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 – 380305.2022.470. ПЗ КР

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

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

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

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Сверчков Я.А./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

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

Работа защищена с оценкой

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

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

Челябинск 2019

**АННОТАЦИЯ**

Сверчков Я.А.

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-142, 2019

В данной работе будет продемонстрировано мои базовые знания языка программирования Python и библиотеки Numpy. Для этого были рассмотрены на эффективность 3 алгоритма сортировки: select, insert, bubble. Также было решено 31 задача с применением библиотеки Numpy, и решена система линейных уравнений методом Гаусса.

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc10311240)

[**ГЛАВА 1. СОРТИРОВКА ОДНОМЕРНОГО МАССИВА** 4](#_Toc10311241)

[**ГЛАВА 2. БИБЛИОТЕКА NUMPY. ОСНОВНЫЕ ФУНКЦИИ** 10](#_Toc10311242)

[**ГЛАВА 3. РЕШЕНИЕ СЛАУ МЕТОДОМ ГАУССА ПРИ ПОМОЩИ ЯП PYTHON** 33](#_Toc10311243)

[**ЗАКЛЮЧЕНИЕ** 37](#_Toc10311244)

[**БИБЛИОГРАФИЧЕСКИЙ СПИСОК** 38](#_Toc10311245)

# **ВВЕДЕНИЕ**

Что же такое Python? Python (Пайтон, Питон) — скриптовый язык программирования. В отличие от С исполняется ровно в тот момент, когда мы его запускаем. До последнего момента программа — это всего лишь текст.

А чем он лучше других языков? Python логичный. Там нет многих болячек других языков. Python изначально был хорошо спроектирован, поэтому достаточно один раз понять его логику, чтобы писать код. Форматирование кода — часть языка. Популярность Python продолжает расти. Его используют в крупных компаниях и стартапах. Например, Instagram, Intel, Positive Technologies, Eve Online, Houdini.

Курсовая работа состоит из 3 частей:

В первой части моей работы было написано три алгоритма сортировки (Сортировка включением, Обменная сортировка, Сортировка выбором). Также был написан код, который сравнивает их. Это сделано чтобы выявить самый эффективный из них.

Во второй части работы было было решено 31 задача с применением библиотеки Numpy. NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

В третей часты было решена СЛАУ методом Гаусса.

# **ГЛАВА 1. СОРТИРОВКА ОДНОМЕРНОГО МАССИВА**

**Алгоритмы сортировки**

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма. Разберем три самых популярных и в это время самых простых алгоритма.

**Сортировка выбором (select)**

Сортировка выбором (Selection sort) — алгоритм сортировки. Может быть как устойчивый, так и неустойчивый. На массиве из n элементов имеет время выполнения в худшем, среднем и лучшем случае Θ(n2), предполагая что сравнения делаются за постоянное время.

Шаги алгоритма:

* находим номер минимального значения в текущем списке
* производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
* теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

Для реализации устойчивости алгоритма необходимо в пункте 2 минимальный элемент непосредственно вставлять в первую неотсортированную позицию, не меняя порядок остальных элементов.

**Сортировка включениями (insert)**

Сортировка вставками (англ. Insertion sort) — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

На вход алгоритма подаётся последовательность n чисел. Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с n элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.

**Обменная сортировка (bubble)**

Сортировка простыми обменами, сортиро́вка пузырько́м (англ. bubble sort) — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов.

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

**Алгоритмы сортироки:**

|  |
| --- |
| def select(arr, dim):  k = 0  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    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   def bubble(arr, dim):  n = 1  alg\_count = [0, 0]   while n < dim:  for i in range(dim - n):  alg\_count[0] += 1  if arr[i] > arr[i + 1]:  arr[i], arr[i + 1] = arr[i + 1], arr[i]  alg\_count[1] += 1  n += 1  return alg\_count |

**Анализ алгоритмов**

В данной части курсовой работы была написана программа, которая определяет эффективность трех алгоритмов сортировки (Сортировка включением, Обменная сортировка, Сортировка выбором). Каждый алгоритм сортировки оформлен в виде функции, которые подключены к основной программе.Проведено 9 опытов (3 метода х 3 реализации одномерного массива), в каждом из которых определено два числа (С - Количество сравнений М - Количество перемещений), но в третьем случае, случайного массива, в отличие от двух первых, сравнительный анализ алгоритмов возможен только по средним значениям. Для этого проведено серия опытов, желательно, 1500 для каждого алгоритма.

Исходя из проделанной работы можно сделать вывод. Самым эффективным методом сортировки для массива является метод "Select"

|  |
| --- |
| import random import algorithms  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)  myfile = open("sort\_methods.txt", "w") # Все результаты запишем в файл  print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Исходный массив") print(select\_arr)  count = [0, 0] count = algorithms.select(select\_arr, DIM) print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив") print(select\_arr) CTotal[0] = count[0] MTotal[0] = count[1]  count = [0, 0] count = algorithms.insert(insert\_arr, DIM) CTotal[1] = count[0] MTotal[1] = count[1]  count = [0, 0] count = algorithms.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 = algorithms.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 = algorithms.insert(insert\_arr, DIM) CTotal[1] = count[0] MTotal[1] = count[1]  count = [0, 0] count = algorithms.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 = algorithms.select(select\_arr, DIM)  CTotal[0] += count[0]  MTotal[0] += count[1]   count = [0, 0]  count = algorithms.insert(insert\_arr, DIM)  CTotal[1] += count[0]  MTotal[1] += count[1]   count = [0, 0]  count = algorithms.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: Исходный 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]  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]  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 |

# **ГЛАВА 2. БИБЛИОТЕКА NUMPY. ОСНОВНЫЕ ФУНКЦИИ**

Поскольку экосистема Python огромна, существуют бесчисленные библиотеки, упрощающие программирование на этом языке. Благодаря таким библиотекам становится возможным более легкое выполнение определенных задач без необходимости написания излишнего кода.

Для начала стоит объяснить что такое библиотеки для Python. Библиотека — сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО).Одна из таких библиотек - это Numpy.

Numpy — это библиотека (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.

Далее представлены задачи и их решения. Во всех задачах были применены функции библиотеки Numpy.

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

случайными элементами. Найти наибольший элемент столбца матрицы A,

для которого сумма абсолютных значений элементов максимальна.

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}".format(A))  sum = A.sum(axis=0) index = sum.argmax(axis=0) max = A.max(axis=0) max = max[index]  print("Наибольшее значение: {}".format(max)) |

|  |
| --- |
| Матрица:  [[ 2 -6 2 -9 -4]  [-6 2 -9 -6 0]  [-2 -6 8 5 -7]  [ 7 -7 9 -5 -8]]  Наибольшее значение: 9 |

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

случайными элементами. Найти наибольшее значение среди средних

значений для каждой строки матрицы.

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}".format(A))  Average = A.mean(axis=1) index = Average.argmax(axis=0) max = Average.max(axis=0)  print("Наибольшее среднее значение: {}".format(max)) |

|  |
| --- |
| Матрица:  [[-5 0 9 0 -6]  [ 9 8 -3 -1 6]  [-8 -3 -3 -4 -5]  [ 1 8 7 5 -7]]  Наибольшее среднее значение: 3.8 |

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

случайными элементами. Найти наименьший элемент столбца матрицы A,

для которого сумма абсолютных значений элементов максимальна.

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}".format(A))  sum = A.sum(axis=0) index = sum.argmin(axis=0) min = A.min(axis=0) min = min[index]  print("Наименьшее значение: {}".format(min)) |

|  |
| --- |
| Матрица:  [[-7 5 6 3 -1]  [-3 -4 9 9 -6]  [-3 6 -7 3 8]  [ 5 -5 5 3 -9]]  Наименьшее значение: -7 |

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

случайными элементами. Найти наименьшее значение среди средних

значений для каждой строки матрицы.

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}".format(A))  Average = A.mean(axis=1) index = Average.argmin(axis=0) min = Average.min(axis=0)  print("Наименьшее среднее значение: {}".format(min)) |

|  |
| --- |
| Матрица:  [[-7 1 -8 1 -2]  [ 1 6 9 -4 -4]  [ 6 -5 -7 7 1]  [-6 -5 2 2 4]]  Наименьшее среднее значение: -3.0 |

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

случайными элементами. Определить средние значения по всем строкам и

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

+ 1 столбцов.

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[ 6 -9 -1 -1 3]  [ 5 2 -9 -2 -1]  [ 6 -6 3 -5 6]  [ 4 -2 2 6 8]]  Новая матрица:  [[ 6. -9. -1. -1. 3. -0.4 ]  [ 5. 2. -9. -2. -1. -1. ]  [ 6. -6. 3. -5. 6. 0.8 ]  [ 4. -2. 2. 6. 8. 3.6 ]  [ 5.25 -3.75 -1.25 -0.5 4. 0. ]] |

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

случайными элементами. Найти сумму элементов всей матрицы.

Определить, какую долю в этой сумме составляет сумма элементов каждого

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

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[-8 -3 1 6 -7]  [-7 5 -5 2 -3]  [-7 -3 -4 -8 9]  [-8 -6 2 0 1]]  Новая матрица:  [[-8. -3. 1. 6. -7. ]  [-7. 5. -5. 2. -3. ]  [-7. -3. -4. -8. 9. ]  [-8. -6. 2. 0. 1. ]  [ 0.69767442 0.1627907 0.13953488 -0. -0. ]] |

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

случайными элементами. Найти сумму элементов всей матрицы.

Определить, какую долю в этой сумме составляет сумма элементов каждой

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

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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(A) |

|  |
| --- |
| Матрица:  [[ 8 -9 1 -7 1]  [ 4 4 7 -9 0]  [ 3 4 5 -8 -7]  [-8 9 -2 8 1]]  Сумма элементов всей матрицы: 5  [[ 8. -9. 1. -7. 1. -1.2]  [ 4. 4. 7. -9. 0. 1.2]  [ 3. 4. 5. -8. -7. -0.6]  [-8. 9. -2. 8. 1. 1.6]] |

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

случайными элементами. Определить, сколько отрицательных элементов

содержится в каждом столбце и в каждой строке матрицы. Результат

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

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  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:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[-8 -9 2 -1 3]  [ 5 -7 -2 7 1]  [-9 -3 -9 -1 1]  [ 7 -4 -9 0 -4]]  Новая матрица:  [[-8. -9. 2. -1. 3. 3.]  [ 5. -7. -2. 7. 1. 2.]  [-9. -3. -9. -1. 1. 4.]  [ 7. -4. -9. 0. -4. 3.]  [ 2. 4. 3. 2. 1. 0.]] |

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

случайными элементами. Определить, сколько нулевых элементов

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

|  |
| --- |
| import numpy as np  N = 4 M = 5 L = np.random.randint(0, 3) K = np.random.randint(0, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  n = 0 for i in range(0, L):  for j in range(0, M):  if A[i,j] == 0:  n += 1 print("Количество нулевых элементов в верхних L солбцах: {}\n".format(n)) m = 0 for i in range(0, K):  for j in range(0, N):  if A[j,i] == 0:  m += 1 print("Количество нулевых элементов в левых L солбцах: {}\n".format(m)) |

|  |
| --- |
| Матрица:  [[-2 -6 -3 -2 -4]  [ 9 -6 1 9 -4]  [ 6 3 0 6 0]  [ 2 -3 -6 6 -8]]  Количество нулевых элементов в верхних L солбцах: 0  Количество нулевых элементов в левых L солбцах: 0 |

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

случайными элементами. Перемножить элементы каждого столбца матрицы

с соответствующими элементами K-го столбца.

|  |
| --- |
| import numpy as np  N = 4 M = 5 K = np.random.randint(1, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[-8 -4 -8 -4 -7]  [ 7 -7 -7 -3 6]  [-7 -1 7 8 -3]  [ 3 -3 -1 0 -9]]  K-ый столбец:  [[-8]  [ 7]  [-7]  [ 3]]  Новая матрица:  [[ 64 32 64 32 56]  [ 49 -49 -49 -21 42]  [ 49 7 -49 -56 21]  [ 9 -9 -3 0 -27]] |

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

случайными элементами. Просуммировать элементы каждой строки

матрицы с соответствующими элементами L-й строки.

|  |
| --- |
| import numpy as np  N = 4 M = 5 L = np.random.randint(1, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  L\_arr = np.array(A[L-1, :]) print("L страка: \r\n{}\n".format(L\_arr)) A = A + L\_arr  print("Новая матрица:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[-6 9 -6 5 -8]  [ 4 -1 5 -1 -9]  [ 7 6 5 -2 -5]  [-6 -7 -4 6 -2]]  L страка:  [ 4 -1 5 -1 -9]  Новая матрица:  [[ -2 8 -1 4 -17]  [ 8 -2 10 -2 -18]  [ 11 5 10 -3 -14]  [ -2 -8 1 5 -11]] |

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

случайными элементами. Разделить элементы каждой строки на элемент

этой строки с наибольшим значением.

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  Max = A.max(axis=1) Max = np.array(Max)[: , np.newaxis] A = A / Max  print("Новая матрица:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[-3 5 -4 -9 2]  [-4 4 5 6 5]  [ 8 7 0 5 8]  [ 2 5 7 4 -4]]  Новая матрица:  [[-0.6 1. -0.8 -1.8 0.4 ]  [-0.66666667 0.66666667 0.83333333 1. 0.83333333]  [ 1. 0.875 0. 0.625 1. ]  [ 0.28571429 0.71428571 1. 0.57142857 -0.57142857]] |

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

случайными элементами. Разделить элементы каждого столбца матрицы на

элемент этого столбца с наибольшим значением.

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  Max = A.max(axis=0) A = A / Max  print("Новая матрица:\r\n{}\n".format(A) |

|  |
| --- |
| Матрица:  [[ 1 -6 -9 8 -4]  [-1 8 8 1 0]  [-1 -1 5 4 2]  [-1 -7 -6 8 3]]  Новая матрица:  [[ 1. -0.75 -1.125 1. -1.33333333]  [-1. 1. 1. 0.125 0. ]  [-1. -0.125 0.625 0.5 0.66666667]  [-1. -0.875 -0.75 1. 1. ]] |

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

случайными элементами. Разделить элементы матрицы на элемент матрицы

с наибольшим значением.

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  Max = A.max() print(Max) A = A / Max  print("Новая матрица:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[-7 8 1 -3 1]  [ 5 -8 -6 3 -3]  [-1 -3 2 6 1]  [ 8 -3 8 7 -8]]  8  Новая матрица:  [[-0.875 1. 0.125 -0.375 0.125]  [ 0.625 -1. -0.75 0.375 -0.375]  [-0.125 -0.375 0.25 0.75 0.125]  [ 1. -0.375 1. 0.875 -1. ]] |

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

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

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

имеют.

|  |
| --- |
| import numpy as np  N = 4 M = 5 H = np.random.randint(1, 4)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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)) |

|  |
| --- |
| Матрица:  [[ 4 5 3 6 -5]  [ 8 -1 4 7 -2]  [-6 -3 5 -9 6]  [-2 7 3 8 -3]]  Столбцы, которые имеют хотя бы одно число H - []  Столбцы, которые не имеют это число - [1, 2, 3, 4, 5] |

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

случайными элементами. Исключить из матрицы строку с номером L.

Сомкнуть строки матрицы.

|  |
| --- |
| import numpy as np  N = 4 M = 5 L = np.random.randint(1, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  print("L = " + str(L)) A = np.delete(A, (L-1), axis=0)  print("Новая матрица:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[ 8 0 3 5 -8]  [ 9 2 -1 -8 4]  [ 4 5 -9 -7 1]  [ 1 6 -8 -1 2]]  L = 1  Новая матрица:  [[ 9 2 -1 -8 4]  [ 4 5 -9 -7 1]  [ 1 6 -8 -1 2]] |

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

случайными элементами. Добавить к матрице строку и вставить ее под

номером L.

|  |
| --- |
| import numpy as np  N = 4 M = 5 L = np.random.randint(1, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  a = np.random.randint(0, 10, M) print("Допонлительная строка: " + str(a)) print("\n L = " + str(L))  row = np.random.randint(low=-9, high=10, size=M) print("Строка для вставки: {}".format(row)) A = np.insert(A, L, row, axis=0)  print("Новая матрица:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[ 4 -6 5 -5 6]  [ 9 6 4 -1 -9]  [-9 6 -4 -8 7]  [ 3 2 -3 -6 2]]  Допонлительная строка: [0 3 9 9 7]  L = 2  Строка для вставки: [ 2 -1 -6 1 -2]  Новая матрица:  [[ 4 -6 5 -5 6]  [ 9 6 4 -1 -9]  [ 2 -1 -6 1 -2]  [-9 6 -4 -8 7]  [ 3 2 -3 -6 2]] |

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

случайными элементами. Найти сумму элементов, стоящих на главной

диагонали, и сумму элементов, стоящих на побочной диагонали (элементы

главной диагонали имеют индексы от [0,0] до [N,N], а элементы побочной

диагонали — от [N,0] до [0,N]).

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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)) |

|  |
| --- |
| Матрица:  [[ 2 5 8 -5 4]  [ 8 1 -9 6 -4]  [ 0 2 2 -8 8]  [ 9 -1 3 -1 6]]  Главная диагональ:  [ 2 1 2 -1]  Её сумма = 4  Побочная диагональ:  [ 4 6 2 -1]  Её сумма = 11 |

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

случайными элементами. Определить сумму элементов, расположенных

параллельно главной диагонали (ближайшие к главной). Элементы главной

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

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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)) |

|  |
| --- |
| Матрица:  [[ 3 -1 3 3 6]  [ 6 -9 -8 5 -4]  [-3 -5 -1 8 4]  [-6 -5 1 -6 8]]  Элементы которые выше главной диагонали:  [-1 -8 8 8]  Их сумма = 7  Элементы которые ниже главной диагонали:  [ 6 -5 1]  Их сумма = 7 |

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

случайными элементами. Определить произведение элементов,

расположенных параллельно побочной диагонали (ближайшие к побочной).

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

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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)) |

|  |
| --- |
| Матрица:  [[-1 0 2 -9 -6]  [ 2 0 4 3 -9]  [ 7 -7 -7 2 0]  [-9 -4 1 -1 2]]  Элементы которые выше побочной диагонали:  [-9 4 -7 -9]  Их сумма = -2268  Элементы которые ниже побочной диагонали:  [-9 2 1]  Их сумма = -18 |

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

случайными элементами. Каждой паре элементов, симметричных

относительно главной диагонали (ближайшие к главной), присвоить

значения, равные полусумме этих симметричных значений (элементы

главной диагонали имеют индексы от [0,0] до [N,N]).

|  |
| --- |
| import numpy as np  N = 4 M = 4  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  B = (A + A.T)/2 print("Новая матрица:\r\n{}\n".format(B)) |

|  |
| --- |
| Матрица:  [[-1 3 -1 -9]  [ 2 -8 3 -7]  [-6 -5 7 3]  [-7 2 -2 1]]  Новая матрица:  [[-1. 2.5 -3.5 -8. ]  [ 2.5 -8. -1. -2.5]  [-3.5 -1. 7. 0.5]  [-8. -2.5 0.5 1. ]] |

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

случайными элементами. Исходная матрица состоит из нулей и единиц.

Добавить к матрице еще один столбец, каждый элемент которого делает

количество единиц в каждой строке чётным.

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  col = [i % 2 for i in np.sum(A, axis=1)] A = np.insert(A, M, col, axis=1)  print("Новая матрица:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[ 1 -5 -6 -1 -1]  [-3 2 1 -1 -1]  [-5 -1 -2 7 2]  [ 1 8 -4 3 -8]]  Новая матрица:  [[ 1 -5 -6 -1 -1 0]  [-3 2 1 -1 -1 0]  [-5 -1 -2 7 2 1]  [ 1 8 -4 3 -8 0]] |

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]).

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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)) |

|  |
| --- |
| Матрица:  [[-1 7 6 5 -4]  [ 3 7 -4 -3 -1]  [-6 0 4 9 -1]  [-2 -6 0 9 5]]  Cумма элементов выше главной диагонали = 20  Произведение элементов выше побочной диагонали = 0 |

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

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

помощи которых исходная матрица разбивается на четыре части. Найти

сумму элементов каждой части.

|  |
| --- |
| import numpy as np  N = 4 M = 5 L = np.random.randint(1, 3) K = np.random.randint(1, 3)  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)) |

|  |
| --- |
| Матрица:  [[-7 0 4 1 -7]  [-3 -5 7 -9 9]  [-1 -5 8 2 2]  [-8 3 4 6 7]]  Вверхняя левая часть: сумма равна = -10  [[-7]  [-3]]  Нижняя левая часть: сумма равна = -9  [[-1]  [-8]]  Вверхняя правая часть: сумма равна = 0  [[ 0 4 1 -7]  [-5 7 -9 9]]  Нижняя правая часть: сумма равна = 27  [[-5 8 2 2]  [ 3 4 6 7]] |

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

случайными элементами. Определить, сколько нулевых элементов

содержится в каждом столбце и в каждой строке матрицы. Результат

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

|  |
| --- |
| import numpy as np  N = 4 M = 5  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[ 0 -8 2 -3 -1]  [ 8 -8 -5 9 5]  [-7 -9 2 9 -5]  [-9 1 -5 1 3]]  Новая матрица:  [[ 0 -8 2 -3 -1 1]  [ 8 -8 -5 9 5 0]  [-7 -9 2 9 -5 0]  [-9 1 -5 1 3 0]  [ 1 0 0 0 0 0]] |

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

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

помощи которых исходная матрица разбивается на четыре части. Найти

среднее арифметическое элементов каждой части.

|  |
| --- |
| import numpy as np  N = 4 M = 5 L = np.random.randint(1, 3) K = np.random.randint(1, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A)) # Разделение матрицы на 4 части и нахождение среднее арифметическое 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)) |

|  |
| --- |
| Матрица:  [[ 0 -7 7 -2 7]  [-6 3 9 4 5]  [-8 7 -8 -5 3]  [-8 -1 -5 7 8]]  Вверхняя левая часть: среднее арифметическое = 0.0  [[0]]  Нижняя левая часть: среднее арифметическое = -7.333333333333333  [[-6]  [-8]  [-8]]  Вверхняя правая часть: среднее арифметическое = 1.25  [[-7 7 -2 7]]  Нижняя правая часть: среднее арифметическое = 2.25  [[ 3 9 4 5]  [ 7 -8 -5 3]  [-1 -5 7 8]] |

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

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

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

имеют.

|  |
| --- |
| import numpy as np  N = 4 M = 5 H = np.random.randint(1, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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()) |

|  |
| --- |
| Матрица:  [[ 6 8 5 9 0]  [-4 4 0 -1 -5]  [-2 8 -7 2 7]  [-6 -9 4 3 -9]]  Строки в которых встречается значение 2:  [2]  Строки в которых нет значения 2:  [0 1 3] |

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

случайными элементами. Исключить из матрицы столбец с номером K.

Сомкнуть столбцы матрицы.

|  |
| --- |
| import numpy as np  N = 4 M = 5 K = np.random.randint(1, 3)  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:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[ 8 -2 -5 1 -1]  [-7 6 4 -6 2]  [-6 0 3 -8 8]  [ 5 -4 -5 1 0]]  K = 1  Новая матрица:  [[-2 -5 1 -1]  [ 6 4 -6 2]  [ 0 3 -8 8]  [-4 -5 1 0]] |

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

случайными элементами. Добавить к матрице столбец чисел и вставить его

под номером K.

|  |
| --- |
| import numpy as np  N = 4 M = 5 H = np.random.randint(1, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(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()) |

|  |
| --- |
| Матрица:  [[ 8 -6 3 4 8]  [-8 -7 6 1 -6]  [-8 7 2 6 0]  [ 9 2 -2 -5 -6]]  Строки в которых встречается значение 2:  [2 3]  Строки в которых нет значения 2:  [0 1] |

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

случайными элементами. Добавить к элементам каждого столбца такой

новый элемент, чтобы сумма положительных элементов стала бы равна

модулю суммы отрицательных элементов. Результат оформить в виде

матрицы из N + 1 строк и M столбцов.

|  |
| --- |
| import numpy as np  N = 4 M = 5 K = np.random.randint(1, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  M\_n = np.sum(A, axis=0) \* (-1) A = np.vstack((A, M\_n))  print("Новая матрица:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[ 7 -3 -1 7 0]  [-9 6 -2 1 -5]  [-5 2 8 5 -1]  [ 4 -8 -8 3 8]]  Новая матрица:  [[ 7 -3 -1 7 0]  [ -9 6 -2 1 -5]  [ -5 2 8 5 -1]  [ 4 -8 -8 3 8]  [ 3 3 3 -16 -2]] |

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

случайными элементами. Добавить к элементам каждой строки такой новый

элемент, чтобы сумма положительных элементов стала бы равна модулю

суммы отрицательных элементов. Результат оформить в виде матрицы из N

строк и M + 1 столбцов.

|  |
| --- |
| import numpy as np  N = 4 M = 5 K = np.random.randint(1, 3)  A = np.random.randint(low=-9, high=10, size=(N, M)) print("Матрица:\r\n{}\n".format(A))  M\_n = np.sum(A, axis=1) \* (-1) A = np.hstack((A, M\_n.reshape(-1, 1)))  print("Новая матрица:\r\n{}\n".format(A)) |

|  |
| --- |
| Матрица:  [[ 0 6 -5 7 6]  [ 3 0 -3 -3 2]  [ 1 -7 7 7 6]  [-6 -6 6 2 5]]  Новая матрица:  [[ 0 6 -5 7 6 -14]  [ 3 0 -3 -3 2 1]  [ 1 -7 7 7 6 -14]  [ -6 -6 6 2 5 -1]] |

# **ГЛАВА 3. РЕШЕНИЕ СЛАУ МЕТОДОМ ГАУССА ПРИ ПОМОЩИ PYTHON**

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

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

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

Теперь реализуем это при помощи Python и Numpy

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

Вариант №19

Система №1

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

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

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

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

Система №2

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

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

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

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

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

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

Система №3

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

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

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

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

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

Система №4

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

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

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

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

Система №5

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

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

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

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

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

|  |
| --- |
| import numpy as np  file = open('nympy-gauss-slau.csv', 'wb+') file.truncate() for i in range(1, 6):  task\_file = "slau\_" + str(i) + ".csv"   m = np.genfromtxt(task\_file, delimiter=';')  myA = np.genfromtxt(task\_file, delimiter=';', usecols=(range(len(m-1))))  myB = np.genfromtxt(task\_file, delimiter=';', usecols=(len(m)))   print("Система " + str(i))  print("Матрица A: \n" + str(myA))  print("Матрица B: \n" + str(myB))   slau = np.linalg.solve(myA, myB)   print("Решение ", slau)  np.savetxt(file, np.array([slau]), delimiter=',') file.close() |

# Результат:

|  |
| --- |
| Система 1Матрица A:[[1.26 1.99 1.4 1.71][1.62 1.02 1.08 1.61][1.05 1.13 1.91 1.51][1.41 1.69 1.47 1.58]]Матрица B:[0.07 0.61 0.76 0.56]Решение [ 1.20802617 -0.27312888 0.89429872 -1.26351153]Система 2Матрица A:[[1.35 1.79 1.32 1.91 1.82 1.34][1.7 1.28 1.57 1.85 1.54 1.17][1.6 1.82 1.36 1.29 1.1 1.48][1.43 1.16 1.6 1.22 1.97 1.96][1.92 1.92 1.5 1.56 1.74 1.85][1.11 1.66 1.42 1.07 1.48 1.25]]Матрица B:[0.06 0.89 0.17 0.56 0.9 0.16]Решение [ 2.48001117 -0.42192076 -0.6413774 -1.28376216 1.48067405 -1.4395521 ]Система 3Матрица A:[[1.48 1.54 1.41 1.34 1.26][1.09 1.57 1.56 1.36 1.79][1.43 1.7 1.27 1.41 1.64][1.22 1.45 1.12 1.25 1.72][1.76 1.41 1.11 1.99 1.11]]Матрица B:[0.43 0.35 0.58 0.11 0.65]Решение [-1.50423036 2.5230766 -0.61546825 0.91598569 -1.26102192]Система 4Матрица A:[[1.9 1.03 1.93 1.65][1.88 1.5 1.49 1.32][1.31 1.17 1.4 1.2 ][1.09 1.77 1.31 1.58]]Матрица B:[0.44 0.59 0.73 0.91]Решение [-0.74731721 0.85827464 1.35480483 -0.99326878]Система 5Матрица A:[[1.05 1.32 1.53 1.48][1.54 1.93 1.44 1.59][1.39 1.21 1.47 1.25][1.49 1.78 1.6 1.67]]Матрица B:[0.87 0.87 0.59 0.03]Решение [ -9.93827065 15.4516691 19.04650378 -25.83254717] |

# **ЗАКЛЮЧЕНИЕ**

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

Также благодаря простоте и гибкости языка Python, его можно рекомендовать пользователям (математикам, физикам, экономистам и т.д.) не являющимся программистами, но использующими вычислительную технику и программирование в своей работе.

Программы на Python разрабатываются в среднем в полтора-два (а порой и в два-три) раза быстрее нежели на компилируемых языках (С, С++, Pascal). Поэтому, язык может представлять не малый интерес и для профессиональных программистов, разрабатывающих приложения, не критичные к скорости выполнения, а также программы, использующие сложные структуры данных. В частности, Python хорошо зарекомендовал себя при разработке программ работы с графами, генерации деревьев.

# **БИБЛИОГРАФИЧЕСКИЙ СПИСОК**

1. Лутц М. Программирование на Python, том I, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.
2. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.
3. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.
4. [https://ru.wikipedia.org](https://ru.wikipedia.org/)
5. <https://waksoft.susu.ru/>
6. <https://www.python.org/>