

Министерство образования Республики Беларусь

Учреждение образования

“Брестский государственный университет”

Кафедра ИИТ

Лабораторная работа №7

По дисциплине “Языки программирования”

Вариант №7

Выполнил:

Кравцевич Г.А. (ПО-7,2)

Проверил:

Дряпко. А. В.

Дата выполнения:

13.11.21

Брест 2021

**Правила заданий:**

1. Для написания кода использовать библиотеки классического Python, NumPy и SciPy.
2. Код демонстрируется в Jupyter Notebook
3. По каждому заданию должно быть предоставлено не менее 3-х вариантов решения, среди которых:
  - а. чистый NumPy (максимально оптимизированный, векторизованный)
  - б. любой не векторизованный вариант
  - с. любой другой вариант, желательно конкурентноспособный
4. Все варианты решения должны быть протестированы на скорость выполнения при помощи `%timeit`
5. Полученные результаты отразить в отчете и сделать выводы о производительности и комфорте использования NumPy в различных задачах.

## Задание 1:

Подсчитать произведение ненулевых элементов на диагонали прямоугольной матрицы.

### Код программы:

In [2]:

```
def np_style():
    x = np.array(
        [[1, 0, 1],
         [2, 0, 2],
         [3, 0, 3],
         [4, 4, 4]])
    output = x.diagonal()[x.diagonal() != 0].prod()
    return output

def py_style():
    x = [[1, 0, 1],
          [2, 0, 2],
          [3, 0, 3],
          [4, 4, 4]]

    prod = 1
    diagonal = [x[i][i] for i in range(min(len(x), len(x[0])))] if x[i][i] != 0
    for x in diagonal:
        prod *= x
    return prod

@njit
def numba_style():
    x = [[1, 0, 1],
          [2, 0, 2],
          [3, 0, 3],
          [4, 4, 4]]

    prod = 1
    diagonal = [x[i][i] for i in range(min(len(x), len(x[0])))] if x[i][i] != 0
    for x in diagonal:
        prod *= x
    return prod

print(f'Result: {numba_style()}')
%timeit np_style()
%timeit numba_style()
%timeit py_style()
```

Result: 3

22.6  $\mu$ s  $\pm$  616 ns per loop (mean  $\pm$  std. dev. of 7 runs, 10000 loops each)

2.04  $\mu$ s  $\pm$  64.2 ns per loop (mean  $\pm$  std. dev. of 7 runs, 100000 loops each)

2.57  $\mu$ s  $\pm$  137 ns per loop (mean  $\pm$  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]]])`.

```
In [3]: def np_style():
        x = np.array([[9, 4, 2], [6, 0, 0], [9, 9, 3]])
        i = np.array([1, 2, 1])
        j = np.array([1, 0, 1])

        return x[i, j]

def py_style():
    x = [[9, 4, 2], [6, 0, 0], [9, 9, 3]]
    i = [1, 2, 1]
    j = [1, 0, 1]

    return [x[i[index]][j[index]] for index in range(len(i))]

@njit
def numba_style():
    x = [[9, 4, 2], [6, 0, 0], [9, 9, 3]]
    i = [1, 2, 1]
    j = [1, 0, 1]

    return [x[i[index]][j[index]] for index in range(len(i))]

print(f'Result: {np_style()}')
%timeit np_style()
%timeit py_style()
%timeit numba_style()

Result: [0 9 0]
23 µs ± 942 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
2.05 µs ± 27 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
The slowest run took 5.86 times longer than the fastest. This could mean that an intermediate result is being cached.
5.16 µs ± 4.6 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

## 3 задание:

Даны два вектора  $x$  и  $y$ . Проверить, задают ли они одно и то же мультимножество.

```
In [4]: def np_style():
        x = np.array([1, 2, 2, 4])
        y = np.array([4, 2, 1, 2])
        x.sort()
        y.sort()
        return all(x == y)

def py_style():
    x = [1, 2, 2, 4]
    y = [4, 2, 1, 2]
    return x.sort() == y.sort()

@njit
def numba_style():
    x = [1, 2, 2, 4]
    y = [4, 2, 1, 2]
    return x.sort() == y.sort()

print(f'Result: {py_style()}')
%timeit np_style()
%timeit py_style()
%timeit numba_style()

Result: True
12.6 µs ± 163 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
628 ns ± 11.2 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
The slowest run took 5.84 times longer than the fastest. This could mean that an intermediate result is being cached.
5.8 µs ± 5.28 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

#### 4 задание:

Найти максимальный элемент в векторе x среди элементов, перед которыми стоит нулевой

```
In [5]: def np_style():
        x = np.array([6, 2, 0, 3, 0, 0, 5, 7, 0])
        return x[1:][x == 0][: -1].max()

        def py_style():
            x = [6, 2, 0, 3, 0, 0, 5, 7, 0]
            return max([x[i] for i in range(1, len(x)) if x[i - 1] == 0])

        @jit
        def numba_style():
            x = [6, 2, 0, 3, 0, 0, 5, 7, 0]
            return max([x[i] for i in range(1, len(x)) if x[i - 1] == 0])

        print(f'Result: {py_style()}')
        %timeit np_style()
        %timeit py_style()
        %timeit numba_style()
```

```
Result: 5
14.9 µs ± 366 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
2.36 µs ± 113 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
The slowest run took 8.93 times longer than the fastest. This could mean that an intermediate result is being cached.
3.39 µs ± 3.75 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

### 5 задание:

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, используя коэффициенты `np.array([0.299, 0.587, 0.114])`.

### Исходная картинка:

```
In [6]: coefs = np.array([0.299, 0.587, 0.114, 0])  
img = np.asarray(PIL.Image.open("1.png"))  
  
PIL.Image.fromarray(img)
```

Out[6]:



## Код программы:

```
In [7]: def np_style():
        new_img = img * ([[coefs] * img.shape[1]]) * img.shape[0]
        return new_img.sum(axis=2)

def py_style():
    result = []
    for image_pixels_line in img:
        line = []
        for pixel in image_pixels_line:
            line.append(sum([pixel[i] * coefs[i] for i in range(len(coefs))]))
        result.append(line)
    return result

@njit
def numba_style():
    result = []
    for image_pixels_line in img:
        line = []
        for pixel in image_pixels_line:
            line.append(sum([pixel[i] * coefs[i] for i in range(len(coefs))]))
        result.append(line)
    return result

%timeit np_style()
%timeit py_style()
%timeit numba_style()
```

219 ms ± 14 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

7.95 s ± 113 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

152 ms ± 6.13 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)

## Результаты:

```
In [8]: PIL.Image.fromarray(np.asarray(py_style(), dtype=np.uint8))
```

Out[8]:



```
In [9]: PIL.Image.fromarray(np.asarray(np_style(), dtype=np.uint8))
```

Out[9]:





## 6 задание:

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

In [10]:

```
def np_style():
    x = np.array([2, 2, 2, 3, 3, 5])
    return np.unique(np.array(x), return_counts=True)

def py_style():
    x = [2, 2, 2, 3, 3, 5]
    values = list(set(x))
    counts = [x.count(value) for value in values]
    return [values, counts]

@njit
def numba_style():
    x = [2, 2, 2, 3, 3, 5]
    values = list(set(x))
    counts = [x.count(value) for value in values]
    return [values, counts]

print(f'Result: {py_style()}')
%timeit np_style()
%timeit py_style()
%timeit numba_style()
```

```
Result: [[2, 3, 5], [3, 2, 1]]
39.7 µs ± 1.63 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
2.08 µs ± 74 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
The slowest run took 9.18 times longer than the fastest. This could mean that an intermediate result is being cached.
7.84 µs ± 9.57 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

## 7 задание:

Даны две выборки объектов - X и Y. Вычислить матрицу евклидовых расстояний между объектами

```
In [11]: def np_style():
x = np.array([2, 7, 6, 6, 9, 6, 3, 4, 9])
y = np.array([1, 0, 0, 7, 2, 2, 4, 3, 0])
return math.sqrt(((x - y) ** 2).sum())

def np_style2():
x = np.array([2, 7, 6, 6, 9, 6, 3, 4, 9])
y = np.array([1, 0, 0, 7, 2, 2, 4, 3, 0])
return np.linalg.norm(x - y)

def py_style():
x = [2, 7, 6, 6, 9, 6, 3, 4, 9]
y = [1, 0, 0, 7, 2, 2, 4, 3, 0]
return math.sqrt(sum([(x[i] - y[i]) ** 2 for i in range(len(x))]))

@jit
def numba_style():
x = [2, 7, 6, 6, 9, 6, 3, 4, 9]
y = [1, 0, 0, 7, 2, 2, 4, 3, 0]
return math.sqrt(sum([(x[i] - y[i]) ** 2 for i in range(len(x))]))

x = np.array([2, 7, 6, 6, 9, 6, 3, 4, 9])
y = np.array([1, 0, 0, 7, 2, 2, 4, 3, 0])

print(f'Result: {np_style()}\n')

print('NumPy')
# numpy
%timeit np_style()
%timeit np_style2()
print('SciPy')
# scipy
%timeit euclidean(x, y)
print('Python')
# python
%timeit py_style()
#numba
print('Numba')
%timeit numba_style()

Result: 15.329709716755891

NumPy
17.3 µs ± 164 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
20.9 µs ± 810 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
SciPy
17.3 µs ± 277 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
Python
5.77 µs ± 112 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
Numba
The slowest run took 11.48 times longer than the fastest. This could mean that an intermediate result is being cached.
5.56 µs ± 7.58 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

## 8 задание:

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  $(D, D)$ .

```
In [12]: N = 5
D = 6

mean = np.random.rand(D)
cov = np.random.rand(D, D)

X = np.random.rand(N, D)

def np_style():
    return np.random.multivariate_normal(mean=mean, cov=cov)

def scipy_style():
    return [multivariate_normal(mean=1, cov=1).pdf(x_line) for x_line in X]

print(f'Result: {np_style()}')

%timeit np_style()
%timeit scipy_style()
```

C:\Users\PC\AppData\Local\Temp\ipykernel\_14800\181984646.py:10: RuntimeWarning: cc  
return np.random.multivariate\_normal(mean=mean, cov=cov)  
Result: [-1.69954788 -0.79868439 -2.76519186 1.67408273 1.44050679 1.29776053]  
189 µs ± 11.8 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)  
1.45 ms ± 33.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)

## Вывод:

Ознакомился с библиотекой numru, применил её на практике и сравнил с другими вариантами решения поставленной задачи.