MC-102 — Aula 16
Matrizes e Vetores Multidimensionais

Eduardo C. Xavier

Instituto de Computação – Unicamp

11 de Novembro de 2020

Roteiro

Matrizes e Vetores Multidimensionais

- Podemos usar listas de Python para representar Matrizes e Vetores Multidimensionais.
- Suponha por exemplo que devemos armazenar as notas de cada aluno em cada laboratório de MC102.
- Podemos criar 15 listas distintas (um para cada lab.) de tamanho 50 (tamanho da turma), onde cada lista representa as notas de um laboratório específico.
- Matrizes e Vetores Multidimensionais permitem fazer a mesma coisa mas com todas as informações sendo acessadas por um nome em comum (ao invés de 15 nomes distintos).

Declarando uma matriz com Listas

- Para criar uma matriz de dimensões $l \times c$ inicialmente zerada podemos utilizar compreensão de listas.
- Exemplo de uma matriz 3×4 inicialmente com zeros:

```
>>> mat = [ [0 for j in range(4)] for i in range(3)]
>>> mat
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

- Cada lista interna representa uma linha da matriz.

Exemplo de declaração de matriz

- Utilizando laços, exemplo de criação de matriz 3×4, com 0s:

```
mat = []
for i in range(3): #para cada linha de 0 até 2
    l = []     #linha começa vazia
    for j in range(4):  #para cada coluna de 0 até 3
        l.append(0) #preenche colunas da linha i
    mat.append(l) #adiciona linha na matriz
print(mat)
```

- Saída é:
  $[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]$

Exemplo de declaração de matriz

- Utilizando compreensão de listas, exemplo de criação de matriz 3×4 (posição (i, j) contém o valor de $i \cdot j$):

```
mat = [ [i*j for j in range(4)] for i in range(3)]
print(mat)
```

- Saída é:
  $[[0, 0, 0, 0], [0, 1, 2, 3], [0, 2, 4, 6]]]$

Acessando dados de uma Matriz

- Em qualquer lugar onde você usaria uma variável no seu programa, você pode usar um elemento específico de uma matriz da seguinte forma:

`nome_da_matriz` [**ind_linha**][**ind_coluna**]

onde **ind_linha** (respectivamente **ind_coluna**) é um índice inteiro especificando a linha (respectivamente coluna) a ser acessada.

- No exemplo abaixo é criada uma matriz 10×20 inicializada com 0s, e depois é atribuído o valor 67 para a posição $(5, 13)$ dela.

```
#cria matriz 10x20 toda com zeros
mat = [ [0 for j in range(20)] for i in range (10)]
mat[5][13] = 67
```

Acessando uma matriz

- Imprime elemento da posição $(2, 3)$ da matriz:

```
mat = [ [i*j for j in range(4)] for i in range(3)]
print(mat[2][3])
```

Saída:

6

- Acessa posição inválida $(2, 4)$ da matriz:

```
mat = [ [i*j for j in range(4)] for i in range(3)]
print(mat[2][4])
```

Saída:

```
IndexError: list index out of range
```

Declarando Vetores Multidimensionais

- Podemos criar vetores multi-dimensionais utilizando listas de listas como no caso bidimensional.
- Para criar um vetor de dimensões $d_1 \times d_2 \ldots \times d_l$ inicialmente vazio podemos utilizar compreensão de listas:

$$[[[[]\ \text{for}\ i_{l-1}\ \text{in}\ \text{range}(d_{l-1})]\ldots]\ \text{for}\ i_2\ \text{in}\ \text{range}(d_2)]\ \text{for}\ i_1\ \ \text{in}\ \text{range}(d_1)]$$

Exemplo de vetor $3 \times 4 \times 5$ inicialmente com zeros em todas as posições:

```
>>mat = [ [ [0 for j in range(5)] for j in range(4) ] for i in range(3) ]
[ [ [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0] ],
  [ [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0] ],
  [ [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0] ] ]
```

Declarando Vetores Multidimensional

- O exemplo abaixo cria um vetor de quais dimensões??

```
mat = [ [ [0 for k in range(2)] for j in range(4)]  for i in range(5)]
```

- O acesso abaixo é válido?

```
mat[4][3][1] = 10
```

Exemplo

Criar programas com operações básicas sobre matrizes quadradas:

- Soma de 2 matrizes com dimensões $n \times n$.
- Subtração de 2 matrizes com dimensões $n \times n$.

Exemplos com Matrizes

- Primeiramente vamos implementar funções para se fazer a leitura e a impressão de uma matriz:
- Função para leitura de uma matriz $n \times n$:

```
def leMatriz(n):
  '''Esta função cria uma matriz nxn lendo dados do teclado'''

  #cria matriz inicialmente com 0s
  m = [ [0 for j in range(n)] for i in range(n)]
  #le dados do teclado
  for i in range(n):
    for j in range(n):
      m[i][j] = float(input('Dado pos(%d,%d): ' %(i,j)))
  return m
```

Exemplos com Matrizes

- Função para impressão de uma matriz $n \times n$.

```
def imprimeMatriz(m):
  '''Esta função imprime uma matriz quadrada nxn'''
    n = len(m)
    for i in range(n):
      for j in range(n):
        print('%.2f ' %m[i][j], end='')
      print()
```

Exemplo: Soma de Matrizes

- Vamos implementar a função que soma duas matrizes quadradas.
- Para cada posição (i, j) fazemos

$$\text{m3}[i][j] = \text{m1}[i][j] + \text{m2}[i][j]$$

tal que o resultado da soma das matrizes estará em **m3**.

```
def soma2(m1, m2):
  '''Função que calcula soma de duas matrizes nxn'''
  if len(m1) != len(m2):
    return None
  n = len(m1)
  #cria matriz resposta inicialmente com 0s
  m3 = [ [0 for j in range(n)] for i in range(n)]
  for i in range(n):
    for j in range(n):
      m3[i][j] = m1[i][j] + m2[i][j]
  return m3
```

Exemplo: Soma de Matrizes

- Note que podemos fazer a soma com compreensão de listas:

```
def soma(m1, m2):
  '''Função que calcula soma de duas matrizes nxn'''
  if len(m1) != len(m2):
    return None
  n = len(m1)
  m3 = [ [m1[i][j]+m2[i][j] for j in range(n)] for i in range(n)]
  return m3
```

Exemplo: Soma de Matrizes

Podemos criar uma função **main** para ler as matrizes e imprimir a soma:

```
    print('Lendo Matriz 1 (3x3)')
    m1 = leMatriz(3)
    imprimeMatriz(m1)
    print('Lendo Matriz 2 (3x3)')
    m2 = leMatriz(3)
    imprimeMatriz(m2)
    print('Soma')
    m3 = soma(m1, m2)
    imprimeMatriz(m3)

main()
```

Exercícios

- Faça um programa para realizar operações com matrizes que tenha as seguintes funcionalidades:
  - Um menu para escolher a operação a ser realizada:
    1. Leitura de uma matriz_1.
    2. Leitura de uma matriz_2.
    3. Impressão da matriz_1 e matriz_2.
    4. Cálculo da soma de matriz_1 com matriz_2, e impressão do resultado.
    5. Cálculo da multiplicação de matriz_1 com matriz_2, e impressão do resultado.
    6. Cálculo da subtração de matriz_1 com matriz_2, e impressão do resultado.
    7. Impressão da transposta de matriz_1 e matriz_2.

Exercícios

Escreva um programa que leia todas as posições de uma matriz 10×10.
O programa deve então exibir o número de posições não nulas na matriz.

Exercícios

- Escreva um programa que lê todos os elementos de uma matriz 4×4 e mostra a matriz e a sua transposta na tela.

Matriz

$$\begin{bmatrix} 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 2 \end{bmatrix}$$

Transposta

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

Exercícios

- Escreva um programa leia uma matriz do teclado e então imprime os elementos com menor e maior frequência de ocorrência na matriz.

NumPy

- NumPy é uma biblioteca para Python que contém tipos para representar vetores e matrizes juntamente com diversas operações de algebra linear.
- NumPy é implementado para trazer maior eficiência do programa para aplicações científicas.

NumPy

- Arrays em numpy são implementados como vetores estáticos em C, por isso possuem operações de acesso a posições realizados eficientemente ao contrário de listas de Python.
- Por outro lado, após definido o tamanho de um vetor em numpy é custoso alterar suas dimensões ao contrário de listas em Python.

NumPy

- Primeiramente deve-se instalar o NumPy baixando-se o pacote de `http://www.numpy.org/`
- Para usar os itens deste pacote deve-se importá-lo inicialmente com o comando

  ```
  >>> import numpy
  ```

  ou alternativamente com

  ```
  >>> import numpy as np
  ```

NumPy

- O objeto mais simples da biblioteca é o **array** que serve para criar vetores homogêneos multi-dimensionais.
- Um **array** pode ser criado a partir de uma lista:

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> a
array([1, 2, 3])
>>> a.ndim
1
>>> a.size
3
```

- Neste exemplo criamos um array de dimensão 1 com 3 elementos.

NumPy

- Um **array** pode ser criado a partir de uma lista de mais do que uma dimensão:

```
>>> a = np.array([[1,2,3],[4,5,6]])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a.ndim
2
>>> a.size
6
```

- Neste exemplo criamos um array de dimensão 2 com 6 elementos no total.

NumPy

- Um **array** pode ser criado com mais do que uma dimensão utilizando as funções **arange** e **reshape**.

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a = np.arange(10).reshape(2,5)
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>>
```

- Neste exemplo criamos um array de dimensão 1 com tamanho 10 e depois outro bidimensional 2×5.

NumPy

- NumPy oferece a função **zeros** que cria um **array** contendo apenas zeros. Seu argumento de entrada é uma tupla com a quantidade de elementos em cada dimensão.

```
>>> np.zeros((3))
array([ 0.,  0.,  0.])
>>> np.zeros((3,4))
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>>
```

- Também existe a função **ones** que cria um **array** inicializado com uns.

```
>>> numpy.ones((2,5))
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])
>>>
```

NumPy

- Também podemos criar uma matriz com valores aleatórios:

```
>>> a = np.random.randint(100, size=(3, 3))
>>> a
array([[77, 36, 57],
       [ 2, 27, 63],
       [72, 87, 60]])
```

NumPy

- Podemos acessar posições específicas de um **array** ou fazer um **slice** da mesma forma como com listas:

```
>>> a = np.random.randint(100, size=(3, 3))
>>> a
array([[77, 36, 57],
       [ 2, 27, 63],
       [72, 87, 60]])
>>> a[0][1]
36
>>> a[0][-1]
57
>>> a[0][0:2]
array([77, 36])
```

NumPy

- Os operadores *, - , + , /, **, quando utilizados sob **arrays**, são aplicados em cada posição do **array**.

```
>>> m = np.ones((2,3))
>>> m+1
array([[ 2.,  2.,  2.],
       [ 2.,  2.,  2.]])
>>> m*4
array([[ 4.,  4.,  4.],
       [ 4.,  4.,  4.]])
>>> m = m +1
>>> m
array([[ 2.,  2.,  2.],
       [ 2.,  2.,  2.]])
>>> m**3
array([[ 8.,  8.,  8.],
       [ 8.,  8.,  8.]])
>>>
```

NumPy

- O **dot product** (produto escalar) é feito com **np.dot**.
- Para matrizes é a multiplicação usual de matrizes.

```
>>> import numpy as np
>>> a = np.arange(9).reshape(3,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> b = np.array([ [2,2,2], [2,2,2], [2,2,2] ])
>>> b
array([[2, 2, 2],
       [2, 2, 2],
       [2, 2, 2]])
>>> np.dot(a,b)
array([[ 6,  6,  6],
       [24, 24, 24],
       [42, 42, 42]])
>>> c = np.ones(3)
>>> c
array([ 1.,  1.,  1.])
>>> c = c.reshape(3,1)
>>> c
array([[ 1.],
       [ 1.],
       [ 1.]])
>>> np.dot(a,c)
array([[  3.],
       [ 12.],
       [ 21.]])
```

NumPy

- NumPy oferece operações de algebra linear no pacote **numpy.linalg**.
- Por exemplo, **inv** calcula a inversa de uma matriz (caso exista inversa).

```
>>> import numpy as np
>>> a = np.random.randint(100, size=(3, 3))
>>> a
array([[32,  9, 77],
       [95, 25, 87],
       [48,  8, 81]])
>>> b = np.linalg.inv(a)
>>> b
array([[-0.05772488,  0.00490814,  0.04960257],
       [ 0.15284715,  0.04795205, -0.1968032 ],
       [ 0.01911132, -0.00764453,  0.00238892]])
>>> np.dot(a,b)
array([[  1.00000000e+00,   1.11022302e-16,   0.00000000e+00],
       [  2.22044605e-16,   1.00000000e+00,   0.00000000e+00],
       [  2.22044605e-16,   0.00000000e+00,   1.00000000e+00]])
```

- Note erro de precisão numérica.

NumPy

- Outro exemplo de **inv** para calcular a inversa de uma matriz.

```
>>> a = np.array([ [2, 7, 2], [0,1, 3], [1,0,2]])
>>> a
array([[2, 7, 2],
       [0, 1, 3],
       [1, 0, 2]])
>>> b = np.linalg.inv(a)
>>> b
array([[ 0.08695652, -0.60869565,  0.82608696],
       [ 0.13043478,  0.08695652, -0.26086957],
       [-0.04347826,  0.30434783,  0.08695652]])
>>> np.dot(a,b)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

NumPy

- Na biblioteca **linalg** existe uma variedade de outras funções como aquelas para calcular autovalores e autovetores, resolução de um sistema de equações lineares, etc.
- Mais informações podem ser encontradas no tutorial do numpy: `https://docs.scipy.org/doc/numpy-dev/user/quickstart.html`