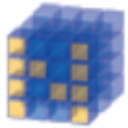
1. Библиотеки (пакеты модулей) для научно-инженерных расчетов



NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

IP[y]: IPython

IPython
Enhanced Interactive Console

Sympy
Symbolic mathematics

pandas
Data structures & analysis

Pythonic DS:

- `numpy` — большие многомерные массивы и матрицы;
- `pandas` — манипуляции с табличными данными (сохранение, загрузка, обработка) («panel data»);
- `sympy` — библиотека **символьных** вычислений.
- `scipy` — библиотека модулей для математической оптимизации, интегрирования, специальных функций, обработки сигналов, обработки изображений, генетических алгоритмов, решения ОДУ и т.д.; `scipy.stats` включает некоторые статистические функции.
- `statsmodels` — статистические модели и тесты;
- `matplotlib` — 2D-визуализация, построение графиков;
- `seaborn` — визуализация (использует matplotlib);
- `Bokeh` — визуализация данны для веб-приложений;
- `NetworkX` — построение и визуализация графов;
- `PySpark` — интерфейс для Spark, фреймворка для распределенной обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop;

ML/DL:

- `Scikit-learn` — библиотека для машинного обучения;
- `Theano` — это расширение языка Python, позволяющее эффективно вычислять математические выражения, содержащие многомерные массивы, с использование CPU и GPU;
- `TensorFlow` — библиотека для глубинного обучения от Google;
- `Keras` — надстройка над Theano/TensorFlow, которая позволяет на более высоком уровне работать с нейросетями.
- `PyTorch` — интерфейс на Python к библиотеке для глубинного обучения Torch;
- `NLTK` — пакет библиотек и программ для символьной и статистической обработки естественного языка.
- `Gensim` — инструмент для автоматической обработки языка, основанный на машинном обучении.
- `XGboost` (Extreme Gradient Boosting) — библиотека машинного обучения, реализующая модель градиентного бустинга (на основе деревьев решений).

Распределенные вычисления:

- `PySpark` — интерфейс к Apache Spark, фреймворку для распределенной обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop;

2. NumPy

NumPy (Numerical Python) — пакет для высокопроизводительных вычислений над большими многомерными массивами (класс `ndarray`), в частности для работы с матрицами (класс `matrix`).

```
sudo pip3 install numpy
```

Основные классы:

- `ndarray` – многомерный массив; создаются с помощью функции `np.array`;
- `matrix(numpy.ndarray)` – матрицы; создаются с помощью `np.matrix`, `np.mat` или `asmatrix`;

Подпакеты Nympy:

- `numpy.random` – содержит ряд функций для генерации случайных чисел, например, random(), rand(), randn(), randint() и др.
- `numpy.matlib` — содержит функции для работы с матрицами.

2.1. **Класс ndarray.**

Пример создания массивов **NumPy** (объекта класса `ndarray`):

```
1 import numpy as np
2 a = np.array([1, 2, 3])
3 b = np.array([[1.5, 2, 3], [4, 5, 6]])
4 c = np.array([[1.5, 2, 3], [4, 5, 6]], dtype=np.complex)
```

`array()` — функция, создающая объект типа `ndarray` — преобразует различного вида последовательности в многомерные массивы.

По умолчанию тип данных `int32/float64`. Можно задать/получить с помощью `dtype`.

Атрибуты объектов `ndarray`:

- `ndim` — число осей (измерений) массива;
- `shape` — размеры массива, его форма;
- `size` — число всех элементов массива;
- `dtype` — объект, описывающий тип элементов массива;
- `itemsize` — размер каждого элемента массива в байтах;
- `data` — буфер, содержащий фактические элементы массива;

Функции модуля numpy для создания массивов ndarray:

- `np.array()` — конвертирует коллекцию/последовательность в массив ndarray;
- `np.asarray()` — конвертирует без копирования (если на входе ndarray);
- `np.arange()` — для создания массива как последовательности чисел (аналогично `range`);
- `np.eye(n), identity(n)` — единичная матрица (двумерный массив);
- `np.empty()` — массив без заполнения;
- `np.zeros((nx, ny))` — создает массив из нулей,
- `np.ones((nx, ny, nz))` — массив из единиц,
- `zeros_like, ones_like, empty_like` — создания массива той же формы и типа данных.

```
>>> np.arange(10, 30, 5)
array([10, 15, 20, 25])
```

2.2. Модуль random.

```
1 import numpy.random as random
2 deviation = 0.2
3 random.normal(size=10**3, scale=deviation)
```

- `random.seed([X], version=2)` — инициализация генератора случайных чисел;
- `permutation` — случайная перестановка;
- `randint(5, size=10)` — посл-ть целых чисел;
- `rand(d0,d1,...)` — равномерное распределение `[0,1)`;
- `rand(d0,d1,...)` — нормальное распределение ($M[X] = 0, \sigma = 1$);
- `normal(loc=0.0, scale=1.0, size=None)` — нормальное распределение ($M[X] = \text{loc}, \sigma = \text{scale}$);
- `uniform(low=0.0, high=1.0, size=None)` — равномерное распределение;

2.3. **Типы данных.**

```
1 arr = np.array([-2, 0, -3], dtype=np.float64)
```

Таблица 1. Типы данных NumPy

Типы данных	**Код**	**Коммент**
`int8, uint8`	`i1, u1`	1-байтный целый
`int16, uint16`	`i2, u2`	
`int32, uint32`	`i4, u4`	
`int64, uint64`	`i8, u8`	
`float16`	`f2`	
`float32`	`f4 or f`	
`float64 / float128`	`f8 or d`	Python `float`
`float128`	`f16 or g`	для 64-битных систем
`complex64 / 128 / 256`	`c8, c16, c32`	комплексные числа
`bool`	`?`	True, False
`object`	`O`	Python object type
`string_`	`S`	1 байт на символ, напр., `S10`
`unicode`	`U`	платформеннозависим, напр., `U10`

```
arr = np.array([-2, 0], dtype=np.float64)
arr = np.array([-2, 0], dtype=float)
arr = np.array([-2, 0], dtype='d')
arr = np.array([-2, 0], 'f8')
```

```
np.array(['python', 'numpy'], dtype='S6')
```

Метод `astype` — конвертация типа данных:

```
a = np.array([-2.2, 0.1])
b = a.astype(np.int32)
```

2.4. **Вычисления.** *Поэлементные операции:*

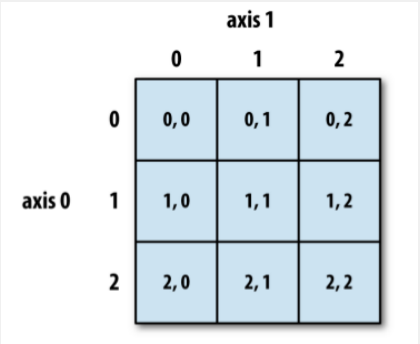
```
arr3 = arr1 * arr2
arr4 = arr1 + arr3
```

Broadcasting (распространение):

```
arr2 = arr * 5
arr2 = 5 * arr
arr *= 5
```

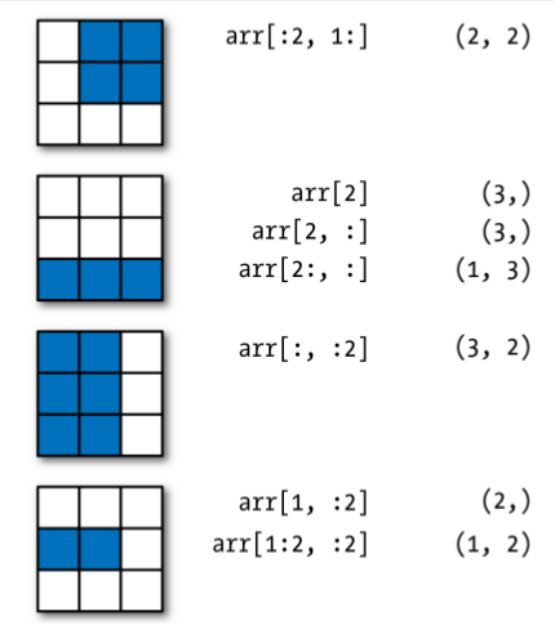
Индексация.

```
x = arr[1, 3]
x = arr[1][3]
```



Срезы (slicing).

```
s = arr[8:10]
arr[3:6] = 5
arr[3:6] = [0,0,0]
s[:] = -1
```



Условная логика и булева индексация.

```
arr == 5
arr[arr == 5]
arr[(arr >= 5) & (arr <= 8)]
```

`numpy.where(cond, xarr, yarr)` — векторизованная версия `x if cond else y`

```
np.where(arr >= 5, arr1, arr2)
np.where(arr >= 5, arr, -1)
np.where(arr >= 5, 1, -1)
np.where((arr >= 5) & (arr <= 8), 1, -1)
```

Методы/функции `any()`, `all()`.

Статистика. `sum, mean, std, var, min, max, argmin, argmax, cumsum, cumprod`

Суммирование:

```
arr.sum()
np.sum(arr)  # или sum(arr)
```

Среднее значение:

```
arr.mean()
arr.mean(axis=1)
```

Универсальные поэлементные функции. С одним аргументом (1 массив):

- `abs, sign, sqrt, square`
- `ceil, floor`
- `exp, log, log2, log10, cos, sin, tan, arccos, ...`

```
np.sin(arr)
```

С двумя аргументами (2 массива):

- `add, substract, multiply, divide`
- `maximum, minimum`

```
np.maximum(arr1, arr2)
```

Сортировка.

```
arr.sort()
arr.sort(kind='quicksort') # mergesort, heapsort
```

Kind	Speed	Stable	Work space	Worst-case
'quicksort'	1	No	0	$O(n^2)$
'mergesort'	2	Yes	n / 2	O(n log n)
'heapsort'	3	No	0	O(n log n)

2.5. **Reshape.**

```
arr = np.array([1, 2, 3, 4])
arr = arr.reshape(2, 2)
```

`ravel()`, `flatten()` — выравнивание в одномерный массив

`concatenate([arr1, arr2], axis=0)` — объединение

2.6. **Векторизация функций.**

`numpy.vectorize(pyfunc, otypes=None)` — определяет векторизованной вариант функции.

`numpy.frompyfunc(func, nin, nout)` — создание NumPy ufunc.

```
a = np.array([[1,2],[3,4]])
func = lambda x: math.sin(x)
vfunc = np.vectorize(func)
vfunc(a)
```

```
add = np.frompyfunc(lambda x,y: x+y, 2, 1)
add(np.arange(8), np.arange(8))   # dtype=object
```

```
add = np.vectorize(lambda x,y: x+y, otypes=[np.float64])
add(np.arange(8), np.arange(8))
```

```
np.add(arr1, arr2)
```

2.7. **Доп. возможности.** Функции `all, any`:

```
any(x < 5 for x in ls)
all(type(x) is np.int64 for x in s)
```

Методы `all, any`:

```
(arr[:,3] == 0).any()
(arr[:,3] == 0).all()
```

Срезы:

```
arr[:,[3,2,1]]
```

Индексация с помощью массивов:

```
b = np.array([(1,1), (1,2), (2,2)])
arr[b[:,0], b[:,1]] = -1
```

2.8. **Упражнения.** 1) Преобразовать массив `np.array([[1, 1],[2, 2]])` в список кортожей `[(1, 1), (2, 2)]`.

2) Удалить строки из массива numpy, в которых встречается `NaN`. Например,

```
x = np.array([[1,2,3], [4,5,np.nan], [7,8,9], [1,0,1]])
```

2.9. **Matrix.** Начнем с рассмотрения матриц. Следующий пример демонстрирует создание матрицы размера 2 на 2 и ее вывода на консоль.

```
>>> import numpy as np
>>> A = np.matrix([[1,2],[3,4]])
>>> print(A)
[[1 2]
 [3 4]]
```

С точки зрения ООП написанное здесь можно понимать следующим образом. Во второй строчке происходит создание экземпляра класса matrix (из пакета numpy), причем в конструктор класса передаем стандартный “питоновский” список, из которого и формируется “матрица” `A`.

После того, как матрица создана, над ней можно производить различные операции. Например, можно вычеслить квадрат этой матрицы:

```
>>> B = A**2
>>> print(B)
[[ 7 10]
 [15 22]]
```

Заметим, что для новой матрицы B нет необходимости явно вызывать конструктор. Он будет вызыван автоматически.

Аналогичным образом можно перемножать, складывать и вычитать матрицы, а также умножать матрицы на числа и т.п. (все эти операции уже перегружены в классе matrix).

Для более сложных вычислений над матрицами, например нахождения обратной матрицы, необходимо импортировать модуль `linalg` пакета `numpy`, либо отдельные фукнции из него. Например,

```
>>> from numpy.linalg import inv
>>> print(inv(A))  # вычисление обратной матрицы
[[-2.   1. ]
 [ 1.5 -0.5]]
```