# Introdução à linguagem Python

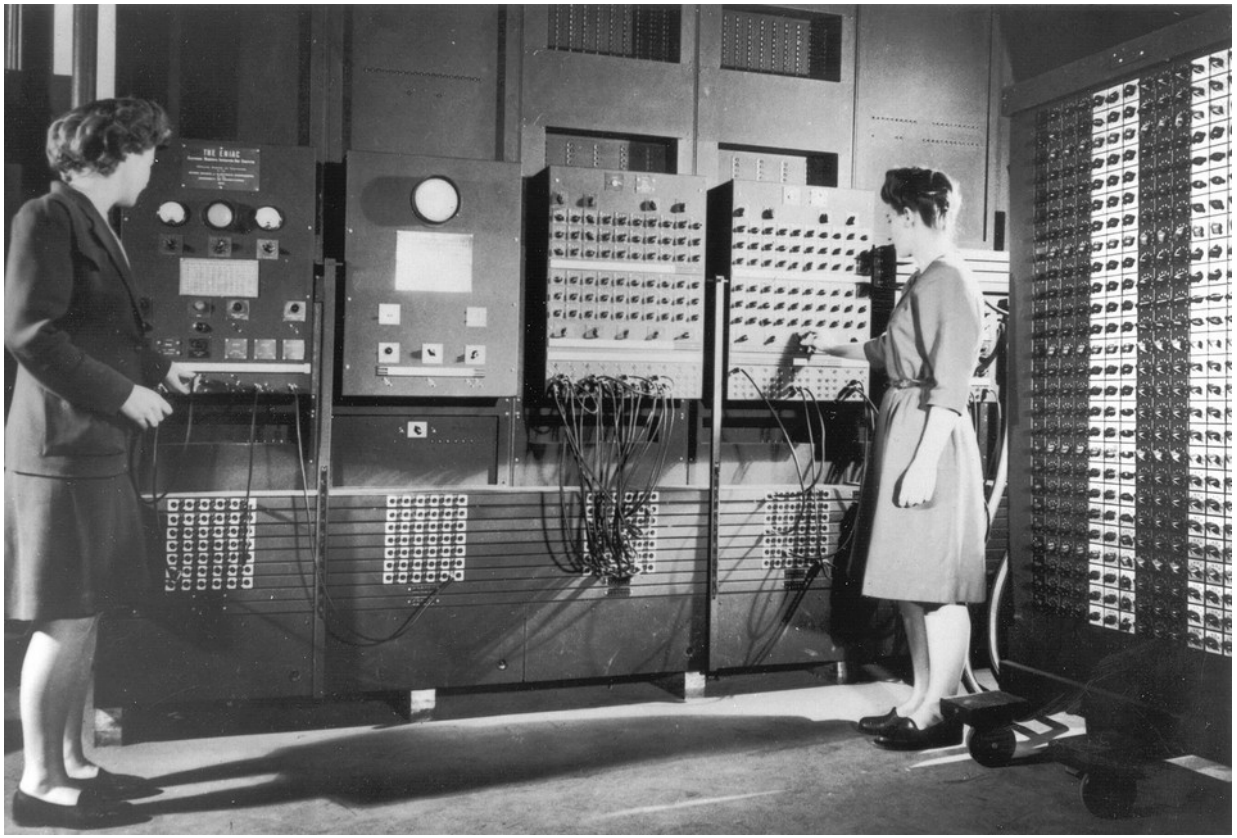
Thiago Martins

# As bibliotecas SciPY

- Um projeto *open source* de extensões Python para computação científica
- Apresenta-se como uma alternativa significativa a pacotes tradicionais de computação científica como o Matlab.
- Numpy
- Pandas
- Matplotlib

# Computação Científica

- ENIAC - 1946



# Computação Científica

- CDC - 6600

# Computação Científica

- Cray 1

  A revolução vetorial

# Numpy

- O “coração” de Scipy, uma biblioteca de álgebra linear para Python.
- Implementação de *alto desempenho* (em C e FORTRAN) de operações matemáticas sobre *tensores* densos.
  - Tensor é uma extensão n-dimensional de matrizes:

    Um escalar é um tensor de ordem zero, um vetor é um tensor de ordem 1, uma matriz é um tensor de ordem 2...
- Algoritmos clássicos, como soma, multiplicação de vetores.
- Algoritmos avançados como decomposição espectral, solução de sistemas lineares, FFT...
- https://numpy.org/

# Numpy

- `numpy.array`: O objeto fundamental de numpy
- O `numpy.array` é um armazenador de dados *homogêneos*, *multidimensional, prismático* e *denso*.
    - Homogêneos: todos os tipos primitivos são iguais (normalmente numéricos).
    - Multidimensional: Como em um tensor, os coeficientes se organizam ao longo de um número arbitrário de dimensões (*axes*). Um escalar tem zero dimensões, um vetor tem uma, uma matriz tem duas... E tipos de mais alta ordem são possíveis.
    - Prismático: Todos os sub-componentes tem sempre o mesmo tamanho ao longo de uma dimensão.
    - Denso: Não há otimização para objetos esparsos (como matrizes em que a maioria dos coeficiente é nula)

# `numpy.array`

- Criação:
  - Construtor básico a partir de tipos sequenciáveis limitados (listas, tuplas, etc.).
    - Se os elementos são por sua vez são tipos sequenciáveis *do mesmo tamanho*, uma nova dimensão é criada.
      - Isso acontece de forma recursiva. Note que os elementos em um dado nível devem ter todos o mesmo tamanho para que seja criada uma nova dimensão no objeto.

        `numpy.array([([1,2],range(2)),[[1,1],(2,1)]])`

        Note que o objeto é criado com uma combinação de listas, sequências e ranges.
  - `numpy.arange(start, stop, step)`:
    - Constrói um objeto unidimensional (um vetor) com sintaxe similar à de range: start é o início, stop é o limite superior, step é o incremento.
    -

# numpy.array

- Criação:
  - Construtor básico a partir de tipos sequenciáveis limitados (listas, tuplas, etc.).
    - Se os elementos são por sua vez são tipos sequenciáveis *do mesmo tamanho*, uma nova dimensão é criada.
      - Isso acontece de forma recursiva. Note que os elementos em um dado nível devem ter todos o mesmo tamanho para que seja criada uma nova dimensão no objeto.

        `numpy.array([([1,2],range(2)),[[1,1],(2,1)]])`

        Note que o objeto é criado com uma combinação de listas, sequências e ranges.
  - `numpy.arange(start, stop, step)`:
    - Constrói um objeto unidimensional (um vetor) com sintaxe similar à de range: start é o início, stop é o limite superior, step é o incremento.
  - `numpy.linspace(start, stop, n)`:
    - Constrói um objeto unidimensional (um vetor) com n números, de start a stop.

# numpy.array

- Criação:
  - Construtores com todos os coeficientes idênticos
    - `numpy.zeros(dim)`: Constrói um objeto com zeros seguindo as múltiplas dimensões passadas em dim.
    - `numpy.ones(dim)`: Constrói um objeto com uns seguindo as múltiplas dimensões passadas em dim.
    - `numpy.full(dim, val)`: Constrói um objeto com cópias de val seguindo as múltiplas dimensões passadas em dim.
  - Construtor com base em valores calculados:
    - `numpy.fromfunction(func, dim)`: Inicializa o objeto com o resultado de chamadas a func. Func recebe n parâmetros, um para cada dimensão do objeto.
  - Construtor sem inicialização:
    - `numpy.empty(dim)`: Constrói um objeto com coeficientes não-incializados. O parâmetro opcional dtype especifica o tipo de coeficiente (default float64).

# Numpy: Indexando, fatiando, iterando

- O operador [] retorna um coeficiente ou sub-objeto.
- Podem ser especificadas *fatias* do objeto com o operador ":"

```
numpy.fromfunction(lambda i, j, k: i*(2*3)+j*2+k, (4,3,2))
[1:3,1,1]
array([11., 17.])
```

- A sintaxe é similar a de um slice tradicional:

```
numpy.fromfunction(lambda i, j, k: i*(2*3)+j*2+k,(4,3,2))[1:,1,:2]
array([[ 8.,  9.],
       [14., 15.],
       [20., 21.]])
```

# Numpy: Indexando, fatiando, iterando

- Sintaxe completa de slices: `inicio: fim: incremento]`
- Quando valores são omitidos:

  `0: n : 1`

  Onde n é o tamanho da dimensão
- Ao contrário de sequências, Slices são *views* e não cópias

```
a = numpy.ones(6)
a[::2] = numpy.full(3, 2)
print(a)
```

```
[2. 1. 2. 1. 2. 1.]
```

# Numpy: Indexando, fatiando, iterando

- Sintaxe completa de slices: `inicio: fim: incremento]`
- Quando valores são omitidos:

  `0: n : 1`

  Onde n é o tamanho da dimensão
- Ao contrário de sequências, Slices são *views* e não cópias

```
a = numpy.ones(6)
a[::2] = numpy.full(3, 2)
print(a)
```

```
[2. 1. 2. 1. 2. 1.]
```

# Numpy: Indexando, fatiando, iterando

- Elípse (...) é o mesmo que um índice vazio (ou 0:n:1), seleciona todos os elementos naquele eixo
- Um índice None (ou numpy.newaxis) introduz um novo eixo de dimensão 1.

```
a = np.ones(3)
a[:,numpy.newaxis]
array([[1.],
       [1.],
       [1.]])
```

# Numpy: Indexando, fatiando, iterando

- Iterações são feitas sempre em relação à *primeira* dimensão.
  - Use slices para iterar sobre outras dimensões.
- O campo flat permite iterar sobre *todos* os coeficientes.
- A função transpose permite obter uma transposição dos eixos da matriz.
- A função reshape permite REDIMENSIONAR uma matriz, desde que o total de elementos seja o mesmo.
- Estas funções criam *views*!

# Numpy: Operações básicas

- Operações aritméticas são aplicadas coeficiente-a-coeficiente.

```
>>> numpy.array([1,2,3])*numpy.array([1,2,3])
    array([1, 4, 9])
```

  - Os objetos devem ter os *mesmos* tamanhos.
    - Se a dimensão dos objetos for diferente, os objetos são combinados a partir da seu *último eixo:*

```
>>> numpy.array([[1,1],[2,2],[3,3]])*numpy.array([1,0])
    array([[1, 0],
           [2, 0],
           [3, 0]])
```

# Numpy: Operações básicas

- Mais sobre regras de broadcast:

  Tensores são alinhados a partir do seu *último* eixo.

  Se os tensores têm uma quantidade distinta de eixos, o menor é completado com eixos com dimensão unitária.

  Todos os eixos de dimensão unitária são expandidos (copiando-se os elementos ao longo das novas dimensões) de forma a igualar-se ao correspondente eixo do outro tensor.

  Se após isso os tensores têm dimensões diferentes ao longo de algum eixo, a operação falha.

# Numpy: Operações básicas

- Mais sobre regras de broadcast:

```
(numpy.ones((2,2))+numpy.ones(2)).shape
    (2, 2)
```

| | | | |
|---|---|---|---|
| | | 2 | 2 |
| | | | 2 |
| | | 2 | 2 |

# Numpy: Operações básicas

- Mais sobre regras de broadcast:

```
(numpy.ones((2,3))+numpy.ones(2)).shape
```

```
ValueError: operands could not be broadcast
  together with shapes (2,3) (2,)
```

| | | | |
|---|---|---|---|
| | | 2 | 3 |
| | | | 2 |
| | | - | - |

# Numpy: Operações básicas

- Mais sobre regras de broadcast:

```
(numpy.ones((3,1,2))+np.ones((5,1,3,2))).shape
    (5, 3, 3, 2)
```

| | 3 | 1 | 2 |
|---|---|---|---|
| 5 | 1 | 3 | 2 |
| 5 | 3 | 3 | 2 |

# Numpy: Operações básicas

- Numpy tem versões especializadas de funções matemáticas que são aplicadas coeficiente a coeficente:

```
>>> numpy.sqrt(numpy.array([1,4,9,16,25]))
    array([1., 2., 3., 4., 5.])
```

# Numpy: Álgebra linear básica

- Percebe-se que o operador de produto é diferente da multiplicação de vetores ou a de matrizes x vetores!
- A operação de multiplicação é feita pelo operador @ ou pelo método dot.

```
>>> numpy.array([1,2,3]).dot(numpy.array([[1,1],[2,2],
[3,3]]))
    array([14, 14])
```

Os objetos devem ter dimensões compatíveis: A última dimensão do primeiro objeto deve ser igual à primeira do segundo e assim por diante.

# Numpy: Operações de redução

- sum(): Retorna o total dos coeficientes
- min(): retorna o menor coeficiente.
- max(): Retorna o maior coeficiente.
- mean(): Retorna a média dos coeficientes.
- ...
- Tipicamente a operação reduz o objeto inteiro. Mas é possível especificar os eixos sobre os quais se dá a redução:

```
>>> numpy.ones((3,2,4)).sum(axis=(0,1))
    array([6., 6., 6., 6.])
```



.

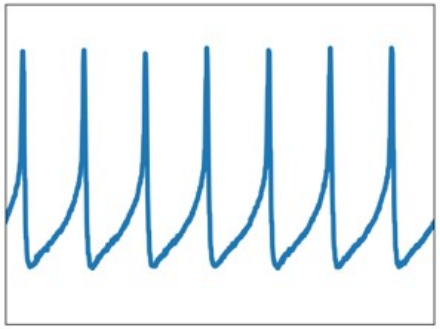
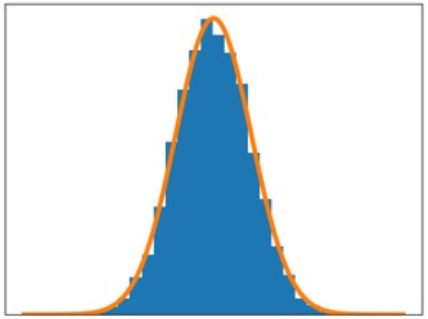
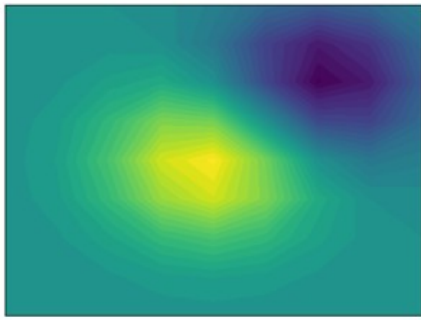
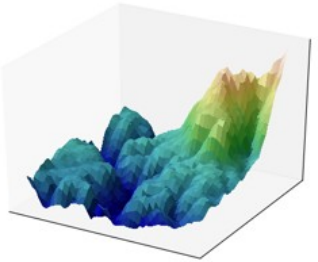
# Numpy: Álgebra linear avançada

numpy.linalg.inv(matriz): Inverte a matriz (geralmente mais pesado do que solve).

- numpy.linalg.solve(a, y): Acha a solução para o sistema ax=y
- np.linalg.eig(matriz): Retorna uma tupla cujo primeiro elemento é um vetor com os autovalores da matriz e o segundo é a matriz de autovetores.

# Matplotlib

- Biblioteca de visualização científica em Python.
-

- https://matplotlib.org/