Programación: Multiplicación de matices por vectores, versión con axpy

Objetivos. Recordar que el producto de una matriz por un vector se puede ver como una combinación lineal de las columnas de la matriz, y programar el algoritmo correspondiente en varias formas. Comparar la eficiencia real de varios algoritmos que multiplican matrices por vectores.

Requisitos. Programación con funciones, arreglos y ciclos. Definición del producto de una matriz por un vector. El producto de una matriz por un vector como una combinación lineal de las columnas de la matriz. Operación axpy.

1. Problema. Escribir una función de dos argumentos A, b que calcule y regrese el producto Ab, donde A es una matriz y b es un vector.

Entrada: una matriz A y un vector b.

Propiedades de la entrada: se supone que el número de columnas de A coincide con la longitud de b (no es necesario verificar que se cumple esta condición).

Salida: el producto Ab.

Vamos a resolver este problema de varias maneras, representando Ab como una combinación lineal de las columnas de A.

2. El producto de una matriz por un vector como una combinación lineal de las columnas de la matriz, ejemplo con $m = 2$ y $n = 3$. Sean $A \in \mathcal{M}_{2\times 3}(\mathbb{R})$, $b \in \mathbb{R}^3$. Entonces

$$Ab = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \left[\begin{array}{ccc} & + & \\ \hline & + & \end{array}\right]$$

$$= \begin{bmatrix} A_{1,1}b_1 \\ \\ \end{bmatrix} + \begin{bmatrix} A_{1,2}b_2 \\ \\ \end{bmatrix} + \begin{bmatrix} A_{1,3}b_3 \\ \\ \end{bmatrix}$$

$$= \underbrace{}_{?} \begin{bmatrix} A_{1,1} \\ \\ \end{bmatrix} + \underbrace{}_{?} \begin{bmatrix} \\ \\ \end{bmatrix} + \underbrace{}_{?} \begin{bmatrix} \\ \\ \end{bmatrix}$$

$$= b_1 A_{*,1} + \quad + \quad = \sum_{k=1}^{3} \quad .$$

3. Los pasos del algoritmo para $n = 3$. Supongamos que $A \in \mathcal{M}_{m\times 3}(\mathbb{R})$, $b \in \mathbb{R}^3$. Entonces el producto Ab se puede calcular con los siguientes comandos:

```
function [] = ideamulmatvecaxpy(),
   A = [3, -2, 5; 7, -1, 4]; b = [-4; 1; 2];
   m = 2; n = 3;
   display(A); display(b);
   c = zeros(???, ???);
   c = c + b(???) * A(???, ???);
   display(c);
   c = c + b(???) * A(???, ???);
   display(c);
   c = c + b(???) * A(???, ???);
   display(c);
   display(A * b);
end
```

4. El producto de una matriz por un vector como una combinación lineal de las columnas de la matriz, ejemplo con $m = 3$ y $n = 2$. Sean $A \in \mathcal{M}_{3\times 2}(\mathbb{R})$, $b \in \mathbb{R}^2$. Entonces

$$Ab = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} \quad + \quad \\ \hline \quad + \quad \\ \hline \quad + \quad \end{bmatrix}$$

$$= \begin{bmatrix} A_{1,1}b_1 \\ \\ \\ \\ \\ \\ \end{bmatrix} + \begin{bmatrix} A_{1,2}b_2 \\ \\ \\ \\ \\ \\ \end{bmatrix} = \underbrace{\quad}_{?} \begin{bmatrix} A_{1,1} \\ \\ \\ \\ \\ \\ \end{bmatrix} + \underbrace{\quad}_{?} \begin{bmatrix} \\ \\ \\ \\ \\ \\ \end{bmatrix}$$

$$= \qquad + \qquad = \sum_{k=1}^{2} \qquad .$$

5. Algoritmo con axpy. Resolver el Problema 1 usando la función `axpy` programada anteriormente. Un esquema de solución en Matlab:

```
function [c] = mulmatvec1c(A, b),
   [m, n] = size(A);
   c = zeros(???, ???);
   for k = 1 : ???,
      c = axpy(???, ???, ???);
   end
end
```

El número 1 en el título de la función significa que dentro de la función usamos una función auxiliar de nivel 1, a saber, la función `axpy` que contiene un ciclo `for`. Notemos que la función `mulmatvec1r` es de nivel 2 porque tiene dos ciclos `for` añidados: el ciclo exterior está escrito de manera explícita, el ciclo interior está escondido dentro de la función `axpy`. La letra `c` en el título de la función significa que la función trabaja con las columnas de la matriz.

6. Algoritmo con axpy usando las operaciones del lenguaje. Si el lenguaje de programación tiene un operaciones internas para sumar vectores y multiplicarlos por escalares, entonces se pueden usar estas operaciones internas en vez de nuestra función `axpy`. Un esquema de solución en Matlab:

```
function [c] = mulmatvec1cinternal(A, b),
   [m, n] = size(A);
   c = zeros(???, ???);
   for k = 1 : ???,
      c = c + ???;
   end
end
```

7. Solución con dos ciclos, por columnas. Recordamos que la función `axpy` tiene un ciclo. Podemos encajar este ciclo dentro del ciclo de los algoritmos anteriores:

```
function [c] = mulmatvec0c(A, b),
   [m, n] = size(A);
   c = zeros(???, ???);
   for k = 1 : ???,
      for j = 1 : ???,
         c(j) = c(j) + ??? * ???;
      end
   end
end
```

El número 0 en el título de la función significa que dentro de la función hacemos operaciones solamente con objetos de dimesión 0, es decir, con entradas de A y b.

8. Comprobación con datos pequeños pseudoaleatorios.

```
function [] = test2mulmatvecr(),
   m = 8; n = 5; A = randn(m, n); b = randn(???, ???);
   c1 = mulmatvec1c(A, b);
   c2 = mulmatvec1cinternal(A, b);
   c3 = mulmatvec0c(A, b);
   c4 = A * b;
   er2 = norm(c1 - c2); er3 = norm(c1 - c3); er4 = norm(c1 - c4);
   display([er2, er3, er4]);
```

9. Comprobación con datos pseudoaleatorios de tamaños grandes y comparación de la rapidez. Comparar varias funciones que calculan el producto de una matriz por un vector, por ejemplo, la función `mulmatvec1rinternal` (programada anteriormente) y `mulmatvec1cinternal`.

```
function [] = testmulmatvecr(m, n, numrep),
   A = ???; b = ???;
   t0 = cputime();
   for rep = 1 : numrep,
      c1 = mulmatvec1rinternal(A, b);
   end
   t1 = cputime() - t0;
   t0 = cputime();
   for rep = 1 : numrep,
      c2 = mulmatvec1cinternal(A, b);
   end
   t2 = cputime() - t0;
   display([t1, t2]);
end
```

¿Cuál versión es más rápida?