**Оглавление**

[**ВВЕДЕНИЕ** 1](bookmark://_Toc19508963#_Toc19508963)

[**ГЛАВА 1 МЕТОДЫ СОРТИРОВКИ** 2](bookmark://_Toc19508964#_Toc19508964)

[**ГЛАВА 2 БИБЛИОТЕКА NumPy** 13](bookmark://_Toc19508965#_Toc19508965)

[**ГЛАВА 3 РЕШЕНИЕ СЛАУ МЕТОДОМ ГАУССА** 38](bookmark://_Toc19508966#_Toc19508966)

[**ЗАКЛЮЧЕНИЕ** 43](bookmark://_Toc19508967#_Toc19508967)

# **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. Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

NumPy — библиотека с [открытым](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [исходным кодом](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) для [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [Python](https://ru.wikipedia.org/wiki/Python).

Возможности:

* поддержка многомерных [массивов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2) (включая [матрицы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)));
* поддержка [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) математических функций, предназначенных для работы с многомерными массивами.

В этой работе будут рассмотрены: методы сортировки, применение библиотеки NumPy, решение СЛАУ методом Гаусса.

# **ГЛАВА 1 МЕТОДЫ СОРТИРОВКИ**

**1.1 Сортировка вставками**

Сортировка вставками — достаточно простой алгоритм. Как в и любом другом алгоритме сортировки, с увеличением размера сортируемого массива увеличивается и время сортировки. Основным преимуществом алгоритма сортировки вставками является возможность сортировать массив по мере его получения. То есть имея часть массива, можно начинать его сортировать. В параллельном программирование такая особенность играет не маловажную роль.

Задача заключается в следующем: есть часть массива, которая уже отсортирована, и требуется вставить остальные элементы массива в отсортированную часть, сохранив при этом упорядоченность. Для этого на каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированной части массива, до тех пор пока весь набор входных данных не будет отсортирован. Метод выбора очередного элемента из исходного массива произволен, однако обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве.

Так как в процессе работы алгоритма могут меняться местами только соседние элементы, каждый обмен уменьшает число инверсий на единицу. Следовательно, количество обменов равно количеству инверсий в исходном массиве вне зависимости от реализации сортировки. Максимальное количество инверсий содержится в массиве, элементы которого отсортированы не по возрастанию. Число инверсий в таком массиве n(n−1)2*n(n−1)2*.

Алгоритм работает за O(n+k)*O(n+k)*, где k*k* — число обменов элементов входного массива, равное числу инверсий. В среднем и в худшем случае — за O(n2)*O(n2)*. Минимальные оценки встречаются в случае уже упорядоченной исходной последовательности элементов, наихудшие — когда они расположены в обратном порядке.

Рассмотрим алгоритм сортировки вставками на примере колоды игральных карт. Процесс их упорядочивания по возрастанию (в колоде карты расположены в случайном порядке) будет следующим. Обратим внимание на вторую карту, если ее значение меньше первой, то меняем эти карты местами, в противном случае карты сохраняют свои позиции, и алгоритм переходит к шагу 2. На 2-ом шаге смотрим на третью карту, здесь возможны четыре случая отношения значений карт:

1. первая и вторая карта меньше третьей;
2. первая и вторая карта больше третьей;
3. первая карта уступает значением третьей, а вторая превосходит ее;
4. первая карта превосходит значением третью карту, а вторая уступает ей.

В первом случае не происходит никаких перестановок. Во втором – вторая карта смещается на место третьей, первая на место второй, а третья карта занимает позицию первой. В предпоследнем случае первая карта остается на своем месте, в то время как вторая и третья меняются местами. Ну и наконец, последний случай требует рокировки лишь первой и третьей карт. Все последующие шаги полностью аналогичны расписанным выше.

Реализация алгоритма:

*def fast\_insertion\_sort(l):*

*for i in xrange(1, len(l)):*

*j = i - 1*

*value = l.pop(i)*

*while (j >= 0) and (l[j] > value):*

*j -= 1*

*l.insert(j + 1, value)*

*return l*

|  |  |
| --- | --- |
| **Худшее время** | О(*n^*2) сравнений, обменов |
| **Лучшее время** | O(*n*) сравнений, O(*1*) обмен |
| **Среднее время** | О(*n^*2) сравнений, обменов |
| **Затраты памяти** | О(*n*) всего, O(*1*) вспомогательный |

**1.2 Обменная сортировка (метод «пузырька»)**

Сортировка пузырьком (обменная сортировка) – простой в реализации и малоэффективный алгоритм сортировки. Метод изучается одним из первых на курсе теории алгоритмов, в то время как на практике используется очень редко.

Идея алгоритма заключается в следующем. Соседние элементы последовательности сравниваются между собой и, в случае необходимости, меняются местами. В качестве примера рассмотрим упорядочивание методом пузырьковой сортировки массива, количество элементов N которого равно 5: 9, 1, 4, 7, 5. В итоге должен получиться массив с элементами, располагающимися в порядке возрастания их значений (см. рис.).

Вначале сравниваются два первых элемента последовательности: 9 и 1. Так как значение первого элемента больше значения второго, т. е. 9>1, они меняются местами. Далее сравниваются второй и третий элементы: девятка больше четверки, следовательно, элементы снова обмениваются позициями. Аналогично алгоритм продолжает выполняться до тех пор, пока все элементы массива не окажутся на своих местах. Всего для этого потребуется N\*(N-1) сравнений.

При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма).

Пример работы алгоритма:

Возьмём массив с числами «5 1 4 2 8» и отсортируем значения по возрастанию, используя сортировку пузырьком. Выделены те элементы, которые сравниваются на данном этапе.

Первый проход:

(**5 1** 4 2 8) (**1 5** 4 2 8), Здесь алгоритм сравнивает два первых элемента и меняет их местами.

(1 **5 4** 2 8) (1 **4 5** 2 8), Меняет местами, так как 5>4

(1 4 **5 2** 8) (1 4 **2 5** 8), Меняет местами, так как 5>2

(1 4 2 **5 8**) (1 4 2 **5 8**), Теперь, ввиду того, что элементы стоят на своих местах (8>5), алгоритм не меняет их местами.

Второй проход:

(**1 4** 2 5 8) (**1 4** 2 5 8)

(1 **4 2** 5 8) (1 **2 4** 5 8), Меняет местами, так как 4>2

(1 2 **4 5** 8) (1 2 **4 5** 8)

(1 2 4 **5 8**) (1 2 4 **5 8**)

Теперь массив полностью отсортирован, но алгоритм не знает так ли это. Поэтому ему необходимо сделать полный проход и определить, что перестановок элементов не было.

Третий проход:

(**1 2** 4 5 8) (**1 2** 4 5 8)

(1 **2 4** 5 8) (1 **2 4** 5 8)

(1 2 **4 5** 8) (1 2 **4 5** 8)

(1 2 4 **5 8**) (1 2 4 **5 8**)

Реализация алгоритма:

*def swap(arr, i, j):*

*arr[i], arr[j] = arr[j], arr[i]*

*def bubble\_sort(arr):*

*i = len(arr)*

*while i > 1:*

*for j in xrange(i - 1):*

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

*swap(arr, j, j + 1)*

*i -= 1*

|  |  |
| --- | --- |
| **Худшее время** | О(*n^*2) сравнений, обменов |
| **Лучшее время** | O(*n*) сравнений, O(*1*) обмен |
| **Среднее время** | О(*n^*2) сравнений, обменов |
| **Затраты памяти** | O(*1*) |

**1.3 Сортировка выбором**

Сортировка выбором – возможно, самый простой в реализации алгоритм сортировки. Как и в большинстве других подобных алгоритмов, в его основе лежит операция сравнения. Сравнивая каждый элемент с каждым, и в случае необходимости производя обмен, метод приводит последовательность к необходимому упорядоченному виду.

Идея алгоритма очень проста. Пусть имеется массив A размером N, тогда сортировка выбором сводится к следующему:

1. берем первый элемент последовательности A[i], здесь i – номер элемента, для первого i равен 1;
2. находим минимальный (максимальный) элемент последовательности и запоминаем его номер в переменную key;
3. если номер первого элемента и номер найденного элемента не совпадают, т. е. если key≠1, тогда два этих элемента обмениваются значениями, иначе никаких манипуляций не происходит;
4. увеличиваем i на 1 и продолжаем сортировку оставшейся части массива, а именно с элемента с номером 2 по N, так как элемент A[1] уже занимает свою позицию;

С каждым последующим шагом размер подмассива, с которым работает алгоритм, уменьшается на 1, но на способ сортировки это не влияет, он одинаков для каждого шага.

Реализация алгоритма:

*def swap(arr, i, j):*

*arr[i], arr[j] = arr[j], arr[i]*

*def select\_sort(arr):*

*i = len(arr)*

*while i > 1:*

*max = 0*

*for j in xrange(i):*

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

*max = j*

*swap(arr, i - 1, max)*

*i -= 1*

|  |  |
| --- | --- |
| **Худшее время** | О(*n*2) |
| **Лучшее время** | О(*n*2) |
| **Среднее время** | О(*n*2) |
| **Затраты памяти** | О(*n*) всего, O(1) дополнительно |

**1.4 Сравнение методов сортировки**

Для сравнения методов необходимо написать код сравнения сортировок. Были созданы три Python File, а именно: bubble, insert и select. В данных файлах записаны коды, которые будут сортировать мой массив.

Код из файла bubble:

*def bubble(arr, dim):*

*alg\_count = [0, 0]*

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

*for j in range(dim - i - 1):*

*alg\_count[0] += 1*

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

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

*alg\_count[1] += 1*

*return alg\_count*

Код из файла insert:

*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*

*return alg\_count*

Код из файла select:

*def select(arr, dim):*

*alg\_count = [0, 0]*

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

*m = k*

*i = k + 1*

*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*

После того как были созданы данные файлы, был написан код который будет сравнивать данные элементы на примере одномерного массива, состоящего из 65 элементов, после чего код будет показывать количество сравнений и перестановок через счетчик. При этом, массивы были как упорядоченные, так и обратно упорядоченные, и случайные. Сами же результаты занесены в сравнительную таблицу, которая представлена ниже:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Массив | Показатель | 65 элементов | | |
|  |  | Select | Insert | Bubble |
| Упорядоченный | С | 2080 | 2080 | 2080 |
| М | 0 | 0 | 0 |
| Обратно упорядоченные | С | 2080 | 2080 | 2080 |
| М | 32 | 2080 | 2080 |
| Случайные | С | 2080 | 2080 | 2080 |
| М | 59.884 | 1029.3305 | 1029.3305 |

Анализируя данную таблицу, можно заметить, что наиболее эффективным мотетом сортировки является “Select”, потому что в упорядоченном, обратно упорядоченном и случайном массиве он показал наименьшее количество перестановок (показатель M в сравнительной таблице эффективности). Сам алгоритм сравнения методов:

*import random*

*import select\_2*

*import bubble\_2*

*import insert\_2*

*DIM = 65*

*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()*

# **ГЛАВА 2 БИБЛИОТЕКА NumPy**

NumPy это open-source модуль для python, который предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций. Они объединяются в высокоуровневые пакеты. Они обеспечивают функционал, который можно сравнить с функционалом MatLab. NumPy (Numeric Python) предоставляет базовые методы для манипуляции с большими массивами и матрицами. SciPy (Scientific Python) расширяет функционал numpy огромной коллекцией полезных алгоритмов, таких как минимизация, преобразование Фурье, регрессия, и другие прикладные математические техники.

**Задачи, решенные с помощью NumPy.**

№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))***

Матрица:

[[ 2 -6 2 -9 -4]

[-6 2 -9 -6 0]

[-2 -6 8 5 -7]

[ 7 -7 9 -5 -8]]

Наибольшее значение: 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.

***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))***

Матрица:

[[-5 0 9 0 -6]

[ 9 8 -3 -1 6]

[-8 -3 -3 -4 -5]

[ 1 8 7 5 -7]]

Наибольшее среднее значение: 3.8

№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))***

Матрица:

[[-7 5 6 3 -1]

[-3 -4 9 9 -6]

[-3 6 -7 3 8]

[ 5 -5 5 3 -9]]

Наименьшее значение: -7

№4. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти наименьшее значение среди средних значений для каждой строки матрицы.

***import numpy as np***

***import random***

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

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

***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))***

Матрица:

[[-7 1 -8 1 -2]

[ 1 6 9 -4 -4]

[ 6 -5 -7 7 1]

[-6 -5 2 2 4]]

Наименьшее среднее значение: -3.0

№5. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Определить средние значения по всем строкам и столбцам матрицы. Результат оформить в виде матрицы из N + 1 строк и M + 1 столбцов.

***import numpy as np***

***import random***

***N = 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)***

Матрица:

[[ 6 -9 -1 -1 3]

[ 5 2 -9 -2 -1]

[ 6 -6 3 -5 6]

[ 4 -2 2 6 8]]

Новая матрица:

[[ 6. -9. -1. -1. 3. -0.4 ]

[ 5. 2. -9. -2. -1. -1. ]

[ 6. -6. 3. -5. 6. 0.8 ]

[ 4. -2. 2. 6. 8. 3.6 ]

[ 5.25 -3.75 -1.25 -0.5 4. 0. ]]

№6. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти сумму элементов всей матрицы. Определить, какую долю в этой сумме составляет сумма элементов каждого столбца. Результат оформить в виде матрицы из N + 1 строк и M столбцов.

***import numpy as np***

***import random***

***N = 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("�"оля элементов каждого столбца-"+str(np.sum(A, axis=0)/np.sum(A)))***

***A = np.vstack((A,M\_sum))***

***print("Матрица с N +1-",A)***

Матрица:

[[-8 -3 1 6 -7]

[-7 5 -5 2 -3]

[-7 -3 -4 -8 9]

[-8 -6 2 0 1]]

Новая матрица:

[[-8. -3. 1. 6. -7. ]

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

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

[-8. -6. 2. 0. 1. ]

[ 0.69767442 0.1627907 0.13953488 -0. -0. ]]

№7. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти сумму элементов всей матрицы. Определить, какую долю в этой сумме составляет сумма элементов каждой строки. Результат оформить в виде матрицы из N строк и M+1 столбцов.

***import numpy as np***

***import random***

***N = 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("поля 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)***

Матрица:

[[ 8 -9 1 -7 1]

[ 4 4 7 -9 0]

[ 3 4 5 -8 -7]

[-8 9 -2 8 1]]

Сумма элементов всей матрицы: 5

[[ 8. -9. 1. -7. 1. -1.2]

[ 4. 4. 7. -9. 0. 1.2]

[ 3. 4. 5. -8. -7. -0.6]

[-8. 9. -2. 8. 1. 1.6]]

№8. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Определить, сколько отрицательных элементов содержится в каждом столбце и в каждой строке матрицы. Результат оформить в виде матрицы из N + 1 строк и M + 1 столбцов.

***import numpy as np***

***import random***

***N = 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)***

Матрица:

[[-8 -9 2 -1 3]

[ 5 -7 -2 7 1]

[-9 -3 -9 -1 1]

[ 7 -4 -9 0 -4]]

Новая матрица:

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

[ 5. -7. -2. 7. 1. 2.]

[-9. -3. -9. -1. 1. 4.]

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

[ 2. 4. 3. 2. 1. 0.]]

№9. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Определить, сколько нулевых элементов содержится в верхних L строках матрицы и в левых К столбцах матрицы.

***import numpy as np***

***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))***

Матрица:

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

[ 9 -6 1 9 -4]

[ 6 3 0 6 0]

[ 2 -3 -6 6 -8]]

Количество нулевых элементов в верхних L солбцах: 0

Количество нулевых элементов в левых L солбцах: 0

№10. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Перемножить элементы каждого столбца матрицы с соответствующими элементами K-го столбца.

***import numpy as np***

***import random***

***N = random.randint(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)***

Матрица:

[[-8 -4 -8 -4 -7]

[ 7 -7 -7 -3 6]

[-7 -1 7 8 -3]

[ 3 -3 -1 0 -9]]

K-ый столбец:

[[-8]

[ 7]

[-7]

[ 3]]

Новая матрица:

[[ 64 32 64 32 56]

[ 49 -49 -49 -21 42]

[ 49 7 -49 -56 21]

[ 9 -9 -3 0 -27]]

№11. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Просуммировать элементы каждой строки матрицы с соответствующими элементами L-й строки.

***import numpy as np***

***import random***

***N = random.randint(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)***

Матрица:

[[-6 9 -6 5 -8]

[ 4 -1 5 -1 -9]

[ 7 6 5 -2 -5]

[-6 -7 -4 6 -2]]

L страка:

[ 4 -1 5 -1 -9]

Новая матрица:

[[ -2 8 -1 4 -17]

[ 8 -2 10 -2 -18]

[ 11 5 10 -3 -14]

[ -2 -8 1 5 -11]]

№12. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Разделить элементы каждой строки на элемент этой строки с наибольшим значением.

***import numpy as np***

***import random***

***N = random.randint(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)***

Матрица:

[[-3 5 -4 -9 2]

[-4 4 5 6 5]

[ 8 7 0 5 8]

[ 2 5 7 4 -4]]

Новая матрица:

[[-0.6 1. -0.8 -1.8 0.4 ]

[-0.66666667 0.66666667 0.83333333 1. 0.83333333]

[ 1. 0.875 0. 0.625 1. ]

[ 0.28571429 0.71428571 1. 0.57142857 -0.57142857]]

№13. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Разделить элементы каждого столбца матрицы на элемент этого столбца с наибольшим значением.

***import numpy as np***

***import random***

***N = 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))***

Матрица:

[[ 1 -6 -9 8 -4]

[-1 8 8 1 0]

[-1 -1 5 4 2]

[-1 -7 -6 8 3]]

Новая матрица:

[[ 1. -0.75 -1.125 1. -1.33333333]

[-1. 1. 1. 0.125 0. ]

[-1. -0.125 0.625 0.5 0.66666667]

[-1. -0.875 -0.75 1. 1. ]]

№14. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Разделить элементы матрицы на элемент матрицы с наибольшим значением

***import numpy as np***

***import random***

***N = 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))***

Матрица:

[[-7 8 1 -3 1]

[ 5 -8 -6 3 -3]

[-1 -3 2 6 1]

[ 8 -3 8 7 -8]]

Новая матрица:

[[-0.875 1. 0.125 -0.375 0.125]

[ 0.625 -1. -0.75 0.375 -0.375]

[-0.125 -0.375 0.25 0.75 0.125]

[ 1. -0.375 1. 0.875 -1. ]]

№15. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Все элементы имеют целый тип. Дано целое число H. Определить, какие столбцы имеют хотя бы одно такое число, а какие не имеют.

***import numpy as np***

***import random***

***N = 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())***

Матрица:

[[ 4 5 3 6 -5]

[ 8 -1 4 7 -2]

[-6 -3 5 -9 6]

[-2 7 3 8 -3]]

Столбцы, которые имеют хотя бы одно число H - []

Столбцы, которые не имеют это число - [1, 2, 3, 4, 5]

№16. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Исключить из матрицы строку с номером L. Сомкнуть строки матрицы

***import numpy as np***

***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))***

Матрица:

[[ 8 0 3 5 -8]

[ 9 2 -1 -8 4]

[ 4 5 -9 -7 1]

[ 1 6 -8 -1 2]]

L = 1

Новая матрица:

[[ 9 2 -1 -8 4]

[ 4 5 -9 -7 1]

[ 1 6 -8 -1 2]]

№17. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Добавить к матрице строку и вставить ее под номером L.

***import numpy as np***

***import random***

***N = 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))***

Матрица:

[[ 4 -6 5 -5 6]

[ 9 6 4 -1 -9]

[-9 6 -4 -8 7]

[ 3 2 -3 -6 2]]

Допонлительная строка: [0 3 9 9 7]

L = 2

Строка для вставки: [ 2 -1 -6 1 -2]

Новая матрица:

[[ 4 -6 5 -5 6]

[ 9 6 4 -1 -9]

[ 2 -1 -6 1 -2]

[-9 6 -4 -8 7]

[ 3 2 -3 -6 2]]

№18. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти сумму элементов, стоящих на главной диагонали, и сумму элементов, стоящих на побочной диагонали (элементы главной диагонали имеют индексы от [0,0] до [N,N], а элементы побочной диагонали — от [N,0] до [0,N]).

***import numpy as np***

***import random***

***N = 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))***

Матрица:

[[ 2 5 8 -5 4]

[ 8 1 -9 6 -4]

[ 0 2 2 -8 8]

[ 9 -1 3 -1 6]]

Главная диагональ:

[ 2 1 2 -1]

Её сумма = 4

Побочная диагональ:

[ 4 6 2 -1]

Её сумма = 11

№19. Создать квадратную матрицу A, имеющую N строк и N столбцов со случайными элементами. Определить сумму элементов, расположенных параллельно главной диагонали (ближайшие к главной). Элементы главной диагонали имеют индексы от [0,0] до [N,N].

***import numpy as np***

***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))***

Матрица:

[[ 3 -1 3 3 6]

[ 6 -9 -8 5 -4]

[-3 -5 -1 8 4]

[-6 -5 1 -6 8]]

Элементы которые выше главной диагонали:

[-1 -8 8 8]

Их сумма = 7

Элементы которые ниже главной диагонали:

[ 6 -5 1]

Их сумма = 7

№20. Создать квадратную матрицу A, имеющую N строк и N столбцов со случайными элементами. Определить произведение элементов, расположенных параллельно побочной диагонали (ближайшие к побочной). Элементы побочной диагонали имеют индексы от [N,0] до [0,N].

***import numpy as np***

***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))***

Матрица:

[[-1 0 2 -9 -6]

[ 2 0 4 3 -9]

[ 7 -7 -7 2 0]

[-9 -4 1 -1 2]]

Элементы которые выше побочной диагонали:

[-9 4 -7 -9]

Их сумма = -2268

Элементы которые ниже побочной диагонали:

[-9 2 1]

Их сумма = -18

№21. Создать квадратную матрицу A, имеющую N строк и N столбцов со случайными элементами. Каждой паре элементов, симметричных относительно главной диагонали (ближайшие к главной), присвоить значения, равные полусумме этих симметричных значений (элементы главной диагонали имеют индексы от [0,0] до [N,N])

***import numpy as np***

***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))***

Матрица:

[[-1 3 -1 -9]

[ 2 -8 3 -7]

[-6 -5 7 3]

[-7 2 -2 1]]

Новая матрица:

[[-1. 2.5 -3.5 -8. ]

[ 2.5 -8. -1. -2.5]

[-3.5 -1. 7. 0.5]

[-8. -2.5 0.5 1. ]]

№22. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Исходная матрица состоит из нулей и единиц. Добавить к матрице еще один столбец, каждый элемент которого делает количество единиц в каждой строке чётным.

***import numpy as np***

***import random***

***N = 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))***

Матрица:

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

[-3 2 1 -1 -1]

[-5 -1 -2 7 2]

[ 1 8 -4 3 -8]]

Новая матрица:

[[ 1 -5 -6 -1 -1 0]

[-3 2 1 -1 -1 0]

[-5 -1 -2 7 2 1]

№24. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Дан номер строки L и номер столбца K, при помощи которых исходная матрица разбивается на четыре части. Найти сумму элементов каждой части.

***import numpy as np***

***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])))***

Матрица:

[[-1 7 6 5 -4]

[ 3 7 -4 -3 -1]

[-6 0 4 9 -1]

[-2 -6 0 9 5]]

Cумма элементов выше главной диагонали = 20

Произведение элементов выше побочной диагонали = 0

№25. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Определить, сколько нулевых элементов содержится в каждом столбце и в каждой строке матрицы. Результат оформить в виде матрицы из N + 1 строк и M + 1 столбцов.

***import numpy as np***

***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))***

Матрица:

[[ 0 -8 2 -3 -1]

[ 8 -8 -5 9 5]

[-7 -9 2 9 -5]

[-9 1 -5 1 3]]

Новая матрица:

[[ 0 -8 2 -3 -1 1]

[ 8 -8 -5 9 5 0]

[-7 -9 2 9 -5 0]

[-9 1 -5 1 3 0]

[ 1 0 0 0 0 0]]

№26. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Дан номер строки L и номер столбца K, при помощи которых исходная матрица разбивается на четыре части. Найти среднее арифметическое элементов каждой части.

***import numpy as np***

***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])))***

Матрица:

[[ 0 -7 7 -2 7]

[-6 3 9 4 5]

[-8 7 -8 -5 3]

[-8 -1 -5 7 8]]

Вверхняя левая часть: среднее арифметическое = 0.0

[[0]]

Нижняя левая часть: среднее арифметическое = -7.333333333333333

[[-6]

[-8]

[-8]]

Вверхняя правая часть: среднее арифметическое = 1.25

[[-7 7 -2 7]]

Нижняя правая часть: среднее арифметическое = 2.25

[[ 3 9 4 5]

[ 7 -8 -5 3]

[-1 -5 7 8]]

№27. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Все элементы имеют целый тип. Дано целое число H. Определить, какие строки имеют хотя бы одно такое число, а какие не имеют.

***import numpy as np***

***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())***

Матрица:

[[ 6 8 5 9 0]

[-4 4 0 -1 -5]

[-2 8 -7 2 7]

[-6 -9 4 3 -9]]

Строки в которых встречается значение 2:

[2]

Строки в которых нет значения 2:

[0 1 3]

№28. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Исключить из матрицы столбец с номером K. Сомкнуть столбцы матрицы

***import numpy as np***

***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))***

Матрица:

[[ 8 -2 -5 1 -1]

[-7 6 4 -6 2]

[-6 0 3 -8 8]

[ 5 -4 -5 1 0]]

K = 1

Новая матрица:

[[-2 -5 1 -1]

[ 6 4 -6 2]

[ 0 3 -8 8]

[-4 -5 1 0]]

№29. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Добавить к матрице столбец чисел и вставить его под номером K.

***import numpy as np***

***import random***

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

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

***K=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))***

Матрица:

[[ 8 -6 3 4 8]

[-8 -7 6 1 -6]

[-8 7 2 6 0]

[ 9 2 -2 -5 -6]]

Строки в которых встречается значение 2:

[2 3]

Строки в которых нет значения 2:

[0 1]

№30. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Добавить к элементам каждого столбца такой новый элемент, чтобы сумма положительных элементов стала бы равна модулю суммы отрицательных элементов. Результат оформить в виде матрицы из N + 1 строк и M столбцов.

***import numpy as np***

***import random***

***N= 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))***

Матрица:

[[ 7 -3 -1 7 0]

[-9 6 -2 1 -5]

[-5 2 8 5 -1]

[ 4 -8 -8 3 8]]

Новая матрица:

[[ 7 -3 -1 7 0]

[ -9 6 -2 1 -5]

[ -5 2 8 5 -1]

[ 4 -8 -8 3 8]

[ 3 3 3 -16 -2]]

№31. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Добавить к элементам каждой строки такой новый элемент, чтобы сумма положительных элементов стала бы равна модулю суммы отрицательных элементов. Результат оформить в виде матрицы из N строк и M + 1 столбцов.

***import numpy as np***

***import random***

***N= 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))***

Матрица:

[[ 0 6 -5 7 6]

[ 3 0 -3 -3 2]

[ 1 -7 7 7 6]

[-6 -6 6 2 5]]

Новая матрица:

[[ 0 6 -5 7 6 -14]

[ 3 0 -3 -3 2 1]

[ 1 -7 7 7 6 -14]

[ -6 -6 6 2 5 -1]]

# **ГЛАВА 3 РЕШЕНИЕ СЛАУ МЕТОДОМ ГАУССА**

Метод Гаусса — классический метод решения системы линейных алгебраических уравнений (СЛАУ). Назван в честь немецкого математика Карла Фридриха Гаусса. Это метод последовательного исключения переменных, когда с помощью элементарных преобразований система уравнений приводится к равносильной системе треугольного вида, из которой последовательно, начиная с последних (по номеру), находятся все переменные системы. Алгоритм решения СЛАУ методом Гаусса подразделяется на два этапа.

1) На первом этапе осуществляется так называемый прямой ход, когда путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна. А именно, среди элементов первого столбца матрицы выбирают ненулевой, перемещают его на крайнее верхнее положение перестановкой строк и вычитают получившуюся после перестановки первую строку из остальных строк, домножив её на величину, равную отношению первого элемента каждой из этих строк к первому элементу первой строки, обнуляя тем самым столбец под ним. После того, как указанные преобразования были совершены, первую строку и первый столбец мысленно вычёркивают и продолжают пока не останется матрица нулевого размера. Если на какой-то из итераций среди элементов первого столбца не нашёлся ненулевой, то переходят к следующему столбцу и проделывают аналогичную операцию.

2) На втором этапе осуществляется так называемый обратный ход, суть которого заключается в том, чтобы выразить все получившиеся базисные переменные через небазисные и построить фундаментальную систему решений, либо, если все переменные являются базисными, то выразить в численном виде единственное решение системы линейных уравнений. Эта процедура начинается с последнего уравнения, из которого выражают соответствующую базисную переменную (а она там всего одна) и подставляют в предыдущие уравнения, и так далее, поднимаясь по «ступенькам» наверх. Каждой строчке соответствует ровно одна базисная переменная, поэтому на каждом шаге, кроме последнего (самого верхнего), ситуация в точности повторяет случай последней строки.

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

Вариант №19

Система №1

| 1.26 1.99 1.40 1.71 | \* | X1| = | 0.07|

| 1.62 1.02 1.08 1.61 | \* | X2| = | 0.61|

| 1.05 1.13 1.91 1.51 | \* | X3| = | 0.76|

| 1.41 1.69 1.47 1.58 | \* | X4| = | 0.56|

Система №2

| 1.35 1.79 1.32 1.91 1.82 1.34 | \* | X1| = | 0.06|

| 1.70 1.28 1.57 1.85 1.54 1.17 | \* | X2| = | 0.89|

| 1.60 1.82 1.36 1.29 1.10 1.48 | \* | X3| = | 0.17|

| 1.43 1.16 1.60 1.22 1.97 1.96 | \* | X4| = | 0.56|

| 1.92 1.92 1.50 1.56 1.74 1.85 | \* | X5| = | 0.90|

| 1.11 1.66 1.42 1.07 1.48 1.25 | \* | X6| = | 0.16|

Система №3

| 1.48 1.54 1.41 1.34 1.26 | \* | X1| = | 0.43|

| 1.09 1.57 1.56 1.36 1.79 | \* | X2| = | 0.35|

| 1.43 1.70 1.27 1.41 1.64 | \* | X3| = | 0.58|

| 1.22 1.45 1.12 1.25 1.72 | \* | X4| = | 0.11|

| 1.76 1.41 1.11 1.99 1.11 | \* | X5| = | 0.65|

Система №4

| 1.90 1.03 1.93 1.65 | \* | X1| = | 0.44|

| 1.88 1.50 1.49 1.32 | \* | X2| = | 0.59|

| 1.31 1.17 1.40 1.20 | \* | X3| = | 0.73|

| 1.09 1.77 1.31 1.58 | \* | X4| = | 0.91|

Система №5

| 1.05 1.32 1.53 1.48 | \* | X1| = | 0.87|

| 1.54 1.93 1.44 1.59 | \* | X2| = | 0.87|

| 1.39 1.21 1.47 1.25 | \* | X3| = | 0.59|

| 1.49 1.78 1.60 1.67 | \* | X4| = | 0.03|

Для решения каждой системы был написан алгоритм, который последовательно читает файл с системой, и отделяет A-часть матрицы от B-части матрицы. При помощи функции np.linalg.solve мы решаем систему методом Гаусса.

***import numpy as np***

***file = open('nympy-gauss-slau.csv', 'wb+')***

***file.truncate()***

***for i in range(1, 6):***

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

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

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

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

***print("Система " + str(i))***

***print("Матрица A: \n" + str(myA))***

***print("Матрица B: \n" + str(myB))***

***slau = np.linalg.solve(myA, myB)***

***print("Решение ", slau)***

***np.savetxt(file, np.array([slau]), delimiter=',')***

***file.close()***

Результат:

Система 1

Матрица A:

[[1.26 1.99 1.4 1.71]

[1.62 1.02 1.08 1.61]

[1.05 1.13 1.91 1.51]

[1.41 1.69 1.47 1.58]]

Матрица B:

[0.07 0.61 0.76 0.56]

Решение [ 1.20802617 -0.27312888 0.89429872 -1.26351153]

Система 2

Матрица A:

[[1.35 1.79 1.32 1.91 1.82 1.34]

[1.7 1.28 1.57 1.85 1.54 1.17]

[1.6 1.82 1.36 1.29 1.1 1.48]

[1.43 1.16 1.6 1.22 1.97 1.96]

[1.92 1.92 1.5 1.56 1.74 1.85]

[1.11 1.66 1.42 1.07 1.48 1.25]]

Матрица B:

[0.06 0.89 0.17 0.56 0.9 0.16]

Решение [ 2.48001117 -0.42192076 -0.6413774 -1.28376216 1.48067405 -1.4395521 ]

Система 3

Матрица A:

[[1.48 1.54 1.41 1.34 1.26]

[1.09 1.57 1.56 1.36 1.79]

[1.43 1.7 1.27 1.41 1.64]

[1.22 1.45 1.12 1.25 1.72]

[1.76 1.41 1.11 1.99 1.11]]

Матрица B:

[0.43 0.35 0.58 0.11 0.65]

Решение [-1.50423036 2.5230766 -0.61546825 0.91598569 -1.26102192]

Система 4

Матрица A:

[[1.9 1.03 1.93 1.65]

[1.88 1.5 1.49 1.32]

[1.31 1.17 1.4 1.2 ]

[1.09 1.77 1.31 1.58]]

Матрица B:

[0.44 0.59 0.73 0.91]

Решение [-0.74731721 0.85827464 1.35480483 -0.99326878]

Система 5

Матрица A:

[[1.05 1.32 1.53 1.48]

[1.54 1.93 1.44 1.59]

[1.39 1.21 1.47 1.25]

[1.49 1.78 1.6 1.67]]

Матрица B:

[0.87 0.87 0.59 0.03]

Решение [ -9.93827065 15.4516691 19.04650378 -25.83254717]

# **ЗАКЛЮЧЕНИЕ**

В заключение хочется сказать, что с помощью языка программирования Python можно быстро и удобно автоматизировать рутинные задачи. Но для того что бы полностью раскрыть потенциал Python'a, нудно использовать библиотеки.

Также 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 не малый 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. Лутц М. Программирование на Python, том I, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

2. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

3. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.

4. https://ru.wikipedia.org

5. Книга Дж.Вандера Пласа: “Python для сложных задач: наука о данных и машинное обучение”.