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. 2022.123.ПЗ КР

|  |  |
| --- | --- |
|  |  |
| Рецензент, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_/    «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. | Руководитель, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_/    «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. |
| Нормоконтролёр, \_\_\_\_\_\_\_\_\_\_\_\_\_\_    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_/    «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. | Волков Егор Андреевич ЭУ - 142    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ |
|  | Работа защищена с оценкой  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ |

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Челябинск 2019

# Аннотация

Волков Е. A. – Челябинск: ЮУрГУ, ЭиУ - 142, 2019. - 44с.

Данная курсовая работа состоит из трех частей.

В рамках первой части этой курсовой работы были написаны и исследованы на эффективность 3 основные метода сортировки: сортировка включением (insert), обменная сортировка (bubble) и сортировка выбором (select).

Во второй части курсовой работы была рассмотрена библиотека numpy и при помощи её решена 31 задача на работу с матрицами.

Третья часть курсовой работы состоит из решения систем линейных уравнений методом Гаусса.

Оглавление

[Аннотация 2](#_Toc11655696)

[Введение 3](#_Toc11655697)

[Курсовая работа — часть 1: "Методы сортировки" 4](#_Toc11655698)

[1.1. Алгоритмы сортировки. 4](#_Toc11655699)

[1.1.1 Сортировка включением (insert). 4](#_Toc11655700)

[1.1.2 Обменная сортировка (bubble) 6](#_Toc11655701)

[1.1.3 Сортировка выбором (select) 7](#_Toc11655702)

[1.2 Анализ эффективности. 8](#_Toc11655703)

[Курсовая работа — часть 2: Матричная математика и работа с пакетами. 12](#_Toc11655704)

[2.1 Примеры использования библиотек. 12](#_Toc11655705)

[2.2 Решение задач с использованием библиотеки numpy. 16](#_Toc11655706)

[Курсовая работа — часть 3: Решение слау 40](#_Toc11655707)

[Заключение. 42](#_Toc11655708)

# 

# Введение

Актуальность темы. На данный момент одним из самых популярных языков программирования является python. В данной курсовой работе рассмотрено его применение для сортировки массивов, также рассмотрено применение одной из его библиотек для решения примеров линейной алгебры и систем линейных уравнений методом Гаусса.

В постоянно изменяющимся мире, в особенности мира программирования, является очень гибкость, то есть способность разными методами решить ту или иную задачу. Благодаря разнообразию библиотек при помощи python мы можем решать множество различных задач, что очень важно бля программиста.

Цели работы:

1. Познакомиться ближе с языком программирования python.
2. Методом анализа выявить наиболее эффективный метод сортировки.
3. Познакомиться с библиотекой numpy и с её помощью решить 31 задачу.
4. При помощи библиотеки numpy решить системы линейных уравнений методом Гаусса.

# Курсовая работа — часть 1: "Методы сортировки"

## 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 включением (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, N):
2. alg\_count = [0, 0]
3. **for** i **in** range(N - 1):
4. m = arr[i]
5. **j = i - 1**
6. alg\_count[0] += 1
7. **while** j >= 0 **and** m < arr[j]:
8. arr[j + 1] = arr[j]
9. alg\_count[1] += 1
10. **j -= 1**
11. arr[j + 1] = m
12. **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, N):
2. alg\_count = [0, 0]
3. **for** i **in** range(N - 1):
4. alg\_count[0] += 1
5. **j = 0**
6. **for** j **in** range(N - 1 - i):
7. **if** arr[j] > arr[j + 1]:
8. arr[j], arr[j + 1] = arr[j + 1], arr[j]
9. j += 1
10. **alg\_count[1] += 1**
11. i += 1
12. **return** alg\_count

### 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.

1. **def** select(arr, N):
2. alg\_count = [0, 0]
3. **for** i **in** range(N - 1):
4. m = i
5. **j = i + 1**
6. **while** j < N:
7. alg\_count[0] += 1
8. **if** arr[j] < arr[m]:
9. m = j
10. **j += 1**
11. arr[i], arr[m] = arr[m], arr[i]
12. alg\_count[1] += 1
13. i += 1
14. **return** alg\_count

## Анализ эффективности.

При помощи программы замеряем количество сравнений и перестановок для каждого метода сортировки, которые мы применяем к упорядоченному массиву, обратно упорядоченному массиву, и исследуем количество перестановок со случайными массивами. Конечные данные записываем 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("Упорядоченный массив:**\n** ")
44. file.write("Размер массива: " + str(DIM) + "**\n**")
45. **file.write("Сравнений: " + str(CTotal[0]) + " " + str(CTotal[1]) + " " + str(CTotal[2]) + "\n")**
46. file.write("Перестановок: " + str(MTotal[0]) + " " + str(MTotal[1]) + " " + str(MTotal[2]) + "**\n**")
48. select\_arr.clear()
49. bubble\_arr.clear()
50. **insert\_arr.clear()**
52. **for** i **in** range(DIM, 0, -1):
53. bubble\_arr.append(i)
54. insert\_arr.append(i)
55. **select\_arr.append(i)**
57. **print**("Обратно упорядоченный массив: ")
58. **print**(bubble\_arr)
60. ***# метод Select***
61. count = select\_f.select(select\_arr, DIM)
62. **print**("Обратно упорядоченный массив: результат")
63. **print**(select\_arr)
65. **CTotal[0] = count[0]**
66. MTotal[0] = count[1]
68. *# метод вставки Insert*
69. count = insert\_f.insert(insert\_arr, DIM)
70. **CTotal[1] = count[0]**
71. MTotal[1] = count[1]
73. *# метод пузырька Bubble*
74. count = bubble\_f.bubble(bubble\_arr, DIM)
75. **CTotal[2] = count[0]**
76. MTotal[2] = count[1]
77. **print**("Размер массива:", DIM)
78. **print**("Сравнений:", CTotal[0], CTotal[1], CTotal[2])
79. **print**("Перестановок:", MTotal[0], MTotal[1], MTotal[2])
81. file.write("Методы: select -- insert -- bubble**\n**")
82. file.write("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**("Случайный массив: **\n**")
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()

Результаты тестирования методов сортировки.

Упорядоченный массив:  
 Размер массива: 40  
Сравнений: 780 39 39  
Перестановок: 39 0 780  
Методы: select -- insert -- bubble  
Обратно упорядоченный массив:  
 Размер массива: 40  
Сравнений: 780 39 39  
Перестановок: 39 741 780  
  
Случайный массив:  
Проведено экспериментов: 1500  
Размер массива: 40  
Сравнений: 780.0 39.0 39.0  
Перестановок: 39.0 365.4633333333333 780.0

Исходя из полученный данных можем сделать вывод, что наиболее эффективным методом сортировки является сортировка выбором (select).

# Курсовая работа — часть 2: Матричная математика и работа с пакетами.

## Примеры использования библиотек.

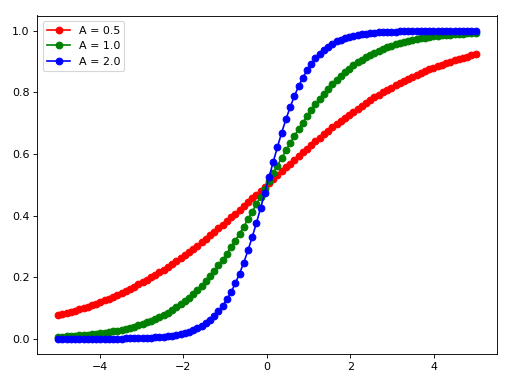
Ниже приведены примеры кодов по использованию библиотек python и возможности их применения.

1. *# имя проекта: numpy-example*
2. *# имя файла: example\_1.py*
3. *# номер версии: 1.011*
4. *# автор и его учебная группа: Е. Волков, ЭУ-142*
5. ***# дата создания: 20.03.2019***
6. *# дата последней модификации: 25.03.2019*
7. *# связанные файлы: пакеты numpy, matplotlib*
8. *# описание: построение графика сигма-функции*
9. *# версия Python: 3.6*
10. **import matplotlib.pyplot as plt**
11. **import** numpy **as** np
13. x = np.linspace(-5, 5, 100)

16. **def** sigmoid(alpha):
17. **return** 1 / (1 + np.exp(- alpha \* x))

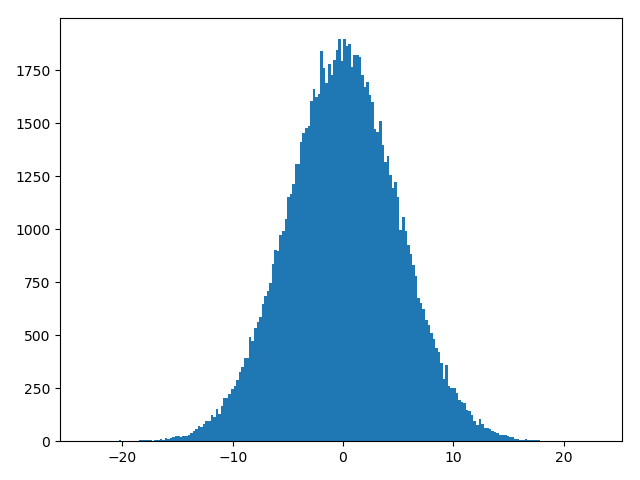
20. **dpi = 80**
21. fig = plt.figure(dpi=dpi, figsize=(512 / dpi, 384 / dpi))
22. plt.plot(x, sigmoid(0.5), 'ro-')
23. plt.plot(x, sigmoid(1.0), 'go-')
24. plt.plot(x, sigmoid(2.0), 'bo-')
25. **plt.legend(['A = 0.5', 'A = 1.0', 'A = 2.0'], loc='upper left')**
26. plt.show()
27. fig.savefig('sigmoid.png')

Результат



1. *# имя проекта: numpy-example*
2. *# номер версии: 1.0*
3. *# имя файла: example\_2.py*
4. *# автор и его учебная группа: Е. Волков, ЭУ-142*
5. ***# дата создания: 20.03.2019***
6. *# дата последней модификации: 25.03.2019*
7. *# связанные файлы: пакеты numpy, matplotlib*
8. *# описание: простейшие статистические вычисления*
9. *# версия Python: 3.6*
10. **import numpy as np**
11. **import** matplotlib.pyplot **as** plt
13. ones = np.ones(50)
14. rnd = np.random.random(50) \* 0.1
15. **samples = ones + rnd**
16. *# Посчитаем среднее:*
17. np.average(samples)
18. np.mean(samples)
19. *# Медиану:*
20. **np.median(samples)**
21. *# Процентили:*
22. np.percentile(samples, 50)
23. np.percentile(samples, 95)
24. np.percentile(samples, 99)
25. ***# Максимум, минимум, peak-to-peak:***
26. samples.max()
27. samples.min()
28. samples.ptp()
29. *# А заодно уж и стандартное отклонение с дисперсией:*
30. **np.std(samples)**
31. np.var(samples)
32. *# Использованная выше функция np.random.random генерирует случайные числа с равномерным распределением.*
33. *# А если мы хотели бы использовать нормальноераспределение? Нет проблем:*
35. **samples = np.random.normal(loc=0, scale=5, size=100000)**
36. plt.hist(samples, 200)
37. plt.show()

Результат.



1. *#автор и его учебная группа: Е. Волков, ЭУ-142*
2. *#дата создания: 20.03.2019*
3. *# дата последней модификации: 25.03.2019*
4. *#связанные файлы: пакеты numpy, matplotlib*
5. ***# описание: линейная алгебра***
6. *#версия Python: 3.6*
7. **import** numpy **as** np
8. **import** numpy.matlib
9. **import** numpy.linalg
10. **m1 = np.arange(1, 10).reshape(3,3)**
11. **print**(m1)
12. m2 = np.identity(3)
13. **print**(m2)
14. *# Транспонируем первую матрицу, а также посчитаем след и детерминант второй:*
15. **m1.transpose()**
16. **print**(m1)
17. m2.trace()
18. **print**(m2)
19. det = np.linalg.det(m2)
20. **print(det)**
21. *# Матрицы можно складывать, умножать на число, умножать на вектор, а также умножать на другую матрицу:*
22. **print**(m1 + m2)
23. **print**(m1 \* 3)
24. m1 + np.array([1,2,3])
25. **print(m1 \* m2)**
26. *# Посчитать матрицу, обратную к данной, можно функцией np.linalg.inv:*
27. m3 = np.matlib.rand(3, 3)
28. (m3 \* np.linalg.inv(m3))
29. **print**(m3)
30. **print((m3 \* np.linalg.inv(m3)).round())**

Результат

[[1 2 3]

[4 5 6]

[7 8 9]]

[[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]]

[[1 2 3]

[4 5 6]

[7 8 9]]

[[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]]

1.0

[[ 2. 2. 3.]

[ 4. 6. 6.]

[ 7. 8. 10.]]

[[ 3 6 9]

[12 15 18]

[21 24 27]]

[[1. 0. 0.]

[0. 5. 0.]

[0. 0. 9.]]

[[0.26462571 0.44905269 0.93966339]

[0.51742193 0.140354 0.58910975]

[0.58209412 0.80455231 0.98531022]]

[[ 1. -0. -0.]

[-0. 1. -0.]

[-0. -0. 1.]]

## Решение задач с использованием библиотеки numpy.

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*
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**")
14. a = 0
15. **for i in range(M):**
16. **if** np.sum(A[:, i]) > a:
17. a = i
18. k = np.max(A[:, a])
19. **print**("Строка с наибольшей суммой - ", str(A[:, a]))
20. **print("Наибольший элемент столбца - ", str(k))**

Результат

[[ 29 41 47]

[-38 -14 -26]]

Строка с наибольшей суммой - [ 47 -26]

Наибольший элемент столбца - 47

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.*
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**")
14. a = 0
15. **for** i **in** range(M):
16. **if np.mean(A[:, i], axis=0) > a:**
17. a = i
18. k = np.mean(A[:, a], axis=0)
19. **print**("Столбец с наибольшим средним значением - ", str(A[:, a]))
20. **print**("Наибольшее значение среди средних значений - ", str(k))

Решение

[[13 11]

[17 -4]

[-42 -49]]

Столбец с наибольшим средним значением - [13 17 -42]

Наибольшее значение среди средних значений - -4.0

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.*
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**")
14. a = 0
15. **for** i **in** range(M):
16. **if np.sum(A[:, i]) > a:**
17. a = i
18. k = np.min(A[:, a])
19. **print**("Строка с наибольшей суммой - ", str(A[:, a]))
20. **print**("Наименьший эллемент столбца - ", str(k))

Результат

[[-37 47 19 21 41]

[ 10 -19 35 -18 16]]

Строка с наибольшей суммой - [41 16]

Наименьший эллемент столбца - 16

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(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**")
14. a = np.average(A[0, :]) *# Минимальное среднее значение строки*
15. **b = 0 *# Индекс строки с мин. значением***
16. **for** i **in** range(N):
17. **if** np.average(A[i, :]) < a:
18. a = np.average(A[i, :])
19. b = i
20. **print("Строка с наименьшим средним значением - ", str(A[b, :]))**
21. **print**("Наименьшее среднее значение - ", str(a))

Результат

[[-29 41 25 18 -26]

[-47 38 -37 -9 -6]

[ 2 25 -20 12 17]]

Строка с наименьшим средним значением - [-47 38 -37 -9 -6]

Наименьшее среднее значение - -12.2

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.***
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(-50, 50, (N, M))
14. **print**(str(A) + "**\n**")
16. M\_sr = np.mean(A, axis=0)
17. N\_sr = np.mean(A, axis=1)
19. N\_sr = np.append(N\_sr, None)
21. A = np.vstack((A, M\_sr))
22. A = np.hstack((A, N\_sr.reshape(-1, 1)))
23. **print**(A)

Результат

[[ 40 -38 5]

[ 7 -3 -20]

[ 35 -26 33]

[ 46 12 -42]

[-33 -12 -8]]

[[40.0 -38.0 5.0 2.3333333333333335]

[7.0 -3.0 -20.0 -5.333333333333333]

[35.0 -26.0 33.0 14.0]

[46.0 12.0 -42.0 5.333333333333333]

[-33.0 -12.0 -8.0 -17.666666666666668]

[19.0 -13.4 -6.4 None]]

1. *# Задача 6*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Найти сумму элементов всей матрицы.*
4. *# Определить, какую долю в этой сумме составляет сумма элементов каждого*
5. ***# столбца. Результат оформить в виде матрицы из N + 1 строк и M 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(-50, 50, (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

[[-34 -5]

[-15 28]

[-21 34]

[-40 2]]

[[-34. -5. ]

[-15. 28. ]

[-21. 34. ]

[-40. 2. ]

[ 2.16 -1.16]]

1. *# Задача 7*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. Найти сумму элементов всей матрицы.*
4. *# Определить, какую долю в этой сумме составляет сумма элементов каждой*
5. ***# строки. 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 =", str(N), " - ", "M =", str(M))
13. A = np.random.randint(-50, 50, (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

[[ 38 46 -33 34]

[ 30 -17 10 31]

[ 6 -31 46 -3]

[-45 -21 -4 -3]]

[[ 38. 46. -33. 34. 1.01]

[ 30. -17. 10. 31. 0.64]

[ 6. -31. 46. -3. 0.21]

[-45. -21. -4. -3. -0.87]]

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 =", str(N), " - ", "M =", str(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

[[ 17 41 -47 -47 18]

[-46 36 -26 -50 4]

[ 49 15 -40 27 30]]

[[17 41 -47 -47 18 2]

[-46 36 -26 -50 4 3]

[49 15 -40 27 30 1]

[1 0 3 2 0 None]]

1. *# Задача 9*
2. *# Создать прямоугольную матрицу A, имеющую N строк и M столбцов со*
3. *# случайными элементами. 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, 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. **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

[[-42 41 -33 18 -43]

[ 48 41 -43 -24 -4]]

L = 2 K = 4

Количество нулевых элементов в верхних 2 строках матрицы - 0

Количество нулевых элементов в левых 4 столбцах матрицы – 0

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 =", 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)
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

[[-41 -3]

[ 9 0]

[-11 -17]

[ -2 -33]]

K = 1

[[-41 123]

[ 9 0]

[-11 187]

[ -2 66]]

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 =", str(N), " - ", "M =", str(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

[[ 24 -35 -24 19 45]

[ -3 23 -5 -45 41]

[-35 -6 -46 9 -11]

[ 28 -9 43 48 17]

[-47 49 -27 -43 -27]]

L = 3

[[-11 -41 -70 28 34]

[-38 17 -51 -36 30]

[-35 -6 -46 9 -11]

[ -7 -15 -3 57 6]

[-82 43 -73 -34 -38]]

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 =", str(N), " - ", "M =", str(M))
12. A = np.random.randint(-50, 50, (N, M))
13. **print**(str(A) + "**\n**")
15. **for i in range(N):**
16. mx = np.max(A[i, :])
17. **for** k **in** range(M):
18. **if** A[i, k] != mx:
19. A[i, k] = round(A[i, k] / mx, 2)
20. **print(A)**

Результат

[[ -3 -44 1]

[-50 49 45]

[-23 -28 43]

[-17 24 5]]

[[ -3 -44 1]

[ -1 49 0]

[ 0 0 43]

[ 0 24 0]]

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 =", str(N), " - ", "M =", str(M))
12. A = np.random.randint(-50, 50, (N, M))
13. **print**(str(A) + "**\n**")
15. **for i in range(M):**
16. mx = np.max(A[:, i])
17. **for** k **in** range(N):
18. **if** A[k, i] != mx:
19. A[k, i] = round(A[k, i] / mx, 2)
20. **print(A)**

Результат

[[-44 -27 -35 8 21]

[ 8 -5 -7 46 43]

[ 14 -12 -34 24 46]

[ 48 19 -12 -12 -33]]

[[ 0 -1 5 0 0]

[ 0 0 -7 46 0]

[ 0 0 4 0 46]

[48 19 1 0 0]]

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 =", str(N), " - ", "M =", str(M))
12. A = np.random.randint(-50, 50, (N, M))
13. **print**(str(A) + "**\n**")
15. **mx = A.max()**
16. A\_n = A.flat
17. **for** i **in** range(len(A\_n)):
18. **if** A\_n[i] != mx:
19. A\_n[i] = round(A\_n[i] / mx, 2)
21. A\_n = np.array(A\_n)
22. A\_n = A\_n.reshape(N, M)
23. **print**(A\_n)

Результат

[[ 49 -44 24]

[ 41 43 3]

[ 16 33 -41]

[-17 -34 30]]

[[49 0 0]

[ 0 0 0]

[ 0 0 0]

[ 0 0 0]]

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(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. **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 -47]

[ 43 25]

[-37 36]]

H = 4

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

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

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 =", str(N), " - ", "M =", str(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

[[ 45 32]

[-19 26]]

L = 2

[[45 32]]

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 =", str(N), " - ", "M =", str(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

[[ 41 -22 -29]

[ 43 -22 14]

[-14 10 6]

[-18 -15 -44]]

L = 2

Вставляемая строка - [[ 37 -42 -49]]

[[ 41 -22 -29]

[ 37 -42 -49]

[ 43 -22 14]

[-14 10 6]

[-18 -15 -44]]

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

2 3

[[-13 32 38]

[-17 29 47]]

Сумма элементов главной диагонали = 16

Сумма элементов побочной диагонали = 15

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 = 3

[[ -4 -11 13]

[ 30 28 47]

[-20 -48 9]]

Сумма элементов, расположенных параллельно главной диагонали = 18

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))

N = 3

[[ 25 -37 -19]

[-36 38 -47]

[-14 31 9]]

Произведение элементов, расположенных параллельно главной диагонали = -1940724

1. *# Задача 21*
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]**
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

[[ -9 -27]

[ 15 -49]]

Элементы выше главной диагонали - [-27]

Сумма элементов выше главное диагонали = -27

Элементы выше побочной диагонали - [-9]

Произведение элементов выше побочной диагонали = -9

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

[[0 1]

[1 0]

[1 0]]

[[0 1 1]

[1 0 1]

[1 0 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

[[ 42 19 -18 13]

[-26 -21 20 -2]

[-46 24 20 5]

[ 25 -17 -3 -49]]

Элементы выше главной диагонали - [19, 20, 5, -18, -2, 13]

Сумма элементов выше главное диагонали = 37

Элементы выше побочной диагонали - [-46, -21, -18, -26, 19, 42]

Произведение элементов выше побочной диагонали = 360766224

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

[[ 4 -21 37 -36 -27]

[ 43 -42 17 -45 32]

[-17 -2 0 7 -31]

[ 6 -29 -20 -21 -44]

[-35 16 35 -12 -37]]

L = 4 K = 4

[[ 4 -21 37]

[ 43 -42 17]

[-17 -2 0]] Сумма элементов = 19

[[-36 -27]

[-45 32]

[ 7 -31]] Сумма элементов = -100

[[ 6 -29 -20]

[-35 16 35]] Сумма элементов = -27

[[-21 -44]

[-12 -37]] Сумма элементов = -114

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(-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

[[ 33 28 1]

[-19 -20 21]

[-40 -24 42]

[-47 15 37]

[-19 42 -21]]

[[33 28 1 0]

[-19 -20 21 0]

[-40 -24 42 0]

[-47 15 37 0]

[-19 42 -21 0]

[0 0 0 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

[[-13 -9 -18 32 47]

[-29 -16 45 21 -40]

[ -3 -8 26 -11 -20]

[ -8 31 -3 -44 47]

[ 47 15 31 -49 29]]

L = 3 K = 4

[[-13 -9 -18]

[-29 -16 45]] Среднее арифметическое элементов = -20.0

[[ 32 47]

[ 21 -40]] Среднее арифметическое элементов = 30.0

[[-3 -8 26]

[-8 31 -3]

[47 15 31]] Среднее арифметическое элементов = 42.666666666666664

[[-11 -20]

[-44 47]

[-49 29]] Среднее арифметическое элементов = -16.0

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

[[1 2 3 2 3]

[2 2 1 1 2]]

H = 2

Строки,которые имеют хотя бы одно число H - [1, 2]

Строки, которые не имеют число H - []

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

[[3 1 1 1 3]

[2 1 3 3 3]

[1 2 1 2 1]

[1 1 2 1 3]]

K = 3

[[3 1 1 3]

[2 1 3 3]

[1 2 2 1]

[1 1 1 3]]

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

[[-42 30 -14 -9 -13]

[ 23 38 14 -3 32]

[-40 -43 18 43 25]]

K = 3

Вставляемый столбец - [[ 45 -33 -5]]

[[-42 30 45 -14 -9 -13]

[ 23 38 -33 14 -3 32]

[-40 -43 -5 18 43 25]]

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

[[-24 -10 25]

[ 20 -7 -18]

[-42 45 11]]

[[-24 -10 25]

[ 20 -7 -18]

[-42 45 11]

[ 46 -28 -18]]

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

[[ 36 -39 40]

[-37 -27 43]

[-35 35 -29]]

[[ 36 -39 40 -37]

[-37 -27 43 21]

[-35 35 -29 29]]

# Курсовая работа — часть 3: Решение слау

Метод Гаусса при решении СЛАУ позволяет ответить на вопросы о совместности или несовместности, определенности или неопределенности системы линейных уравнений, а также отыскать все решения совместной системы. В основе метода лежит идея последовательного исключения неизвестных с помощью подстановок, суть которой состоит в приведении данной системы к другой, равносильной ей, но более простой системе. Это приведение одной системы к другой осуществляется путем элементарных преобразований, которые производятся над уравнениями системы или, что удобнее, над строками расширенной матрицы. Элементарному преобразованию системы линейных уравнений соответствует одноименное элементарное преобразование строк ее расширенной матрицы.

Дано 5 систем.

1. 1.69;1.02;1.59;1.11;0.28
2. 1.08;1.55;1.23;1.65;0.63
3. 1.50;1.34;1.50;1.13;0.09
4. 1.86;1.34;1.46;1.66;0.84
6. 1.11;1.40;1.58;1.04;0.82
7. 1.61;1.49;1.98;1.48;0.39
8. 1.45;1.21;1.12;1.04;0.45
9. 1.38;1.31;1.31;1.86;0.33
11. 1.52;1.93;1.40 1.54;0.99
12. 1.61;1.31;1.25 1.83;0.96
13. 1.81;1.30;1.26 1.61;0.99
14. 1.73;1.03;1.44 1.12;0.29
16. 1.32;1.33;1.56;1.86;1.90;1.84;0.15
17. 1.86;1.57;1.84;1.10;1.91;1.61;0.67
18. 1.93;1.55;1.49;1.29;1.08;1.89;0.37
19. 1.08;1.66;1.80;1.20;1.13;1.17;0.08
20. **1.78;1.79;1.86;1.29;1.72;1.16;0.52**
21. 1.14;1.23;1.05;1.80;1.29;1.39;0.42
23. 1.15;1.46;1.73;1.70;0.84
24. 1.89;1.78;1.54;1.49;0.79
25. **1.09;1.39;1.64;1.02;0.15**
26. 1.62;1.83;1.04;1.18;0.20

Все они записаны в личном файле в формате .csv. Найдем их решение, для этого применим код.

1. **import** numpy **as** np *# импортируем библиотеку*
3. input\_path = "date/"
4. my\_variant = "7"
5. **file\_name = input\_path + "gauss\_" + my\_variant + "\_"**
7. file = open('result-gauss-slv.csv', 'wb+')
8. file.truncate()
9. **for** i **in** range(1, 6):
10. **task\_file = file\_name + str(i) + ".csv"**
11. **print**("**\n\n**Читаем: ", task\_file)
13. *# --- исходные данные читаем из файла task\_file*
14. m = np.genfromtxt(task\_file, delimiter=';')
15. **myA = np.genfromtxt(task\_file, delimiter=';', usecols=(range(len(m - 1))))**
16. myB = np.genfromtxt(task\_file, delimiter=';', usecols=(len(m)))
17. *# --- end of исходные данные*
19. **print**("Задача " + str(i))
20. **print("Матрица A:")**
21. **print**(myA)
22. **print**("Вектор правых частей B:")
23. **print**(myB)
25. **slv = np.linalg.solve(myA, myB)**
27. **print**("Решение задачи " + str(i) + " ", slv)
28. np.savetxt(file, np.array([slv]), delimiter=',')
29. file.close()

Результат

1. 2.103687301841093149e-01,-6.796135015038191618e-01,-4.746532162454872306e-01,1.236379487705473235e+00
2. -6.049391223118242289e-01,1.692913796732238962e+00,-3.420280934024901298e-01,-3.251850981568030763e-01
3. 7.004627634461985108e-01,6.886930645501976311e-01,-1.446476441718549388e+00,4.033675346661465544e-01
4. -8.706420292735096433e-01,3.778932012676440078e+00,-3.384042619121155671e+00,-1.595482287504195851e+00,1.548140759240835163e+00,8.578826085927105938e-01
5. **7.723446247043498358e-01,-1.252588343931595682e+00,-3.173993538355492083e-02,1.079701618790398499e+00**

# Заключение.

1. Мы ближе познакомились с языком программирования python.
2. Методом анализа выявили наиболее эффективный метод сортировки.
3. Изучили библиотеку numpy и с её помощью решили 31 задачу.
4. При помощи библиотеки numpy решили системы линейных уравнений методом Гаусса.

В итоге можно сделать вывод, что python достаточно прост в понимании, а благодаря библиотекам имеет очень большой круг применения .