Lab: Cinco formas de multiplicar matrices
Tratamiento de señales

Multiplicación de matrices Sin pérdida de generalidad, trabajaremos con matrices de componentes reales.

Para poder multiplicar dos matrices tienen que ser compatibles en sus dimensiones. Así, si $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{p\times q}$, podremos multiplicar AB si y sólo si $n = p$ y podremos multiplicar BA si y sólo si $q = m$. Construimos una función `check` que tiene dos argumentos y comprueba si la matriz del primer argumento puede multiplicarse por la matriz del segundo argumento. Si no son compatibles avisa y se termina la ejecución:

```
function [m, n, q] = check(A,B)
    [m,n] = size(A);
    [p,q] = size(B);
    assert(n == p, 'Las matrices no son compatibles');
end
```

Componente a componente Si A y B son compatibles para calcular $C = AB$, será porque $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{n\times q}$. Las componentes de C se calculan mediante la ecuación siguiente:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \qquad i = 1, \ldots, m; \quad j = 1, \ldots, q.$$

- Construimos una función `mult1` que calcule el producto de matrices aplicando esta ecuación:

```
function C = mult1(A, B)
    [m, n, q] = check(A, B);
    C = zeros(m, q);
    for i=1:m
        for j=1:q
            for k=1:n
                C(i,j) = C(i,j) + A(i,k) * B(k,j);
            end
        end
    end
end
```

- Comprobamos que la función funciona correctamente (a partir de este punto, comprobaremos siempre el funcionamiento correcto de cada una de las implementaciones que hagamos):

```
A = rand(4,3); B = rand(3, 5); A*B - mult1(A,B)
```

  si la diferencia entre ambas formas de calcular matrices es del orden de $1e^{-15}$ o similar se debe a errores numéricos...

- Otra forma de calcular la multiplicación $C = AB$ componente a componente es calculando mq productos internos (de una fila por una columna). Utilizando una notación[1] similar a la de matlab:

$$c_{ij} = a_{i,:} * b_{:,j}, \qquad i = 1, \ldots, m; \quad j = 1, \ldots, q;$$

  Construimos una función `mult1b` que utiliza esta técnica:

```
function C = mult1b(A, B)
    [m, n, q] = check(A, B);
    C = zeros(m, q);
    for i=1:m
        for j=1:q
            C(i,j) = A(i,:) * B(:,j);
        end
    end
end
```

[1] En matlab, si `A` es una matriz, `A(i,j)` es la componente escalar en la fila `i` y la columna `j`, `A(i,:)` es la fila `i`-ésima y `A(:,j)` es la columna `j`-ésima. Análogamente, nosotros utilizaremos a_{ij} para identificar el elemento escalar de la fila i y columna j, $a_{i,:}$ para identificar al vector fila i-ésimo y $a_{:,j}$ para identificar al vector columna j-ésimo.

Combinando columnas

- Construimos una función `mulmatvec` que tiene como entrada una matriz A y un vector columna b y devuelve el vector columna $c = Ab$ calculado como combinación lineal de las columnas de la matriz. Como comprobación adicional, comprobamos que efectivamente b sea un vector columna. Con notación similar a Matlab:

$$c = \sum_{k=1}^{n} a_{:,k} b_k$$

```
function c = multmatvec(A, b)
    [m,n,q] = check(A,b);
    assert(q == 1, 'b debe ser un vector columna');
    c = zeros(m,1);
    for k=1:n
        c = c + A(:,k) * b(k);
    end
end
```

- Comprobamos que la función se comporta adecuadamente:

```
A = rand(4, 3); b = rand(3, 1); A*b - multmatvec(A,b)
```

- Construimos una función `mult2` que calcule el producto de matrices de tal manera que cada columna es una combinación lineal de las columnas de la matriz de la izquierda. Para ello, invocamos reiteradamente a la función `multmatvec`:

```
function C = mult2(A, B)
    [m, n, q] = check(A, B);
    C = zeros(m, q);
    for j=1:q
        C(:,j) = multmatvec(A, B(:,j));
    end
end
```

- Comprobamos que la función se comporta adecuadamente:

```
A = rand(4,3); B = rand(3, 5); A*B - mult2(A,B)
```

Combinando filas Este caso es muy similar al anterior, y podemos hacerlo de la siguiente manera:

- Construimos una función `mulvecmat` que tiene como entrada un vector fila a y una matriz B y devuelve el vector fila $c = aB$ calculado como combinación lineal de las filas de la matriz:

$$c = \sum_{k=1}^{n} a_k b_{k,:}$$

```
function c = multmatvec(a, B)
    [m,n,q] = check(a,B);
    assert(m == 1, 'a debe ser un vector fila');
    c = zeros(1,q);
    for k=1:n
        c = c + a(k) * B(k,:);
    end
end
```

- Comprobamos que la función se comporta adecuadamente:

```
a = rand(1, 4); B = rand(4, 3); a*B - multvecmat(a,B)
```

- Construimos una función `mult3` que calcule el producto de matrices de tal manera que cada fila es una combinación lineal de las filas de la matriz de la derecha. Para ello, invocamos reiteradamente a la función `multvecmatmat`:

```
function C = mult3(A, B)
    [m, n, q] = check(A, B);
    C = zeros(m, q);
    for i=1:m
        C(i,:) = multvecmat(A(i,:), B);
    end
end
```

- Comprobamos que la función se comporta adecuadamente:

```
A = rand(4,3); B = rand(3, 5); A*B - mult3(A,B)
```

Como suma de matrices de rango 1

- En primer lugar, construimos la función `multcolrow` que a partir de un vector columna a y un vector fila b devuelva la matriz de rango 1 $C = ab$. Como comprobación adicional, aseguramos que efectivamente a sea vector columna y b vector fila:

```
function C = multcolrow(a, b)
    [m,n,q] = check(a,b);
    assert(n == 1, 'a debe ser un vector columna y b un vector fila');
    C = a*b;
end
```

- Comprobamos que funciona adecuadamente obteniendo una matriz 4×2:

```
a = rand(4, 1); b = rand(1, 2); C = multcolrow(a, b)
```

- Construimos la función `mult4` que calcula el producto de matrices $C = AB$ como suma de matrices de rango 1:

```
function C = mult4(A, B)
    [m, n, q] = check(A, B);
    C = zeros(m, q);
    for k=1:n
        C = C + multcolrow(A(:,k), B(k, :));
    end
end
```

- Comprobamos que la función se comporta adecuadamente:

```
A = rand(4,3); B = rand(3, 5); A*B - mult4(A,B)
```

Por cajas Al multiplicar dos matrices por cajas, hay que dividir cada una de las matrices en una retícula rectangular de submatrices. Si queremos calular $C = AB$ donde $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{n\times q}$, de lo que nos tenemos que asegurar es de que la división de las n columnas de A coincida con la división de las n filas de B. Así por ejemplo, si $n = 11$, podríamos dividir A verticalmente en 2 bloques de 8 y 3 columnas, o en 2 bloques de 6 y 5 o en 3 bloques de 2, 3 y 6 columnas, etc. Esta misma división de columnas que hagamos sobre A la debemos hacer sobre las filas de B. En cuanto a las filas de A, las podemos dividir como queramos siempre y cuando los bloques tengan al menos una fila y entre todos los bloques tengamos las m filas. Independientemente, las columnas de B se pueden dividir como queramos siempre y cuando cada bloque tenga al menos una columna y entre todos los bloques sumen las q columnas. Por supuesto, el orden de las filas y columnas originales deben respetarse y no intercambiarse

- Realizamos una función `particion` que dado un número natural[1] (que será en su momento el número de filas o columnas de una matriz) nos devuelva un vector aleatorio de números naturales tal que su suma sea el número original. Para ello, inicializamos la variable disponibles al número original y vamos seleccionando un número aleatorio menor o igual a disponibles. Cuando ya no queden más, terminamos de realizar la particion.

  ```
  function v = particion(m)
      disponibles = m;
      v = [];
      while (disponibles >= 1)
          n = randi(disponibles);
          v = [v, n];
          disponibles = disponibles - n;
      end
      assert(sum(v) == m);
  end
  ```

  como comprobación adicional, en este caso redundante por cómo hemos realizado el algoritmo, comprobamos que la suma de los elementos de la partición es en efecto el número original.

- Comprobamos el funcionamiento de la función:

  ```
  for k=1:10:100, particion(k), end
  ```

- Realizamos una función `desdehasta` que dada una partición, devuelva donde empieza y donde acaba el elemento k-ésimo de la partición. Por ejemplo, si tenemos la partición $[10, 12, 13, 24]$ de 59, la primer elemento va desde el 1 hasta el 10; el segundo desde el 11 hasta el 22; el tercero desde el 23 hasta el 35; y el cuarto desde 36 hasta 59:

  ```
  function [d,h] = desdehasta(v, k)
      w = cumsum([0, v]);
      d = w(k) + 1;
      h = w(k+1);
  end
  ```

- Comprobar el funcionamiento de la función con:

  ```
  v = [10, 12, 13, 24];
  for k=1:4, [d,h] = desdehasta(v, k), end
  ```

- Realizamos una función `caja` que dada una matriz A, una partición de sus filas y una partición de sus columnas, devuelva la caja i, j:

  ```
  function C = caja(A, M, N, i, j)
      [dm, hm] = desdehasta(M, i);
      [dn, hn] = desdehasta(N, j);
      C = A(dm:hm, dn:hn);
  end
  ```

- Comprobamos el funcionamiento de la función con:

[1] Consideramos que los números naturales no incluyen al 0.

```
A = reshape(1:20, 4, 5)
M = [2,2];
N = [3,2];
caja(A, M, N, 1, 1)
caja(A, M, N, 1, 2)
caja(A, M, N, 2, 1)
caja(A, M, N, 2, 2)
```

- Realizamos una funcion `mult5` que dadas dos matrices A y B, calcule $C = AB$ realizando particiones aleatorias de las filas de A, una partición aleatoria común para las columnas de A y las filas de B y una partición aleatoria de las columnas de B. A continuación, calcula la matriz por cajas multiplicando las particiones anteriores de forma similar a como hicimos con la primera forma de multiplicar matrices, salvo que ahora, en vez de ir multiplicando escalares vamos multiplicando cajas:

```
function C = mult5(A, B)
    [m, n, q] = check(A, B);

    M = particion(m)
    N = particion(n)
    Q = particion(q)

    C = zeros(m, q);

    for i=1:length(M)
        for j=1:length(Q)
            c = zeros(M(i), Q(j));
            for k=1:length(N)
                a = caja(A, M, N, i, k);
                b = caja(B, N, Q, k, j);
                c = c + a*b;
            end
            [dm, hm] = desdehasta(M, i);
            [dn, hn] = desdehasta(Q, j);
            C(dm:hm, dn:hn) = c;
        end
    end
end
```

- Comprobamos[1] que funciona con:

```
A = rand(9, 7); B = rand(7, 11); mult5(A,B)-A*B
```

[1] Observamos que dentro de la función no se ha puesto un punto y coma al final de la creación de las particiones para ver que cada vez hace una distinta, pero cuando se esté seguro del funcionamiento, ya se puede poner un punto y coma para que no tenga salidas indeseadas por pantalla.