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 – 380305.2022.10. ПЗ КР

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

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

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

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Копытов Д.А./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

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

Работа защищена с оценкой

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

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

Челябинск 2019

# АННОТАЦИЯ

Копытов Д.А.

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-142, 2019

Работа состоит из трёх частей:

* В первой части говориться о методах сортировках
* Во второй части говориться о библиотеке Numpy
* В третьей части говориться немного о линейной алгебре

ОГЛАВЛЕНИЕ

[**АННОТАЦИЯ 2**](#_Toc10580766)

[**ВВЕДЕНИЕ 5**](#_Toc10580767)

[**ГЛАВА 1: СОРТИРОВКА МАССИВОВ 6**](#_Toc10580768)

[**ГЛАВА 2: БИБЛИОТЕКА NUMPY 9**](#_Toc10580769)

[**ГЛАВА 3: ЛИНЕЙНАЯ АЛГЕБРА 21**](#_Toc10580770)

[**ЗАКЛЮЧЕНИЕ 24**](#_Toc10580771)

[**БИБЛИОГРАФИЧЕСКИЙ СПИСОК 25**](#_Toc10580772)

# ВВЕДЕНИЕ

На данный момент один из самых востребованных языков программирования является Python. На нём можно делать почти всё: игры, веб-приложения, бизнесс-приложения, анализировать данные и т.д., без ощутимых проблем.

Более того, порог вхождения низкий, а код во многом лаконичный и понятный даже тому, кто никогда на нём не писал. За счёт простоты кода, дальнейшее сопровождение программ, написанных на Python, становится легче и приятнее по сравнению с Java или C++. А с точки зрения бизнеса это влечёт за собой сокращение расходов и увеличение производительности труда сотрудников.

Далее будут раскрыты основные возможности языка, и решено пару задач с помощью пакета Numpy.

# ГЛАВА 1: СОРТИРОВКА МАССИВОВ

Сортировка — 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.

В данной работе будут представлено всего три метода сортировки:

• Простая обменная сортировка

• Сортировка включением

• Сортировка выбором

Начнем немного с теории, я расскажу о каждом методе и приведу для каждого метода функцию, в которой будет находиться определённый алгоритм сортировки.

Bubble Sort - алгоритм, в котором два соседних элемента, составляющих список, сначала проверяются, а затем меняются местами. Если соседние элементы расположены в неправильном порядке, процесс повторяется до тех пор, пока не будет получен полностью отсортированный список. Каждый проход, который проходит через список, будет помещать следующее наибольшее значение элемента на свое место. Таким образом, по сути, каждый предмет всплывает с намерением достичь места, к которому он по праву принадлежит.

Insertion Sort - это алгоритм сортировки, в котором создается отсортированный массив. Элементы массива сравниваются друг с другом последовательно, а затем располагаются одновременно в некотором конкретном порядке. Аналогию можно понять из стиля, который мы устраиваем колоду карт. Этот тип работает по принципу вставки элемента в определенную позицию, отсюда и название Insertion Sort.

Select Sort - это простой алгоритм сортировки. Этот алгоритм сортировки представляет собой алгоритм сравнения на месте, в котором список делится на две части: отсортированная часть на левом конце и несортированная часть на правом конце. Изначально отсортированная часть пуста, а несортированная часть - весь список. Наименьший элемент выбирается из несортированного массива и заменяется на крайний левый элемент, и этот элемент становится частью отсортированного массива. Этот процесс продолжает перемещать несортированную границу массива на один элемент вправо.

Коды:

|  |
| --- |
| 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  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):  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 |

Теперь стоит вопрос. Какой из алгоритмов использовать? Для была написана программа, которая определит количество сранений и количество перестановок для каждого алгоритма.

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 |

Результат показал, что Select sort самый быстрый из трёх представленных алгоритмов.

**ГЛАВА 2: БИБЛИОТЕКА NUMPY**

Пакет NumPy обеспечивает:

• новый набор типов данных — многомерные массивы чисел (объект array) и

• т.н. универсальные функции (ufunc), позволяющие выполнять операции над этими массивами в один присест

Для использования этих возможностей необходимо в начале программы указать: import numpy.

Решим несколько задач для демонстрации функций пакета 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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  sum = A.sum(axis=0)  i = sum.argmax(axis=0)  max = A.max(axis=0)  max = max[i]  print("Наибольший элемент: " + str(max)) |
| Матрица:[[ 8 -2 2 -1 9 1]  [ 7 9 -8 8 -8 -4]  [ 8 9 2 -1 9 -9]  [ 3 6 2 -5 -3 -5]]  Наибольший элемент: 8 |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  average = np.average(A, axis=1)  max = np.max(average)  print("Наибольшее среднее значение: " + str(max)) |
| Матрица:[[ 7 -9 -2 9 8 6]  [ 0 9 -7 -10 5 -7]  [ -5 8 -4 -8 -7 8]  [ 3 -8 3 -4 4 3]]  Наибольшее среднее значение: 3.1666666666666665 |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  sum = A.sum(axis=0)  i = sum.argmin(axis=0)  min = A.min(axis=0)  min = min[i]  print("Минимальный элемент: " + str(min)) |
| Матрица:[[ 0 -10 5 -9 -7 6]  [ -5 -3 5 -8 9 -1]  [ 5 6 -3 -8 -6 3]  [-10 6 2 -7 -6 4]]  Минимальный элемент: -9 |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  Average = A.mean(axis=1)  index = Average.argmin(axis=0)  min = Average.min(axis=0)  print("Наименьшее значение среди средних значений: " + str(min)) |
| Матрица:[[-6 4 3 -6 -5 9]  [-4 6 -6 -5 -8 -3]  [ 6 2 -3 -4 -5 3]  [-6 -1 4 0 9 3]]  Наименьшее значение среди средних значений: -3.3333333333333335 |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  average\_rows = np.average(A, axis=1)  average\_cols = np.average(A, axis=0)  A = np.vstack((A, [average\_cols]))  A = np.column\_stack((A, np.append(average\_rows, 0)))  print("Новая матрица: " + str(A)) |
| Матрица:[[-6 2 -3 9 6 -6]  [ 7 1 1 -3 -7 -1]  [ 4 -5 3 8 1 -5]  [ 0 -5 -9 -6 6 8]]  Новая матрица: [[-6. 2. -3. 9. 6. -6.  0.33333333]  [ 7. 1. 1. -3. -7. -1.  -0.33333333]  [ 4. -5. 3. 8. 1. -5.  1. ]  [ 0. -5. -9. -6. 6. 8.  -1. ]  [ 1.25 -1.75 -2. 2. 1.5 -1.  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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  sum\_e = np.sum(A)  sum\_c = np.sum(A, axis=0)  A = np.vstack((A, [sum\_c/sum\_e]))  print("Новая матрица: " + str(A)) |
| Матрица:[[ -4 -10 7 -9 -7 -5]  [-10 -9 -4 7 3 2]  [ -9 6 8 7 -6 2]  [ 8 -10 2 4 -7 -2]]  Новая матрица: [[ -4. -10. 7. -9. -7.  -5. ]  [-10. -9. -4. 7. 3.  2. ]  [ -9. 6. 8. 7. -6.  2. ]  [ 8. -10. 2. 4. -7.  -2. ]  [ 0.41666667 0.63888889 -0.36111111 -0.25 0.47222222  0.08333333]] |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  sum\_elements = np.sum(A)  sum\_cols = np.sum(A, axis=1)  A = np.column\_stack((A, sum\_cols/sum\_elements))  print("Новая матрица: " + str(A)) |
| Матрица:[[ -8 -4 -5 2 1 1]  [ -1 -9 3 1 9 1]  [-10 7 4 1 8 7]  [ -5 3 -9 -9 4 -7]]  Новая матрица: [[ -8. -4. -5. 2. 1.  1. 0.86666667]  [ -1. -9. 3. 1. 9.  1. -0.26666667]  [-10. 7. 4. 1. 8.  7. -1.13333333]  [ -5. 3. -9. -9. 4.  -7. 1.53333333]] |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  A\_bool = A < 0  num\_cols = np.sum(A\_bool, axis=0)  num\_rows = np.sum(A\_bool, axis=1)  A = np.vstack((A, [num\_cols]))  A = np.column\_stack((A, np.append(num\_rows, 0)))  print("Новая матрица: " + str(A)) |
| Матрица:[[ -9 -10 8 -5 1 -10]  [ -4 -8 -10 -10 6 7]  [ 8 -9 -2 -1 -5 4]  [ -6 6 -3 9 -9 4]]  Новая матрица: [[ -9 -10 8 -5 1 -10 4]  [ -4 -8 -10 -10 6 7 4]  [ 8 -9 -2 -1 -5 4 4]  [ -6 6 -3 9 -9 4 3]  [ 3 3 3 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 = 6  K = 3  L = 2  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  A\_slice = A[:L, :K]  print("Срез: "+str(A\_slice))  slice\_bool = A\_slice == 0  zero\_elements = slice\_bool.sum()  print("Количество элементов: "+str(zero\_elements)) |
| Матрица:[[ 3 -2 -3 -4 -6 9]  [ -8 5 3 -7 -5 -7]  [ -3 7 -3 9 -5 -10]  [ 8 1 -6 1 5 6]]  Срез: [[ 3 -2 -3]  [-8 5 3]]  Количество элементов: 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 = 6  K = 3  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  a = A[:, K]  a = a.reshape(len(a), 1)  A = A \* a  print("Новая матрица: " + str(A)) |
| Матрица:[[ 5 3 -2 7 -5 -6]  [ 8 0 -8 -5 5 5]  [ 6 6 -1 -3 -3 -3]  [-4 8 -1 1 7 7]]  Новая матрица: [[ 35 21 -14 49 -35 -42]  [-40 0 40 25 -25 -25]  [-18 -18 3 9 9 9]  [ -4 8 -1 1 7 7]] |

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 = 6  L = 3  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  A = A + A[L]  print("Новая матрица: " + str(A)) |
| Матрица:[[ 0 5 0 -5 -5 -3]  [-10 6 1 -7 6 8]  [ -1 2 -10 5 -5 9]  [ 0 -9 -4 -8 -9 6]]  Новая матрица: [[ 0 -4 -4 -13 -14 3]  [-10 -3 -3 -15 -3 14]  [ -1 -7 -14 -3 -14 15]  [ 0 -18 -8 -16 -18 12]] |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  max = np.max(A, axis=1)  A = A / max.reshape(len(max), 1)  print("Новая матрица: " + str(A)) |
| Матрица:[[-3 2 -9 6 5 2]  [-3 -8 2 1 6 3]  [ 3 -7 8 8 -6 -9]  [ 8 0 8 -2 -6 6]]  Новая матрица: [[-0.5 0.33333333 -1.5 1. 0.83333333 0.33333333]  [-0.5 -1.33333333 0.33333333 0.16666667 1. 0.5 ]  [ 0.375 -0.875 1. 1. -0.75 -1.125 ]  [ 1. 0. 1. -0.25 -0.75 0.75 ]] |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  max = np.max(A, axis=0)  A = A / max  print("Новая матрица: " + str(A)) |
| Матрица:[[ 3 -4 2 -8 -2 5]  [ 5 -6 0 -1 6 9]  [ 3 4 7 1 6 8]  [ 1 5 2 6 1 7]]  Новая матрица: [[ 0.6 -0.8 0.28571429 -1.33333333 -0.33333333 0.55555556]  [ 1. -1.2 0. -0.16666667 1. 1. ]  [ 0.6 0.8 1. 0.16666667 1. 0.88888889]  [ 0.2 1. 0.28571429 1. 0.16666667 0.77777778]] |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  max = np.max(A)  A = A / max  print("Новая матрица: " + str(A)) |
| Матрица:[[-6 -9 7 8 0 0]  [ 3 2 -9 1 -7 -8]  [-7 2 -2 3 6 7]  [ 5 3 6 -1 -7 -4]]  Новая матрица: [[-0.75 -1.125 0.875 1. 0. 0. ]  [ 0.375 0.25 -1.125 0.125 -0.875 -1. ]  [-0.875 0.25 -0.25 0.375 0.75 0.875]  [ 0.625 0.375 0.75 -0.125 -0.875 -0.5 ]] |

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 = 6  H = 4  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  A\_bool = A == H  row\_sum = np.sum(A\_bool, axis=0)  print("Столбцы в которых встречается значение:" + str(H))  print(np.argwhere(row\_sum).flatten())  print("Столбцы в которых нет значения:" + str(H))  print(np.argwhere(row\_sum == 0).flatten()) |
| Матрица:[[ 7 6 4 5 -9 1]  [ 3 5 3 7 6 7]  [ 3 -3 -3 -10 -7 2]  [ 3 -2 -1 -4 6 2]]  Столбцы в которых встречается значение:4  [2]  Столбцы в которых нет значения:4  [0 1 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 = 6  L = 3  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  A = np.delete(A, L, axis=0)  print("Новая матрица: " + str(A)) |
| Матрица:[[ 8 -6 -10 -2 2 -2]  [ 8 -7 4 -4 -8 -5]  [ 6 -4 -7 -10 -2 -6]  [ 0 3 5 -7 4 -10]]  Новая матрица: [[ 8 -6 -10 -2 2 -2]  [ 8 -7 4 -4 -8 -5]  [ 6 -4 -7 -10 -2 -6]] |

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 = 6  L = 3  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  row = np.random.randint(-9, 10, M)  print("Строка для вставки: " + str(row))  A = np.insert(A, L, row, axis=0)  print("Новая матрица: " + str(A)) |
| Матрица:[[ 3 -7 -1 9 7 -3]  [-2 -1 -3 1 -7 5]  [-8 6 -3 9 4 8]  [-2 -3 4 -4 4 3]]  Строка для вставки: [ 4 8 8 4 6 -2]  Новая матрица: [[ 3 -7 -1 9 7 -3]  [-2 -1 -3 1 -7 5]  [-8 6 -3 9 4 8]  [ 4 8 8 4 6 -2]  [-2 -3 4 -4 4 3]] |

18. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти сумму 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 = 6  L = 3  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  diagonal\_main = np.diagonal(A)  print("Элементы главной диагонали: "+ str(diagonal\_main))  sum\_diagonal\_main = np.sum(diagonal\_main)  print("Cумма элементов главной диагонали: "+ str(sum\_diagonal\_main))  diagonal\_side = np.diagonal(A[::-1])  print("Элементы побочной диагонали: "+ str(diagonal\_side))  sum\_diagonal\_side = np.sum(diagonal\_side)  print("сумму элементов побочной диагонали: "+ str(sum\_diagonal\_side)) |
| Матрица:[[-10 -9 1 6 5 -3]  [ -8 8 4 4 -10 7]  [ 4 -5 -7 -2 -10 -7]  [ 2 3 9 -3 -3 -10]]  Элементы главной диагонали: [-10 8 -7 -3]  Cумма элементов главной диагонали: -12  Элементы побочной диагонали: [ 2 -5 4 6]  сумму элементов побочной диагонали: 7 |

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 = 6  A = np.random.randint(-10, 10, (N, N))  print("Матрица:" + str(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:[[ 5 -5 -1 9]  [ 3 -5 -10 -2]  [ -2 -8 -9 3]  [-10 -4 -1 6]]  Элементы которые выше главной диагонали:  [ -5 -10 3]  Их сумма = -12  Элементы которые ниже главной диагонали:  [ 3 -8 -1]  Их сумма = -12 |

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 = 6  A = np.random.randint(-10, 10, (N, N))  print("Матрица:" + str(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:[[-4 8 -2 6]  [ 7 9 7 -7]  [ 0 7 8 -3]  [ 3 3 1 2]]  Элементы которые выше побочной диагонали:  [-2 9 0]  Их сумма = 0  Элементы которые ниже побочной диагонали:  [-7 8 3]  Их сумма = -168 |

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 = 6  A = np.random.randint(-10, 10, (N, N))  print("Матрица:" + str(A))  for i in range(1, N-1):  b = (A[i+1, i-1] + A[i-1, i+1])/2  A[i+1, i-1] = b  A[i-1, i+1] = b  print("Новая матрица: " + str(A)) |
| Матрица:[[-1 4 -9 2]  [-4 -2 -9 8]  [-8 -2 -5 5]  [ 3 -7 1 7]]  Новая матрица: [[-1 4 -8 2]  [-4 -2 -9 0]  [-8 -2 -5 5]  [ 3 0 1 7]] |

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 = 6  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  a = [i % 2 for i in np.sum(A, axis=1)]  A = np.insert(A, M, a, axis=1)  print("Новая матрица: " + str(A)) |
| Матрица:[[-5 -9 7 8 1 2]  [ 8 -2 -1 -6 -7 0]  [-7 -1 -3 3 2 5]  [-8 0 -6 -1 7 4]]  Новая матрица: [[-5 -9 7 8 1 2 0]  [ 8 -2 -1 -6 -7 0 0]  [-7 -1 -3 3 2 5 1]  [-8 0 -6 -1 7 4 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 = 6  A = np.random.randint(-10, 10, (N, N))  print("Матрица:" + str(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:[[ 3 -3 3 7]  [-8 -4 -8 -9]  [ 0 4 0 7]  [-5 -2 1 -4]]  Cумма элементов выше главной диагонали = -3  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 = 6  K = 2  L = 3  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  parts = [  A[:L, :K],  A[:L, K:],  A[L:, :K],  A[L:, K:],]  for i in range(len(parts)):  print("Cумма элементов" + str(i+1) + "части:" + str(np.sum(parts[i]))) |
| Матрица:[[ 0 0 4 -3 -8 -8]  [-6 -4 -2 7 0 4]  [ 4 5 4 0 -9 8]  [ 3 4 8 0 8 0]]  Cумма элементов1части:-1  Cумма элементов2части:-3  Cумма элементов3части:7  Cумма элементов4части:16 |

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 = 6  K = 2  L = 3  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  A\_bool = A == 0  col = np.sum(A\_bool, axis=1)  A = np.insert(A, M, col, axis=1)  row = np.append(np.sum(A\_bool, axis=0), 0)  A = np.insert(A, N, row, axis=0)  print("Новая матрица: " + str(A)) |
| Матрица:[[ -3 5 7 1 8 -1]  [ 4 -5 -2 -5 -4 7]  [ 4 0 -8 -6 -7 1]  [ 8 4 -1 -10 -3 -5]]  Новая матрица: [[ -3 5 7 1 8 -1 0]  [ 4 -5 -2 -5 -4 7 0]  [ 4 0 -8 -6 -7 1 1]  [ 8 4 -1 -10 -3 -5 0]  [ 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 = 6  K = 2  L = 3  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  parts = [  A[:L, :K],  A[:L, K:],  A[L:, :K],  A[L:, K:],]  for i in range(len(parts)):  print("Cумма элементов" + str(i+1) + "части:" + str(np.average(parts[i]))) |
| Матрица:[[ -4 -8 -10 -2 -3 -9]  [ 6 -6 1 -2 -9 -6]  [ -6 5 -8 -2 7 8]  [ -8 -5 -10 -1 -2 9]]  Cумма элементов1части:-2.1666666666666665  Cумма элементов2части:-2.9166666666666665  Cумма элементов3части:-6.5  Cумма элементов4части:-1.0 |

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 = 6  H = 3  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  A\_bool = A == H  col\_sum = np.sum(A\_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()) |
| Матрица:[[ 5 -4 -3 -3 8 0]  [ 2 5 2 -7 -9 4]  [ -2 8 1 -2 -4 8]  [ -7 9 3 -10 -1 -2]]  Строки в которых встречается значение: 3  [3]  Строки в которых нет значения: 3  [0 1 2] |

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 = 6  K = 2  A = np.random.randint(-10, 10, (N, M))  print("Матрица:" + str(A))  A = np.delete(A, K, axis=1)  print("Новая матрица: " + str(A)) |
| Матрица:[[ -6 2 2 8 -6 8]  [ -9 2 6 -7 -6 -4]  [ -7 -10 -1 1 -9 3]  [ -2 -9 -6 -5 1 -9]]  Новая матрица: [[ -6 2 8 -6 8]  [ -9 2 -7 -6 -4]  [ -7 -10 1 -9 3]  [ -2 -9 -5 1 -9]] |

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 = 6  K = 2  A = np.random.randint(0, 100, (N, M))  print("Матрица: " + str(A))  col = np.random.randint(-9, 10, N)  print("Столбец для вставки: " + str(col))  A = np.insert(A, K, col, axis=1)  print("Новая матрица: " + str(A)) |
| Матрица: [[20 43 12 65 98 63]  [53 40 84 84 11 15]  [47 53 94 17 40 31]  [20 40 88 11 62 4]]  Столбец для вставки: [-7 8 -3 5]  Новая матрица: [[20 43 -7 12 65 98 63]  [53 40 8 84 84 11 15]  [47 53 -3 94 17 40 31]  [20 40 5 88 11 62 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.

|  |
| --- |
| import numpy as np  N = 4  M = 6  K = 2  A = np.random.randint(0, 100, (N, M))  print("Матрица: " + str(A))  M\_n = np.sum(A, axis=0) \* (-1)  A = np.vstack((A, M\_n))  print("Новая матрица: " + str(A)) |
| Матрица: [[96 87 64 30 50 5]  [46 71 28 36 87 11]  [68 97 12 56 50 14]  [73 35 28 51 6 61]]  Новая матрица: [[ 96 87 64 30 50 5]  [ 46 71 28 36 87 11]  [ 68 97 12 56 50 14]  [ 73 35 28 51 6 61]  [-283 -290 -132 -173 -193 -91]] |

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 = 6  K = 2  A = np.random.randint(0, 100, (N, M))  print("Матрица: " + str(A))  M\_n = np.sum(A, axis=1) \* (-1)  A = np.hstack((A, M\_n.reshape(-1, 1)))  print("Новая матрица: " + str(A)) |
| Матрица: [[21 3 45 31 21 25]  [42 82 2 74 78 7]  [28 44 54 64 51 48]  [91 81 32 99 7 96]]  Новая матрица: [[ 21 3 45 31 21 25 -146]  [ 42 82 2 74 78 7 -285]  [ 28 44 54 64 51 48 -289]  [ 91 81 32 99 7 96 -406]] |

# ГЛАВА 3: ЛИНЕЙНАЯ АЛГЕБРА

В программирование не обязательно быть математиком, но знать основы линейной алгебры не помешает. Т.к. в Numpy установлена функция которая помогает решить СЛАУ, написав при этом всего лишь одну строчку кода. Но всё таки что такое метод Гаусса?

Метод Гаусса — это метод, который применяется при решении систем линейных алгебраических уравнений и имеет следующие преимущества: отсутствует необходимость проверять систему уравнений на совместность; есть возможность решать системы уравнений, где: количество определителей совпадает с количеством неизвестных переменных; количество определителей не совпадает с количеством неизвестных переменных; определитель равен нулю. результат выдается при сравнительно небольшом количестве вычислительных операций.

Теперь решим СЛАУ методом Гаусса при помощи языка программирования Python.

Все системы записаны в csv-файле.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1,03 | 1,77 | 1,08 | 1,62 | 1,27 | 1,69 | 0,41 |
| 1,4 | 1,35 | 1,32 | 1,83 | 1,65 | 1,75 | 0,44 |
| 1,07 | 1,27 | 1,17 | 1 | 1,92 | 1,82 | 0,05 |
| 1,27 | 1,34 | 1,52 | 1,8 | 1,12 | 1,84 | 0,31 |
| 1,87 | 1,49 | 1,3 | 1,31 | 1,46 | 1,75 | 0,41 |
| 1,16 | 1,58 | 1,52 | 1,92 | 1,26 | 1,97 | 0,16 |
|  |  |  |  |  |  |  |
| 1,3 | 1,5 | 0,54 |  |  |  |  |
| 1,76 | 1,24 | 0,66 |  |  |  |  |
|  |  |  |  |  |  |  |
| 1,46 | 1,34 | 1,41 | 1,9 | 1,33 | 1,27 | 0 |
| 1,84 | 1,76 | 1,28 | 1,86 | 1,25 | 1,34 | 0,23 |
| 1,42 | 1,64 | 1,59 | 1,49 | 1,77 | 1,56 | 0,95 |
| 1,07 | 1,56 | 1,46 | 1,28 | 1,03 | 1,43 | 0,13 |
| 1,17 | 1,73 | 1,95 | 1,92 | 1,96 | 1,46 | 0,19 |
| 1,76 | 1,65 | 1,5 | 1,35 | 1,12 | 1,57 | 0,72 |
|  |  |  |  |  |  |  |
| 1,02 | 1,09 | 1,1 | 1,1 | 1,96 | 1,1 | 0,11 |
| 1,05 | 1,88 | 1,28 | 1,43 | 1,78 | 1,85 | 0,07 |
| 1,9 | 1,82 | 1,62 | 1,52 | 1,73 | 1,89 | 0,21 |
| 1,71 | 1,14 | 1,53 | 1,23 | 1,42 | 1,87 | 0,89 |
| 1,5 | 1,23 | 1,92 | 1,62 | 1,52 | 1,49 | 0,4 |
| 1,41 | 1,78 | 1,28 | 1,3 | 1,48 | 1,68 | 0,41 |
|  |  |  |  |  |  |  |
| 1,39 | 1,8 | 1,85 | 1,35 | 1,73 | 1,42 | 0,71 |
| 1,75 | 1,3 | 1,95 | 1,95 | 1,54 | 1,94 | 0,21 |
| 1,29 | 1,87 | 1,36 | 1,74 | 1,03 | 1 | 0,5 |
| 1,06 | 1,17 | 1,01 | 1,13 | 1,91 | 1,21 | 0,14 |
| 1,05 | 1,7 | 1,95 | 1,95 | 1,45 | 1,99 | 0,68 |
| 1,25 | 1,68 | 1,29 | 1,43 | 1,69 | 1,77 | 0,56 |

Код:

|  |
| --- |
| import numpy as np  file = open('nympy-gauss-slv.csv', 'wb+')  file.truncate()  for i in range(1, 6):  task\_file = "task\_" + str(i) + ".csv"  m = np.genfromtxt(task\_file, delimiter=';')  A = np.genfromtxt(task\_file, delimiter=';', usecols=(range(len(m-1))))  B = np.genfromtxt(task\_file, delimiter=';', usecols=(len(m)))  print("\nМатрица:")  print(m)  slv = np.linalg.solve(A, B)  print("\n Решение системы " + str(i) + " ", slv)  np.savetxt(file, np.array([slv]), delimiter=',')  file.close() |
| Матрица:  [[1.03 1.77 1.08 1.62 1.27 1.69 0.41 nan]  [1.4 1.35 1.32 1.83 1.65 1.75 0.44 nan]  [1.07 1.27 1.17 1. 1.92 1.82 0.05 nan]  [1.27 1.34 1.52 1.8 1.12 1.84 0.31 nan]  [1.87 1.49 1.3 1.31 1.46 1.75 0.41 nan]  [1.16 1.58 1.52 1.92 1.26 1.97 0.16 nan]]  Решение системы 1 [ 8.99971813 -22.46963345 -53.59254537 9.00452064 -13.13884729  53.78136281]  Матрица:  [[1.3 1.5 0.54 nan]  [1.76 1.24 0.66 nan]]  Решение системы 2 [0.31167315 0.08988327]  Матрица:  [[1.46 1.34 1.41 1.9 1.33 1.27 0. nan]  [1.84 1.76 1.28 1.86 1.25 1.34 0.23 nan]  [1.42 1.64 1.59 1.49 1.77 1.56 0.95 nan]  [1.07 1.56 1.46 1.28 1.03 1.43 0.13 nan]  [1.17 1.73 1.95 1.92 1.96 1.46 0.19 nan]  [1.76 1.65 1.5 1.35 1.12 1.57 0.72 nan]]  Решение системы 3 [ 0.78007425 -0.30513963 -0.44769356 -1.21730619 1.09061789 0.60124927]  Матрица:  [[1.02 1.09 1.1 1.1 1.96 1.1 0.11 nan]  [1.05 1.88 1.28 1.43 1.78 1.85 0.07 nan]  [1.9 1.82 1.62 1.52 1.73 1.89 0.21 nan]  [1.71 1.14 1.53 1.23 1.42 1.87 0.89 nan]  [1.5 1.23 1.92 1.62 1.52 1.49 0.4 nan]  [1.41 1.78 1.28 1.3 1.48 1.68 0.41 nan]]  Решение системы 4 [ -2.38267858 4.05225305 11.24228091 -15.04186524 1.1868965  -0.02127095]  Матрица:  [[1.39 1.8 1.85 1.35 1.73 1.42 0.71 nan]  [1.75 1.3 1.95 1.95 1.54 1.94 0.21 nan]  [1.29 1.87 1.36 1.74 1.03 1. 0.5 nan]  [1.06 1.17 1.01 1.13 1.91 1.21 0.14 nan]  [1.05 1.7 1.95 1.95 1.45 1.99 0.68 nan]  [1.25 1.68 1.29 1.43 1.69 1.77 0.56 nan]]  Решение системы 5 [-0.24860265 0.65708781 0.29652515 -0.47899696 -0.22640873 0.25532473] |

# ЗАКЛЮЧЕНИЕ

Язык Python невероятно эффективен: программы делают больше, чем многие другие языки, в меньшем объеме кода. Синтаксис Python также позволяет писать «чистый» код. Ваш код будет легко читаться, у вас будет меньше проблем с отладкой и расширением программ по сравнению с другими языками. Python используется для разных целей: для создания игр, построения вебприложений, решений бизнес-задач и разработки внутренних инструментов для всевозможных интересных проектов. Python также широко применяется в научной области для теоретических исследований и решения прикладных задач.

Впрочем, одной из самых важных причин для использования Python - это сообщество Python, состоящее из невероятно разных и благожелательных людей. Сообщество играет исключительно важную роль в программировании, потому что программирование не является сугубо индивидуальным делом. Существование дружного, доброжелательного сообщества помогает решать задачи, и сообщество Python готово прийти на помощь людям, у которых Python является первым языком программирования.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Марк Лутц - Изучаем Python, 4-е издание
2. Прохоренок Н.А. - Python 3 и PyQt. Разработка приложений
3. Марк Саммерфилд - Программирование на Python 3. Подробное руководство
4. python.org
5. numpy.org