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 – 380305.2022.118. ПЗ КР

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

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

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

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

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

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

Работа защищена с оценкой

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

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

Челябинск 20\_\_

**АННОТАЦИЯ**

Уржунцев Ю.В

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-142, 2019.

39 с., 1 таблица, 36 рисунков.

Библиографический список – 4 наим.

В данной работе рассматриваются методы сортировки и принцип их работы, а также их эффективность. Также, рассматривается библиотека Numpy, основные ее функции и применение ее на практике.

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc9471820)

[**ГЛАВА 1 МЕТОДЫ СОРТИРОВКИ** 5](#_Toc9471821)

[**1.1 Метод сортировки выбором** 5](#_Toc9471822)

[**1.2 Обменная сортировка** 6](#_Toc9471823)

[**1.3 Сортировка включением** 8](#_Toc9471824)

[**1.4 Сравнение методов сортировки** 9](#_Toc9471825)

[**ГЛАВА 2 БИБЛИОТЕКА NUMPY** 12](#_Toc9471826)

[**2.1 Знакомство с библиотекой** 12](#_Toc9471827)

[**2.2Применение Numpy на практике** 14](#_Toc9471828)

[**ГЛАВА 3 РЕШЕНИЕ СЛАУ МЕТОДОМ ГАУССА** 36](#_Toc9471829)

[**ЗАКЛЮЧЕНИЕ** 38](#_Toc9471830)

[**БИБЛИОГРАФИЧЕСКИЙ СПИСОК** 39](#_Toc9471831)

# **ВВЕДЕНИЕ**

Целью данной работы было изучение методов сортировки, также работа заключалась в том, чтобы понять принцип действия этих методов, а также выделить самый эффективны метод. Также в ходе работы нужно было познакомиться с такой библиотекой как, Numpy. После чего нужно было решить и разобрать 31 задачу, ориентированную на раскрытие функций данной библиотеки.

Метод сортировки – это метод, который предназначен для упорядочивания элементов, расположенных в хаотичном порядке. Определенный метод по своему принципу перемещает элементы для их упорядочивания. Каждый из методов по-своему анализирует и перемещает элементы. После упорядочивания элементы будут расположены в порядке возрастания или убывания, в зависимости от представленной задачи.

Что же такое библиотека Nympy – это open-source модуль для python, который предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций. Они объединяются в высокоуровневые пакеты. Они обеспечивают функционал, который можно сравнить с функционалом MatLab. Numpy (Numeric Python) предоставляет базовые методы для манипуляции с большими массивами и матрицами. Scipy (Scientific Python) расширяет функционал Numpy огромной коллекцией полезных алгоритмов, таких как минимизация, преобразование Фурье, регрессия и другие прикладные математические техники.

Также существуют множество других библиотек, которые значительно упрощают жизнь в написании кодов и оптимизации программ.

# **ГЛАВА 1 МЕТОДЫ СОРТИРОВКИ**

## **1.1 Метод сортировки выбором**

Данный метод заключается в том, чтобы найти элемент с минимальным значением и переместить его на первое место, после чего найти следующий наименьший элемент и переместить его на второе место. После того, как все элементы будут сравнены и перемещены, вы получите полностью отсортированную числовую последовательность.

Код алгоритма можно увидеть ниже:

**def** select(arr):

count\_compare = 0 # Счетчик сравнений

count\_move = 0 # Счетчик перемещений

n = len(arr) # Количество элементов в массиве

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

min = i # Индекс минимального элемента на текущем шаге

**for** j **in** range(i+1, n): # Перебор хвоста

count\_compare += 1 # Считаем количество сравнений

**if** arr[j] < arr[min]: # Находим минимальный элемент

min = j # Записываем индекс минимального элемента

**if** min != i:

# Если индекс минимального элемента не равен индексу текущего,

# тогда меняем элементы местами

arr[i], arr[min] = arr[min], arr[i]

count\_move += 1 # Считаем количество перемещений

**return** [count\_compare, count\_move]

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** random

arry = [random.randint(0, 20) **for** i **in** range(20)]

**print**(arry)

select(arry)

**print**(arry)

Результат представлен на рисунке 1

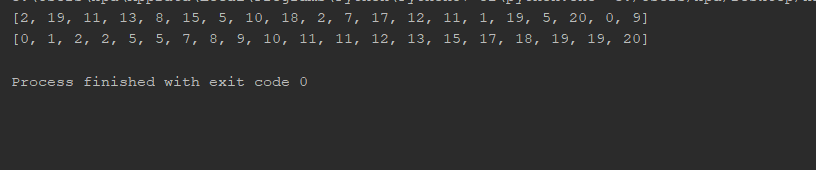


Рис. 1 Результат метода сортировки выбором

## **1.2 Обменная сортировка**

Следующий метод сортировки – это обменная сортировка, также называемая метод «пузырька». Данный метод реализуется очень просто, он сравнивает элемент с последующим, если первый элемент меньше следующего, то он оставляет его на месте, но если второй меньше, то он перемещает его перед первым элементом и так далее сравнивает все элементы. Таким образом элементы с большим значением окажутся в конце, а элементы с меньшим значением начнут перемещаться в начало.

Код алгоритма можно увидеть ниже:

**def** bubble(arr):

count\_compare = 0 # Счетчик количества сравнений

count\_move = 0 # Счетчик количества перемещений

n = len(arr) # Количество элементов массива

min\_index = 0 # Минимальный индекс элемента, для которого выполнялась перестановка

max\_index = n # Максимальный индекс элемента, для которого выполнялась перестановка

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

count\_move\_step = 0 # Количество перемещений на текущем шаге

min\_index\_step = 0 # Минимальный индекс перемещенного элемента на текущем шаге

max\_index\_step = n # Максимальный индекс перемещенного элемента на текущем шаге

**for** j **in** range(1, n):

# Если индекс стал меньше минимального индекса перемещений с предыдущего шага,

# тогда перейдем к следующему шагу

**if** n-j-1 < min\_index:

**break**

count\_compare += 1 # Считаем количество сравнений

**if** arr[n-j-1] > arr[n-j]: # Arr[n-2] > Arr[n-1] ... Arr[0] > Arr[1]

arr[n-j], arr[n-j-1] = arr[n-j-1], arr[n-j] # Меняем местами

count\_move += 1 # Считаем количество перемещений

count\_move\_step += 1 # Считаем количество перемещений на текущем шаге

min\_index\_step = n-j-1 # Запоминаем минимальный индекс перемещенного элемента на текущем шаге

# Сохраняем минимальный индекс перемещенного элемента на текущем шаге

min\_index = min\_index\_step

**for** j **in** range(1, n):

# Если индекс стал больше максимального индекса перемещений с предыдущего шага,

# тогда перейдем к следующему шагу

**if** j+1 >= max\_index:

**break**

count\_compare += 1 # Считаем количество сравнений

**if** arr[j] > arr[j+1]: # Arr[0] > Arr[1] ... Arr[n-3] > Arr[n-2]

arr[j], arr[j+1] = arr[j+1], arr[j] # Меняем местами

count\_move += 1 # Считаем количество перемещений

count\_move\_step += 1 # Считаем количество перемещений на текущем шаге

max\_index\_step = j+1 # Запоминаем максимальный индекс перемещенного элемента на текущем шаге

# Сохраняем максимальный индекс перемещенного элемента на текущем шаге

max\_index = max\_index\_step

# Если на текущем шаге не было перемещений заканчиваем сортировку

**if** count\_move\_step == 0:

**break**

**return** [count\_compare, count\_move]

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** random

arry = [random.randint(0, 20) **for** i **in** range(20)]

**print**(arry)

bubble(arry)

**print**(arry)

Результат представлен на рисунке 2

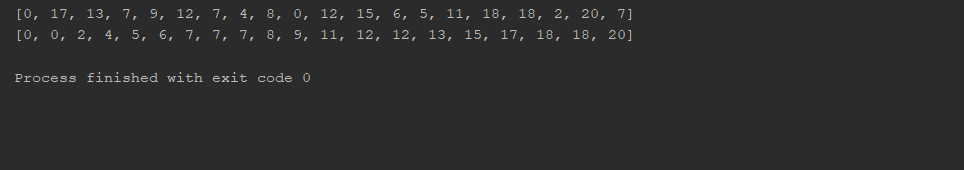


Рис. 2 Результат обменной сортировки

## **1.3 Сортировка включением**

Данный метод сортировки основан на том, что на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Например, порядок чисел 3,5 данный метод после нахождения числа «4» вставит его между 3 и 5. И таким образом отсортирует всю последовательность чисел.

Данный метод очень удобен, когда часть последовательности уже частично отсортирована.

Код алгоритма представлен ниже:

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

alg\_count = [0, 0] # массив для показателей эффективности

**for** i **in** range(1, dim): # Основной цикл со 2-го элемента право

temp = arr[i] # Запомним элемент для сравнения

j = i - 1

**while** j >= 0: # Ищем влево ближайший меньший

alg\_count[0] += 1 # Считаем операции сравнения

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

alg\_count[1] += 1 # Считаем операции перестановки

arr[j+1] = arr[j] # Сдвигаем элемент влево, а на его место ставим наименьший

arr[j] = temp

j -= 1

**return** alg\_count

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** random

arry = [random.randint(0, 20) **for** i **in** range(20)]

**print**(arry)

insert(arry, len(arry))

**print**(arry)

Результат кода представлен на рисунке 3

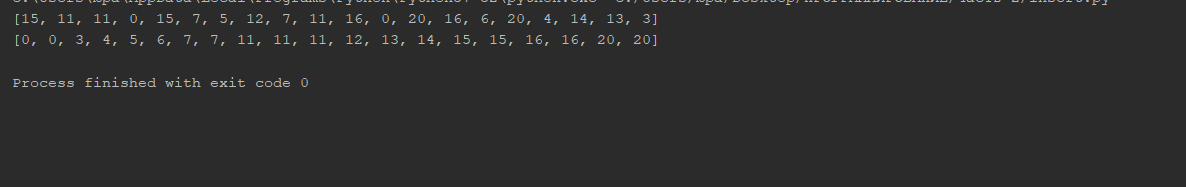


Рис.3 Результат сортировки включением

## **1.4 Сравнение методов сортировки**

После исследования всех методов сортировки необходимо провести сравнение данных методов, чтобы выяснить какой же метод является самым эффективным в использовании. Сравнение методов сортировки будет происходить по количеству сравнений и количеству перемещений элементов. Чем эти показатели ниже, тем быстрее выполняется сортировка. Для сравнения методов сортировки был написан код, который производит сравнение.

Код сравнения методов сортировки представлен ниже:

**from** random **import** randint

**from** insert **import** insert

**from** select **import** select

**from** bubble **import** bubble

N = 10

**def** get\_array(type='up'):

**if** type == 'up':

**return** [i **for** i **in** range(0, N)]

**if** type == 'down':

**return** [i **for** i **in** range(N - 1, -1, -1)]

**if** type == 'rand':

**return** [randint(0,N) **for** i **in** range(0, N)]

**def** check\_sort():

result = {

'insert': {},

'select': {},

'bubble': {},

}

array\_insert = get\_array('up')

array\_select = array\_insert.copy()

array\_bubble = array\_insert.copy()

result['insert']['up'] = insert(array\_insert, len(array\_insert))

result['select']['up'] = select(array\_select)

result['bubble']['up'] = bubble(array\_bubble)

array\_insert = get\_array('down')

array\_select = array\_insert.copy()

array\_bubble = array\_insert.copy()

result['insert']['down'] = insert(array\_insert, len(array\_insert))

result['select']['down'] = select(array\_select)

result['bubble']['down'] = bubble(array\_bubble)

avg = {

'insert': [0, 0],

'select': [0, 0],

'bubble': [0, 0]

}

**for** i **in** range(400):

array\_insert = get\_array('rand')

array\_select = array\_insert.copy()

array\_bubble = array\_insert.copy()

sort\_result = insert(array\_insert, len(array\_insert))

avg['insert'][0] += sort\_result[0]

avg['insert'][1] += sort\_result[1]

sort\_result = select(array\_select)

avg['select'][0] += sort\_result[0]

avg['select'][1] += sort\_result[1]

sort\_result = bubble(array\_bubble)

avg['bubble'][0] += sort\_result[0]

avg['bubble'][1] += sort\_result[1]

result['insert']['rand'] = [avg['insert'][0]/400, avg['insert'][1]/400]

result['select']['rand'] = [avg['select'][0]/400, avg['select'][1]/400]

result['bubble']['rand'] = [avg['bubble'][0]/400, avg['bubble'][1]/400]

**return** result

**print**( check\_sort() )

Результат предоставлен на рисунке 4

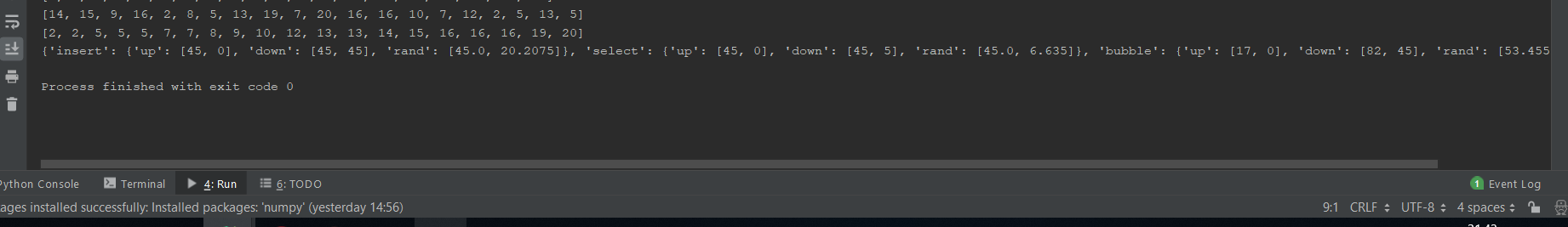


Рис.4 Результат сравнения методов сортировки

Сравнение методов сортировки было произведено сразу по трем последовательностям: упорядоченная последовательность, обратно упорядоченная последовательность, а также случайная последовательность. Сравнение производилось по двум показателям: количество сравнений (С), количество перемещений (M). Результат вы можете видеть на рисунке 4 либо на таблице 1, приведенной ниже.

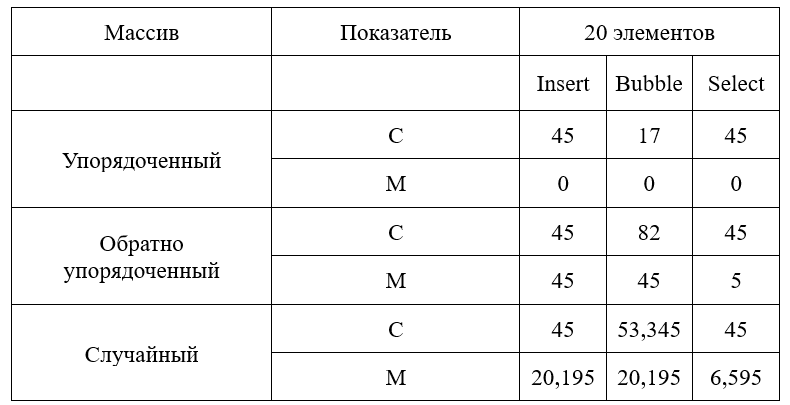


Таблица 1 – Результаты сравнения методов сортировки

Исходя из данных приведенных в таблице, можно сделать вывод, что самым эффективным методом сортировки для последовательности является метод «Select». Этот вывод я сделал на основании тех показателей, по которым происходило сравнение. Метод «Select» или метод сортировки выбором является самым эффективным методом сортировки, так как для упорядочивания случайной последовательности значений этому методу потребовалась гораздо меньше перемещений чем остальным методам.

# **ГЛАВА 2 БИБЛИОТЕКА NUMPY**

## **2.1 Знакомство с библиотекой**

Numpy — это open-source модуль для python, который предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций. Они объединяются в высокоуровневые пакеты. Они обеспечивают функционал, который можно сравнить с функционалом MatLab. Numpy (Numeric Python) предоставляет базовые методы для манипуляции с большими массивами и матрицами. Scipy (Scientific Python) расширяет функционал Numpy огромной коллекцией полезных алгоритмов, таких как минимизация, преобразование Фурье, регрессия и другие прикладные математические техники.

Назначение. Математические алгоритмы, реализованные на [интерпретируемых языках](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) (например, Python), часто работают гораздо медленнее тех же алгоритмов, реализованных на [компилируемых языках](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) (например, [Фортран](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D1%82%D1%80%D0%B0%D0%BD), [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Java](https://ru.wikipedia.org/wiki/Java" \o "Java)). Библиотека Numpy предоставляет реализации [вычислительных](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) [алгоритмов](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)(в виде функций и операторов), оптимизированные для работы с многомерными массивами. В результате любой алгоритм, который может быть выражен в виде последовательности операций над массивами (матрицами) и реализованный с использованием Numpy, работает так же быстро, как эквивалентный код, выполняемый в [MATLAB](https://ru.wikipedia.org/wiki/MATLAB).

История. В [1995 году](https://ru.wikipedia.org/wiki/1995_%D0%B3%D0%BE%D0%B4) программист Jim Hugunin написал библиотеку Numeric для языка Python. Библиотека развивалась при участии многих людей, среди которых были Jim Fulton, David Ascher, Paul DuBois и Konrad Hinsen. Библиотека доступна по сей день, считается вполне стабильной и полной, но устаревшей.

Предлагалось добавить Numeric в стандартную библиотеку языка Python, но [Гвидо Ван Россум](https://ru.wikipedia.org/wiki/%D0%92%D0%B0%D0%BD_%D0%A0%D0%BE%D1%81%D1%81%D1%83%D0%BC,_%D0%93%D0%B2%D0%B8%D0%B4%D0%BE) (автор Python) чётко дал понять, что код в его тогдашнем состоянии было невозможно поддерживать.

Кроме того, библиотека Numeric медленно обрабатывала большие массивы данных.

На основе библиотеки Numeric была создана библиотека NumArray. Код Numeric был полностью переписан.

Библиотека NumArray обрабатывала большие массивы данных быстрее библиотеки Numeric, но малые массивы обрабатывала медленнее.

Некоторое время использовалась и библиотека Numeric, и библиотека NumArray. Последняя версия Numeric (v24.2) была выпущена [11 ноября](https://ru.wikipedia.org/wiki/11_%D0%BD%D0%BE%D1%8F%D0%B1%D1%80%D1%8F) [2005 года](https://ru.wikipedia.org/wiki/2005_%D0%B3%D0%BE%D0%B4). Последняя версия NumArray (v1.5.2) вышла [24 августа](https://ru.wikipedia.org/wiki/24_%D0%B0%D0%B2%D0%B3%D1%83%D1%81%D1%82%D0%B0) [2006 года](https://ru.wikipedia.org/wiki/2006_%D0%B3%D0%BE%D0%B4)[[3]](https://ru.wikipedia.org/wiki/NumPy#cite_note-3). Библиотека NumArray более не рекомендуется к использованию

В начале 2005 года программист Трэвис Олифант захотел объединить сообщество вокруг одного проекта и для замены библиотек Numeric и NumArray создал библиотеку NumPy. NumPy был создан на основе кода Numeric. Код Numeric был переписан так, чтобы его было легче поддерживать, и в библиотеку можно было добавить новые возможности. Возможности NumArray были добавлены в NumPy.

Изначально NumPy был частью библиотеки SciPy. Чтобы позволить другим проектам использовать библиотеку NumPy, её код был помещён в отдельный пакет.

Исходный код NumPy находится в открытом доступе. Существует большое количество документации. Имеется даже подробный «*Путеводитель по NumPy*»].

NumPy v1.3.0 выпущен [5 апреля](https://ru.wikipedia.org/wiki/5_%D0%B0%D0%BF%D1%80%D0%B5%D0%BB%D1%8F) [2009 года](https://ru.wikipedia.org/wiki/2009_%D0%B3%D0%BE%D0%B4) и поддерживает Python v2.6. Поддержка Python v3 была добавлена начиная с версии 1.5.0.

**2.2Применение Numpy на практике**

Для раскрытия функций данной библиотеки мне была предоставлена 31 задача. Ниже вы можете видеть решение каждой из них.

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

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

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

sum\_cols = np.sum(np.absolute(matrix), axis=0)

**print**("Суммы столбцов:\r\n {}".format(sum\_cols))

max\_sum = np.max(sum\_cols)

max\_sum\_col = np.argmax(sum\_cols)

col = matrix[:, max\_sum\_col]

**print**("Наибольшая сумма: {} в столбце №{}".format(max\_sum, max\_sum\_col+1))

**print**("Элементы столбца: {}".format(col))

max\_el = np.max(col)

**print**("Наибольший элемент: {}".format(max\_el))

Результат задачи 1 представлен на рисунке 5

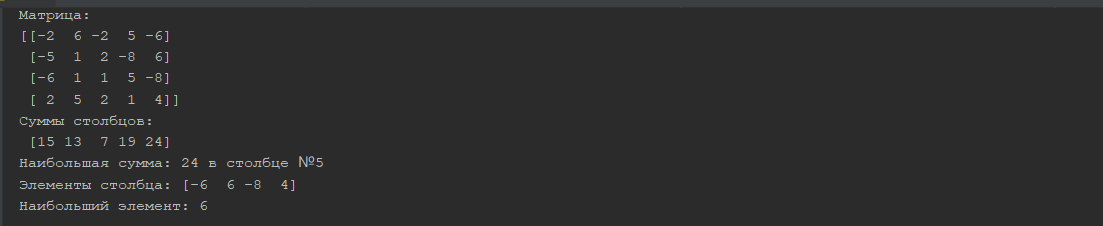


Рис.5 Результат задачи 1

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

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

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

average = np.average(matrix, axis=1)

**print**("Cредние значения для каждой строки:\r\n {}".format(average))

max\_average = np.max(average)

**print**("Наибольшее среднее значение: {}".format(max\_average))

Результат задачи 2 представлен на рисунке 6

Рис. 6 Результат задачи 2

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

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

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

sum\_cols = np.sum(np.absolute(matrix), axis=0)

**print**("Суммы столбцов:\r\n {}".format(sum\_cols))

max\_sum = np.max(sum\_cols)

max\_sum\_col = np.argmax(sum\_cols)

col = matrix[:, max\_sum\_col]

**print**("Наибольшая сумма: {} в столбце №{}".format(max\_sum, max\_sum\_col+1))

**print**("Элементы столбца: {}".format(col))

min\_el = np.min(col)

**print**("Наименьший элемент: {}".format(min\_el))

Результат задачи 3 представлен на рисунке 7

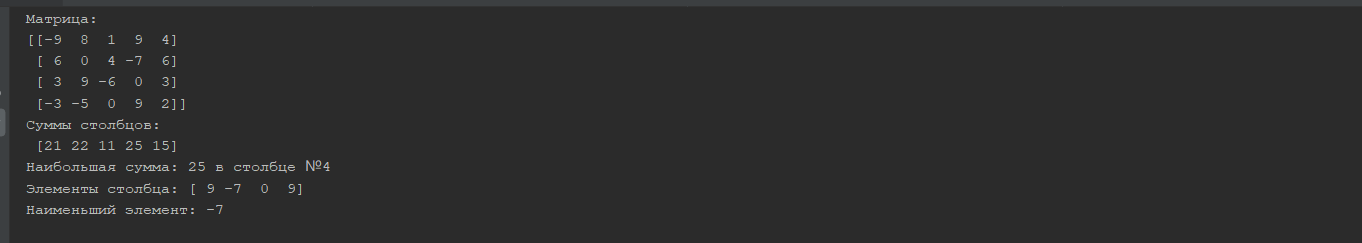


Рис.7 Результат задачи 3

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

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

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

average = np.average(matrix, axis=1)

**print**("Cредние значения для каждой строки:\r\n {}".format(average))

min\_average = np.min(average)

**print**("Наименьшее среднее значение: {}".format(min\_average))

Результат задачи 4 представлен на рисунке 8



Рис.8 Результат задачи 4

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

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

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

average\_rows = np.average(matrix, axis=1)

**print**("Cредние значения для каждой строки:\r\n {}".format(average\_rows))

average\_cols = np.average(matrix, axis=0)

**print**("Cредние значения для каждого столбца:\r\n {}".format(average\_cols))

matrix = np.vstack((matrix, [average\_cols]))

matrix = np.column\_stack((matrix, np.append(average\_rows, 0)))

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 5 представлен на рисунке 9

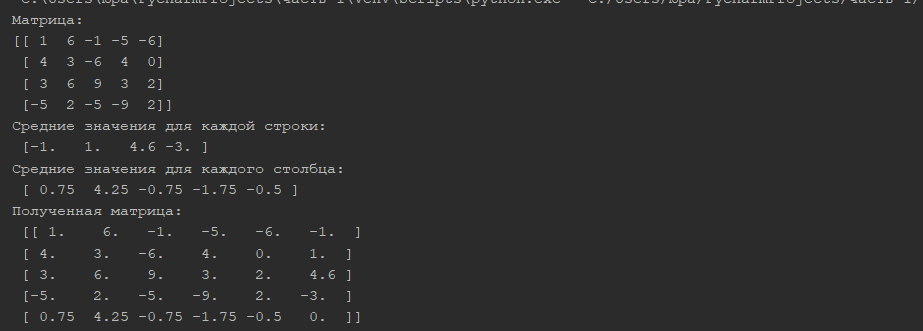


Рис. 9 Результат задачи 5

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

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

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

sum\_elements = np.sum(matrix)

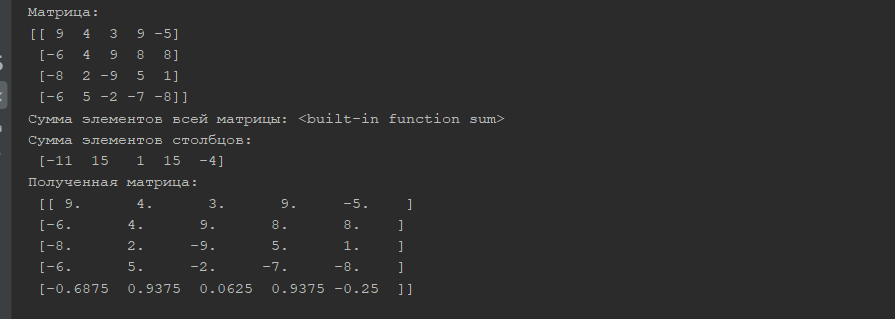
**print**("Cумма элементов всей матрицы: {}".format(sum))

sum\_cols = np.sum(matrix, axis=0)

**print**("Cумма элементов столбцов:\r\n {}".format(sum\_cols))

matrix = np.vstack((matrix, [sum\_cols/sum\_elements]))

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 6 представлен на рисунке 10Рис.10 Результат задачи 6

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

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

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

sum\_elements = np.sum(matrix)

**print**("Cумма элементов всей матрицы: {}".format(sum))

sum\_cols = np.sum(matrix, axis=1)

**print**("Cумма элементов строк:\r\n {}".format(sum\_cols))

matrix = np.column\_stack((matrix, sum\_cols/sum\_elements))

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 7 представлен на рисунке 11

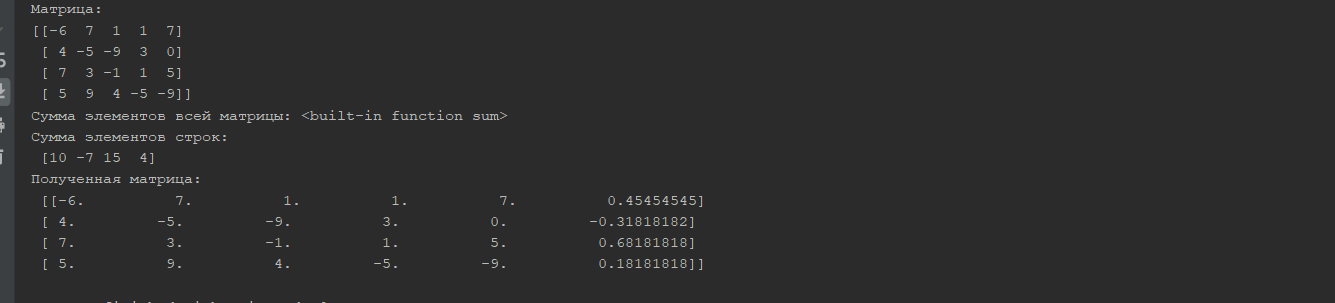


Рис.11 Результат задачи 7

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

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

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

matrix\_bool = matrix < 0

num\_cols = np.sum(matrix\_bool, axis=0)

num\_rows = np.sum(matrix\_bool, axis=1)

matrix = np.vstack((matrix, [num\_cols]))

matrix = np.column\_stack((matrix, np.append(num\_rows, 0)))

**print**("Количество отрицательных элементов в каждом столбце и в каждой строке")

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 8 представлен на рисунке 12

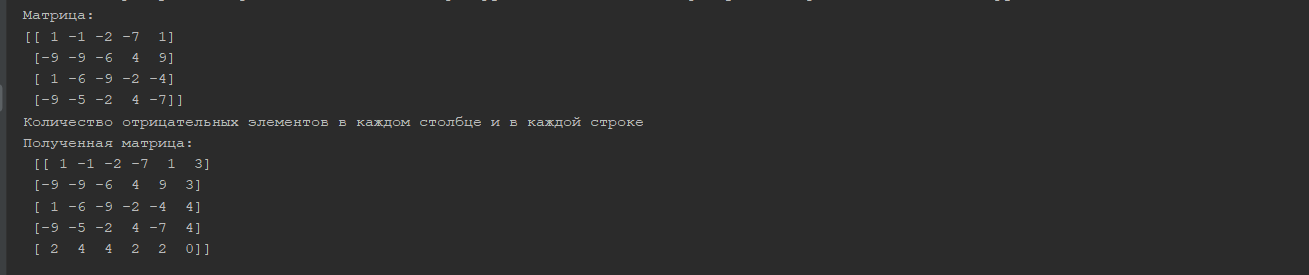


Рис.12 Результат задачи 8

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 = 2

K = 2

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

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

matrix\_slice = matrix[:L, :K]

**print**("Срез:\r\n{}".format(matrix\_slice))

slice\_bool = matrix\_slice == 0

count\_zero\_elements = slice\_bool.sum()

**print**("Количество нулевых элементов в срезе: {}".format(count\_zero\_elements))

Результат задачи 9 представлен на рисунке 13

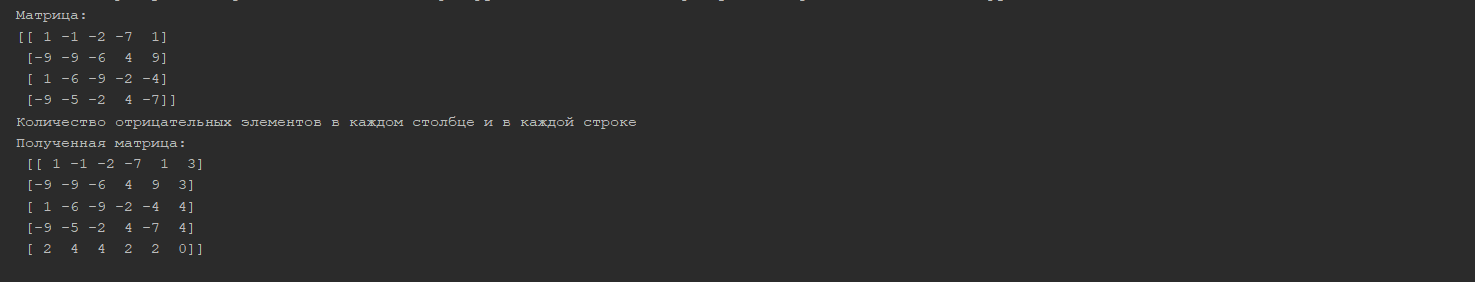


Рис.13 Результат задачи 9

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 = 2

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

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

col = matrix[:, K]

col = col.reshape(len(col), 1)

matrix = matrix \* col

**print**("Полученная матрица:\r\n {}".format(matrix))

slice\_bool = matrix\_slice == 0

count\_zero\_elements = slice\_bool.sum()

**print**("Количество нулевых элементов в срезе: {}".format(count\_zero\_elements))

Результат задачи 10 представлен на рисунке 14

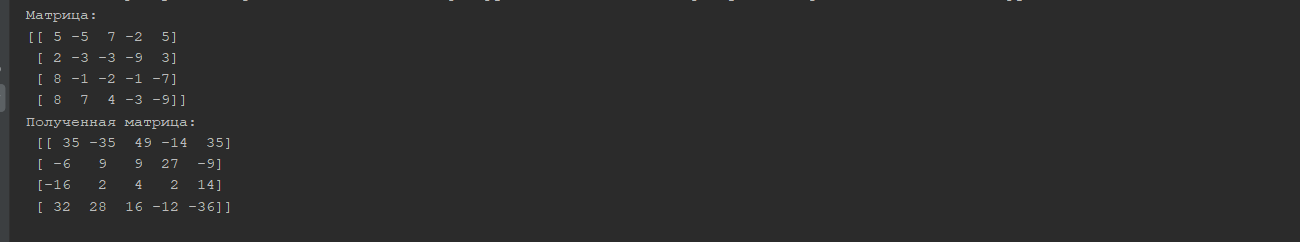


Рис.14 Результат задачи 10

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 = 2

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

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

matrix = matrix + matrix[L]

**print**("Полученная матрица:\r\n {}".format(matrix))

**print**("Полученная матрица:\r\n {}".format(matrix))

slice\_bool = matrix\_slice == 0

count\_zero\_elements = slice\_bool.sum()

**print**("Количество нулевых элементов в срезе: {}".format(count\_zero\_elements))

Результат задачи 11 представлен на рисунке 15

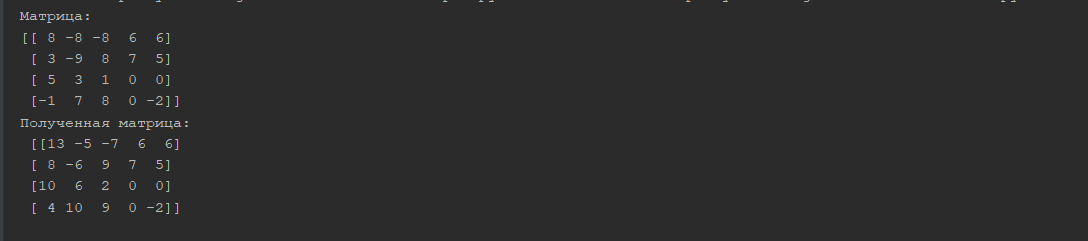


Рис.15 Результат задачи 11

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

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

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

row\_max = np.max(matrix, axis=1)

matrix = matrix / row\_max.reshape(len(row\_max), 1)

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 12 представлен на рисунке 16

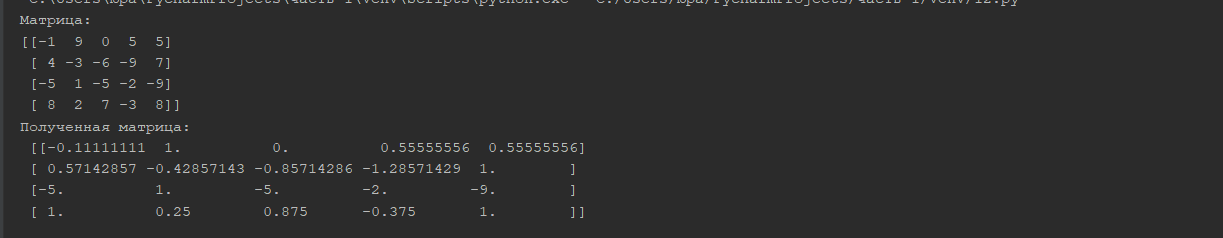


Рис.16 Результат задачи 12

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

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

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

col\_max = np.max(matrix, axis=0)

matrix = matrix / col\_max

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 13 представлен на рисунке 17



Рис.17 Результат задачи 13

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

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

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

max\_el = np.max(matrix)

matrix = matrix / max\_el

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 14 представлен на рисунке 18

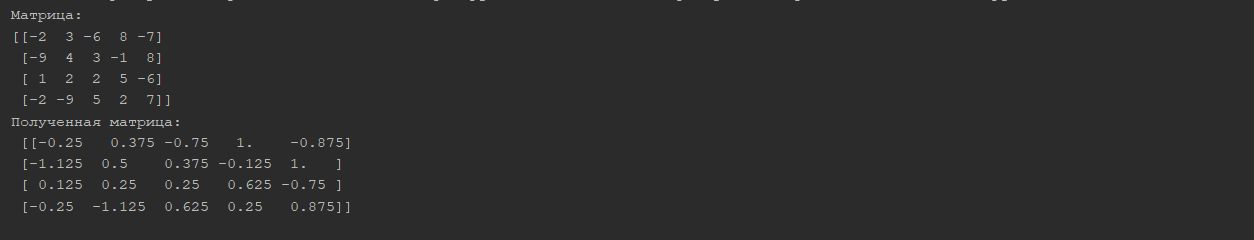


Рис.18 Результат задачи 14

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 = 2

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

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

matrix\_bool = matrix == H

row\_sum = np.sum(matrix\_bool, axis=0)

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

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

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

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

Результат задачи 15 представлен на рисунке 19



Рис.19 Результат задачи 15

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 = 2

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

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

matrix = np.delete(matrix, L, axis=0)

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 16 представлен на рисунке 20

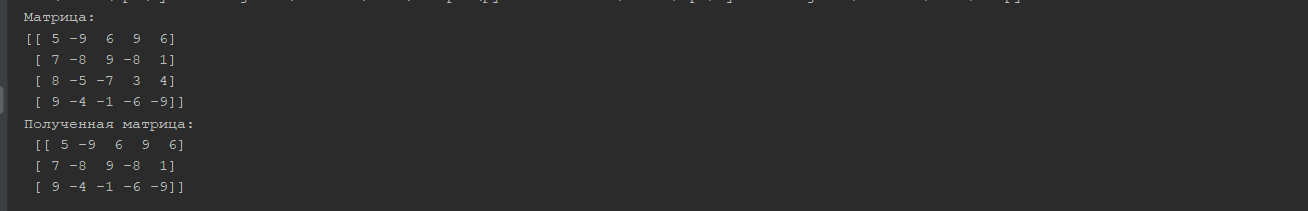


Рис.20 Результат задачи 16

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 = 1

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

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

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

**print**("Строка для вставки: {}".format(row))

matrix = np.insert(matrix, L, row, axis=0)

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 17 представлен на рисунке 21

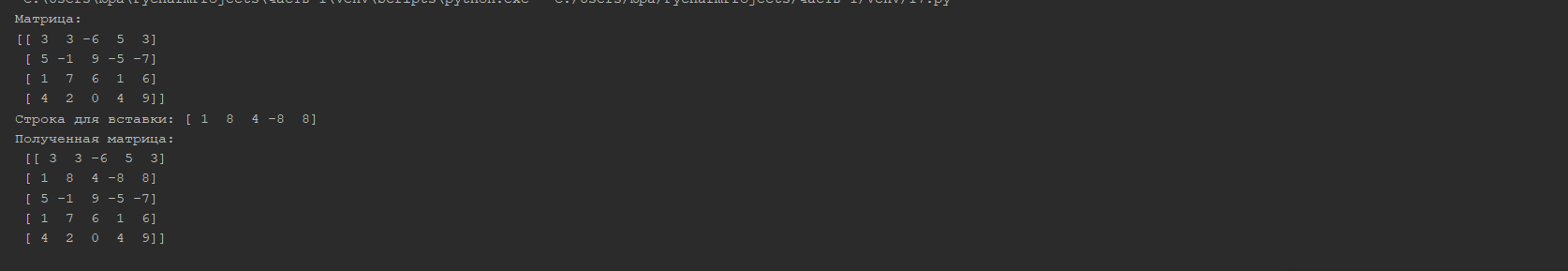


Рис.21 Результат задачи 17

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

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

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

diagonal\_main = np.diagonal(matrix)

**print**("Элементы главной диагонали:\r\n{}".format(diagonal\_main))

sum\_diagonal\_main = np.sum(diagonal\_main)

**print**("Cумма элементов главной диагонали:\r\n{}".format(sum\_diagonal\_main))

diagonal\_side = np.diagonal(matrix[::-1])

**print**("Элементы побочной диагонали:\r\n{}".format(diagonal\_side))

sum\_diagonal\_side = np.sum(diagonal\_side)

**print**("сумму элементов побочной диагонали:\r\n{}".format(sum\_diagonal\_side))

Результат задачи 18 представлен на рисунке 22



Рис.22 Результат задачи 18

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

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

**print**("Матрица:\r\n{}".format(A))

diagonal\_elements = np.array([np.diagonal(A, i) **for** i **in** [1, -1]]).flatten()

**print**("Элементы, расположенные параллельно главной диагонали:")

**print**(diagonal\_elements)

**print**("Сумма элементов, расположенные параллельно главной диагонали:")

**print**(np.sum(diagonal\_elements))

Результат задачи 19 представлен на рисунке 23

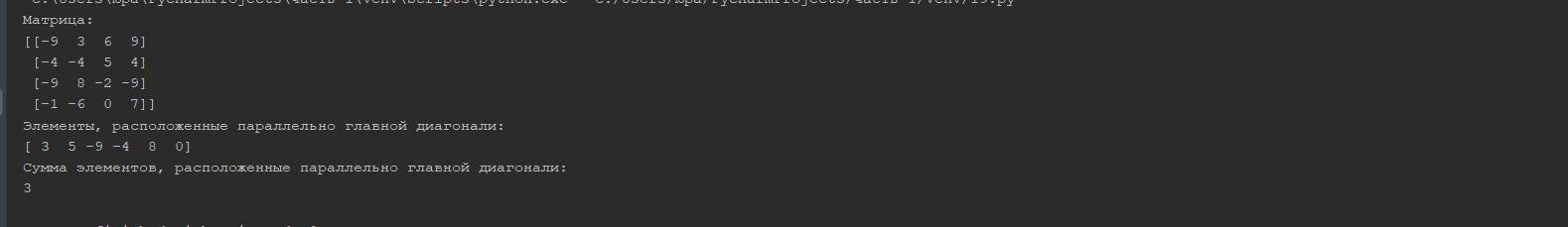


Рис.23 Результат задачи 19

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].

**import** numpy **as** np

N = 4

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

**print**("Матрица:\r\n{}".format(A))

diagonal\_elements = np.array([np.diagonal(A[::-1], i) **for** i **in** [1, -1]]).flatten()

**print**("Элементы, расположенные параллельно побочной диагонали:")

**print**(diagonal\_elements)

**print**("Сумма элементов, расположенные параллельно побочной диагонали:")

**print**(np.prod(diagonal\_elements))

Результат задачи 20 представлен на рисунке 24



Рис.24 Результат задачи 20

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

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

**print**("Матрица:\r\n{}".format(A))

diagonal\_elements = [np.diagonal(A, i) **for** i **in** [1, -1]]

**print**("Элементы, расположенные параллельно главной диагонали:")

**print**(diagonal\_elements)

values = (diagonal\_elements[0] + diagonal\_elements[1])/2

**print**(values)

rng = np.arange(N-1)

A[rng, rng+1] = values[rng]

A[rng+1, rng] = values[rng]

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

Результат задачи 21 представлен на рисунке 25

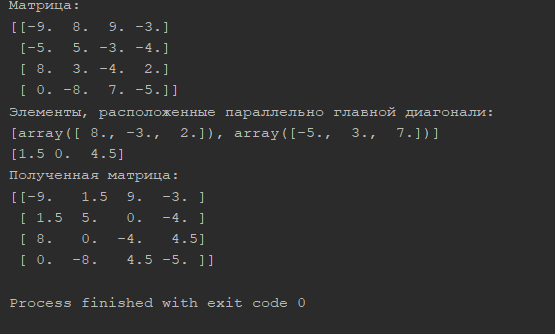


Рис.25 Результат задачи 21

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=0, high=2, size=(N, M))

**print**("Матрица:\r\n{}".format(A))

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

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

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

Результат задачи 22 представлен на рисунке 26



Рис.26 Результат задачи 22

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]

**import** numpy **as** np

N = 4

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

**print**("Матрица:\r\n{}".format(A))

elements = np.diagonal(A, 1)

**print**("Элементы, расположенные выше главной диагонали:")

**print**(elements)

**print**("Сумма элементов, расположенных выше главной диагонали:")

**print**(np.sum(elements))

elements = np.diagonal(A[::-1], -1)

**print**("Элементы, расположенные выше побочной диагонали:")

**print**(elements)

Результат задачи 23 представлен на рисунке 27



Рис.27 Результат задачи 23

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 = 2

K = 2

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

**print**("Матрица:\r\n{}".format(A))

parts = [

A[:L, :K],

A[:L, K:],

A[L:, :K],

A[L:, K:],

]

**for** i **in** range(len(parts)):

**print**("Cумма элементов {} части: {}".format(i+1, np.sum(parts[i])))

Результат задачи 24 представлен на рисунке 28

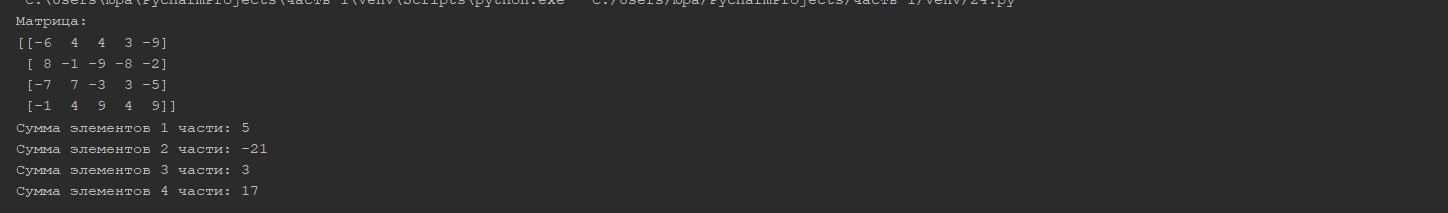


Рис.28 Результат задачи 24

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

L = 2

K = 2

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

**print**("Матрица:\r\n{}".format(A))

matrix\_bool = A == 0

col = np.sum(matrix\_bool, axis=1)

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

row = np.append(np.sum(matrix\_bool, axis=0), 0)

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

Результат задачи 25 представлен на рисунке 29

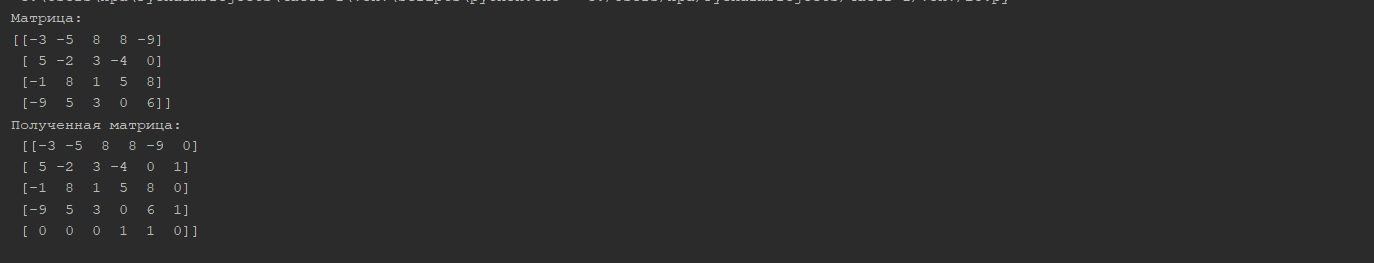


Рис.29 Результат задачи 25

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 = 2

K = 2

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

**print**("Матрица:\r\n{}".format(A))

parts = [

A[:L, :K],

A[:L, K:],

A[L:, :K],

A[L:, K:],

]

**for** i **in** range(len(parts)):

**print**("Cреднее арифметическое {} части: {}".format(i+1, np.average(parts[i])))

Результат задачи 26 представлен на рисунке 30

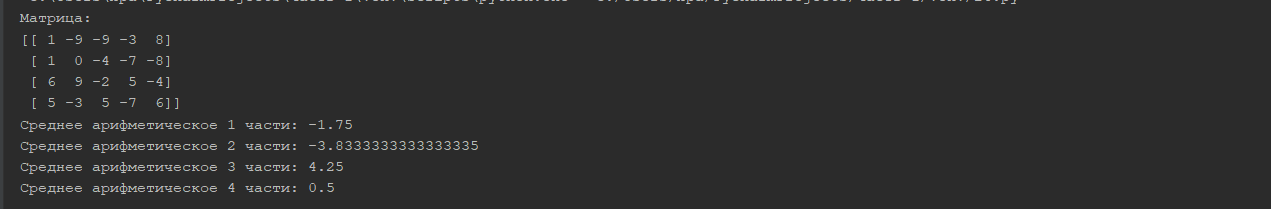


Рис.30 Результат задачи 26

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 = 2

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

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

matrix\_bool = matrix == H

col\_sum = np.sum(matrix\_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())

Результат задачи 27 представлен на рисунке 31

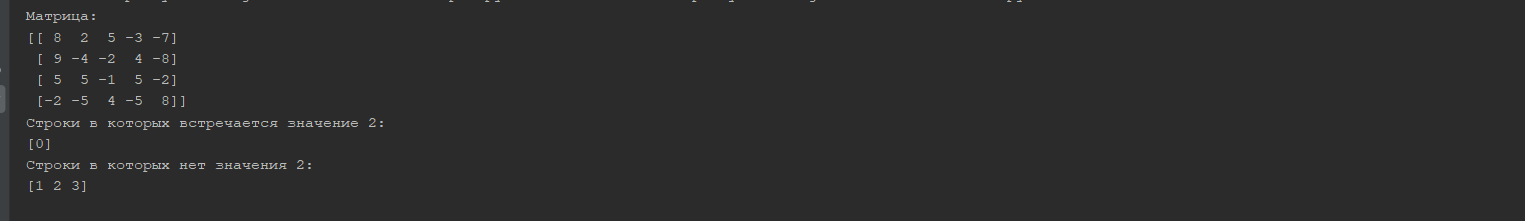


Рис.31 Результат задачи 27

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 = 2

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

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

matrix = np.delete(matrix, K, axis=1)

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 28 представлен на рисунке 32

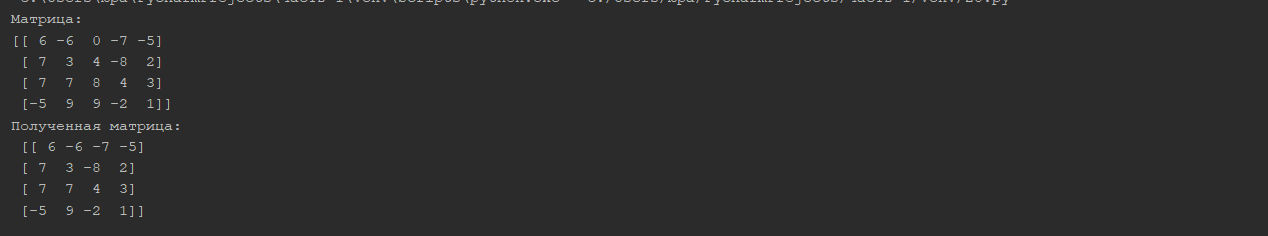


Рис.32 Результат задачи 28

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

K = 1

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

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

col = np.random.randint(low=-9, high=10, size=N)

**print**("Столбец для вставки: {}".format(col))

matrix = np.insert(matrix, K, col, axis=1)

**print**("Полученная матрица:\r\n {}".format(matrix))

Результат задачи 29 представлен на рисунке 33



Рис.33 Результат задачи 29

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

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

**print**("Матрица:\r\n{}".format(A))

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

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

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

Результат задачи 30 представлен на рисунке 34



Рис.34 Результат задачи 30

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

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

**print**("Матрица:\r\n{}".format(A))

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

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

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

Результат задачи 31 представлен на рисунке 35

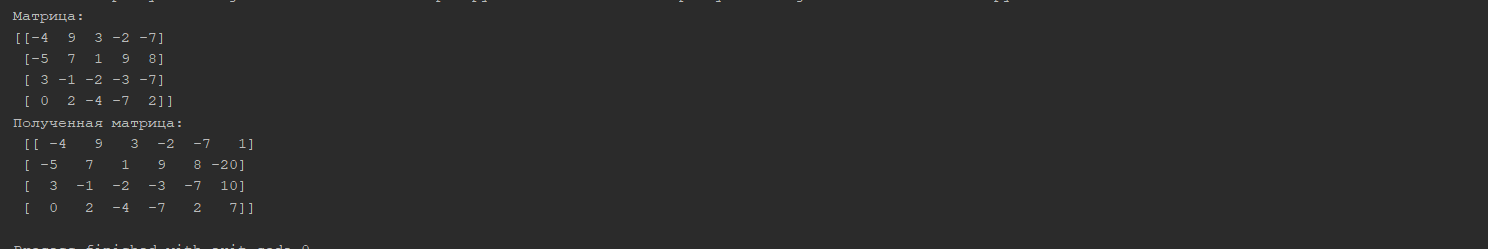


Рис.35 Результат задачи 31

Вывод: Я рассмотрел библиотеку Numpy на практике. После того, как я разобрал 31 практическую задачу, я могу сделать вывод, что библиотека существенно помогает в написании кодов. Она имеет очень полезные функции, которые сильно помогают оптимизировать код, облегчить его.

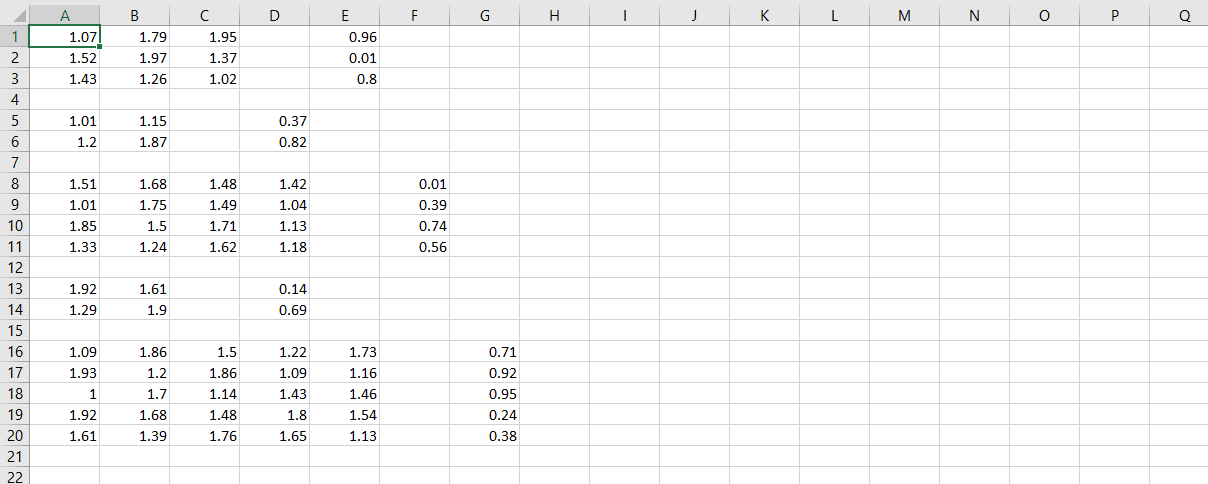
Данная библиотека поможет быстро обучить людей программированию и существенно упростит им жизнь.

**ГЛАВА 3 РЕШЕНИЕ СЛАУ МЕТОДОМ ГАУССА**

Метод Гаусса — классический метод решения [системы линейных алгебраических уравнений](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BB%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D1%85_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D1%85_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B9) (СЛАУ). Назван в честь немецкого математика [Карла Фридриха Гаусса](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D1%83%D1%81%D1%81,_%D0%9A%D0%B0%D1%80%D0%BB_%D0%A4%D1%80%D0%B8%D0%B4%D1%80%D0%B8%D1%85). Это метод последовательного исключения [переменных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0), 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.

Для решения систем линейных уравнений методом Гаусса был написан код, который решает линейное уравнение данным методом. Для оптимизации кода была проделана следующая работа:

1) Было создан файл формата csv, чтобы записать в него исходные матрицы, файл выглядит следующим образом:



2) Был оптимизирован код, таким образом, чтобы чтение происходило из файла с исходными матрицами.

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

**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]))

# загружаем данные из csv файла

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

matrix\_list = []

matrix = []

**for** row **in** data: # перебираем файл построчно

first\_col = row[0] # возьмем значение первой колонки

**if** numpy.isnan(first\_col): # если значение первой колонки пустое

matrix\_list.append(matrix) # в список матриц добавим матрицу собранную из строк

matrix = [] # очистим собираемую матрицу

**continue** # перейдем к следующей строке

mask = ~numpy.isnan(row) # получим маску непустых элементов

matrix.append(row[mask]) # вставим в матрицу непустые значения строки

matrix\_list.append(matrix) # добавим в список матриц последнюю получившуюся матрицу

f = open('numpy-gauss-slv.csv', 'wb+') # открываем файл для чтения

f.truncate() # очищаем файл

**for** matrix **in** matrix\_list: # перебираем список матриц

M = numpy.array(matrix) # создаем массив numpy

myA = numpy.delete(M, M.shape[1] - 1, axis=1) # получим матрицу A, путем удаления последней колонки

myB = M[:, [-1]].flatten() # получаем массив со значениями, путем выбора последнего столбца

**print**("Исходная система:")

FancyPrint(myA, myB, None)

slv = numpy.linalg.solve(myA, myB) # решаем СЛАУ

**print**("Решаем:")

**print**(slv)

numpy.savetxt(f, numpy.array([slv]), delimiter=',') #сохраняем строку с решением в файл

f.close() # закрываем файл

Результат решения линейного уравнения записывает в новый созданный файл.

Результат данного кода можно увидеть ниже на рисунке 36

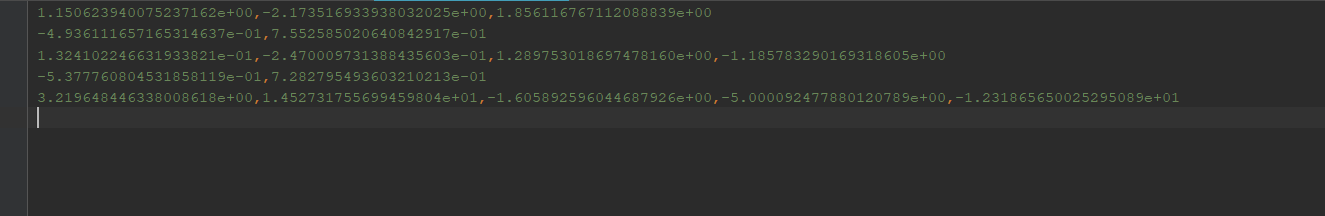


Рис.36 Результат решения методом Гаусса

**ЗАКЛЮЧЕНИЕ**

В проделанной работе я разобрал три метода сортировки последовательностей и сделал вывод, что самым эффективным является метод «Select», так как для упорядочивания случайной последовательности он делает наименьшее количество перестановок. Что можно сказать про библиотеку Numpy, эта библиотека является очень полезной, она отлично подойдет для работы с математическими функциями и задачами. Данная библиотека существенно упростит жизнь людям.

**БИБЛИОГРАФИЧЕСКИЙ СПИСОК**

1. Интернет статья «Самоучитель по Python

<https://pythonworld.ru/samouchitel-python>

2. Интернет статья «Numpy»

<https://pythonworld.ru/numpy>

3. Краткое руководство по NumPy: “Учусь NumPy”.

4. Автоматизация рутинных задач с помощью Python.Эл Свейгарт.