

TOMSK
POLYTECHNIC
UNIVERSITY



ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Системный анализ процессов химической технологии

Лабораторная работа №5 Массивы NumPy

Вячеслав Алексеевич Чузлов,
к.т.н., доцент ОХИ ИШПР

20 февраля 2023 г.

Преобразование списков Python в массивы NumPy

- Для того, чтобы создать объект массива NumPy из объекта списка Python, можно использовать функцию `np.array`:

```
1 >>> import numpy as np
2 >>> np.array([1, 3, 5, 4, 2]) # Массив целочисленных значений
3 array([1, 3, 5, 4, 2])
```

- В отличие от стандартных списков Python, массивы NumPy могут содержать элементы только одного типа. Если типы элементов не совпадают, NumPy сделает попытку повышающего приведения типов:

```
4 >>> np.array([3.14, 4, 2, 3, 2.71])
5 array([3.14, 4. , 2. , 3. , 2.71])
```

- В тех случаях, когда требуется явно задать тип результирующего массива, необходимо воспользоваться ключевым аргументом `dtype`:

```
6 >>> np.array([1, 3, 5, 4, 2], dtype='float32')
7 array([1., 3., 5., 4., 2.], dtype=float32)
```

Преобразование списков Python в массивы NumPy

- В отличие от списков, массивы NumPy можно явным образом описать как многомерные:

```
8 >>> np.array([range(i, i + 3) for i in [2, 4, 6]])
9 array([[2, 3, 4],
10        [4, 5, 6],
11        [6, 7, 8]])
```

Создание массивов

Массивы больших размеров эффективнее генерировать с помощью встроенных методов.

- Создаем массив целых чисел длины 10, заполненный нулями:

```
1 >>> np.zeros(10, dtype=int)
2 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

- Создадим массив размером 3×5 значений с плавающей точкой, заполненный единицами:

```
3 >>> np.ones((3, 5), dtype=float)
4 array([[1., 1., 1., 1., 1.],
5        [1., 1., 1., 1., 1.],
6        [1., 1., 1., 1., 1.]])
```

- Создадим массив размером 3×5 , заполненный значением 2.98:

```
7 >>> np.full((3, 5), 2.98)
8 array([[2.98, 2.98, 2.98, 2.98, 2.98],
9        [2.98, 2.98, 2.98, 2.98, 2.98],
10       [2.98, 2.98, 2.98, 2.98, 2.98]])
```

Создание массивов

- Создадим массив, заполненный линейной последовательностью, начинающейся с 0 и заканчивающейся 20, с шагом 2 (аналогично встроенной функции `range()`):

```
11 >>> np.arange(0, 20, 2)
12 array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

- Создадим массив из пяти значений, равномерно располагающихся между 0 и 1:

```
13 >>> np.linspace(0, 1, 5)
14 array([0.   , 0.25, 0.5  , 0.75, 1.   ])
```

- Создадим массив размером 3×3 равномерно распределенных случайных значений от 0 до 1

```
15 >>> np.random.random((3, 3))
16 array([[0.28209864, 0.0014687 , 0.71104325],
17        [0.58773349, 0.89099475, 0.87835838],
18        [0.81093666, 0.70139097, 0.12816416]])
```

- Создадим массив размером 3×3 случайных целых чисел в интервале $[0, 10]$

```
19 >>> np.random.randint(0, 10, (3, 3))
20 array([[0, 4, 0],
21        [6, 7, 2],
22        [0, 7, 3]])
```

Доступ к элементам массива

Индексация массивов NumPy во многом похожа на индексацию списков в Python. В одномерном массиве обратиться к i -му (считая с 0) значению можно по требуемому индексу в квадратных скобках, по аналогии со стандартными списками:

```
1 >>> x1 = np.array([5, 0, 3, 3, 7, 9])
2 >>> x1[0]
3 5
4 >>> x1[4]
5 7
```

Для индексации элементов с конца массива следует использовать отрицательные индексы:

```
6 >>> x1[-1]
7 9
8 >>> x1[-2]
9 7
```

Доступ к элементам массива



Для обращения к элементам многомерного массива нужно указать кортеж индексов, разделенных запятыми:

```
10 >>> x2 = np.array([[3, 5, 2, 4], [7, 6, 8, 8], [1, 6, 7, 7]])
11 >>> x2
12 array([[3, 5, 2, 4],
13         [7, 6, 8, 8],
14         [1, 6, 7, 7]])
15 >>> x2[0, 0]
16 3
17 >>> x2[2, 0]
18 1
19 >>> x2[2, -1]
20 7
```


Доступ к элементам массива

При помощи любой из указанных выше нотаций можно изменять значения элементов массива:

```
21 >>> x2[0, 0] = 24
22 >>> x2
23 array([[24,  5,  2,  4],
24        [ 7,  6,  8,  8],
25        [ 1,  6,  7,  7]])
```

Следует помнить, что, в отличие от списков, массивы NumPy имеют фиксированный тип данных. Если вставить в массив целых чисел значение с плавающей точкой, оно будет неявно усечено:

```
26 >>> x1[0] = 2.71828 # Это значение будет усечено!
27 >>> x1
28 array([2, 0, 3, 3, 7, 9])
```

Срезы массивов: доступ к подмассивам

- По аналогии с доступом к определенным элементам массива квадратные скобки используются для доступа к отдельным частям массива при помощи операции среза (slicing), обозначаемой знаком двоеточия (:).
- Синтаксически срезы массивов NumPy соответствуют срезам стандартных списков Python:

`x[начало:конец:шаг]`

при этом любое из значений можно не указывать, тогда по умолчанию будут приняты следующие значения: начало = 0, конец = размер соответствующего измерения, шаг = 1.

Одномерные подмассивы

```
1 >>> x = np.arange(10)
2 >>> x
3 array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
4 >>> x[:5]      # первые пять элементов
5 array([0, 1, 2, 3, 4])
6 >>> x[5:]      # элементы после индекса = 5
7 array([5, 6, 7, 8, 9])
8 >>> x[4:7]     # подмассив из середины
9 array([4, 5, 6])
```

Срезы массивов: доступ к подмассивам

```
10 >>> x[::2]    # каждый второй элемент
11 array([0, 2, 4, 6, 8])
12 >>> x[1::2]   # каждый второй элемент, начиная с индекса 1
13 array([1, 3, 5, 7, 9])
```

- Также сохранена возможность использования отрицательного значения параметра шаг. Тогда значения по умолчанию для параметров **начало** и **конец** будут поменены местами. Это быстрый способ перевернуть массив:

```
14 >>> x[::-1]    # все элементы в обратном порядке
15 array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
16 >>> x[5::-1]    # все элементы в обратном порядке, начиная с индекса 5
17 array([5, 4, 3, 2, 1, 0])
18 >>> x[5::-2]    # каждый второй элемент в обратном порядке, начиная с индекса 5
19 array([5, 3, 1])
```

Срезы массивов: доступ к подмассивам

Многомерные подмассивы

```
1 >>> x2
2 array([[24,  5,  2,  4],
3        [ 7,  6,  8,  8],
4        [ 1,  6,  7,  7]])
5 >>> x2[:2, :3]           # две строки, три столбца
6 array([[24,  5,  2],
7        [ 7,  6,  8]])
8 >>> x2[:3, ::2]          # все строки, каждый второй столбец
9 array([[24,  2],
10       [ 7,  8],
11       [ 1,  7]])
12 >>> x2[::-1, ::-1]       # обратный порядок строк и столбцов
13 array([[ 7,  7,  6,  1],
14       [ 8,  8,  6,  7],
15       [ 4,  2,  5, 24]])
```

Срезы массивов: доступ к подмассивам

Доступ к строкам и столбцам массива

Распространенной задачей является доступ к отдельным строкам или столбцам массива. Получить такой доступ можно при помощи комбинации операций индексации и среза:

```
16 >>> print(x2[:, 0])    # первый столбец массива x2
17 [24  7  1]
18 >>> print(x2[0, :])    # первая строка массива x2
19 [24  5  2  4]
```

При необходимости получения доступа к строке, операция взятия среза может быть опущена для более лаконичной записи:

```
1 >>> print(x2[0])    # эквивалентно x2[0, :]
2 [24  5  2  4]
```

Срезы массивов создают разделяемые ссылки

- Срезы массивов возвращают **разделяемые ссылки** (синонимы), а не **копии** данных массива.
- Этим срезы массивов библиотеки NumPy отличаются от срезов списков языка Python (срезы списков создают новые объекты):

```
1 >>> print(x2)
2 [[24  5  2  4]
3  [ 7  6  8  8]
4  [ 1  6  7  7]]
```

Получим из него матрицу 2×2 :

```
5 >>> x2_sub = x2[:2, :2]
6 >>> print(x2_sub)
7 [[24  5]
8  [ 7  6]]
```

Срезы массивов создают разделяемые ссылки



Теперь, если изменить значения этой матрицы, исходный массив также поменялся:

```
9 >>> x2_sub[0, 0] = 100
10 >>> print(x2_sub)
11 [[100  5]
12  [ 7  6]]
13 >>> print(x2)
14 [[100  5  2  4]
15  [ 7  6  8  8]
16  [ 1  6  7  7]]
```

Создание копий массивов

В ряде случаев требуется явно скопировать содержимое массива или его части. Для решения данной задачи существует метод `copy()` :

```
17 >>> x2_sub_copy = x2[:2, :2].copy()
18 >>> print(x2_sub_copy)
19 [[100  5]
20  [ 7  6]]
```

Теперь, если изменить значения этого подмассива, то исходный массив не изменится:

```
21 >>> x2_sub_copy[0, 0] = 24
22 >>> print(x2_sub_copy)
23 [[24  5]
24  [ 7  6]]
25 >>> print(x2)
26 [[100  5  2  4]
27  [ 7  6  8  8]
28  [ 1  6  7  7]]
```


TOMSK
POLYTECHNIC
UNIVERSITY



ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Вычисления с массивами NumPy

Арифметические функции с массивами

Универсальные функции NumPy могут быть легко использованы, т.к. основаны на нативных арифметических операторах Python. Доступны обычные операторы сложения, вычитания, умножения и деления:

```
1 >>> x = np.arange(4)
2 >>> print('x      =', x)
3 x      = [0 1 2 3]
4 >>> print('x + 5  =', x + 5)
5 x + 5  = [5 6 7 8]
6 >>> print('x - 5  =', x - 5)
7 x - 5  = [-5 -4 -3 -2]
8 >>> print('x * 2  =', x * 2)
9 x * 2  = [0 2 4 6]
10 >>> print('x / 2  =', x / 2)
11 x / 2  = [0.  0.5 1.  1.5]
12 >>> print('x // 2 =', x // 2)
13 x // 2 = [0 0 1 1]
```

Арифметические функции с массивами



Определены также унарная универсальная функция изменения знака, оператор `**` для возведения в степень и оператор `%` для деления по модулю:

```
14 >>> print('-x      =', -x)
15 -x      = [ 0 -1 -2 -3]
16 >>> print('x ** 2 =', x ** 2)
17 x ** 2 = [0 1 4 9]
18 >>> print('x % 2  =', x % 2)
19 x % 2  = [0 1 0 1]
```

Арифметические функции с массивами

Данные операции могут быть использованы в выражениях любыми способами с соблюдением стандартных приоритетов:

```
20 >>> -(0.5 * x + 1) ** 2  
21 array([-1.   , -2.25, -4.   , -6.25])
```

Все арифметические операторы – удобные аналоги для встроенных функций библиотеки NumPy. Например, оператор `+` является аналогом функции `add()` :

Оператор	Эквивалентная функция	Описание
+	<code>np.add</code>	Сложение: $1 + 1 = 2$
-	<code>np.subtract</code>	Вычитание: $3 - 2 = 1$
-	<code>np.negative</code>	Унарная операция изменения знака: -2
*	<code>np.multiply</code>	Умножение: $2 * 3 = 6$
/	<code>np.divide</code>	Деление: $3 / 2 = 1.5$
//	<code>np.floor_divide</code>	Деление с округлением в меньшую сторону: $3 // 2 = 1$
**	<code>np.power</code>	Возведение в степень: $3 ** 2 = 9$
%	<code>np.mod</code>	Модуль/остаток: $5 \% 2 = 1$

Получение абсолютного значения

Наряду со встроенными арифметическими операторами, с массивами NumPy можно использовать стандартную функцию `abs()` языка Python для получения абсолютного значения:

```
1 >>> x = np.array([-2, -1, 0, 1, 2])
2 >>> abs(x)
3 array([2, 1, 0, 1, 2])
```

Аналогичная универсальная функция NumPy – `np.absolute()`, доступна также под псевдонимом `np.abs()`:

```
4 >>> np.absolute(x)
5 array([2, 1, 0, 1, 2])
6 >>> np.abs(x)
7 array([2, 1, 0, 1, 2])
```

Тригонометрические функции



Библиотека NumPy предоставляет набор тригонометрических функций:

```
1 >>> alpha = np.linspace(0, np.pi, 3)
2 >>> print('alpha      = ', alpha)
3 alpha      = [0.          1.57079633  3.14159265]
4 >>> print('sin(alpha) = ', np.sin(alpha))
5 sin(alpha) = [0.00000000e+00  1.00000000e+00  1.2246468e-16]
6 >>> print('cos(alpha) = ', np.cos(alpha))
7 cos(alpha) = [ 1.0000000e+00  6.123234e-17 -1.0000000e+00]
8 >>> print('tan(alpha) = ', np.tan(alpha))
9 tan(alpha) = [ 0.00000000e+00  1.63312394e+16 -1.22464680e-16]
```

Значения вычисляются в пределах точности конкретной вычислительной машины, вследствие чего некоторые из них не всегда точно равны нулю, хотя должны.

Тригонометрические функции



Определены также и обратные тригонометрические функции:

```
10 >>> x = [-1, 0, 1]
11 >>> print('x = ', x)
12 x = [-1, 0, 1]
13 >>> x = [-1, 0, 1]
14 >>> print('x = ', x)
15 x = [-1, 0, 1]
16 >>> print('arcsin(x) = ', np.arcsin(x))
17 arcsin(x) = [-1.57079633  0.          1.57079633]
18 >>> print('arccos(x) = ', np.arccos(x))
19 arccos(x) = [3.14159265 1.57079633 0.          ]
20 >>> print('arctan(x) = ', np.arctan(x))
21 arctan(x) = [-0.78539816  0.          0.78539816]
```

Показательные функции и логарифмы



Показательные функции – один из распространенных типов операций, доступных в NumPy:

```
1 >>> x = [1, 2, 3]
2 >>> print('x = ', x)
3 x = [1, 2, 3]
4 >>> print('e^x = ', np.exp(x))
5 e^x = [ 2.71828183  7.3890561 20.08553692]
6 >>> print('2^x = ', np.exp2(x))
7 2^x = [2.  4.  8.]
8 >>> print('3^x = ', np.power(3, x))
9 3^x = [ 3  9 27]
```


Показательные функции и логарифмы

Определены также и логарифмы. Простейшая функция `np.log()` возвращает натуральный логарифм числа. Если Вам требуется логарифм по основанию 2 или 10, они также доступны:

```
10 >>> x = [1, 2, 4, 10]
11 >>> print('x      =', x)
12 x      = [1, 2, 4, 10]
13 >>> print('ln(x)   =', np.log(x))
14 ln(x)   = [0.          0.69314718  1.38629436  2.30258509]
15 >>> print('log2(x) =', np.log2(x))
16 log2(x) = [0.          1.          2.          3.32192809]
17 >>> print('log10(x) =', np.log10(x))
18 log10(x) = [0.          0.30103    0.60205999  1.          ]
```

TOMSK
POLYTECHNIC
UNIVERSITY



ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Функции агрегирования

Суммирование значений массива

В стандартном Python данная задача решается при помощи встроенной функции `sum()`. Синтаксис этой функции крайне похож на функцию `sum()` библиотеки NumPy:

```
1 >>> import numpy as np
2 >>> arr = np.random.random(100)
3 >>> sum(arr)
4 49.496408327779065
5 >>> np.sum(arr)
6 49.49640832777906
```

NumPy версия функции `sum()` работает намного быстрее:

```
7 >>> big_array = np.random.rand(1000000)
8 >>> %timeit sum(big_array) # Магическая команда в IPython
9 62.8 ms ± 598 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
10 >>> %timeit np.sum(big_array)
11 522 µs ± 5.53 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Важно заметить, что функции `sum()` и `np.sum()` не идентичны. Так, смысл их опциональных аргументов отличается и функция `np.sum()` может работать с многомерными массивами.

Минимум и максимум

В стандартном Python определены встроенные функции `min()` и `max()`, служащие для вычисления минимального и максимального значений любой коллекции:

```
1 >>> min(big_array), max(big_array)
2 (1.4200179176970806e-07, 0.9999998044567884)
```

Соответствующие функций NumPy имеют аналогичный синтаксис и работают быстрее:

```
3 >>> np.min(big_array), np.max(big_array)
4 (1.4200179176970806e-07, 0.9999998044567884)
5 >>> %timeit min(big_array)
6 43.4 ms ± 719 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
7 >>> %timeit np.min(big_array)
8 304 µs ± 2.03 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Существует возможность вычисления некоторых сводных показателей путем вызова соответствующих методов объекта массива NumPy для более лаконичной записи:

```
9 >>> print(big_array.min(), big_array.max(), big_array.sum())
10 1.4200179176970806e-07 0.9999998044567884 499940.4506893933
```

TOMSK
POLYTECHNIC
UNIVERSITY



ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Примеры

Пример 1



Заполнить матрицу $a(3, 3)$ случайными целыми числами от 1 до 10. Вывести значения ее элементов. Вычислить сумму элементов первой и последней строк данной матрицы и вывести результат вычисления.

```
1 >>> import numpy as np
2 >>> a = np.random.randint(1, 10, (3, 3))
3 >>> a
4 array([[5, 2, 7],
5        [7, 2, 1],
6        [8, 2, 3]])
7 >>> s = a[0].sum() + a[-1].sum()
8 >>> s
9 27
```

Пример 2

Матрица p размером 2×3 заполнена случайными целыми числами в диапазоне $[-3; 5]$. Массив x содержит 10 случайных чисел в диапазоне $[-10; 10]$. Необходимо найти сумму минимального элемента первой строки матрицы p и максимального элемента массива x и поделить ее на количество элементов массива x , больших нуля.

```
1 >>> import numpy as np
2 >>> p = np.random.randint(-3, 3, (2, 3))
3 >>> p
4 array([[ -2,  2,  2],
5        [ 0, -2,  2]])
6 >>> x = np.random.randint(-10, 10, 10)
7 >>> x
8 array([ 7, -7, -8, -7, -8, -3, -8,  6, -5,  6])
9 >>> min_p = p[0].min()
10 >>> max_x = x.max()
11 >>> min_p, max_x
12 (-2, 7)
13 >>> xshape = x[x > 0].shape
14 >>> xshape
15 (3,)
16 >>> (max_x + min_first_row) / xshape[0]
17 1.6666666666666667
```

Пример 3

Заполнить матрицу $a(4, 4)$ случайными целыми числами от 1 до 100. Вывести значения ее элементов. Найти максимальный элемент в каждой строке. Среди максимальных элементов каждой строки найти минимальный.

```
1 >>> import numpy as np
2 >>> a = np.random.randint(1, 100, (4, 4))
3 >>> a
4 array([[62, 97, 91, 19],
5        [34, 34, 92, 70],
6        [96, 92, 46, 80],
7        [97,  2, 75, 52]])
8 >>> max_of_rows = a.max(axis=1)  # axis=0 - поиск максимума по столбцам
9 >>> max_of_rows
10 array([97, 92, 96, 97])
11 >>> max_of_rows.min()
12 92
```


Пример 4

Заполнить матрицу $a(4, 4)$ случайными числами от -3 до 6 . Вывести значения ее элементов. Вычислить среднее арифметическое значений неотрицательных элементов каждого столбца данной матрицы.

```
1 >>> import numpy as np
2 >>> a = np.random.randint(-3, 6, (4, 4))
3 >>> a
4 array([[ 3, -3,  0, -1],
5         [-2,  1, -1,  4],
6         [ 3, -2,  0,  5],
7         [-1, -2,  0,  0]])
8 >>> mean_of_cols = a.mean(axis=0, where=a>=0)
9 >>> mean_of_cols
10 array([3., 1., 0., 3.] )
```

TOMSK
POLYTECHNIC
UNIVERSITY



ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Задания

Задание 1



Заполнить матрицу a $(3, 3)$ случайными целыми числами от -5 до 5 . Вывести значения ее элементов на экран. Найти произведение минимального элемента матрицы на сумму ее положительных элементов. Вывести результат.

Задание 2



Заполнить матрицу a (5, 10) случайными целыми числами от 0 до 9. Вывести значения элементов матрицы. Найти столбец матрицы с максимальной суммой элементов и вывести эту сумму на экран.

Задание 3

Найти сумму положительных (p) и количество отрицательных (o) элементов массива z (10), заполненного случайными целыми числами в диапазоне $[-7; 5]$ и минимальный элемент (b_{min}) второго столбца матрицы b (5, 5), заполненной случайными числами в диапазоне $[-10; 20]$, и вычислить значения элементов массива x_i :

$$x_i = \frac{\sqrt{p}}{o + a \cdot c} + b_{min} \cdot k^2 + z_i$$

где $a = 2.5e-3$; $c = 175$; $k = 8$.

TOMSK
POLYTECHNIC
UNIVERSITY



ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Контакты

Вячеслав Алексеевич Чузлов,
к.т.н., доцент ОХИ ИШПР



Учебный корпус №2, ауд. 136



chuva@tpu.ru



+7-962-782-66-15

Благодарю за внимание!