

Introdução às matrizes

Nesta seção examinaremos resumidamente como utilizar matrizes com a linguagem Python.

Da mesma forma que o conceito de *vetor* é útil quando necessita-se de uma grande quantidade de valores associados, todos eles, a um mesmo tipo de dado (como as várias notas de determinado aluno), o conceito de **matriz** é útil quando naturalmente os dados considerados apresentam duas dimensões. Um exemplo é quando precisamos processar dados de um grupo de indivíduos (e.g. alunos), cada um deles com várias medidas (e.g. notas - vide figura 1).

Também vale a pena destacar que, em várias linguagens de programação (como C e *Python*), a implementação de matriz é feita por meio de vetor de vetores. Desse modelo, em ambas pode-se tratar cada linha como um vetor. Por exemplo, se a matriz recebe o nome de *M*, então *M*[0] é sua primeira linha, *M*[1] a segunda linha e assim por diante. Se for necessário pegar um elemento específico da matriz, deve-se usar dois colchetes, como em *M*[*i*][*j*], que devolve o valor da linha *i* na coluna *j* (e.g. *M*[0][0] devolve o valor da primeira posição da matriz e *M*[1][0] devolve o valor do primeiro elemento sua segunda linha).

1. Matrizes: sequência contígua de variáveis

Do ponto de vista computacional a implementação de matrizes segue o princípio dos *vetores*, uma *matriz* ocupa uma porção contígua da memória do computador, servindo para representar o conceito matemático associado. Lembrando que ao representar um *vetor* o acesso ao elemento da posição *i* pode ser feito por meio da sintaxe *vet*[*i*], no caso de *matriz* é necessário indicar em qual linha *i* e em qual coluna *j* o elemento está, por exemplo, se a variável tem por nome *Mat*, usaria *Mat*[*i*][*j*].

Este tipo de estrutura de dados é natural quando os dados apresentam 2 atributos. Por exemplo, em uma sala de aula, o professor precisa manter informações dos resultados obtidos pelos alunos em várias atividades, assim pode-se utilizar uma matriz para representar estes dados: as atividades de cada aluno estão em uma única linha da matriz *Mat* (e.g., *Mat*[*i*][0] é o resultado da atividade 0 para o aluno *i*).

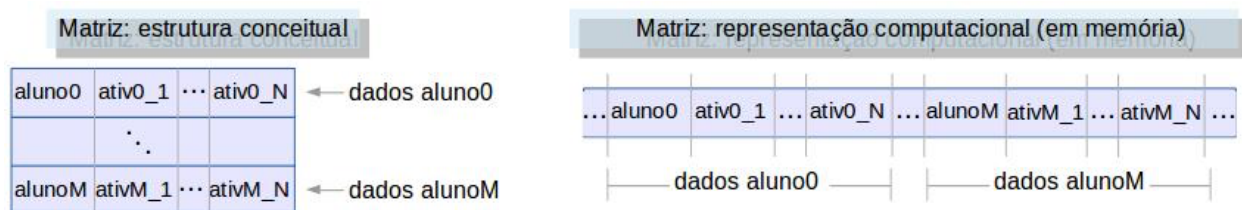


Fig. 1. Ilustração de como uma matriz é representada na "memória" do computador.

Mas como é possível implementar computacionalmente esse tipo de estrutura? Na figura acima está representado uma matriz conceitual, com a ideia de alunos e notas, e à direita como estes dados estão na memória do computador, supondo-se *M*+1 alunos e *N*+1 atividades. Se na representação computacional, o nome da matriz for *Mat*, então os dados correspondentes ao *aluno 0* são: *Mat*[0][1], *Mat*[0][1] e assim por diante até o *Mat*[0][*N*].

1.1. Matrizes: como parâmetro de função

Do mesmo modo que vetores, ao passar uma matriz como parâmetro de uma função, este funcionará como **parâmetro por referência** (ou por **endereço**). Ou seja, é passada uma referência ao *parâmetro efetivo* (em que chamou a função) de modo que qualquer alteração dentro da função, no *parâmetro formal*, significará que o valor na matriz que foi passada como parâmetro (no local que chamou a função e portanto o *parâmetro efetivo*) será alterado.

Na linguagem Python as matrizes (e vetores) são passados por referência. O exemplo abaixo ilustra que alterar os dados de uma matriz dentro de uma função implica em alterar a matriz que lhe foi passada (parâmetro efetivo).

Tab. 1. Matrizes passadas via parâmetro para funções (equivalente à uma referência à uma matriz "externa").

Função com computa soma de 2 matrizes em Python
Exemplo em Python
<pre>from __future__ import print_function; # imprime sem mudar linha Python 2 # Funcao para gerar nova matriz MA + MB def somaMat (MA, MB, nL, nC) : MC = []; # inicia matriz que sera devolvida linha = []; # inicia vetor para compor linha de MC for i in range(nL) : # i varia entre 0 e nL-1 for j in range(nC) : # j varia entre 0 e nC-1 linha.append(MA[i][j] + MB[i][j]); # novo elemento de "linha" MC.append(linha); # para conseguir estrutura adequada return MC; # Funcao para entrar matriz nL x nC, linha por linha def lerMatriz (mat, nL, nC) : for i in range(nL) : linha = []; for j in range(nC) : linha.append(input()); mat.append(linha); # Funcao para imprimir matriz nL x nC, linha por linha def imprimeMatriz (mat, nL, nC) : for i in range(0, nL) : # i entre 0 e nL-1 print("%2d: " % i, end=""); # "Imprime" numero desta linha for j in range(0, nC) : print("%3d " % mat[i][j], end=""); # em 3 espacos print(); # Mude de linha def main () : matA = []; matB = []; linha = raw_input(); # ler linha em Python 2 (no 3 usar apenas "input()") m, n = map(int, linha.split()); # quebrar entrada em 2 inteiros print("Matriz A:"); lerMatriz(matA, m, n); print("Matriz B:"); lerMatriz(matB, m, n); matC = somaMat(matA, matB, m, n); # parametros efetivos print("Matriz A:"); imprimeMatriz(matA, m, n); # 'matA' e' parametro efetivo print("Matriz B:"); imprimeMatriz(matB, m, n); # aqui e' 'matB' print("Matriz C:"); imprimeMatriz(matC, m, n); # aqui e' 'matC' main();</pre>

1.2. Matrizes: uma linha equivale a um vetor

Uma vez que os elementos em uma linha da matriz estão em posições contíguas da memória, eles podem ser olhados como um vetor, novamente o *contexto* determina o significado dos dados. Assim, se tivermos uma função *soma_vetor* que recebe como parâmetro um vetor (e sua dimensão) e que gera a soma de seus elementos, pode-se fazer a seguinte chamada: *soma_vetor(mat[i],n)*, para qualquer *i* entre 0 e *M* do exemplo acima.

Desse modo, o exemplo abaixo ilustra uma função preparada para somar elementos de um vetor (*soma += vet[i];*) sendo usada para somar as linhas de uma matriz. Então, na execução do laço dentro função, o comando usando o *parâmetro formal soma*, *soma += vet[i]* equivalerá ao código *soma += mat[k][i];* no trecho que invocou a função *soma_vetor*.

Tab. 2. Uma linha de um matriz é na verdade um vetor.

Função de função para somar elementos de vetor sendo usado com linhas de matrizes em Python
Exemplo em Python
<pre> from __future__ import print_function; # imprime sem mudar linha Python 2 def soma_vetor (vet, n) : soma = 0; for i in range(n) : soma += vet[i]; return soma; def main () : ... print("soma linha %d : %d\n" % (i, soma_vetor(mat[i], n))); ... main(); </pre>

3. Matrizes em Python

Em Python pode-se gerar listas de variadas maneiras, a forma sugerida abaixo é gerar uma lista de listas, sendo que a segunda lista será uma linha da matriz. No código abaixo ilustramos isso.

3.1. A linha de uma matriz é na verdade um vetor

O exemplo abaixo ilustra a declaração e uso de matrizes, com os tipos básicos *int*, *char* e *float*.

Tab. 4. Exemplos de tratamento de matrizes em Python (com inteiros, caracteres e "flutuantes").

Matrizes em Python		
Matriz de inteiros	Matriz de caracteres	Matriz de flutuantes
<pre> def main () : nL = int(input()); nC = int(input()); # "Ler" dimensoes nL, nC mat = []; # Matriz: iniciar uma lista vazia for i in range(0, nL) : linha = []; # iniciar lista vazia para a linha atual for j in range(0, nC) : # ler nI inteiros </pre>	<pre> def main () : nL = int(input()); nC = int(input()); # "Ler" dimensoes nL, nC mat = []; # Matriz: iniciar uma lista vazia for i in range(0, nL) : linha = []; # iniciar lista vazia para a linha atual for j in range(0, nC) : # ler nI inteiros </pre>	<pre> def main () : nL = int(input()); nC = int(input()); # "Ler" dimensoes nL, nC mat = []; # Matriz: iniciar uma lista vazia for i in range(0, nL) : linha = []; # iniciar lista vazia para a linha atual for j in range(0, nC) : # ler nI inteiros </pre>

<pre> linha.append(int(input())); # anexar mais um elemento mat.append(linha); # anexar a linha `a` matriz # Em Python para imprimir de modo mais "bonito" a matriz, # pode-se fazer impressao dos elementos da linha sem "quebre", # ou compondo uma "string". Neste exemplo usamos o segundo "truque". for i in range(0, nL) : strLinha = ""; # Truque: compor um "string" com a linha i toda for j in range(0, nC) : strLinha += "%3i" % mat[i][j]; # formatar ajustando 3 pos. print("%2i: " % i + strLinha); # "Imprimir" a linha i main(); </pre>	<pre> linha.append(raw_input()); # anexar mais um elemento mat.append(linha); # anexar a matriz a linha # Nesse exemplo imprimimos usando a biblioteca "print_function" # para nao "quebrar" linha entre 2 impressoes. for i in range(0, nL) : print("%2i: " % i, end=""); # parametro "end" impede a quebra for j in range(0, nC) : print("%4c" % mat[i][j], end=""); print(); # agora "quebre" a linha (final linha i)! main(); </pre>	<pre> linha.append(float(input())); # anexar mais um elemento mat.append(linha); # anexar a matriz a linha # Neste exemplo usamos novamente o "truque" de usar # "string" para imprimir cada linha da matriz for i in range(0, nL) : strLinha = ""; # Truque para imprimir "por linha" for j in range(0, nC) : strLinha += "%5.1f" % mat[i][j]; # usar 5 espacos, 1 pto dec. print("%2i: " % i + strLinha); # "Imprimir" esta linha linha = []; main(); </pre>
---	--	--

No segundo exemplo acima, utilizamos recurso da biblioteca `print_function`, assim no *Python 2* é obrigatório incluir esta biblioteca, como indicado nas primeiras linhas do exemplo abaixo. No caso do *Python 3* a inclusão é desnecessária.

O exemplo abaixo ilustra outro conceito importante: cada linha da matriz comporta-se como lista e portanto pode-se aplicar sobre cada linha da matriz, qualquer função que tenha como parâmetro formal uma lista. O exemplo será aquele da função *soma_vetor*, que soma os elementos de um vetor.

Exemplo de matriz em Python passando linha em funcao que trabalha com vetor

```

# Python2 para 'print' sem pular linha : print("*" , end = "");
from __future__ import print_function

```

```

def soma_vet (vet, n) :
    soma = 0;
    for i in range(0,n) :
        soma += vet[i];
    return soma;

```

```

mat = []; # declara que existira uma "lista" (que sera' lista de lista - ou
vetor de vetor)

```

```

m = 3; n = 4; # dimensoes fixas por simplicidade

```

```

for i in range(0,m) :
    linha = []; # declara um "lista" (ou vetor) que sera' um linha da matriz
    'mat[i][j]'
    for j in range(0,n) :
        linha.append(i*n + j); # equivale a fazer "mat[i][j] = i*m + j" (mas
        assim resulta erro em Python)
        print(" %3d" % linha[j], end=""); # imprime sem mudar de linha
    print(); # muda de linha

```

```
mat.append(linha); # define a linha i da matriz (algo como mat[i] = linha)

print("Imprime somas das linhas");
for i in range(0,m) :
    print("Linha %d tem soma: %3d" % (i, soma_vet(mat[i], n))); # %3d ajusta 3
casas 'a direita
```

Muita atenção ao método usado para construir a matriz, usando uma lista auxiliar `linha` e a necessidade de anexar cada linha separadamente (`mat.append(linha)`). Se isso não está suficientemente claro, sugiro que peguem [este exemplo](#) e "brinquem" com ele, até entendê-lo bem.

Bem, em resumo é isso.

[Leônidas de Oliveira Brandão](#)
<http://line.ime.usp.br>

Alterações 