Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №7

По дисциплине: “Языки программирования”

Тема: «Изучение NumPy. Сравнение производительности с классическими библиотеками Python»Вариант №1

**Выполнил**:

студент 2 курса группы ПО-7 Белев С.В.

**Проверила:**

Дряпко А.В.

Брест 2021

**Задание 1:** Подсчитать произведение ненулевых элементов на диагонали прямоугольной матрицы.

Пример: x = np.array([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]])

Ответ: 3.

**Код:**

import numpy as np

arr\_py = [[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]]

arr\_np = np.array(arr\_py)

def diag\_py(arr\_py):

product = 1

for i in range(len(arr\_py[0])):

elem = arr\_py[i][i]

if elem:

product \*= elem

return product

def diag\_np(arr\_np):

diag = np.diagonal(arr\_np)

return np.prod(diag[diag != 0])

print('Python:', diag\_py(arr\_py))

print('Time: ', end='')

%timeit diag\_py(arr\_py)

print('NumPy:', diag\_np(arr\_np))

print('Time: ', end='')

%timeit diag\_np(arr\_np)

**Результат:**

Python: 3

Time: 541 ns ± 5.15 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

NumPy: 3

Time: 8.42 µs ± 108 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

**Задание 2:** Дана матрица x и два вектора одинаковой длины i и j. Построить вектор np.array([X[i[0], j[0]], X[i[1], j[1]], . . . , X[i[N-1], j[N-1]]]).  
Пример:

x = [[9 4 2], [6 0 0], [9 9 3]]

i: [1 2 1]

j: [1 0 1]

Ответ: [0 9 0]

**Код:**

x\_py = [[9, 4, 2], [6, 0, 0], [9, 9, 3]]

i\_py = [1, 2, 1]

j\_py = [1, 0, 1]

x\_np = np.array(x\_py)

i\_np = np.array(i\_py)

j\_np = np.array(j\_py)

def new\_vector\_py(x\_py, i\_py, j\_py):

vector = []

for k in range(len(i\_py)):

vector.append(x\_py[i\_py[k]][j\_py[k]])

return vector

def new\_vector\_np(x\_np, i\_np, j\_np):

return x\_np[i\_np, j\_np]

print('Python:', new\_vector\_py(x\_py, i\_py, j\_py))

print('Time: ', end='')

%timeit new\_vector\_py(x\_py, i\_py, j\_py)

print('NumPy:', new\_vector\_np(x\_np, i\_np, j\_np))

print('Time: ', end='')

%timeit new\_vector\_np(x\_np, i\_np, j\_np)

**Результат:**

Python: [0, 9, 0]

Time: 841 ns ± 87.4 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

NumPy: [0 9 0]

Time: 3.14 µs ± 81.5 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

**Задание 3:** Даны два вектора x и y. Проверить, задают ли они одно и то же мультимножество.

Пример: x = np.array([1,2, 2, 4]), y = np.array([4, 2, 1, 2])

Ответ: True.

**Код:**

x\_py = [1, 2, 2, 4]

y\_py = [4, 2, 1, 2]

x\_np = np.array(x\_py)

y\_np = np.array(y\_py)

def equal\_py(x\_py, y\_py):

return sorted(x\_py) == sorted(y\_py)

def equal\_np(x\_np, y\_np):

return np.array\_equal(np.bincount(x\_np), np.bincount(y\_np))

print('Python:', equal\_py(x\_py, y\_py))

print('Time: ', end='')

%timeit equal\_py(x\_py, y\_py)

print('NumPy:', equal\_np(x\_np, y\_np))

print('Time: ', end='')

%timeit equal\_np(x\_np, y\_np)

**Результат:**

Python: True

Time: 476 ns ± 12.2 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

NumPy: True

Time: 6.81 µs ± 75.8 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

**Задание 4:** Найти максимальный элемент в векторе x среди элементов, перед которыми стоит нулевой.

Пример: x = np.array([6, 2, 0, 3, 0, 0, 5, 7, 0])

Ответ: 5.

**Код:**

x\_py = [6, 2, 0, 3, 0, 0, 5, 7, 0]

x\_np = np.array(x\_py)

def max\_py(x\_py):

max\_num = -1

for i in range(1, len(x\_py)):

if x\_py[i - 1] == 0 and x\_py[i] > max\_num:

max\_num = x\_py[i]

return max\_num

def max\_np(x\_np):

zeroes = x\_np == 0

return x\_np[1:][zeroes[:-1]].max()

print('Python:', max\_py(x\_py))

print('Time: ', end='')

%timeit max\_py(x\_py)

print('NumPy:', max\_np(x\_np))

print('Time: ', end='')

%timeit max\_np(x\_np)

**Результат:**

Python: 5

Time: 910 ns ± 22 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)

NumPy: 5

Time: 4.41 µs ± 46 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

**Задание 5:** Дан трёхмерный массив, содержащий изображение, размера (height, width, numChannels), а также вектор длины numChannels. Сложить каналы изображения с указанными весами, и вернуть результат в виде матрицы размера (height, width). Считать реальное изображение можно при помощи функции scipy.misc.imread (если изображение не в формате png, установите пакет pillow: conda install pillow).  
Преобразуйте цветное изображение в оттенки серого, использовав коэффициенты np.array([0.299, 0 .587, 0.114]).

**Код:**

import matplotlib.pyplot as plt

photo = plt.imread('task.png')

photo\_py = photo.tolist()

photo\_np = np.array(photo\_py)

rgb\_ration\_py = [0.299, 0.587, 0.114]

rgb\_ration\_np = np.array(rgb\_ration\_py)

def to\_gray\_py(photo\_py, rgb\_ration\_py):

res = []

for h in photo\_py:

new\_width = []

for w in h:

color = 0

for i in range(3):

color = w[i] \* rgb\_ration\_py[i]

new\_width.append(color)

res.append(new\_width)

return res

def to\_gray\_np(photo\_np, rgb\_ration\_np):

return np.dot(photo\_np[...,:3], rgb\_ration\_np)

arr = to\_gray\_py(photo\_py, rgb\_ration\_py)

plt.imshow(arr, cmap='gray')

plt.show()

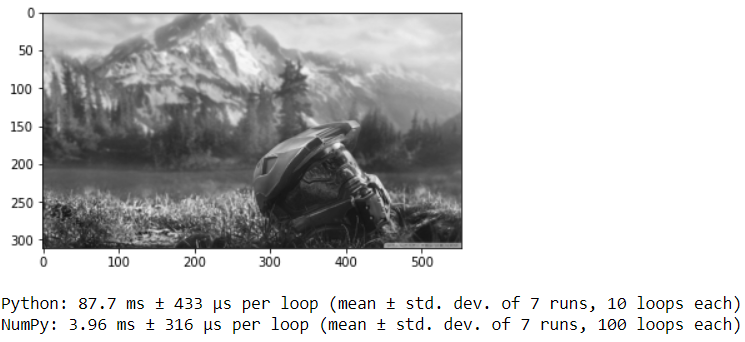
print('Python:', end=' ')

%timeit to\_gray\_py(photo\_py, rgb\_ration\_py)

print('NumPy:', end=' ')

%timeit to\_gray\_np(photo\_np, rgb\_ration\_np)

**Результат:**



**Задание 6:** Реализовать кодирование длин серий (Run-length encoding). Дан вектор x. Необходимо вернуть кортеж из двух векторов одинаковой длины. Первый содержит числа, а второй - сколько раз их нужно повторить.

Пример: x = np.array([2, 2, 2, 3, 3, 3, 5]).

Ответ: (np.array([2, 3, 5]), np.array([3, 3, 1])).

**Код:**

from collections import Counter

x\_py = [2, 2, 2, 3, 3, 3, 5]

x\_np = np.array(x\_py)

def RLE\_py(x\_py):

return tuple(zip(\*Counter(x\_py).items()))

def RLE\_np(x\_np):

bin\_count = np.bincount(x\_np)

non\_zero = np.nonzero(bin\_count)[0]

return non\_zero, bin\_count[non\_zero]

print('Python:', RLE\_py(x\_py))

print('Time: ', end='')

%timeit RLE\_py(x\_py)

print('NumPy:', RLE\_np(x\_np))

print('Time: ', end='')

%timeit RLE\_np(x\_np)

**Результат:**

Python: ((2, 3, 5), (3, 3, 1))

Time: 2.28 µs ± 48.7 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

NumPy: (array([2, 3, 5], dtype=int64), array([3, 3, 1], dtype=int64))

Time: 2.98 µs ± 65.9 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

**Задание 7:** Даны две выборки объектов - X и Y. Вычислить матрицу евклидовых расстояний между объектами.  
Сравнить с функцией scipy.spatial.distance.euclidean.

Пример:

x: [2 7 6 6 9 6 3 4 9]

y: [1 0 0 7 2 2 4 3 0]

Ответ: 15.329709716755891

**Код:**

import scipy.spatial

x\_py = [2, 7, 6, 6, 9, 6, 3, 4, 9]

y\_py = [1, 0, 0, 7, 2, 2, 4, 3, 0]

x\_np = np.array(x\_py)

y\_np = np.array(y\_py)

def euclid\_py(x\_py, y\_py):

return (sum((x - y) \*\* 2 for x, y in zip(x\_py, y\_py)))\*\*0.5

def euclid\_np(x\_np, y\_np):

return np.linalg.norm(x\_np - y\_np)

print('Python:', euclid\_py(x\_py, y\_py))

print('Time: ', end='')

%timeit euclid\_py(x\_py, y\_py)

print('NumPy:', euclid\_np(x\_np, y\_np))

print('Time: ', end='')

%timeit euclid\_np(x\_np, y\_np)

print("Scipy:", scipy.spatial.distance.euclidean(x\_py, y\_py))

print('Time: ', end='')

%timeit scipy.spatial.distance.euclidean(x\_py, y\_py)

**Результат:**

Python: 15.329709716755891

Time: 3.18 µs ± 67.8 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

NumPy: 15.329709716755891

Time: 6.3 µs ± 101 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

Scipy: 15.329709716755891

Time: 12.6 µs ± 141 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

**Задание 8:** Реализовать функцию вычисления логарифма плотности многомерного нормального распределения. Входные параметры: точки X, размер (N, D), мат. ожидание m, вектор длины D, матрица ковариаций C, размер (D, D). Разрешается использовать библиотечные функции для подсчета определителя матрицы, а также обратной матрицы, в том числе в невекторизованном варианте. Сравнить с scipy.stats.multivariate\_normal(m, C).logpdf(X) как по скорости работы, так и по точности вычислений

**Код:**

import math

import scipy.stats

sigma = np.matrix([[2.3, 0, 0, 0],

[ 0, 1.5, 0, 0],

[ 0, 0, 1.7, 0],

[ 0, 0, 0, 2]])

mu = np.array([2, 3, 8, 10])

x = np.array([2.1, 3.5, 8, 9.5])

def norm\_pdf(x, mu, sigma):

size = len(x)

if size == len(mu) and (size, size) == sigma.shape:

det = np.linalg.det(sigma)

norm\_const = 1.0/ ( math.pow((2\*math.pi),float(size)/2) \* math.pow(det,1.0/2) )

x\_mu = np.matrix(x - mu)

inv = sigma.I

result = math.pow(math.e, -0.5 \* (x\_mu \* inv \* x\_mu.T))

return norm\_const \* result

print("Python:", norm\_pdf(x, mu, sigma))

print('Time: ', end='')

%timeit norm\_pdf(x, mu, sigma)

print("Scipy:", scipy.stats.multivariate\_normal(mu, sigma).pdf(x))

print('Time: ', end='')

%timeit scipy.stats.multivariate\_normal(mu, sigma).pdf(x)

**Результат:**

Python: 0.006378411393413104

Time: 81.6 µs ± 2.89 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)

Scipy: 0.006378411393413108

Time: 167 µs ± 1.14 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)

**Вывод:**

Благодаря NumPy код выглядит чище, а так-же легче реализовать некоторые функции, но чаще всего код через NumPy требует больше времени на обработку.