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 – 380305.2022.118. ПЗ КР

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

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

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г. «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

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

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

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Работа защищена с оценкой

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

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Челябинск 20\_\_

# **АННОТАЦИЯ**

Александровский А.П.

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-142, 2019.

44 с., 1 таблица, 0 рисунков.

Библиографический список – 14 наим.

В данной курсовой работе представлено: рассмотрение методов сортировки и принципы их работы, а также их эффективность; рассматривается библиотека Numpy, основные ее функции и применение ее на практике, а также решение СЛАУ с помощью библиотеки Numpy.

# **ОГЛАВЛЕНИЕ**

[**АННОТАЦИЯ** 2](#_Toc9833114)

[**ОГЛАВЛЕНИЕ** 3](#_Toc9833115)

[**ВВЕДЕНИЕ** 4](#_Toc9833116)

[**ГЛАВА 1 МЕТОДЫ СОРТИРОВКИ** 5](#_Toc9833117)

[**1.1** **Метод сортировки выбором** 5](#_Toc9833118)

[**1.2 Обменная сортировка** 5](#_Toc9833119)

[**1.3 Сортировка включением** 6](#_Toc9833120)

[**1.4 Сравнение методов сортировки** 6](#_Toc9833121)

[**ГЛАВА 2 БИБЛИОТЕКА NUMPY** 11](#_Toc9833122)

[**2.1 Знакомство с библиотекой** 11](#_Toc9833123)

[**2.2Применение Numpy на практике** 11](#_Toc9833124)

[**ГЛАВА 3 РЕШЕНИЕ СЛАУ МЕТОДОМ ГАУССА** 41](#_Toc9833125)

[**ЗАКЛЮЧЕНИЕ** 43](#_Toc9833126)

[**БИБЛИОГРАФИЧЕСКИЙ СПИСОК** 44](#_Toc9833127)

# **ВВЕДЕНИЕ**

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. Большой плюс Python в том, что для него есть огромное множество библиотек.

Что же такое библиотека? Библиоте́ка — это сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО), одна из них - это Numpy.

NumPy — NumPy — это open-source модуль для python, который предоставляет общие математические и числовые операции в виде прескомпилированных, быстрых функций. Основные возможности: поддержка многомерных массивов (включая матрицы); поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

В данной курсовой работе было выдвинуто три цели:

1. Изучение методов сортировки (Сортировка включением, Обменная сортировка, Сортировка выбором). В ходе данной работы было написано три алгоритма сортировки и один алгоритм сравнения методов сортировки.
2. Знакомство с библиотекой Numpy. В ходе работы было решено тридцать одна задача с двумерными массивами с применением библиотеки Numpy.
3. Понять суть и алгоритм решения СЛАУ методом Гаусса.

# **ГЛАВА 1 МЕТОДЫ СОРТИРОВКИ**

## **Метод сортировки выбором**

Метод сортировки выбором(Select sort) - судя по названию сортировки, необходимо что-то выбирать (максимальный или минимальный элементы массива). Алгоритм сортировки выбором находит в исходном массиве максимальный или минимальный элементы, в зависимости от того как необходимо сортировать массив, по возрастанию или по убыванию. Если массив должен быть отсортирован по возрастанию, то из исходного массива необходимо выбирать минимальные элементы. Если же массив необходимо отсортировать по убыванию, то выбирать следует максимальные элементы.

Код алгоритма:

**def** select\_sort(arr, dim):

alg\_count = [0, 0]

**for** i **in** range(0, dim):

min = i

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

alg\_count[0] += 1

**if** arr[j] < arr[min]:

min = j

**if** min != i:

arr[i], arr[min] = arr[min], arr[i]

alg\_count[1] += 1

**return** alg\_count

## **1.2 Обменная сортировка**

Обменная сортировка (Bubble\_sort) - это метод сортировки массивов и списков путем последовательного сравнения и обмена соседних элементов, если предшествующий оказывается больше последующего. В процессе выполнения данного алгоритма элементы с большими значениями оказываются в конце списка, а элементы с меньшими значениями постепенно перемещаются по направлению к началу списка. Образно говоря, тяжелые элементы падают на дно, а легкие медленно всплывают подобно пузырькам воздуха.

Код алгоритма:

**def** bubble\_sort(arr, dim):

alg\_count = [0, 0]

**for** j **in** range(0, dim-1):

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

alg\_count[0] += 1

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

alg\_count[1] += 1

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

**return** alg\_count

## **1.3 Сортировка включением**

Сортировка вставками (*Insertion Sort*) — это простой алгоритм сортировки. Суть его заключается в том что, на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Стоит отметить что массив из 1-го элемента считается отсортированным.

Код алгоритма:

**def insert\_sort(arr, dim):**

**alg\_count = [0, 0]**

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

**top = arr[i]**

**j = i - 1**

**while j >= 0:**

**alg\_count[0] += 1**

**if arr[j] > top:**

**alg\_count[1] += 1**

**arr[j + 1] = arr[j]**

**arr[j] = top**

**j -= 1**

**return alg\_count**

## **1.4 Сравнение методов сортировки**

В данной части курсовой работы была написана программа, которая определяет эффективность трех алгоритмов сортировки (Сортировка включением, Обменная сортировка, Сортировка выбором). Каждый алгоритм сортировки оформлен в виде функции, которые подключены к основной программе.

Проведено 9 опытов (3 метода х 3 реализации одномерного массива), в каждом из которых определено два числа (С - Количество сравнений М - Количество перемещений).

Код:

**import** Insert\_sort *#Сортировка включением*

**import** Bubble\_sort *#Обменная сортировка*

**import** Select\_sort *#Сортировка выбором*

**import** random

dim = 40

count\_move = [0, 0, 0]

count\_compare = [0, 0, 0]

insert\_arr = []

bubble\_arr = []

select\_arr = []

arr = []

*#1) УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ*

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

insert\_arr.append(i)

bubble\_arr.append(i)

select\_arr.append(i)

arr.append(i)

*# Сортировка включением*

Count = Insert\_sort.insert\_sort(insert\_arr, dim)

count\_move[0] = Count[0]

count\_compare[0] = Count[1]

*# Обменная сортировка*

Count = Bubble\_sort.bubble\_sort(bubble\_arr, dim)

count\_move[1] = Count[0]

count\_compare[1] = Count[1]

*# Сортировка выбором*

Count = Select\_sort.select\_sort(select\_arr, dim)

count\_move[2] = Count[0]

count\_compare[2] = Count[1]

print(**"УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: "**)

print(**"Начальный массив: "** + str(arr))

print(count\_move)

print(count\_compare)

*# 2) ОБРАТНОУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ*

insert\_arr = []

bubble\_arr = []

select\_arr = []

arr = []

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

insert\_arr.append(i)

bubble\_arr.append(i)

select\_arr.append(i)

arr.append(i)

*# Сортировка включением*

Count = Insert\_sort.insert\_sort(insert\_arr, dim)

count\_move[0] = Count[0]

count\_compare[0] = Count[1]

*# Обменная сортировка*

Count = Bubble\_sort.bubble\_sort(bubble\_arr, dim)

count\_move[1] = Count[0]

count\_compare[1] = Count[1]

*# Сортировка выбором*

Count = Select\_sort.select\_sort(select\_arr, dim)

count\_move[2] = Count[0]

count\_compare[2] = Count[1]

print(**"ОБРАТНОУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: "**)

print(**"Начальный массив: "** + str(arr))

print(count\_move)

print(count\_compare)

*# 3) СЛУЧАЙНАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ*

insert\_arr = []

bubble\_arr = []

select\_arr = []

arr = []

insert\_arr = [random.randint(0, 20) **for** i **in** range(dim)]

bubble\_arr = [random.randint(0, 20) **for** i **in** range(dim)]

select\_arr = [random.randint(0, 20) **for** i **in** range(dim)]

arr = [random.randint(0, 20) **for** i **in** range(dim)]

*# Сортировка включением*

Count = Insert\_sort.insert\_sort(insert\_arr, dim)

count\_move[0] = Count[0]

count\_compare[0] = Count[1]

*# Обменная сортировка*

Count = Bubble\_sort.bubble\_sort(bubble\_arr, dim)

count\_move[1] = Count[0]

count\_compare[1] = Count[1]

*# Сортировка выбором*

Count = Select\_sort.select\_sort(select\_arr, dim)

count\_move[2] = Count[0]

count\_compare[2] = Count[1]

print(**"СЛУЧАЙНАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: "**)

print(**"Случайный массив массив: "** + str(arr))

print(count\_move)

print(count\_compare)

Результат кода:

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

Начальный 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]

[780, 1521, 780]

[0, 0, 0]

ОБРАТНОУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

Начальный массив: [41, 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]

[780, 1521, 780]

[780, 780, 20]

СЛУЧАЙНАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

Случайный массив массив: [9, 16, 13, 17, 2, 10, 6, 10, 11, 2, 15, 7, 7, 3, 4, 3, 5, 19, 0, 19, 7, 17, 4, 18, 18, 15, 16, 3, 17, 10, 20, 3, 4, 4, 2, 11, 14, 4, 8, 13]

[780, 1521, 780]

[383, 376, 36]

Итоги:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Массив 40 элементов | Показатель | Insert | Buble | Select |
| УПОРЯДОЧЕННАЯ  ПОСЛЕДОВАТЕЛЬНОСТЬ | Сравнения | 780 | 1521 | 780 |
| Перемещения | 0 | 0 | 0 |
| ОБРАТНОУПОРЯДОЧЕННАЯ  ПОСЛЕДОВАТЕЛЬНОСТЬ | Сравнения | 780 | 1521 | 780 |
| Перемещения | 780 | 780 | 20 |
| СЛУЧАЙНАЯ  ПОСЛЕДОВАТЕЛЬНОСТЬ | Сравнения | 780 | 1521 | 780 |
| Перемещения | 383 | 376 | 36 |

## 

Таблица 1 - Итоги

Исходя из проделанной работы можно сделать вывод. Самым эффективным методом сортировки для массива является метод "Select", т. к. количество сравнений и перемещений меньше чем у других методов.

# **ГЛАВА 2 БИБЛИОТЕКА NUMPY**

## **2.1 Знакомство с библиотекой**

Сам по себе «чистый» Python пригоден только для несложных вычислений. Ключевая особенность Python — его расширяемость. Это, пожалуй, самый расширяемый язык из получивших широкое распространение. Как следствие этого для Python не только написаны и приспособлены многочисленные библиотеки алгоритмов на C и Fortran, но и имеются возможности использования других программных средств и математических пакетов, в частности, R и SciLab, а также графопостроителей, например, Gnuplot и PLPlot.

Одной из самых значимых библиотек для превращения Python в математический пакет являются Numpy и Matplotlib.

Numpy — это библиотека (модуль, в действительности, набор модулей) языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большим набором высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

Matplotlib — это библиотека (модуль, в действительности, набор модулей) на языке программирования Python для визуализации данных двумерной (2D) графикой (3D графика также поддерживается). Получаемые изображения могут быть использованы в качестве иллюстраций в публикациях.

Кроме Numpy и Matplotlib популярен модуль scipy для специализированных математических вычислений (поиск минимума и корней функции многих переменных, аппроксимация сплайнами, вейвлет-преобразования), sympy для символьных вычислений (аналитическое взятие интегралов, упрощение математических выражений), ffnet для построения искусственных нейронных сетей, pyopencl/pycuda для вычисления на видеокартах и некоторые другие. Возможности numpy и scipy покрывают практически все потребности в математических алгоритмах. Но далее речь пойдет о Numpy.

## **2.2Применение Numpy на практике**

В данной части курсовой работе было решено 31 задача с применением библиотеке Numpy. Далее пойдут задания и результаты их решения.

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Нахождение суммы для каждого столбца матрицы*

sum = A.sum(axis=0)

*# Определение номера столбца, для которого,*

*# сумма абсолютных значений элементов максимальна*

index = sum.argmax(axis=0)

*# Нахождение наибольшего элемента в каждом столбце матрицы*

max = A.max(axis=0)

*# Нахождение наибольшего элемента столбца матрицы,*

*# для которого сумма абсолютных значений элементов максимальна*

max = max[index]

print(**"Максимальный элемент: "** + str(max))

Результат:

[[5 0 9 5 3 5]

[2 9 9 0 8 2]

[0 7 0 3 0 1]

[9 5 8 2 6 9]

[8 9 5 4 7 8]]

Максимальный элемент: 9

2. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти наибольшее значение среди средних значений для каждой строки матрицы.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Нахождение средних значений для каждой строки матрицы*

Average = A.mean(axis=1)

*# Нахождение номера строки с наибольшим средним значением*

index = Average.argmax(axis=0)

*# Нахождение наибольшее значение среди средних*

*# значений для каждой строки матрицы.*

max = Average.max(axis=0)

print(**"Наибольшее значение среди средних значений: "** + str(max))

Результат:

[[7 1 4 9 6 1]

[1 9 4 3 0 0]

[2 5 4 3 6 2]

[3 4 3 9 5 3]]

Наибольшее значение среди средних значений: 4.666666666666667

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Нахождение суммы для каждого столбца матрицы*

sum = A.sum(axis=0)

*# Определение номера столбца, для которого,*

*# сумма абсолютных значений элементов минимальна*

index = sum.argmin(axis=0)

*# Нахождение наименьшего элемента в каждом столбце матрицы*

min = A.min(axis=0)

*# Нахождение наименьшего элемента столбца матрицы,*

*# для которого сумма абсолютных значений элементов максимальна*

min = min[index]

print(**"Минимальный элемент: "** + str(min))

Результат:

[[6 5 6 0 6 7 3 7 5 5]

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

[9 1 2 8 2 1 0 4 9 0]

[3 8 1 5 2 1 2 8 0 4]

[2 3 5 3 9 6 6 0 6 7]

[3 6 6 4 7 2 8 9 4 6]

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

[3 3 6 4 5 6 3 4 4 0]

[0 7 8 6 7 1 0 9 4 8]]

Минимальный элемент: 1

4. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти наименьшее значение среди средних значений для каждой строки матрицы.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Нахождение средних значений для каждой строки матрицы*

Average = A.mean(axis=1)

*# Нахождение номера строки с наибольшим средним значением*

index = Average.argmin(axis=0)

*# Нахождение наибольшее значение среди средних*

*# значений для каждой строки матрицы.*

min = Average.min(axis=0)

print(**"Наименьшее значение среди средних значений: "** + str(min))

Результат:

[[3 2 9 9]

[1 5 1 8]

[4 1 8 6]

[3 0 7 0]

[1 7 4 5]

[7 5 2 8]

[0 6 1 7]

[9 1 8 3]]

Наименьшее значение среди средних значений: 2.5

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

A = np.random.randint(0, 10, (N, M)).astype(np.float64)

print(str(A) + **"\n"**)

*# Нахождение средних значений для каждой строки и столбца матрицы*

Average\_line = A.mean(axis=1)

Average\_column = A.mean(axis=0)

*# Добавление столбца и строки к матрице A*

Average\_line = Average\_line[: , np.newaxis]

A = np.hstack((A, Average\_line))

Average\_column = np.hstack((Average\_column, [0.]))

A = np.vstack((A, Average\_column))

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[9. 8. 1. 6. 2.]

[2. 8. 6. 0. 2.]

[9. 7. 8. 4. 1.]

[0. 7. 8. 1. 4.]

[6. 5. 2. 4. 1.]

[4. 2. 8. 3. 1.]

[0. 0. 9. 1. 9.]

[7. 1. 2. 3. 8.]

[3. 2. 9. 9. 2.]

[5. 1. 7. 3. 3.]]

Новая матрица:

[[9. 8. 1. 6. 2. 5.2]

[2. 8. 6. 0. 2. 3.6]

[9. 7. 8. 4. 1. 5.8]

[0. 7. 8. 1. 4. 4. ]

[6. 5. 2. 4. 1. 3.6]

[4. 2. 8. 3. 1. 3.6]

[0. 0. 9. 1. 9. 3.8]

[7. 1. 2. 3. 8. 4.2]

[3. 2. 9. 9. 2. 5. ]

[5. 1. 7. 3. 3. 3.8]

[4.5 4.1 6. 3.4 3.3 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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

A = np.random.randint(0, 10, (N, M)).astype(np.float64)

print(str(A) + **"\n"**)

*# Определение суммы элементов всей матрицы*

Sum = A.sum()

print(**"Сумма элементов всей матрицы: "** + str(Sum) + **"\n"**)

*#Определение суммы всех элентов кождого столбца матрицы*

Sum\_column = A.sum(axis=0)

*#Определение доли суммы*

X = []

**for** i **in** range(0, M):

n = Sum\_column[i] / Sum

X.append(n)

*#Добавление строки*

A = np.vstack((A, X))

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[9. 6. 5. 3.]

[2. 1. 5. 3.]

[8. 3. 3. 4.]

[9. 4. 5. 5.]

[9. 1. 7. 9.]]

Сумма элементов всей матрицы: 101.0

Новая матрица:

[[9. 6. 5. 3. ]

[2. 1. 5. 3. ]

[8. 3. 3. 4. ]

[9. 4. 5. 5. ]

[9. 1. 7. 9. ]

[0.36633663 0.14851485 0.24752475 0.23762376]]

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

A = np.random.randint(0, 10, (N, M)).astype(np.float64)

print(str(A) + **"\n"**)

*# Определение суммы элементов всей матрицы*

Sum = A.sum()

print(**"Сумма элементов всей матрицы: "** + str(Sum) + **"\n"**)

*#Определение суммы всех элентов кождого столбца матрицы*

Sum\_column = A.sum(axis=1)

*#Определение доли суммы*

X = []

**for** i **in** range(0, N):

n = Sum\_column[i] / Sum

X.append(n)

*#Добавление строки*

X = np.array(X)[: , np.newaxis]

A = np.hstack((A, X))

print(**"Новая матрица: \n"** + str(A))

Результа:

[[2. 8. 7.]

[9. 8. 2.]

[8. 2. 0.]

[2. 6. 0.]

[4. 1. 8.]]

Сумма элементов всей матрицы: 67.0

Новая матрица:

[[2. 8. 7. 0.25373134]

[9. 8. 2. 0.28358209]

[8. 2. 0. 0.14925373]

[2. 6. 0. 0.11940299]

[4. 1. 8. 0.19402985]]

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Определение количества отрицательных элементов в строке*

X = []

**for** i **in** range(0, N):

K = 0

**for** j **in** range(0, M):

**if** A[i,j] < 0:

K += 1

X.append(K)

*# Определение количества отрицательных элементов в столбце*

Y = []

**for** i **in** range(0, M):

K = 0

**for** j **in** range(0, N):

**if** A[j,i] < 0:

K += 1

Y.append(K)

*# Добавление столбца и строки к матрице A*

X = np.array(X)[: , np.newaxis]

A = np.hstack((A, X))

Y = np.hstack((Y, [0.]))

A = np.vstack((A, Y))

print(**"\nНовая матрица: \n"** + str(A))

Результат:

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

[ 3 6 -7 2 -6 -8 3 -3]

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

Новая матрица:

[[-3. -2. 6. -7. 7. -8. -5. 3. 5.]

[ 3. 6. -7. 2. -6. -8. 3. -3. 4.]

[ 6. -8. 2. 8. -4. 3. -7. -3. 4.]

[ 1. 2. 1. 1. 2. 2. 2. 2. 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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*#Верхние L строки и левые К столбцы матрицы*

L = random.randint(1, N)

K = random.randint(1, M)

print(**"L = "** + str(L) + **"\nK = "** + str(K) + **"\n"**)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Определение нулевых элементов в верхних L строках*

n = 0

**for** i **in** range(0, L):

**for** j **in** range(0, M):

**if** A[i,j] == 0:

n += 1

print(**"Количество нулевых элементов в верхних L строках: "** + str(n))

*# Определение нулевых элементов в левых K солбцах*

m = 0

**for** i **in** range(0, K):

**for** j **in** range(0, N):

**if** A[j,i] == 0:

m += 1

print(**"Количество нулевых элементов в левых K солбцах: "** + str(m))

Результат:

[[0 5]

[6 0]

[4 7]

[3 7]

[5 6]

[5 6]]

Количество нулевых элементов в верхних L строках: 2

Количество нулевых элементов в левых K солбцах: 1

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(2, 10)

*# К-ый столбец матрицы*

K = random.randint(1, M)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(2, 10)

M = random.randint(2, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Определение K-ого столбца*

K\_arr = np.array(A[:, K-1])

K\_arr = K\_arr[: , np.newaxis]

print(**"K-ый столбец: \n"** + str(K\_arr))

*# Перемножение элементов*

A = A \* K\_arr

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[5 6 6 1 1 5 6 3 7]

[1 8 7 0 8 9 1 7 9]

[0 2 6 5 2 6 1 8 8]]

K-ый столбец:

[[6]

[7]

[6]]

Новая матрица:

[[30 36 36 6 6 30 36 18 42]

[ 7 56 49 0 56 63 7 49 63]

[ 0 12 36 30 12 36 6 48 48]]

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

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(2, 10)

*# К-ый столбец матрицы*

L = random.randint(1, N)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(2, 10)

M = random.randint(2, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Определение L строки*

L\_arr = np.array(A[L-1, :])

print(**"L страка: \n"** + str(L\_arr))

*# Сложение элеменотов*

A = A + L\_arr

Результат:

[[8 7]

[8 8]

[0 5]]

L страка:

[8 7]

Новая матрица:

[[16 14]

[16 15]

[ 8 12]]

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

A = np.random.randint(0, 10, (N, M)).astype(np.float64)

print(str(A) + **"\n"**)

*# Нахождение наибольшее значение для каждой строки матрицы*

Max = A.max(axis=1)

Max = np.array(Max)[: , np.newaxis]

*# Деление элеменотов*

A = A / Max

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[5. 0. 7. 7. 1. 1.]

[4. 9. 7. 8. 8. 4.]

[5. 5. 9. 9. 3. 9.]

[4. 4. 3. 4. 7. 3.]

[6. 2. 9. 6. 1. 5.]

[0. 3. 8. 1. 9. 2.]

[8. 3. 0. 6. 6. 1.]

[0. 2. 8. 8. 1. 3.]

[9. 7. 6. 8. 5. 8.]

[8. 4. 3. 7. 9. 0.]]

Новая матрица:

[[0.71428571 0. 1. 1. 0.14285714 0.14285714]

[0.44444444 1. 0.77777778 0.88888889 0.88888889 0.44444444]

[0.55555556 0.55555556 1. 1. 0.33333333 1. ]

[0.57142857 0.57142857 0.42857143 0.57142857 1. 0.42857143]

[0.66666667 0.22222222 1. 0.66666667 0.11111111 0.55555556]

[0. 0.33333333 0.88888889 0.11111111 1. 0.22222222]

[1. 0.375 0. 0.75 0.75 0.125 ]

[0. 0.25 1. 1. 0.125 0.375 ]

[1. 0.77777778 0.66666667 0.88888889 0.55555556 0.88888889]

[0.88888889 0.44444444 0.33333333 0.77777778 1. 0. ]]

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 значениемии

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

A = np.random.randint(0, 10, (N, M)).astype(np.float64)

print(str(A) + **"\n"**)

*# Нахождение наибольшее значение для каждого столбца матрицы*

Max = A.max(axis=0)

*# Деление элеменотов*

A = A / Max

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[3. 4.]

[8. 9.]

[8. 9.]

[2. 0.]]

Новая матрица:

[[0.375 0.44444444]

[1. 1. ]

[1. 1. ]

[0.25 0. ]]

14. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Разделить элементы матрицы на элемент матрицы с наибольшим значением.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

A = np.random.randint(0, 10, (N, M)).astype(np.float64)

print(str(A) + **"\n"**)

*# Нахождение наибольшее значение матрицы*

Max = A.max()

print(Max)

*# Деление элеменотов*

A = A / Max

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[7.]

[2.]

[5.]

[3.]

[4.]

[0.]

[1.]

[7.]

[7.]

[4.]]

7.0

Новая матрица:

[[1. ]

[0.28571429]

[0.71428571]

[0.42857143]

[0.57142857]

[0. ]

[0.14285714]

[1. ]

[1. ]

[0.57142857]]

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число Н*

N = random.randint(2, 10)

M = random.randint(1, 10)

H = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Определение количества Н элементов в столбце*

print(**"Н элемент = "** + str(H))

**for** i **in** range(0, M):

n = 0

**for** j **in** range(0, N):

**if** A[j,i] == H:

n += 1

**if** n > 0:

print(**"В столбце "** + str(i+1) + **" - "** + str(n) + **" Н элементов"**)

**elif** n == 0:

print(**"В столбце "** + str(i + 1) + **" - нет Н элементов"**)

Результат:

[[8 9 2 6 5 4]

[3 0 7 5 3 7]

[7 3 0 4 0 3]

[4 2 5 3 0 5]

[3 7 8 6 0 5]]

Н элемент = 8

В столбце 1 - 1 Н элементов

В столбце 2 - нет Н элементов

В столбце 3 - 1 Н элементов

В столбце 4 - нет Н элементов

В столбце 5 - нет Н элементов

В столбце 6 - нет Н элементов

16. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Исключить из матрицы строку с номером L. Сомкнуть строки матрицы.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число L*

N = random.randint(2, 10)

M = random.randint(1, 10)

L = random.randint(1, N)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Удаление строки L*

print(**"L = "** + str(L))

A = np.delete(A, (L-1), axis=0)

print(**"\nНовая матрица: \n"** + str(A))

Результат:

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

[3 8 8 9 3 3 5 2 2 9]

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

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

[9 7 2 0 1 2 9 2 5 3]

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

L = 2

Новая матрица:

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

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

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

[9 7 2 0 1 2 9 2 5 3]

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

17. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Добавить к матрице строку и вставить ее под номером L.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число L*

N = random.randint(2, 10)

M = random.randint(1, 10)

L = random.randint(1, N)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Создание дополнительной строки*

a = np.random.randint(0, 10, M)

print(**"Допонлительная строка: "** + str(a))

*# Добавление строки под номером L*

print(**"\n L = "** + str(L))

**if** L == N:

X = np.vstack((A, a))

**elif** L != N:

X = np.array(A[0, :])

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

**if** i == L:

X = np.vstack((X, a))

X = np.vstack((X, A[i, :]))

**elif** i != L:

X = np.vstack((X, A[i, :]))

print(**"\nНовая матрица: \n"** + str(X))

Результат:

[[2 8 0 6 5 6 2 9]

[9 4 0 0 6 0 6 4]

[2 2 1 8 2 5 4 4]

[6 6 2 8 7 0 7 2]

[0 6 3 2 0 1 8 7]

[7 3 4 6 8 3 1 1]

[8 3 3 3 6 9 1 7]

[7 5 1 6 1 6 8 6]

[2 0 0 7 0 0 9 5]]

Допонлительная строка: [7 6 4 9 3 0 3 4]

L = 1

Новая матрица:

[[2 8 0 6 5 6 2 9]

[7 6 4 9 3 0 3 4]

[9 4 0 0 6 0 6 4]

[2 2 1 8 2 5 4 4]

[6 6 2 8 7 0 7 2]

[0 6 3 2 0 1 8 7]

[7 3 4 6 8 3 1 1]

[8 3 3 3 6 9 1 7]

[7 5 1 6 1 6 8 6]

[2 0 0 7 0 0 9 5]]

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]).

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(2, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(2, 10)

M = random.randint(2, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Нахождение главной диагонали и её суммы*

a = np.diagonal(A)

a\_sum = a.sum()

print(**"Главная диагональ: \n"** + str(a) + **"\n Её сумма = "** + str(a\_sum))

*# Нахождение побочной диагонали и её суммы*

b = np.fliplr(A).diagonal(0)

b\_sum = b.sum()

print(**"Побочная диагональ: \n"** + str(b) + **"\n Её сумма = "** + str(b\_sum))

Результа:

[[8 5 4 4 6 8 9 2 2 5]

[9 5 6 1 5 3 3 7 5 2]

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

[6 0 1 4 5 2 0 0 8 0]

[2 5 6 0 9 9 0 4 6 8]]

Главная диагональ:

[8 5 2 4 9]

Её сумма = 28

Побочная диагональ:

[5 5 8 0 9]

Её сумма = 27

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].

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Нахождение элементов расположенных параллельно главной диагонали*

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))

Результат:

[[1 9 3 5 5 8]

[1 5 0 3 6 0]

[9 8 9 2 2 4]

[0 7 3 6 3 9]

[1 6 4 4 5 6]

[4 8 6 7 5 0]]

Элементы которые выше главной диагонали:

[9 0 2 3 6]

Их сумма = 20

Элементы которые ниже главной диагонали:

[1 8 3 4 5]

Их сумма = 20

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].

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Нахождение элементов расположенных параллельно побочной диагонали*

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))

Результат:

[[0 0 9 8 6 7 2]

[4 8 7 8 2 9 7]

[8 0 5 0 3 6 6]

[9 3 3 3 8 7 0]

[4 2 5 5 2 8 4]

[9 8 1 0 7 4 1]

[9 5 1 9 7 1 9]]

Элементы которые выше побочной диагонали:

[7 2 0 3 2 9]

Их сумма = 0

Элементы которые ниже побочной диагонали:

[7 6 8 5 1 5]

Их сумма = 8400

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]).

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Создание симметричной матрицы*

B = (A + A.T)/2

print(**"Новая матрица: \n"** + str(B))

Результат:

[[6 5 7 9 4]

[8 5 9 8 9]

[5 8 8 8 9]

[4 8 8 7 2]

[7 6 1 7 2]]

Новая матрица:

[[6. 6.5 6. 6.5 5.5]

[6.5 5. 8.5 8. 7.5]

[6. 8.5 8. 8. 5. ]

[6.5 8. 8. 7. 4.5]

[5.5 7.5 5. 4.5 2. ]]

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Подсчет едениц в каждой строке*

X = []

**for** i **in** range(0, N):

n = 0

**for** j **in** range(0, M):

**if** A[i,j] == 1:

n += 1

**if** n%2 != 0:

X.append(1)

**elif** n%2 == 0:

X.append(0)

*# Добавление столбца*

X = np.array(X)[: , np.newaxis]

A = np.hstack((A, X))

print(A)

Результат:

[[1 0 0 0 1 0 1]

[1 1 1 0 0 0 0]

[1 1 0 1 0 0 1]

[0 1 1 0 0 1 1]]

[[1 0 0 0 1 0 1 1]

[1 1 1 0 0 0 0 1]

[1 1 0 1 0 0 1 0]

[0 1 1 0 0 1 1 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]

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 5)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Нахождение главной диагонали и её суммы*

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

a = A[iu]

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

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

*# Нахождение побочной диагонали и её суммы*

b = np.fliplr(A)[iu]

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

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

Резудьтат:

[[7 9 4 6 4]

[2 2 8 2 1]

[5 3 1 4 8]

[4 6 6 7 9]

[2 5 4 2 2]]

Cумма элементов выше главной диагонали = 55

Произведение элементов выше побочной диагонали = 2903040

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число L*

N = random.randint(2, 10)

M = random.randint(1, 10)

L = random.randint(1, N)

K = random.randint(1, M)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

print(**"Число N = "** + str(N))

print(**"Число M = "** + str(M))

print(**"Число L = "** + str(L))

print(**"Число K = "** + str(K))

*# Разделение матрицы на 4 части и нахлждение их суммы*

x = A[0:L ,0:K]

x\_sum = x.sum()

print(**"Вверхняя левая часть: сумма равна = "** + str(x\_sum) + **"\n"** + str(x))

y = A[L: ,0 :K]

y\_sum = y.sum()

print(**"\nНижняя левая часть: сумма равна = "** + str(y\_sum) + **"\n"** + str(y))

z = A[0:L ,K:]

z\_sum = z.sum()

print(**"\nВверхняя правая часть: сумма равна = "** + str(z\_sum) + **"\n"** + str(z))

a = A[L: ,K:]

a\_sum = a.sum()

print(**"\nНижняя правая часть: сумма равна = "** + str(a\_sum) + **"\n"** + str(a))

Результат:

[[4 9 2 8 8 8]

[2 9 7 7 7 7]

[5 4 9 1 7 6]

[7 2 5 7 2 0]]

Число N = 4

Число M = 6

Число L = 2

Число K = 1

Вверхняя левая часть: сумма равна = 6

[[4]

[2]]

Нижняя левая часть: сумма равна = 12

[[5]

[7]]

Вверхняя правая часть: сумма равна = 72

[[9 2 8 8 8]

[9 7 7 7 7]]

Нижняя правая часть: сумма равна = 43

[[4 9 1 7 6]

[2 5 7 2 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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов*

N = random.randint(2, 10)

M = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

A = np.random.randint(-1, 3, (N, M))

print(str(A) + **"\n"**)

*# Определение количества нулевых элементов в строке*

X = []

**for** i **in** range(0, N):

K = 0

**for** j **in** range(0, M):

**if** A[i,j] == 0:

K += 1

X.append(K)

*# Определение количества нулевых элементов в столбце*

Y = []

**for** i **in** range(0, M):

K = 0

**for** j **in** range(0, N):

**if** A[j,i] == 0:

K += 1

Y.append(K)

*# Добавление столбца и строки к матрице A*

X = np.array(X)[: , np.newaxis]

A = np.hstack((A, X))

Y = np.hstack((Y, [0.]))

A = np.vstack((A, Y))

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[-1 2 -1 1 -1]

[ 1 0 -1 1 2]

[ 1 -1 1 0 1]

[ 1 -1 -1 -1 0]]

Новая матрица:

[[-1. 2. -1. 1. -1. 0.]

[ 1. 0. -1. 1. 2. 1.]

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

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

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

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число L*

N = random.randint(2, 10)

M = random.randint(1, 10)

L = random.randint(1, N)

K = random.randint(1, M)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M **or** L == N **or** K == M :

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

print(**"Число N = "** + str(N))

print(**"Число M = "** + str(M))

print(**"Число L = "** + str(L))

print(**"Число K = "** + str(K))

*# Разделение матрицы на 4 части и нахождение среднее арифметическое*

x = A[0:L ,0:K]

x\_sum = x.mean()

print(**"Вверхняя левая часть: среднее арифметическое = "** + str(x\_sum) + **"\n"** + str(x))

y = A[L: ,0 :K]

y\_sum = y.mean()

print(**"\nНижняя левая часть: среднее арифметическое = "** + str(y\_sum) + **"\n"** + str(y))

z = A[0:L ,K:]

z\_sum = z.mean()

print(**"\nВверхняя правая часть: среднее арифметическое = "** + str(z\_sum) + **"\n"** + str(z))

a = A[L: ,K:]

a\_sum = a.mean()

print(**"\nНижняя правая часть: среднее арифметическое = "** + str(a\_sum) + **"\n"** + str(a))

Результат:

[[4 4 1 8 8 9 0 1 4 7]

[9 0 0 2 2 3 0 5 1 9]

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

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

[4 4 7 7 4 4 5 0 6 0]

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

[3 0 2 9 4 7 5 9 3 0]

[4 5 0 8 0 5 9 4 5 4]]

Число N = 8

Число M = 10

Число L = 3

Число K = 1

Вверхняя левая часть: среднее арифметическое = 6.666666666666667

[[4]

[9]

[7]]

Нижняя левая часть: среднее арифметическое = 4.6

[[3]

[4]

[9]

[3]

[4]]

Вверхняя правая часть: среднее арифметическое = 3.5925925925925926

[[4 1 8 8 9 0 1 4 7]

[0 0 2 2 3 0 5 1 9]

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

Нижняя правая часть: среднее арифметическое = 3.7555555555555555

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

[4 7 7 4 4 5 0 6 0]

[0 7 5 0 1 6 5 2 1]

[0 2 9 4 7 5 9 3 0]

[5 0 8 0 5 9 4 5 4]]

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число Н*

N = random.randint(2, 10)

M = random.randint(1, 10)

H = random.randint(1, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Определение количества Н элементов в строке*

print(**"Н элемент = "** + str(H))

**for** i **in** range(0, N):

n = 0

**for** j **in** range(0, M):

**if** A[i, j] == H:

n += 1

**if** n > 0:

print(**"В строке "** + str(i+1) + **" - "** + str(n) + **" Н элементов"**)

**elif** n == 0:

print(**"В строке "** + str(i + 1) + **" - нет Н элементов"**)

Результат:

[[4 1 5 5 1 3 7 6 0 9]

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

Н элемент = 6

В строке 1 - 1 Н элементов

В строке 2 - 1 Н элементов

28. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Исключить из матрицы столбец с номером K. Сомкнуть столбцы матрицы.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число K*

N = random.randint(2, 10)

M = random.randint(1, 10)

K = random.randint(1, M)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Удаление столбца K*

print(**"K = "** + str(K))

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

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[5 2 6 8 4 8 1 4]

[9 7 4 6 6 2 4 1]

[3 0 7 2 8 1 4 1]

[9 8 7 8 9 4 1 4]

[3 7 3 3 3 6 0 5]

[3 6 1 2 4 2 2 5]

[7 9 7 4 2 0 0 1]]

K = 7

Новая матрица:

[[5 2 6 8 4 8 4]

[9 7 4 6 6 2 1]

[3 0 7 2 8 1 1]

[9 8 7 8 9 4 4]

[3 7 3 3 3 6 5]

[3 6 1 2 4 2 5]

[7 9 7 4 2 0 1]]

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число L*

N = random.randint(2, 10)

M = random.randint(2, 10)

K = random.randint(1, M)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(1, 10)

M = random.randint(1, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Создание дополнительной столбца*

a = np.random.randint(0, 10, N)

a = a[: , np.newaxis]

print(**"Допонлительный столбец: \n"** + str(a))

*# Добавление столбца под номером K*

print(**"\n K = "** + str(K))

**if** K == M:

X = np.hstack((A, a))

**elif** K != M:

X = np.array(A[:, 0])

X = X[: , np.newaxis]

**for** i **in** range (1, M):

**if** i == K:

X = np.hstack((X, a))

Y = A[:, i]

Y = Y[: , np.newaxis]

X = np.hstack((X, Y))

**elif** i != K:

Y = A[:, i]

Y = Y[:, np.newaxis]

X = np.hstack((X, Y))

print(**"\nНовая матрица: \n"** + str(X))

Результат:

[[0 4 2 8 6 0 8 6 0]

[4 4 2 6 9 0 1 6 9]

[0 9 5 0 1 3 1 9 3]

[8 1 0 0 1 5 0 0 7]

[3 1 1 2 1 9 9 0 6]

[0 7 1 5 3 7 8 7 5]

[6 5 3 6 3 1 3 6 0]]

Допонлительный столбец:

[[7]

[4]

[0]

[6]

[8]

[0]

[2]]

K = 2

Новая матрица:

[[0 4 7 2 8 6 0 8 6 0]

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

[0 9 0 5 0 1 3 1 9 3]

[8 1 6 0 0 1 5 0 0 7]

[3 1 8 1 2 1 9 9 0 6]

[0 7 0 1 5 3 7 8 7 5]

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

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число Н*

N = random.randint(2, 10)

M = random.randint(2, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(2, 10)

M = random.randint(2, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Определение cуммы положительных и отрицательных элементов*

X = []

**for** i **in** range(0, N):

n = 0

m = 0

k = n - m

**for** j **in** range(0, M):

**if** A[i,j] < 0:

n -= A[i,j]

**elif** A[i,j] >= 0:

m += A[i, j]

print(**"Сумма положительных элементов "** + str(i+1) + **" строки = "** + str(m))

print(**"Модуль суммы отрицательных элементов "** + str(i+1) + **" строки = "** + str(n))

X.append(n-m)

X = np.array(X)[: , np.newaxis]

A = np.hstack((A, X))

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[ 4 0 -1 -1 -4 -3 1]

[ 6 4 -7 1 4 -8 -10]

[-10 7 6 -7 -7 -7 6]]

Сумма положительных элементов 1 строки = 5

Модуль суммы отрицательных элементов 1 строки = 9

Сумма положительных элементов 2 строки = 15

Модуль суммы отрицательных элементов 2 строки = 25

Сумма положительных элементов 3 строки = 19

Модуль суммы отрицательных элементов 3 строки = 31

Новая матрица:

[[ 4 0 -1 -1 -4 -3 1 4]

[ 6 4 -7 1 4 -8 -10 10]

[-10 7 6 -7 -7 -7 6 12]]

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.

*# Подключение библиотеки Numpy и Random*

**import** numpy **as** np

**import** random

*# Число строк и столбцов, число Н*

N = random.randint(2, 10)

M = random.randint(2, 10)

*# Так как матрица должна быть прямоугольной, то N не может быть равно M*

**while** N == M:

N = random.randint(2, 10)

M = random.randint(2, 10)

*# Создание матрицы*

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

print(str(A) + **"\n"**)

*# Определение cуммы положительных и отрицательных элементов*

X = []

**for** i **in** range(0, M):

n = 0

m = 0

k = n - m

**for** j **in** range(0, N):

**if** A[j,i] < 0:

n -= A[j,i]

**elif** A[j,i] >= 0:

m += A[j,i]

print(**"Сумма положительных элементов "** + str(i+1) + **" столбца = "** + str(m))

print(**"Модуль суммы отрицательных элементов "** + str(i+1) + **" столбца = "** + str(n))

X.append(n-m)

X = np.array(X)

A = np.vstack((A, X))

print(**"\nНовая матрица: \n"** + str(A))

Результат:

[[-7 -3 -6 -6 7]

[ 8 -6 -9 -2 2]

[ 1 -1 6 9 7]]

Сумма положительных элементов 1 столбца = 9

Модуль суммы отрицательных элементов 1 столбца = 7

Сумма положительных элементов 2 столбца = 0

Модуль суммы отрицательных элементов 2 столбца = 10

Сумма положительных элементов 3 столбца = 6

Модуль суммы отрицательных элементов 3 столбца = 15

Сумма положительных элементов 4 столбца = 9

Модуль суммы отрицательных элементов 4 столбца = 8

Сумма положительных элементов 5 столбца = 16

Модуль суммы отрицательных элементов 5 столбца = 0

Новая матрица:

[[ -7 -3 -6 -6 7]

[ 8 -6 -9 -2 2]

[ 1 -1 6 9 7]

[ -2 10 9 -1 -16]]

Вывод: Выполнив 31 задание, можносделать вывод, что модуль Numpy очень быстро и удобно помогает в решении задач с массивами. Модуль имеет очень полезные функции, которые сильно помогают оптимизировать код, облегчить его написание.

**ГЛАВА 3 РЕШЕНИЕ СЛАУ МЕТОДОМ ГАУССА**

Метод Гаусса — классический метод решения системы линейных алгебраических уравнений (СЛАУ).

Метод Гаусса включает в себя прямой (приведение расширенной матрицы к ступенчатому виду, то есть получение нулей под главной диагональю) и обратный (получение нулей над главной диагональю расширенной матрицы) ходы. Прямой ход и называется методом Гаусса, обратный - методом Гаусса-Жордана, который отличается от первого только последовательностью исключения переменных.

Метод Гаусса идеально подходит для решения систем содержащих больше трех линейных уравнений, для решения систем уравнений, которые не являются квадратными (чего не скажешь про метод Крамера иматричный метод). То есть метод Гаусса - наиболее универсальный метод для нахождения решения любой системы линейных уравнений, он работает в случае, когда система имеет бесконечно много решений или несовместна.

Структура программы:

1. Было создан файл формата csv, чтобы записать в него исходные матрицы, файл выглядит следующим образом:
2. Был оптимизирован код, таким образом, чтобы чтение происходило из файла с исходными матрицами.
3. Был оптимизирован код, таким образом, чтобы результат полученный входе работы кода был записан в новый файл формата csv.

Код программы:

**import** numpy

**def** FancyPrint(A, B, selected):

**for** row **in** range(len(B)):

print(**"("**, end=**''**)

**for** col **in** range(len(A[row])):

print(**"\t{1:10.2f}{0}"**.format(**" " if** (selected **is None**

**or** selected != (row, col)) **else "\*"**, A[row][col]), end=**''**)

print(**"\t) \* (\tX{0}) = (\t{1:10.2f})"**.format(row + 1,B[row]))

data = numpy.genfromtxt(**'./numpy-gauss.csv'**, delimiter=**';'**)

print(data)

matrix\_list = []

matrix = []

**for** row **in** data:

first\_col = row[0]

**if** numpy.isnan(first\_col):

matrix\_list.append(matrix)

matrix = []

**continue**

mask = ~numpy.isnan(row)

matrix.append(row[mask])

matrix\_list.append(matrix)

f = open(**'numpy-gauss-slv.csv'**, **'wb+'**)

f.truncate()

**for** matrix **in** matrix\_list:

M = numpy.array(matrix)

myA = numpy.delete(M, M.shape[1] - 1, axis=1)

myB = M[:, [-1]].flatten()

print(**"Исходная система:"**)

FancyPrint(myA, myB, **None**)

slv = numpy.linalg.solve(myA, myB)

print(**"Решаем:"**)

print(slv)

numpy.savetxt(f, numpy.array([slv]), delimiter=**','**)

f.close()

**ЗАКЛЮЧЕНИЕ**

Благодаря простоте и гибкости языка Python, его можно рекомендовать пользователям (математикам, физикам, экономистам и т.д.) не являющимся программистами, но использующими вычислительную технику и программирование в своей работе.

Программы на Python разрабатываются в среднем в полтора-два (а порой и в два-три) раза быстрее нежели на компилируемых языках (С, С++, Pascal). Поэтому, язык может представлять немалый интерес и для профессиональных программистов, разрабатывающих приложения, не критичные к скорости выполнения, а также программы, использующие сложные структуры данных. В частности, Python хорошо зарекомендовал себя при разработке программ работы с графами, генерации деревьев.

**БИБЛИОГРАФИЧЕСКИЙ СПИСОК**

1. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.
2. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с.
3. Лутц М. Программирование на Python, том I, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.
4. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.
5. Прохоренок Н.А. Python 3 и PyQt. Разработка приложений. – СПб.: БХВ-Петербург, 2012. – 704 с.
6. Вабищевич П. Н. Численные методы. Вычислительный практикум. – – 320 c.
7. Пилгрим Марк. Погружение в Python 3 (Dive into Python 3 на русском)
8. Прохоренок Н.А. Самое необходимое. — СПб.: БХВ-Петербург, 2011. — 416 с.
9. Хахаев И.А. Практикум по алгоритмизации и программированию на Python. – М.: Альт Линукс, 2010. — 126 с. (Библиотека ALT Linux).
10. Чаплыгин А.Н. Учимся программировать вместе с питоном.
11. Шапошникова С. Основы программирования на Python. Вводный курс.
12. Briggs J. R. — Python for Kids — 2012
13. Deitel H.M. et al. Python — How to Program
14. Allen Downey – ThinkPython+Kart[Python\_3.2]