Programación: Multiplicación de matrices por vectores

Objetivos. Realizar varios algoritmos de multiplicación de matrices por vectores. Analizar la complejidad de estos algoritmos y comparar su eficiencia real.

Requisitos. Programación con funciones, arreglos y ciclos. Definición del producto de una matriz por un vector. Producto punto.

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.

2. Recordamos la fórmula para las componentes del producto de una matriz por un vector. Sean $A \in \mathcal{M}_{3\times 4}(\mathbb{R})$, $b \in \mathbb{R}^4$. Denotemos Ab por c:

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\ A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\ A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}.$$

Entonces $c \in \mathbb{R}^3$, y cada entrada del vector c se puede escribir como el producto punto de un renglón de A por la columna b. Complete las fórmulas:

$$c_1 = A_{1,*}b = A_{1,1}b_1 + \underbrace{\qquad}_{?} + \underbrace{\qquad}_{?} + \underbrace{\qquad}_{?} = \sum_{k=1}^{3} \underbrace{\qquad}_{?};$$

$$c_2 = \qquad = \qquad + \qquad + \qquad + \qquad = \sum_{k=1}^{3} \qquad ;$$

$$c_3 = \qquad = \qquad + \qquad + \qquad + \qquad = \sum \qquad .$$

3. Fórmulas para $m = 3$, $n = 4$. Sean $A \in \mathcal{M}_{m\times n}(\mathbb{R})$ y $b \in \mathbb{R}^4$. Escribamos las fórmulas del ejercicio anterior en forma breve.

El producto $c = Ab$ es un vector real de longitud $\underbrace{\quad}_{?}$; en otras palabras, $c \in \underbrace{\quad}_{?}$.

Para cada j en $\{1, 2, 3\}$, la j-ésima componente del vector c es el producto punto del j-ésimo renglón de A por la columna b:

$$c_j = A_{???,???}b. \tag{1}$$

En forma más detallada,

$$c_j = \sum_{k=???}^{???} A_{???,???}b_{???} \tag{2}$$

4. Fórmula general para las entradas del producto de una matriz por un vector. Sean $A \in \mathcal{M}_{m\times n}(\mathbb{R})$ y $b \in \mathbb{R}^n$. Generalice los resultados del ejercicio anterior:

El producto $c := Ab$ es un vector real de longitud $\underbrace{\quad}_{?}$; en otras palabras, $c \in \underbrace{\quad}_{?}$.

Para cada $j \in \{1, \ldots, \underbrace{\quad}_{?}\}$, la j-ésima componente del vector c se escribe como el producto punto

$$c_j = \tag{3}$$

o como la suma

$$c_j = \sum \qquad . \tag{4}$$

5. Algoritmo con el producto punto. Resolver el Problema 1 usando la función programada anteriormente que calcula el producto punto (digamos, `dotproduct`). Un esquema de solución en Matlab:

```
function [c] = mulmatvec1r(A, b),
   [m, n] = size(A);
   c = zeros(???, 1);
   for j = 1 : ???,
      c(j) = dotproduct(???, ???);
   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 `dotproduct` 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 `dotproduct`. La letra `r` en el título de la función significa que la función trabaja con renglones de la matriz. La multiplicación de matrices por vectores se puede realizar también de otra manera, por columnas (ese método se explica en otro archivo).

6. Algoritmo con el producto punto usando la operación del lenguaje. Si el lenguaje de programación tiene una operación interna que calcula el producto punto, entonces se puede usar esta operación interna en vez de nuestra función `dotproduct`. Un esquema de solución en Matlab:

```
function [c] = mulmatvec1rinternal(A, b),
   [m, n] = size(A);
   c = zeros(???, 1);
   for j = 1 : ???,
      c(j) = ??? * ???;
   end
end
```

7. Encajar el cálculo del producto punto. En vez de usar la función `dotproduct` o la operación interna correspondiente, se puede poner un ciclo que haga los mismos cálculos. Recordamos que el producto punto se calcula como cierta suma; este cálculo se puede realizar como un ciclo:

```
s = ???;
for k = 1 : ???,
   s = s + ??? * ???;
end
```

En nuestro caso no es necesario introducir una variable auxiliar `s` para acumular la suma. El papel de la variable acumulador `s` puede hacer la componente `c(j)` del vector `c`. Notamos que con el comando `c = zeros(???, 1)` todas las componentes del vector `c` ya están inicializadas y tienen valor 0.

8. Solución con dos ciclos añidados, por renglones.

```
function [c] = mulmatvec0r(A, b),
   [m, n] = size(A);
   c = zeros(???, 1);
   for j = 1 : ???,
      for k = 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.

9. Análisis de complejidad. Calcular el número de operaciones de multiplicación en alguno de los algoritmos anteriores. La respuesta es un polinomio en las variables m y n.

10. Comprobación con datos pequeños fijos.

```
function [] = test1mulmatvecr(),
   A = [3, -2, 5; 7, -1, 4]; b = [-4; 1; 2];
   c1 = mulmatvec1r(A, b); c2 = mulmatvec1rinternal(A, b);
   c3 = mulmatvec0r(A, b); c4 = A * b;
   display([c1, c2, c3, c4]);
end
```

11. Comprobación con datos pequeños pseudoaleatorios.

```
function [] = test2mulmatvecr(),
   m = 4; n = 3; A = randn(m, n); b = randn(???, ???);
   c1 = ???; c2 = ???; c3 = ???; c4 = ???;
   er2 = norm(c1 - c2); er3 = norm(c1 - c3); er4 = norm(c1 - c4);
   display([er2, er3, er4]);
```

12. Comprobación con datos pseudoaleatorios de tamaños grandes y comparación de la rapidez.

```
function [] = test3mulmatvecr(m, n, numrep),
   A = ???; b = ???;
   t0 = cputime();
   for rep = 1 : numrep,
      c1 = mulmatvec1r(A, b);
   end
   t1 = cputime() - t0;
   t0 = cputime();
   for rep = 1 : numrep,
      c2 = mulmatvec1rinternal(A, b);
   end
   t2 = cputime() - t0;
   ??? # similar fragment with mulmatvec0r
   ??? # similar fragment with A * b
   er2 = ???; er3 = ???; er4 = ???;
   display([t1, t2, t3, t4]); display([er2, er3, er4]);
end
```

Ejecutar esta función con varios valores de m y n, eligiendo `numrep` de tal manera que el tiempo sea de 0.1 a 100 segundos.

```
test3mulmatvecr(100, 100, 10);
test3mulmatvecr(200, 200, 10);
test3mulmatvecr(400, 400, 10);
```

¿Cómo se cambia el tiempo al multiplicar m y n por 2? ¿Cuál de las dos funciones, `mulmatvec1r` y `mulmatvec0r`, es más rápida?