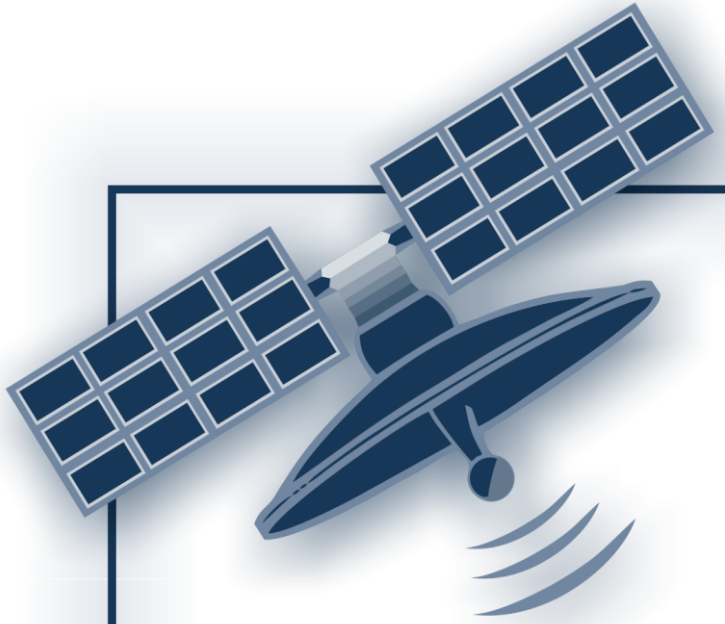


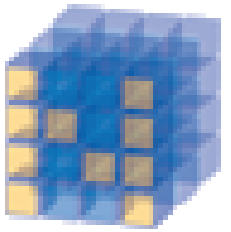
Introdução à Programação para Sensoriamento Remoto

Aula 10 – Introdução ao NumPy

Gilberto Ribeiro de Queiroz
Thales Sehn Körting
Fabiano Morelli

16 de Abril de 2018

NumPy - descrição



- http://www.numpy.org/
- NumPy is the fundamental package for scientific computing with Python. It contains (*among other things*):
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data (...)

O básico sobre NumPy

- O principal objeto manipulado pela NumPy é o `homogeneous multidimensional array`
  - Uma tabela de elementos de mesmo tipo (em geral números) indexados por uma tupla de inteiros positivos
  - As dimensões são chamadas de `axes`
  - O número de `axes` é chamado de `rank`

```
exemplo 1:

[1, 2, 1] → array com rank = 1

exemplo 2:

[[ 1.0, 0.0, 0.0],
 [0.0, 1.0, 2.0]] → rank = 2, a primeira dimensão
                    tem tamanho 2, a segunda
                    dimensão tem tamanho 3
```

O básico sobre NumPy

- A classe de `array` do NumPy é chamada `ndarray` (`numpy.array`)
- Supondo um `ndarray` contendo uma matriz de 30 linhas e 40 colunas:
  - `ndarray.ndim` é o número de `axes` da `array`
  - `ndarray.shape` é uma tupla de inteiros indicando o tamanho da `array` em cada dimensão (no exemplo acima, `shape` → 30, 40)
  - `ndarray.size` é o total de elementos no `array` (no exemplo acima, `size` → `1200`)
  - `ndarray.dtype` é um objeto descrevendo o tipo dos elementos no `array` (por exemplo, `numpy.int32`, `numpy.int16`, `numpy.float64`)
  - `ndarray.itemsize` é o tamanho em bytes de cada elemento no `array` (por exemplo, em um `array` de float64 itemsize → 8)
  - `ndarray.data` é um `buffer` contendo os elementos do `array`, geralmente não é usado dessa forma pois temos facilidades de indexação

Utilizando NumPy

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
```

```
>>> a.itemsize
8
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

Criando `arrays`

```
>>> import numpy as np

>>> a = np.array([2,3,4])

>>> a

array([2, 3, 4])

>>> a.dtype

dtype('int64')

>>> b = np.array([1.2, 3.5, 5.1])

>>> b.dtype

dtype('float64')
```

```
# transformação automática de sequências de sequências em 2D-array

>>> b = np.array([(1.5,2,3), (4,5,6)])

>>> b

array([[ 1.5, 2. , 3. ],
       [ 4. , 5. , 6. ]])
```

Criando `arrays`

```
# as funções zeros/ones criam
arrays contendo zeros/uns

>>> np.zeros( (3,4) )

array([[ 0.,  0.,  0.,  0.],

       [ 0.,  0.,  0.,  0.],

       [ 0.,  0.,  0.,  0.]])

>>> np.ones( (2,3) )

array([[ 1.,  1.,  1.],

       [ 1.,  1.,  1.]])
```

```
# a função arange cria números em
sequência

# iniciando em 10

# finalizando em 30

# pulando de 5 em 5

>>> np.arange( 10, 30, 5 )

array([10, 15, 20, 25])


# de 0 a 2, pulando de 0.3 em 0.3

>>> np.arange( 0, 2, 0.3 )

array([ 0. ,  0.3,  0.6,  0.9,
1.2,  1.5,  1.8])
```

Criando `arrays`

```
# a função linspace gera um intervalo de números com um tamanho específico
>>> from numpy import pi
# iniciando em 0
# finalizando em 2
# tamanho 9
>>> np.linspace( 0, 2, 9 )
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ])
# 100 números entre 0 e 2pi
>>> x = np.linspace( 0, 2*pi, 100 )
# seno de todos os números, calculados de uma só vez
>>> f = np.sin(x)
```

Operações básicas com `arrays`

```
# as operações aritméticas se aplicam sobre cada elemento do array, gerando um novo array de resultado

>>> a = np.array( [20,30,40,50] )

>>> b = np.arange( 4 )

>>> b

array([0, 1, 2, 3])

>>> c = a-b

>>> c

array([20, 29, 38, 47])
```

```
>>> b**2

array([0, 1, 4, 9])

>>> 10*np.sin(a)

array([ 9.12945251, -9.88031624,
7.4511316 , -2.62374854])

>>> a<35

array([ True, True, False, False])
```

Operações básicas com `arrays`

```
# o operador * calcula o produto por elemento, e não por matriz (para isso usa-se a função dot)

>>> A = np.array( [[1,1],
                   [0,1]] )

>>> B = np.array( [[2,0],
                   [3,4]] )

>>> A*B

array([[2, 0],
       [0, 4]])

>>> A.dot(B)

array([[5, 4],
       [3, 4]])
```

```
# operações como += e *= modificam a própria matriz

>>> a = np.ones((2,3))

>>> a *= 3 # é o mesmo que a = a * 3

>>> a

array([[3, 3, 3],
       [3, 3, 3]])

>>> b = np.random.random((2,3))

>>> b += a

>>> b

array([[ 3.417, 3.720, 3.001],
       [ 3.302, 3.146, 3.092]])
```

Operações básicas com `arrays`

```
# as funções sum, min e max funcionam para ndarrays

>>> a = np.random.random((2,3))

>>> a

array([[ 0.186, 0.345, 0.396],

       [ 0.538, 0.419, 0.685]])

>>> a.sum()

2.571

>>> a.min()

0.186

>>> a.max()

0.685
```

```
# a função reshape altera a estrutura do ndarray

>>> b = np.arange(12).reshape(3,4)

>>> b

array([[ 0,  1,  2,  3],

       [ 4,  5,  6,  7],

       [ 8,  9, 10, 11]])

>>> b.sum(axis=0)

array([12, 15, 18, 21])

>>> b.min(axis=1)

array([0, 4, 8])

# soma cumulativa

>>> b.cumsum(axis=1)

array([[ 0,  1,  3,  6],

       [ 4,  9, 15, 22],

       [ 8, 17, 27, 38]])
```

Referências Bibliográficas

- NumPy. Acesso: Abril, 2018.