Tarea 6

Luis Fernando Cantú Díaz de León

31/3/2018

Investiga* sobre la subrutina de Fortran `dgemv` (parámetros que recibe y la salida).

La función `dgemv` realiza cualquiera de las siguientes operaciones matriz-vector:

$$y = \alpha A x + \beta y$$
$$y = \alpha A^T x + \beta y$$

Recibe como parámetros:

- **TRANS**: Especifica cuál de las dos operaciones es la que se llevará a cabo.
- **M**: INTEGER. Especifica el número de renglones de la matriz A.
- **N**: INTEGER. Número de columnas de la matriz A.
- **ALPHA**: DOUBLE PRECISION. EL valor del escalar α.
- **A**: DOUBLE PRECISION array, dimension(LDA, N). Antes de la entrada, la parte principal by parte del arreglo A debe contener la matriz de coeficientes.

- **LDA**: INTEGER. Especifica la primera dimensión de A tal como fue declarada en el programa. LDA debe de ser de al menos max(1, m).
- **X**: DOUBLE PRECISION array de dimensión de al menos `(1 + (n - 1)*abs(INCX))` cuando `TRANS == N o n` y de al menos `(1 + (m - 1)*abs(INCX))` de otra forma. Antes de la entrada, el arreglo incrementado X debe de contener al vector x.
- **INCX**: INTEGER. Especifica el incremento para los elementos de X. No puede ser cero.
- **BETA**: DOUBLE PRECISION. Especifica el escalar β. SI $\beta = 0$, entonces no es necesario especificar a Y como input.
- **Y**: DOUBLE PRECISION array de dimensión de al menos `(1 + (m - 1)*abs(INCY))` cuando `TRANS == N o n` y de al menos `(1 + (n - 1)*abs(INCY))` de otra forma. Antes de la entrada, el arreglo incrementado Y debe de contener al vector y. Y se sobreescribe en la salida con el nuevo vector y.
- **INCY**: INTEGER. Especifica el incremento para los elementos de Y. No debe de ser cero.

En la carpeta `analisis-numerico-computo-cientifico/C/BLAS/ejemplos/level2/` ejecuta el programa dgemv_mult_mat_vec.c y realiza pruebas con diferentes matrices y vectores definidos por ti.

Primero realizamos una prueba con el vector y la matriz especificados inicialmente.

```
gcc -Wall dgemv_mult_mat_vec.c funciones.c -o dgemv_mult_mat_vec.out -lblas
./dgemv_mult_mat_vec.out 3 2
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= -5.00000
## matriz[2][0]= -1.00000   matriz[2][1]= 2.50000
## ------------
## vector :
## vector[0]= 1.00000
```

```
## vector[1]= 0.00000
## ------------
## vector resultado:
## vector[0]= 0.00000
## vector[1]= 4.00000
## vector[2]= -1.00000
```

Después le agregamos un renglón a la matriz A.

```
./dgemv_mult_mat_vec.out 4 2
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= -5.00000
## matriz[2][0]= -1.00000   matriz[2][1]= 2.50000
## matriz[3][0]= 1.00000    matriz[3][1]= 2.00000
## ------------
## vector :
## vector[0]= 1.00000
## vector[1]= 0.00000
## ------------
## vector resultado:
## vector[0]= 0.00000
## vector[1]= 4.00000
## vector[2]= -1.00000
## vector[3]= 1.00000
```

Finalmente, volvemos a agregar otro renglón a la matriz A.

```
./dgemv_mult_mat_vec.out 5 2
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= -5.00000
## matriz[2][0]= -1.00000   matriz[2][1]= 2.50000
## matriz[3][0]= 1.00000    matriz[3][1]= 2.00000
## matriz[4][0]= 3.00000    matriz[4][1]= 4.00000
## ------------
## vector :
## vector[0]= 1.00000
## vector[1]= 0.00000
## ------------
## vector resultado:
## vector[0]= 0.00000
## vector[1]= 4.00000
## vector[2]= -1.00000
## vector[3]= 1.00000
## vector[4]= 3.00000
```

Haz un programa que utilice la subrutina `dger` de Fortran.

```
gcc -Wall dger.c funciones.c -o dger.out -lblas
./dger.out 3 3
```

```
## matriz 1:
```

```
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000   matriz[0][2]= 4.00000
## matriz[1][0]= -5.00000   matriz[1][1]= -1.00000  matriz[1][2]= 2.50000
## matriz[2][0]= 1.00000    matriz[2][1]= 2.00000   matriz[2][2]= 3.00000
## ------------
## vector x:
## vector[0]= 1.00000
## vector[1]= 0.00000
## vector[2]= 2.00000
## ------------
## vector y:
## vector[0]= 2.00000
## vector[1]= 2.00000
## vector[2]= 2.00000
## ------------
## matriz resultado:
## matriz[0][0]= 2.00000    matriz[0][1]= 3.50000   matriz[0][2]= 6.00000
## matriz[1][0]= -5.00000   matriz[1][1]= -1.00000  matriz[1][2]= 2.50000
## matriz[2][0]= 5.00000    matriz[2][1]= 6.00000   matriz[2][2]= 7.00000
```

Después de haber estudiado y entendido los archivos de `definiciones.h` y `funciones.c` y realizado los puntos anteriores y la tarea 5 responde: ¿cómo fue que pudimos llamar a las rutinas de Fortran (que almacena en una forma column-major order los arreglos de dos dimensiones) para operaciones con arreglos 2-dimensionales sin haber instalado CBLAS, si en clase se dijo que almacenar arreglos de dos dimensiones en C es en un row-major order?

En el archivo `definiciones.h` se define un `struct` llamado **arreglo_2d** en el que se almacenan los datos de forma column-major.

Investiga* sobre la subrutina de Fortran dgemm (parámetros que recibe y la salida).

`dgemm` es una función que permite realizar la siguiente operación:

$$C = \alpha op(A) op(B) + \beta C$$

Los parámetros son los siguientes (tomado de www.netlib.org):

- TRANSA: TRANSA is CHARACTER*1. On entry, TRANSA specifies the form of op(A) to be used in the matrix multiplication as follows:

  ```
  TRANSA = 'N' or 'n',  op( A ) = A.

  TRANSA = 'T' or 't',  op( A ) = A**T.

  TRANSA = 'C' or 'c',  op( A ) = A**T.
  ```

- TRANSB: TRANSB is CHARACTER*1. On entry, TRANSB specifies the form of op(B) to be used in the matrix multiplication as follows:

  ```
  TRANSB = 'N' or 'n',  op( B ) = B.

  TRANSB = 'T' or 't',  op( B ) = B**T.
  ```

```
TRANSB = 'C' or 'c',  op( B ) = B**T.
```

- M:
  M is INTEGER. On entry, M specifies the number of rows of the matrix op(A) and of the matrix C. M must be at least zero.
- N:
  N is INTEGER. On entry, N specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. N must be at least zero.
- K:
  K is INTEGER. On entry, K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero.
- ALPHA:
  ALPHA is DOUBLE PRECISION. On entry, ALPHA specifies the scalar alpha.
- A:
  A is DOUBLE PRECISION array, dimension (LDA, ka), where ka is k when TRANSA = «N» or «n», and is m otherwise. Before entry with TRANSA = «N» or «n», the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.
- LDA:
  LDA is INTEGER. On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = «N» or «n» then LDA must be at least max(1, m), otherwise LDA must be at least max(1, k).
- B:
  B is DOUBLE PRECISION array, dimension (LDB, kb), where kb is n when TRANSB = «N» or «n», and is k otherwise. Before entry with TRANSB = «N» or «n», the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.
- LDB:
  LDB is INTEGER. On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = «N» or «n» then LDB must be at least max(1, k), otherwise LDB must be at least max(1, n).
- BETA:
  BETA is DOUBLE PRECISION. On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.
- C: C is DOUBLE PRECISION array, dimension (LDC, N) Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry. On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta*C).
- LDC:
  LDC is INTEGER. On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least max(1, m).

En la carpeta `analisis-numerico-computo-cientifico/C/BLAS/ejemplos/level3/` ejecuta el programa dgemm_mult_mat.c y realiza pruebas con diferentes matrices definidas por ti.

Primero vemos el caso especificado inicialmente.

```
gcc -Wall dgemm_mult_mat.c funciones.c -o dgemm_mult_mat.out -lblas
./dgemm_mult_mat.out 3 2 2 3
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= -5.00000
## matriz[2][0]= -1.00000   matriz[2][1]= 2.50000
## ------------
## matriz 2:
## matriz[0][0]= 1.00000    matriz[0][1]= 0.00000   matriz[0][2]= 0.00000
## matriz[1][0]= 0.00000    matriz[1][1]= -1.00000  matriz[1][2]= 1.00000
## ------------
## matriz resultado:
## matriz[0][0]= 0.00000    matriz[0][1]= -1.50000  matriz[0][2]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= 5.00000   matriz[1][2]= -5.00000
## matriz[2][0]= -1.00000   matriz[2][1]= -2.50000  matriz[2][2]= 2.50000
```

Ahora resolvemos para el caso en que agregamos un renglón a la matriz 1 y una columna a la matriz 2.

```
./dgemm_mult_mat.out 4 2 2 4
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= -5.00000
## matriz[2][0]= -1.00000   matriz[2][1]= 2.50000
## matriz[3][0]= 1.00000    matriz[3][1]= 2.00000
## ------------
## matriz 2:
## matriz[0][0]= 1.00000    matriz[0][1]= 0.00000   matriz[0][2]= 0.00000   matriz[0][3]= 0.00000
## matriz[1][0]= -1.00000   matriz[1][1]= 1.00000   matriz[1][2]= 0.00000   matriz[1][3]= 1.00000
## ------------
## matriz resultado:
## matriz[0][0]= -1.50000   matriz[0][1]= 1.50000   matriz[0][2]= 0.00000   matriz[0][3]= 1.50000
## matriz[1][0]= 9.00000    matriz[1][1]= -5.00000  matriz[1][2]= 0.00000   matriz[1][3]= -5.00000
## matriz[2][0]= -3.50000   matriz[2][1]= 2.50000   matriz[2][2]= 0.00000   matriz[2][3]= 2.50000
## matriz[3][0]= -1.00000   matriz[3][1]= 2.00000   matriz[3][2]= 0.00000   matriz[3][3]= 2.00000
```

Finalmente, resolvemos para el caso en que tenemos matrices cuadradas de 3x3.

```
./dgemm_mult_mat.out 3 3 3 3
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000   matriz[0][2]= 4.00000
## matriz[1][0]= -5.00000   matriz[1][1]= -1.00000  matriz[1][2]= 2.50000
## matriz[2][0]= 1.00000    matriz[2][1]= 2.00000   matriz[2][2]= 3.00000
## ------------
## matriz 2:
## matriz[0][0]= 1.00000    matriz[0][1]= 0.00000   matriz[0][2]= 0.00000
## matriz[1][0]= 0.00000    matriz[1][1]= -1.00000  matriz[1][2]= 1.00000
## matriz[2][0]= 0.00000    matriz[2][1]= 1.00000   matriz[2][2]= 2.00000
## ------------
## matriz resultado:
## matriz[0][0]= 0.00000    matriz[0][1]= 2.50000   matriz[0][2]= 9.50000
## matriz[1][0]= -5.00000   matriz[1][1]= 3.50000   matriz[1][2]= 4.00000
## matriz[2][0]= 1.00000    matriz[2][1]= 1.00000   matriz[2][2]= 8.00000
```

En la carpeta del punto anterior encuentras la sección Multiplicación matriz-matriz con trick. Ejecuta el programa de esta sección con diferentes matrices definidas por ti y resuelve la pregunta ¿por qué funciona este trick?.

Primero realizamos la operación con las matrices especificadas inicialmente.

```
gcc -Wall trick.c funciones_trick.c -o trick.out -lblas
./trick.out 3 2 2 3
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= -5.00000
## matriz[2][0]= -1.00000   matriz[2][1]= 2.50000
## ------------
## matriz 2:
## matriz[0][0]= 1.00000    matriz[0][1]= 0.00000   matriz[0][2]= 0.00000
## matriz[1][0]= 0.00000    matriz[1][1]= -1.00000  matriz[1][2]= 1.00000
## matriz 3:
## matriz[0][0]= 0.00000    matriz[0][1]= -1.50000  matriz[0][2]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= 5.00000   matriz[1][2]= -5.00000
## matriz[2][0]= -1.00000   matriz[2][1]= -2.50000  matriz[2][2]= 2.50000
```

Ahora resolvemos para el caso en que agregamos un renglón a la matriz 1 y una columna a la matriz 2.

```
./trick.out 4 2 2 4
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= -5.00000
## matriz[2][0]= -1.00000   matriz[2][1]= 2.50000
## matriz[3][0]= 1.00000    matriz[3][1]= 2.00000
## ------------
## matriz 2:
## matriz[0][0]= 1.00000    matriz[0][1]= 0.00000   matriz[0][2]= 0.00000   matriz[0][3]= 0.00000
## matriz[1][0]= -1.00000   matriz[1][1]= 1.00000   matriz[1][2]= 0.00000   matriz[1][3]= 1.00000
## matriz 3:
## matriz[0][0]= -1.50000   matriz[0][1]= 1.50000   matriz[0][2]= 0.00000   matriz[0][3]= 1.50000
## matriz[1][0]= 9.00000    matriz[1][1]= -5.00000  matriz[1][2]= 0.00000   matriz[1][3]= -5.00000
## matriz[2][0]= -3.50000   matriz[2][1]= 2.50000   matriz[2][2]= 0.00000   matriz[2][3]= 2.50000
## matriz[3][0]= -1.00000   matriz[3][1]= 2.00000   matriz[3][2]= 0.00000   matriz[3][3]= 2.00000
```

Finalmente, resolvemos para el caso en que tenemos matrices cuadradas de 3x3.

```
./trick.out 3 3 3 3
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000   matriz[0][2]= 4.00000
## matriz[1][0]= -5.00000   matriz[1][1]= -1.00000  matriz[1][2]= 2.50000
## matriz[2][0]= 1.00000    matriz[2][1]= 2.00000   matriz[2][2]= 3.00000
## ------------
## matriz 2:
## matriz[0][0]= 1.00000    matriz[0][1]= 0.00000   matriz[0][2]= 0.00000
## matriz[1][0]= 0.00000    matriz[1][1]= -1.00000  matriz[1][2]= 1.00000
## matriz[2][0]= 0.00000    matriz[2][1]= 1.00000   matriz[2][2]= 2.00000
## matriz 3:
## matriz[0][0]= 0.00000    matriz[0][1]= 2.50000   matriz[0][2]= 9.50000
## matriz[1][0]= -5.00000   matriz[1][1]= 3.50000   matriz[1][2]= 4.00000
```

```
## matriz[2][0]= 1.00000    matriz[2][1]= 1.00000   matriz[2][2]= 8.00000
```

Como podemos apreciar, los resultados son idénticos a los del programa anterior. La razón por la que esta rutina funciona es sencilla: se invierte el orden en que se multiplican las matrices y también se invierten las matrices a multiplicar. Todo esto es posible gracias a la siguiente propiedad:

$$AB = (B^T A^T)^T$$

Haz un programa que utilice la subrutina `dsymm` de Fortran.

`dsymm` lleva a cualquiera de las siguientes operaciones:

$$C = \alpha AB + \beta C$$
$$C = \alpha BA + \beta C$$

Donde α y β son escalares, A es una matriz simétrica y B, C son matrices de tamaño $M \times N$.

A continuación un ejemplo de lo anterior:

```
gcc -Wall dsymm.c funciones.c -o dsymm.out -lblas
./dsymm.out 2 3
```

```
## matriz 1:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000
## matriz[1][0]= 4.00000    matriz[1][1]= -5.00000
## ------------
## matriz 2:
## matriz[0][0]= 1.00000    matriz[0][1]= 0.00000   matriz[0][2]= 0.00000
## matriz[1][0]= 0.00000    matriz[1][1]= -1.00000  matriz[1][2]= 1.00000
## ------------
## matriz 3:
## matriz[0][0]= 0.00000    matriz[0][1]= 1.50000   matriz[0][2]= 4.00000
## matriz[1][0]= -5.00000   matriz[1][1]= -1.00000  matriz[1][2]= 2.50000
## ------------
## matriz resultado:
## matriz[0][0]= 0.00000    matriz[0][1]= 0.00000   matriz[0][2]= 5.00000
## matriz[1][0]= 1.50000    matriz[1][1]= -1.50000  matriz[1][2]= 0.00000
```