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 «РаботасJupyterNotebook, JupyterLab, GoogleColab» по дисциплине «Искусственный интеллект и машинное обучение»

Выполнил:
Гончаров Серафим Ростиславович 2 курс, группа ИВТ-б-о-23-1,09.03.01 «Информатика и вычислительная техника», направленность (профиль) «Автоматизированные системы обработки информации и управления», очная форма обучения

Руководитель практики:
Воронкин Роман Александрович

Ставрополь 2025

Тема: Основы работы с библиотекой NumPy.

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

Порядок выполнения лабораторной работы:

Задание 1. Создание и изменение массивов.

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

[5]:
```
import numpy as np
arr = np.matrix('1 2 3 ; 4 5 6 ; 7 8 9')
result = arr * 2 # Быстрое поэлементное умножение
result
```

[5]:
```
matrix([[ 2,  4,  6],
        [ 8, 10, 12],
        [14, 16, 18]])
```

[11]:
```
result[result > 10] = 0
result
```

[11]:
```
matrix([[ 2,  4,  6],
        [ 8, 10,  0],
        [ 0,  0,  0]])
```

Рисунок 1 – Изменение массива

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

[73]:
```
import random
array = [random.randint(1, 100) for _ in range(20)]
print(f"Массив случайных чисел: {array}")
```

```
Массив случайных чисел: [39, 12, 8, 27, 25, 96, 27, 96, 50, 50, 29, 17, 33, 27, 38, 82, 24, 52, 50, 82]
```

[75]:
```
print(f"Элементы, которые делятся на 5 без остатка: ")
i = 0
for i in range(20):
    if array[i] % 5 == 0:
        five = array[i]
        print(five)
        array[i] = -1
print(f"Обновленный массив: {array}")
```

```
Элементы, которые делятся на 5 без остатка:
25
50
50
50
Обновленный массив: [39, 12, 8, 27, -1, 96, 27, 96, -1, -1, 29, 17, 33, 27, 38, 82, 24, 52, -1, 82]
```

Рисунок 2 – Булевые маски

Задание 3. Объединение разбиение массивов.

Создайте два массива NumPy размером 1×5, заполненные случайными числамиот 0 до 50.

Объедините эти массивы в один двумерный массив (по строкам).

Разделите полученный массив на два массива, каждый из которых содержит 5 элементов.

[169]:
```
import random
array1 = [random.randint(0, 50) for _ in range(5)]
print(f"Первый массив случайных чисел: {array1}")
array2 = [random.randint(0, 50) for _ in range(5)]
print(f"Второй массив случайных чисел: {array2}")
```

```
Первый массив случайных чисел: [7, 6, 25, 12, 44]
Второй массив случайных чисел: [7, 28, 28, 11, 11]
```

[173]:
```
soed = np.vstack((array1, array2))
soed
```

[173]:
```
array([[ 7,  6, 25, 12, 44],
       [ 7, 28, 28, 11, 11]])
```

[175]:
```
raz1, raz2 = np.vsplit(soed, 2)
print(f"Первый разделенный массив: {raz1}")
print(f"Второй разделенный массив: {raz2}")
```

```
Первый разделенный массив: [[ 7  6 25 12 44]]
Второй разделенный массив: [[ 7 28 28 11 11]]
```

Рисунок 3 – Объединение разбиение массивов

Задание 4. Генерация 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.

[187]:
```
array = np.sort(np.random.randint(-10, 10, 50))
array
```

[187]:
```
array([-10, -10, -10, -10,  -9,  -9,  -9,  -7,  -7,  -7,  -7,  -6,  -5,
        -4,  -4,  -3,  -3,  -3,  -2,  -2,  -2,  -1,  -1,   0,   0,   0,
         1,   1,   2,   3,   3,   3,   3,   3,   5,   5,   5,   5,   5,
         6,   6,   6,   7,   7,   8,   8,   8,   9,   9,   9])
```

[189]:
```
sum_all = np.sum(array)
sum_pol = np.sum(array[array > 0])
sum_otr = np.sum(array[array < 0])

print(f"Сумма всех элементов: {sum_all}")
print(f"Сумма положительных элементов: {sum_pol}")
print(f"Сумма отрицательных элементов: {sum_otr}")
```

```
Сумма всех элементов: -4
Сумма положительных элементов: 127
Сумма отрицательных элементов: -131
```

Рисунок 4 – Генерация работа с линейными последовательностями

Задание 5. Работа с диагональными и единичными матрицами.

Создайте:
Единичную матрицу 4x4
Диагональную матрицу размером 4x4 с диагональными элементами [5, 10, 15, 20] (не использовать циклы)

Найдите сумму всех элементов каждой из этих матриц и сравните результаты.

[199]:
```
array_one = np.eye(4)
array_diag = np.diag([5, 10, 15, 20])
sum_arr1 = np.sum(array_one)
sum_arr2 = np.sum(array_diag)
print(f"Единичная матрица:\n", array_one)
print(f"Сумма всех элементов: {sum_arr1}")
print(f"Диагональная матрица:\n", array_diag)
print(f"Сумма всех элементов: {sum_arr2}")
```

```
Единичная матрица:
 [[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
Сумма всех элементов: 4.0
Диагональная матрица:
 [[ 5  0  0  0]
 [ 0 10  0  0]
 [ 0  0 15  0]
 [ 0  0  0 20]]
Сумма всех элементов: 50
```

Рисунок 5 – Работа с диагональными и единичными матрицами

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:

Их сумму
Их разность
Их поэлементное произведение

[215]:
```
mat1 = np.matrix(np.random.randint(1, 20, (3, 3)))
mat2 = np.matrix(np.random.randint(1, 20, (3, 3)))
sum_mat = mat1 + mat2
raz_mat = mat1 - mat2
proiz_mat = np.multiply(mat1, mat2)
print(f"Первая матрица:\n", mat1)
print(f"Вторая матрица:\n", mat2)
print(f"Сумма матриц:\n", sum_mat)
print(f"Разность матриц:\n", raz_mat)
print(f"Их поэлементное произведение матриц:\n", proiz_mat)
```

```
Первая матрица:
 [[ 6  4 16]
 [ 6  9  1]
 [ 9  6  3]]
Вторая матрица:
 [[15  7 12]
 [ 9 15 16]
 [ 5  7  4]]


Сумма матриц:
 [[21 11 28]
 [15 24 17]
 [14 13  7]]
Разность матриц:
 [[ -9  -3   4]
 [ -3  -6 -15]
 [  4  -1  -1]]
Их поэлементное произведение матриц:
 [[ 90  28 192]
 [ 54 135  16]
 [ 45  42  12]]
```

Рисунок 6 – Создание и базовые операции с матрицами

Задание 7. Умножение матриц.

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.

[230]:
```
mat1 = np.matrix(np.random.randint(1, 10, (2, 3)))
mat2 = np.matrix(np.random.randint(1, 10, (3, 2)))
proiz_mat = mat1 @ mat2
print(f"Первая матрица:\n", mat1)
print(f"Вторая матрица:\n", mat2)
print(f"Произведение матриц:\n", proiz_mat)
```

```
Первая матрица:
 [[2 2 6]
 [7 8 4]]
Вторая матрица:
 [[2 5]
 [5 5]
 [4 9]]
Произведение матриц:
 [[ 38  74]
 [ 70 111]]
```

Рисунок 7 – Умножение матриц

Задание 8. Определитель и обратная матрица.

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 .

[162]:
```
mat = np.matrix(np.random.randint(1, 100, (3, 3)))
opr = np.linalg.det(mat)
print("Первоначальная матрица: \n", mat)
if opr == 0:
    print("Матрица вырождена!")
else:
    print("Обратная матрица: \n", np.linalg.inv(mat))
    print("ее определитль равен: \n", opr)
```

```
Первоначальная матрица:
 [[91 76 24]
 [ 7 51 68]
 [ 8 42 47]]
Обратная матрица:
 [[ 0.01629682  0.09103497 -0.14003195]
 [-0.00763359 -0.14503817  0.21374046]
 [ 0.00404758  0.11411326 -0.14589029]]
ее определитль равен:
 -28165.000000000004
```

Рисунок 8 – Определитель и обратная матрица

Задание 9. Транспонирование и след матрицы.

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 нахождния следа

```
169]: mat = np.matrix(np.random.randint(1, 50, (4, 4)))
tran = mat.T
trac = np.trace(mat)
print("Первоначальная матрица: \n", mat)
print("Транспонированную матрица: \n", tran)
print("След матрицы: \n", trac)
```

```
Первоначальная матрица:
 [[37 39 25  1]
 [26 22 40 40]
 [31 29 37 32]
 [45 24  2 34]]
Транспонированную матрица:
 [[37 26 31 45]
 [39 22 29 24]
 [25 40 37  2]
 [ 1 40 32 34]]
След матрицы:
 130
```

Рисунок 9 – Транспонирование и след матрицы

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.

```
98]: 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: \n", x)
```

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

Рисунок 10 – Системы линейных уравнений

Задание 11. Индивидуальное задание.

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

Финансовая стратегия инвестора

Инвестор вложил деньги в три компании. В первую компанию он вложил в два раза больше, чем во вторую, а в третью — на 10 000 рублей больше, чем во вторую. Общая сумма инвестиций составила 300 000 рублей. Сколько денег вложено в каждую компанию?

x - первая комания, y - вторая компания, z - третья компания

x = 2y
z = y + 10000
x + y + z = 300000

$$\begin{cases} -x + 2y = 0 \\ y - z + 10000 = 0 \\ x + y + z = 300000 \end{cases}$$

[22]:
```
arr1 = np.array([
    [-1, 2, 0],
    [0, 1, -1],
    [1, 1, 1]
])
arr2 = np.array([0, -10000, 300000])
detA1 = np.linalg.det(arr1)
x = arr1.copy()
x[:, 0] = arr2
```

```
detX = np.linalg.det(x)
y = arr1.copy()
y[:, 1] = arr2
detY = np.linalg.det(y)
z = arr1.copy()
z[:, 2] = arr2
detZ = np.linalg.det(z)
q = detX / detA1
w = detY / detA1
e = detZ / detA1
print("Решение методом Крамера:\n")
print("Первая компания = ", q)
print("Вторая компания = ", w)
print("Третья компания = ", e)
```

```
Решение методом Крамера:

Первая компания =  144999.99999999997
Вторая компания =  72499.99999999997
Третья компания =  82499.99999999988
```

[224]:
```
matr = np.linalg.inv(arr1) @ arr2
print("Решение матричным методом:\n")
print("Первая компания = ", matr[0])
print("Вторая компания = ", matr[1])
print("Третья компания = ", matr[2])
```

```
Решение матричным методом:

Первая компания =  145000.0
Вторая компания =  72500.0
Третья компания =  82500.0
```

[226]:
```
lin = np.linalg.solve(arr1, arr2)
print("Решение с помощью np.linalg.solve:\n")
print("Первая компания = ", lin[0])
print("Вторая компания = ", lin[1])
print("Третья компания = ", lin[2])
```

```
Решение с помощью np.linalg.solve:

Первая компания =  145000.0
Вторая компания =  72500.0
Третья компания =  82500.0
```

Рисунок 11 – Индивидуальное задание

https://github.com/GoncharovSerafim/Hesko

Ответы на контрольные вопросы:

1. **Каково назначение библиотеки NumPy?**

NumPy (от "Numerical Python") — это открытая библиотека для языка Python, предназначенная для эффективных вычислений с многомерными массивами и матрицами.

2. **Что такое массивы ndarray?**

Основная структура данных в NumPy — это многомерный массив, называемый ndarray. Это объект, который представляет собой таблицу данных (например, вектор или матрицу) и позволяет выполнять операции с ними с высокой эффективностью.

3. **Как осуществляется доступ к частям многомерного массива?**

В NumPy доступ к частям многомерного массива осуществляется с помощью индексации и срезов. Базовая индексация:

Поэлементный доступ: arr[0, 0].

Срезы (Slicing): arr[0, :].

4. **Как осуществляется расчет статистик по данным?**

В библиотеке NumPy расчет статистик по данным осуществляется с помощью встроенных функций, которые оптимизированы для работы с массивами NumPy. Эти функции позволяют быстро и эффективно вычислять различные статистические показатели.

5. **Как выполняется выборка данных из массивов ndarray?**

Выборка данных из массивов NumPy ndarray выполняется с помощью различных методов, которые позволяют извлекать нужные подмножества данных. К основным методам относятся:

ИндексацияиСрезы (IndexingandSlicing);

Булевая индексация (Boolean Indexing);

Целочисленная индексация;

Случайная выборка (Random Sampling).

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

Основные виды матриц и векторов:

– Вектор-строка: Матрица размерности 1 x n (одна строка, n столбцов). vector_row = np.array([1, 2, 3])

– Вектор-столбец: Матрица размерности n x 1 (n строк, один столбец). vector_column = np.array([[1], [2], [3]])

– Квадратная матрица: Матрица, у которой число строк равно числу столбцов (n x n). square_matrix = np.array([[1, 2], [3, 4]])

– Нулевая матрица: Матрица, состоящая только из нулей. zero_matrix = np.zeros((3, 4))

– Матрица из единиц: Матрица, все элементы которой равны 1. ones_matrix = np.ones((2, 3))

– Диагональная матрица: Квадратная матрица, у которой все элементы вне главной диагонали равны нулю. diagonal_matrix = np.diag([1, 2, 3])

– Единичная матрица: Диагональная матрица, у которой все элементы на главной диагонали равны 1. identity_matrix = np.identity(3)

7. Как выполняется транспонирование матриц?

Транспонирование матрицы в NumPy – это операция, при которой строки и столбцы матрицы меняются местами. Это означает, что элемент, находящийся в строке *i* и столбце *j*, после транспонирования окажется в строке *j* и столбце *i*. NumPy предоставляет несколько способов для выполнения этой операции, каждый из которых имеет свои особенности и сценарии применения.

8. 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.

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

Метод `.T` (Transposition Attribute): transposed_matrix = matrix.T.

Функция `np.transpose()`: transposed_matrix = np.transpose(matrix)

9. Какие существуют основные действия над матрицами?

1. Сложение и вычитание матриц;

2. Умножения на скаляр;

3. Матричное умножение;

4. Транспонирование;

5. Определитель.

10.Как осуществляется умножение матрицы на число?

Умножение матрицы на число (скаляр) – это простая операция, при которой каждый элемент матрицы умножается на это число. Пример в NumPy:

```
A = np.array([[1, 2, 3], [4, 5, 6]])
k = 2
C = k * A
```

11.Какие свойства операции умножения матрицы на число?

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. Как осуществляется операции сложения и вычитания матриц?

Операции сложения и вычитания матриц — это одни из самых базовых операций в линейной алгебре. Они достаточно просты, но требуют соблюдения одного *очень важного* условия: **матрицы должны иметь одинаковую размерность.** Это означает, что число строк и число столбцов обеих матриц должны быть в точности одинаковыми.

14. 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.

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

В библиотеке NumPy есть несколько способов для выполнения операций сложения и вычитания матриц, хотя самый распространенный и рекомендуемый – это простое использование операторов + и −.

16. Как осуществляется операция умножения матриц?

Чтобы матрицу A можно было умножить на матрицу B, необходимо, чтобы количество столбцов в матрице A было равно количеству строк в матрице B.

Если A имеет размерность m x n (m строк и n столбцов), а B имеет размерность n x p (n строк и p столбцов), то их произведение C = A * B будет иметь размерность m x p (m строк и p столбцов).

17. Каковы свойства операции умножения матриц?

1. Некоммутативность:A* B ≠ B * A;

2. Ассоциативность: (A * B) * C = A * (B * C);

3. Дистрибутивность относительно сложения: A * (B + C) = A * B + A * C;

4. Умножение на единичную матрицу: A * I = A, I – единичная матрица.

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

1. Оператор `@`:

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = A @ B
```

2. Метод `.dot()`:

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
C = A.dot(B)
```

19. 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. 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.

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

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

```
A = np.array([[1, 2],
              [3, 4]])
determinant = linalg.det(A)
```

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

Обратная матрица (обозначается как A^{-1}) — это такая матрица, которая, будучи умноженной на исходную матрицу A, дает в результате единичную матрицу (I). То есть:

$A * A^{-1} = A^{-1} * A = I$

Где I — единичная матрица (матрица с единицами на главной диагонали и нулями во всех остальных местах).

Существует несколько алгоритмов для нахождения обратной матрицы. Метод Гаусса-Жордана:

1. Создание расширенной матрицы: Объедините исходную матрицу A с единичной матрицей I того же размера, создав расширенную матрицу [A | I].

2. Приведение к ступенчатому виду (прямой ход Гаусса): Используйте элементарные преобразования строк для приведения левой части расширенной матрицы к верхнему треугольному виду. Элементарные преобразования строк включают:

– Перестановку двух строк.

– Умножение строки на ненулевой скаляр.

– Добавление к одной строке другой строки, умноженной на скаляр.

3. Приведение к диагональному виду (обратный ход Гаусса-Жордана): Продолжайте выполнять ЭПС, чтобы превратить верхнюю треугольную матрицу в единичную матрицу. Важно, чтобы эти же преобразования применялись к правой части расширенной матрицы (которая изначально была единичной матрицей I).

4. Получение обратной матрицы: После того как левая часть расширенной матрицы станет единичной матрицей, правая часть будет представлять собой обратную матрицу A^{-1}. То есть, мы получили матрицу [I | A^{-1}].

22. Каковы свойства обратной матрицы?

1. Обратная от обратной: $(A^{-1})^{-1} = A$. Если взять обратную от обратной матрицы, то получится исходная матрица.

2. Обратная от произведения: $(A * B)^{-1} = B^{-1} * A^{-1}$. Обратная матрица от произведения двух матриц равна произведению обратных матриц, взятых в обратном порядке.

3. Обратная от транспонированной матрицы: $(A^T)^{-1} = (A^{-1})^T$. Обратная от транспонированной матрицы равна транспонированной матрице от обратной.

4. Обратная от скалярного произведения: If A is invertible: $(kA)^{-1} = (1/k) A^{-1}$, где k — скаляр, отличный от нуля.

5. Определитель обратной матрицы: $\det(A^{-1}) = 1 / \det(A)$

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

В библиотеке NumPy для нахождения обратной матрицы используется функция `numpy.linalg.inv()`. Эта функция, как

и `numpy.linalg.det()`, является частью модуля `linalg` (linear algebra) библиотеки NumPy.

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

Метод Крамера (Cramer's Rule) — это аналитический метод решения систем линейных алгебраических уравнений (СЛАУ) с квадратной матрицей (то есть, когда число уравнений равно числу неизвестных). Метод Крамера использует определители матриц для нахождения решения.

Алгоритм решения методом Крамера:

1. Вычислить определитель основной матрицы A (det(A)).
2. Если det(A) = 0, то система либо не имеет решений, либо имеет бесконечно много решений. Метод Крамера здесь не применим.
3. Поочередно для каждой неизвестной x_i (i = 1, 2, …, n):
   - Заменить i-й столбец матрицы A на вектор правых частей b, получив новую матрицу A_i.
   - Вычислить определитель матрицы A_i (det(A_i)).
   - Найти значение x_i по формуле x_i = det(A_i) / det(A).

```
A=np.array([
[1,1,1,],
[-1.5,1,0],
[0,-1.2,1]
])
B=np.array([5000000,0,0])
det_A=np.linalg.det(A)
A_x=A.copy()
A_x[:,0]=B
det_A_x=np.linalg.det(A_x)
A_y=A.copy()
A_y[:,1]=B
```

```
det_A_y=np.linalg.det(A_y)
A_z=A.copy()
A_z[:,2]=B
det_A_z=np.linalg.det(A_z)
x=det_A_x/det_A
y=det_A_y/det_A
z=det_A_z/det_A
print("Методкрамера:\n")
print(f"x = {x:.2f} рублей")
print(f"y = {y:.2f} рублей")
print(f"z = {z:.2f} рублей")
```

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

Матричный метод (также известный как метод обратной матрицы) — это способ решения систем линейных алгебраических уравнений (СЛАУ) с использованием обратной матрицы. Этот метод применим, когда количество уравнений равно количеству неизвестных, и определитель матрицы коэффициентов не равен нулю (то есть, матрица является обратимой).

Алгоритм решения матричным методом:

1. Записать систему уравнений в матричной форме: определить матрицу коэффициентов A и вектор правых частей b.
2. Вычислить определитель матрицы A (det(A)): Если det(A) = 0, то матрица A не имеет обратной, и матричный метод не применим. Система либо не имеет решений, либо имеет бесконечно много решений.
3. Найти обратную матрицу A^{-1}: Использовать один из алгоритмов нахождения обратной матрицы (например, метод Гаусса-Жордана или встроенные функции в библиотеках).

4. Вычислить решение: Умножить обратную матрицу A^{-1} на вектор правых частей b: $x = A^{-1}b$.

```
A=np.array([
[1,1,1,],
[-1.5,1,0],
[0,-1.2,1]
])
B=np.array([5000000,0,0])
results=np.linalg.inv(A)@B
print("Решение матричным методом:\n")
print(f"x: {results[0]:.2f}рублей")
print(f"y: {results[1]:.2f}рублей")
print(f"z: {results[2]:.2f}рублей")
```

Вывод: в ходе лабораторной работы были исследованы базовые возможности библиотеки NumPy языка программирования Python.