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 – 380305.2022.07. ПЗ КР

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

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

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019г. «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019г.

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Гайникамалов В.Н./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019г.

Работа защищена с оценкой

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

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019г.

Челябинск 2019

Аннотация

Гайникамалов В.Н.

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-142, 2019

В курсовой работе были раскрыты темы сортировки одномерными массивами, работы с двумерными массивами при помощи библиотеки Numpy и решение систем линейных уравнений методом Гаусса.

Оглавление

[Введение 4](#_Toc10559114)

[Методы сортировки 5](#_Toc10559115)

[Numpy 12](#_Toc10559116)

[Системы линейных уравнений 41](#_Toc10559117)

[Вывод 44](#_Toc10559118)

[Список литературы 45](#_Toc10559119)

# Введение

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Будучи удачно спроектированным языком программирования, Python прекрасно подходит для решения ежедневных реальных задач. Он имеет самый широкий спектр применений: как инструмент управления другими программными компонентами и для реализации самостоятельных программ. Фактически, круг ролей, которые может играть Python как многоцелевой язык программирования, не включает только области встроенных устройств и системного программирования, где ограничения на использование памяти и требования к скорости исполнения настолько велики, что время и удобство написания программы не играют существенной роли, причём можно нанять программистов сколь угодно высокой квалификации.

За счёт чего Python получил столь широкое распространение? Python имеет огромное количество высококачественных уже готовых модулей, распространяемых бесплатно, которые вы можете использовать в любой части программы. В модуле уже реализованы многие нужные вам детали программы. Написание программы с использованием уже готовых модулей можно сравнить со строительством сборного каркасного дома: отдельные детали: фундамент, стены, крыша, коммуникации уже сделаны до вас, вам нужно только выбрать подходящие детали и собрать вместе. Модули подключаются при помощи команды import, которая присутствует в начале каждого примера. Далее я расскажу об этом подробно.

# Методы сортировки

В рамках данной работы мы разберем 3 метода сортировки и выясним какой из них самый эффективный. А сейчас немного теории.

**Простая обменная сортировка**

Сравнивается i-ый элемент списка с i+1 — ым. Если больший из них имеет меньший порядковый номер, то они меняются местами (для сортировки по возрастанию)

1. Таким образом, самый большой элемент сдвигается в конец списка
2. Далее процесс повторяется, но не до конца списка, а до последнего отсортированного элемента



Пример метода:

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

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

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

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

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

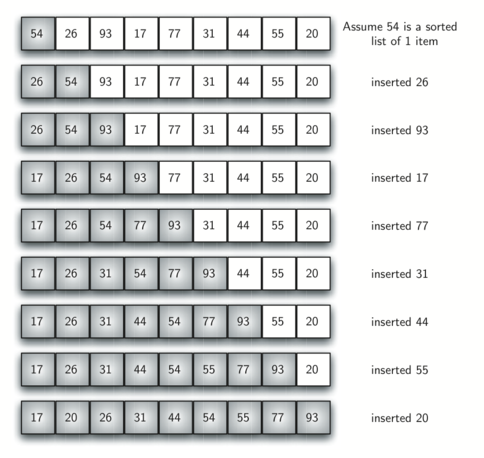
**if arr[j] > arr[j + 1]:**

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

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

**return alg\_count**

**Сортировка включением**, всегда поддерживает отсортированный подсписок в нижних позициях списка. Каждый новый элемент затем «вставляется» обратно в предыдущий подсписок, так что отсортированный подсписок на один элемент больше. Затененные элементы представляют упорядоченные подсписки, поскольку алгоритм выполняет каждый проход.



Пример метода:

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

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

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

**temp = arr[i]**

**j = i - 1**

**while j >= 0:**

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

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

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

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

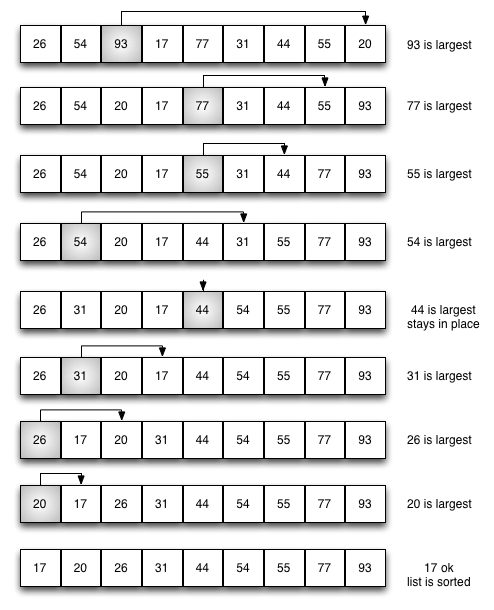
**arr[j] = temp**

**j -= 1**

**return alg\_count**

**Сортировка выбором**

Cортировка выбором ищет наибольшее значение, когда она делает проход, и, после завершения прохода, помещает его в правильное местоположение. Как и в случае с bubble sort, после первого прохода самый большой предмет находится в правильном месте. После второго прохода следующий по величине на месте.



Пример метода:

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

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

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

**m = k**

**i = k + 1**

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

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

**if arr[i] < arr[m]:**

**m = i**

**if k != m:**

**t = arr[k]**

**arr[k] = arr[m]**

**arr[m] = t**

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

**return alg\_count**

Теперь определим какой из данных методов является самым эффективным. Для это был написан код, который создаёт три вида массива и к которым применяются функции сортировок. В каждой функции установлен счётчик перемещений и сравнений, чем меньше число у счётчика, тем эффективней метод.

Код:

**import select**

**import bubble**

**import insert**

**import random**

**dim = 40**

**bubble\_arr = []**

**insert\_arr = []**

**select\_arr = []**

**CTotal = [0, 0, 0]**

**MTotal = [0, 0, 0]**

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

**select\_arr.append(i)**

**bubble\_arr.append(i)**

**insert\_arr.append(i)**

**print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:")**

**print(select\_arr)**

**count = [0, 0]**

**count = select.select(select\_arr, dim)**

**print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:")**

**print(select\_arr)**

**CTotal[0] = count[0]**

**MTotal[0] = count[1]**

**count = [0, 0]**

**count = insert.insert(insert\_arr, dim)**

**CTotal[1] = count[0]**

**MTotal[1] = count[1]**

**count = [0, 0]**

**count = bubble.bubble(bubble\_arr, dim)**

**CTotal[2] = count[0]**

**MTotal[2] = count[1]**

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

**print("Размер массива: ", dim)**

**print("Сравнений: ", CTotal[0], " ", CTotal[1], " ", CTotal[2])**

**print("Перестановок: ", MTotal[0], " ", MTotal[1], " ", MTotal[2])**

**select\_arr.clear()**

**bubble\_arr.clear()**

**insert\_arr.clear()**

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

**select\_arr.append(i)**

**bubble\_arr.append(i)**

**insert\_arr.append(i)**

**print("\nОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Исходный массив")**

**print(select\_arr)**

**count = [0, 0]**

**count = select.select(select\_arr, dim)**

**print("\nОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив")**

**print(select\_arr)**

**CTotal[0] = count[0]**

**MTotal[0] = count[1]**

**count = [0, 0]**

**count = insert.insert(insert\_arr, dim)**

**CTotal[1] = count[0]**

**MTotal[1] = count[1]**

**count = [0, 0]**

**count = bubble.bubble(bubble\_arr, dim)**

**CTotal[2] = count[0]**

**MTotal[2] = count[1]**

**print("Размер массива: ", dim)**

**print("Сравнений: ", CTotal[0], " ", CTotal[1], " ", CTotal[2])**

**print("Перестановок: ", MTotal[0], " ", MTotal[1], " ", MTotal[2])**

**NUM = 1500**

**CTotal.clear()**

**MTotal.clear()**

**CTotal = [0, 0, 0]**

**MTotal = [0, 0, 0]**

**for n in range(0, NUM):**

**select\_arr.clear()**

**bubble\_arr.clear()**

**insert\_arr.clear()**

**select\_arr = [random.randint(0, 100) for i in range(dim)]**

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

**bubble\_arr.append(select\_arr[i])**

**insert\_arr.append(select\_arr[i])**

**count = [0, 0]**

**count = select.select(select\_arr, dim)**

**CTotal[0] += count[0]**

**MTotal[0] += count[1]**

**count = [0, 0]**

**count = insert.insert(insert\_arr, dim)**

**CTotal[1] += count[0]**

**MTotal[1] += count[1]**

**count = [0, 0]**

**count = bubble.bubble(bubble\_arr, dim)**

**CTotal[2] += count[0]**

**MTotal[2] += count[1]**

**print("\nСЛУЧАЙНАЯ РЕАЛИЗАЦИЯ:")**

**print("Проведено экспериментов: ", NUM)**

**print("Размер массива: ", dim)**

**print("Сравнений: ", CTotal[0]/NUM, " ", CTotal[1]/NUM, " ", CTotal[2]/NUM)**

**print("Перестановок: ", MTotal[0]/NUM, " ", MTotal[1]/NUM, " ", MTotal[2]/NUM)**

**Результат:**

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

Размер массива: 40

Сравнений: 780 780 780

Перестановок: 0 0 0

ОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Исходный массив

[40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

ОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]

Размер массива: 40

Сравнений: 780 780 780

Перестановок: 20 780 780

**Вывод:**

Исходя из результатов, можно сделать вывод что самый эффективный метод - это Select, т.к. у него наименьшее число сравнений и перестановок.

# Numpy

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

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

Решим задачи для демонстрации возможностей модуля Numpy.

1. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти наибольший элемент столбца матрицы A,

для которого сумма абсолютных значений элементов максимальна.

Решение:

**import numpy as np**

**N = 7**

**M = 10**

**A = np.random.randint(0, 100, (N, M))**

**print(A)**

**sum = A.sum(axis=0)**

**i = sum.argmax(axis=0)**

**max = A.max(axis=0)**

**max = max[i]**

**print("Наибольшее значение: " + str(max))**

Ответ:

[[ 5 79 43 86 44 46 76 9 45 35]

[ 0 63 10 45 12 1 40 49 77 19]

[15 46 8 82 92 27 18 86 11 46]

[21 87 87 77 53 24 53 76 11 83]

[21 54 91 85 63 32 59 27 43 64]

[60 47 28 40 69 56 78 53 69 75]

[16 22 69 61 3 35 75 4 63 71]]

Наибольшее значение: 86

2. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти наибольшее значение среди средних

значений для каждой строки матрицы.

Решение:

**import numpy as np**

**N = 7**

**M = 10**

**A = np.random.randint(0, 100, (N, M))**

**print(A)**

**Average = A.mean(axis=1)**

**i = Average.argmax(axis=0)**

**max = Average.max(axis=0)**

**print("Наибольшее среднее значение: " + str(max))**

Ответ:

[[34 0 31 33 72 38 60 20 29 72]

[95 45 49 27 36 77 9 57 85 99]

[11 70 66 77 30 91 1 76 20 35]

[59 32 50 7 85 47 57 31 25 61]

[49 3 19 31 72 20 70 8 76 62]

[10 43 13 88 18 77 59 87 33 75]

[48 91 78 68 57 71 99 80 72 9]]

Наибольшее среднее значение: 67.3

3. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти наименьший элемент столбца матрицы A,

для которого сумма абсолютных значений элементов максимальна.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**A = np.random.randint(0, 100, (N, M))**

**print(A)**

**sum = A.sum(axis=0)**

**i = sum.argmin(axis=0)**

**min = A.min(axis=0)**

**min = min[i]**

**print("Наименьшее значение: " + str(min))**

Ответ

[[52 55 76 70 88 60 5 35 83 28]

[ 0 9 60 47 46 16 91 67 16 53]

[78 55 80 36 4 91 83 4 51 66]

[76 18 96 92 43 21 34 47 39 48]

[89 66 84 57 47 37 70 41 56 68]

[80 77 54 76 14 0 8 19 13 57]

[46 2 59 97 49 6 85 30 43 31]]

Наименьшее значение: 0

4. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти наименьшее значение среди средних

значений для каждой строки матрицы.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**A = np.random.randint(0, 100, (N, M))**

**print(A)**

**Average = A.mean(axis=1)**

**i = Average.argmin(axis=0)**

**min = Average.min(axis=0)**

**print("Наименьшее среднее значение: " + str(min))**

Ответ

[[68 35 72 85 16 58 66 35 7 73]

[79 30 52 25 80 15 4 30 16 82]

[75 24 77 62 88 99 74 12 72 69]

[75 38 51 24 42 36 55 92 97 85]

[49 54 95 63 0 44 37 94 89 99]

[87 14 49 34 2 11 54 32 21 32]

[90 14 1 29 57 72 82 0 2 17]]

Наименьшее среднее значение: 33.6

5. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Определить средние значения по всем строкам и

столбцам матрицы. Результат оформить в виде матрицы из N + 1 строк и M

+ 1 столбцов.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**A = np.random.randint(0, 100, (N, M))**

**print(A)**

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

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

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

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

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

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

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

Ответ

[[66 44 58 63 22 85 18 61 99 18]

[71 96 13 59 42 28 35 54 81 32]

[19 71 96 48 62 91 6 77 78 81]

[59 76 67 14 4 97 24 49 94 23]

[71 40 32 86 0 32 61 21 9 77]

[21 86 64 18 14 52 68 44 96 25]

[28 8 51 50 59 38 51 59 75 56]]

Новая матрица:

[[66. 44. 58. 63. 22. 85.

18. 61. 99. 18. 53.4 ]

[71. 96. 13. 59. 42. 28.

35. 54. 81. 32. 51.1 ]

[19. 71. 96. 48. 62. 91.

6. 77. 78. 81. 62.9 ]

[59. 76. 67. 14. 4. 97.

24. 49. 94. 23. 50.7 ]

[71. 40. 32. 86. 0. 32.

61. 21. 9. 77. 42.9 ]

[21. 86. 64. 18. 14. 52.

68. 44. 96. 25. 48.8 ]

[28. 8. 51. 50. 59. 38.

51. 59. 75. 56. 47.5 ]

[47.85714286 60.14285714 54.42857143 48.28571429 29. 60.42857143

37.57142857 52.14285714 76. 44.57142857 0. ]]

6. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти сумму элементов всей матрицы.

Определить, какую долю в этой сумме составляет сумма элементов каждого

столбца. Результат оформить в виде матрицы из N + 1 строк и M столбцов.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**A = np.random.randint(0, 100, (N, M))**

**print(A)**

**Sum = A.sum()**

**B=np.sum(A)**

**M\_sum = np.sum(A, axis=0)/np.sum(A)**

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

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

Ответ

[[48 35 5 5 21 21 66 4 57 89]

[57 34 84 82 35 38 98 69 56 82]

[76 17 90 42 7 13 56 18 23 87]

[59 80 94 23 36 94 45 0 93 85]

[23 10 39 30 43 78 16 5 27 4]

[13 80 87 27 23 57 80 45 26 79]

[26 48 61 7 63 70 53 9 82 46]]

Новая матрица:

[[4.80000000e+01 3.50000000e+01 5.00000000e+00 5.00000000e+00

2.10000000e+01 2.10000000e+01 6.60000000e+01 4.00000000e+00

5.70000000e+01 8.90000000e+01]

[5.70000000e+01 3.40000000e+01 8.40000000e+01 8.20000000e+01

3.50000000e+01 3.80000000e+01 9.80000000e+01 6.90000000e+01

5.60000000e+01 8.20000000e+01]

[7.60000000e+01 1.70000000e+01 9.00000000e+01 4.20000000e+01

7.00000000e+00 1.30000000e+01 5.60000000e+01 1.80000000e+01

2.30000000e+01 8.70000000e+01]

[5.90000000e+01 8.00000000e+01 9.40000000e+01 2.30000000e+01

3.60000000e+01 9.40000000e+01 4.50000000e+01 0.00000000e+00

9.30000000e+01 8.50000000e+01]

[2.30000000e+01 1.00000000e+01 3.90000000e+01 3.00000000e+01

4.30000000e+01 7.80000000e+01 1.60000000e+01 5.00000000e+00

2.70000000e+01 4.00000000e+00]

[1.30000000e+01 8.00000000e+01 8.70000000e+01 2.70000000e+01

2.30000000e+01 5.70000000e+01 8.00000000e+01 4.50000000e+01

2.60000000e+01 7.90000000e+01]

[2.60000000e+01 4.80000000e+01 6.10000000e+01 7.00000000e+00

6.30000000e+01 7.00000000e+01 5.30000000e+01 9.00000000e+00

8.20000000e+01 4.60000000e+01]

[9.20451082e-02 9.26546785e-02 1.40201158e-01 6.58335873e-02

6.94910088e-02 1.13075282e-01 1.26181042e-01 4.57177690e-02

1.10941786e-01 1.43858580e-01]]

7. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти сумму элементов всей матрицы.

Определить, какую долю в этой сумме составляет сумма элементов каждой

строки. Результат оформить в виде матрицы из N строк и M+1 столбцов.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**A = np.random.randint(0, 100, (N, M))**

**print(A)**

**Sum = A.sum()**

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

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

**X = []**

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

**n = Sum\_column[i] / Sum**

**X.append(n)**

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

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

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

Ответ

[[86 49 53 95 85 47 68 79 26 80]

[20 11 39 82 21 88 1 42 92 5]

[50 4 55 30 9 78 18 58 24 82]

[63 56 84 2 68 36 44 67 24 23]

[20 60 88 16 86 18 64 82 8 5]

[36 0 36 56 69 90 29 63 71 48]

[97 36 90 90 24 14 26 41 94 47]]

Сумма элементов всей матрицы: 3448

Новая матрица:

[[86. 49. 53. 95. 85. 47.

68. 79. 26. 80. 0.1937355 ]

[20. 11. 39. 82. 21. 88.

1. 42. 92. 5. 0.1162993 ]

[50. 4. 55. 30. 9. 78.

18. 58. 24. 82. 0.11832947]

[63. 56. 84. 2. 68. 36.

44. 67. 24. 23. 0.13544084]

[20. 60. 88. 16. 86. 18.

64. 82. 8. 5. 0.12964037]

[36. 0. 36. 56. 69. 90.

29. 63. 71. 48. 0.14443155]

[97. 36. 90. 90. 24. 14.

26. 41. 94. 47. 0.16212297]]

8. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Определить, сколько отрицательных элементов

содержится в каждом столбце и в каждой строке матрицы. Результат

оформить в виде матрицы из N + 1 строк и M + 1 столбцов.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**A = np.random.randint(-100, 100, (N, M))**

**print(A)**

**Sum = A.sum()**

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

**X = []**

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

**K = 0**

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

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

**K += 1**

**X.append(K)**

**Y = []**

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

**K = 0**

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

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

**K += 1**

**Y.append(K)**

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

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

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

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

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

Ответ

[[ -70 48 -64 22 13 32 85 -37 -59 -28]

[ -33 40 -16 54 45 -67 -14 12 11 96]

[ 19 -86 -18 41 29 68 -25 -89 -40 -15]

[ 42 -82 -43 58 4 -6 76 -61 7 -81]

[ -19 12 -10 -97 -90 -5 -47 57 -85 -100]

[ -25 -27 -11 31 -52 -82 -26 18 98 -17]

[ -98 -29 6 12 -21 72 -47 26 -8 50]]

Сумма элементов всей матрицы: -646

Новая матрица:

[[ -70. 48. -64. 22. 13. 32. 85. -37. -59. -28. 5.]

[ -33. 40. -16. 54. 45. -67. -14. 12. 11. 96. 4.]

[ 19. -86. -18. 41. 29. 68. -25. -89. -40. -15. 6.]

[ 42. -82. -43. 58. 4. -6. 76. -61. 7. -81. 5.]

[ -19. 12. -10. -97. -90. -5. -47. 57. -85. -100. 8.]

[ -25. -27. -11. 31. -52. -82. -26. 18. 98. -17. 7.]

[ -98. -29. 6. 12. -21. 72. -47. 26. -8. 50. 5.]

[ 5. 4. 6. 1. 3. 4. 5. 3. 4. 5. 0.]]

9. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Определить, сколько нулевых элементов

содержится в верхних L строках матрицы и в левых К столбцах матрицы.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**L = 2**

**K = 3**

**A = np.random.randint(-1, 2, (N, M))**

**print(A)**

**Ln = 0**

**Kn = 0**

**for i in A[:L].flat:**

**if i == 0:**

**Ln += 1**

**for i in A[:, : K].flat:**

**if i == 0:**

**Kn += 1**

**print("Количество нулевых элементов в верхних " + str(L) + " строках матрицы - " + str(Ln))**

**print("Количество нулевых элементов в левых " + str(K) + " столбцах матрицы - " + str(Kn))**

Ответ

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

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

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

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

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

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

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

Количество нулевых элементов в верхних 2 строках матрицы - 8

Количество нулевых элементов в левых 3 столбцах матрицы - 8

10. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Перемножить элементы каждого столбца матрицы

с соответствующими элементами K-го столбца.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**K = 3**

**A = np.random.randint(-1, 2, (N, M))**

**print(A)**

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

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

**print("K-ый столбец: \r\n{}\n".format(K\_arr))**

**A = A \* K\_arr**

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

Ответ

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

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

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

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

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

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

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

K-ый столбец:

[[ 0]

[ 1]

[-1]

[ 0]

[ 0]

[ 1]

[-1]]

Новая матрица:

[[ 0 0 0 0 0 0 0 0 0 0]

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

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

[ 0 0 0 0 0 0 0 0 0 0]

[ 0 0 0 0 0 0 0 0 0 0]

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

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

11. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Просуммировать элементы каждой строки

матрицы с соответствующими элементами L-й строки.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**L = 3**

**A = np.random.randint(-1, 2, (N, M))**

**print(A)**

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

**print("L страка: \r\n{}\n".format(L\_arr))**

**A = A + L\_arr**

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

Ответ

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

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

[ 1 0 0 0 1 1 1 1 1 0]

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

[ 1 0 0 0 1 1 1 1 0 0]

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

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

L страка:

[1 0 0 0 1 1 1 1 1 0]

Новая матрица:

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

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

[ 2 0 0 0 2 2 2 2 2 0]

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

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

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

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

12. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Разделить элементы каждой строки на элемент

этой строки с наибольшим значением.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**A = np.random.randint(-1, 2, (N, M))**

**print(A)**

**Max = A.max(axis=1)**

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

**A = A / Max**

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

Ответ

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

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

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

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

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

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

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

Новая матрица:

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

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

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

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

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

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

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

13. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Разделить элементы каждого столбца матрицы на

элемент этого столбца с наибольшим значением.

Решение

**import numpy as np**

**N = 7**

**M = 10**

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

**print(A)**

**Max = A.max(axis=0)**

**A = A / Max**

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

Ответ

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

[9 9 5 0 3 0 4 9 9 3]

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

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

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

[8 3 2 8 7 4 7 8 2 8]

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

Новая матрица:

[[0.55555556 0.77777778 0. 0.625 0.77777778 0.33333333

0.22222222 0.66666667 1. 1. ]

[1. 1. 0.55555556 0. 0.33333333 0.

0.44444444 1. 1. 0.375 ]

[0.77777778 0.66666667 1. 0.25 1. 0.77777778

0.66666667 0.22222222 0.33333333 0. ]

[1. 0.44444444 0.44444444 0.5 0.22222222 1.

0.33333333 0. 0.55555556 0.875 ]

[0.88888889 0.44444444 0. 0. 0.44444444 0.88888889

1. 0.66666667 0. 0.625 ]

[0.88888889 0.33333333 0.22222222 1. 0.77777778 0.44444444

0.77777778 0.88888889 0.22222222 1. ]

[0.55555556 0.44444444 0. 0.625 0.33333333 0.44444444

0.44444444 0.55555556 0.55555556 0.5 ]]

14. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Разделить элементы матрицы на элемент матрицы

с наибольшим значением.

Решение

**import numpy as np**

**N = 7**

**M = 10**

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

**print(A)**

**Max = A.max()**

**print(Max)**

**A = A / Max**

print("Новая матрица:\n" + str(A))

Ответ

[[6 4 3 2 7 0 0 1 1 5]

[3 5 9 6 4 8 4 0 6 0]

[6 1 0 6 8 9 5 1 6 0]

[0 0 5 6 6 0 3 0 7 8]

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

[7 5 4 3 1 7 4 6 4 8]

[3 9 7 3 7 3 4 4 7 8]]

9

Новая матрица:

[[0.66666667 0.44444444 0.33333333 0.22222222 0.77777778 0.

0. 0.11111111 0.11111111 0.55555556]

[0.33333333 0.55555556 1. 0.66666667 0.44444444 0.88888889

0.44444444 0. 0.66666667 0. ]

[0.66666667 0.11111111 0. 0.66666667 0.88888889 1.

0.55555556 0.11111111 0.66666667 0. ]

[0. 0. 0.55555556 0.66666667 0.66666667 0.

0.33333333 0. 0.77777778 0.88888889]

[0.44444444 0.11111111 0. 0.55555556 0.11111111 0.77777778

1. 0.77777778 1. 0.33333333]

[0.77777778 0.55555556 0.44444444 0.33333333 0.11111111 0.77777778

0.44444444 0.66666667 0.44444444 0.88888889]

[0.33333333 1. 0.77777778 0.33333333 0.77777778 0.33333333

0.44444444 0.44444444 0.77777778 0.88888889]]

15. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Все элементы имеют целый тип. Дано целое число

H. Определить, какие столбцы имеют хотя бы одно такое число, а какие не

имеют.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**H = 4**

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

**print(A)**

**a = []**

**b = []**

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

**if H in A[:, i]:**

**a.append(i+1)**

**else:**

**b.append(i+1)**

**print("Столбцы, которые имеют хотя бы одно число H - {}\n".format(a))**

**print("Столбцы, которые не имеют это число - {}\n".format(b))**

Ответ

[[9 4 9 0 4 3 9 1 2 6]

[7 6 7 3 1 2 4 6 9 5]

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

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

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

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

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

Столбцы, которые имеют хотя бы одно число H - [2, 4, 5, 6, 7, 10]

Столбцы, которые не имеют это число - [1, 3, 8, 9]

16. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Исключить из матрицы строку с номером L.

Сомкнуть строки матрицы.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**L = 3**

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

**print(A)**

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

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

Ответ

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

[9 9 7 4 6 2 2 9 1 3]

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

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

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

[3 9 0 8 3 8 4 9 5 0]

[6 9 0 0 4 3 2 7 1 3]]

Новая матрица:

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

[9 9 7 4 6 2 2 9 1 3]

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

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

[3 9 0 8 3 8 4 9 5 0]

[6 9 0 0 4 3 2 7 1 3]]

17. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Добавить к матрице строку и вставить ее под

номером L.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**L = 3**

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

**print(A)**

**row = np.random.randint(low=-9, high=10, size=M)**

**print("Строка для вставки: " + str(row))**

**A = np.insert(A, L, row, axis=0)**

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

Ответ

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

[3 1 8 3 2 9 2 3 9 1]

[7 2 2 7 3 5 1 0 7 6]

[3 7 5 5 5 6 4 6 3 5]

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

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

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

Строка для вставки: [-6 -5 -9 6 0 4 9 -1 0 3]

Новая матрица:

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

[ 3 1 8 3 2 9 2 3 9 1]

[ 7 2 2 7 3 5 1 0 7 6]

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

[ 3 7 5 5 5 6 4 6 3 5]

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

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

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

18. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Найти сум

расположенных параллельно побочной диагонали (ближайшие к побочной).

Элементы побочной диагонали имеют индексы от [N,0] до [0,N].

Решение

**import numpy as np**

**N = 7**

**M = 10**

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

**print(A)**

**a = np.diagonal(A)**

**a\_sum = a.sum()**

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

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

**b\_sum = b.sum()**

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

Ответ

[[2 0 8 5 9 7 1 9 4 9]

[4 0 4 3 4 2 5 9 6 4]

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

[0 2 3 4 9 3 2 8 1 7]

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

[2 4 7 3 4 4 4 4 1 1]

[8 9 4 3 9 7 5 4 0 1]]

Главная диагональ:

[2 0 7 4 7 4 5]

Её сумма = 29

Побочная диагональ:

[9 6 5 2 9 4 3]

Её сумма = 38

19.Создать квадратную матрицу A, имеющую N строк и N столбцов со случайными элементами. Определить сумму элементов, расположенных параллельно главной диагонали (ближайшие к главной). Элементы главной диагонали имеют индексы от [0,0] до [N,N].

Решение

**import numpy as np**

**N = 7**

**M = 10**

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

**print(A)**

**a = np.diagonal(A, 1)**

**a\_sum = a.sum()**

**print("Элементы которые выше главной диагонали: \n" + str(a) + "\nИх сумма = " + str(a\_sum))**

**b = np.diagonal(A, -1)**

**b\_sum = b.sum()**

**print("Элементы которые ниже главной диагонали: \n" + str(b) + "\nИх сумма = " + str(a\_sum))**

Ответ

[[5 2 5 2 3 1 7 8 4 8]

[0 3 1 7 8 5 4 9 7 7]

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

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

[1 1 5 0 7 0 1 8 4 9]

[6 1 2 7 1 7 4 6 6 3]

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

Элементы которые выше главной диагонали:

[2 1 2 3 0 4 7]

Их сумма = 19

Элементы которые ниже главной диагонали:

[0 6 4 0 1 8]

Их сумма = 19

20.Создать квадратную матрицу A, имеющую N строк и N столбцов со случайными элементами. Определить произведение элементов, расположенных параллельно побочной диагонали (ближайшие к побочной). Элементы побочной диагонали имеют индексы от [N,0] до [0,N].

Решение

**import numpy as np**

**N = 7**

**M = 10**

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

**print(A)**

**a = b = np.fliplr(A).diagonal(1)**

**a\_prod = a.prod()**

**print("Элементы которые выше побочной диагонали: \n" + str(a) + "\nИх сумма = " + str(a\_prod))**

**b = np.fliplr(A).diagonal(-1)**

**b\_prod = b.prod()**

**print("Элементы которые ниже побочной диагонали: \n" + str(b) + "\nИх сумма = " + str(b\_prod))**

Ответ

[[5 5 4 2 1 7 6 8 2 4]

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

[5 2 8 1 4 5 4 3 4 7]

[3 5 7 4 9 4 3 7 3 9]

[3 7 5 3 8 7 0 8 3 9]

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

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

Элементы которые выше побочной диагонали:

[2 5 4 4 8 8 4]

Их сумма = 40960

Элементы которые ниже побочной диагонали:

[2 4 7 0 6 0]

Их сумма = 0

21. Создать квадратную матрицу A, имеющую N строк и N столбцов со случайными элементами. Каждой паре элементов, симметричных относительно главной диагонали (ближайшие к главной), присвоить значения, равные полусумме этих симметричных значений (элементы главной диагонали имеют индексы от [0,0] до [N,N]).

Решение

import numpy as np

**N = 7**

**M = 7**

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

**print(A)**

**B = (A + A.T)/2**

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

Ответ

[[0 0 5 6 5 8 6]

[6 2 3 9 6 9 2]

[4 9 8 9 0 1 9]

[3 9 9 1 9 9 6]

[9 3 2 9 8 1 8]

[4 9 8 6 6 8 5]

[6 1 9 5 0 6 2]]

Новая матрица:

[[0 0 5 6 5 8 6]

[6 2 3 9 6 9 2]

[4 9 8 9 0 1 9]

[3 9 9 1 9 9 6]

[9 3 2 9 8 1 8]

[4 9 8 6 6 8 5]

[6 1 9 5 0 6 2]]

22. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Исходная матрица состоит из нулей и единиц.

Добавить к матрице еще один столбец, каждый элемент которого делает

количество единиц в каждой строке чётным.

Решение

**import numpy as np**

**N = 7**

**M = 10**

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

**print(A)**

**col = [i % 2 for i in np.sum(A, axis=1)]**

**A = np.insert(A, M, col, axis=1)**

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

Ответ

[[0 1 1 1 1 1 0 1 1 1]

[0 1 1 1 0 1 0 1 0 1]

[0 1 0 1 0 0 1 1 1 0]

[1 1 0 1 1 0 0 1 0 0]

[0 1 1 0 1 0 1 0 1 1]

[1 0 0 0 1 0 1 1 1 0]

[0 0 1 0 0 0 1 1 1 1]]

Новая матрица:

[[0 1 1 1 1 1 0 1 1 1 0]

[0 1 1 1 0 1 0 1 0 1 0]

[0 1 0 1 0 0 1 1 1 0 1]

[1 1 0 1 1 0 0 1 0 0 1]

[0 1 1 0 1 0 1 0 1 1 0]

[1 0 0 0 1 0 1 1 1 0 1]

[0 0 1 0 0 0 1 1 1 1 1]]

23. Создать квадратную матрицу A, имеющую N строк и N столбцов со

случайными элементами. Найти сумму элементов, расположенных выше

главной диагонали, и произведение элементов, расположенных выше

побочной диагонали (элементы главной диагонали имеют индексы от [0,0]

до [N,N], а элементы побочной диагонали — от [N,0] до [0,N]).

Решение

**import numpy as np**

**N = 7**

**M = 10**

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

**print(A)**

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

**a = A[iu]**

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

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

**b = np.fliplr(A)[iu]**

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

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

Ответ

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

[4 4 9 9 3 9 3 2 4 5]

[2 4 6 6 2 6 4 7 6 3]

[5 6 9 4 7 1 1 6 9 8]

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

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

[0 0 5 9 2 0 9 8 3 4]]

Cумма элементов выше главной диагонали = 90

Произведение элементов выше побочной диагонали = 203685888

24. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Дан номер строки L и номер столбца K, при

помощи которых исходная матрица разбивается на четыре части. Найти

сумму элементов каждой части.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**L = 4**

**K = 2**

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

**print(A)**

**A = np.random.randint(low=-9, high=10, size=(N, M))**

**print("Матрица:\r\n{}\n".format(A))**

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

**x\_sum = x.sum()**

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

**y = A[L: ,0 :K]**

**y\_sum = y.sum()**

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

**z = A[0:L ,K:]**

**z\_sum = z.sum()**

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

**a = A[L: ,K:]**

**a\_sum = a.sum()**

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

Ответ

[[9 9 8 6 0 3 8 0 4 2]

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

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

[8 4 5 8 1 3 2 6 9 8]

[7 8 0 1 7 0 2 3 4 1]

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

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

Матрица:

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

[-8 -7 -7 -3 -2 -6 6 -9 -1 -9]

[-2 6 9 1 -1 1 4 -7 1 -9]

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

[ 7 5 3 -8 -2 -5 9 1 -6 7]

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

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

Вверхняя левая часть: сумма равна = 0

[[-1 -1]

[-8 -7]

[-2 6]

[ 6 7]]

Нижняя левая часть: сумма равна = 7

[[ 7 5]

[-9 -6]

[ 9 1]]

Вверхняя правая часть: сумма равна = 17

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

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

[ 9 1 -1 1 4 -7 1 -9]

[ 9 8 -6 6 9 5 -4 3]]

Нижняя правая часть: сумма равна = -3

[[ 3 -8 -2 -5 9 1 -6 7]

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

[ 2 0 -2 1 6 6 -2 -5]]

25. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Определить, сколько нулевых элементов

содержится в каждом столбце и в каждой строке матрицы. Результат

оформить в виде матрицы из N + 1 строк и M + 1 столбцов.

Решение

**import numpy as np**

**N = 7**

**M = 7**

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

**print(A)**

**bool = A == 0**

**col = np.sum(bool, axis=1)**

**A = np.insert(A, M, col, axis=1)**

**row = np.append(np.sum(bool, axis=0), 0)**

**A = np.insert(A, N, row, axis=0)**

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

Ответ

[[9 6 4 3 1 4 1]

[8 1 9 6 8 8 2]

[3 2 9 8 2 9 9]

[4 6 8 4 1 3 9]

[4 6 0 2 9 2 9]

[4 7 1 6 6 1 8]

[0 3 6 2 9 2 0]]

Новая матрица:

[[9 6 4 3 1 4 1 0]

[8 1 9 6 8 8 2 0]

[3 2 9 8 2 9 9 0]

[4 6 8 4 1 3 9 0]

[4 6 0 2 9 2 9 1]

[4 7 1 6 6 1 8 0]

[0 3 6 2 9 2 0 2]

[1 0 1 0 0 0 1 0]]

26. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Дан номер строки L и номер столбца K, при

помощи которых исходная матрица разбивается на четыре части. Найти

среднее арифметическое элементов каждой части.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**L = 4**

**K = 2**

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

**print(A)**

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

**x\_sum = x.mean()**

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

**y = A[L: ,0 :K]**

**y\_sum = y.mean()**

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

**z = A[0:L ,K:]**

**z\_sum = z.mean()**

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

**a = A[L: ,K:]**

**a\_sum = a.mean()**

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

Ответ

[[2 3 6 9 0 4 6 5 5 4]

[3 4 8 4 1 7 7 9 6 5]

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

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

[0 8 6 3 2 8 4 3 4 0]

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

[1 5 4 1 2 8 4 1 7 8]]

Вверхняя левая часть: среднее арифметическое = 4.25

[[2 3]

[3 4]

[4 7]

[3 8]]

Нижняя левая часть: среднее арифметическое = 4.5

[[0 8]

[4 9]

[1 5]]

Вверхняя правая часть: среднее арифметическое = 4.65625

[[6 9 0 4 6 5 5 4]

[8 4 1 7 7 9 6 5]

[4 1 2 3 9 4 5 1]

[5 8 2 1 3 6 3 6]]

Нижняя правая часть: среднее арифметическое = 4.75

[[6 3 2 8 4 3 4 0]

[9 9 9 7 2 3 2 8]

[4 1 2 8 4 1 7 8]]

27. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Все элементы имеют целый тип. Дано целое число

H. Определить, какие строки имеют хотя бы одно такое число, а какие не

имеют.

Решение

**import numpy as np**

**N = 7**

**M = 10**

**H = 4**

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

**print(A)**

**bool = A == H**

**col\_sum = np.sum(bool, axis=1)**

**print("Строки в которых встречается значение: " + str(H))**

**print(np.argwhere(col\_sum).flatten())**

**print("Строки в которых нет значения: " + str(H))**

**print(np.argwhere(col\_sum == 0).flatten())**

Ответ

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

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

[6 4 7 9 2 3 8 1 9 7]

[0 0 0 9 4 0 7 9 0 7]

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

[5 5 5 1 0 4 2 0 7 2]

[1 9 5 6 9 4 8 9 7 4]]

Строки в которых встречается значение: 4

[0 2 3 4 5 6]

Строки в которых нет значения: 4

[1]

28. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Исключить из матрицы столбец с номером K.

Сомкнуть столбцы матрицы.

Решение

**import numpy as np**

**N = 7**

**M = 7**

**K = 2**

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

**print(A)**

**A = np.random.randint(low=-9, high=10, size=(N, M))**

**print("Матрица:\r\n{}\n".format(A))**

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

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

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

Ответ

[[3 8 9 6 8 9 3]

[8 9 5 1 2 1 5]

[4 7 2 6 3 8 8]

[1 6 8 6 0 9 5]

[3 2 6 9 7 9 9]

[6 1 2 6 7 6 5]

[3 7 9 6 9 2 5]]

Матрица:

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

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

[-9 3 2 -8 7 9 9]

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

[-1 -4 4 0 -8 -8 -5]

[-3 7 5 -4 7 -1 6]

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

K = 2

Новая матрица:

[[ 5 -4 4 8 6 -7]

[-8 3 4 -4 -9 7]

[-9 2 -8 7 9 9]

[ 6 4 -9 -2 -1 1]

[-1 4 0 -8 -8 -5]

[-3 5 -4 7 -1 6]

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

29. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Добавить к матрице столбец чисел и вставить его

под номером K.

Решение

**import numpy as np**

**N = 7**

**M = 7**

**H = 2**

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

**print(A)**

**bool = A == H**

**col\_sum = np.sum(bool, axis=1)**

**print("Строки в которых встречается значение {}:".format(H))**

**print(np.argwhere(col\_sum).flatten())**

**print("Строки в которых нет значения {}:".format(H))**

**print(np.argwhere(col\_sum == 0).flatten())**

Ответ

[[2 0 3 6 6 0 7]

[2 2 3 7 3 5 1]

[6 8 9 0 5 9 8]

[9 2 3 9 6 2 2]

[0 8 4 5 1 8 8]

[3 4 4 0 7 2 7]

[7 5 4 4 2 9 7]]

Строки в которых встречается значение 2:

[0 1 3 5 6]

Строки в которых нет значения 2:

[2 4]

30. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Добавить к элементам каждого столбца такой

новый элемент, чтобы сумма положительных элементов стала бы равна

модулю суммы отрицательных элементов. Результат оформить в виде

матрицы из N + 1 строк и M столбцов.

Решение

**import numpy as np**

**N = 7**

**M = 7**

**K = 2**

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

**print(A)**

**M\_n = np.sum(A, axis=0) \* (-1)**

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

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

Ответ

[[1 4 5 7 0 1 9]

[6 1 2 2 5 3 4]

[8 5 6 0 8 7 2]

[2 4 8 6 2 0 7]

[9 4 3 1 5 3 7]

[9 7 0 9 1 2 2]

[0 9 8 4 5 1 6]]

Новая матрица:

[[ 1 4 5 7 0 1 9]

[ 6 1 2 2 5 3 4]

[ 8 5 6 0 8 7 2]

[ 2 4 8 6 2 0 7]

[ 9 4 3 1 5 3 7]

[ 9 7 0 9 1 2 2]

[ 0 9 8 4 5 1 6]

[-35 -34 -32 -29 -26 -17 -37]]

31. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со

случайными элементами. Добавить к элементам каждой строки такой новый

элемент, чтобы сумма положительных элементов стала бы равна модулю

суммы отрицательных элементов. Результат оформить в виде матрицы из N

строк и M + 1 столбцов.

Решение

import numpy as np

N = 7

M = 7

K = 2

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

print(A)

M\_n = np.sum(A, axis=1) \* (-1)

A = np.hstack((A, M\_n.reshape(-1, 1)))

print("Новая матрица:\n" + str(A))

Ответ

[[5 7 5 6 8 9 6]

[9 1 4 9 4 4 4]

[6 8 6 2 9 1 1]

[9 0 9 0 5 3 3]

[5 2 0 8 3 1 2]

[9 9 1 8 2 6 9]

[9 7 1 1 0 3 3]]

Новая матрица:

[[ 5 7 5 6 8 9 6 -46]

[ 9 1 4 9 4 4 4 -35]

[ 6 8 6 2 9 1 1 -33]

[ 9 0 9 0 5 3 3 -29]

[ 5 2 0 8 3 1 2 -21]

[ 9 9 1 8 2 6 9 -44]

[ 9 7 1 1 0 3 3 -24]]

Системы линейных уравнений

Одним из наиболее универсальных методов решений линейных алгебраических систем является метод Гаусса. Применение этого метода основано на переходе от данной системы к равносильной ей, но более простого вида.

Процесс решения системы методом Гаусса состоит из двух этапов:

* Первый – исключение неизвестных. Вместо исключения неизвестных непосредственно из уравнений системы, выполняются элементарные преобразования над строками расширенной матрицы системы с целью получения ступенчатой матрицы, эквивалентной исходной.
* Второй этап решения системы – нахождение неизвестных из преобразованной системы – называется обратным ходом метода Гаусса. Сначала находится значение одного неизвестного из последнего уравнения. Если в нем было несколько неизвестных, то это означает, что одно из них (назовем его базисным) выражается через остальные (назовем их свободными).

Что бы реализовать это на Python, мы напишем код который будет считывать csv-файлы, в которых будут находиться СЛАУ, и при помощи пакета Numpy решать их.

Системы:

Вариант №7

Задача №1

| 1,69 1,02 1,59 1,11 | \* | X1| = | 0,28|

| 1,08 1,55 1,23 1,65 | \* | X2| = | 0,63|

| 1,50 1,34 1,50 1,13 | \* | X3| = | 0,09|

| 1,86 1,34 1,46 1,66 | \* | X4| = | 0,84|

Задача №2

| 1,11 1,40 1,58 1,04 | \* | X1| = | 0,82|

| 1,61 1,49 1,98 1,48 | \* | X2| = | 0,39|

| 1,45 1,21 1,12 1,04 | \* | X3| = | 0,45|

| 1,38 1,31 1,31 1,86 | \* | X4| = | 0,33|

Задача №3

| 1,52 1,93 1,40 1,54 | \* | X1| = | 0,99|

| 1,61 1,31 1,25 1,83 | \* | X2| = | 0,96|

| 1,81 1,30 1,26 1,61 | \* | X3| = | 0,99|

| 1,73 1,03 1,44 1,12 | \* | X4| = | 0,29|

Задача №4

| 1,32 1,33 1,56 1,86 1,90 1,84 | \* | X1| = | 0,15|

| 1,86 1,57 1,84 1,10 1,91 1,61 | \* | X2| = | 0,67|

| 1,93 1,55 1,49 1,29 1,08 1,89 | \* | X3| = | 0,37|

| 1,08 1,66 1,80 1,20 1,13 1,17 | \* | X4| = | 0,08|

| 1,78 1,79 1,86 1,29 1,72 1,16 | \* | X5| = | 0,52|

| 1,14 1,23 1,05 1,80 1,29 1,39 | \* | X6| = | 0,42|

Задача №5

| 1,15 1,46 1,73 1,70 | \* | X1| = | 0,84|

| 1,89 1,78 1,54 1,49 | \* | X2| = | 0,79|

| 1,09 1,39 1,64 1,02 | \* | X3| = | 0,15|

| 1,62 1,83 1,04 1,18 | \* | X4| = | 0,20|

Код:

**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('./Gauss.csv', delimiter=';')**

**m\_list = []**

**m = []**

**for row in data:**

**first\_col = row[0]**

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

**m\_list.append(m)**

**m = []**

**continue**

**mask = ~numpy.isnan(row)**

**m.append(row[mask])**

**m\_list.append(m)**

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

**f.truncate()**

**for m in m\_list:**

**M = numpy.array(m)**

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

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

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

**print("Решение:")**

**print(slv)**

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

**f.close()**

Результат:

Решение:

[ 0.21036873 -0.6796135 -0.47465322 1.23637949]

Решение:

[-0.60493912 1.6929138 -0.34202809 -0.3251851 ]

Решение:

[ 0.70046276 0.68869306 -1.44647644 0.40336753]

Решение:

[-0.87064203 3.77893201 -3.38404262 -1.59548229 1.54814076 0.85788261]

Решение:

[ 0.77234462 -1.25258834 -0.03173994 1.07970162]

Вывод

В своём выводе я хочу объяснить почему язык программирования Python один из лучших.

Любить Python куда проще. Во-первых, просто потому, что он одним из первых перечеркнул миф о сложности программирования. Это, конечно, не современный Scratch, но по крайней мере для того, чтобы накодить что-то интересное, не надо несколько дней сидеть за книгами и интернет-руководствами. Python имеет предельно простой синтаксис, на нем легко писать, его легко читать, а больше и желать трудно. И да, если вы, как и я, после первого знакомства решили, что из-за доступности код на Python у каждого разработчика имеет свой внешний вид, а потому это вещь в себе, вот опровержение. По ссылке находится ничто иное, как pep8 — гид по стилю в среде Python.

Развитие Python позволяет сегодня создавать сложный код, не затрачивая на это много временных и «строчных» ресурсов. Вы можете использовать динамическую типизацию для упрощения кода и встроенные функции языка, чтобы избавить себя от написания шаблонных кодов. Вкупе с простым синтаксисом Python выглядит настоящим подарком для тех, кто устал от «строгого» программирования. Да, это немного снизит быстродействие, но это не выбор из разряда «хорошо/плохо», а просто иной взгляд на написание кода.

Список литературы

1. Программирование для "нормальных" с нуля на языке Python Часть1 - М.В. Сысоева, И.В. Сысоев.
2. python.org
3. geekbrains.ru
4. numpy.org