INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL, IP

PPT12– Fundamentos de Python (UFCD 10793)

Sandra Liliana Meira de Oliveira

NumPy

NumPy

- `NumPy` é uma biblioteca Python que contém tipos para representar vetores e matrizes juntamente com suporte para diversas operações, entre elas operações comuns de álgebra linear e transformadas de Fourier.
- `NumPy` foi desenvolvida para obter uma maior eficiência do código em Python para aplicações científicas.

WHY NUMPY

Por que usar o NumPy?

Em Python temos listas que servem ao propósito de arrays, mas são lentas para processar.

O NumPy visa fornecer um objeto array que é até 50x mais rápido que as listas tradicionais do Python.

WHY NUMPY

Por que o NumPy é mais rápido que o Lists?

As matrizes NumPy são armazenadas em um local contínuo na memória, diferente das listas, para que os processos possam acessá-las e manipulá-las com muita eficiência.

Esse comportamento é chamado de localidade de referência em ciência da computação.

Esta é a principal razão pela qual o NumPy é mais rápido que as listas. Também é otimizado para trabalhar com as mais recentes arquiteturas de CPU.

Em qual idioma o NumPy está escrito?

NumPy é uma biblioteca Python e é escrita parcialmente em Python, mas a maioria das partes que requerem computação rápida são escritas em C ou C++.

Onde está a base de código NumPy?

O código-fonte do NumPy está localizado neste repositório do github https://github.com/numpy/numpy

https://github.com/numpy/numpy

NumPy

- Para usar esta biblioteca deve-se importá-la com o comando `import numpy`

Array

Array

- O objeto mais simples da biblioteca é o `array` que serve para criar objetos homogêneos multidimensionais - MATRIZES.
- Um `array` pode ser criado a partir de uma lista:

```
1  import numpy
2  obj = numpy. array([1, 2, 3])
3  print(type(obj))
4  # <class 'numpy. ndarray'>
5  print(obj)
6  # [1 2 3]
7  print(obj.ndim)
8  # 1
9  print(obj.size)
10 # 3
```

- Neste exemplo, usamos a biblioteca `NumPy` para criar um `array` de dimensão 1 com tamanho 3.

Array

- Um `array` pode ser criado a partir de um objeto multidimensional:

```
import numpy
obj = numpy. array([[1 , 2, 3], [4, 5, 6]])
print(obj)
# [[1 2  3]
#  [4 5  6]]
print(obj.ndim)
# 2
print(obj.size)
# 6
print(obj.shape)
# (2, 3)
```

- Neste exemplo, usamos a biblioteca `NumPy` para criar um `array` de dimensão 2 com tamanho 6.

Método arange

- Um `array` também pode ser criado utilizando o método `arange`.
- O método `arange` cria um `array` com apenas uma dimensão (lista), similar a função `range`.
- Exemplo:

```
1  import numpy
2  obj =numpy. arange(6)
3  print(obj)
4  # [0 1 2 3 4 5]
5  obj = numpy. arange(1, 6)
6  print(obj)
7  # [1 2 3 4 5]
8  obj = numpy. arange(1, 6, 2)
9  print(obj)
10 # [1 3 5]
```

Método `reshape`

- Objetos do tipo `array` podem ter suas dimensões modificadas utilizando o método `reshape`.
- Esse método recebe como parâmetros os tamanhos das dimensões desejadas.
- Exemplo:

```
1  import numpy
2  obj = numpy. arange(6)
3  print(obj)
4  # [0 1 2 3 4 5]
5  print(obj.reshape(2, 3))
6  # [[0 1 2]
7  #  [3 4 5]]
8  print(obj.reshape(3, 2))
9  # [[0 1]
10 #  [2 3]
11 #  [4 5]]
```

Método zeros

- `NumPy` possui o método `zeros` que cria um `array` contendo apenas zeros. O parâmetro desse método é uma tupla com os tamanhos de cada dimensão.

```
1 import numpy
2 print(numpy.zeros((3)))
3 # [0. 0. 0.]
4 print(numpy.zeros((3, 4)))
5 # [[0. 0. 0. 0.]
6 #  [0. 0. 0. 0.]
7 #  [0. 0. 0. 0.]]
```

Método ones

- `NumPy` possui também um método `ones` que cria um `array` inicializado todos os elementos com valor 1.

```
1 import numpy
2 print(numpy.ones((2, 5)))
3 # [[1. 1. 1. 1. 1.]
4 #  [1. 1. 1. 1.  1.]]
```

Método `full`

- Além dos métodos `zeros` e `ones`, o `NumPy` possui o método `full`, que cria um `array` contendo apenas o valor indicado.
- Neste método, o primeiro parâmetro é um tuplo com os tamanhos de cada dimensão e o segundo parâmetro é o número utilizado para criar o `array`.

```
1 import numpy
2 print(numpy.full((3, 2), 20.0))
3 # [[20. 20.]
4 #  [20. 20.]
5 #  [20. 20.]]
6 print(numpy.full((2, 3), 1/3))
7 # [[0.33333333 0.33333333 0.33333333]
8 #  [0.33333333 0.33333333 0.33333333]]
```

Aceder ao `Array`

Acessando os Elementos do `Array`

- Para acedermos aos elementos de um `array`, podemos indicar tanto a posição específica quanto uma fatia (`slice`).
- Os `slices` podem ser acedidos com a mesma sintaxe utilizada em listas em Python (através do caractere :).

Acessando as Linhas do `Array`

- Exemplo:

```
1 import  numpy
2 A = numpy. array([[10 , 9, 8], [1, 2, 3]])
3 print(A[0,  :])
4 #  [10   9   8]
5 print(A[1,  :])
6 #  [1 2  3]
```

Acessando as Colunas do `Array`

- Exemplo:

```
1 import  numpy
2 A = numpy. array([[10 , 9, 8], [1, 2, 3]])
3 print(A[:,  1:3])
4 # [[9  8]
5 #  [2  3]]
6 print(A[:,  0:2])
7 # [[10   9]
8 #  [ 1   2]]
```

Acessando um Elemento do `Array`

- Exemplo:

```
1 import  numpy
2 A = numpy. array([[10 , 9, 8], [1, 2, 3]])
3 print(A[0,  0])
4 # 10
5 print(A[0,  2])
6 # 8
7 print(A[1,  2])
8 # 3
```

Alterando Elementos do `Array`

- Exemplo:

```
import numpy
A = numpy. array([[10 , 9, 8], [1, 2, 3]])
A[1, 1] = 1000
print(A)
 # [[  10    9    8]
 #  [   1 1000    3]]
A[0, :] = [-1, -2, -3]
 # [[  -1   -2   -3]
 #  [   1 1000    3]]
```

Operadores para Arrays

Operadores para Arrays

- Os operadores `+`, `-`, `*`, `/`, `**`, `//` e `%` quando utilizados sobre `arrays`, são aplicados em cada posição dos mesmos.

```
1  import numpy
2  M1 = numpy. array([[1 , 2, 3], [4, 5, 6]])
3  M2 = numpy. array([[6 , 5, 4], [3, 2, 1]])
4  print(M1 + 2)
5  # [[3 4 5]
6  #  [6 7 8]]
7  print(M2 % 2)
8  # [[0 1 0]
9  #  [1 0 1]]
10 print(M2 * 2)
11 # [[12 10  8]
12 #  [ 6  4  2]]
```

Operadores para Arrays

- Os operadores `+`, `-`, `*`, `/`, `**`, `//` e, `%` quando utilizados sobre `arrays`, são aplicados em cada posição dos mesmos.

```
1  import numpy
2  M1 = numpy. array([[1 , 2, 3], [4, 5, 6]])
3  M2 = numpy. array([[6 , 5, 4], [3, 2, 1]])
4  print(M1 + M2)
5  # [[7 7 7]
6  #  [7 7 7]]
7  print(M1 // M2)
8  # [[0 0 0]
9  #  [1 2 6]]
10 print(M1 * M2)
11 # [[ 6 10 12]
12 #  [12 10  6]]
```

Operações de Comparação

Operadores de Comparação

- Os operadores de comparação ==, !=, >, >=, < e <=, quando utilizados sobre `arrays`, são aplicados em cada posição dos mesmos.
- Podemos aplicar as operações tanto `array` e número, quanto entre `arrays`

```
1  import numpy
2  M1 = numpy. array([[1 , 5, 7], [1, 2, 6]])
3  M2 = numpy. array([[6 , 5, 4], [3, 2, 1]])
4  print (M1 == 1)
5  # [[ True False False]
6  #  [ True False False]]
7  print(M1[M1 == 1])
8  # [1 1]
9  print (M2 > 5)
10 # [[ True False False]
11 #  [ False False False]]
12 print(M2[M2 > 5])
13 # [6]
```

Operadores de Comparação

- Os operadores de comparação ==, !=, >, >=, < e <=, quando utilizados sobre `arrays`, são aplicados em cada posição dos mesmos.
- Podemos aplicar as operações tanto `array` e número, quanto entre `arrays`

```
1  import numpy
2  M1 = numpy. array([[1 , 5, 7], [1, 2, 6]])
3  M2 = numpy. array([[6 , 5, 4], [3, 2, 1]])
4  print(M1 != M2)
5  # [[ True False  True]
6  #  [ True False  True]]
7  print(M1[M1 != M2])
8  # [1 7 1 6]
9  print(M1 <= M2)
10 # [[ True  True False]
11 #  [ True  True False]]
12 print(M2[M1 <= M2])
13 # [6 5 3 2]
```

Operações Algébricas

Método `transpose`

- `Arrays` possuem o método `transpose`, que retorna a matriz transposta.
- Exemplo:

```
1  import numpy
2  v = numpy. arange(1, 7)
3  m = v. reshape(2, 3)
4  print (m)
5  # [[1 2  3]
6  #  [4 5  6]]
7  t =m. transpose()
8  pr i nt (t)
9  # [[1  4]
10 #  [2 5]
11 #  [3  6]]
```

Método `dot`

- `Arrays` possuem o método `dot`, que calcula a multiplicação de matrizes.
- Como parâmetro, o método recebe outra matriz.
- Exemplo:

```
1  import numpy
2  A = numpy. arange(2, 6). reshape(2, 2)
3  B = numpy. arange(1, 5). reshape(2, 2)
4  print(A)
5  # [[2 3]
6  # [4 5]]
7  print(B)
8  # [[1 2]
9  #  [3 4]]
10 print(A.dot(B))
11 # [[11 16]
12 #  [19 28]]
```

Método `linalg.det`

- A biblioteca `numpy` possui o método `linalg.det`, que calcula a determinante.
- Como parâmetro, o método recebe um `array`.
- Exemplo:

```
1 import numpy
2 A = numpy. array([[1 , 9, 5], [3, 7, 8], [10 , 4, 2]])
3 B = numpy.array([[4, 2], [10, 4]])
4 print(numpy.linalg.det(A))
5 # 358.0
6 print(numpy.linalg.det(B))
7 # -4.0
```

Métodos Matemáticos

Método `sqrt`

- A biblioteca `numpy` possui o método `sqrt`, que calcula a raiz quadrada.
- Como parâmetro, o método recebe um `array`.
- Exemplo:

```
1 import numpy
2 print(numpy.sqrt([25, 16, 9, 4, 3]))
3  # [5.         4.         3.         2.         1.73205081]
4 print(numpy.sqrt([[1, 2, 3], [4, 5, 6]]))
5  # [[1.         1.41421356 1.73205081]
6  #  [2.         2.23606798 2.44948974]]
```

Método exp

- A biblioteca `numpy` possui o método `exp`, que calcula `e`, o número de Euller, elevado aos elementos do `array`.
- Como parâmetro, o método recebe um `array`.
- Exemplo:

```
1 import numpy
2 print(numpy.exp([2, 3, 4]))
3 # [ 7.3890561  20.08553692 54.59815003]
```

Cálculo de Logaritmos

- Na biblioteca `numpy`, existem diversos métodos para calcular logaritmos.
- Os métodos mais utilizadas são `log`, `log10` e `log2`, que calculam o logaritmo na base `e`, 10 e 2, respectivamente.
- Como parâmetro, todas eles recebem um `array`.
- Exemplo:

```
1 import numpy
2 print(numpy.log([1, 10, 100]))
3 # [0.          2.30258509 4.60517019]
4 print(numpy.log10([1, 10, 100]))
5 # [0. 1. 2.]
6 print(numpy.log2([1, 10, 100]))
7 # [0.          3.32192809 6.64385619]
```

Métodos para Conjuntos

Operações para Conjuntos

- Podemos aplicar métodos de conjuntos em `arrays`.
- Alguns exemplos de métodos são `unique`, que verifica os itens únicos de um ou mais `arrays`, `union1d`, que verifica a união dos `arrays`, `intersect1d`, que verifica a intersecção dos `arrays`, e `array_equal`, que verifica se dois `arrays` são iguais.

Método `unique`

- Exemplo:

```
1 import  numpy
2 A = numpy. array([1 , 2, 3, 3, 3, 2])
3 B = numpy. array([3, 4, 5])
4 print(numpy.unique(A))
5 # [1 2  3]
6 print(numpy.unique(B))
7 # [3 4  5]
```

Métodos `union1d` e `intersect1d`

- Exemplo:

```
1 import  numpy
2 A = numpy. array([1 , 2, 3, 3, 3, 2])
3 B = numpy. array([3 , 4, 5])
4 print( numpy.union1d(A, B))
5 # [1 2 3 4 5]
6 print( numpy.intersect1d(A, B))
7 # [3]
```

Método array_equal

- Exemplo:

```
1  import  numpy
2  A = numpy. array([1, 2, 3])
3  B = numpy. array([1, 2, 3])
4  C = numpy. array([1 , 3, 2])
5  print(numpy.array_equal(A, B))
6  # True  print(numpy.
7  array_equal(C, A))    # False
8  print(numpy.array_equal(B, C))
9  # False
10
```

Métodos Probabilísticos

Método `random.randint`

- Podemos sortear um número inteiro usando o método `random.randint`.
- O método recebe dois parâmetros, um valor para `low`, indicando o ínicio do intervalo, e um valor para `high`, indicando o final do intervalo.
- O valor em `low` é incluído no intervalo, enquanto o valor de `high` é excluído do intervalo.
- O método também pode receber apenas o valor de `low`. Neste caso, o sorteio será feito de 0 (incluído) até o valor passado como parâmetro (excluído).

Método `random.randint`

- Exemplo:

```
1 import numpy
2 # sorteando um número entre 0 e 9
3 print(numpy.random.randint(10))
4 # sorteando um número entre 5 e 9
5 print(numpy.random.randint(low=5, high=10))
6 # sorteando três números entre 0 e 9
7 print(numpy.random.randint(10, size=3))
8 # criando um array com 2 dimensões de tamanho 3
9 print(numpy.random.randint(10, size=(2, 3)))
```

Método `random.choice`

- Podemos sortear números de um `array` usando o método `random.choice`.
- Para selecionarmos se existirá ou não reposição, utilizamos o parâmetro `replace`.
- Por padrão, o parâmetro `replace` é `True`.
- Exemplo:

```
1 import numpy
2 # sorteando um número do array
3 print(numpy.random.choice([1, -2, 3, 1000, 5]))
4 # sorteando três números do array
5 print(numpy.random.choice([1, -2, 3, 1000, 5], size=3))
6 # criando um array com 2 dimensões de tamanho 2
7 print(numpy.random.choice([1, -2, 3, 1000, 5], size=(2, 2))
      )
8 # criando um array com 2 dimensões de tamanho 2 sem
      reposição
9 print(numpy.random.choice([1, -2, 3, 1000, 5], size=(2, 2),
       replace=False))
```

Método `random.uniform`

- Podemos sortear números reais seguindo uma distribuição uniforme.
- Para isto, podemos utilizar o método `random.uniform`.
- O valor mínimo e máximo do itervalo são definidos pelos parâmetros `low` e `high`, respectivamente.
- Exemplo:

```
1 import numpy
2 # sorteando um número seguindo a distribuição uniforme
3 print(numpy.random.uniform(low=0, high=10))
4 # sorteando três números seguindo a distribuição uniforme
5 print(numpy.random.uniform(low=0, high=10, size=3))
6 # criando um array com 2 dimensões de tamanho 3
7 print(numpy.random.uniform(low=0, high=10, size=(2, 3)))
```

Método `random.normal`

- Podemos sortear números reais seguindo uma distribuição normal.
- Para isto, podemos utilizar o método `random.normal`.
- O valor da média e do desvio padrão são definidos pelos parâmetros `loc` e `scale`, respectivamente.
- Exemplo:

```
1 import numpy
2 # sorteando um número seguindo a distribuição normal
3 print(numpy.random.normal(loc=0, scale=10))
4 # sorteando três números seguindo a distribuição normal
5 print(numpy.random.normal(loc=0, scale=10, size=3))
6 # criando um array com 2 dimensões de tamanho 3
7 print(numpy.random.normal(loc=0, scale=10, size=(2, 3)))
```

Métodos Estatísticos

Métodos estatísticos

- A partir de um array, podemos calcular diversas métodos estatísticos.
- Os métodos mais básicos são `mean`, para calcular a média, `min`, para obter o menor valor, `max`, para obter o maior valor, `median`, para calcular a mediana, `std`, para calcular o desvio padrão, e `var`, para calcular a variância.

Métodos estatísticos

- Exemplo:

```
1  import  numpy
2  A = numpy. array([3 , 2, 5, 7, 9, 1])
3  print(numpy.mean(A))
4  # 4.5
5  print(numpy.min(A))
6  # 1
7  print(numpy.max(A))
8  # 9
9  print(numpy.median(A))
10 # 4.0
11 print(numpy.std(A))
12 # 2.8136571693556887
13 print(numpy.var(A))
14 # 7.916666666666667
```

Concatenação

Concatenação

- Podemos concatenar dois ou mais `arrays` a partir do método `concatenate`.
- Para isto, indicamos os `arrays` que serão concatenados em um formato de lista.
- Além disto, devemos indicar em qual dimensão queremos concatenar os `arrays`, utilizando o parâmetro `axis`.
- Para a primeira dimensão, usamos o valor 0, para a segunda dimensão, usamos o valor 1, assim sucessivamente.

Concatenação

- Exemplo:

```
1  import numpy
2  A = numpy. array([[1 , 2, 3], [4, 5, 6]])
3  B = numpy. zeros((2 , 3))  print(numpy.
4  concatenate([A, B], axis=0))
5  #  [1. 2. 3.]
6  #  [4. 5. 6.]
7  #  [0. 0. 0.]
8  #  [0. 0. 0.]]
9  print(numpy.concatenate([A, B], axis=1))
10 # [[1. 2. 3. 0. 0. 0.]
11 #  [4. 5. 6. 0. 0. 0.]]
```

Documentação

NumPy

- Na biblioteca `NumPy` existe uma variedade de outras funções e métodos, por exemplo, para calcular autovalores e autovetores, para resolução de sistemas de equações lineares, etc.
- A biblioteca fornece uma documentação completa: `https://numpy.org/devdocs/reference`.