Ejercicio de programación: la suma de las columnas de una matriz triangular (un tema del curso "Álgebra Lineal Numérica")

Egor Maximenko

Instituto Politécnico Nacional
Escuela Superior de Física y Matemáticas
México

3 de abril de 2021

Definición

Sea $A \in \mathcal{M}_n(\mathbb{R})$. Se dice que A es triangular inferior, si

$$\forall j, k \in \{1, \ldots, n\} \qquad j < k \qquad \Longrightarrow \qquad A_{j,k} = 0.$$

Denotamos por $\mathfrak{lt}_n(\mathbb{R})$ al conjunto de todas las matrices cuadradas reales triangulares inferiores de orden n:

$$\mathfrak{lt}_n(\mathbb{R}) \coloneqq \Big\{ A \in \mathcal{M}_n(\mathbb{R}) \colon \quad \forall j, k \in \{1, \ldots, n\} \quad j < k \quad \Longrightarrow \quad A_{j,k} = 0 \Big\}.$$

Problema: programar una función que devuelve la sumas de las columnas de la matriz triangular inferior dada

Entrada: una matriz triangular inferior A.

Salida: la suma de las columnas de A.

Restricción: evitar el trabajo con los elementos de A por arriba de la diagonal principal.

Problema: programar una función que devuelve la sumas de las columnas de la matriz triangular inferior dada

Entrada: una matriz triangular inferior A.

Salida: la suma de las columnas de A.

Restricción: evitar el trabajo con los elementos de A por arriba de la diagonal principal.

Ejemplo.

$$\begin{bmatrix} A_{1,1} & 0 & 0 \\ A_{2,1} & A_{2,2} & 0 \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \quad \mapsto \quad \begin{bmatrix} A_{1,1} \\ A_{2,1} + A_{2,2} \\ A_{3,1} + A_{3,2} + A_{3,3} \end{bmatrix}.$$

Solución por renglones, con un ciclo j y operación sum, para $n = 4$

Vamos a resolver el problema en el lenguaje Python 3 con la librería numpy.

$$A = \begin{bmatrix} A_{0,0} & 0 & 0 & 0 \\ A_{1,0} & A_{1,1} & 0 & 0 \\ A_{2,0} & A_{2,1} & A_{2,2} & 0 \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}.$$

Solución por renglones, con un ciclo j y operación sum, para $n = 4$

Vamos a resolver el problema en el lenguaje Python 3 con la librería numpy.

$$A = \begin{bmatrix} A_{0,0} & 0 & 0 & 0 \\ A_{1,0} & A_{1,1} & 0 & 0 \\ A_{2,0} & A_{2,1} & A_{2,2} & 0 \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}.$$

```
from numpy import *

def sum_columns_lt_j_4(A):
    s = zeros(4)
    s[0] = sum(A[0, 0 : 1])
    s[1] = sum(A[1, 0 : 2])
    s[2] = sum(A[2, 0 : 3])
    s[3] = sum(A[3, 0 : 4])
    return s
```

Solución por renglones, con un ciclo j y operación sum

```
def sum_columns_lt_j(A):
    n = A.shape[0]
    s = zeros(n)
    for j in range(n):
        s[j] = sum(A[j, 0 : j])
    return s
```

Solución por renglones, con dos ciclos anidados

En la solución anterior sustituimos la operación “sum” por un ciclo.

```
def sum_columns_lt_jk(A):
    n = A.shape[0]
    s = zeros(n)
    for j in range(n):
        for k in range(j):
            s[j] += A[j, k]
    return s
```

Solución por columnas, $n = 4$

$$A = \begin{bmatrix} A_{0,0} & 0 & 0 & 0 \\ A_{1,0} & A_{1,1} & 0 & 0 \\ A_{2,0} & A_{2,1} & A_{2,2} & 0 \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \mapsto \begin{bmatrix} A_{0,0} \\ A_{1,0} \\ A_{2,0} \\ A_{3,0} \end{bmatrix} + \begin{bmatrix} 0 \\ A_{1,1} \\ A_{2,1} \\ A_{3,1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ A_{2,2} \\ A_{3,2} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ A_{3,3} \end{bmatrix}.$$

Solución por columnas, $n = 4$

$$A = \begin{bmatrix} A_{0,0} & 0 & 0 & 0 \\ A_{1,0} & A_{1,1} & 0 & 0 \\ A_{2,0} & A_{2,1} & A_{2,2} & 0 \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \mapsto \begin{bmatrix} A_{0,0} \\ A_{1,0} \\ A_{2,0} \\ A_{3,0} \end{bmatrix} + \begin{bmatrix} 0 \\ A_{1,1} \\ A_{2,1} \\ A_{3,1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ A_{2,2} \\ A_{3,2} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ A_{3,3} \end{bmatrix}.$$

```
def sum_columns_lt_k_4(A):
    s = zeros(4)
    s[0 : 4] += A[0 : 4, 0]
    s[1 : 4] += A[1 : 4, 1]
    s[2 : 4] += A[2 : 4, 2]
    s[3 : 4] += A[3 : 4, 3]
    return s
```

Solución por columnas, un ciclo sobre k, operación axpy

```
def sum_columns_lt_k(A):
    n = A.shape[0]
    s = zeros(n)
    for k in range(n):
        s[k : n] += A[k : n, k]
    return s
```

Solución por columnas, dos ciclos anidados

En el algoritmo anterior sustituimos la operación axpy por un ciclo y obtenemos otra solución.

```
def sum_columns_lt_kj(A):
    n = A.shape[0]
    s = zeros(n)
    for k in range(n):
        for j in range(k, n):
            s[j] += A[j, k]
    return s
```

Pruebas de validez

```
def test_sum_columns_lt(n):
    X = random.randn(n, n)
    Y = tril(X)
    s0 = sum_columns_lt_j(Y)
    s1 = sum_columns_lt_jk(Y)
    s2 = sum_columns_lt_k(Y)
    s3 = sum_columns_lt_kj(Y)
    er1 = norm(s1 - s0)
    er2 = norm(s2 - s0)
    er3 = norm(s3 - s0)
    return max([er1, er2, er3])
```

Ejercicios

Ejercicio. Contar el número de adiciones de números reales.

No contar los incrementos de los ciclos.

Ejercicios

Ejercicio. Contar el número de adiciones de números reales.
No contar los incrementos de los ciclos.

Ejercicio. Medir los tiempos de ejecución.

Ejercicios

Ejercicio. Contar el número de adiciones de números reales.
No contar los incrementos de los ciclos.

Ejercicio. Medir los tiempos de ejecución.

Ejercicio obligatorio.
Resolver el problema similar para las matrices triangulares superiores.
Escribir 4 funciones que corresponden a 4 métodos (j, jk, k, kj)
y la función que verifica la validez.

Ejercicios

Ejercicio. Contar el número de adiciones de números reales.
No contar los incrementos de los ciclos.

Ejercicio. Medir los tiempos de ejecución.

Ejercicio obligatorio.
Resolver el problema similar para las matrices triangulares superiores.
Escribir 4 funciones que corresponden a 4 métodos (j, jk, k, kj)
y la función que verifica la validez.

Ejercicio. Sumar los renglones, en vez de las columnas.