BCM0505-15 - Processamento da Informação

May 19, 2020

Laboratório L05 – 19/05

Instruções

- Membros das turmas NA1 e NB{4,5} deverão submeter códigos em Python para todos os exercícios nesta página em um único arquivo `.py` via Tidia até 26/05 às 19h – sob a entrada **E9.**

Exercícios

- **(01)** *Vetorização de Matrizes.*
  Seja $A : [0\,..\,m-1] \times [0\,..\,n-1] \to \mathbb{R}$ uma matriz de reais com m linhas e n colunas. A *vetorização* de A, denotada por $\mathcal{V}ec(A)$ é a imersão de A em uma lista de dimensão mn, em que as linhas de A ocorrem uma após a outra. Exemplo:

$$A = \begin{pmatrix} 2 & 1 & 0 & 4 \\ 3 & 0 & 2 & -1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad \text{e} \qquad \mathcal{V}ec(A) = [2, 1, 0, 4, 3, 0, 2, -1, 0, 1, 0, 1].$$

  Escreva uma função que recebe A e devolve $\mathcal{V}ec(A)$.

  Solução:
  Uma versão *feijão-com-arroz.*

```
def vectorize (A: [[float]]) -> [float]:
  m, n, k = len(A), len(A[0]), 0
  vecA = [0] * (m+n)
  for i in range (m):
    for j in range (n):
      vecA[k] = A[i][j]
      k += 1
  return vecA
```

  Uma versão com `append`.

```
def vectorize (A: [[float]]) -> [float]:
  vecA = []
  for i in range (len (A)):
    for j in range (len (A[0])):
      vecA.append (A[i][j])
  return vecA
```

  Uma versão mais sucinta utilizando *list comprehensions.*

```
def vectorize (A: [[float]]) -> [float]:
  return [A[i][j] for j in range (len (A[0])) for i in range (len (A))]
```

  Uma versão utilizando `itertools`.

```
import itertools as itt
def vectorize (A: [[float]]) -> [float]:
  return list (itt.chain.from_iterable (A))
```

  □

- **(02)** *Matriz Diagonal Dominante.*
  Uma matriz $A : [0\,..\,n-1] \times [0\,..\,n-1] \to \mathbb{R}$ é *diagonal dominante* se os elementos de cada linha são menores ou iguais ao elemento da mesma linha que reside na diagonal principal. Em forma simbólica, A é diagonal dominante se

$$A[i][i] \geq A[i][j] \qquad \text{para todo} \qquad 0 \leq i, j \leq n-1.$$

  Escreva uma função que recebe uma matriz A e determina se A é diagonal dominante.

  Solução:

```
def is_diagonal_dominant (A: [[float]]) -> bool:
  n = len(A)
  for i in range (n):
    for j in range (n):
      if A[i][i] < A[i][j]:
        return False
  return True
```

  □

- **(03)** *Matrizes Seguras.*
  Seja $A : [0\,..\,n-1] \times [0\,..\,n-1] \to \{0, 1\}$ uma matriz quadrada de números binários. Dizemos que A é *segura* se dois ou mais 1's não ocorrem: (i) na mesma linha; (ii) na mesma coluna; (iii) na mesma diagonal (nos sentidos principal e secundário). Por exemplo,

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

  a matriz da esquerda é segura, a do meio e a da direita não. Escreva uma função que recebe A e decide se ela é segura.

  Solução:

```
def is_safe (A: [[int]]) -> bool:
  n = len(A)

  # verificação das linhas e colunas
  for i in range (n):
    onesrow = onescol = 0
    for j in range (n):
      if A[i][j] == 1: onesrow += 1
      if A[j][i] == 1: onescol += 1
    if onesrow > 1 or onescol > 1:
      return False

  # verificação das diagonais principais e secundárias
  for d in range (-n+1,n):
    onesfst = onessnd = 0
    for k in range (n-abs(d)):
      if A[k-min(0,d)][k+max(0,d)]       == 1: onesfst += 1
      if A[k-min(0,d)][n-1-(k+max(0,d))] == 1: onessnd += 1
    if onesfst > 1 or onessnd > 1:
      return False

  return True
```

  □

- **(04)** *Produto de Krönecker.*
  Para duas matrizes $A \in \mathbb{R}^{k \times \ell}$ e $B \in \mathbb{R}^{m \times n}$, o *produto de Krönecker* ou *tensorial* entre A e B, denotado por $A \otimes B$, é uma matriz de blocos que possui dimensões $km \times \ell n$ e é dada por:

$$A \otimes B = \begin{pmatrix} a_{0,0}B & \cdots & a_{0,\ell-1}B \\ \vdots & \ddots & \vdots \\ a_{k-1,0}B & \cdots & a_{k-1,\ell-1}B \end{pmatrix}, \qquad \text{em que} \qquad a_{i,j}B = \begin{pmatrix} a_{i,j}b_{0,0} & \cdots & a_{i,j}b_{0,n-1} \\ \vdots & \ddots & \vdots \\ a_{i,j}b_{m-1,0} & \cdots & a_{i,j}b_{m-1,n-1} \end{pmatrix}.$$

  Por exemplo, se

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \text{e} \quad B = \begin{pmatrix} 0 & 5 \\ 6 & 7 \end{pmatrix}, \qquad A \otimes B = \begin{pmatrix} 1\begin{pmatrix} 0 & 5 \\ 6 & 7 \end{pmatrix} & 2\begin{pmatrix} 0 & 5 \\ 6 & 7 \end{pmatrix} \\ 3\begin{pmatrix} 0 & 5 \\ 6 & 7 \end{pmatrix} & 4\begin{pmatrix} 0 & 5 \\ 6 & 7 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 & 5 & 0 & 10 \\ 6 & 7 & 12 & 14 \\ 0 & 15 & 0 & 20 \\ 18 & 21 & 24 & 28 \end{pmatrix}.$$

  Escreva uma função que recebe A e B e devolve $A \otimes B$.

  Solução:

```
def tensor_product (A: [[float]], B: [[float]]) -> [[float]]:
  mA, nA = len (A), len(A[0])
  mB, nB = len (B), len(B[0])
  C = [[0 for j in range (nA*nB)] for i in range (mA*mB)]

  for i in range (mA):
    for k in range (mB):
      for j in range (nA):
        for l in range (nB):
          C[i*mB+k][j*nB+l] = A[i][j]*B[k][l]
  return C
```

  □

- **(05)** *Autovetor Dominante.*
  Seja $A : [0\,..\,n-1] \times [0\,..\,n-1] \to \mathbb{R}$ uma matriz quadrada de valores reais. Um vetor $x[0\,..\,n-1]$ de reais não nulos é um *auto-vetor* de A se $Ax = \lambda x$, para algum $\lambda \in \mathbb{R}$ chamado *auto-valor* de A associado a x. Um auto-vetor é *dominante* se o valor absoluto de seu auto-valor é um máximo dentre os demais auto-valores (em absoluto) de A.

  O método iterativo a seguir calcula uma aproximação para o auto-vetor dominante de A: seja $x_0[0\,..\,n-1]$ um vetor inicial não nulo e $m > 0$ um inteiro; faça

$$x_{k+1} = \frac{Ax_k}{\|Ax_k\|} \qquad \text{enquanto} \qquad k \leq m.$$

  Note que: Ax_k é o produto da matriz A com o vetor x_k que resulta em um vetor y, e que $\|Ax_k\| = \|y\|$ é a norma euclidiana (comprimento) do vetor y, que pode ser calculada como a raiz quadrada do produto interno $\langle Ax_k, Ax_k \rangle$; isto é, $\|y\| = \sqrt{\sum_{i=0}^{n-1} y_i^2}$.

  (a) Escreva uma função que recebe A e m e devolve uma aproximação para o auto-vetor dominante de A.

  Solução:

```
from math import sqrt

def euclid_norm (z: [float]) -> float:
  return sqrt (zi*zi for zi in z)

def matrix_prod_vector (A: [[float]], x: [float]) -> [float]:
  return [sum (aij*xj for aij, xj in zip (ai, x)) for ai in A]

def power_method (A: [[float]], x: [float], m: int):
  for k in range (1,m+1):
    x = matrix_prod_vector (A, x)
    c = euclid_norm (x)
    if c == 0: return (False, x)
    for j in range (len(x)):
      x[j] /= c
   return (True, x)
```

  □

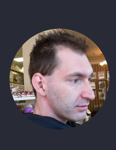
  (b) É sempre possível devolver um auto-vetor dominante? Por que? O que ocorre caso não seja possível devolvê-lo?

  Solução:
  Subjacente está o fato de que estamos trabalhando em $\mathbb{R}$: as matrizes, os vetores e os escalares são todos reais. Existem, no entanto, matrizes reais (i.é, todas as entradas são valores reais) nas quais:

  - todos os auto-valores são complexos (ex., matrizes anti-simétricas),
  - todos os auto-vetores são todos complexos (ex., matrizes de rotação).

  Para esses casos, precisaríamos revisar o código acima para lidar com valores complexos.

  □



Aritanan Gruber
Assistant Professor

"See, if y'all haven't the same feeling for *this*, I really don't give a damn. If you ain't feeling it, then dammit *this* ain't for you!"
(desconheço a autoria; agradeço a indicação)

CV