EAE1106 - Métodos Computacionais para Economia

Danilo Souza

Índice

1 Métodos Computacionais para Economia

Este é o material da disciplina **EAE1106 — Métodos Computacionais para Economia**, ministrada no Departamento de Economia da USP.

O curso tem como objetivo introduzir lógica de programação e análise de dados utilizando principalmente **Python**, com uma breve introdução a **R** ao final do semestre.

Parte I

Alfabetização computacional e lógica básica

2 Fundamentos de computação

In summary, this book has no content whatsoever.

Parte II

Computação numérica, análise de dados e visualização

3 Arrays, matrizes e álgebra linear

Uma das principais aplicações para o Python bla bla bla

3.1 O que é o NumPy?

NumPy (**Num**erical **Py**thon) é o pacote fundamental para computação científica em Python. Essa biblioteca é peça fundamental em outras bibliotecas igualmente importantes, como o Pandas e o Matplotlib. É uma biblioteca Python que tem como principal objeto o `ndarray`, um array multidimensional que guarda bastante semelhança com a ideia de vetores e matrizes, embora seja um objeto específico dentro da linguagem, com suas características e métodos próprios. O pacote contém também uma variedade de rotinas para operações rápidas em arrays, incluindo matemática, lógica, álgebra linear básica, operações estatísticas básicas e muito mais.

Mas o que é de fato um `ndarray`? É um objeto multidimensional que nos permite armazenar dados de forma sequencial e que podem ser acessados via indexação. Ué, mas isso é muito parecido com uma lista (ou um conjunto de listas). Qual a diferença então?

- NumPy arrays têm um tamanho fixo na criação, ao contrário das listas, que podem crescer. Alterar o tamanho de um ndarray criará um novo array e excluirá o original.
- Todos os elementos em um array devem ser do mesmo tipo de dados, diferentemente de listas, que são objetos mais genéricos. Isso facilita a gestão de memória e torna operações com esse tipo de objeto ordens de magnitude mais rápidas do que se utilizássemos listas.
- A maior velocidade e eficiência de armazenamento fazem do NumPy uma das bibliotecas mais utilizadas em aplicações matemáticas e científicas. Saber apenas as ferramentas nativas do Python, como listas, hoje já não é mais suficiente.

São muitas as qualidades do `NumPy` que fazem dele a melhor escolha quanto o assunto é lidar com objetos sequenciais, multidimensionais, e com os quais queremos operar tal qual vetores e matrizes. Mas chega de lenga lenga, vamos ao trabalho!

3.2 Elementos básicos do NumPy

Antes de tudo, é preciso importar o `NumPy`, já que se trata de uma biblioteca não nativa do Python.

```
import numpy as np
```

3.2.1 Arrays unidimensionais

Comecemos criando um `numpy.ndarray` do zero, contendo os números 1, 2 e 3. Podemos fazê-lo da seguinte forma:

```
a = np.array([1, 2, 3])
print(a)
print(type(a))
```

```
[1 2 3]
<class 'numpy.ndarray'>
```

A sintaxe é essa mesma: um par de colchetes dentro dos parênteses. Se tentarmos passar sem os colchetes, o Python retornará um erro.

```
a = np.array(1, 2, 3)
```

```
TypeError: array() takes from 1 to 2 positional arguments but 3 were given
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 a = np.array(1, 2, 3)
TypeError: array() takes from 1 to 2 positional arguments but 3 were given
```

Da mesma forma que uma sequência de 3 números inteiros, podemos criar um `numpy.ndarray` que repete 3 vezes o número 0 ou 3 vezes o número 1.

```
a = np.zeros(3)
b = np.ones(3)

print(a)
print(b)
```

```
[0. 0. 0.]
[1. 1. 1.]
```

Note que em ambos os casos os números aparecem com o ponto da casa decimal, o que é um indicativo de que estão armazenados como valores do tipo `float`. E se quiséssemos criar esses mesmos arrays, mas especificando que os números são inteiros, i.e., do tipo `int`?

```
a = np.zeros(3, dtype=int)
b = np.ones(3, dtype=int)

print(a)
print(b)
```

```
[0 0 0]
[1 1 1]
```

A função `numpy.linspace(x,y,z)` nos permite criar um array que vai de `x` até `y`, com `z` elementos igualmente espaçados.

```
a = np.linspace(0,8,5, dtype=int)

print(a)
```

```
[0 2 4 6 8]
```

Podemos acessar os elementos de um array qualquer utilizando a mesma ideia de indexação de listas:

```
print('Array a =',a)
print('\nPrimeiro elemento de a = ',a[0])
print('Segundo elemento de a = ',a[1])
print('Último elemento de a = ',a[-1])
print('Dois primeiros elementos de a = ',a[0:2])
```

```
Array a = [0 2 4 6 8]

Primeiro elemento de a =  0
Segundo elemento de a =  2
Último elemento de a =  8
Dois primeiros elementos de a =  [0 2]
```

3.2.2 Arrays multidimensionais

Como falamos anteriormente, um objeto do tipo `numpy.ndarray` é um array *n-dimensional* (por isso o `nd` em `ndarray`). Até agora trabalhamos apenas com uma dimensão, mas para as nossas aplicações é particularmente interessante o caso em que o número de dimensões é igual a 2, i.e., para o caso em que o array assume a forma de uma **matriz**.

Podemos criar esse tipo de array usando a mesma lógica de antes, com pequenas alterações na sintaxe:

```
A_22 = np.array([[1, 2], [3, 4]], dtype=int)

print('Matriz A =\n',A_22)
```

```
Matriz A =
 [[1 2]
 [3 4]]
```

O NumPy nos oferece algumas funções interessantes para criarmos matrizes específicas:

```
print('Matriz identidade 2x2 =\n',np.eye(2,dtype=int))
print('\nMatriz diagonal 3x3 =\n',np.diag([1,2,3]))
```

```
Matriz identidade 2x2 =
 [[1 0]
 [0 1]]

Matriz diagonal 3x3 =
 [[1 0 0]
 [0 2 0]
 [0 0 3]]
```

Podemos criar a matriz **transposta** de uma dada matriz utilizando a função `np.transpose()` (ou apenas o método `.T`):

```
print('Matriz A =\n',A_22)
print('\nTransposta da matriz A =\n',np.transpose(A_22))
print('\nTransposta da matriz A =\n',A_22.T)
```

```
Matriz A =
 [[1 2]
 [3 4]]

Transposta da matriz A =
 [[1 3]
 [2 4]]

Transposta da matriz A =
 [[1 3]
 [2 4]]
```

Assim como em arrays unidimensionais, podemos acessar os elementos de uma matriz utilizando indexação, mas agora em 2 dimensões:

```
A = np.array([[1,2,3], [4,5,6], [7,8,9]])

print('Matriz A =\n',A)
print('\nElemento 11 de A = ',A[0,0])
print('Elemento 23 de A = ',A[1,2])
print('Primeira linha de A = ',A[0,:])
print('Segunda coluna de A = ',A[:,1])
print('\nSubmatriz de A delimitada pelos elementos 22 e 33 =\n',A[1:,1:])
```

```
Matriz A =
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

Elemento 11 de A =  1
Elemento 23 de A =  6
Primeira linha de A =  [1 2 3]
Segunda coluna de A =  [2 5 8]

Submatriz de A delimitada pelos elementos 22 e 33 =
 [[5 6]
 [8 9]]
```

3.2.3 Propriedades de arrays

O NumPy fornece alguns métodos úteis que, apesar de não receberem nenhum argumento, nos permitem acessar algumas das características dos arrays que criamos.

- `ndim` retorna o número de dimensões do array;
- `shape` retorna o tamanho do array em cada uma de suas dimensões;
- `dtype` retorna o tipo de dado contido no array;
- `size` retorna o número total de elementos contidos no array.

Parte III

Temas complementares

4 Introdução à programação orientada a objetos (OOP)

In summary, this book has no content whatsoever.

5 Introdução ao R

In summary, this book has no content whatsoever.

References