Python

Introduction

Jeff Abrahamson

18 janvier 1 février 8 février 2019

A replacement for MatLab

numpy

- `ndarray` : *an n-dimensional array*

```
np.array([1,2,3])
np.array([[1, 2], [3, 4]])
np.array([1, 2, 3], dtype = complex)

a = np.array(...)
a.shape
a.shape = (n, m)
a.reshape(n, m)

a.dtype
a.itemsize
```

numpy

```
np.empty()
np.empty(shape)

np.zeros(shape)
np.ones(shape)

np.zeros(48).reshape(6,8)
np.zeros((6,8), dtype=np.int_)
```

numpy

```
np.asarray(a)
np.array(a)
np.fromiter(iterable, dtype, count = -1)


np.arange(start, stop, step, dtype)

np.linspace(start, stop, num, endpoint,
retstep, dtype)

np.logspace(start, stop, num, endpoint, base,
dtype)
```

numpy

```
a = arange(30)
s = slice(2,7,2)
a[s]

a[2:7:2]


m = np.arange(48).reshape(6,8)
m[1:]
m[1:3, 2:5]
m[:, 1]
m[..., 1]
m[:, :-1]
m[1:3, [1,3]]
```

numpy

```
m = np.arange(48).reshape(6,8)
m[m > 20]

n = np.zeros((6,8), dtype=np.int_)
n[3, 5] = 4
n[5, 3] = 2
m[n > 0]
```

numpy

Numerical operations are naively element-wise:

```
a = np.array([1, 2, 3, 4])
b = np.array([10, 20, 30, 40])
a + b
a * b
```

There are rules (called broadcasting) for how to extend arrays when they aren't the same size. (So be careful if that's not what you mean.)

numpy

```
a = np.arange(0, 60, 5).reshape(3, 4)
for x in np.nditer(a):
 print(x)

for x in np.nditer(a, order='C'):
 print(x)

for x in np.nditer(a, order='F'):
 print(x)

a.T
```

numpy

`reshape()` : change shape without changing data
`flat()` : a 1-D iterator over the array
`flatten()` : return a 1-D copy of the array

`concatenate()` : join arrays along existing axis
`stack()` : join arrays along new axis
`hstack()`
`vstack()`

`split()`
`hsplit()`
`vsplit()`

numpy

`+, −, /, ·`

`np.sin()`, etc.

`np.reciprocal()`
`np.power()`
`np.mod()`

`np.real()`, `np.imag()`, `np.conj()`, `np.angle()`

numpy

`np.amin(), np.amax()`

`np.ptp()`

`np.percentile()`

`np.median()`

`np.mean()`

`np.average()` : weighted average

`np.var()`

`np.sqrt(np.mean(np.abs(x - x.mean()) ** 2))`

$\Longleftrightarrow$ `math.sqrt(np.var())`

numpy

```
import numpy.matlib as npmat

npmat.empty()
npmat.zeros()
npmat.ones()
npmat.eye()),npmat.identity()
npmat.rand()
npmat.ones()
npmat.ones()
```

numpy

```
import numpy.linalg as npl
```

```
npl.dot() npl.vdot() npl.inner() npl.matmul()
npl.determinant() npl.solve() npl.inv()
```

scipy

Wrappers on top of classic fortran libraries (LAPACK, etc.).
Built on top of `numpy` data multi-dimensional arrays.

```
import scipy
import scipy.cluster
```
, etc.

- `scipy.cluster` : vector quantisation, *k*-means
- `scipy.constants`
- `scipy.fftpack`
- `scipy.integrate`
- `scipy.interpolation`
- `scipy.io`

scipy

- `scipy.linalg`
- `scipy.ndimage`
- `scipy.odr` : orthogonal distance regression
- `scipy.optimze`
- `scipy.signal`
- `scipy.sparse`

scipy

- `scipy.spatial`
- `scipy.special` : special mathematical functions
- `scipy.stats`

Retours

- Qu’est-ce qui vous a plu ?
- Qu’est-ce qui vous a manqué ?

Questions?