 6; 7 1 1')
C = np.matrix('9 1 7; 9 6 3')
print(f"A:\n{A},\nB:\n{B},\nC:\n{C}")

print("\nA + (B + C)")
L = A + (B + C)
print(L)
print("\n(A + B) + C")
R = (A + B) + C
print(R)
```

```
A:
[[0 2 1]
 [9 9 2]],
B:
[[23  4  6]
 [ 7  1  1]],
C:
[[9 1 7]
 [9 6 3]]
```

```
A + (B + C)
[[32  7 14]
 [25 16  6]]
```

```
(A + B) + C
[[32  7 14]
 [25 16  6]]
```

Рисунок 25 – Проработка свойств

Свойство 3. Для любой матрицы существует противоположная ей, такая, что их сумма является нулевой матрицей:

```
In [32]: A = np.matrix('9 4 3; 2 3 3; 9 0 1')
```

```
print(f"A:\n{A}")
print(f"-A:\n{A * (-1)}")
```

```
print(f"Сумма:")
L = A + (-1)*A
print(L)
```

```
A:
[[9 4 3]
 [2 3 3]
 [9 0 1]]
-A:
[[-9 -4 -3]
 [-2 -3 -3]
 [-9  0 -1]]
Сумма:
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

Рисунок 26 – Проработка свойств

Умножение матриц

Свойство 1. Ассоциативность умножения. Результат умножения матриц не зависит от порядка, в котором будет выполняться эта операция.

```
In [35]: A = np.matrix('4 7; 1 2')
B = np.matrix('1 1; 9 3')
C = np.matrix('6 1; 2 0')
print(f"A:\n{A}, \nB:\n{B}, \nC:\n{C}")

print("A*(B*C)")
L = A.dot(B.dot(C))
print(L)
print("(A*B)*C")
R = (A.dot(B)).dot(C)
print(R)
```

```
A:
[[4 7]
 [1 2]],
B:
[[1 1]
 [9 3]],
C:
[[6 1]
 [2 0]]
A*(B*C)
[[452 67]
 [128 19]]
(A*B)*C
[[452 67]
 [128 19]]
```

Свойство 2. Дистрибутивность умножения. Произведение матрицы на сумму матриц равно сумме произведений матриц.

```
In [36]: A = np.matrix('0 2 1; 9 9 2; 5 6 1')
B = np.matrix('23 4 6; 7 1 1; 2 0 8')
C = np.matrix('9 1 7; 9 6 3; 5 6 1')
print(f"A:\n{A}, \nB:\n{B}, \nC:\n{C}")

print("A * (B+C)")
L = A.dot(B + C)
print(L)
print("A*B + A*C")
R = A.dot(B) + A.dot(C)
print(R)
```

```
A:
[[0 2 1]
 [9 9 2]
 [5 6 1]],
B:
[[23 4 6]
 [7 1 1]
 [2 0 8]],
C:
[[9 1 7]
 [9 6 3]
 [5 6 1]]
A * (B+C)
[[ 39 20 17]
 [446 120 171]
 [263 73 98]]
A*B + A*C
[[ 39 20 17]
 [446 120 171]
 [263 73 98]]
```

Рисунок 27 – Проработка свойств

Свойство 3. Умножение матриц в общем виде не коммутативно. Это означает, что для матриц не выполняется правило независимости произведения от перестановки множителей:

```
In [40]: A = np.matrix('0 9; 3 5')
B = np.matrix('1 8; 11 3')
print(f"A:\n{A},\nB:\n{B}\n")

print("A*B")
L = A.dot(B)
print(L)
print("B*A")
R = B.dot(A)
print(R)
```

```
A:
[[0 9]
 [3 5]],
B:
[[ 1  8]
 [11  3]]

A*B
[[99 27]
 [58 39]]
B*A
[[ 24 49]
 [ 9 114]]
```

Свойство 4. Произведение заданной матрицы на единичную равно исходной матрице:

```
In [41]: A = np.matrix('0 2 1; 9 9 2; 5 6 1')
E = np.eye(3)
print(f"A:\n{A}")

print("E*A")
L = E.dot(A)
print(L)
print("A*E")
R = A.dot(E)
print(R)
```

```
A:
[[0 2 1]
 [9 9 2]
 [5 6 1]]
E*A
[[0. 2. 1.]
 [9. 9. 2.]
 [5. 6. 1.]]
A*E
[[0. 2. 1.]
 [9. 9. 2.]
 [5. 6. 1.]]
```

Рисунок 28 – Проработка свойств

Свойство 5. Произведение заданной матрицы на нулевую матрицу равно нулевой матрице:

```
In [42]: A = np.matrix('0 2 1; 9 9 2; 5 6 1')
E = np.zeros((3, 3))
print(f"A:\n{A}")

print("Z*A")
L = Z.dot(A)
print(L)
print("A*Z")
R = A.dot(Z)
print(R)
```

```
A:
[[0 2 1]
 [9 9 2]
 [5 6 1]]
Z*A
[[0 0 0]
 [0 0 0]
 [0 0 0]]
A*Z
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

Рисунок 29 – Проработка свойств

Определитель матрицы

Свойство 1. Определитель матрицы остается неизменным при ее транспонировании:

```
In [47]: A = np.matrix('9 1 7; 9 6 3; 5 6 1')
print(f"A:\n{A}\n")
det_A = round(np.linalg.det(A), 3)
print(f"|A| = {det_A}\n")

print(f"A(T):\n{A.T}\n")
det_A_t = round(np.linalg.det(A.T), 3)
print(f"|A(T)| = {det_A_t}")
```

```
A:
[[9 1 7]
 [9 6 3]
 [5 6 1]]
```

```
|A| = 66.0
```

```
A(T):
[[9 9 5]
 [1 6 6]
 [7 3 1]]
```

```
|A(T)| = 66.0
```

Рисунок 30 – Проработка свойств

Свойство 2. Если у матрицы есть строка или столбец, состоящие из нулей, то определитель такой матрицы равен нулю:

```
In [53]: A = np.matrix('9 1 7; 9 6 3; 5 6 1')
print(f"A:\n{A}\n")
det_A = round(np.linalg.det(A), 3)
print(f"|A| = {det_A}\n")

A = np.matrix('0 1 7; 0 6 3; 0 6 1')
print(f"A:\n{A}\n")

det_A = round(np.linalg.det(A), 3)
print(f"|A| = {det_A}\n")

A = np.matrix('9 1 7; 9 6 3; 0 0 0')
print(f"A:\n{A}\n")
det_A = round(np.linalg.det(A), 3)
print(f"|A| = {det_A}\n")

A:
[[9 1 7]
 [9 6 3]
 [5 6 1]]

|A| = 66.0

A:
[[0 1 7]
 [0 6 3]
 [0 6 1]]

|A| = 0.0

A:
[[9 1 7]
 [9 6 3]
 [0 0 0]]

|A| = 0.0
```

Свойство 3. При перестановке строк матрицы знак ее определителя меняется на противоположный:

```
In [58]: A = np.matrix('9 1 7; 9 6 3; 5 6 1')
print(f"A:\n{A}\n")
det_A = round(np.linalg.det(A), 3)
print(f"|A| = {det_A}\n")

B = np.matrix('9 1 7; 5 6 1; 9 6 3')
print(f"B:\n{B}\n")
det_B = round(np.linalg.det(B), 3)
print(f"|B| = {det_B}\n")

A:
[[9 1 7]
 [9 6 3]
 [5 6 1]]

|A| = 66.0

B:
[[9 1 7]
 [5 6 1]
 [9 6 3]]

|B| = -66.0
```

Рисунок 31 – Проработка свойств

Свойство 4. Если у матрицы есть две одинаковые строки, то ее определитель равен нулю:

```
In [59]: A = np.matrix('9 1 7; 9 6 3; 5 6 1')
print(f"A:\n{A}\n")
det_A = round(np.linalg.det(A), 3)
print(f"|A| = {det_A}\n")

A = np.matrix('9 1 7; 5 6 1; 5 6 1')
print(f"A:\n{A}\n")
det_A = round(np.linalg.det(A), 3)
print(f"|A| = {det_A}\n")
```

```
A:
[[9 1 7]
 [9 6 3]
 [5 6 1]]
```

```
|A| = 66.0
```

```
A:
[[9 1 7]
 [5 6 1]
 [5 6 1]]
```

```
|A| = 0.0
```

Свойство 5. Если все элементы строки или столбца матрицы умножить на какое-то число, то и определитель будет умножен на это число:

```
In [67]: A = np.matrix('9 1 7; 9 6 3; 5 6 1')

k = 3

print(f"A:\n{A}\n")
det_A = round(np.linalg.det(A), 3)
print(f"|A| = {det_A}")
print(f"|A| * k = {det_A * k}\n")

B = A.copy()
B[2, :] = k * B[2, :]
print(f"B:\n{B}\n")

det_B = round(np.linalg.det(B), 3)
print(f"|B| = {det_B}\n")
```

```
A:
[[9 1 7]
 [9 6 3]
 [5 6 1]]
```

```
|A| = 66.0
|A| * k = 198.0
```

```
B:
[[ 9  1  7]
 [ 9  6  3]
 [15 18  3]]
```

```
|B| = 198.0
```

Рисунок 32 – Проработка свойств

Свойство 6. Если все элементы строки или столбца можно представить как сумму двух слагаемых, то определитель такой матрицы равен сумме определителей двух соответствующих матриц:

```
In [71]: A = np.matrix('-4 -1 2; -4 -1 2; 8 3 1')
B = np.matrix('-4 -1 2; 8 3 2; 8 3 1')
C = A.copy()
C[1, :] += B[1, :]
print("C")
print(C)
print("A")
print(A)
print("B")
print(B)
print("|C|")
print(round(np.linalg.det(C), 3))
print("|A| + |B|")
print(round(np.linalg.det(A), 3) + round(np.linalg.det(B), 3))

C
[[-4 -1 2]
 [ 4  2 4]
 [ 8  3 1]]
A
[[-4 -1 2]
 [-4 -1 2]
 [ 8  3 1]]
B
[[-4 -1 2]
 [ 8  3 2]
 [ 8  3 1]]
|C|
4.0
|A| + |B|
4.0
```

Свойство 7. Если к элементам одной строки прибавить элементы другой строки, умноженные на одно и тоже число, то определитель матрицы не изменится:

```
In [74]: A = np.matrix('9 1 7; 9 6 3; 5 6 1')
k = 3

B = A.copy()
B[1, :] = B[1, :] + k * B[0, :]

print("A")
print(A)
print("B")
print(B)

print("\n|A|")
print(round(np.linalg.det(A), 3))
print("|B|")
print(round(np.linalg.det(B), 3))

A
[[ 9  1  7]
 [ 9  6  3]
 [ 5  6  1]]
B
[[ 9  1  7]
 [36  9 24]
 [ 5  6  1]]

|A|
66.0
|B|
66.0
```

Рисунок 33 – Проработка свойств

Свойство 8. Если строка или столбец матрицы является линейной комбинацией других строк (столбцов), то определитель такой матрицы равен нулю

```
In [75]: A = np.matrix('9 1 7; 9 6 3; 5 6 1')
print("A")
print(A)
k = 2
A[1, :] = A[0, :] + k * A[2, :]
print("\n|A|")
print(round(np.linalg.det(A), 3))
```

```
A
[[9 1 7]
 [9 6 3]
 [5 6 1]]

|A|
0.0
```

Свойство 9. Если матрица содержит пропорциональные строки, то ее определитель равен нулю:

```
In [78]: A = np.matrix('9 1 7; 9 6 3; 5 6 1')
print("A")
print(A)
print(round(np.linalg.det(A), 3))

k = 4
A[1, :] = k * A[0, :]
print("\nnew A")
print(A)
print(round(np.linalg.det(A), 3))
```

```
A
[[9 1 7]
 [9 6 3]
 [5 6 1]]
66.0

new A
[[ 9 1 7]
 [36 4 28]
 [ 5 6 1]]
0.0
```

Рисунок 34 – Проработка свойств

Обратная матрица

Свойство 1. Обратная матрица обратной матрицы есть исходная матрица:

```
In [80]: A = np.matrix('4 6; 7 1')
print("A")
print(A)
A_inv = np.linalg.inv(A)
print("A_inv")
print(A_inv)
A_inv_inv = np.linalg.inv(A_inv)
print("A_inv_inv")
print(A_inv_inv)

A
[[4 6]
 [7 1]]
A_inv
[[-0.02631579  0.15789474]
 [ 0.18421053 -0.10526316]]
A_inv_inv
[[4. 6.]
 [7. 1.]]
```

Свойство 2. Обратная матрица транспонированной матрицы равна транспонированной матрице от обратной матрицы:

```
In [81]: A = np.matrix('4 6; 7 1')

print("Обратная матрица транспонированной матрицы:")
L = np.linalg.inv(A.T)
print(L)

print("Транспонированная матрица от обратной матрицы:")
R = (np.linalg.inv(A)).T
print(R)

Обратная матрица транспонированной матрицы:
[[-0.02631579  0.18421053]
 [ 0.15789474 -0.10526316]]
Транспонированная матрица от обратной матрицы:
[[-0.02631579  0.18421053]
 [ 0.15789474 -0.10526316]]
```

Свойство 3. Обратная матрица произведения матриц равна произведению обратных матриц:

```
In [82]: A = np.matrix('4 6; 7 1')
B = np.matrix('1 9; 8 2')

print("Обратная матрица произведения матриц:")
L = np.linalg.inv(A.dot(B))
print(L)

print("Произведение обратных матриц:")
R = np.linalg.inv(B).dot(np.linalg.inv(A))
print(R)

Обратная матрица произведения матриц:
[[ 0.02443609 -0.01804511]
 [-0.0056391  0.01954887]]
Произведение обратных матриц:
[[ 0.02443609 -0.01804511]
 [-0.0056391  0.01954887]]
```

Рисунок 35 – Проработка свойств

8. Создать ноутбук, в котором будут приведены собственные примеры решения систем линейных уравнений матричным методом и методом Крамера.

Матричным методом

```
In [56]: import numpy as np
slau = np.random.randint(0, 15, size = (3, 4))
slau
```

```
Out[56]: array([[ 7, 14,  4,  8],
                [ 1,  0,  5,  6],
                [ 7,  3, 10,  5]])
```

```
In [62]: slau_a = slau[:, 0:3]
slau_b = slau[:, 3]
x = np.ones((3, 1))
```

```
print("slau_a")
print(slau_a)
print("slau_b")
print(slau_b)
```

```
slau_a
[[ 7 14  4]
 [ 1  0  5]
 [ 7  3 10]]
slau_b
[ 8  6  5]
```

```
In [63]: #Найдем определитель
A_det = np.linalg.det(slau_a)

if A_det != 0:
    #Найдем обратную матрицу
    A_inv = np.linalg.inv(slau_a)
    x = A_inv.dot(slau_b)
    print("Решения:")
    print(x)
else:
    print("Матрица вырожденная, нельзя продолжить")
```

```
Решения:
[-2.09338521  1.15564202  1.61867704]
```

Рисунок 36 – Проработка свойств

Метод Крамера

```
In [46]: slau = np.random.randint(0, 15, size = (3, 4))

In [47]: slau_a = slau[:, 0:3]
slau_b = np.ones((3, 1))
x = np.ones((3, 1))
slau_b[0, 0] = slau[0, 3]
slau_b[1, 0] = slau[1, 3]
slau_b[2, 0] = slau[2, 3]

print("slau_a")
print(slau_a)
print("slau_b")
print(slau_b)

slau_a
[[ 2  7  3]
 [13 11  0]
 [ 9 14  2]]
slau_b
[[6.]
 [1.]
 [3.]]

In [52]: #Найдем определитель
A_det = np.linalg.det(slau_a)

if A_det != 0:
    #Найдем дополнительные определители
    for i in range(3):
        A_dop = slau_a.copy()
        A_dop[:, i] = slau_b[:, 0]
        x[i,0] = round(np.linalg.det(A_dop), 3) / round(np.linalg.det(slau_a), 3)
    print("Решения:")
    print(x)
else:
    print("Матрица вырожденная, нельзя продолжить")

Решения:
[[ 0.54954955]
 [-0.55855856]
 [ 2.93693694]]
```

Рисунок 37 – Проработка свойств

9. Зафиксируйте сделанные изменения в репозитории.




| | | |
|---|-------------------|----|
|  | Примеры.ipynb | ex |
|  | ид_СЛАУ.ipynb | id |
|  | ид_свойства.ipynb | id |

Рисунок 38 – Фиксирование изменений в репозитории

Вопросы для защиты работы

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

Вектор строка

```
In [4]: v_hor_np = np.array([1, 2])  
print(v_hor_np)  
  
[1 2]
```

```
In [7]: v_hor_zeros_v1 = np.zeros((5,))  
print(v_hor_zeros_v1 )  
v_hor_zeros_v2 = np.zeros((1, 5))  
print(v_hor_zeros_v2 )  
  
[0. 0. 0. 0. 0.]  
[[0. 0. 0. 0. 0.]]
```

```
In [8]: v_hor_one_v1 = np.ones((5,))  
print(v_hor_one_v1)  
v_hor_one_v2 = np.ones((1, 5))  
print(v_hor_one_v2)  
  
[1. 1. 1. 1. 1.]  
[[1. 1. 1. 1. 1.]]
```

Вектор-столбец

```
In [10]: v_vert_np = np.array([[1], [2]])  
print(v_vert_np)  
  
[[1]  
 [2]]
```

```
In [22]: v_vert_zeros = np.zeros((5, 1))  
print(v_vert_zeros)  
v_vert_ones = np.ones((5, 1))  
print(v_vert_ones)  
  
[[0.]  
 [0.]  
 [0.]  
 [0.]  
 [0.]]  
[[1.]  
 [1.]  
 [1.]  
 [1.]  
 [1.]]
```


Квадратная матрица

```
In [13]: m_sqr_arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(m_sqr_arr)
```

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

```
In [15]: m_sqr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
m_sqr_arr = np.array(m_sqr)  
print(m_sqr_arr)
```

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

```
In [16]: m_sqr_mx = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(m_sqr_mx)
```

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

```
In [28]: #Matlab  
m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')  
print(m_sqr_mx)
```

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

Диагональная матрица

```
In [18]: m_diag = [[1, 0, 0], [0, 5, 0], [0, 0, 9]]  
m_diag_np = np.matrix(m_diag)  
print(m_diag_np)
```

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

```
In [21]: m_sqr_mx = np.matrix('1 2 3; 4 5 6; 7 8 9')  
diag = np.diag(m_sqr_mx)  
print(diag)  
print()  
  
m_diag_np = np.diag(np.diag(m_sqr_mx))  
print(m_diag_np)
```

```
[1 5 9]
```

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

Единичная матрица

```
In [24]: 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]]
```

```
In [26]: m_eye = np.eye(3)  
print(m_eye)
```

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

```
In [27]: m_idnt = np.identity(3)  
print(m_idnt)
```

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

Нулевая матрица

```
In [29]: m_zeros = np.zeros((3, 3))  
print(m_zeros)
```

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

Задание матрицы в общем виде

```
In [32]: m_mx = np.matrix('1 2 3; 4 5 6')  
print(m_mx)  
print()  
  
m_var = np.zeros((2, 5))  
print(m_var)
```

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

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

2. Как выполняется транспонирование матриц?

С помощью `transpose()`

```
A = np.matrix('1 2 3; 4 5 6')
```

```
A_t = A.transpose()
```

3. Приведите свойства операции транспонирования матриц.

Свойство 1. Дважды транспонированная матрица равна исходной матрице.

Свойство 2. Транспонирование суммы матриц равно сумме транспонированных матриц.

Свойство 3. Транспонирование произведения матриц равно произведению транспонированных матриц, расставленных в обратном порядке.

Свойство 4. Транспонирование произведения матрицы на число равно произведению этого числа на транспонированную матрицу.

Свойство 5. Определители исходной и транспонированной матрицы совпадают.

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

В библиотеки NumPy для транспонирования двумерных матриц используется метод `transpose()`.

5. Какие существуют основные действия над матрицами?

- Сложение матриц
- Умножение матрицы на число
- Умножение матриц

6. Как осуществляется умножение матрицы на число?

Умножение матрицы на число

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

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

7. Какие свойства операции умножения матрицы на число?

Свойство 1. Произведение единицы и любой заданной матрицы равно заданной матрице.

Свойство 2. Произведение нуля и любой матрицы равно нулевой матрице, размерность которой равна исходной матрицы.

Свойство 3. Произведение матрицы на сумму чисел равно сумме произведений матрицы на каждое из этих чисел.

Свойство 4. Произведение матрицы на произведение двух чисел равно произведению второго числа и заданной матрицы, умноженному на первое число.

Свойство 5. Произведение суммы матриц на число равно сумме произведений этих матриц на заданное число.

8. Как осуществляется операции сложения и вычитания матриц?

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]]
```

Аналогично вычитание

9. Каковы свойства операций сложения и вычитания матриц?

Свойство 1. Коммутативность сложения. От перестановки матриц их сумма не изменяется.

Свойство 2. Ассоциативность сложения. Результат сложения трех и более матриц не зависит от порядка, в котором эта операция будет выполняться.

Свойство 3. Для любой матрицы существует противоположная ей, такая, что их сумма является нулевой матрицей

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

Для сложения и вычитания используется + и -.

11. Как осуществляется операция умножения матриц?

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]]
```

12. Каковы свойства операции умножения матриц?

Свойство 1. Ассоциативность умножения. Результат умножения матриц не зависит от порядка, в котором будет выполняться эта операция.

Свойство 2. Дистрибутивность умножения. Произведение матрицы на сумму матриц равно сумме произведений матриц.

Свойство 3. Умножение матриц в общем виде не коммутативно. Это означает, что для матриц не выполняется правило независимости произведения от перестановки множителей.

Свойство 4. Произведение заданной матрицы на единичную равно исходной матрице.

Свойство 5. Произведение заданной матрицы на нулевую матрицу равно нулевой матрице.

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

В библиотеки NumPy для выполнения операции умножения матриц используется функция `dot()`.

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

Определитель матрицы размера (n-го порядка) является одной из ее численных характеристик. Определитель матрицы A обозначается как $|A|$ или $\det(A)$, его также называют детерминантом.

Свойство 1. Определитель матрицы остается неизменным при ее транспонировании.

Свойство 2. Если у матрицы есть строка или столбец, состоящие из нулей, то определитель такой матрицы равен нулю.

Свойство 3. При перестановке строк матрицы знак ее определителя меняется на противоположный.

Свойство 4. Если у матрицы есть две одинаковые строки, то ее определитель равен нулю.

Свойство 5. Если все элементы строки или столбца матрицы умножить на какое-то число, то и определитель будет умножен на это число.

Свойство 6. Если все элементы строки или столбца можно представить как сумму двух слагаемых, то определитель такой матрицы равен сумме определителей двух соответствующих матриц.

Свойство 7. Если к элементам одной строки прибавить элементы другой строки, умноженные на одно и тоже число, то определитель матрицы не изменится.

Свойство 8. Если строка или столбец матрицы является линейной комбинацией других строк (столбцов), то определитель такой матрицы равен нулю

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

В библиотеке NumPy для нахождения значения определителя матрицы используется функция `linalg.det()`.

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

Обратной матрицей матрицы называют матрицу, удовлетворяющую следующему равенству: $A \cdot A^{-1} = E$

Обратную матрицу A^{-1} к матрице A можно найти по формуле:

$$A^{-1} = 1/\det A \cdot A^*,$$

где $\det A$ — определитель матрицы A ,

A^* — транспонированная матрица алгебраических дополнений к матрице A .

17. Каковы свойства обратной матрицы?

Свойство 1. Обратная матрица обратной матрицы есть исходная матрица.

Свойство 2. Обратная матрица транспонированной матрицы равна транспонированной матрице от обратной матрицы.

Свойство 3. Обратная матрица произведения матриц равна произведению обратных матриц.

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

В библиотеки NumPy для нахождения обратной матрицы используется функция `linalg.inv()`.

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

Матричным методом

```
In [56]: import numpy as np
slau = np.random.randint(0, 15, size = (3, 4))
slau
```

```
Out[56]: array([[ 7, 14,  4,  8],
                [ 1,  0,  5,  6],
                [ 7,  3, 10,  5]])
```

```
In [62]: slau_a = slau[:, 0:3]
slau_b = slau[:, 3]
x = np.ones((3, 1))
```

```
print("slau_a")
print(slau_a)
print("slau_b")
print(slau_b)
```

```
slau_a
[[ 7 14  4]
 [ 1  0  5]
 [ 7  3 10]]
slau_b
[8 6 5]
```

```
In [63]: #Найдем определитель
A_det = np.linalg.det(slau_a)

if A_det != 0:
    #Найдем обратную матрицу
    A_inv = np.linalg.inv(slau_a)
    x = A_inv.dot(slau_b)
    print("Решения:")
    print(x)
else:
    print("Матрица вырожденная, нельзя продолжить")
```

```
Решения:
[-2.09338521  1.15564202  1.61867704]
```


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

Метод Крамера

```
In [46]: slau = np.random.randint(0, 15, size = (3, 4))
```

```
In [60]: slau_a = slau[:, 0:3]
slau_b = np.ones((3, 1))
x = np.ones((3, 1))
slau_b[0, 0] = slau[0, 3]
slau_b[1, 0] = slau[1, 3]
slau_b[2, 0] = slau[2, 3]

print("slau_a")
print(slau_a)
print("slau_b")
print(slau_b)
```

```
slau_a
[[ 7 14  4]
 [ 1  0  5]
 [ 7  3 10]]
slau_b
[[8.]
 [6.]
 [5.]]
```

```
In [61]: #Найдем определитель
A_det = np.linalg.det(slau_a)

if A_det != 0:
    #Найдем дополнительные определители
    for i in range(3):
        A_dop = slau_a.copy()
        A_dop[:, i] = slau_b[:, 0]
        x[i,0] = round(np.linalg.det(A_dop), 3) / round(np.linalg.det(slau_a), 3)
    print("Решения:")
    print(x)
else:
    print("Матрица вырожденная, нельзя продолжить")
```

```
Решения:
[[-2.09338521]
 [ 1.15564202]
 [ 1.61867704]]
```