Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего

образования «Южно-Уральский государственный университет (национальный

исследовательский университет)»

Высшая школа экономики и управления

Кафедра Информационных технологий в экономике

Программирование на языке Python (курс молодого бойца)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по дисциплине «ПРОГРАММИРОВАНИЕ»

ЮУрГУ – 380305.2022.362. ПЗ КР

Рецензент, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Руководитель, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019г. «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019г.

Нормоконтролёр, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Автор, студент группы ЭУ-142

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Абдрахманов Р.В./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019г.

Работа защищена с оценкой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019г.

Челябинск 2019

**АННОТАЦИЯ**

Абдрахманов Р.В.

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-142, 2019

В курсовой работе мы разобрали основные возможности языка программирования Python, подключили библиотеку numpy и решили 31 задачу. Также разобрали алгоритмы сортировки и решение СЛАУ методом Гаусса.

**Оглавление**

[**Введение** 3](#_Toc10386715)

[**ГЛАВА 1. Методы сортирвки** 4](#_Toc10386716)

[**Bubble Sort** 4](#_Toc10386717)

[**Insert Sort** 4](#_Toc10386718)

[**Select Sort** 4](#_Toc10386719)

[**Анализ методов сортировок:** 5](#_Toc10386720)

[**Глава 2. Библиотека Numpy** 9](#_Toc10386721)

[**Что такое Numpy?** 9](#_Toc10386722)

[**Numpy на практике** 9](#_Toc10386723)

[**Глава 3. Решаем СЛАУ на Python** 32](#_Toc10386724)

[**Заключение** 34](#_Toc10386725)

[**Библиографический список** 35](#_Toc10386726)

# **Введение**

Для начала хочется сказать, что же такое программирование. Программирование — процесс создания компьютерных программ.По выражению одного из основателя языков программирования Никлауса Вирта «Программы = алгоритмы + структуры данных».

Программирование не может обойтись без языков программирования. Язык программирования — формальный язык, предназначенный для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель (обычно — ЭВМ) под её управлением. Существует большое различие языков, но в рамках нашей курсовой работы мы будем говорить о Python.

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Наша работа состоит из 3 глав:

1. В первой главе поднята тема алгоритмов сортировки для одномерных массивов.
2. Во второй главе поднята тема библиотеки Numpy, здесь же мы применяли основные её возможности на практике.
3. В третей части поднята тема линейной алгебры, мы решили СЛАУ методом Гаусса при помощи Python и Numpy.

Цель очень проста: выяснить самый эффективный алгоритм сортировки из трех представленных (Сортировка включением, Обменная сортировка, Сортировка выбором); набить руку и разобраться в Numpy, решая задачи; решить СЛАУ при помощи программирования.

# **ГЛАВА 1. Методы сортирвки**

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

## **Bubble Sort**

Bubble Sort — это самый простой алгоритм сортировки, который работает путем многократного обмена смежными элементами, если они находятся в неправильном порядке.

Пример алгоритма:

n = 1

while n < dim:

for i in range(dim - n):

if arr[i] > arr[i + 1]:

arr[i], arr[i + 1] = arr[i + 1], arr[i]

n += 1

## 

## **Insert Sort**

Insert Sort — это простой алгоритм сортировки, который работает так, как мы сортируем игральные карты в наших руках.

Пример алгоритма:

for i in range(1, dim):

temp = arr[i]

j = i - 1

while j >= 0:

if arr[j] > temp:

arr[j + 1] = arr[j]

arr[j] = temp

j -= 1

## **Select Sort**

Select Sort — это алгоритм, который сортирует массив путем многократного поиска минимального элемента (с учетом возрастания) из несортированной части и размещения его в начале. Алгоритм поддерживает два подмассива в данном массиве.

Пример алгоритма:

k = 0

for k in range(0, dim - 1):

m = k

i = k + 1

for i in range(i, dim):

if arr[i] < arr[m]:

m = i

if k != m:

t = arr[k]

arr[k] = arr[m]

arr[m] = t

## **Анализ методов сортировок:**

Теперь хочется выяснить какой из представленных алгоритмов самый эффективный. Для этого мы написали программный код, который создаёт три массива (со случайной, упорядоченной и обратно упорядоченной последовательностью). Наш код будет считать сколько было сравнений и сколько было перестановок, чем меньше показатели, тем эффективней алгоритм сортировки.

Программа:

import select

import bubble

import insert

import random

dim = 40

bubble\_arr = []

insert\_arr = []

select\_arr = []

CTotal = [0, 0, 0]

MTotal = [0, 0, 0]

for i in range(1, dim+1):

select\_arr.append(i)

bubble\_arr.append(i)

insert\_arr.append(i)

print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:")

print(select\_arr)

count = [0, 0]

count = select.select(select\_arr, dim)

print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:")

print(select\_arr)

CTotal[0] = count[0]

MTotal[0] = count[1]

count = [0, 0]

count = insert.insert(insert\_arr, dim)

CTotal[1] = count[0]

MTotal[1] = count[1]

count = [0, 0]

count = bubble.bubble(bubble\_arr, dim)

CTotal[2] = count[0]

MTotal[2] = count[1]

print("УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:\n")

print("Размер массива: ", dim)

print("Сравнений: ", CTotal[0], " ", CTotal[1], " ", CTotal[2])

print("Перестановок: ", MTotal[0], " ", MTotal[1], " ", MTotal[2])

select\_arr.clear()

bubble\_arr.clear()

insert\_arr.clear()

for i in range(dim, 0, -1):

select\_arr.append(i)

bubble\_arr.append(i)

insert\_arr.append(i)

print("\nОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Исходный массив")

print(select\_arr)

count = [0, 0]

count = select.select(select\_arr, dim)

print("\nОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив")

print(select\_arr)

CTotal[0] = count[0]

MTotal[0] = count[1]

count = [0, 0]

count = insert.insert(insert\_arr, dim)

CTotal[1] = count[0]

MTotal[1] = count[1]

count = [0, 0]

count = bubble.bubble(bubble\_arr, dim)

CTotal[2] = count[0]

MTotal[2] = count[1]

print("Размер массива: ", dim)

print("Сравнений: ", CTotal[0], " ", CTotal[1], " ", CTotal[2])

print("Перестановок: ", MTotal[0], " ", MTotal[1], " ", MTotal[2])

NUM = 1500

CTotal.clear()

MTotal.clear()

CTotal = [0, 0, 0]

MTotal = [0, 0, 0]

for n in range(0, NUM):

select\_arr.clear()

bubble\_arr.clear()

insert\_arr.clear()

select\_arr = [random.randint(0, 100) for i in range(dim)]

for i in range(0, dim):

bubble\_arr.append(select\_arr[i])

insert\_arr.append(select\_arr[i])

count = [0, 0]

count = select.select(select\_arr, dim)

CTotal[0] += count[0]

MTotal[0] += count[1]

count = [0, 0]

count = insert.insert(insert\_arr, dim)

CTotal[1] += count[0]

MTotal[1] += count[1]

count = [0, 0]

count = bubble.bubble(bubble\_arr, dim)

CTotal[2] += count[0]

MTotal[2] += count[1]

print("\nСЛУЧАЙНАЯ РЕАЛИЗАЦИЯ:")

print("Проведено экспериментов: ", NUM)

print("Размер массива: ", dim)

print("Сравнений: ", CTotal[0]/NUM, " ", CTotal[1]/NUM, " ", CTotal[2]/NUM)

print("Перестановок: ", MTotal[0]/NUM, " ", MTotal[1]/NUM, " ", MTotal[2]/NUM)

Результат:

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

Размер массива: 40

Сравнений: 780 780 780

Перестановок: 0 0 0

ОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Исходный массив

[40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

ОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]

Размер массива: 40

Сравнений: 780 780 780

Перестановок: 20 780 780

Результаты показали, что наименьшее число сравнений и перестановок у метода Selеct. Это значит, что этот алгоритм является самым эффективным.

# **Глава 2. Библиотека Numpy**

## **Что такое Numpy?**

NumPy - это фундаментальный пакет для научных вычислений на Python. Он содержит среди прочего:

* мощный N-мерный массив объектов
* сложные (вещательные) функции
* инструменты для интеграции C / C ++ и кода Fortran
* полезная линейная алгебра, преобразование Фурье и возможности случайных чисел

Помимо очевидного научного использования, NumPy также может использоваться как эффективный многомерный контейнер общих данных. Произвольные типы данных могут быть определены. Это позволяет NumPy легко и быстро интегрироваться с широким спектром баз данных. NumPy лицензируется по лицензии BSD , что позволяет использовать ее с небольшими ограничениями.

## **Numpy на практике**

1. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти наибольший элемент столбца матрицы A,

для которого сумма абсолютных значений элементов максимальна.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

sum = A.sum(axis=0)

i = sum.argmax(axis=0)

max = A.max(axis=0)

max = max[i]

print("Наибольший элемент: {}".format(max))

Матрица:

[[42 94 49 33 35 39 25 49]

[53 24 16 83 40 57 91 62]

[32 64 96 67 2 15 48 55]

[16 56 10 46 86 29 28 94]

[33 1 21 31 49 35 46 8]

[50 87 38 12 55 76 68 33]

[85 62 30 73 93 35 48 90]]

Наибольший элемент: 94

2. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти наибольшее значение среди средних

значений для каждой строки матрицы.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

average = np.average(A, axis=1)

max = np.max(average)

print("Наибольшее среднее значение: {}".format(max))

Матрица:

[[95 47 36 68 75 82 18]

[94 80 95 24 9 21 14]

[34 12 66 40 50 97 15]

[82 76 22 22 10 41 72]

[20 98 70 52 17 36 37]

[57 28 21 45 79 96 83]

[14 35 23 1 51 23 60]

[63 43 15 12 30 67 9]

[76 35 25 30 4 15 48]]

Наибольшее среднее значение: 60.142857142857146

3. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти наименьший элемент столбца матрицы A,

для которого сумма абсолютных значений элементов максимальна.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

sum = A.sum(axis=0)

i = sum.argmin(axis=0)

min = A.min(axis=0)

min = min[i]

print("Минимальный элемент: {}".format(min))

Матрица:

[[62 76 71 64 76 55 50 97 55 87]

[92 88 17 79 3 5 70 90 0 30]

[40 65 43 70 62 63 98 67 36 22]

[80 83 81 3 36 97 87 40 25 2]

[34 89 95 28 81 79 73 9 16 87]

[ 3 36 87 79 24 20 75 42 81 80]

[66 26 88 40 38 65 88 48 93 46]

[29 50 48 89 39 55 67 72 80 47]

[ 8 98 82 58 8 5 62 32 13 0]

[88 43 75 70 32 90 36 21 19 94]]

Минимальный элемент: 3

4. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти наименьшее значение среди средних

значений для каждой строки матрицы.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

Average = A.mean(axis=1)

index = Average.argmin(axis=0)

min = Average.min(axis=0)

print("Наименьшее значение среди средних значений: {}".format(min))

Матрица:

[[83 41 3 42 80 19 10]

[24 24 17 22 39 69 24]

[29 17 71 35 82 36 42]

[65 80 66 10 54 50 6]

[51 63 49 46 49 93 78]

[59 68 58 31 37 8 89]]

Наименьшее значение среди средних значений: 31.285714285714285

5. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Определить средние значения по всем строкам и

столбцам матрицы. Результат оформить в виде матрицы из N + 1 строк и M

+ 1 столбцов.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

average\_rows = np.average(A, axis=1)

average\_cols = np.average(A, axis=0)

A = np.vstack((A, [average\_cols]))

A = np.column\_stack((A, np.append(average\_rows, 0)))

print("Новая матрица:\n {} ".format(A))

Матрица:

[[90 5 34 83 72]

[48 62 3 44 84]

[ 5 53 33 75 73]

[28 15 95 91 22]

[20 8 40 49 9]]

Новая матрица:

[[90. 5. 34. 83. 72. 56.8]

[48. 62. 3. 44. 84. 48.2]

[ 5. 53. 33. 75. 73. 47.8]

[28. 15. 95. 91. 22. 50.2]

[20. 8. 40. 49. 9. 25.2]

[38.2 28.6 41. 68.4 52. 0. ]]

6. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти сумму элементов всей матрицы.

Определить, какую долю в этой сумме составляет сумма элементов каждого

столбца. Результат оформить в виде матрицы из N + 1 строк и M столбцов.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

sum\_e = np.sum(A)

sum\_c = np.sum(A, axis=0)

A = np.vstack((A, [sum\_c/sum\_e]))

print("Новая матрица:\n {} ".format(A))

Матрица:

[[75 35 37]]

Новая матрица:

[[75. 35. 37. ]

[ 0.51020408 0.23809524 0.25170068]]

7. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти сумму элементов всей матрицы.

Определить, какую долю в этой сумме составляет сумма элементов каждой

строки. Результат оформить в виде матрицы из N строк и M+1 столбцов.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

sum\_elements = np.sum(A)

sum\_cols = np.sum(A, axis=1)

A = np.column\_stack((A, sum\_cols/sum\_elements))

print("Новая матрица:\n {} ".format(A))

Матрица:

[[19 28 65 19]

[16 18 90 47]

[26 69 66 15]

[63 33 32 33]

[14 35 41 28]]

Новая матрица:

[[19. 28. 65. 19. 0.17305152]

[16. 18. 90. 47. 0.22589168]

[26. 69. 66. 15. 0.2324967 ]

[63. 33. 32. 33. 0.21268164]

[14. 35. 41. 28. 0.15587847]]

8. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Определить, сколько отрицательных элементов

содержится в каждом столбце и в каждой строке матрицы. Результат

оформить в виде матрицы из N + 1 строк и M + 1 столбцов.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(-100, 100, (N, M))

print("Матрица:\r\n{}".format(A))

A\_bool = A < 0

num\_cols = np.sum(A\_bool, axis=0)

num\_rows = np.sum(A\_bool, axis=1)

A = np.vstack((A, [num\_cols]))

A = np.column\_stack((A, np.append(num\_rows, 0)))

print("Новая матрица:\n {} ".format(A))

Матрица:

[[-39 17 -38 -54 -61 -78 92 -25 -47 -61]

[-55 62 -83 57 31 1 -27 39 44 -71]

[-17 88 70 89 90 -97 -85 -63 33 33]

[-87 -24 80 -98 81 23 65 97 -59 -7]

[-44 42 -72 -76 -57 45 -2 12 85 17]

[-80 -45 -76 -44 -28 33 -15 61 48 64]

[-52 -84 -62 67 -27 68 -72 -35 -32 -16]

[-84 10 -45 -13 -83 -7 -74 85 63 64]

[-77 -35 -18 26 -99 6 77 -24 -90 -91]

[-27 95 61 -6 -2 2 -98 80 -31 38]]

Новая матрица:

[[-39 17 -38 -54 -61 -78 92 -25 -47 -61 8]

[-55 62 -83 57 31 1 -27 39 44 -71 4]

[-17 88 70 89 90 -97 -85 -63 33 33 4]

[-87 -24 80 -98 81 23 65 97 -59 -7 5]

[-44 42 -72 -76 -57 45 -2 12 85 17 5]

[-80 -45 -76 -44 -28 33 -15 61 48 64 6]

[-52 -84 -62 67 -27 68 -72 -35 -32 -16 8]

[-84 10 -45 -13 -83 -7 -74 85 63 64 6]

[-77 -35 -18 26 -99 6 77 -24 -90 -91 7]

[-27 95 61 -6 -2 2 -98 80 -31 38 5]

[ 10 4 7 6 7 3 7 4 5 5 0]]

9. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Определить, сколько нулевых элементов

содержится в верхних L строках матрицы и в левых К столбцах матрицы. import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

K = random.randint(1, M)

L = random.randint(1, N)

A = np.random.randint(0, 2, (N, M))

print("Матрица:\r\n{}".format(A))

A\_slice = A[:L, :K]

print("Срез:\r\n{}".format(A\_slice))

slice\_bool = A\_slice == 0

zero\_elements = slice\_bool.sum()

print("Количество элементов:\n {} ".format(zero\_elements))

Матрица:

[[0 0 0 0 0 1]

[0 0 1 0 1 1]

[0 1 0 0 1 0]

[1 1 0 1 0 0]]

Срез:

[[0 0 0]]

Количество элементов:

3

10. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Перемножить элементы каждого столбца матрицы

с соответствующими элементами K-го столбца.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

K = random.randint(1, M)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

a = A[:, K]

a = a.reshape(len(a), 1)

A = A \* a

print("Новая матрица:\n {} ".format(A))

Матрица:

[[18 92 94 45]

[ 7 7 41 1]

[ 8 20 54 18]]

Новая матрица:

[[ 810 4140 4230 2025]

[ 7 7 41 1]

[ 144 360 972 324]]

11. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Просуммировать элементы каждой строки

матрицы с соответствующими элементами L-й строки.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

L = random.randint(1, N)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

A = A + A[L]

print("Новая матрица:\n {} ".format(A))

Матрица:

[[47 16 76 74 31 3 27 12 3 75]

[72 8 66 42 24 1 27 87 28 71]

[69 52 20 78 54 1 17 53 15 72]]

Новая матрица:

[[116 68 96 152 85 4 44 65 18 147]

[141 60 86 120 78 2 44 140 43 143]

[138 104 40 156 108 2 34 106 30 144]]

12. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Разделить элементы каждой строки на элемент

этой строки с наибольшим значением.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

max = np.max(A, axis=1)

A = A / max.reshape(len(max), 1)

print("Новая матрица:\n {} ".format(A))

Матрица:

[[58 77 14 44 67 75]

[90 86 54 96 35 47]

[ 0 97 39 53 63 66]]

Новая матрица:

[[0.75324675 1. 0.18181818 0.57142857 0.87012987 0.97402597]

[0.9375 0.89583333 0.5625 1. 0.36458333 0.48958333]

[0. 1. 0.40206186 0.54639175 0.64948454 0.68041237]]

13. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Разделить элементы каждого столбца матрицы на

элемент этого столбца с наибольшим значением.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

max = np.max(A, axis=0)

A = A / max

print("Новая матрица:\n {} ".format(A))

Матрица:

[[10 38 69 36 61 87]

[31 88 48 25 97 78]

[76 72 94 91 46 0]

[98 48 86 39 87 3]

[28 60 56 13 50 50]]

Новая матрица:

[[0.10204082 0.43181818 0.73404255 0.3956044 0.62886598 1. ]

[0.31632653 1. 0.5106383 0.27472527 1. 0.89655172]

[0.7755102 0.81818182 1. 1. 0.4742268 0. ]

[1. 0.54545455 0.91489362 0.42857143 0.89690722 0.03448276]

[0.28571429 0.68181818 0.59574468 0.14285714 0.51546392 0.57471264]]

14. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Разделить элементы матрицы на элемент матрицы

с наибольшим значением.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

max = np.max(A)

A = A / max

print("Новая матрица:\n {} ".format(A))

Матрица:

[[76 84 41 62 69]

[76 79 10 98 34]

[81 73 70 89 24]

[91 6 12 8 11]

[20 19 5 19 23]

[81 46 57 70 15]

[ 8 24 7 58 69]

[59 22 53 33 35]

[23 96 19 2 77]

[68 44 85 85 94]]

Новая матрица:

[[0.7755102 0.85714286 0.41836735 0.63265306 0.70408163]

[0.7755102 0.80612245 0.10204082 1. 0.34693878]

[0.82653061 0.74489796 0.71428571 0.90816327 0.24489796]

[0.92857143 0.06122449 0.12244898 0.08163265 0.1122449 ]

[0.20408163 0.19387755 0.05102041 0.19387755 0.23469388]

[0.82653061 0.46938776 0.58163265 0.71428571 0.15306122]

[0.08163265 0.24489796 0.07142857 0.59183673 0.70408163]

[0.60204082 0.2244898 0.54081633 0.33673469 0.35714286]

[0.23469388 0.97959184 0.19387755 0.02040816 0.78571429]

[0.69387755 0.44897959 0.86734694 0.86734694 0.95918367]]

15. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Все элементы имеют целый тип. Дано целое число

H. Определить, какие столбцы имеют хотя бы одно такое число, а какие не

имеют.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

H = random.randint(1, 10)

A = np.random.randint(0, 10, (N, M))

print("Матрица:\r\n{}".format(A))

A\_bool = A == H

row\_sum = np.sum(A\_bool, axis=0)

print("Столбцы в которых встречается значение {}:".format(H))

print(np.argwhere(row\_sum).flatten())

print("Столбцы в которых нет значения {}:".format(H))

print(np.argwhere(row\_sum == 0).flatten())

Матрица:

[[1 3 5 0 1 0 1 4 2 6]

[7 1 0 6 3 1 1 8 9 9]

[8 5 2 9 9 0 8 0 7 3]

[7 4 0 1 1 3 6 7 0 3]

[1 5 9 1 4 4 5 3 8 7]

[5 9 1 7 2 0 4 4 9 6]]

Столбцы в которых встречается значение 5:

[0 1 2 6]

Столбцы в которых нет значения 5:

[3 4 5 7 8 9]

16. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Исключить из матрицы строку с номером L.

Сомкнуть строки матрицы.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

L = random.randint(1, M)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

A = np.delete(A, L, axis=0)

print("Новая матрица:\n {} ".format(A))

Матрица:

[[89 57 44 24 31]

[ 7 97 17 24 87]

[38 48 92 16 70]

[26 54 5 56 81]

[ 4 95 50 93 18]

[92 59 93 10 25]

[ 0 51 99 72 61]]

Новая матрица:

[[89 57 44 24 31]

[ 7 97 17 24 87]

[38 48 92 16 70]

[26 54 5 56 81]

[ 4 95 50 93 18]

[ 0 51 99 72 61]]

17. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Добавить к матрице строку и вставить ее под

номером L.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

L = random.randint(1, M)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

row = np.random.randint(-9, 10, M)

print("Строка для вставки: {}".format(row))

A = np.insert(A, L, row, axis=0)

print("Новая матрица:\n {} ".format(A))

Матрица:

[[86 47]

[75 5]

[68 77]

[26 59]

[56 57]

[78 62]]

Строка для вставки: [-8 -1]

Новая матрица:

[[86 47]

[-8 -1]

[75 5]

[68 77]

[26 59]

[56 57]

[78 62]]

18. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти сумму элементов, стоящих на главной

диагонали, и сумму элементов, стоящих на побочной диагонали (элементы

главной диагонали имеют индексы от [0,0] до [N,N], а элементы побочной

диагонали — от [N,0] до [0,N]).

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

L = random.randint(1, M)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

diagonal\_main = np.diagonal(A)

print("Элементы главной диагонали:\r\n{}".format(diagonal\_main))

sum\_diagonal\_main = np.sum(diagonal\_main)

print("Cумма элементов главной диагонали:\r\n{}".format(sum\_diagonal\_main))

diagonal\_side = np.diagonal(A[::-1])

print("Элементы побочной диагонали:\r\n{}".format(diagonal\_side))

sum\_diagonal\_side = np.sum(diagonal\_side)

print("сумму элементов побочной диагонали:\r\n{}".format(sum\_diagonal\_side))

Матрица:

[[70 8 20 44 39 50 58]

[98 3 70 64 49 46 71]

[43 3 4 43 10 25 0]

[11 24 56 55 39 63 40]

[60 89 78 47 44 89 42]

[71 73 83 81 66 35 33]

[31 92 84 82 87 35 3]

[48 77 3 80 72 68 90]]

Элементы главной диагонали:

[70 3 4 55 44 35 3]

Cумма элементов главной диагонали:

214

Элементы побочной диагонали:

[48 92 83 47 39 25 71]

сумму элементов побочной диагонали:

405

19. Создать квадратную матрицу A, имеющую N строк и N столбцов со

случайными элементами. Определить сумму элементов, расположенных

параллельно главной диагонали (ближайшие к главной). Элементы главной

диагонали имеют индексы от [0,0] до [N,N].

import numpy as np

import random

N = random.randint(1, 10)

A = np.random.randint(0, 100, (N, N))

print("Матрица:\r\n{}".format(A))

a = np.diagonal(A, 1)

a\_sum = a.sum()

print("Элементы которые выше главной диагонали: \n" + str(a) + "\nИх сумма = " + str(a\_sum))

b = np.diagonal(A, -1)

b\_sum = b.sum()

print("Элементы которые ниже главной диагонали: \n" + str(b) + "\nИх сумма = " + str(a\_sum))

Матрица:

[[28 77 46 33 6 21 16 92 89]

[12 39 75 66 81 1 7 87 12]

[ 3 41 71 23 60 56 28 47 90]

[ 2 1 58 87 51 97 93 99 41]

[ 2 49 20 73 84 97 65 89 18]

[52 45 65 54 9 56 83 69 53]

[16 68 9 50 75 20 48 76 83]

[57 70 41 15 24 33 71 68 46]

[22 23 59 19 2 4 46 15 86]]

Элементы которые выше главной диагонали:

[77 75 23 51 97 83 76 46]

Их сумма = 528

Элементы которые ниже главной диагонали:

[12 41 58 73 9 20 71 15]

Их сумма = 528

20. Создать квадратную матрицу A, имеющую N строк и N столбцов со

случайными элементами. Определить произведение элементов,

расположенных параллельно побочной диагонали (ближайшие к побочной).

Элементы побочной диагонали имеют индексы от [N,0] до [0,N].

import numpy as np

import random

N = random.randint(1, 10)

A = np.random.randint(0, 100, (N, N))

print("Матрица:\r\n{}".format(A))

a = b = np.fliplr(A).diagonal(1)

a\_prod = a.prod()

print("Элементы которые выше побочной диагонали: \n" + str(a) + "\nИх сумма = " + str(a\_prod))

b = np.fliplr(A).diagonal(-1)

b\_prod = b.prod()

print("Элементы которые ниже побочной диагонали: \n" + str(b) + "\nИх сумма = " + str(b\_prod))

Матрица:

[[66 59 85 45 55 76 86 42]

[48 40 51 34 75 69 11 88]

[15 64 61 51 97 53 49 71]

[89 31 12 75 12 48 7 59]

[17 56 39 87 2 4 37 76]

[48 29 82 65 42 10 29 82]

[36 64 4 45 52 15 36 79]

[57 18 11 58 47 40 24 2]]

Элементы которые выше побочной диагонали:

[86 69 97 75 39 29 36]

Их сумма = 1061988536

Элементы которые ниже побочной диагонали:

[88 49 48 2 65 4 18]

Их сумма = 1937295360

21. Создать квадратную матрицу A, имеющую N строк и N столбцов со

случайными элементами. Каждой паре элементов, симметричных

относительно главной диагонали (ближайшие к главной), присвоить

значения, равные полусумме этих симметричных значений (элементы

главной диагонали имеют индексы от [0,0] до [N,N]).

import numpy as np

import random

N = random.randint(1, 10)

A = np.random.randint(0, 100, (N, N))

print("Матрица:\r\n{}".format(A))

for i in range(1, N-1):

b = (A[i+1, i-1] + A[i-1, i+1])/2

A[i+1, i-1] = b

A[i-1, i+1] = b

print("Новая матрица:\n {} ".format(A))

Матрица:

[[54 68 28 56 7 41 46]

[19 84 49 11 56 89 84]

[44 84 64 16 78 78 50]

[96 21 98 47 15 10 22]

[ 6 25 35 32 93 28 57]

[98 7 91 63 51 3 51]

[34 52 65 95 72 34 33]]

Новая матрица:

[[54 68 36 56 7 41 46]

[19 84 49 16 56 89 84]

[36 84 64 16 56 78 50]

[96 16 98 47 15 36 22]

[ 6 25 56 32 93 28 64]

[98 7 91 36 51 3 51]

[34 52 65 95 64 34 33]]

22. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Исходная матрица состоит из нулей и единиц.

Добавить к матрице еще один столбец, каждый элемент которого делает

количество единиц в каждой строке чётным.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

A = np.random.randint(0, 2, (N, M))

print("Матрица:\r\n{}".format(A))

a = [i % 2 for i in np.sum(A, axis=1)]

A = np.insert(A, M, a, axis=1)

print("Новая матрица:\n {} ".format(A))

Матрица:

[[1 1 1 0 1 1]

[1 1 0 1 0 0]

[1 1 1 0 0 1]

[1 1 1 1 1 0]

[0 1 1 1 1 0]]

Новая матрица:

[[1 1 1 0 1 1 1]

[1 1 0 1 0 0 1]

[1 1 1 0 0 1 0]

[1 1 1 1 1 0 1]

[0 1 1 1 1 0 0]]

23. Создать квадратную матрицу A, имеющую N строк и N столбцов со

случайными элементами. Найти сумму элементов, расположенных выше

главной диагонали, и произведение элементов, расположенных выше

побочной диагонали (элементы главной диагонали имеют индексы от [0,0]

до [N,N], а элементы побочной диагонали — от [N,0] до [0,N]).

import numpy as np

import random

N = random.randint(1, 10)

A = np.random.randint(0, 100, (N, N))

print("Матрица:\r\n{}".format(A))

iu = np.triu\_indices(N, 1)

a = A[iu]

a = np.sum(np.array(a))

print("\nCумма элементов выше главной диагонали = " + str(a))

b = np.fliplr(A)[iu]

b = np.prod(np.array(b))

print("\nПроизведение элементов выше побочной диагонали = " + str(b))

Матрица:

[[91 38 89]

[45 93 33]

[ 5 81 86]]

Cумма элементов выше главной диагонали = 160

Произведение элементов выше побочной диагонали = 155610

24. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Дан номер строки L и номер столбца K, при

помощи которых исходная матрица разбивается на четыре части. Найти

сумму элементов каждой части.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

K = random.randint(1, M)

L = random.randint(1, N)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

parts = [

A[:L, :K],

A[:L, K:],

A[L:, :K],

A[L:, K:],]

for i in range(len(parts)):

print("Cумма элементов {} части: {}".format(i+1, np.sum(parts[i])))

Матрица:

[[24 23 56 27 21 93]

[55 32 20 23 33 41]

[82 35 69 24 76 82]

[52 64 80 21 36 35]]

Cумма элементов 1 части: 592

Cумма элементов 2 части: 512

Cумма элементов 3 части: 0

Cумма элементов 4 части: 0

25. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Определить, сколько нулевых элементов

содержится в каждом столбце и в каждой строке матрицы. Результат

оформить в виде матрицы из N + 1 строк и M + 1 столбцов.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

K = random.randint(1, M)

L = random.randint(1, N)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

A\_bool = A == 0

col = np.sum(A\_bool, axis=1)

A = np.insert(A, M, col, axis=1)

row = np.append(np.sum(A\_bool, axis=0), 0)

A = np.insert(A, N, row, axis=0)

print("Новая матрица:\n {} ".format(A))

Матрица:

[[58]

[71]

[87]

[82]

[84]

[71]

[97]

[34]

[76]

[ 1]]

Новая матрица:

[[58 0]

[71 0]

[87 0]

[82 0]

[84 0]

[71 0]

[97 0]

[34 0]

[76 0]

[ 1 0]

[ 0 0]]

26. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Дан номер строки L и номер столбца K, при

помощи которых исходная матрица разбивается на четыре части. Найти

среднее арифметическое элементов каждой части.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

K = random.randint(1, M-1)

L = random.randint(1, N-1)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

parts = [

A[:L, :K],

A[:L, K:],

A[L:, :K],

A[L:, K:],]

for i in range(len(parts)):

print("Cреднее арифметическое {} части: {}".format(i+1, np.average(parts[i])))

Матрица:

[[95 25 10 29]

[23 4 13 3]

[77 97 18 83]

[ 2 20 5 48]

[39 15 93 65]

[ 5 67 88 32]

[53 79 66 16]]

Cреднее арифметическое 1 части: 40.166666666666664

Cреднее арифметическое 2 части: 39.72222222222222

Cреднее арифметическое 3 части: 53.0

Cреднее арифметическое 4 части: 53.666666666666664

27. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Все элементы имеют целый тип. Дано целое число

H. Определить, какие строки имеют хотя бы одно такое число, а какие не

имеют.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

H = random.randint(1, M-1)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

A\_bool = A == H

col\_sum = np.sum(A\_bool, axis=1)

print("Строки в которых встречается значение {}:".format(H))

print(np.argwhere(col\_sum).flatten())

print("Строки в которых нет значения {}:".format(H))

print(np.argwhere(col\_sum == 0).flatten())

Матрица:

[[39 66 70 40 44 90 23 82]

[37 84 59 63 70 33 93 31]

[59 84 52 99 53 40 96 27]

[ 2 94 83 85 13 22 7 13]

[23 86 42 48 63 4 43 29]

[44 7 42 51 0 81 85 79]

[39 25 58 88 62 95 19 1]

[46 85 34 14 94 91 93 79]

[51 60 33 1 62 1 54 0]]

Строки в которых встречается значение 4:

[4]

Строки в которых нет значения 4:

[0 1 2 3 5 6 7 8]

28. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Исключить из матрицы столбец с номером K.

Сомкнуть столбцы матрицы.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

K = random.randint(1, M-1)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

A = np.delete(A, K, axis=1)

print("Новая матрица:\n {} ".format(A))

Матрица:

[[36 34 57 85 54 2]

[44 15 36 39 36 72]

[52 84 14 98 89 15]

[62 7 27 13 58 63]

[84 71 58 84 90 52]]

Новая матрица:

[[36 34 57 85 54]

[44 15 36 39 36]

[52 84 14 98 89]

[62 7 27 13 58]

[84 71 58 84 90]]

29. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Добавить к матрице столбец чисел и вставить его

под номером K.

import numpy as np

import random

N = random.randint(1, 10)

M = random.randint(1, 10)

K = random.randint(1, M)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

col = np.random.randint(-9, 10, N)

print("Столбец для вставки: {}".format(col))

A = np.insert(A, K, col, axis=1)

print("Новая матрица:\n {} ".format(A))

Матрица:

[[ 9 82 80 3 27 54 41 71 68]

[40 31 19 86 51 20 26 40 15]

[73 38 1 54 98 83 94 98 74]

[45 22 21 87 8 93 96 55 54]

[74 86 64 77 33 0 95 9 51]

[96 97 53 55 2 46 28 11 65]

[75 91 81 15 44 95 33 58 13]]

Столбец для вставки: [ 6 -1 -7 -9 -9 -9 -3]

Новая матрица:

[[ 9 82 80 3 27 54 41 71 6 68]

[40 31 19 86 51 20 26 40 -1 15]

[73 38 1 54 98 83 94 98 -7 74]

[45 22 21 87 8 93 96 55 -9 54]

[74 86 64 77 33 0 95 9 -9 51]

[96 97 53 55 2 46 28 11 -9 65]

[75 91 81 15 44 95 33 58 -3 13]]

30. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Добавить к элементам каждого столбца такой

новый элемент, чтобы сумма положительных элементов стала бы равна

модулю суммы отрицательных элементов. Результат оформить в виде

матрицы из N + 1 строк и M столбцов.

import numpy as np

import random

N = np.random.randint(1, 10)

M = np.random.randint(1, 10)

K = np.random.randint(1, N)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

M\_n = np.sum(A, axis=0) \* (-1)

A = np.vstack((A, M\_n))

print("Новая матрица:\r\n{}\n".format(A))

Матрица:

[[85 37 23 70 13 5]

[32 10 54 63 16 56]

[22 53 91 26 22 96]

[20 40 28 5 41 81]

[69 55 85 35 71 39]

[43 92 35 53 3 72]

[45 51 71 61 24 47]

[32 70 80 49 95 1]]

Новая матрица:

[[ 85 37 23 70 13 5]

[ 32 10 54 63 16 56]

[ 22 53 91 26 22 96]

[ 20 40 28 5 41 81]

[ 69 55 85 35 71 39]

[ 43 92 35 53 3 72]

[ 45 51 71 61 24 47]

[ 32 70 80 49 95 1]

[-348 -408 -467 -362 -285 -397]]

31. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Добавить к элементам каждой строки такой новый

элемент, чтобы сумма положительных элементов стала бы равна модулю

суммы отрицательных элементов. Результат оформить в виде матрицы из N

строк и M + 1 столбцов.

import numpy as np

import random

N = np.random.randint(1, 10)

M = np.random.randint(1, 10)

K = np.random.randint(1, M)

A = np.random.randint(0, 100, (N, M))

print("Матрица:\r\n{}".format(A))

M\_n = np.sum(A, axis=1) \* (-1)

A = np.hstack((A, M\_n.reshape(-1, 1)))

print("Новая матрица:\r\n{}\n".format(A))

Матрица:

[[90 73]

[27 18]

[42 97]

[74 35]

[37 69]]

Новая матрица:

[[ 90 73 -163]

[ 27 18 -45]

[ 42 97 -139]

[ 74 35 -109]

[ 37 69 -106]]

# **Глава 3. Решаем СЛАУ на Python**

Метод Гаусса прекрасно подходит для решения систем линейных алгебраических уравнений (СЛАУ). Он обладает рядом преимуществ по сравнению с другими методами:

* во-первых, нет необходимости предварительно исследовать систему уравнений на совместность;
* во-вторых, методом Гаусса можно решать не только СЛАУ, в которых число уравнений совпадает с количеством неизвестных переменных и основная матрица системы невырожденная, но и системы уравнений, в которых число уравнений не совпадает с количеством неизвестных переменных или определитель основной матрицы равен нулю;
* в-третьих, метод Гаусса приводит к результату при сравнительно небольшом количестве вычислительных операций.

Так как в пакете Numpy есть функция, которая решает СЛАУ методом гаусса, то мы не будем описывать полный алгоритм метода, а срезу перейдём к практике.

Вариант №2

Дано:

Задача №1

| 1.06 1.76 1.47 1.03 1.69 | \* | X1| = | 0.10|

| 1.74 1.28 1.59 1.49 1.39 | \* | X2| = | 0.35|

| 1.19 1.11 1.23 1.62 1.39 | \* | X3| = | 0.95|

| 1.88 1.14 1.34 1.73 1.87 | \* | X4| = | 0.81|

| 1.68 1.51 1.14 1.52 1.67 | \* | X5| = | 0.15|

Задача №2

| 1.55 1.33 1.36 1.15 1.90 1.68 | \* | X1| = | 0.09|

| 1.95 1.29 1.66 1.14 1.10 1.25 | \* | X2| = | 0.25|

| 1.44 1.21 1.35 1.63 1.80 1.51 | \* | X3| = | 0.98|

| 1.95 1.48 1.92 1.47 1.20 1.62 | \* | X4| = | 0.20|

| 1.62 1.54 1.29 1.37 1.73 1.50 | \* | X5| = | 0.52|

| 1.21 1.21 1.58 1.98 1.03 1.12 | \* | X6| = | 0.25|

Задача №3

| 1.08 1.55 1.96 | \* | X1| = | 0.68|

| 1.17 1.90 1.22 | \* | X2| = | 0.46|

| 1.28 1.33 1.54 | \* | X3| = | 0.42|

Задача №4

| 1.61 1.34 1.33 | \* | X1| = | 0.59|

| 1.55 1.35 1.60 | \* | X2| = | 0.98|

| 1.01 1.64 1.04 | \* | X3| = | 0.60|

Задача №5

| 1.18 1.16 | \* | X1| = | 0.60|

| 1.88 1.75 | \* | X2| = | 0.99|

Что бы решить данные системы мы написали программу которая будет открывать csv-файлы с задачами:

Программный код:

import numpy as np

for i in range(1, 6):

task\_file = "СЛАУ\_" + str(i) + ".csv"

m = np.genfromtxt(task\_file, delimiter=';')

A = np.genfromtxt(task\_file, delimiter=';', usecols=(range(len(m-1))))

B = np.genfromtxt(task\_file, delimiter=';', usecols=(len(m)))

slau = np.linalg.solve(A, B)

print("Решение" + str(i), slau)

Результат:

Решение1 [-0.85092993 -1.05625414 0.49268539 0.99840056 0.6558538 ]

Решение2 [ 3.25505781 -2.25412623 -4.4343031 2.47189657 -1.39564867 2.31092375]

Решение3 [-0.31928289 0.20921571 0.35741897]

Решение4 [-0.66715547 -0.0463372 1.29790387]

Решение5 [ 0.84974093 -0.34715026]

# **Заключение**

Мы выяснили, что Python сегодня один из самых востребованных языков программирования, потому как наилучшим образов приспособлен для решения популярных и востребованных задач глубокого машинного обучения, классификации, обработки больших данных и прочее, прочее, прочее. Уже давно есть мнение, что каждый человек XXI века должен уметь программировать и потому начинать надо уже в школе.

# **Библиографический список**

1. https://ru.wikipedia.org
2. https://waksoft.susu.ru/
3. https://www.python.org/