MC-102 — Aula 18
Matrizes e Vetores Multidimensionais

Instituto de Computação – Unicamp

16 de Setembro de 2015

Roteiro

Matrizes e Vetores Multidimensionais

- Matrizes e Vetores Multidimensionais são generalizações de vetores simples vistos anteriormente.
- Suponha por exemplo que devemos armazenar as notas de cada aluno em cada laboratório de MC102.
- Podemos alocar 15 vetores (um para cada lab.) de tamanho 50 (tamanho da turma), onde cada vetor 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 vazia podemos utilizar compreensão de listas.
- Exemplo de uma matriz 3×4 inicialmente vazia:

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

- Lembre-se que os indices de uma lista começam em 0.
- Note que cada lista interna representa uma linha da matriz, e seu tamanho pode ser 4 ou qualquer outro valor.

Exemplo de declaração de matriz

- Criar matriz 3×4 onde cada posição (i, j) contém o valor de $i \cdot j$.

Utilizando laços:

```
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(i*j) #preenche colunas da linha i
    mat.append(l)
print(mat)
```

Utilizando compreensão de listas:

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

Em ambos os casos a saída é:
$[[0, 0, 0, 0], [0, 1, 2, 3], [0, 2, 4, 6]]$

Acessando uma matriz

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

```
nome_da_matriz [<linha>] [<coluna>]
```

  Ex: `matriz [1][10]` — Refere-se a variável na 2^a linha e na 11^a coluna da matriz.

- **Lembre-se que, como a matriz está implementada com listas, a primeira posição em uma determinada dimensão começa no índice 0.**
- O acesso a posições inválidas causa um erro de execução.

Acessando uma matriz

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

```
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])
```

  IndexError: list index out of range

Declarando uma matriz de múltiplas dimensões

- 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 vazio

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

Declarando uma matriz de múltiplas dimensões

Exemplo de matriz $3 \times 4 \times 5$ inicialmente com zeros.

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

Exemplos com Matrizes

Lendo uma matriz 4×4 do teclado:

```
mat = [[] for i in range(4)]
for i in range(4):
    for j in range(4):
        aux = float(input("Num. da pos. ("+str(i+1)+","+str(j+1)+")"))
        mat[i].append(aux)
print(mat)
```

Exemplos com Matrizes

Lendo uma matriz 4×4 do teclado com compreensão de listas:

```
def LePosicao(i, j):
    aux = float(input("Num. da pos. ("+str(i+1)+","+str(j+1)+"):"))
    return aux

mat = [[LePosicao(i,j) for j in range(4)] for i in range(4)]
print(mat)
```

Exemplos com Matrizes

Escrevendo uma matriz 4×4 na tela:

```
mat = [[] for i in range(4)]
for i in range(4):
    for j in range(4):
        aux = float(input("Num. da pos. ("+str(i+1)+","+str(j+1)+")"))
        mat[i].append(aux)


for l in mat:
    for j in l:
        print(j, end=",")
    print()
```

Exemplo Maior

Criar uma aplicação com operações básicas sobre matrizes quadradas:

- Soma de 2 matrizes com dimensões $n \times n$.
- Subtração de 2 matrizes com dimensões $n \times n$.
- Cálculo da transposta de uma matriz de dimensão $n \times n$.
- Multiplicação de 2 matrizes com dimensões $n \times n$.

Exemplo Maior

Primeiramente criamos uma função para fazer a leitura de uma matriz:

```
def lePos(i, j):
    aux = float(input("Num. da pos. ("+str(i+1)+","+str(j+1)+"):"))
    return aux

def leMat(n):
    print("Lendo dados de Matriz "+str(n)+" x "+str(n))
    mat  = [[lePos(i,j) for j in range(n)] for i in range(n)]
    return mat
```

Exemplo Maior

Depois criamos uma função para fazer a impressão de uma matriz:

```
def imprimeMat(mat):
    for l in mat:
        for j in l:
            print(j, end=",")
        print()
```

Exemplo Maior

Dentre as funcionalidades, vamos implementar a multiplicação:

- Vamos multiplicar duas matrizes M_1 e M_2 (de dimensão $n \times n$).
- O resultado será uma terceira matriz M_3.
- Lembre-se que uma posição (i, j) de M_3 terá o produto interno do vetor linha i de M_1 com o vetor coluna j de M_2:

$$M_3[i, j] = \sum_{k=0}^{n-1} M_1[i, k] \cdot M_2[k, j]$$

Exemplo Maior

Em Python temos:

```
def multMat(mat1, mat2):
    n = len(mat1)
    mat3 = [ [0 for j in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                mat3[i][j] = mat3[i][j] + (mat1[i][k]*mat2[k][j])
    return mat3
```

Exemplo Maior

Um código usando as funções anteriores:

```
def lePos(i, j):
    aux = float(input("Num. da pos. ("+str(i+1)+","+str(j+1)+"):"))
    return aux

.
.
.

mat1 = leMat(3)
mat2 = leMat(3)
mat3 = multMat(mat1, mat2)
imprimeMat(mat3)
```

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

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

Exercício

- Complete o código do programa com operações de matrizes, para conter:
  - Um menu para escolher a operação a ser realizada.
  - Uma função para cada uma das operações faltantes.

NumPy

- NumPy é uma biblioteca para Python que contém tipos para representar vetores e matrizes juntamente com diversas operações, dentre elas operações comuns de algebra linear e transformadas de Fourier.
- NumPy é implementado para trazer maior eficiência do código em Python para aplicações científicas.

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
```

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:

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

- Neste exemplo criamos um array de dimensão 2 com tamanho 6.

NumPy

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

```
>>> a = 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 1 com tamanho 3.

NumPy

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

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

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

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

NumPy

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

```
>>> m = 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

- NumPy oferece operações de algebra linear no pacote **mumpy.linalg**.

```
>>> from numpy.linalg import *
>>> a = arange(9).reshape(3,3)
>>> b = array([ [2,2,2], [2,2,2], [2,2,2] ])
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> b
array([[2, 2, 2],
       [2, 2, 2],
       [2, 2, 2]])
>>> dot(a,b)
array([[ 6,  6,  6],
       [24, 24, 24],
       [42, 42, 42]])
>>> dot(b,a)
array([[18, 24, 30],
       [18, 24, 30],
       [18, 24, 30]])
```

- **dot** corresponde à multiplicação de matrizes.

NumPy

- **inv** calcula a inversa de uma matriz.

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

NumPy

- Na biblioteca existe uma variedade de outras função como funções para calcular autovalores e autovetores, resolução de um sistema de equações lineares, etc.