SCC 124 - Introdução à Programação para Engenharias

Python para disciplinas básicas

Professor: André C. P. L. F. de Carvalho, ICMC-USP
Pos-doutorando: Isvani Frias-Blanco
Monitor: Henrique Bonini de Britto Menezes

Aula de hoje

- Integração da disciplina programação com o ciclo básico de Engenharia
- Computação científica
- Álgebra Linear
- Operações com matrizes
- Sistemas de equações

Introdução

- Python possui bibliotecas para várias áreas de conhecimento
  - Inclusive disciplinas do ciclo básico de Engenharia
- Propósito deste módulo do curso
  - Mostrar programas que implementam alguns conceitos a serem vistos no básico
    - Computação científica

Computação científica

- Utiliza linguagens de programação para escrita de programas para
  - Álgebra linear
  - Cálculo diferencial e integral
  - Cálculo numérico
  - Estatística
  - Gráficos
  - Otimização

Por meio de bibliotecas científicas (adds on)
NumPy, SciPy, Matplotlib

Álgebra Linear

- Área da matemática que estuda vetores, espaços vetoriais e equações lineares
  - Usada pela Geometria Analítica para calcular formatos geométricos complexos
  - Representa e manipula matrizes
  - Utilizada em várias operações matemáticas importantes para as Engenharias

Álgebra linear

- Componentes básicos
  - Valores escalares: 5
  - Vetores: $\begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$
  - Matrizes: $\begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 0 \end{bmatrix}$

Matrizes

- Uma matriz m por n (*m x n*) é uma tabela com m linhas e n colunas
  - Ex. Matriz 2x3

$$A = \begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 0 \end{bmatrix}_{2x3}$$

- Uma matriz 1 por n (ou n por 1) é um vetor
  - Ex. Vetor $\begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$

Tipos especiais de matrizes

$\begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 0 \end{bmatrix}$ Retangular

$\begin{bmatrix} 2 & 4 & 3 \\ -3 & 1 & 2 \\ 1 & 0 & 1 \end{bmatrix}$ Quadrada

$\begin{bmatrix} 2 & 4 & -3 \\ 4 & 1 & 2 \\ -3 & 2 & 1 \end{bmatrix}$ Simétrica

$\begin{bmatrix} 3 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 5 \end{bmatrix}$ Diagonal

$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ Identidade

Matrizes em Python

- Python tem uma biblioteca arrays nativa
  - Sequências de valores, parecidas com listas
  - Não implementa várias operações sobre matrizes
  - Para duas dimensões, matriz é simulada por um array de arrays (2D array)
    - Ex.: representa $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ por [[1,2][3,4]]
    - Para três dimensões, é usado um array de arrays de arrays
    - ...

Matrizes em Python

- Biblioteca arrays nativa de Python
  - Para manipular matrizes, necessário percorrer as listas
    - Problemas para manipular matrizes são mais complicados e lentos
- Biblioteca *numpy* tem duas classes com funções para criar e manipular matrizes
  - Classe numpy matrix
  - Classe numpy ndarray

Matriz x array

- Operações com matrizes são mais fáceis utilizando matrix
  - Multiplicação
    - Array: multiplica elementos correspondentes de dos arrays
    - Matrix: multiplica matriz por outra matriz
  - Outras operações
    - Matrix tem métodos específicos

Classe numpy ndarray

- Trabalha com N dimensões
  - Semelhante a listas
    - Mas todos os elementos devem ser do mesmo tipo
      - Em geral int ou float
  - Criação de um array pode ser feita com a função numpy.array
    - Retorna numpy.ndarray

Classe numpy ndarray

- Torna rápidas operações com grande quantidade de valores numéricos
  - Muito mais rápido que usar listas
- Grande número de operações matriciais
- Multiplicação de matrizes
  - Multiplica elemento a elemento
    - Ao invés de linha por coluna
  - Opção: método *matmul*

Matrizes em Python

- Tipo numpy array

```
import numpy as np
A = np.array (((1, 3, 2), (3, 4 ,0)))
B = np.array (((1, 0), (0, 1), (0, 1)))
C = np.array (((4), (2), (5)))
D = np.array ((1, 2, 4))
print ("Array A: \n", A)
print ("Array B: \n", B)
print ("Array C: \n", C)
print ("Array D: \n", D)
```


Matrizes em Python

- Tipo numpy array

```
import numpy as np
A = np.array (((1, 3, 2), (3, 4 ,0)))
B = np.array (((1, 0), (0, 1), (0, 1)))
C = np.array (((4), (2), (5)))
D = np.array ((1, 2, 4))
print ("Array A: \n", A)
print ("Array B: \n", B)
print ("Array C: \n", C)
print ("Array D: \n", D)
print ("Elemento de C: \n", C[0])
print ("Elemento de D: \n", D)
```

```
Array A:
 [[1 3 2]
 [3 4 0]]
Array B:
 [[1 0]
 [0 1]
 [0 1]]
Array C:
 [4 2 5]
Array D:
 [1 2 4]
```


Matrizes em Python

- Matrizes podem ser inicializadas de diferentes formas:

```
import numpy as np
A1 = np.array (((1, 3, 2), (3, 4 ,0)))
A2 = np. array ([[1, 3, 2], [3, 4 ,0]])
print ("Matriz A: \n", A1)
print ("Matriz A2: \n", A2)
```


Matrizes em Python

- Matrizes podem ser inicializadas de diferentes formas:

```
import numpy as np
A1 = np.array (((1, 3, 2), (3, 4 ,0)))
A2 = np. array ([[1, 3, 2], [3, 4 ,0]])
print ("Matriz A: \n", A1)
print ("Matriz A2: \n", A2)
```

```
Matriz A1:
 [[1 3 2]
 [3 4 0]]
Matriz A2:
 [[1 3 2]
 [3 4 0]]
```


Operações com arrays

```
import numpy as np
C = np.array ([[4], [2], [5]])
D = np.array ((1, 2, 4))
print ("C: ")
print(C)
print (C.shape)
print (C.ndim)
print (C.size)
print (C[0])
print ("D: ")
print(D)
print (D.shape)
print (D.ndim)
print (D.size)
print (D[1])
```

Operações com arrays

```
import numpy as np
C = np.array ([[4], [2], [5]])
D = np.array ((1, 2, 4))
print ("C: ")
print(C)
print (C.shape)
print (C.ndim)
print (C.size)
print (C[1,0])
print ("D: ")
print(D)
print (D.shape)
print (D.ndim)
print (D.size)
print (D[1])
```

```
C:
[[4]
 [2]
 [5]]
(3, 1)
2
3
2
D:
[1 2 4]
(3,)
1
3
2
```

Retorna tupla com apenas 1 elemento

Operações com arrays

```
import numpy as np
A = np.array(((1, 3, 2), (3, 4 ,0)))
B = np.array ((((1, 0), (0, 1), (0, 1)))
print ("A: ")
print(A)
print (A.shape)
print (A.ndim)
print (A.size)
print (A[1,2])
print ("B: ")
print(B)
print (B.shape)
print (B.ndim)
print (B.size)
print (B[1,1])
```


Operações com arrays

```
import numpy as np
A = np.array(((1, 3, 2), (3, 4 ,0)))
B = np.array ((((1, 0), (0, 1), (0, 1)))
print ("A: ")
print(A)
print (A.shape)
print (A.ndim)
print (A.size)
print (A[1,2])
print ("B: ")
print(B)
print (B.shape)
print (B.ndim)
print (B.size)
print (B[1,1])
```

```
A:
[[1 3 2]
 [3 4 0]]
(2, 3)
2
6
0
B:
[[1 0]
 [0 1]
 [0 1]]
(3, 2)
2
6
1
```


Função arrange de numpy

- Equivalente a função range de Python

```
>>> np.arange( 3, 20, 5 )
array([3, 8, 13, 18])
```

  - Aceita argumentos float

```
>>> np.arange( 1, 2, 0.2,)
array([1, 1.2, 11.4, 1.6, 1.8])
```


Operações com matrizes

- Multiplicação por escalar
- Soma
- Produto de matrizes
- Transposta de uma matriz
- Inversa de uma matriz
- Determinante de uma matriz
- Resolução de sistemas de equações

Multipicação de matriz por escalar

- $\lambda A = A\lambda$
- Supor:

$$\lambda = 2 \quad A = \begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 0 \end{bmatrix}$$

$$2A = \begin{bmatrix} 2 & 6 & 4 \\ 6 & 8 & 0 \end{bmatrix}$$

Multipicação de matriz por escalar

```
import numpy as np
A = np.array (((1, 3, 2), (3, 4 ,0)))
B = 2*A
C = A*2.5
print ("Matrix A: \n", A)
print ("Matrix B: \n", B)
print ("Matrix C: \n", C)
```


Multipicação de matriz por escalar

```
import numpy as np
A = np.array (((1, 3, 2), (3, 4 ,0)))
B = 2*A
C = A*2.5
print ("Matrix A: \n", A)
print ("Matrix B: \n", B)
print ("Matrix C: \n", C)
```

```
Matrix A:
 [[1 3 2]
 [3 4 0]]
Matrix B:
 [[2 6 4]
 [6 8 0]]
Matrix C:
 [[ 2.5  7.5  5. ]
 [ 7.5 10.   0. ]]
```


Soma de matrizes

- A + B = B + A
- Supor:

$$A = \begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 3 & -1 & 2 \\ 2 & 0 & 1 \end{bmatrix}$$

$$A + B = \begin{bmatrix} 4 & 2 & 4 \\ 5 & 4 & 1 \end{bmatrix}$$

Soma de matrizes

```
import numpy as np
A = np.array (((1, 3, 2), (3, 4 ,0)))
B = np.array ([[3, -1, 2], [2, 0 ,1]])
print ("Matriz A: \n", A)
print ("Matriz B: \n", B)
print ("A+B = \n", (A+B))
```

```
Matriz A:
 [[1 3 2]
 [3 4 0]]
Matriz B:
 [[ 3 -1  2]
 [ 2  0  1]]
A+B =
 [[4 2 4]
 [5 4 1]]
```


Multiplicação de matrizes

- A x B ≠ B x A
  - A_{mxn} x B_{nxk} = C_{mxk}

$$c_{ij} = \sum_{p=1}^{m} a_{ip} \times b_{pj}$$

- Supor:

$$A = \begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 3 & -1 \\ 2 & 1 \\ 5 & 2 \end{bmatrix} \quad 1x3+3x2+2x5$$

$$A \times B = \begin{bmatrix} 19 & 6 \\ 17 & 1 \end{bmatrix}$$

Multiplicação de matrizes

- Na classe array, multiplicação de matrizes multiplica os elementos correspondentes
  - Matrizes precisam ter as mesmas dimensões

$$A = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A * B = \begin{bmatrix} 4 \times 1 & 3 \times 2 \\ 2 \times 3 & 1 \times 4 \end{bmatrix} = \begin{bmatrix} 4 & 6 \\ 6 & 4 \end{bmatrix}$$

Multiplicação de matrizes

```
import numpy as np
A = np.array ([[1, 2],[3, 4]])
B = np.array ([[2, -1],[0, 2]])
print ("Matriz A: \n", A)
print ("Matriz B: \n", B)
print ("Matriz AxB: \n", A*B)
```


Multiplicação de matrizes

```
import numpy as np
A = np.array ([[1, 2],[3, 4]])
B = np.array ([[2, -1],[0, 2]])
print ("Matriz A: \n", A)
print ("Matriz B: \n", B)
print ("Matriz AxB: \n", A*B)
```

```
Matriz A:
 [[1 2]
 [3 4]]
Matriz B:
 [[ 2 -1]
 [ 0  2]]
Matriz AxB:
 [[ 2 -2]
 [ 0  8]]
```


Multiplicação de matrizes

```
import numpy as np
A = np.array ([[1, 3, 2], [3, 4, 0]])
B = np.array ([[3, -1], [2, 1], [5, 2]])
C = np.matmul(A, B)
print ("Matriz A: \n", A)
print ("Matriz B: \n", B)
print ("Matriz AxB: \n", C)
```

Método *matmul()* permite multiplicar matrizes do tipo array

Multiplicação de matrizes

```
import numpy as np
A = np.array ([[1, 3, 2], [3, 4, 0]])
B = np.array ([[3, -1], [2, 1], [5, 2]])
C = np.matmul(A, B)
print ("Matriz A: \n", A)
print ("Matriz B: \n", B)
print ("Matriz AxB: \n", C)
```

Método *matmul()* permite multiplicar matrizes do tipo array

```
Matriz A:
 [[1 3 2]
 [3 4 0]]
Matriz B:
 [[ 3 -1]
 [ 2  1]
 [ 5  2]]
AxB:
 [[19  6]
 [17  1]]
```


Transposta de uma matriz

- Inverte linha com coluna

$$a = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} \qquad a^T = \begin{bmatrix} 1 & 2 & 4 \end{bmatrix} \qquad (a^T)^T = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 3 & 2 \\ 3 & 4 & 0 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 3 \\ 3 & 4 \\ 2 & 0 \end{bmatrix}$$

Transposta de uma matriz

```
import numpy as np
A = np.array ([[1, 3, 2], [3, 4, 0]])
print ("Matriz A: \n", A)
print ("Tansposta de A: \n", np.transpose(A))
```

Transposta de uma matriz

```
import numpy as np
A = np.array ([[1, 3, 2], [3, 4, 0]])
print ("Matriz A: \n", A)
print ("Tansposta de A: \n", np.transpose(A))
```

```
Matriz A:
 [[1 3 2]
 [3 4 0]]
Tansposta de A:
 [[1 3]
 [3 4]
 [2 0]]
```


Determinante de uma matriz

- Outra operação comum é o cálculo do determinante de uma matriz
  - Matriz deve ser quadrada
- O determinante é uma propriedade escalar de uma matriz
- Permite ver se uma matriz tem uma inversa

Autovalores e autovetores

- Um vetor de n-dimensões pode ser visto uma direção em um espaço n-dimensional
- Autovetores
  - Vetores *x* que não mudam de direção quando multiplicados por uma matriz A
- Autovalores
  - Indicam se o vetor aumento ou reduziu seu tamanho e se manteve ou mudou seu sentido ao ser multiplicado por A

Determinante, autovetores e autovalores

```
import numpy as np
A = np.array (((1, 3, 2), (3, 4 ,0), (1, 2, 3)))
determinante = np.linalg.det(A)
autovals, autovecs =  np.linalg.eig(A)
print ("Matriz A \n",A)
print ("Determinante de A: \n", determinante)
print ("Autovetores de A: \n", autovecs)
print ("Autovalores de A: \n",autovals)
```

```
Matriz A
 [[1 3 2]
 [3 4 0]
 [1 2 3]]
Determinante de A:
 -11.0
Autovetores de A:
 [[-0.54579703 -0.84419959  0.26706531]
 [-0.65179135  0.53277533 -0.45560193]
 [-0.52656778 -0.0589704   0.84917784]]
Autovalores de A:
 [ 6.51213995 -0.75359619  2.24145624]
```


Sistemas de equações

- Uma das principais operações de AL é resolver sistemas de equações lineares
  - Duas ou mais equações lineares com as mesmas variáveis
  - Encontrar, se possível, valores das variáveis

$$\begin{cases} x & +3y & +2z & = 1 \\ -3x & +y & +2z & = -2 \\ x & -2y & -4z & = 4 \end{cases}$$

Sistemas de equações

```
import numpy as np
A = np.array (((1, 3, 2), (3, 4 ,0), (1, 2, 3)))
print ("Matriz A \n",A)
b = np.array((1, -2, 4))
print ("Vetor b \n",b)
c  =np.linalg.solve(A, b)
print ("Solução de Ax=b é c = ",c)
```

Sistemas de equações

```
import numpy as np
A = np.array (((1, 3, 2), (3, 4 ,0), (1, 2, 3)))
print ("Matriz A \n",A)
b = np.array((1, -2, 4))
print ("Vetor b \n",b)
c  =np.linalg.solve(A, b)
print ("Solução de Ax=b é c = ",c)
```

```
Matriz A
 [[1 3 2]
 [3 4 0]
 [1 2 3]]
Vetor b
 [ 1 -2  4]
Solução de Ax=b é c =  [ 0.90909091 -1.18181818  1.81818182]
```


Conclusão

- Integração da displina programação com o ciclo básico de Engenharia
- Computação científica
- Álgebra Linear
- Operações com matrizes
- Sistemas de equações

Perguntas