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 – 380305.2022.08. ПЗ КР

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

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

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

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Гайфулина К.Р./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

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

Работа защищена с оценкой

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

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

Челябинск 2019

АННОТАЦИЯ

Гайфулина К.Р.

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-142, 2019

В курсовой работе разобраны и описаны основные возможности языка программирования Python, подключение библиотеки Numpy, решена 31 задача. Также разобраны алгоритмы методов сортировки, их особенности, принципы работы и решение СЛАУ методом Гаусса.

ОГЛАВЛЕНИЕ

[**ВВЕДЕНИЕ** 4](#_Toc11621898)

[**Глава 1. Алгоритмы сортировки** 5](#_Toc11621899)

[**1.1 Сортировка выбором (Selection sort)** 5](#_Toc11621900)

[**1.2 Сортировка вставками (Сортировка включениями) (Insertion sort)** 5](#_Toc11621901)

[**1.3 Сортировка “Методом Пузырька” (Обменная сортировка) (Bubble sort)** 6](#_Toc11621902)

[**1.4 Анализ алгоритмов сортировки** 7](#_Toc11621903)

[**Глава 2. Библиотека NumPy.** 11](#_Toc11621904)

[**2.1 Что такое NumPy?** 11](#_Toc11621905)

[**2.2 NumPy на практике (31 задача)** 11](#_Toc11621906)

[**Глава 3. Решение СЛАУ методом Гаусса на Python с помощью NumPy (5 персональных задач)** 43](#_Toc11621907)

[**ЗАКЛЮЧЕНИЕ** 47](#_Toc11621908)

[**БИБЛИОГРАФИЧЕСКИЙ СПИСОК** 48](#_Toc11621909)

# **ВВЕДЕНИЕ**

В связи с наблюдаемым в настоящее время стремительным развитием персональной вычислительной техники, происходит постепенное изменение требований, предъявляемых к языкам программирования. Все большую роль начинают играть интерпретируемые языки, поскольку возрастающая мощь персональных компьютеров начинает обеспечивать достаточную скорость выполнения интерпретируемых программ.

Python является интерпретируемым, изначально объектно-ориентированным языком программирования. Он чрезвычайно прост и содержит небольшое число ключевых слов, вместе с тем очень гибок и выразителен. Это язык более высокого уровня, что достигается, в основном, за счет встроенных высокоуровневых структур данных.

Отличительные характеристики языка:

− минималистичный язык;

− краткий код;

− существует большое количество реализаций;

− очень хорошая поддержка математических вычислений;

− используется для обработки естественных языков.

Язык программирования Python — это мощный инструмент для создания программ самого разнообразного назначения, доступный даже для новичков. С его помощью можно решать задачи различных типов.

Курсовая работа состоит из 3 глав:

1. В первой главе раскрывается тема алгоритмов методов сортировки для одномерных массивов;
2. Во второй главе раскрывается тема библиотеки Numpy и применение её основных возможностей на практике;
3. В третьей главе раскрывается тема линейной алгебры, решения СЛАУ методом Гаусса при помощи Python и библиотеки Numpy.

Цель работы: выяснение эффективного алгоритма сортировки из трех представленных (Сортировка включением, Обменная сортировка, Сортировка выбором); обзор, разбор и понимание библиотеки Numpy с помощью решения задач; решение СЛАУ при помощи языка программирования Python.

# **Глава 1. 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.

## **1.1 Сортировка выбором (Selection sort)**

Этот алгоритм сегментирует список на две части: отсортированную и неотсортированную. Наименьший элемент удаляется из второго списка и добавляется в первый.

На практике не нужно создавать новый список для отсортированных элементов. В качестве него используется крайняя левая часть списка. Находится наименьший элемент и меняется с первым местами.

Теперь, когда нам известно, что первый элемент списка отсортирован, находим наименьший элемент из оставшихся и меняем местами со вторым. Повторяем это до тех пор, пока не останется последний элемент в списке.

Код:

def select\_sort(array):

for i in range(len(array)):

indxMin = i

for j in range(i+1, len(array)):

if array[j] < array[indxMin]:

indxMin = j

tmp = array[indxMin]

array[indxMin] = array[i]

array[i] = tmp

return a

## **1.2 Сортировка вставками (Сортировка включениями) (Insertion sort)**

Как и сортировка выборкой, этот алгоритм сегментирует список на две части: отсортированную и неотсортированную. Алгоритм перебирает второй сегмент и вставляет текущий элемент в правильную позицию первого сегмента.

Предполагается, что первый элемент списка отсортирован. Переходим к следующему элементу, обозначим его х. Если х больше первого, оставляем его на своём месте. Если он меньше, копируем его на вторую позицию, а х устанавливаем, как первый элемент.

Переходя к другим элементам несортированного сегмента, перемещаем более крупные элементы в отсортированном сегменте вверх по списку, пока не встретим элемент меньше x или не дойдём до конца списка. В первом случае x помещается на правильную позицию.

Код:

def insert\_sort(array):

for i in range(len(array)):

v = array[i]

j = i

while (array[j-1] > v) and (j > 0):

array[j] = array[j-1]

j = j - 1

array[j] = v

print array

return array

## **1.3 Сортировка “Методом Пузырька” (Обменная сортировка) (Bubble sort)**

Этот простой алгоритм выполняет итерации по списку, сравнивая элементы попарно и меняя их местами, пока более крупные элементы не «всплывут» в начало списка, а более мелкие не останутся на «дне».

Сначала сравниваются первые два элемента списка. Если первый элемент больше, они меняются местами. Если они уже в нужном порядке, оставляем их как есть. Затем переходим к следующей паре элементов, сравниваем их значения и меняем местами при необходимости. Этот процесс продолжается до последней пары элементов в списке.

При достижении конца списка процесс повторяется заново для каждого элемента. Это крайне неэффективно, если в массиве нужно сделать, например, только один обмен. Алгоритм повторяется n² раз, даже если список уже отсортирован.

Код:

def bubble\_sort(array):

a = array

for i in range(len(a),0,-1):

for j in range(1, i):

if a[j-1] > a[j]:

tmp = a[j-1]

a[j-1] = a[j]

a[j] = tmp

print a

return a

## **1.4 Анализ алгоритмов сортировки**

В этой части представлен код программы, определяющей эффективность трех алгоритмов методов сортировки (вставками (insertion), обменная (bubble), выбором (selection)).

Код:

import select\_sort

import bubble\_sort

import insert\_sort

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)  
myfile = open("sort\_methods.txt", "w")   
print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Исходный массив")  
print(select\_arr)  
count = [0, 0]  
count = select\_sort.select(select\_arr, DIM)  
print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив")  
print(select\_arr)  
CTotal[0] = count[0]  
MTotal[0] = count[1]  
count = [0, 0]  
count = insert\_sort.insert(insert\_arr, DIM)  
CTotal[1] = count[0]  
MTotal[1] = count[1]  
count = [0, 0]  
count = bubble\_sort.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\_sort.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\_sort.insert(insert\_arr, DIM)  
CTotal[1] = count[0]  
MTotal[1] = count[1]  
count = [0, 0]  
count = bubble\_sort.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\_sort.select(select\_arr, DIM)  
 CTotal[0] += count[0]  
 MTotal[0] += count[1]  
 count = [0, 0]  
 count = insert\_sort.insert(insert\_arr, DIM)  
 CTotal[1] += count[0]  
 MTotal[1] += count[1]  
 count = [0, 0]  
 count = bubble\_sort.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)

myfile.write("\nСЛУЧАЙНАЯ РЕАЛИЗАЦИЯ:\n")

wr\_str = "Проведено экспериментов: " + str(NUM) + "\n"

myfile.write(wr\_str)

wr\_str = "Размер массива: " + str(DIM) + "\n"

myfile.write(wr\_str)

wr\_str = "Сравнений: " + str(CTotal[0]/NUM) + " " + str(CTotal[1]/NUM) + " " + str(CTotal[2]/NUM) + "\n"

myfile.write(wr\_str)

wr\_str = "Перестановок: " + str(MTotal[0]/NUM) + " " + str(MTotal[1]/NUM) + " " + str(MTotal[2]/NUM) + "\n"

myfile.write(wr\_str)

myfile.close()

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

Таким образом, можно сделать вывод о том, что самым эффективным методом сортировки для массива является метод выбора (selection).

# **Глава 2. Библиотека NumPy.**

## **2.1 Что такое NumPy?**

NumPy — библиотека с открытым исходным кодом для языка программирования Python.

Возможности:

* поддержка многомерных массивов (включая матрицы);
* поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

Математические алгоритмы, реализованные на интерпретируемых языках (например, Python), часто работают гораздо медленнее тех же алгоритмов, реализованных на компилируемых языках (например, Фортран, Си, Java). Библиотека NumPy предоставляет реализации вычислительных алгоритмов (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.

NumPy можно рассматривать как свободную альтернативу MATLAB. Язык программирования MATLAB внешне напоминает NumPy: оба они интерпретируемые, оба позволяют выполнять операции над массивами (матрицами), а не над скалярами. Преимущество MATLAB в наличии большого количества пакетов («тулбоксов»), например, Simulink. Для NumPy тоже существуют подобные «пакеты», например, библиотека SciPy предоставляет больше MATLAB-подобной функциональности, библиотека Matplotlib позволяет создавать графики в стиле MATLAB. И MATLAB, и NumPy для решения основных задач линейной алгебры используют код, основанный на коде библиотеки LAPACK.

## **2.2 NumPy на практике (31 задача)**

Для начала понимания, как работать с матрицами, решим следующие задачи, количеством 31:

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.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

sum = A.sum(axis=0)

i = sum.argmax(axis=0)

max = A.max(axis=0)

max = max[i]

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

Результат:

Матрица:

[[92 29 87 61 96]

[60 1 51 67 39]

[56 0 28 65 7]]

Наибольшее значение: 92

2. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти наибольшее значение среди средних значений для каждой строки матрицы.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

Average = A.mean(axis=1)

i = Average.argmax(axis=0)

max = Average.max(axis=0)

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

Результат:

Матрица:

[[47 79 55 76 85]

[90 39 16 35 56]

[35 93 62 57 19]]

Наибольшее среднее значение: 68.4

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.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

sum = A.sum(axis=0)

i = sum.argmin(axis=0)

min = A.min(axis=0)

min = min[i]

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

Результат:

Матрица:

[[64 88 42 47 53]

[75 27 35 71 52]

[35 99 29 50 74]]

Наименьшее значение: 29

4. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти наименьшее значение среди средних значений для каждой строки матрицы.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

Average = A.mean(axis=1)

i = Average.argmin(axis=0)

min = Average.min(axis=0)

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

Результат:

Матрица:

[[57 70 22 65 71]

[43 56 66 89 51]

[62 16 66 11 14]]

Наименьшее среднее значение: 33.8

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.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

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

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

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

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

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[13 87 69 20 39]

[23 71 15 1 91]

[38 85 62 72 20]]

Новая матрица:

[[ 13. 87. 69. 20. 39. 45.6 ]

[ 23. 71. 15. 1. 91. 40.2 ]

[ 38. 85. 62. 72. 20. 55.4 ]

[ 24.66666667 81. 48.66666667 31. 50. 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.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

Sum = A.sum()

B=np.sum(A)

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[76 33 80 70 94]

[99 21 82 28 22]

[84 64 45 90 12]]

Новая матрица:

[[ 76. 33. 80. 70. 94. ]

[ 99. 21. 82. 28. 22. ]

[ 84. 64. 45. 90. 12. ]

[ 0.28777778 0.13111111 0.23 0.20888889 0.14222222]]

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.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

Sum = A.sum()

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

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

X = []

for i in range(0, N):

n = Sum\_column[i] / Sum

X.append(n)

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[9 0 1 3 9]

[1 7 7 4 5]

[1 6 1 5 2]]

Сумма элементов всей матрицы: 61

Новая матрица:

[[ 9. 0. 1. 3. 9. 0.36065574]

[ 1. 7. 7. 4. 5. 0.39344262]

[ 1. 6. 1. 5. 2. 0.24590164]]

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.

Код:

import numpy as np

N = 5

M = 8

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

print("Матрица:\n" + str(A))

Sum = A.sum()

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

X = []

for i in range(0, N):

K = 0

for j in range(0, M):

if A[i,j] < 0:

K += 1

X.append(K)

Y = []

for i in range(0, M):

K = 0

for j in range(0, N):

if A[j,i] < 0:

K += 1

Y.append(K)

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

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

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[-97 93 76 10 88 31 -16 -17]

[-70 57 62 11 -4 41 -36 -22]

[-16 -77 -87 58 70 71 -36 -18]

[-61 85 -56 94 70 5 -36 38]

[-41 -99 8 -57 3 -63 10 91]]

Сумма элементов всей матрицы: 163

Новая матрица:

[[-97. 93. 76. 10. 88. 31. -16. -17. 3.]

[-70. 57. 62. 11. -4. 41. -36. -22. 4.]

[-16. -77. -87. 58. 70. 71. -36. -18. 5.]

[-61. 85. -56. 94. 70. 5. -36. 38. 3.]

[-41. -99. 8. -57. 3. -63. 10. 91. 4.]

[ 5. 2. 2. 1. 1. 1. 4. 3. 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.

Код:

import numpy as np

N = 5

M = 8

L = 2

K = 4

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

print("Матрица:\n" + str(A))

Ln = 0

Kn = 0

for i in A[:L].flat:

if i == 0:

Ln += 1

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

if i == 0:

Kn += 1

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

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

Результат:

Матрица:

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

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

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

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

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

Количество нулевых элементов в верхних 2 строках матрицы: 5

Количество нулевых элементов в левых 4 столбцах матрицы: 6

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.

Код:

import numpy as np

N = 5

M = 8

K = 2

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

print("Матрица:\n" + str(A))

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

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

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

A = A \* K\_arr

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[ 0 0 0 0 1 0 1 1]

[ 0 1 0 -1 0 1 0 0]

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

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

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

K-ый столбец:

[[ 0]

[ 1]

[-1]

[-1]

[-1]]

Новая матрица:

[[ 0 0 0 0 0 0 0 0]

[ 0 1 0 -1 0 1 0 0]

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

[ 0 1 -1 0 0 1 1 1]

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

11. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Просуммировать элементы каждой строки матрицы с соответствующими элементами L-й строки.

Код:

import numpy as np

N = 5

M = 7

L = 2

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

print("Матрица:\n" + str(A))

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

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

A = A + L\_arr

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

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

[ 0 -1 1 0 -1 0 0]

[ 1 0 1 1 -1 0 -1]

[ 0 -1 1 -1 0 1 0]

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

L строка:

[ 0 -1 1 0 -1 0 0]

Новая матрица:

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

[ 0 -2 2 0 -2 0 0]

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

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

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

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.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

Max = A.max(axis=1)

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

A = A / Max

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[ 1 0 0 0 0]

[ 1 -1 -1 1 1]

[ 0 1 1 0 -1]]

Новая матрица:

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

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

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

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.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

Max = A.max(axis=0)

A = A / Max

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[2 2 1 1 6]

[2 1 4 3 4]

[5 5 5 6 4]]

Новая матрица:

[[ 0.4 0.4 0.2 0.16666667 1. ]

[ 0.4 0.2 0.8 0.5 0.66666667]

[ 1. 1. 1. 1. 0.66666667]]

14. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Разделить элементы матрицы на элемент матрицы с наибольшим значением.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

Max = A.max()

print("Наибольшее значение:\n" + str(Max))

A = A / Max

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[9 1 8 8 8]

[8 6 0 4 9]

[7 5 6 0 2]]

Наибольшее значение:

9

Новая матрица:

[[ 1. 0.11111111 0.88888889 0.88888889 0.88888889]

[ 0.88888889 0.66666667 0. 0.44444444 1. ]

[ 0.77777778 0.55555556 0.66666667 0. 0.22222222]]

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.

Код:

import numpy as np

N = 4

M = 7

H = 3

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

print("Матрица:\n" + str(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))

Результат:

Матрица:

[[3 1 6 4 9 9 7]

[0 9 4 0 0 9 2]

[2 6 7 9 7 1 7]

[6 0 1 2 0 2 1]]

Столбцы, которые имеют хотя бы одно число H: [1]

Столбцы, которые не имеют это число: [2, 3, 4, 5, 6, 7]

16. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Исключить из матрицы строку с номером L. Сомкнуть строки матрицы.

Код:

import numpy as np

N = 4

M = 7

L = 2

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

print("Матрица:\n" + str(A))

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[5 3 8 0 2 2 8]

[6 8 3 9 7 6 8]

[6 9 7 2 2 3 3]

[2 3 0 0 4 2 8]]

Новая матрица:

[[5 3 8 0 2 2 8]

[6 9 7 2 2 3 3]

[2 3 0 0 4 2 8]]

17. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Добавить к матрице строку и вставить ее под номером L.

Код:

import numpy as np

N = 4

M = 7

L = 2

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

print("Матрица:\n" + str(A))

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

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[9 6 0 7 7 4 3]

[2 9 6 9 5 2 5]

[3 4 4 7 6 4 7]

[6 7 9 7 0 1 2]]

Строка для вставки: [-5 5 -6 -4 -1 -6 -8]

Новая матрица:

[[ 9 6 0 7 7 4 3]

[ 2 9 6 9 5 2 5]

[-5 5 -6 -4 -1 -6 -8]

[ 3 4 4 7 6 4 7]

[ 6 7 9 7 0 1 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]).

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

a = np.diagonal(A)

a\_sum = a.sum()

print("Главная диагональ: \n" + str(a) + "\nСумма главной диагонали = " + str(a\_sum))

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

b\_sum = b.sum()

print("Побочная диагональ: \n" + str(b) + "\nСумма побочной диагонали = " + str(b\_sum))

Результат:

Матрица:

[[1 0 4 3 8]

[5 0 3 7 7]

[9 8 6 2 6]]

Главная диагональ:

[1 0 6]

Сумма главной диагонали = 7

Побочная диагональ:

[8 7 6]

Сумма побочной диагонали = 21

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].

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

a = np.diagonal(A, 1)

a\_sum = a.sum()

print("Элементы, которые выше главной диагонали: \n" + str(a) + "\nСумма элементов, которые выше главной диагонали = " + str(a\_sum))

b = np.diagonal(A, -1)

b\_sum = b.sum()

print("Элементы, которые ниже главной диагонали: \n" + str(b) + "\nСумма элементов, которые ниже главной диагонали = " + str(a\_sum))

Результат:

Матрица:

[[2 7 1 8 4]

[3 9 5 6 1]

[0 7 5 2 3]]

Элементы, которые выше главной диагонали:

[7 5 2]

Сумма элементов, которые выше главной диагонали = 14

Элементы, которые ниже главной диагонали:

[3 7]

Сумма элементов, которые ниже главной диагонали = 14

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].

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

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

a\_prod = a.prod()

print("Элементы, которые выше побочной диагонали: \n" + str(a) + "\nСумма элементов, которые выше побочной диагонали = " + str(a\_prod))

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

b\_prod = b.prod()

print("Элементы, которые ниже побочной диагонали: \n" + str(b) + "\nСумма элементов, которые ниже побочной диагонали = " + str(b\_prod))

Результат:

Матрица:

[[2 4 4 5 0]

[0 2 6 0 1]

[0 8 5 5 5]]

Элементы, которые выше побочной диагонали:

[5 6 8]

Сумма элементов, которые выше побочной диагонали = 240

Элементы, которые ниже побочной диагонали:

[1 5]

Сумма элементов, которые ниже побочной диагонали = 5

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]).

Код:

import numpy as np

N = 5

M = 5

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

print("Матрица:\n" + str(A))

B = (A + A.T)/2

print("Новая матрица:\n" + str(B))

Результат:

Матрица:

[[-5 8 4 1 2]

[ 9 -6 -2 -5 -6]

[-4 0 6 -9 -8]

[ 4 2 8 3 2]

[-1 -1 -1 -2 -5]]

Новая матрица:

[[-5. 8.5 0. 2.5 0.5]

[ 8.5 -6. -1. -1.5 -3.5]

[ 0. -1. 6. -0.5 -4.5]

[ 2.5 -1.5 -0.5 3. 0. ]

[ 0.5 -3.5 -4.5 0. -5. ]]

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.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[0 1 0 1 0]

[0 0 0 1 0]

[0 1 1 0 1]]

Новая матрица:

[[0 1 0 1 0 0]

[0 0 0 1 0 1]

[0 1 1 0 1 1]]

23. Создать квадратную матрицу A, имеющую N строк и N столбцов со случайными элементами. Найти сумму элементов, расположенных выше главной диагонали, и произведение элементов, расположенных выше побочной диагонали (элементы главной диагонали имеют индексы от [0,0] до [N,N], а элементы побочной диагонали — от [N,0] до [0,N]).

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

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

a = A[iu]

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

print("Cумма элементов выше главной диагонали = " + str(a))

b = np.fliplr(A)[iu]

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

print("Произведение элементов выше побочной диагонали = " + str(b))

Результат:

Матрица:

[[4 8 6 6 1]

[5 1 5 1 1]

[4 9 3 0 3]]

Cумма элементов выше главной диагонали = 19

Произведение элементов выше побочной диагонали = 180

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.

Код:

import numpy as np

N = 4

M = 8

L = 3

K = 2

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

print("Матрица:\n" + str(A))

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

print("Матрица:\n" + str(A))

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

x\_sum = x.sum()

print("Сумма верхней левой части = " + str(x\_sum) + "\n" + str(x))

y = A[L: ,0 :K]

y\_sum = y.sum()

print("Сумма нижней левой части = " + str(y\_sum) + "\n" + str(y))

z = A[0:L ,K:]

z\_sum = z.sum()

print("Сумма верхней правой части = " + str(z\_sum) + "\n" + str(z))

a = A[L: ,K:]

a\_sum = a.sum()

print("Сумма нижней правой части = " + str(a\_sum) + "\n" + str(a))

Результат:

Матрица:

[[1 4 0 4 5 0 2 2]

[3 4 8 5 0 4 8 6]

[6 2 8 9 1 5 0 6]

[1 5 5 3 3 2 7 4]]

Матрица:

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

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

[-8 5 8 0 -6 -5 6 -5]

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

Сумма верхней левой части = 11

[[ 8 -1]

[ 9 -2]

[-8 5]]

Сумма нижней левой части = -8

[[ 0 -8]]

Сумма верхней правой части = 23

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

[ 8 6 9 0 6 -3]

[ 8 0 -6 -5 6 -5]]

Сумма нижней правой части = 24

[[ 8 4 1 9 -7 9]]

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.

Код:

import numpy as np

N = 3

M = 5

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

print("Матрица:\n" + str(A))

bool = A == 0

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

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

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[5 8 8 0 0]

[5 6 6 9 2]

[7 9 2 3 0]]

Новая матрица:

[[5 8 8 0 0 2]

[5 6 6 9 2 0]

[7 9 2 3 0 1]

[0 0 0 1 2 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.

Код:

import numpy as np

N = 4

M = 8

L = 3

K = 2

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

print("Матрица:\n" + str(A))

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

x\_sum = x.mean()

print("Среднее арифметическое верхней левой части = " + str(x\_sum) + "\n" + str(x))

y = A[L: ,0 :K]

y\_sum = y.mean()

print("Среднее арифметическое нижней левой части = " + str(y\_sum) + "\n" + str(y))

z = A[0:L ,K:]

z\_sum = z.mean()

print("Среднее арифметическое верхней правой части = " + str(z\_sum) + "\n" + str(z))

a = A[L: ,K:]

a\_sum = a.mean()

print("Среднее арифметическое нижней правой части = " + str(a\_sum) + "\n" + str(a))

Результат:

Матрица:

[[1 6 3 0 7 2 7 9]

[3 2 2 6 9 5 6 8]

[7 6 7 9 3 9 8 3]

[6 3 4 6 8 2 4 9]]

Среднее арифметическое верхней левой части = 4.16666666667

[[1 6]

[3 2]

[7 6]]

Среднее арифметическое нижней левой части = 4.5

[[6 3]]

Среднее арифметическое верхней правой части = 5.72222222222

[[3 0 7 2 7 9]

[2 6 9 5 6 8]

[7 9 3 9 8 3]]

Среднее арифметическое нижней правой части = 5.5

[[4 6 8 2 4 9]]

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.

Код:

import numpy as np

N = 4

M = 7

H = 2

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

print("Матрица:\n" + str(A))

bool = A == H

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

print("Строки, в которых есть значение: " + str(H))

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

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

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

Результат:

Матрица:

[[3 1 5 5 6 6 0]

[4 0 5 6 4 4 3]

[3 8 7 7 6 1 6]

[0 8 2 0 7 4 5]]

Строки, в которых есть значение: 2

[3]

Строки, в которых нет значения: 2

[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.

Код:

import numpy as np

N = 3

M = 5

K = 2

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

print("Матрица:\n" + str(A))

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

print("Матрица:\n" + str(A))

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[8 6 7 0 8]

[6 2 1 7 7]

[4 2 7 3 0]]

Матрица:

[[ 9 -7 0 6 3]

[-8 7 6 -4 -8]

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

K = 2

Новая матрица:

[[ 9 0 6 3]

[-8 6 -4 -8]

[-4 -1 -8 -2]]

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.

Код:

import numpy as np

import random

N= random.randint (1,5)

M= random.randint (1,5)

K=1

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

print("Матрица:\n" + str(A))

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

print("Столбец для вставки: " + str(B))

A = np.insert(A, K, B, axis=1)

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[-2 9 7]

[ 1 -8 -6]

[-1 -6 -4]

[ 7 -7 -1]

[-4 4 -6]]

Столбец для вставки: [-8 -9 -7 -3 -4]

Новая матрица:

[[-2 -8 9 7]

[ 1 -9 -8 -6]

[-1 -7 -6 -4]

[ 7 -3 -7 -1]

[-4 -4 4 -6]]

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.

Код:

import numpy as np

import random

N= random.randint (1,5)

M= random.randint (1,5)

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

print("Матрица:\n" + str(A))

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[-4 -6 -3]

[-7 1 3]

[-7 -9 9]

[ 6 3 1]]

Новая матрица:

[[ -4 -6 -3]

[ -7 1 3]

[ -7 -9 9]

[ 6 3 1]

[ 12 11 -10]]

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.

Код:

import numpy as np

import random

N = random.randint (1,5)

M = random.randint (1,5)

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

print("Матрица:\n" + str(A))

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

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

print("Новая матрица:\n" + str(A))

Результат:

Матрица:

[[ 0 0]

[-3 4]

[-2 -7]

[-4 -9]]

Новая матрица:

[[ 0 0 0]

[-3 4 -1]

[-2 -7 9]

[-4 -9 13]]

Таким образом, можно сделать вывод, что библиотека NumPy хорошая альтернатива MatLab и отлично подходит для работы с математическими и числовыми операциями, которые используются с большими матрицами и массивами.

# **Глава 3. Решение СЛАУ методом Гаусса на Python с помощью NumPy (5 персональных задач)**

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.

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.

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.

Вариант 8:

Задача №1

| 1.57 1.40 1.71 | \* | X1| = | 0.98|

| 1.71 1.46 1.12 | \* | X2| = | 0.54|

| 1.43 1.98 1.85 | \* | X3| = | 0.14|

Задача №2

| 1.77 1.13 1.17 1.69 1.12 | \* | X1| = | 0.02|

| 1.50 1.19 1.36 1.63 1.33 | \* | X2| = | 0.06|

| 1.01 1.62 1.83 1.84 1.70 | \* | X3| = | 0.82|

| 1.15 1.85 1.48 1.61 1.95 | \* | X4| = | 0.08|

| 1.27 1.26 1.12 1.59 1.39 | \* | X5| = | 0.15|

Задача №3

| 1.62 1.15 1.65 1.62 | \* | X1| = | 0.21|

| 1.78 1.98 1.17 1.81 | \* | X2| = | 0.60|

| 1.20 1.23 1.67 1.14 | \* | X3| = | 0.40|

| 1.97 1.10 1.03 1.04 | \* | X4| = | 0.53|

Задача №4

| 1.61 1.59 1.18 1.85 1.96 | \* | X1| = | 0.92|

| 1.04 1.79 1.04 1.49 1.14 | \* | X2| = | 0.58|

| 1.34 1.68 1.90 1.22 1.30 | \* | X3| = | 0.21|

| 1.31 1.84 1.26 1.39 1.65 | \* | X4| = | 0.33|

| 1.93 1.60 1.65 1.13 1.97 | \* | X5| = | 0.86|

Задача №5

| 1.36 1.90 1.96 1.03 1.72 | \* | X1| = | 0.25|

| 1.94 1.42 1.72 1.79 1.45 | \* | X2| = | 0.47|

| 1.64 1.99 1.29 1.03 1.83 | \* | X3| = | 0.85|

| 1.24 1.74 1.27 1.24 1.46 | \* | X4| = | 0.98|

| 1.78 1.54 1.82 1.88 1.56 | \* | X5| = | 0.62|

Код:

import numpy

def FancyPrint(A, B, selected):

for row in range(len(B)):

print("(", end='')

for col in range(len(A[row])):

print("\t{1:10.2f}{0}".format(" " if (selected is None or selected != (row, col)) else "\*", A[row][col]), end='')

print("\t) \* (\tX{0}) = (\t{1:10.2f})".format(row + 1,

B[row]))

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

matrix\_list = []

matrix = []

for row in data:

first\_col = row[0]

if numpy.isnan(first\_col):

matrix\_list.append(matrix)

matrix = []

continue

mask = ~numpy.isnan(row)

matrix.append(row[mask])

matrix\_list.append(matrix)

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

f.truncate()

for matrix in matrix\_list:

M = numpy.array(matrix)

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

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

print("Исходная система:")

FancyPrint(myA, myB, None)

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

print("Решение:")

print(slv)

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

f.close()

Результат:

Решение №1:

[ 0.97676603 -1.41375852 0.83376564 ]

Решение №2:

[ -1.08157869 0.63662119 0.30422475 1.35085035 -1.27131046 ]

Решение №3:

[ 0.21145671 0.55012210 0.01586995 -0.48850969 ]

Решение №4:

[ 2.13421893 -0.09723908 -0.82054340 0.47874326 -1.16271232 ]

Решение №5:

[ -0.36930632 0.94456048 -0.93205550 0.85390155 -0.05528961 ]

# **ЗАКЛЮЧЕНИЕ**

Исходя из всей информации можно сделать вывод, Python это язык программирования общего назначения, нацеленный в первую очередь на повышение продуктивности самого программиста, нежели кода, который он пишет. Говоря простым человеческим языком, на Python можно написать практически что угодно (веб-/настольные приложения, игры, скрипты по автоматизации, комплексные системы расчёта, системы управления жизнеобеспечением и многое многое другое) 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. А с точки зрения бизнеса это влечёт за собой сокращение расходов и увеличение производительности труда сотрудников. Он достаточно простой и читабельный, а при всей своей простоте Python позволяет реализовать практически любую идею.

# **БИБЛИОГРАФИЧЕСКИЙ СПИСОК**

1. Вирт, Н. Алгоритмы и структуры данных / Н. Вирт. − М: Изд-во «Мир», 1989. − 360 с.;
2. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.
3. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с.
4. https://ru.wikipedia.org
5. https://www.python.org
6. https://habr.com/ru