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 – 380305.2022.14. ПЗ КР

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

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

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

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Свистунов-Рейвах Ф.Е./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

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

Работа защищена с оценкой

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

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

Челябинск 2019

Аннотация

Свистунов-Рейвах Ф.Е.

Программирование на языке Python (курс молодого бойца)

Челябинск: ЮУрГУ, ЭУ-142, 2019

В курсовой работе показаны основные возможности языка программирования Python.

Введение  
  
 Python – популярный язык программирования, используемый как для разработки самостоятельных программ, так и для создания прикладных сценариев в самых разных областях применения. Это мощный, переносимый, простой в использовании и свободно распространяемый язык. Программисты, работающие в самых разных областях, считают, что ориентация Python на эффективность разработки и высокое качество программного обеспечения дает ему стратегическое преимущество как в маленьких, так и в крупных проектах.  
Количеством числа пользователей Python является число, близкое 3 миллиону человек во всем мире (на 2019). Эта оценка основана на различных статистических показателях, таких как количество загрузок и результаты опросов разработчиков. Дать более точную оценку достаточно сложно, потому что Python является открытым программным обеспечением – для его использования не требуется проходить лицензирование.   
Такие известные компании, как Google и Intel, Cisco и Hewlett-Packard, используют язык Python, выбрав его за гибкость, простоту использования и обеспечиваемую им высокую скорость разработки. Он позволяет создавать эффективные и надежные проекты, которые легко интегрируются с программами и инструментами, написанными на других языках.  
Универсальная природа языка обеспечивает возможность егоприменения в самых разных областях. Фактически с определенной долей уверенности можно утверждать, что Python так или иначе используется практически каждой достаточно крупной организацией, занимающейся разработкой программного обеспечения, – как для решения краткосрочных тактических задач, так и для разработки долгосрочных стратегических проектов.

Раздел 1 Методы сортировки одномерных массивов

Сортировка выбором (Select sort)

Для того, чтобы отсортировать массив в порядке возрастания, следует на каждой итерации найти элемент с наибольшим значением. С ним нужно поменять местами последний элемент. Следующий элемент с наибольшим значением становится уже на предпоследнее место. Так должно происходить, пока элементы, находящиеся на первых местах в массивe, не окажутся в надлежащем порядке.

Пузырьковая сортировка (Bubble sort)

При пузырьковой сортировке сравниваются соседние элементы и меняются местами, если следующий элемент меньше предыдущего. Требуется несколько проходов по данным. Во время первого прохода сраваются первые два элемента в массиве. Если они не в порядке, они меняются местами и затем сравнивается элементы в следующей паре. При том же условии они так же меняются местами. Таким образом сортировка происходит в каждом цикле пока не будет достигнут конец массива.

Сортировка вставками (Insertion sort)

При сортировке вставками массив разбивается на две области: упорядоченную и и неупорядоченную. Изначально весь массив является неупорядоченной областью. При первом проходе первый элемент из неупорядоченной области изымается и помещается в правильном положении в упорядоченной области.

На каждом проходе размер упорядоченной области возрастает на 1, а размер неупорядоченной области сокращается на 1.

Основной цикл работает в интервале от 1 до N-1. На j-й итерации элемент [i] вставлен в правильное положение в упорядоченной области. Это сделано путем сдвига всех элементов упорядоченной области, которые больше, чем [i], на одну позицию вправо. [i] вставляется в интервал между теми элементами, которые меньше [i], и теми, которые больше [i].

Функции всех алгоритмов:

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

Теперь необходимо убедиться какой из этих алгоритмов самый эффективный. Для был написан данный код:

import random  
import sort\_functions  
  
DIM = 40  
bubble\_arr = []  
insert\_arr = []  
select\_arr = []  
  
CTotal = [0, 0, 0]  
MTotal = [0, 0, 0]  
  
for i in range(1, DIM+1):  
 select\_arr.append(i)  
 bubble\_arr.append(i)  
 insert\_arr.append(i)  
  
myfile = open("sort\_methods.txt", "w") # Все результаты запишем в файл  
  
print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Исходный массив")  
print(select\_arr)  
  
#метод пузырька Select  
count = [0, 0]  
count = sort\_functions.select(select\_arr, DIM)  
print("\nУПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив")  
print(select\_arr)  
CTotal[0] = count[0]  
MTotal[0] = count[1]  
  
#метод вставки Insert  
count = [0, 0]  
count = sort\_functions.insert(insert\_arr, DIM)  
CTotal[1] = count[0]  
MTotal[1] = count[1]  
  
#метод пузырька Bubble  
count = [0, 0]  
count = sort\_functions.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)  
  
#метод пузырька Select  
count = [0, 0]  
count = sort\_functions.select(select\_arr, DIM)  
print("\nОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив")  
print(select\_arr)  
CTotal[0] = count[0]  
MTotal[0] = count[1]  
  
#метод вставки Insert  
count = [0, 0]  
count = sort\_functions.insert(insert\_arr, DIM)  
CTotal[1] = count[0]  
MTotal[1] = count[1]  
  
#метод пузырька Bubble  
count = [0, 0]  
count = sort\_functions.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])  
  
# СЛУЧАЙНАЯ РЕАЛИЗАЦИЯ  
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])  
  
 #метод пузырька Select  
 count = [0, 0]  
 count = sort\_functions.select(select\_arr, DIM)  
 CTotal[0] += count[0]  
 MTotal[0] += count[1]  
  
 #метод вставки Insert  
 count = [0, 0]  
 count = sort\_functions.insert(insert\_arr, DIM)  
 CTotal[1] += count[0]  
 MTotal[1] += count[1]  
  
 #метод пузырька Bubble  
 count = [0, 0]  
 count = sort\_functions.bubble(bubble\_arr, DIM)  
 CTotal[2] += count[0]  
 MTotal[2] += count[1]  
  
print("\nСЛУЧАЙНАЯ РЕАЛИЗАЦИЯ:")  
print("Проведено экспериментов: ", NUM)  
print("Размер массива: ", DIM)  
print("Сравнений: ", CTotal[0]/NUM, " ", CTotal[1]/NUM, " ", CTotal[2]/NUM)  
print("Перестановок: ", MTotal[0]/NUM, " ", MTotal[1]/NUM, " ", MTotal[2]/NUM)

Результат

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Исходный массив

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

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив

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

УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ:

Размер массива: 40

Сравнений: 780 780 780

Перестановок: 0 0 0

ОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Исходный массив

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

ОБРАТНО УПОРЯДОЧЕННАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ: Результирующий массив

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

Размер массива: 40

Сравнений: 780 780 780

Перестановок: 20 780 780

Данный результат показал количество перемещений и сравнений для каждого метода. Отсюда, можно сделать вывод, что самый эффективный является сортировка выбором (Select sort).

Раздел 2 Numpy. Работа с многомерными массивами

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.

Назначение:

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.

Работа с многомерными массивами при помощи пакета Numpy:

ВСЕ КОДЫ РАБОТАЮТ ИСПРАВНО И ВЫПОЛНЯЮТ СВОЮ ФУНКЦИЮ

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 = 15  
M = 30  
  
A = np.random.randint(0,100, (N,M))  
  
m=0  
indm=0  
for i in range (M):  
 if np.sum(A[:, i])>m:  
 m = np.sum(A[:, i])  
 indm=i  
stmax=0  
  
for b in A [: , indm]:  
 if b> stmax:  
 stmax=b  
   
print("Максимальный элемент этого столбца-"+str(stmax))

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 = 15  
M = 30  
  
A = np.random.randint(0,100, (N,M))  
  
mx=0  
aver=0  
for i in range (N):  
 if np.mean (A[i , :])> mx:  
 mx= np.mean(A[i , :])  
 aver=i  
  
print("Среднее значение этой строки-"+str(mx)+ ","+ "номер этой строки-"+ str(aver+1))

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 = 15  
M = 30  
  
A = np.random.randint(0,100, (N,M))  
  
mx=0  
indmx=0  
for i in range (M):  
 if np.sum(A[:, i])>mx:  
 mx = np.sum(A[:, i])  
 indmx=i  
print("Столбец с максимальной суммой элементов-"+ str(A[: , indmx]))  
print("Сумма этого столбца-"+str(mx)+ ","+ "номер этого столбца-"+ str(indmx+1))  
stmin=10  
for b in A [: , indmx]:  
 if b< stmin:  
 stmin=b  
print("Минимальный элемент этого столбца-"+str(stmin))

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 = 15  
M = 30  
  
A = np.random.randint(0,100, (N,M))  
  
print(A)  
  
mx=10  
aver=0  
for i in range (N):  
 if np.mean (A[i , :])< mx:  
 mx= np.mean(A[i , :])  
 aver=i  
  
print("Строка с минимальным средним значением элементов-"+ str(A[aver , :]))  
print("Среднее значение этой строки-"+str(mx)+ ","+ "номер этой строки-"+ str(aver+1))

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 = 15  
M = 30  
  
A = np.random.randint(0,100, (N,M))  
  
M\_mean = A.mean(axis=0)  
N\_mean = A.mean(axis=1)  
N\_mean = np.append(N\_mean, None)  
  
A = np.vstack((A, M\_mean))  
A = np.hstack((A, N\_mean.reshape(-1,1)))  
  
print(A)

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 = 15  
M = 30  
  
A = np.random.randint(-10,10, (N,M))  
  
B=np.sum(A)  
  
M\_sum = np.sum(A, axis=0)/np.sum(A)  
print("Доля элементов каждого столбца-"+str(np.sum(A, axis=0)/np.sum(A)))  
  
A = np.vstack((A,M\_sum))  
  
print(A)

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 = 15  
M = 30  
  
A = np.random.randint(-10,10, (N,M))  
  
  
N\_sum = np.sum(A, axis=1)/np.sum(A)  
print("Доля элементов каждой строки-"+str(np.sum(A, axis=1)/np.sum(A)))  
  
A = np.hstack((A, N\_sum.reshape(-1,1)))  
  
print(A)

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 = 15  
M = 30  
  
A = np.random.randint(-10,10, (N,M))  
  
M\_sum = (A < 0).sum(axis=0)  
N\_sum = (A < 0).sum(axis=1)  
  
N\_sum = np.append(N\_sum, None)  
  
A = np.vstack((A, M\_sum))  
A = np.hstack((A, N\_sum.reshape(-1,1)))  
  
print(A)

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 = 15  
M = 30  
K = 10  
L = 12  
  
A = np.random.randint(-10,10, (N,M))  
  
  
New\_A = A[:L, :K]  
  
print(A)  
  
print(np.sum(New\_A == 0))

10. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Перемножить элементы каждого столбца матрицы с соответствующими элементами K-го столбца.

import numpy as np  
  
N = 15  
M = 30  
K = 10  
  
A = np.random.randint(-10,10, (N,M))  
K\_m = A[:, K].flat  
  
for i in range(M):  
 if i != K:  
 for k in range(N):  
 A[k, i] = A[k, i] \* K\_m[k]  
print(A)

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 = 15  
M = 30  
L = 12  
  
A = np.random.randint(-10,10, (N,M))  
  
L\_m = A[L, :].flat  
  
for i in range(N):  
 if i != L:  
 for k in range(M):  
 A[i, k] = A[i, k] + L\_m[k]  
print(A)

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 = 15  
M = 30  
K = 10  
  
A = np.random.randint(-10,10, (N,M))  
A = np.array(A, float)  
  
for i in range(N):  
 mx = np.max(A[i, :])  
 for k in range(M):  
 if A[i, k] != mx:  
 A[i, k] = round(A[i, k] / mx, 2)  
print(A)

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 = 15  
M = 30  
  
A = np.random.randint(-10,10, (N,M))  
  
  
col\_max = np.max(A, axis=0)  
A = A / col\_max  
  
print("Полученная матрица:\r\n {}".format(A))

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 = 15  
M = 30  
  
A = np.random.randint(-10,10, (N,M))  
  
max\_el = np.max(A)  
A = A / max\_el  
print("Полученная матрица:\r\n {}".format(A))

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 = 15  
M = 30  
H = 5  
  
A = np.random.randint(-10,10, (N,M))  
  
A\_bool = A == H  
row\_sum = np.sum(A\_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())

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 = 15  
M = 30  
L = 12  
  
A = np.random.randint(-10,10, (N,M))  
  
A = np.delete(A, L, axis=0)  
  
print("Полученная матрица:\r\n {}".format(A))

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 = 15  
M = 30  
K = 10  
L = 12  
  
A = np.random.randint(-10,10, (N,M))  
row = np.random.randint(low=-9, high=10, size=M)  
  
A = np.insert(A, L, row, axis=0)  
  
print("Полученная матрица:\r\n {}".format(A))

18. Создать прямоугольную матрицу A, имеющую N строк и M столбцов со случайными элементами. Найти сумму расположенных параллельно побочной диагонали (ближайшие к побочной). Элементы побочной диагонали имеют индексы от [N,0] до [0,N].

import numpy as np  
  
N = 15  
M = 30  
  
A = np.random.randint(-10,10, (N,M))  
  
diagonal\_main = np.diagonal(A)  
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(A[::-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))

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 = 15  
  
A = np.random.randint(-10,10, (N,N))  
  
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))

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 = 15  
  
A = np.random.randint(-10,10, (N,N))  
  
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))

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 = 15  
  
A = np.random.randint(-10,10, (N,N)).astype(float)  
print("Матрица:\r\n{}".format(A))  
  
diagonal\_elements = [np.diagonal(A, i) for i in [1, -1]]  
values = (diagonal\_elements[0] + diagonal\_elements[1])/2  
  
rng = np.arange(N-1)  
A[rng, rng+1] = values[rng]  
A[rng+1, rng] = values[rng]  
  
print("Полученная матрица:\r\n{}".format(A))

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 = 15  
M = 30  
  
A = np.random.randint(0,2, (N,M))  
  
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))

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

import numpy as np  
  
N = 15  
  
A = np.random.randint(-10,10, (N,N))  
  
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)  
print("Произведение элементов, расположенных выше побочной диагонали:")  
print(np.prod(elements))

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 = 15  
M = 30  
L = 12  
K = 10  
  
A = np.random.randint(-10,10, (N,M))  
  
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)  
print("Полученная матрица:\r\n {}".format(A))

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 = 15  
M = 30  
L = 12  
K = 10  
  
A = np.random.randint(-10,10, (N,M))  
  
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)  
  
print("Полученная матрица:\r\n {}".format(A))

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 = 15  
M = 30  
L = 12  
K = 10  
  
A = np.random.randint(-10,10, (N,M))  
  
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])))

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 = 15  
M = 30  
H = 2  
  
A = np.random.randint(-10,10, (N,M))  
  
A\_bool = A == H  
col\_sum = np.sum(A\_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())

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 = 15  
M = 30  
K = 10  
  
A = np.random.randint(-10,10, (N,M))  
  
A = np.delete(A, K, axis=1)  
  
print("Полученная матрица:\r\n {}".format(A))

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 = 15  
M = 30  
K = 10  
  
A = np.random.randint(-10,10, (N,M))  
A = np.random.randint(low=-9, high=10, size=(N, M))  
print("Матрица:\r\n{}".format(A))  
  
col = np.random.randint(low=-9, high=10, size=N)  
print("Столбец для вставки: {}".format(col))  
  
A = np.insert(A, K, col, axis=1)  
print("Полученная матрица:\r\n {}".format(A))

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 = 15  
M = 30  
  
A = np.random.randint(-10,10, (N,M))  
  
row = np.sum(A, axis=0) \* -1  
  
A = np.insert(A, N, row, axis=0)  
  
print("Полученная матрица:\r\n {}".format(A))

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 = 15  
M = 30  
  
A = np.random.randint(-10,10, (N,M))  
  
col = np.sum(A, axis=1) \* -1  
  
A = np.insert(A, M, col, axis=1)  
  
print("Полученная матрица:\r\n {}".format(A))

Вывод:

Благодаря библиотеке Numpy можно производить огромное количество математических действий, при это не загромождая программный код.

Раздел 3 СЛАУ методом Гаусса

Перед нами поставлена задача: Реализовать решение СЛАУ методом Гаусса при помощи языка программирования Python.

Для это я создал csv-файл, в котором храниться данные системы линейных алгебраических уравнений:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1,48 | 1,72 | 1,75 |  | 0,3 |  |  |  |
| 1,45 | 1,98 | 1,74 |  | 0,45 |  |  |  |
| 1,49 | 1,97 | 1,48 |  | 0,03 |  |  |  |
|  |  |  |  |  |  |  |  |
| 1,27 | 1,82 | 1,92 | 1,17 | 1,07 | 1,74 |  | 0,79 |
| 1,63 | 1,45 | 1,97 | 1,72 | 1,8 | 1,39 |  | 0,19 |
| 1,21 | 1,36 | 1,64 | 1,95 | 1,58 | 1,65 |  | 0,98 |
| 1,62 | 1,28 | 1,28 | 1,4 | 1,13 | 1,75 |  | 0,72 |
| 1,22 | 1,47 | 1,42 | 1,54 | 1,73 | 1,8 |  | 0,39 |
| 1,62 | 1,67 | 1,03 | 1,75 | 1,15 | 1,66 |  | 0,39 |
|  |  |  |  |  |  |  |  |
| 1,11 | 1,17 | 1,23 | 1,02 |  | 0,75 |  |  |
| 1,28 | 1,48 | 1,22 | 1,84 |  | 0,99 |  |  |
| 1,79 | 1,58 | 1,81 | 1,5 |  | 0,12 |  |  |
| 1,93 | 1,12 | 1,51 | 1,95 |  | 0,95 |  |  |
|  |  |  |  |  |  |  |  |
| 1,86 | 1,89 | 1,93 | 1,16 | 1,91 | 1,52 |  | 0,05 |
| 1,02 | 1,13 | 1,33 | 1,24 | 1,43 | 1,3 |  | 0,97 |
| 1,68 | 1,8 | 1,67 | 1,35 | 1,44 | 1,14 |  | 0,04 |
| 1,26 | 1,69 | 1,45 | 1,91 | 1,02 | 1,6 |  | 0,9 |
| 1,47 | 1,99 | 1,89 | 1,28 | 1,6 | 1,55 |  | 0,82 |
| 1,19 | 1,37 | 1,01 | 1,51 | 2 | 1,61 |  | 0,69 |
|  |  |  |  |  |  |  |  |
| 1,6 | 1,79 | 1,76 | 1,99 | 1,56 | 1,23 |  | 0,62 |
| 1,08 | 1,64 | 1,06 | 1,13 | 1,61 | 1,52 |  | 0,17 |
| 1,41 | 1,86 | 1,93 | 1,72 | 1,5 | 1,03 |  | 0,11 |
| 1,81 | 1,66 | 1,09 | 1,02 | 1,57 | 1,63 |  | 0,98 |
| 1,72 | 1,33 | 1,65 | 1,22 | 1,91 | 1,32 |  | 0,04 |
| 1,89 | 1,91 | 1,88 | 1,27 | 1,46 | 1,5 |  | 0,91 |
|  |  |  |  |  |  |  |  |

Далее программный код, который представлен ниже, будет считывать данный файл, определять А-часть и В-часть, и решит систему.

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('./SLAY.csv', delimiter=';')  
print(data)  
  
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("Решаем:")  
 print(slv)  
 numpy.savetxt(f, numpy.array([slv]), delimiter=',')  
f.close()

Результат

[[1.48 1.72 1.75 nan 0.3 nan nan nan]

[1.45 1.98 1.74 nan 0.45 nan nan nan]

[1.49 1.97 1.48 nan 0.03 nan nan nan]

[ nan nan nan nan nan nan nan nan]

[1.27 1.82 1.92 1.17 1.07 1.74 nan 0.79]

[1.63 1.45 1.97 1.72 1.8 1.39 nan 0.19]

[1.21 1.36 1.64 1.95 1.58 1.65 nan 0.98]

[1.62 1.28 1.28 1.4 1.13 1.75 nan 0.72]

[1.22 1.47 1.42 1.54 1.73 1.8 nan 0.39]

[1.62 1.67 1.03 1.75 1.15 1.66 nan 0.39]

[ nan nan nan nan nan nan nan nan]

[1.11 1.17 1.23 1.02 nan 0.75 nan nan]

[1.28 1.48 1.22 1.84 nan 0.99 nan nan]

[1.79 1.58 1.81 1.5 nan 0.12 nan nan]

[1.93 1.12 1.51 1.95 nan 0.95 nan nan]

[ nan nan nan nan nan nan nan nan]

[1.86 1.89 1.93 1.16 1.91 1.52 nan 0.05]

[1.02 1.13 1.33 1.24 1.43 1.3 nan 0.97]

[1.68 1.8 1.67 1.35 1.44 1.14 nan 0.04]

[1.26 1.69 1.45 1.91 1.02 1.6 nan 0.9 ]

[1.47 1.99 1.89 1.28 1.6 1.55 nan 0.82]

[1.19 1.37 1.01 1.51 2. 1.61 nan 0.69]

[ nan nan nan nan nan nan nan nan]

[1.6 1.79 1.76 1.99 1.56 1.23 nan 0.62]

[1.08 1.64 1.06 1.13 1.61 1.52 nan 0.17]

[1.41 1.86 1.93 1.72 1.5 1.03 nan 0.11]

[1.81 1.66 1.09 1.02 1.57 1.63 nan 0.98]

[1.72 1.33 1.65 1.22 1.91 1.32 nan 0.04]

[1.89 1.91 1.88 1.27 1.46 1.5 nan 0.91]]

Исходная система:

( 1.48 1.72 1.75 ( 1.45 1.98 1.74 ( 1.49 1.97 1.48 ) \* ( X3) = ( 0.03)

Решаем:

[-1.83723432 0.41557871 1.31674938]

Исходная система:

( 1.27 1.82 1.92 1.17 1.07 1.74 ( 1.63 1.45 1.97 1.72 1.80 1.39 ( 1.21 1.36 1.64 1.95 1.58 1.65 ( 1.62 1.28 1.28 1.40 1.13 1.75 ( 1.22 1.47 1.42 1.54 1.73 1.80 ( 1.62 1.67 1.03 1.75 1.15 1.66 ) \* ( X6) = ( 0.39)

Решаем:

[-0.56088313 -0.82252079 0.70322804 0.81697716 -1.10816122 1.07987403]

Исходная система:

( 1.11 1.17 1.23 1.02 ( 1.28 1.48 1.22 1.84 ( 1.79 1.58 1.81 1.50 ( 1.93 1.12 1.51 1.95 ) \* ( X4) = ( 0.95)

Решаем:

[-16.71402739 -11.58505981 19.76502341 8.37854077]

Исходная система:

( 1.86 1.89 1.93 1.16 1.91 1.52 ( 1.02 1.13 1.33 1.24 1.43 1.30 ( 1.68 1.80 1.67 1.35 1.44 1.14 ( 1.26 1.69 1.45 1.91 1.02 1.60 ( 1.47 1.99 1.89 1.28 1.60 1.55 ( 1.19 1.37 1.01 1.51 2.00 1.61 ) \* ( X6) = ( 0.69)

Решаем:

[-2.23988062 0.0879665 1.19987959 0.7013543 0.47940202 0.00324 ]

Исходная система:

( 1.60 1.79 1.76 1.99 1.56 1.23 ( 1.08 1.64 1.06 1.13 1.61 1.52 ( 1.41 1.86 1.93 1.72 1.50 1.03 ( 1.81 1.66 1.09 1.02 1.57 1.63 ( 1.72 1.33 1.65 1.22 1.91 1.32 ( 1.89 1.91 1.88 1.27 1.46 1.50 ) \* ( X6) = ( 0.91)

Решаем:

[ 0.96203797 -0.18928318 -0.31965728 0.50633944 -1.4956813 1.06325227]

Вывод:

Благодаря Python и Numpy, получилось решить систему очень быстрее, чем решать в ручную. Также этот код можно читать и другие файлы, это позволяет ещё больше СЛАУ.

Заключение

Это курсовая работа была посвящена знакомству с Python, его возможностям и основам синтаксиса. Как и говорилось, изучение Python в качестве первого языка программирования не представляет особой сложности, так как у него очень простой синтаксис и большое количество русской литературы.

Python — стабильный и распространённый язык. Он используется во многих проектах и в различных качествах: как основной язык программирования или для создания расширений и интеграции приложений. На Python реализовано большое количество проектов, также он активно используется для создания прототипов будущих программ. Python используется во многих крупных компаниях: Dropbox, Google (например некоторые части Youtube и Youtube API написаны на Python), Facebook, Instagram.

Python с пакетами NumPy, SciPy и MatPlotLib активно используется как универсальная среда для научных расчётов в качестве замены распространённым специализированным коммерческим пакетам Matlab, IDL и другим. Библиотека Astropy — популярный инструмент для астрономических расчётов.

Сочетание простоты и лаконичности с возможностью использования сложных абстракций и мощных разнообразных инструментов делает Python удобным в качестве скриптового языка. Возможность его встраивания ограничивается объёмом интерпретатора, но в крупных системах это ограничение несущественно. В профессиональных программах трёхмерной графики, таких как Autodesk Maya, Blender, Houdini и Nuke, Python используется для расширения стандартных возможностей программ. В Microsoft Power BI Desktop Python, наряду со встроенными языками запросов и языком программирования R, может использоваться на этапе загрузки данных в ETL-процессах, расчётах и графической визуализации данных

Библиографический список

1. Коэльё Л. П., Ричерт В. Построение систем машинного обучения на языке Python. — Перевод с английского. — М.: ДМК Пресс, 2015. — с.
2. Маккинли У. Python и анализ данных. — Перевод с английского. — М.: ДМК Пресс, 2015. — 482 с.
3. С. Шапошникова. Основы программирования на Python. Учебник. Вводный курс. — версия 2. — 2011. — 44 с.
4. [https://ru.wikipedia.org](https://ru.wikipedia.org/)