Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего

образования «Южно-Уральский государственный университет (национальный

исследовательский университет)»

Высшая школа экономики и управления

Кафедра Информационных технологий в экономике

Программирование на языке Python (курс молодого бойца)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

по дисциплине «ПРОГРАММИРОВАНИЕ»

ЮУрГУ – 380305.2022.117. ПЗ КР

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

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

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

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Сурин Д. А. /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

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

Работа защищена с оценкой

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

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

Челябинск 2019

**АННОТАЦИЯ**

Сурин Д. А.

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-142, 2019

44 с., 1 рисунок.

Библиографический список – 5 наим.

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.

Оглавление

[ВВЕДЕНИЕ 4](#_Toc9787505)

[ГЛАВА 1. СОРТИРОВКА ОДНОМЕРНОГО МАССИВА 5](#_Toc9787506)

[1.1 Алгоритмы сортировки 5](#_Toc9787507)

[1.1.1 Сортировка выбором (select) 5](#_Toc9787508)

[1.1.2 Сортировка включениями (insert) 6](#_Toc9787509)

[1.1.3 Обменная сортировка (bubble) 7](#_Toc9787510)

[1.2 Анализ эффективности 9](#_Toc9787511)

[ГЛАВА 2. МАТРИЧНАЯ МАТЕМАТИКА И РАБОТА С БИБЛИОТЕКАМИ 13](#_Toc9787512)

[2.1 Примеры использования библиотек NumPy и MatPlotLib 13](#_Toc9787514)

[2.2 Решение задач с использованием библиотеки numpy 1](#_Toc9787515)4

[ГЛАВА 3. РЕШЕНИЕ СЛАУ С ПОМОЩЬЮ NUMPY](#_Toc9787516) 39

[ВЫВОД 43](#_Toc9787517)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 44](#_Toc9787518)

# ВВЕДЕНИЕ

В первой части работы мы проанализируем три данных нам алгоритма сортировки: select, insert, bubble. Конечно, помимо них существует множество других алгоритмов, но наша задача выбрать наиболее быстрый и удобный из представленных трех.

Во второй части работы мы рассмотрим библиотеку numpy. Для чего она нужна? 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 нам предстоит сделать то, что еще в том семестре мы делали самостоятельно на обычном листе бумаги: решение системы линейных уравнений методом Гаусса. Использование библиотеки numpy значительно ускоряет и упрощает эту задачу, в чем мы и убедимся позже.

# 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:

1. **def** select(arr, dim):
2. alg\_count = [0, 0]
3. **for** k **in** range(0, dim - 1): *# -1, т.к. последний элемент обменивать уже не надо*
4. m = k *# в m хранится минимальное значение*
5. **i = k + 1 *# откуда начинать поиск минимума (элемент следующий за k)***
6. **for** i **in** range(i, dim):
7. alg\_count[0] += 1
8. **if** arr[i] < arr[m]:
9. m = i
10. **if k != m:**
11. t = arr[k]
12. arr[k] = arr[m]
13. arr[m] = t
14. alg\_count[1] += 1
15. **return alg\_count**

### 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.

1. **def** insert(arr, dim):
2. alg\_count = [0, 0] *# массив для показателей эффективности*
4. **for** i **in** range(1, dim): *# Основной цикл со 2-го элемента право*
5. **temp = arr[i] *# Запомним элемент для сравнения***
6. j = i - 1
7. **while** j >= 0: *# Ищем влево ближайший меньший*
8. alg\_count[0] += 1 *# Считаем операции сравнения*
9. **if** arr[j] > temp:
10. **alg\_count[1] += 1 *# Считаем операции перестановки***
11. arr[j + 1] = arr[j] *# Сдвигаем элемент влево, а на его место ставим наименьший*
12. arr[j] = temp
13. j -= 1
14. **return** alg\_count

### 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.

1. **def** bubble(arr, dim):
2. alg\_count = [0, 0] *# Массив для показателей эффективности*
3. **for** i **in** range(dim - 1): *# Основной цикл для всех элементов*
4. **for** j **in** range(dim - i - 1): *# Цикл для оставшейся части меньших элементов*
5. **alg\_count[0] += 1**
6. **if** arr[j] > arr[j + 1]: *# Сравниваем элементы*
7. arr[j], arr[j + 1] = arr[j + 1], arr[j] *# Меняем элементы местами*
8. alg\_count[1] += 1
9. **return** alg\_count

## 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.

1. **import** random
2. **import** select\_f
3. **import** bubble\_f
4. **import** insert\_f
6. DIM = 40
7. bubble\_arr = []
8. insert\_arr = []
9. select\_arr = []
10. **CTotal = [0, 0, 0]**
11. MTotal = [0, 0, 0]
13. **for** i **in** range(1, DIM+1):
14. bubble\_arr.append(i)
15. **insert\_arr.append(i)**
16. select\_arr.append(i)
18. file = open("Result.txt", "w")
19. **print**("Упорядоченный массив: ")
20. **print(bubble\_arr)**
22. *# метод Select*
23. count = select\_f.select(select\_arr, DIM)
24. **print**("Упорядоченный массив: результат")
25. **print(select\_arr)**
27. CTotal[0] = count[0]
28. MTotal[0] = count[1]
30. ***# метод вставки Insert***
31. count = insert\_f.insert(insert\_arr, DIM)
32. CTotal[1] = count[0]
33. MTotal[1] = count[1]
35. ***# метод пузырька Bubble***
36. count = bubble\_f.bubble(bubble\_arr, DIM)
37. CTotal[2] = count[0]
38. MTotal[2] = count[1]
39. **print**("Размер массива:", DIM)
40. **print("Сравнений:", CTotal[0], CTotal[1], CTotal[2])**
41. **print**("Перестановок:", MTotal[0], MTotal[1], MTotal[2])
43. file.write("Метеды: select -- insert -- bubble**\n**")
44. file.write("Упорядоченный массив:**\n** ")
45. **file.write("Размер массива: " + str(DIM) + "\n")**
46. file.write("Сравнений: " + str(CTotal[0]) + " " + str(CTotal[1]) + " " + str(CTotal[2]) + "**\n**")
47. file.write("Перестановок: " + str(MTotal[0]) + " " + str(MTotal[1]) + " " + str(MTotal[2]) + "**\n**")
49. select\_arr.clear()
50. **bubble\_arr.clear()**
51. insert\_arr.clear()
53. **for** i **in** range(DIM, 0, -1):
54. bubble\_arr.append(i)
55. **insert\_arr.append(i)**
56. select\_arr.append(i)
58. **print**("Обратно упорядоченный массив: ")
59. **print**(bubble\_arr)
61. *# метод Select*
62. count = select\_f.select(select\_arr, DIM)
63. **print**("Обратно упорядоченный массив: результат")
64. **print**(select\_arr)
66. CTotal[0] = count[0]
67. MTotal[0] = count[1]
69. *# метод вставки Insert*
70. **count = insert\_f.insert(insert\_arr, DIM)**
71. CTotal[1] = count[0]
72. MTotal[1] = count[1]
74. *# метод пузырька Bubble*
75. **count = bubble\_f.bubble(bubble\_arr, DIM)**
76. CTotal[2] = count[0]
77. MTotal[2] = count[1]
78. **print**("Размер массива:", DIM)
79. **print**("Сравнений:", CTotal[0], CTotal[1], CTotal[2])
80. **print("Перестановок:", MTotal[0], MTotal[1], MTotal[2])**
82. file.write("**\n**Обратно упорядоченный массив:**\n** ")
83. file.write("Размер массива: " + str(DIM) + "**\n**")
84. file.write("Сравнений: " + str(CTotal[0]) + " " + str(CTotal[1]) + " " + str(CTotal[2]) + "**\n**")
85. **file.write("Перестановок: " + str(MTotal[0]) + " " + str(MTotal[1]) + " " + str(MTotal[2]) + "\n")**
87. KOL = 1500
88. CTotal = [0, 0, 0]
89. MTotal = [0, 0, 0]
91. **for** n **in** range(0, KOL):
92. select\_arr.clear()
93. insert\_arr.clear()
94. bubble\_arr.clear()
96. select\_arr = [random.randint(0, 100) **for** i **in** range(DIM)]
97. **for** i **in** range(0, DIM):
98. bubble\_arr.append(select\_arr[i])
99. insert\_arr.append(select\_arr[i])
101. *# метод Select*
102. count = select\_f.select(select\_arr, DIM)
103. CTotal[0] += count[0]
104. MTotal[0] += count[1]
106. *# метод вставки Insert*
107. count = insert\_f.insert(insert\_arr, DIM)
108. CTotal[1] += count[0]
109. MTotal[1] += count[1]
111. *# метод пузырька Bubble*
112. count = bubble\_f.bubble(bubble\_arr, DIM)
113. CTotal[2] += count[0]
114. MTotal[2] += count[1]
116. **print**("**\n**Случайный массив:")
117. **print**("Проведено экспериментов:", KOL)
118. **print**("Размер массива:", DIM)
119. **print**("Сравнений:", CTotal[0]/KOL, CTotal[1]/KOL, CTotal[2]/KOL)
120. **print("Перестановок:", MTotal[0]/KOL, MTotal[1]/KOL, MTotal[2]/KOL)**
122. file.write("**\n**Случайный массив:**\n**")
123. file.write("Проведено экспериментов: " + str(KOL) + "**\n**")
124. file.write("Размер массива: " + str(DIM) + "**\n**")
125. **file.write("Сравнений: " + str(CTotal[0]/KOL) + " " + str(CTotal[1]/KOL) + " " + str(CTotal[2]/KOL) + "\n")**
126. file.write("Перестановок: " + str(MTotal[0]/KOL) + " " + str(MTotal[1]/KOL) + " " + str(MTotal[2]/KOL))
128. file.close()

Результат исполнения программы:

Метеды: select -- insert -- bubble

Упорядоченный массив:

Размер массива: 40

Сравнений: 780 780 780

Перестановок: 0 0 0

Обратно упорядоченный массив:

Размер массива: 40

Сравнений: 780 780 780

Перестановок: 20 780 780

Случайный массив:

Проведено экспериментов: 1500

Размер массива: 40

Сравнений: 780.0 780.0 780.0

Перестановок: 35.55733333333333 386.1673333333333 386.1673333333333

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*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Найти наибольший элемент столбца матрицы A,*
4. *# для которого сумма абсолютных значений элементов максимальна.*
5. **import numpy as np**
6. **import** random
7. N = random.randint(1, 10)
8. M = random.randint(1, 10)
9. **print(N, M)**
10. A = np.random.randint(0, 100, (N, M))
11. **print**(A)
12. mx = 0
13. indmx = 0
14. **for i in range(M):**
15. **if** np.sum(A[:, i]) > mx:
16. mx = np.sum(A[:, i])
17. indmx = i
18. **print**("Столбец с максимальной суммой элементов - " + str(A[:, indmx]))
19. **print("Сумма этого столбца - " + str(mx) + ", " + "номер этого столбца - " + str(indmx + 1))**
20. stmax = 0
21. **for** b **in** A[:, indmx]:
22. **if** b > stmax:
23. stmax = b
24. **print("Максимальный элемент этого столбца - " + str(stmax))**

Результат:

6 6

[[43 54 57 30 81 5]

[46 79 74 90 16 82]

[95 90 68 56 98 48]

[33 1 20 94 5 0]

[33 79 8 42 5 96]

[80 44 88 31 56 10]]

Столбец с максимальной суммой элементов - [54 79 90 1 79 44]

Сумма этого столбца - 347, номер этого столбца - 2

Максимальный элемент этого столбца – 90

1. *# Задача 2*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Найти наибольшее значение среди средних*
4. *# значений для каждой строки матрицы.*
5. **import numpy as np**
6. **import** random
8. N = random.randint(1, 10)
9. M = random.randint(1, 10)
10. **print(N, M)**
11. A = np.random.randint(0, 100, (N, M))
12. **print**(A)
14. srmax = 0 *# Максимальное среднее значение строки*
15. **indstr = 0 *# Индекс строки с макс. значением***
16. **for** i **in** range(N):
17. **if** np.average(A[i, :]) > srmax:
18. srmax = np.average(A[i, :])
19. indstr = i
20. **print("Строка с наибольшим средним значением - " + str(A[indstr, :]))**
21. **print**("Наибольшее среднее значение - " + str(srmax))

Результат:

5 6

[[13 59 9 58 68 97]

[41 30 5 81 45 20]

[12 34 89 35 92 1]

[77 46 92 48 90 1]

[25 70 63 42 13 4]]

Строка с наибольшим средним значением - [77 46 92 48 90 1]

Наибольшее среднее значение - 59.0

1. *# Задача 3*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Найти наименьший элемент столбца матрицы A,*
4. *# для которого сумма абсолютных значений элементов максимальна.*
5. **import numpy as np**
6. **import** random
8. N = random.randint(1, 10)
9. M = random.randint(1, 10)
10. **print(N, M)**
11. A = np.random.randint(0, 100, (N, M))
12. **print**(A)
13. mx = 0
14. indmx = 0
15. **for i in range(M):**
16. **if** np.sum(A[:, i]) > mx:
17. mx = np.sum(A[:, i])
18. indmx = i
19. **print**("Столбец с максимальной суммой элементов - " + str(A[:, indmx]))
20. **print("Сумма этого столбца - " + str(mx) + ", " + "номер этого столбца - " + str(indmx + 1))**
21. stmin = A[0, indmx]
22. **for** b **in** A[:, indmx]:
23. **if** b < stmin:
24. stmin = b
25. **print("Минимальный элемент этого столбца - " + str(stmin))**

Результат:

5 5

[[53 76 70 51 78]

[87 53 6 54 85]

[51 59 40 99 90]

[12 46 85 59 51]

[26 84 47 24 77]]

Столбец с максимальной суммой элементов - [78 85 90 51 77]

Сумма этого столбца - 381, номер этого столбца - 5

Минимальный элемент этого столбца – 51

1. *# Задача 4*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Найти наименьшее значение среди средних*
4. *# значений для каждой строки матрицы.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(1, 10)
10. **M = random.randint(1, 10)**
11. **print**(N, M)
12. A = np.random.randint(0, 100, (N, M))
13. **print**(A)
15. **srmin = np.average(A[0, :]) *# Минимальное среднее значение строки***
16. indstr = 0 *# Индекс строки с мин. значением*
17. **for** i **in** range(N):
18. **if** np.average(A[i, :]) < srmin:
19. srmin = np.average(A[i, :])
20. **indstr = i**
21. **print**("Строка с наименьшим средним значением - " + str(A[indstr, :]))
22. **print**("Наименьшее среднее значение - " + str(srmin))

Результат:

9 5

[[53 60 52 31 91]

[50 35 0 56 70]

[75 69 45 35 25]

[15 32 30 41 74]

[96 62 22 98 99]

[79 46 37 73 66]

[24 70 57 62 38]

[70 69 32 29 84]

[54 56 99 15 45]]

Строка с наименьшим средним значением - [15 32 30 41 74]

Наименьшее среднее значение - 38.4

1. *# Задача 5*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Определить средние значения по всем строкам и*
4. *# столбцам матрицы. Результат оформить в виде матрицы из N + 1 строк и M*
5. ***# + 1 столбцов.***
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(1, 10)
10. **M = random.randint(1, 10)**
11. **print**(N, M)
12. A = np.random.randint(0, 100, (N, M))
13. **print**(str(A) + "**\n**")
15. **M\_sr = np.mean(A, axis=0)**
16. N\_sr = np.mean(A, axis=1)
18. N\_sr = np.append(N\_sr, None)
20. **A = np.vstack((A, M\_sr))**
21. A = np.hstack((A, N\_sr.reshape(-1, 1)))
23. **print**(A)

Результат:

2 8

[[41 10 64 29 97 65 60 44]

[17 29 60 46 31 48 40 10]]

[[41.0 10.0 64.0 29.0 97.0 65.0 60.0 44.0 51.25]

[17.0 29.0 60.0 46.0 31.0 48.0 40.0 10.0 35.125]

[29.0 19.5 62.0 37.5 64.0 56.5 50.0 27.0 None]]

1. *#* Задача 6
2. # Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами.
3. # Найти сумму элементов всей матрицы.
4. # Определить, какую долю в этой сумме составляет сумма элементов каждого столбца.
5. # Результат оформить в виде матрицы из N + 1 строк и M столбцов**.**
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(1, 10)**
11. M = random.randint(1, 10)
12. **print**(N, M)
13. A = np.random.randint(0, 100, (N, M))
14. **print**(str(A) + "**\n**")
16. sm = np.sum(A)
18. m = []
19. **for** i **in** range(M):
20. **m.append(np.sum(A[:, i]))**
21. **for** i **in** range(M):
22. m[i] = round(m[i] / sm, 2)
23. A = np.vstack((A, m))
24. **print**(A)

Результат:

1 3

[[32 21 85]]

[[32. 21. 85. ]

[ 0.23 0.15 0.62]]

1. *# Задача 7*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами.*
3. *# Найти сумму элементов всей матрицы.*
4. *# Определить, какую долю в этой сумме составляет сумма элементов каждой строки.*
5. ***# Результат оформить в виде матрицы из N строк и M+1 столбцов.***
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(2, 5)**
11. M = random.randint(2, 5)
12. **print**(N, M)
13. A = np.random.randint(0, 100, (N, M))
14. **print**(str(A) + "**\n**")
16. sm = np.sum(A)
18. m = []
19. **for** i **in** range(N):
20. **m.append(np.sum(A[i, :]))**
21. **for** i **in** range(N):
22. m[i] = round(m[i] / sm, 2)
23. m = np.array(m)
24. A = np.hstack((A, m.reshape(-1, 1)))
25. **print(A)**

Результат:

2 2

[[77 69]

[92 26]]

[[77. 69. 0.55]

[92. 26. 0.45]]

1. *# Задача 8*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Определить, сколько отрицательных элементов*
4. *# содержится в каждом столбце и в каждой строке матрицы. Результат*
5. ***# оформить в виде матрицы из N + 1 строк и M + 1 столбцов.***
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(2, 5)**
11. M = random.randint(2, 5)
12. **print**(N, M)
13. A = np.random.randint(-50, 50, (N, M))
14. **print**(str(A) + "**\n**")
16. M\_n = (A < 0).sum(axis=0)
17. N\_n = (A < 0).sum(axis=1)
19. N\_n = np.append(N\_n, None)
21. A = np.vstack((A, M\_n))
22. A = np.hstack((A, N\_n.reshape(-1, 1)))
23. **print**(A)

Результат:

5 5

[[ 26 -47 15 -27 47]

[ 18 15 41 48 49]

[ 32 25 -6 -13 16]

[ 28 9 21 34 -12]

[ 22 -31 -11 25 33]]

[[26 -47 15 -27 47 2]

[18 15 41 48 49 0]

[32 25 -6 -13 16 2]

[28 9 21 34 -12 1]

[22 -31 -11 25 33 2]

[0 2 2 2 1 None]]

1. *# Задача 9*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами.*
3. *# Определить, сколько нулевых элементов*
4. *# содержится в верхних L строках матрицы и в левых К столбцах матрицы.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 10)
10. **M = random.randint(2, 10)**
11. **print**(N, M)
12. A = np.random.randint(-2, 2, (N, M))
13. **print**(str(A) + "**\n**")
15. **L = random.randint(1, N)**
16. K = random.randint(1, M)
17. **print**("L = " + str(L) + " " + "K = " + str(K))
18. L\_n = 0
19. K\_n = 0
21. **for** i **in** A[:L].flat:
22. **if** i == 0:
23. L\_n += 1
25. **for i in A[:, : K].flat:**
26. **if** i == 0:
27. K\_n += 1
29. **print**("Количество нулевых элементов в верхних " + str(L) + " строках матрицы - " + str(L\_n))
30. **print("Количество нулевых элементов в левых " + str(K) + " столбцах матрицы - " + str(K\_n))**

Результат:

8 10

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

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

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

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

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

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

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

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

L = 5 K = 10

Количество нулевых элементов в верхних 5 строках матрицы - 13

Количество нулевых элементов в левых 10 столбцах матрицы – 19

1. *# Задача 10*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Перемножить элементы каждого столбца матрицы*
4. *# с соответствующими элементами K-го столбца.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**(N, M)
12. A = np.random.randint(-50, 50, (N, M))
13. **print**(str(A) + "**\n**")
15. **K = random.randint(0, M)**
16. **print**("K = " + str(K + 1))
17. K\_m = A[:, K].flat
19. **for** i **in** range(M):
20. **if i != K:**
21. **for** k **in** range(N):
22. A[k, i] = A[k, i] \* K\_m[k]
23. **print**(A)

Результат:

2 3

[[-14 4 -8]

[-10 -28 -27]]

K = 3

[[112 -32 -8]

[270 756 -27]]

1. *# Задача 11*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Просуммировать элементы каждой строки*
4. *# матрицы с соответствующими элементами L-й строки.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**(N, M)
12. A = np.random.randint(-50, 50, (N, M))
13. **print**(str(A) + "**\n**")
15. **L = random.randint(0, N)**
16. **print**("L = " + str(L + 1))
17. L\_m = A[L, :].flat
19. **for** i **in** range(N):
20. **if i != L:**
21. **for** k **in** range(M):
22. A[i, k] = A[i, k] + L\_m[k]
23. **print**(A)

Результат:

4 5

[[-49 -21 -42 1 -3]

[ 47 43 47 20 11]

[ -2 15 38 24 -43]

[ 15 6 19 -49 -32]]

L = 1

[[-49 -21 -42 1 -3]

[ -2 22 5 21 8]

[-51 -6 -4 25 -46]

[-34 -15 -23 -48 -35]]

1. *# задача 12*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Разделить элементы каждой строки на элемент*
4. *# этой строки с наибольшим значением.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**(N, M)
12. A = np.random.randint(-50, 50, (N,M))
13. A = np.array(A, float)
15. **print(str(A) + "\n")**
17. **for** i **in** range(N):
18. mx = np.max(A[i, :])
19. **for** k **in** range(M):
20. **if A[i, k] != mx:**
21. A[i, k] = round(A[i, k] / mx, 2)
22. **print**(A)

Результат:

2 3

[[-38. -33. -14.]

[ 22. -41. 23.]]

[[ 2.71 2.36 -14. ]

[ 0.96 -1.78 23. ]]

1. *# Задача 13*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Разделить элементы каждого столбца матрицы на*
4. *# элемент этого столбца с наибольшим значением.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**(N, M)
12. A = np.random.randint(-50, 50, (N, M))
13. A = np.array(A, float)
14. **print**(str(A) + "**\n**")
16. **for** i **in** range(M):
17. mx = np.max(A[:, i])
18. **for** k **in** range(N):
19. **if** A[k, i] != mx:
20. **A[k, i] = round(A[k, i] / mx, 2)**
21. **print**(A)

Результат:

4 3

[[ 43. -28. 40.]

[-33. 31. 43.]

[-25. -27. -6.]

[-43. -18. 48.]]

[[43. -0.9 0.83]

[-0.77 31. 0.9 ]

[-0.58 -0.87 -0.12]

[-1. -0.58 48. ]]

1. *# Задача 14*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Разделить элементы матрицы на элемент матрицы*
4. *# с наибольшим значением.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**(N, M)
12. A = np.random.randint(-50, 50, (N, M))
13. A = np.array(A, float)
14. **print**(str(A) + "**\n**")
16. mx = A.max()
17. A\_n = A.flat
18. **for** i **in** range(len(A\_n)):
19. **if** A\_n[i] != mx:
20. **A\_n[i] = round(A\_n[i] / mx, 2)**
22. A\_n = np.array(A\_n)
23. A\_n = A\_n.reshape(N, M)
24. **print**(A\_n)

Результат:

4 2

[[-35. 14.]

[ 18. -42.]

[-19. 46.]

[ -3. 24.]]

[[-0.76 0.3 ]

[ 0.39 -0.91]

[-0.41 46. ]

[-0.07 0.52]]

1. *# Задача 15*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Все элементы имеют целый тип. Дано целое число H.*
4. *# Определить, какие столбцы имеют хотя бы одно такое число, а какие не имеют.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(3, 5)
10. **M = random.randint(3, 5)**
11. **print**(N, M)
12. A = np.random.randint(1, 5, (N, M))
13. **print**(str(A) + "**\n**")
15. **H = random.randint(1, 4)**
16. **print**("H = " + str(H))
18. im = []
19. nim = []
20. **for i in range(M):**
21. **if** H **in** A[:, i]:
22. im.append(i+1)
23. **else**:
24. nim.append(i+1)
26. **print**("Столбцы, которые имеют хотя бы одно число H - " + str(im))
27. **print**("Столбцы, которые не имеют это число - " + str(nim))

Результат:

5 4

[[2 3 2 4]

[2 4 2 4]

[3 1 1 1]

[3 3 2 4]

[4 2 2 3]]

H = 1

Столбцы, которые имеют хотя бы одно число H - [2, 3, 4]

Столбцы, которые не имеют это число - [1]

1. *# Задача 16*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Исключить из матрицы строку с номером L.*
4. *# Сомкнуть строки матрицы.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**(N, M)
12. A = np.random.randint(-50, 50, (N, M))
13. **print**(str(A) + "**\n**")
15. **L = random.randint(0, N-1)**
16. **print**("L = " + str(L+1))
17. A = np.delete(A, L, axis=0)
19. **print**(A)

Результат:

4 5

[[ -5 26 -2 -22 10]

[ 11 19 -38 -23 -23]

[-40 15 -10 41 24]

[ 4 16 5 21 -50]]

L = 4

[[ -5 26 -2 -22 10]

[ 11 19 -38 -23 -23]

[-40 15 -10 41 24]]

1. *# Задача 17*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Добавить к матрице строку и вставить ее под*
4. *# номером L.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**(N, M)
12. A = np.random.randint(-50, 50, (N, M))
13. **print**(str(A) + "**\n**")
15. **L = random.randint(0, N-1)**
16. **print**("L =", str(L+1))
17. L\_n = np.random.randint(-50, 50, (1, M))
18. **print**("Вставляемая строка -", str(L\_n))
20. **A = np.insert(A, L, L\_n, axis=0)**
21. **print**(A)

Результат:

5 3

[[ 5 -5 -16]

[ 13 12 14]

[ 28 28 29]

[ -8 3 29]

[-14 32 28]]

L = 4

Вставляемая строка - [[-36 -25 43]]

[[ 5 -5 -16]

[ 13 12 14]

[ 28 28 29]

[-36 -25 43]

[ -8 3 29]

[-14 32 28]]

1. *# Задача 18*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Найти сумму элементов, стоящих на главной*
4. *# диагонали, и сумму элементов, стоящих на побочной диагонали (элементы*
5. ***# главной диагонали имеют индексы от [0,0] до [N,N], а элементы побочной***
6. *# диагонали — от [N,0] до [0,N]).*
8. **import** numpy **as** np
9. **import** random
11. N = random.randint(2, 5)
12. M = random.randint(2, 5)
13. **print**(N, M)
14. A = np.random.randint(-50, 50, (N, M))
15. **print(str(A) + "\n")**
17. b = A.trace()
18. c = A[::-1].trace()
20. **print("Сумма элементов главной диагонали =", str(b))**
21. **print**("Сумма элементов побочной диагонали =", str(c))

Результат:

4 4

[[ 18 49 37 -12]

[ 20 21 -49 -39]

[-35 41 -50 7]

[ 25 6 -50 27]]

Сумма элементов главной диагонали = 16

Сумма элементов побочной диагонали = 5

1. *# Задача 19*
2. *# Создать квадратную матрицу A, имеющую N строк и N столбцов со*
3. *# случайными элементами. Определить сумму элементов, расположенных*
4. *# параллельно главной диагонали (ближайшие к главной). Элементы главной*
5. ***# диагонали имеют индексы от [0,0] до [N,N].***
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(2, 5)**
11. **print**("N =", str(N))
12. A = np.random.randint(-50, 50, (N, N))
13. **print**(str(A) + "**\n**")
15. **s = A.diagonal(-1).sum() + A.diagonal(1).sum()**
16. **print**("Сумма элементов, расположенных параллельно главной диагонали =", str(s))

Результат:

N = 5

[[ 10 -34 23 16 47]

[ 42 26 32 1 28]

[ 13 -15 26 -17 23]

[-48 -37 0 -8 41]

[ 19 -28 -48 15 -21]]

Сумма элементов, расположенных параллельно главной диагонали = 64

1. *# Задача 20*
2. *# Создать квадратную матрицу A, имеющую N строк и N столбцов со случайными*
3. *# элементами. Определить произведение элементов,расположенных параллельно*
4. *# побочной диагонали (ближайшие к побочной).*
5. ***# Элементы побочной диагонали имеют индексы от [N,0] до [0,N].***
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(2, 5)**
11. **print**("N =", str(N))
12. A = np.random.randint(-50, 50, (N, N))
13. **print**(str(A) + "**\n**")
14. A = A[::-1]
15. **s = np.prod(A.diagonal(-1)) \* np.prod(A.diagonal(1))**
16. **print**("Произведение элементов, расположенных параллельно главной диагонали =", str(s))

Результат:

N = 4

[[ 26 -8 11 45]

[-30 -3 -21 -37]

[ 31 43 -30 10]

[ 32 -15 49 21]]

Произведение элементов, расположенных параллельно главной диагонали = 17032950

1. *# Задача 21*
2. *# Создать квадратную матрицу A, имеющую N строк и N столбцов со случайными элементами.*
3. *# Каждой паре элементов, симметричныхотносительно главной диагонали*
4. *# (ближайшие к главной), присвоить значения, равные полусумме этих*
5. ***# симметричных значений (элементы главной диагонали имеют индексы от [0,0] до [N,N]).***
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(2, 5)**
11. **print**("N =", str(N))
12. A = np.random.randint(-50, 50, (N, N))
13. **print**(str(A) + "**\n**")
15. **for i in range(1, N-1):**
16. b = (A[i+1, i-1] + A[i-1, i+1])/2
17. A[i+1, i-1] = b
18. A[i-1, i+1] = b
19. **print**(A)

Результат:

N = 5

[[ -3 29 23 9 9]

[ 7 8 37 -24 -45]

[ 29 5 19 -45 -15]

[ -3 -36 17 -36 -38]

[-38 49 8 -41 -7]]

[[ -3 29 26 9 9]

[ 7 8 37 -30 -45]

[ 26 5 19 -45 -3]

[ -3 -30 17 -36 -38]

[-38 49 -3 -41 -7]]

1. *# Задача 22*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Исходная матрица состоит из нулей и единиц.*
4. *# Добавить к матрице еще один столбец, каждый элемент которого делает*
5. ***# количество единиц в каждой строке чётным.***
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(2, 5)**
11. M = random.randint(2, 5)
12. **print**("N =", str(N), ",", "M =", str(M))
13. A = np.random.randint(0, 2, (N, M))
14. **print**(str(A) + "**\n**")
16. New\_s = []
17. **for** i **in** range(N):
18. **if** A[i, :].sum() % 2 == 0:
19. New\_s.append(0)
20. **else:**
21. New\_s.append(1)
22. A = np.insert(A, M, New\_s, axis=1)
23. **print**(A)

Результат:

N = 4 , M = 2

[[0 1]

[1 0]

[0 1]

[0 1]]

[[0 1 1]

[1 0 1]

[0 1 1]

[0 1 1]]

1. *# Задача 23*
2. *# Создать квадратную матрицу A, имеющую N строк и N столбцов со*
3. *# случайными элементами. Найти сумму элементов, расположенных выше*
4. *# главной диагонали, и произведение элементов, расположенных выше*
5. ***# побочной диагонали (элементы главной диагонали имеют индексы от [0,0]***
6. *# до [N,N], а элементы побочной диагонали — от [N,0] до [0,N]).*
8. **import** numpy **as** np
9. **import** random
11. N = random.randint(2, 5)
12. **print**("N =", str(N))
13. A = np.random.randint(-50, 50, (N, N))
14. **print**(str(A) + "**\n**")
16. R\_sum = []
17. **for** i **in** range(1, N):
18. a = (np.diagonal(A, i).tolist())
19. **for** k **in** a:
20. **R\_sum.append(k)**
21. **print**("Элементы выше главной диагонали -", str(R\_sum))
22. **print**("Сумма элементов выше главное диагонали =", str(np.sum(R\_sum)), "**\n**")
24. R\_pr = []
25. **A = A[::-1] *# Отражение матрицы***
26. **for** i **in** range(-1, -N, -1):
27. a = (np.diagonal(A, i).tolist())
28. **for** k **in** a:
29. R\_pr.append(k)
30. **print("Элементы выше побочной диагонали -", str(R\_pr))**
31. **print**("Произведение элементов выше побочной диагонали =", str(np.prod(R\_pr)))

Результат:

N = 4

[[-23 -34 -38 -30]

[-17 -11 17 8]

[ 3 5 35 16]

[-47 6 -36 38]]

Элементы выше главной диагонали - [-34, 17, 16, -38, 8, -30]

Сумма элементов выше главное диагонали = -61

Элементы выше побочной диагонали - [3, -11, -38, -17, -34, -23]

Произведение элементов выше побочной диагонали = -16670676

1. *# Задача 24*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Дан номер строки L и номер столбца K, при*
4. *# помощи которых исходная матрица разбивается на четыре части. Найти*
5. ***# сумму элементов каждой части.***
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(5, 8)**
11. M = random.randint(5, 8)
12. **print**("N =", str(N), " - ", "M =", str(M))
13. A = np.random.randint(-50, 50, (N, M))
14. **print**(str(A) + "**\n**")
16. L = random.randint(2, N-2)
17. K = random.randint(2, M-2)
18. **print**("L =", str(L+1), "K =", str(K+1))
19. M\_n = [A[:L, :K], A[:L, K:], A[L:, :K], A[L:, K:]]
20. **for i in range(len(M\_n)):**
21. **print**(M\_n[i], "Сумма элементов =", str(np.sum(M\_n[i])), "**\n**")

Результат:

N = 5 - M = 8

[[-39 -20 48 11 5 42 10 16]

[ 16 -27 -5 36 15 -48 33 14]

[ 31 -2 -35 13 3 -35 -46 -46]

[-20 47 -20 32 5 -11 -8 2]

[ 1 3 -32 43 36 33 13 19]]

L = 3 K = 5

[[-39 -20 48 11]

[ 16 -27 -5 36]] Сумма элементов = 20

[[ 5 42 10 16]

[ 15 -48 33 14]] Сумма элементов = 87

[[ 31 -2 -35 13]

[-20 47 -20 32]

[ 1 3 -32 43]] Сумма элементов = 61

[[ 3 -35 -46 -46]

[ 5 -11 -8 2]

[ 36 33 13 19]] Сумма элементов = -35

1. *# Задача 25*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами.*
3. *# Определить, сколько нулевых элементов*
4. *# содержится в каждом столбце и в каждой строке матрицы. Результат*
5. ***# оформить в виде матрицы из N + 1 строк и M + 1 столбцов.***
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(2, 5)**
11. M = random.randint(2, 5)
12. **print**("N =", str(N), " - ", "M =", str(M))
13. A = np.random.randint(-1, 1, (N, M))
14. **print**(str(A) + "**\n**")
16. M\_n = (A == 0).sum(axis=0)
17. N\_n = (A == 0).sum(axis=1)
19. N\_n = np.append(N\_n, None)
21. A = np.vstack((A, M\_n))
22. A = np.hstack((A, N\_n.reshape(-1, 1)))
23. **print**(A)

Результат:

N = 4 - M = 5

[[-1 0 0 -1 -1]

[ 0 -1 0 0 0]

[-1 -1 0 0 -1]

[ 0 -1 0 -1 0]]

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

[0 -1 0 0 0 4]

[-1 -1 0 0 -1 2]

[0 -1 0 -1 0 3]

[2 1 4 2 2 None]]

1. *# Задача 26*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Дан номер строки L и номер столбца K, при*
4. *# помощи которых исходная матрица разбивается на четыре части. Найти*
5. ***# среднее арифметическое элементов каждой части.***
7. **import** numpy **as** np
8. **import** random
10. **N = random.randint(5, 8)**
11. M = random.randint(5, 8)
12. **print**("N =", str(N), " - ", "M =", str(M))
13. A = np.random.randint(-50, 50, (N, M))
14. **print**(str(A) + "**\n**")
16. L = random.randint(2, N-2)
17. K = random.randint(2, M-2)
18. **print**("L =", str(L+1), "K =", str(K+1))
19. M\_n = [A[:L, :K], A[:L, K:], A[L:, :K], A[L:, K:]]
20. **for i in range(len(M\_n)):**
21. **print**(M\_n[i], "Среднее арифметическое элементов =", str(np.sum(M\_n[i])/len(M\_n[i])), "**\n**")

Результат:

N = 6 - M = 6

[[-12 -7 32 13 -50 -4]

[ 36 13 48 -49 34 47]

[ -3 40 13 -26 -27 -5]

[ 37 47 5 31 10 -37]

[ 47 49 20 38 -49 24]

[-11 -49 20 4 -41 -25]]

L = 3 K = 3

[[-12 -7]

[ 36 13]] Среднее арифметическое элементов = 15.0

[[ 32 13 -50 -4]

[ 48 -49 34 47]] Среднее арифметическое элементов = 35.5

[[ -3 40]

[ 37 47]

[ 47 49]

[-11 -49]] Среднее арифметическое элементов = 39.25

[[ 13 -26 -27 -5]

[ 5 31 10 -37]

[ 20 38 -49 24]

[ 20 4 -41 -25]] Среднее арифметическое элементов = -11.25

1. *# Задача 27*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами.*
3. *# Все элементы имеют целый тип. Дано целое число H. Определить, какие строки имеют хотя бы одно*
4. *# такое число, а какие не имеют.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**("N =", str(N), " - ", "M =", str(M))
12. A = np.random.randint(1, 4, (N, M))
13. **print**(str(A) + "**\n**")
15. **im = []**
16. neim = []
17. H = random.randint(1, 4)
18. **print**("H =", str(H))
19. **for** i **in** range(N):
20. **if H in A[i, :]:**
21. im.append(i+1)
22. **else**:
23. neim.append(i+1)
24. **print**("Строки,которые имеют хотя бы одно число H -", str(im))
25. **print("Строки, которые не имеют число H -", str(neim))**

Результат:

N = 4 - M = 2

[[2 1]

[2 3]

[2 2]

[1 3]]

H = 1

Строки,которые имеют хотя бы одно число H - [1, 4]

Строки, которые не имеют число H - [2, 3]

1. *# Задача 28*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Исключить из матрицы столбец с номером K.*
4. *# Сомкнуть столбцы матрицы.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**("N =", str(N), " - ", "M =", str(M))
12. A = np.random.randint(1, 4, (N, M))
13. **print**(str(A) + "**\n**")
15. **K = random.randint(1, M-1)**
16. **print**("K =", str(K+1))
18. A = np.delete(A, K, axis=1)
19. **print**(A)

Результат:

N = 2 - M = 4

[[2 1 3 2]

[2 3 2 1]]

K = 4

[[2 1 3]

[2 3 2]]

1. *# Задача 29*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Добавить к матрице столбец чисел и вставить его*
4. *# под номером K.*
6. **import** numpy **as** np
7. **import** random
9. N = random.randint(2, 5)
10. **M = random.randint(2, 5)**
11. **print**("N =", str(N), " - ", "M =", str(M))
12. A = np.random.randint(-50, 50, (N, M))
13. **print**(str(A) + "**\n**")
15. **K = random.randint(0, M-1)**
16. **print**("K =", str(K+1))
17. K\_n = np.random.randint(-50, 50, (1, N))
18. **print**("Вставляемый столбец -", str(K\_n))
20. **A = np.insert(A, K, K\_n, axis=1)**
21. **print**(A)

Результат:

N = 3 - M = 4

[[ 19 -6 -48 -37]

[-46 -23 -29 -3]

[-48 -42 -18 23]]

K = 3

Вставляемый столбец - [[ -9 0 -16]]

[[ 19 -6 -9 -48 -37]

[-46 -23 0 -29 -3]

[-48 -42 -16 -18 23]]

1. *# Задача 30*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Добавить к элементам каждого столбца такой*
4. *# новый элемент, чтобы сумма положительных элементов стала бы равна*
5. ***# модулю суммы отрицательных элементов. Результат оформить в виде***
6. *# матрицы из N + 1 строк и M столбцов.*
8. **import** numpy **as** np
9. **import** random
11. N = random.randint(2, 5)
12. M = random.randint(2, 5)
13. **print**("N =", str(N), " - ", "M =", str(M))
14. A = np.random.randint(-50, 50, (N, M))
15. **print(str(A) + "\n")**
17. M\_n = np.sum(A, axis=0) \* (-1)
18. A = np.vstack((A, M\_n))
19. **print**(A)

Результат:

N = 4 - M = 4

[[ 48 -10 41 -5]

[ 8 9 49 28]

[ 16 -18 -14 -23]

[ 24 16 -25 11]]

[[ 48 -10 41 -5]

[ 8 9 49 28]

[ 16 -18 -14 -23]

[ 24 16 -25 11]

[-96 3 -51 -11]]

1. *# Задача 31*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Добавить к элементам каждой строки такой новый*
4. *# элемент, чтобы сумма положительных элементов стала бы равна модулю*
5. ***# суммы отрицательных элементов. Результат оформить в виде матрицы из N***
6. *# строк и M + 1 столбцов.*
8. **import** numpy **as** np
9. **import** random
11. N = random.randint(2, 5)
12. M = random.randint(2, 5)
13. **print**("N =", str(N), " - ", "M =", str(M))
14. A = np.random.randint(-50, 50, (N, M))
15. **print(str(A) + "\n")**
17. M\_n = np.sum(A, axis=1) \* (-1)
18. A = np.hstack((A, M\_n.reshape(-1, 1)))
19. **print**(A)

Результат:

N = 5 - M = 5

[[ 19 -30 46 12 39]

[ 47 -43 -5 11 18]

[-46 42 19 -15 4]

[ 28 -37 -7 -31 28]

[-44 35 -25 8 -24]]

[[ 19 -30 46 12 39 -86]

[ 47 -43 -5 11 18 -28]

[-46 42 19 -15 4 -4]

[ 28 -37 -7 -31 28 19]

[-44 35 -25 8 -24 50]]

# **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:

1. Было создан файл формата csv, чтобы записать в него исходные матрицы, файл выглядит следующим образом:
2. Был оптимизирован код, таким образом, чтобы чтение происходило из файла с исходными матрицами.

Был оптимизирован код, таким образом, чтобы результат полученный входе работы кода был записан в новый файл формата csv.

Код программы:

**import**numpy

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

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

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

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

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

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

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

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

print(data)

matrix\_list = []

matrix = []

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

first\_col = row[0]

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

matrix\_list.append(matrix)

matrix = []

**continue**

mask = ~numpy.isnan(row)

matrix.append(row[mask])

matrix\_list.append(matrix)

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

f.truncate()

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

M = numpy.array(matrix)

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

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

print(**"Исходная система:"**)

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

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

print(**"Решаем:"**)

print(slv)

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

f.close()

Результат:

Исходная система:

( 1.83 ) \* ( X1) = ( 0.71)

1.34 ) \* ( X1) = ( 0.71)

1.74 ) \* ( X1) = ( 0.71)

1.47 ) \* ( X1) = ( 0.71)

1.54 ) \* ( X1) = ( 0.71)

( 1.55 ) \* ( X2) = ( 0.38)

1.13 ) \* ( X2) = ( 0.38)

1.15 ) \* ( X2) = ( 0.38)

1.26 ) \* ( X2) = ( 0.38)

1.44 ) \* ( X2) = ( 0.38)

( 1.04 ) \* ( X3) = ( 0.76)

1.76 ) \* ( X3) = ( 0.76)

1.83 ) \* ( X3) = ( 0.76)

1.25 ) \* ( X3) = ( 0.76)

1.06 ) \* ( X3) = ( 0.76)

( 1.29 ) \* ( X4) = ( 0.38)

1.51 ) \* ( X4) = ( 0.38)

1.31 ) \* ( X4) = ( 0.38)

1.13 ) \* ( X4) = ( 0.38)

1.50 ) \* ( X4) = ( 0.38)

( 1.42 ) \* ( X5) = ( 0.16)

1.95 ) \* ( X5) = ( 0.16)

1.84 ) \* ( X5) = ( 0.16)

1.31 ) \* ( X5) = ( 0.16)

1.55 ) \* ( X5) = ( 0.16)

Решаем:

[-4.89703872 -3.74080951 2.38835525 3.43916422 3.55386759]

Исходная система:

( 1.05 ) \* ( X1) = ( 0.68)

1.20 ) \* ( X1) = ( 0.68)

1.99 ) \* ( X1) = ( 0.68)

( 1.15 ) \* ( X2) = ( 0.59)

1.02 ) \* ( X2) = ( 0.59)

1.22 ) \* ( X2) = ( 0.59)

( 1.37 ) \* ( X3) = ( 0.77)

1.93 ) \* ( X3) = ( 0.77)

1.16 ) \* ( X3) = ( 0.77)

Решаем:

[0.26207656 0.14220195 0.11767702]

Исходная система:

( 1.61 ) \* ( X1) = ( 0.33)

1.07 ) \* ( X1) = ( 0.33)

1.18 ) \* ( X1) = ( 0.33)

1.03 ) \* ( X1) = ( 0.33)

1.35 ) \* ( X1) = ( 0.33)

( 1.09 ) \* ( X2) = ( 0.84)

1.31 ) \* ( X2) = ( 0.84)

1.78 ) \* ( X2) = ( 0.84)

1.72 ) \* ( X2) = ( 0.84)

1.75 ) \* ( X2) = ( 0.84)

( 1.70 ) \* ( X3) = ( 0.23)

1.78 ) \* ( X3) = ( 0.23)

1.45 ) \* ( X3) = ( 0.23)

1.73 ) \* ( X3) = ( 0.23)

1.93 ) \* ( X3) = ( 0.23)

( 1.71 ) \* ( X4) = ( 0.95)

1.54 ) \* ( X4) = ( 0.95)

1.20 ) \* ( X4) = ( 0.95)

1.14 ) \* ( X4) = ( 0.95)

1.82 ) \* ( X4) = ( 0.95)

( 1.40 ) \* ( X5) = ( 0.72)

1.57 ) \* ( X5) = ( 0.72)

1.68 ) \* ( X5) = ( 0.72)

1.50 ) \* ( X5) = ( 0.72)

1.84 ) \* ( X5) = ( 0.72)

Решаем:

[-0.73925867 -1.78985575 -0.17557436 -1.08824454 3.52846054]

Исходная система:

( 1.10 ) \* ( X1) = ( 0.40)

1.20 ) \* ( X1) = ( 0.40)

1.36 ) \* ( X1) = ( 0.40)

1.12 ) \* ( X1) = ( 0.40)

1.41 ) \* ( X1) = ( 0.40)

1.79 ) \* ( X1) = ( 0.40)

( 1.98 ) \* ( X2) = ( 0.58)

1.04 ) \* ( X2) = ( 0.58)

1.10 ) \* ( X2) = ( 0.58)

1.28 ) \* ( X2) = ( 0.58)

1.22 ) \* ( X2) = ( 0.58)

1.94 ) \* ( X2) = ( 0.58)

( 1.07 ) \* ( X3) = ( 0.45)

1.52 ) \* ( X3) = ( 0.45)

1.06 ) \* ( X3) = ( 0.45)

1.65 ) \* ( X3) = ( 0.45)

1.31 ) \* ( X3) = ( 0.45)

1.79 ) \* ( X3) = ( 0.45)

( 1.60 ) \* ( X4) = ( 0.19)

1.24 ) \* ( X4) = ( 0.19)

1.95 ) \* ( X4) = ( 0.19)

1.74 ) \* ( X4) = ( 0.19)

1.32 ) \* ( X4) = ( 0.19)

1.99 ) \* ( X4) = ( 0.19)

( 1.93 ) \* ( X5) = ( 0.09)

1.37 ) \* ( X5) = ( 0.09)

1.16 ) \* ( X5) = ( 0.09)

1.80 ) \* ( X5) = ( 0.09)

1.80 ) \* ( X5) = ( 0.09)

1.64 ) \* ( X5) = ( 0.09)

( 1.15 ) \* ( X6) = ( 0.11)

1.18 ) \* ( X6) = ( 0.11)

1.96 ) \* ( X6) = ( 0.11)

1.61 ) \* ( X6) = ( 0.11)

1.37 ) \* ( X6) = ( 0.11)

1.51 ) \* ( X6) = ( 0.11)

Решаем:

[ 1.54007355 4.54116669 0.79426157 -2.46366444 -1.87049491 -1.35585177]

Исходная система:

( 1.78 ) \* ( X1) = ( 0.26)

1.19 ) \* ( X1) = ( 0.26)

( 1.68 ) \* ( X2) = ( 0.28)

1.52 ) \* ( X2) = ( 0.28)

Решаем:

[0.08776897 0.08720272]

# ВЫВОД

В связи с наблюдаемым 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