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 – 38.03.05.2023.301. ПЗ КР

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

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

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020г. «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020г.

Нормоконтролёр, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Автор, студент группы ЭУ-120

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Высоцкая Е.О. /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020г.

Работа защищена с оценкой

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

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020г.

Челябинск 2020

АННОТАЦИЯ

Высоцкая Е.О.

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-120, 2020

В данной работе рассматриваются основные методы сортировки, их особенности, принципы работы и их эффективность на практике. Помимо этого, рассматривается библиотека NumPy, а именно: что она из себя представляет и ее применение на практике. С помощью библиотеки NumPy было решено 31 задача и система линейных уравнений методом Гаусса

Оглавление

[ВВЕДЕНИЕ 4](#_Toc41337420)

[ГЛАВА 1. СОРТИРОВКА ОДНОМЕРНОГО МАССИВА 6](#_Toc41337421)

[1.1 Алгоритмы сортировки 6](#_Toc41337422)

[1.1.1 Сортировка выбором (select) 6](#_Toc41337423)

[1.1.2 Сортировка включениями (insert) 7](#_Toc41337424)

[1.1.3 Обменная сортировка (bubble) 8](#_Toc41337425)

[1.2 Анализ эффективности 9](#_Toc41337426)

[ГЛАВА 2. МАТРИЧНАЯ МАТЕМАТИКА И РАБОТА С БИБЛИОТЕКАМИ 13](#_Toc41337427)

[2.1 Примеры использования библиотек NumPy и MatPlotLib 13](#_Toc41337429)

[2.2 Библиотека NumPy и примеры решения задач 13](#_Toc41337430)

[ГЛАВА 3 РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ 44](#_Toc41337431)

[ЗАКЛЮЧЕНИЕ 49](#_Toc41337432)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 50](#_Toc41337433)

# ВВЕДЕНИЕ

Мы живём в цифровую эпоху, где информационные технологии стремительно развиваются. В наше время спрос на изучение программирования быстро растёт. Разработка делится на небольшие специальности, включая системное программирование, разработку игр, проектирование мобильных приложений и веб-программирование. В своей работе я хочу рассказать об одном из языков программирования – Python.

Python – это 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.

Являясь одним из ведущих языков программирования, он имеет много фреймворков (платформ для построения приложений) и библиотек, которыми можно воспользоваться. Библиотека языка программирования — это просто набор модулей и функций, которые облегчают некоторые специфические операции с использованием этого языка программирования. Следовательно в своей работе я рассмотрела одну из библиотек.

NumPy — это расширение языка Python, добавляющее поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых математических функций для операций с этими массивами.

Таким образом, в первой части работы я проанализировала три данных мне алгоритма сортировки: select, insert и bubble на python. Безусловно, помимо них существует множество других алгоритмов, но моя задача была выбрать наиболее быстрый и удобный из представленных трех.

Во второй части работы я рассмотрела библиотеку NumPy. Любая библиотека содержит в себе так называемый набор шаблонов для решения той или иной задачи, что значительно упрощает и ускоряет работу, в нашем случае с линейной алгеброй. Естественно, просто читая документацию к данной библиотеке невозможно понять ее возможности, поэтому «набьем руку», решив 31 задачу с применением нашей библиотеки.

А в конце мне предстояло решить системы линейных уравнений методом Гаусса. Использование библиотеки NumPy значительно ускоряет и упрощает эту задачу, в чем я и убедилась.

# ГЛАВА 1. СОРТИРОВКА ОДНОМЕРНОГО МАССИВА

## Алгоритмы сортировки

«… Сортировка к тому же, еще и сама достаточно хороший пример задачи, которую можно решать с помощью многих различных алгоритмов. Каждый из них имеет и свои достоинства, и свои недостатки, и выбирать алгоритмы нужно исходя из конкретной постановки задачи.

В общем, под сортировкой мы будем понимать процесс перегруппировки заданного множества объектов в некотором определенном порядке. Цель сортировки – облегчить последующий поиск элементов в таком отсортированном множестве. Это почти универсальная, фундаментальная деятельность. Мы встречаемся с отсортированными объектами в телефонных книгах, в списках подоходных налогов, в оглавлениях книг, в библиотеках, в словарях, на складах – почти везде, где нужно искать хранимые объекты. Даже малышей учат держать свои вещи «в порядке», и они уже сталкиваются с некоторыми видами сортировок задолго до того, как познакомятся с азами арифметики».

Н. Вирт — Алгоритмы + данные = программы

### Сортировка выбором (select)

При сортировке массива Arr0, Arr2, ..., ArrN‑1 методом простого выбора среди всех элементов находится элемент с наименьшим значением Arri, и Arr0 и Arri обмениваются значениями. Затем этот процесс повторяется для получаемого подмассива Arr1, Arr2, ..., ArrN‑1, ... Arrj, Arrj+1, ..., ArrN‑1 до тех пор, пока мы не дойдем до подмассива ArrN‑1, содержащего к этому моменту наибольшее значение.

Для метода сортировки простым выбором оценка требуемого числа сравнений – N(N‑1)/2. Порядок требуемого числа пересылок, которые требуются для выбора минимального элемента, в худшем случае составляет O(N2). Однако порядок среднего числа пересылок есть O(N\*Lg(N)), что в ряде случаев делает этот метод предпочтительным.

Функция алгоритма:

def select(arr, dim):  
 alg\_count = [0, 0]  
 for k in range(0, dim - 1): # -1, т.к. последний элемент обменивать уже не надо  
 m = k # в m хранится минимальное значение  
 i = k + 1 # откуда начинать поиск минимума (элемент следующий за k)  
 for i in range(i, dim):  
 alg\_count[0] += 1  
 if arr[i] < arr[m]:  
 m = i  
 if k != m:  
 t = arr[k]  
 arr[k] = arr[m]  
 arr[m] = t  
 alg\_count[1] += 1  
 return alg\_count  
import random  
arry = [random.randint(0, 1000) for i in range(1000)]  
print(arry)  
select(arry, len(arry))  
print(arry)

### Сортировка включениями (insert)

Одним из наиболее простых и естественных методов внутренней сортировки является сортировка простыми включениями. Идея алгоритма очень проста. Пусть имеется массив ключей Arr0, Arr1, ..., ArrN‑1. Для каждого элемента массива, начиная со второго, производится сравнение с элементами с меньшим индексом. Элемент Arri последовательно сравнивается с элементами Arrj, где jЄ[i‑1;0], т.е. изменяется от i‑1 до 0. До тех пор, пока для очередного элемента Arrj выполняется соотношение Arrj > Arri, Arri и Arrj меняются местами. Если удается встретить такой элемент Arrj, что Arrj ≤ Arri, или если достигнута нижняя граница массива, производится переход к обработке элемента Arri+1. Так продолжается до тех пор, пока не будет достигнута верхняя граница массива.

Легко видеть, что в лучшем случае, когда массив уже упорядочен для выполнения алгоритма с массивом из N элементов потребуется N‑1 сравнение и 0 пересылок. В худшем случае, когда массив упорядочен в обратном порядке потребуется N(N‑1)/2 сравнений и столько же пересылок. Таким образом, можно оценивать сложность метода простых включений как O(N2).

Можно сократить число сравнений, применяемых в методе простых включений, если воспользоваться тем, что при обработке элемента Arri массива элементы Arr0, Arr1, ..., Arri‑1 уже упорядочены, и воспользоваться для поиска элемента, с которым должна быть произведена перестановка, методом двоичного деления. В этом случае оценка числа требуемых сравнений становится O(N\*Log(N)). Заметим, что поскольку при выполнении перестановки требуется сдвижка на один элемент нескольких элементов, то оценка числа пересылок остается O(N2). Алгоритм сортировки включением, оформленный в виде функции приведен ниже.

def insert(arr, dim):  
 alg\_count = [0, 0]  
  
 for i in range(1, dim):  
 temp = arr[i]  
 j = i - 1  
 while j >= 0:  
 alg\_count[0] += 1  
 if arr[j] > temp:  
 alg\_count[1] += 1  
 arr[j + 1] = arr[j]  
 arr[j] = temp  
 j -= 1  
 print(alg\_count)  
import random  
arry = [random.randint(0, 1000) for i in range(1000)]  
print(arry)  
insert(arry, len(arry))  
print(arry)

### Обменная сортировка (bubble)

Простая обменная сортировка, называемая «методом пузырька», для массива Arr0, Arr2, ..., ArrN‑1 работает следующим образом. Начиная с конца массива сравниваются два соседних элемента ArrN‑1 и ArrN‑2. Если выполняется условие ArrN‑2 > ArrN‑1, то они меняются местами. Процесс продолжается для ArrN‑2 и ArrN‑3 и т.д., пока не будет произведено сравнение Arr1 и Arr0. Понятно, что после этого на месте Arr0 окажется элемент с наименьшим значением. На втором шаге процесс повторяется, но последними сравниваются Arr2 и Arr1. И так далее. На последнем шаге будут сравниваться только текущие значения ArrN‑1 и ArrN‑2. Понятна аналогия с пузырьком, поскольку наименьшие элементы, самые «легкие», постепенно «всплывают» к верхней границе массива.

Для метода обменной сортировки требуется число сравнений N(N‑1)/2, минимальное число пересылок 0, а среднее и максимальное число пересылок − O(N2).

Метод пузырька допускает три простых усовершенствования. Во-первых, если на некотором шаге не было произведено ни одного обмена, то выполнение алгоритма можно прекращать. Во-вторых, можно запоминать наименьшее значение индекса массива, для которого на текущем шаге выполнялись перестановки. Очевидно, что верхняя часть массива до элемента с этим индексом уже отсортирована, и на следующем шаге можно прекращать сравнения значений соседних элементов при достижении такого значения индекса. В-третьих, метод пузырька работает неравноправно для «легких» и «тяжелых» значений. Легкое значение попадает на нужное место за один шаг, а тяжелое на каждом шаге опускается по направлению к нужному месту на одну позицию.

def bubble(arr, dim):  
 alg\_count = [0, 0] # Массив для показателей эффективности  
 for i in range(dim - 1): # Основной цикл для всех элементов  
 for j in range(dim - i - 1): # Цикл для оставшейся части меньших элементов  
 alg\_count[0] += 1  
 if arr[j] > arr[j + 1]: # Сравниваем элементы  
 arr[j], arr[j + 1] = arr[j + 1], arr[j] # Меняем элементы местами  
 alg\_count[1] += 1  
 return alg\_count  
  
import random  
arry = [random.randint(0, 1000) for i in range(1000)]  
print(arry)  
bubble(arry, len(arry))  
print(arry)

## Анализ эффективности

На данном этапе мы создаем программу, с помощью которой замеряем количество сравнений и перестановок для каждого метода сортировки, которые мы применяем к упорядоченному, обратно упорядоченному, а затем проводим целую серию экспериментов со случайными массивами, где вычисляем средние значения сравнений и перестановок и записываем все результаты в отдельный файл.

import random  
import select\_2  
import bubble\_2  
import insert\_2  
  
DIM = 1000  
bubble\_arr = []  
insert\_arr = []  
select\_arr = []  
CTotal = [0, 0, 0]  
MTotal = [0, 0, 0]  
  
for i in range(1, DIM+1):  
 bubble\_arr.append(i)  
 insert\_arr.append(i)  
 select\_arr.append(i)  
  
file = open("Result.txt", "w")  
print("Упорядоченный массив: ")  
print(bubble\_arr)  
  
  
count = select\_2.select(select\_arr, DIM)  
print("Упорядоченный массив: результат")  
print(select\_arr)  
  
CTotal[0] = count[0]  
MTotal[0] = count[1]  
  
  
count = insert\_2.insert(insert\_arr, DIM)  
CTotal[1] = count[0]  
MTotal[1] = count[1]  
  
  
count = bubble\_2.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])  
  
file.write("Упорядоченный массив:\n ")  
file.write("Размер массива: " + str(DIM) + "\n")  
file.write("Сравнений: " + str(CTotal[0]) + " " + str(CTotal[1]) + " " + str(CTotal[2]) + "\n")  
file.write("Перестановок: " + str(MTotal[0]) + " " + str(MTotal[1]) + " " + str(MTotal[2]) + "\n")  
  
select\_arr.clear()  
bubble\_arr.clear()  
insert\_arr.clear()  
  
for i in range(DIM, 0, -1):  
 bubble\_arr.append(i)  
 insert\_arr.append(i)  
 select\_arr.append(i)  
  
print("Обратно упорядоченный массив: ")  
print(bubble\_arr)  
  
  
count = select\_2.select(select\_arr, DIM)  
print("Обратно упорядоченный массив: результат")  
print(select\_arr)  
  
CTotal[0] = count[0]  
MTotal[0] = count[1]  
  
  
count = insert\_2.insert(insert\_arr, DIM)  
CTotal[1] = count[0]  
MTotal[1] = count[1]  
  
  
count = bubble\_2.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])  
  
file.write("Обратно упорядоченный массив:\n ")  
file.write("Размер массива: " + str(DIM) + "\n")  
file.write("Сравнений: " + str(CTotal[0]) + " " + str(CTotal[1]) + " " + str(CTotal[2]) + "\n")  
file.write("Перестановок: " + str(MTotal[0]) + " " + str(MTotal[1]) + " " + str(MTotal[2]) + "\n")  
  
KOL = 2000  
CTotal = [0, 0, 0]  
MTotal = [0, 0, 0]  
  
for n in range(0, KOL):  
 select\_arr.clear()  
 insert\_arr.clear()  
 bubble\_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 = select\_2.select(select\_arr, DIM)  
 CTotal[0] += count[0]  
 MTotal[0] += count[1]  
  
  
 count = insert\_2.insert(insert\_arr, DIM)  
 CTotal[1] += count[0]  
 MTotal[1] += count[1]  
  
  
 count = bubble\_2.bubble(bubble\_arr, DIM)  
 CTotal[2] += count[0]  
 MTotal[2] += count[1]  
  
print("\nСлучайный массив:")  
print("Проведено экспериментов:", KOL)  
print("Размер массива:", DIM)  
print("Сравнений:", CTotal[0]/KOL, CTotal[1]/KOL, CTotal[2]/KOL)  
print("Перестановок:", MTotal[0]/KOL, MTotal[1]/KOL, MTotal[2]/KOL)  
  
file.write("\nСлучайный массив:\n")  
file.write("Проведено экспериментов: " + str(KOL) + "\n")  
file.write("Размер массива: " + str(DIM) + "\n")  
file.write("Сравнений: " + str(CTotal[0]/KOL) + " " + str(CTotal[1]/KOL) + " " + str(CTotal[2]/KOL) + "\n")  
file.write("Перестановок: " + str(MTotal[0]/KOL) + " " + str(MTotal[1]/KOL) + " " + str(MTotal[2]/KOL))  
  
file.close()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Массив | Показатель | 1000 элементов | | |
|  |  | Insert | Select | Bubble |
| Упорядоченный | C | 499500 | 499500 | 999 |
| M | 0 | 0 | 0 |
| Обратно упорядоченный | C | 499500 | 499500 | 499500 |
| M | 499500 | 250000 | 499500 |
| Случайный | Mean(C) | 499500 | 499500 | 496365,4 |
| Mean(M) | 249939 | 5265,02 | 249789,1 |

Результат исполнения программы:

Из этих цифр мы можем видеть, что наиболее эффективным и быстрым методом сортировки оказался метод select.

# ГЛАВА 2. МАТРИЧНАЯ МАТЕМАТИКА И РАБОТА С БИБЛИОТЕКАМИ



## Примеры использования библиотек NumPy и MatPlotLib

Библиотеки в программировании- это сборники подпрограмм или объектов, которые используются для разработки программного обеспечения или решения близких по тематике задач. То есть, библиотеки представляют собой собрание функций, в которых хранятся названия каждой функции, объектный код самой это функции и информация, касающаяся “перемещаемости файла”, необходимая для редактирования связей. Когда программа делает ссылку на функцию, содержащаяся в библиотеке, то компоновщик отыскивает эту функцию и добавляет ее код к программе. Таким образом, к программе добавляются только те функции, которые будут использоваться в ней. Помимо этого, хочется сказать о том, что функции, содержащиеся в библиотеках, очень сильно облегчают жизнь при написании программ. Ведь программистам не нужно будет писать целый цикл для решения определенного вопроса, потому что они могут просто вызвать определенную функцию, которая этот вопрос решит. Например, мне необходимо посчитать элементы в массиве. Для этого мне не нужно считать отдельно элементы, ведь я могу воспользоваться функцией “sum”, которая все сделает за меня.

## Библиотека NumPy и примеры решения задач

В процессе написания своей курсовой работы, я познакомился с такой интересной библиотекой, как NumPy. NumPy - это библиотека языка Python, которая предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций. Они объединяются в высокоуровневые пакеты, которые обеспечивают функционал, сравнимый с функционалом MatLab. То есть, NumPy предоставляет базовые методы для манипуляции с большими массивами и матрицами.

Теперь для «набивания руки» самостоятельно решаем 31 задачу с помощью библиотеки, которая значительно упрощает работу с двумерными массивами (матрицами).

1. # # Задача 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)  
   print(N, M)  
   A = np.random.randint(0, 100, (N, M))  
   print(A)  
   mx = 0  
   indmx = 0  
   for i in range(M):  
    if np.sum(A[:, i]) > mx:  
    mx = np.sum(A[:, i])  
    indmx = i  
   print("Столбец с максимальной суммой элементов - " + str(A[:, indmx]))  
   print("Сумма этого столбца - " + str(mx) + ", " + "номер этого столбца - " + str(indmx + 1))  
   stmax = 0  
   for b in A[:, indmx]:  
    if b > stmax:  
    stmax = b  
   print("Максимальный элемент этого столбца - " + str(stmax))

Результат:

3 7

[[60 4 83 52 89 61 45]

[29 2 8 80 37 58 18]

[ 2 95 81 91 49 48 80]]

Столбец с максимальной суммой элементов - [52 80 91]

Сумма этого столбца - 223, номер этого столбца - 4

Максимальный элемент этого столбца - 91

# Задача 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)  
  
print(N,M)  
A=np.random.randint(0,10, (N,M))  
  
print(A)  
  
mx=0  
aver=0  
for i in range (N):  
 if np.mean (A[i , :])> mx:  
 mx= np.mean(A[i , :])  
 aver=i  
print("Строка с максимальным средним значением элементов-"+ str(A[aver , :]))  
print("Среднее значение этой строки-"+str(mx)+ ","+ "номер этой строки-"+ str(aver+1))

Результат:

5 8

[[1 1 8 8 4 7 7 5]

[2 2 0 3 3 2 3 9]

[7 3 4 9 0 7 5 8]

[6 4 7 9 6 5 3 8]

[0 0 2 0 9 5 1 8]]

Строка с максимальным средним значением элементов-[6 4 7 9 6 5 3 8]

Среднее значение этой строки-6.0, номер этой строки-4

# 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)  
  
print(N,M)  
A=np.random.randint(0,10, (N,M))  
  
print(A)  
  
mx=0  
indmx=0  
for i in range (M):  
 if np.sum(A[:, i])>mx:  
 mx = np.sum(A[:, i])  
 indmx=i  
print("Столбец с максимальной суммой элементов-"+ str(A[: , indmx]))  
print("Сумма этого столбца-"+str(mx)+ ","+ "номер этого столбца-"+ str(indmx+1))  
stmin=10  
for b in A [: , indmx]:  
 if b< stmin:  
 stmin=b  
print("Минимальный элемент этого столбца-"+str(stmin))

Результат:

2 5

[[6 1 0 0 1]

[4 2 2 6 5]]

Столбец с максимальной суммой элементов-[6 4]

Сумма этого столбца-10,номер этого столбца-1

Минимальный элемент этого столбца-4

#4  
# описание: 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)  
  
print(N,M)  
A=np.random.randint(0,10, (N,M))  
  
print(A)  
  
mx=10  
aver=0  
for i in range (N):  
 if np.mean (A[i , :])< mx:  
 mx= np.mean(A[i , :])  
 aver=i  
print("Строка с минимальным средним значением элементов-"+ str(A[aver , :]))  
print("Среднее значение этой строки-"+str(mx)+ ","+ "номер этой строки-"+ str(aver+1))

Результат:

3 5

[[0 4 9 3 6]

[4 8 0 6 8]

[6 9 2 5 7]]

Строка с минимальным средним значением элементов-[0 4 9 3 6]

Среднее значение этой строки-4.4,номер этой строки-1

# 5  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
  
print(N,M)  
A=np.random.randint(0,10, (N,M))  
  
print(A)  
  
M\_mean = A.mean(axis=0)  
N\_mean = A.mean(axis=1)  
  
print("Матрица со средними значениями-",A)  
  
  
N\_mean = np.append(N\_mean, None)  
  
A = np.vstack((A, M\_mean))  
A = np.hstack((A, N\_mean.reshape(-1,1)))  
  
print("Матрица с N и M +1-",A)

Результат:

9 9

[[0 8 2 5 4 5 8 7 8]

[1 4 7 9 6 0 5 6 7]

[1 3 1 7 9 1 6 2 1]

[4 4 2 9 9 5 8 8 6]

[9 2 6 7 5 3 0 9 2]

[7 8 7 1 3 7 0 6 6]

[7 6 3 0 1 5 2 8 8]

[3 1 9 6 3 7 8 8 7]

[9 2 4 7 8 6 4 0 5]]

Матрица со средними значениями- [[0 8 2 5 4 5 8 7 8]

[1 4 7 9 6 0 5 6 7]

[1 3 1 7 9 1 6 2 1]

[4 4 2 9 9 5 8 8 6]

[9 2 6 7 5 3 0 9 2]

[7 8 7 1 3 7 0 6 6]

[7 6 3 0 1 5 2 8 8]

[3 1 9 6 3 7 8 8 7]

[9 2 4 7 8 6 4 0 5]]

Матрица с N и M +1- [[0.0 8.0 2.0 5.0 4.0 5.0 8.0 7.0 8.0 5.222222222222222]

[1.0 4.0 7.0 9.0 6.0 0.0 5.0 6.0 7.0 5.0]

[1.0 3.0 1.0 7.0 9.0 1.0 6.0 2.0 1.0 3.4444444444444446]

[4.0 4.0 2.0 9.0 9.0 5.0 8.0 8.0 6.0 6.111111111111111]

[9.0 2.0 6.0 7.0 5.0 3.0 0.0 9.0 2.0 4.777777777777778]

[7.0 8.0 7.0 1.0 3.0 7.0 0.0 6.0 6.0 5.0]

[7.0 6.0 3.0 0.0 1.0 5.0 2.0 8.0 8.0 4.444444444444445]

[3.0 1.0 9.0 6.0 3.0 7.0 8.0 8.0 7.0 5.777777777777778]

[9.0 2.0 4.0 7.0 8.0 6.0 4.0 0.0 5.0 5.0]

[4.555555555555555 4.222222222222222 4.555555555555555 5.666666666666667

5.333333333333333 4.333333333333333 4.555555555555555 6.0

5.555555555555555 None]]

# 6  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
  
print(N,M)  
A=np.random.randint(0,10, (N,M))  
  
print(A)  
  
B=np.sum(A)  
print ("Сумаа элементов матрицы A-", B)  
  
M\_sum = np.sum(A, axis=0)/np.sum(A)  
print("Доля элементов каждого столбца-"+str(np.sum(A, axis=0)/np.sum(A)))  
  
A = np.vstack((A,M\_sum))  
  
print("Матрица с N +1-",A)

Результат:

9 9

[[3 2 5 9 0 8 5 2 1]

[1 4 5 8 9 1 0 0 2]

[9 6 1 7 6 2 4 9 9]

[3 0 7 4 8 2 4 0 2]

[6 3 3 9 2 3 1 4 2]

[2 9 4 7 7 9 0 0 6]

[3 1 7 3 4 2 6 0 6]

[6 0 5 2 1 5 4 6 4]

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

Сумаа элементов матрицы A- 327

Доля элементов каждого столбца-[0.11314985 0.10397554 0.11314985 0.17737003 0.13761468 0.09785933

0.09174312 0.06727829 0.09785933]

Матрица с N +1- [[3. 2. 5. 9. 0. 8.

5. 2. 1. ]

[1. 4. 5. 8. 9. 1.

0. 0. 2. ]

[9. 6. 1. 7. 6. 2.

4. 9. 9. ]

[3. 0. 7. 4. 8. 2.

4. 0. 2. ]

[6. 3. 3. 9. 2. 3.

1. 4. 2. ]

[2. 9. 4. 7. 7. 9.

0. 0. 6. ]

[3. 1. 7. 3. 4. 2.

6. 0. 6. ]

[6. 0. 5. 2. 1. 5.

4. 6. 4. ]

[4. 9. 0. 9. 8. 0.

6. 1. 0. ]

[0.11314985 0.10397554 0.11314985 0.17737003 0.13761468 0.09785933

0.09174312 0.06727829 0.09785933]]

# 7  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
  
print(N,M)  
A=np.random.randint(0,10, (N,M))  
  
print(A)  
  
B=np.sum(A)  
print ("Сумаа элементов матрицы A-", B)  
  
N\_sum = np.sum(A, axis=1)/np.sum(A)  
print("Доля элементов каждой строки-"+str(np.sum(A, axis=1)/np.sum(A)))  
  
A = np.hstack((A, N\_sum.reshape(-1,1)))  
  
print("Матрица с M +1-",A)

Результат:

6 3

[[9 4 4]

[0 9 9]

[0 3 3]

[3 9 5]

[9 0 7]

[0 2 4]]

Сумаа элементов матрицы A- 80

Доля элементов каждой строки-[0.2125 0.225 0.075 0.2125 0.2 0.075 ]

Матрица с M +1- [[9. 4. 4. 0.2125]

[0. 9. 9. 0.225 ]

[0. 3. 3. 0.075 ]

[3. 9. 5. 0.2125]

[9. 0. 7. 0.2 ]

[0. 2. 4. 0.075 ]]

# 8  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
  
print(N,M)  
A=np.random.randint(-10,10, (N,M))  
  
print(A)  
M\_sum = (A < 0).sum(axis=0)  
N\_sum = (A < 0).sum(axis=1)  
  
print("Кол-во отрицательных элементов в матрице-"+str(M\_sum ), str(N\_sum))  
  
N\_sum = np.append(N\_sum, None)  
  
A = np.vstack((A, M\_sum))  
A = np.hstack((A, N\_sum.reshape(-1,1)))  
  
print("Матрица с N и M +1-",A)

Результат:

2 5

[[ -4 -2 -4 9 -7]

[ -9 -6 -6 -10 -2]]

Кол-во отрицательных элементов в матрице-[2 2 2 1 2] [4 5]

Матрица с N и M +1- [[-4 -2 -4 9 -7 4]

[-9 -6 -6 -10 -2 5]

[2 2 2 1 2 None]]

# 9  
# описание: 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  
  
N = np.random.randint(2,10)  
M = np.random.randint(2,10)  
L = np.random.randint(1,N)  
K = np.random.randint(1,M)  
  
print(N,M,L,K)  
A=np.random.randint(-10,10, (N,M))  
  
New\_A = A[:L, :K]  
  
print(A)  
  
print(np.sum(New\_A == 0))

Результат:

8 6 7 4

[[ 0 1 8 3 -5 4]

[ -8 0 4 9 -1 -1]

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

[ 6 9 -1 8 -9 0]

[ 4 -10 5 -9 5 -1]

[ -2 -3 -10 -5 5 -3]

[ -9 5 -10 -5 4 4]

[ -4 0 -2 -7 1 -8]]

2

# 10  
# описание: 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(2, 5)  
M = random.randint(2, 5)  
print(N, M)  
A = np.random.randint(-50, 50, (N, M))  
print(str(A) + "\n")  
  
K = random.randint(0, M)  
print("K = " + str(K + 1))  
K\_m = A[:, K].flat  
  
for i in range(M):  
 if i != K:  
 for k in range(N):  
 A[k, i] = A[k, i] \* K\_m[k]  
print(A)

Результат:

4 3

[[ 43 -6 -4]

[ 14 -45 14]

[ 41 -8 -27]

[-17 41 -23]]

K = 3

[[ -172 24 -4]

[ 196 -630 14]

[-1107 216 -27]

[ 391 -943 -23]]

# 11  
# описание: Создать прямоугольную матрицу A, имеющую N строк и M столбцов со  
# случайными элементами. Просуммировать элементы каждой строки  
# матрицы с соответствующими элементами L-й строки.  
#версия Python: 3.6  
  
import numpy as np  
import random  
N = random.randint(2, 5)  
M = random.randint(2, 5)  
print(N, M)  
A = np.random.randint(-50, 50, (N, M))  
print(str(A) + "\n")  
  
L = random.randint(0, N)  
print("L = " + str(L + 1))  
L\_m = A[L, :].flat  
  
for i in range(N):  
 if i != L:  
 for k in range(M):  
 A[i, k] = A[i, k] + L\_m[k]  
print(A)

Результат:

4 3

[[-39 35 43]

[ 36 -6 27]

[-31 -40 19]

[ 18 -26 -8]]

L = 2

[[ -3 29 70]

[ 36 -6 27]

[ 5 -46 46]

[ 54 -32 19]]

# 12  
# описание: 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(2, 5)  
M = random.randint(2, 5)  
print(N, M)  
A = np.random.randint(-50, 50, (N,M))  
A = np.array(A, float)  
  
print(str(A) + "\n")  
  
for i in range(N):  
 mx = np.max(A[i, :])  
 for k in range(M):  
 if A[i, k] != mx:  
 print(A[i, k] / mx)  
 A[i, k] = round(A[i, k] / mx, 2)  
print(A)

Результат:

3 5

[[-24. 26. 25. 25. 22.]

[-34. 25. -15. 5. -3.]

[-25. 14. -32. 9. -17.]]

-0.9230769230769231

0.9615384615384616

0.9615384615384616

0.8461538461538461

-1.36

-0.6

0.2

-0.12

-1.7857142857142858

-2.2857142857142856

0.6428571428571429

-1.2142857142857142

[[-0.92 26. 0.96 0.96 0.85]

[-1.36 25. -0.6 0.2 -0.12]

[-1.79 14. -2.29 0.64 -1.21]]

# 13  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
  
print(N,M)  
  
matrix = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(matrix))  
  
col\_max = np.max(matrix, axis=0)  
matrix = matrix / col\_max  
print("Полученная матрица:\r\n {}".format(matrix))

Результат:

6 3

Матрица:

[[-9 -5 2]

[ 7 6 -4]

[ 8 5 0]

[-7 0 -1]

[ 5 -9 -6]

[ 9 6 -5]]

Полученная матрица:

[[-1. -0.83333333 1. ]

[ 0.77777778 1. -2. ]

[ 0.88888889 0.83333333 0. ]

[-0.77777778 0. -0.5 ]

[ 0.55555556 -1.5 -3. ]

[ 1. 1. -2.5 ]]

# 14  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
  
print(N,M)  
  
matrix = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(matrix))  
  
max\_el = np.max(matrix)  
matrix = matrix / max\_el  
print("Полученная матрица:\r\n {}".format(matrix))

Результат:

8 5

Матрица:

[[-7 1 -2 2 5]

[-5 -4 -7 -6 -7]

[-6 -8 -8 -2 -6]

[-4 6 6 3 -6]

[-4 5 2 -1 2]

[-9 9 -8 -1 9]

[-7 8 1 8 -3]

[ 2 -1 8 2 -1]]

Полученная матрица:

[[-0.77777778 0.11111111 -0.22222222 0.22222222 0.55555556]

[-0.55555556 -0.44444444 -0.77777778 -0.66666667 -0.77777778]

[-0.66666667 -0.88888889 -0.88888889 -0.22222222 -0.66666667]

[-0.44444444 0.66666667 0.66666667 0.33333333 -0.66666667]

[-0.44444444 0.55555556 0.22222222 -0.11111111 0.22222222]

[-1. 1. -0.88888889 -0.11111111 1. ]

[-0.77777778 0.88888889 0.11111111 0.88888889 -0.33333333]

[ 0.22222222 -0.11111111 0.88888889 0.22222222 -0.11111111]]

# 15  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
H = 2  
print(N,M)  
  
matrix = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(matrix))  
  
matrix\_bool = matrix == H  
row\_sum = np.sum(matrix\_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())

Результат:

2 9

Матрица:

[[ 8 5 -1 -9 -2 -6 5 3 6]

[-1 -3 -6 -9 2 4 6 -8 4]]

Столбцы в которых встречается значение 2:

[4]

Столбцы в которых нет значения 2:

[0 1 2 3 5 6 7 8]

# 16  
# описание: 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  
  
N = 4  
M = 5  
  
L = 2  
  
  
matrix = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(matrix))  
  
matrix = np.delete(matrix, L, axis=0)  
print("Полученная матрица:\r\n {}".format(matrix))

Результат:

Матрица:

[[ 3 -8 2 4 -6]

[ 6 2 4 -4 -1]

[ 0 2 6 -5 9]

[ 3 2 8 3 6]]

Полученная матрица:

[[ 3 -8 2 4 -6]

[ 6 2 4 -4 -1]

[ 3 2 8 3 6]]

# 17  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
L = 1  
print(N,M)  
  
matrix = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(matrix))  
  
row = np.random.randint(low=-9, high=10, size=M)  
print("Строка для вставки: {}".format(row))  
  
matrix = np.insert(matrix, L, row, axis=0)  
print("Полученная матрица:\r\n {}".format(matrix))

Результат:

5 2

Матрица:

[[ 9 7]

[ 6 -8]

[ 4 -4]

[ 1 1]

[-4 9]]

Строка для вставки: [2 5]

Полученная матрица:

[[ 9 7]

[ 2 5]

[ 6 -8]

[ 4 -4]

[ 1 1]

[-4 9]]

# 18  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
  
print(N,M)  
  
matrix = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(matrix))  
  
diagonal\_main = np.diagonal(matrix)  
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(matrix[::-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))

Результат:

7 9

Матрица:

[[ 9 9 -6 -3 9 -4 -8 -9 -9]

[-1 -9 5 -3 -4 -4 -1 5 0]

[ 4 1 9 -8 4 -4 -4 9 5]

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

[ 8 -7 1 4 -1 7 -1 9 -2]

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

[-1 -2 -8 -1 -4 0 -1 2 -2]]

Элементы главной диагонали:

[ 9 -9 9 2 -1 2 -1]

Cумма элементов главной диагонали:

11

Элементы побочной диагонали:

[-1 -7 1 2 4 -4 -8]

сумму элементов побочной диагонали:

-13

# 19  
# описание: 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  
  
N = 4  
  
  
A = np.random.randint(low=-9, high=10, size=(N, N))  
print("Матрица:\r\n{}".format(A))  
  
diagonal\_elements = np.array([np.diagonal(A, i) for i in [1, -1]]).flatten()  
print("Элементы, расположенные параллельно главной диагонали:")  
print(diagonal\_elements)  
  
print("Сумма элементов, расположенные параллельно главной диагонали:")  
print(np.sum(diagonal\_elements))

Результат:

Матрица:

[[ 9 8 -5 -1]

[-1 0 6 7]

[ 5 -4 0 1]

[-2 -5 8 -1]]

Элементы, расположенные параллельно главной диагонали:

[ 8 6 1 -1 -4 8]

Сумма элементов, расположенные параллельно главной диагонали:

18

# 20  
# описание: 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  
  
N = 4  
  
  
A = np.random.randint(low=-9, high=10, size=(N, N))  
print("Матрица:\r\n{}".format(A))  
  
diagonal\_elements = np.array([np.diagonal(A[::-1], i) for i in [1, -1]]).flatten()  
print("Элементы, расположенные параллельно побочной диагонали:")  
print(diagonal\_elements)  
  
print("Сумма элементов, расположенные параллельно побочной диагонали:")  
print(np.prod(diagonal\_elements))

Результат:

Матрица:

[[ 8 3 9 -4]

[-8 -5 -1 0]

[ 4 -6 5 -5]

[ 9 -3 5 -5]]

Элементы, расположенные параллельно побочной диагонали:

[-3 5 0 4 -5 9]

Сумма элементов, расположенные параллельно побочной диагонали:

0

# 21  
# описание: 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  
  
N = 4  
  
  
A = np.random.randint(low=-9, high=10, size=(N, N)).astype(float)  
print("Матрица:\r\n{}".format(A))  
  
diagonal\_elements = [np.diagonal(A, i) for i in [1, -1]]  
print("Элементы, расположенные параллельно главной диагонали:")  
print(diagonal\_elements)  
  
values = (diagonal\_elements[0] + diagonal\_elements[1])/2  
print(values)  
  
rng = np.arange(N-1)  
A[rng, rng+1] = values[rng]  
A[rng+1, rng] = values[rng]  
  
print("Полученная матрица:\r\n{}".format(A))

Результат:

Матрица:

[[ 5. 5. -2. 4.]

[ 1. 8. 4. 9.]

[-3. 0. -6. 6.]

[-4. 0. 2. 9.]]

Элементы, расположенные параллельно главной диагонали:

[array([5., 4., 6.]), array([1., 0., 2.])]

[3. 2. 4.]

Полученная матрица:

[[ 5. 3. -2. 4.]

[ 3. 8. 2. 9.]

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

[-4. 0. 4. 9.]]

# 22  
# описание: 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 = np.random.randint(2,10)  
M = np.random.randint(2,10)  
  
print(N,M)  
  
A = np.random.randint(low=0, high=2, size=(N, M))  
print("Матрица:\r\n{}".format(A))  
  
col = [i % 2 for i in np.sum(A, axis=1)]  
  
A = np.insert(A, M, col, axis=1)  
print("Полученная матрица:\r\n {}".format(A))

Результат:

2 6

Матрица:

[[1 1 0 1 0 1]

[0 1 0 0 0 0]]

Полученная матрица:

[[1 1 0 1 0 1 0]

[0 1 0 0 0 0 1]]

# 23  
# описание: 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  
  
N = 4  
  
  
A = np.random.randint(low=-9, high=10, size=(N, N))  
print("Матрица:\r\n{}".format(A))  
  
elements = np.diagonal(A, 1)  
print("Элементы, расположенные выше главной диагонали:")  
print(elements)  
  
print("Сумма элементов, расположенных выше главной диагонали:")  
print(np.sum(elements))  
  
elements = np.diagonal(A[::-1], -1)  
print("Элементы, расположенные выше побочной диагонали:")  
print(elements)  
  
print("Произведение элементов, расположенных выше побочной диагонали:")  
print(np.prod(elements))

Результат:

Матрица:

[[-9 -6 -2 8]

[-7 -5 -6 8]

[ 1 2 -4 1]

[-7 7 -7 -4]]

Элементы, расположенные выше главной диагонали:

[-6 -6 1]

Сумма элементов, расположенных выше главной диагонали:

-11

Элементы, расположенные выше побочной диагонали:

[ 1 -5 -2]

Произведение элементов, расположенных выше побочной диагонали:

10

# 24  
# описание: 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  
  
N = 4  
M = 5  
  
L = 2  
K = 2  
  
A = np.random.randint(low=-9, high=10, size=(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])))

Результат:

Матрица:

[[-7 5 -6 1 3]

[-5 5 4 9 3]

[-6 5 -9 -4 4]

[-1 -8 -8 5 -2]]

Cумма элементов 1 части: -2

Cумма элементов 2 части: 14

Cумма элементов 3 части: -10

Cумма элементов 4 части: -14

# 25  
# описание: 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  
  
N = 4  
M = 5  
  
L = 2  
K = 2  
  
  
A = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(A))  
  
matrix\_bool = A == 0  
col = np.sum(matrix\_bool, axis=1)  
A = np.insert(A, M, col, axis=1)  
  
row = np.append(np.sum(matrix\_bool, axis=0), 0)  
A = np.insert(A, N, row, axis=0)  
  
print("Полученная матрица:\r\n {}".format(A))

Результат:

Матрица:

[[ 8 -4 -3 -9 -9]

[-7 6 -2 8 9]

[ 5 -5 9 7 0]

[ 0 -4 -9 3 5]]

Полученная матрица:

[[ 8 -4 -3 -9 -9 0]

[-7 6 -2 8 9 0]

[ 5 -5 9 7 0 1]

[ 0 -4 -9 3 5 1]

[ 1 0 0 0 1 0]]

# 26  
# описание: 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  
  
N = 4  
M = 5  
  
L = 2  
K = 2  
  
A = np.random.randint(low=-9, high=10, size=(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])))

Результат:

Матрица:

[[ 0 -6 6 -4 1]

[ 0 3 8 8 -8]

[ 6 3 7 -6 4]

[ 6 2 -3 6 -6]]

Cреднее арифметическое 1 части: -0.75

Cреднее арифметическое 2 части: 1.8333333333333333

Cреднее арифметическое 3 части: 4.25

Cреднее арифметическое 4 части: 0.3333333333333333

# 27  
# описание: 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  
  
N = 6  
M = 7  
  
H = 2  
  
  
matrix = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(matrix))  
  
matrix\_bool = matrix == H  
col\_sum = np.sum(matrix\_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())

Результат:

Матрица:

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

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

[ 2 4 4 -4 -9 4 8]

[ 2 2 -2 7 -5 0 5]

[ 0 6 -1 7 8 -4 -3]

[ 0 8 6 5 8 5 -8]]

Строки в которых встречается значение 2:

[1 2 3]

Строки в которых нет значения 2:

[0 4 5]

# 28  
# описание: 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  
  
N = 4  
M = 5  
  
K = 2  
  
  
matrix = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(matrix))  
  
matrix = np.delete(matrix, K, axis=1)  
print("Полученная матрица:\r\n {}".format(matrix))

Результат:

Матрица:

[[ 1 -2 -6 -9 -7]

[-5 -1 8 2 -6]

[ 6 -6 0 4 -6]

[ 5 3 7 9 5]]

Полученная матрица:

[[ 1 -2 -9 -7]

[-5 -1 2 -6]

[ 6 -6 4 -6]

[ 5 3 9 5]]

# 29  
# описание: 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=1  
print(N,M)  
  
matrix = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(matrix))  
  
col = np.random.randint(low=-9, high=10, size=N)  
print("Столбец для вставки: {}".format(col))  
  
matrix = np.insert(matrix, K, col, axis=1)  
print("Полученная матрица:\r\n {}".format(matrix))

Результат:

6 9

Матрица:

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

[-4 -4 -6 4 3 -1 2 -2 2]

[ 8 -8 1 1 4 -2 7 1 8]

[ 7 -9 -1 3 9 3 2 0 -4]

[ 1 -1 2 -6 -9 4 -4 -5 -6]

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

Столбец для вставки: [6 3 4 5 8 3]

Полученная матрица:

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

[-4 3 -4 -6 4 3 -1 2 -2 2]

[ 8 4 -8 1 1 4 -2 7 1 8]

[ 7 5 -9 -1 3 9 3 2 0 -4]

[ 1 8 -1 2 -6 -9 4 -4 -5 -6]

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

# 30  
# описание: 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= random.randint (1,10)  
M= random.randint (1,10)  
  
print(N,M)  
  
A = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(A))  
  
row = np.sum(A, axis=0) \* -1  
  
A = np.insert(A, N, row, axis=0)  
print("Полученная матрица:\r\n {}".format(A))

Результат:

7 9

Матрица:

[[ 8 -7 2 5 5 -3 3 6 1]

[ 6 7 -6 1 -6 5 -6 6 -6]

[ 7 8 7 0 8 9 -4 -6 -5]

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

[ 2 -9 -1 -5 -8 -9 -1 -2 1]

[-9 0 8 6 4 7 3 2 -5]

[-3 -1 -8 6 7 -9 4 -3 -9]]

Полученная матрица:

[[ 8 -7 2 5 5 -3 3 6 1]

[ 6 7 -6 1 -6 5 -6 6 -6]

[ 7 8 7 0 8 9 -4 -6 -5]

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

[ 2 -9 -1 -5 -8 -9 -1 -2 1]

[ -9 0 8 6 4 7 3 2 -5]

[ -3 -1 -8 6 7 -9 4 -3 -9]

[ -4 10 0 -4 -15 1 -5 -11 17]]

# 31  
# описание: 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= random.randint (1,10)  
M= random.randint (1,10)  
  
print(N,M)  
  
A = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(A))  
  
col = np.sum(A, axis=1) \* -1  
  
A = np.insert(A, M, col, axis=1)  
print("Полученная матрица:\r\n {}".format(A))

Результат:

2 2

Матрица:

[[ 6 5]

[-3 -9]]

Полученная матрица:

[[ 6 5 -11]

[ -3 -9 12]]

# ГЛАВА 3 РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

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:

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

Теперь реализуем это при помощи Python и Numpy

Нам дано 5 систем, каждая из них записана в отдельном файле в формате .csv:

Вариант №4

Система №1

| 1.46 1.95 1.59| \* | X1| = | 0.69|

| 1.06 1.41 1.14 | \* | X2| = | 0.06|

| 1.94 1.85 1.57 | \* | X3| = | 0.69|

Система №2

| 1.70 1.51 | \* | X1| = | 0.91|

| 1.17 1.80 | \* | X2| = | 0.67|

Система №3

| 1.77 1.76 1.61 | \* | X1| = | 0.47|

| 1.09 1.72 1.32 | \* | X2| = | 0.55|

| 1.62 1.80 1.44 | \* | X3| = | 0.37|

Система №4

| 1.64 1.71 | \* | X1| = | 0.66|

| 1.06 1.56 | \* | X2| = | 0.24|

Система №5

| 1.66 1.77 1.53 1.08 1.21 | \* | X1| = | 0.31|

| 1.53 1.28 1.10 1.36 1.85 | \* | X2| = | 0.57|

| 1.49 1.59 1.20 1.66 1.30 | \* | X3| = | 0.56|

| 1.37 1.98 1.11 1.22 1.83 | \* | X4| = | 0.57|

| 1.40 1.22 1.83 1.08 1.32 | \* | X5| = | 0.94|

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('/nympy-gauss-slau.csv', delimiter=';')  
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("Решение:")  
 print(slv)  
 numpy.savetxt(f, numpy.array([slv]), delimiter=',')  
f.close()

Результат:

Система 1

Матрица A:

[[1.46 1.95 1.59]

[1.06 1.41 1.14]

[1.94 1.85 1.57]]

Матрица B:

[0.69 0.06 0.69]

Решение [ -4.72573609 -31.32061068 43.18538713]

Система 2

Матрица A:

[[1.7 1.51]

[1.17 1.8]]

Матрица B:

[0.91 0.67]

Решение [ 0.48426505 0.05744993]

Система 3

Матрица A:

[[1.77 1.76 1.61]

[1.09 1.72 1.32]

[1.62 1.80 1.44]]

Матрица B: [0.47 0.55 0.37]

Решение [-0.55715252 -0.13202733 1.048775206]

Система 4

Матрица A:

[[1.64 1.71]

[1.06 1.56]]

Матрица B:

[0.66 0.24]

Решение [0.83024939 -0.41029766]

Система 5

Матрица A:

[[1.66 1.77 1.53 1.08 1.21]

[1.53 1.28 1.1 1.36 1.85]

[1.49 1.59 1.2 1.66 1.3]

[1.37 1.98 1.11 1.22 1.83]

[1.4 1.22 1.83 1.08 1.32]]

Матрица B:

[0.31 0.57 0.56 0.75 0.94]

Решение [ -1.32605917 0.04774626 0.840117126 0.451313922 0.54045381]

# ЗАКЛЮЧЕНИЕ

В настоящее время стремительным развитием персональной вычислительной техники, происходит постепенное изменение требований, предъявляемых к языкам программирования. Все большую роль начинают играть интерпретируемые языки, поскольку возрастающая мощь персональных компьютеров начинает обеспечивать достаточную скорость выполнения интерпретируемых программ. А единственным существенным преимуществом компилируемых языков программирования является создаваемый ими высокоскоростной код. Когда скорость выполнения программы не является критичной величиной, наиболее правильным выбором будет интерпретируемый язык, как более простой и гибкий инструмент программирования.

В связи с этим, определенный интерес и спрос на язык программирования Python возрастает быстрыми темпами.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. − М: Изд-во «Мир», 1989. − 360 с.;
2. Рейтц К., Шлюссер. Автостопом по Python. — СПб.: Питер, 2017. — 336 с.: ил. — (Серия «Бестселлеры O’Reilly»).
3. <https://pythonworld.ru/numpy>
4. <https://habr.com/ru/post/352678/>
5. <https://www.numpy.org>
6. <https://proglib.io/p/7-yazykov-programmirovaniya-kotorye-ty-dolzhen-znat-v-2020-godu-2019-11-17>
7. <https://ru.wikipedia.org/wiki/Python>
8. <https://pythonworld.ru/numpy>