Como pasar matrices como argumentos de subrutinas en FORTRAN 77

Pablo Santamaría

v0.1 (Agosto 2006)

1. Planteo del problema

Los vectores y matrices usuales de la matemática son representados en FORTRAN como *arreglos*. Un arreglo unidimensional es utilizado para almacenar un vector, mientras que un arreglo bidimensional es utilizado para guardar una matriz.

En FORTRAN 77 los arreglos tienen que tener un tamaño predefinido al momento de definirlos por primera vez. De esta manera es usual definir los arreglos de tamaños suficientemente grande como para contener los vectores y o matrices de nuestro problema. La manera "prolija" de declarar explicitamente el tamaño de un arreglo es a través de una sentencia `PARAMETER`. Así el siguiente fragmento de código define un arreglo unidimensional `V` para contener un vector de a lo más `NMAX` elementos reales, y un arreglo bidimensional `A` que puede contener una matriz real de a lo más `NMAX` filas y `MMAX` columnas, donde fijamos, para nuestros propósitos `NMAX` = `MMAX` = 5.

```
      INTEGER NMAX,MMAX
      PARAMETER (NMAX=5,MMAX=5)
      REAL V(NMAX)
      REAL A(NMAX,MMAX)
```

Si bien en este apunte nos queremos centrar específicamente con matrices, comenzaremos discutiendo la situación con vectores para poner en contexto los problemas que surgen cuando pasamos a considerar matrices.

Supóngamos que queremos escribir una subrutina para imprimir los *n* primeros elementos de un vector. La siguiente subrutina cumplirá tal misión:

```
      SUBROUTINE IMPRIMIR_VECTOR(N,V)
C     -------------------------------
C     Primera version (perfectible)
C     -------------------------------
      IMPLICIT NONE
      INTEGER N
      INTEGER NMAX
      PARAMETER (NMAX=5)
      REAL V(NMAX)
      INTEGER I
C     -------------------------------
      DO I=1,N
        WRITE (*,*) V(I)
      ENDDO
C     -------------------------------
      END
```

Esta subrutina puede ser utilizada en el siguiente programa, donde utilizamos un arreglo unidimensional de NMAX componentes para guardar un vector cuya componente *i*-ésima es *i*.

```
      PROGRAM TEST
C     ------------------------------
      IMPLICIT NONE
      INTEGER NMAX
      PARAMETER (NMAX=5)
      REAL V(NMAX)
      INTEGER I,N
C     ------------------------------
C     Inicializamos el vector
C     ------------------------------
      DO I=1,NMAX
          V(I) = REAL(I)
      END DO
C     ------------------------------
C     Pedimos al usuario cual es el número de componentes
C     a imprimir
C     ------------------------------
      WRITE (*,*) 'Ingrese número de componentes a imprimir'
      READ(*,*) N
      IF (N.GT.NMAX) THEN
          WRITE(*,*) 'Número de componentes excedido'
          STOP
      ENDIF
C     ------------------------------
C     Llamamos a la subrutina
C     ------------------------------
      CALL IMPRIMIR_VECTOR(N,V)
C     ------------------------------
      END
```

Si bien esto funciona, nótese que el arreglo es dimensionado a 5 en el programa principal (a través del parámetro NMAX), y nuevamente dimensionado a 5 en la subrutina. Esta codificación no es flexible. Si queremos trabajar con un arreglo mayor, digamos de 100 elementos, tenemos que cambiar la sentencia PARAMETER a lo largo del código dos veces. Imaginemos ahora que tenemos varias subrutinas que operan con el vector codificadas de la misma forma. Para lograr que éstas funcionen correctamente tenemos que revisar de punta a punta el código y cambiar las correspondientes sentencias PARAMETER. Como nosotros estamos intentando ser buenos programadores, queremos escribir subrutinas más generales que puedan ser re-utilizadas sin alterar a las mismas.

Una manera proceder, que nos permite FORTRAN 77, es declarar, *en la subrutina*, la dimensión del arreglo unidimensional al tamaño N que queremos usar (valor que es pasado junto con el arreglo como argumento) en vez del máximo valor asignado en el programa principal. Esta forma de codificación es conocida como *método de arreglos de tamaño "ajustable"*. Aplicando este procedimiento a nuestro problema, la subrutina buscada es como sigue.

```
      SUBROUTINE IMPRIMIR_VECTOR(N,V)
C     -------------------------------
C     Segunda version
C     -------------------------------
      IMPLICIT NONE
      INTEGER N
      REAL V(N)
```

```
      INTEGER I
C     -------------------------------
      DO I=1,N
        WRITE (*,*) V(I)
      ENDDO
C     -------------------------------
      END
```

Ahora esta versión de la subrutina puede ser llamada en el programa principal para cualquier valor de N (en tanto no sea mayor que NMAX). Sólo es necesario alterar el valor de NMAX en el programa principal en el caso que el valor asignado en la sentencia PARAMETER no sea suficiente para nuestro problema.

Existe una segunda alternativa que podemos considerar para nuestro problema. Esta consiste en considerar *asumido* el valor de la dimensión de los arreglos que son argumentos en la subrutina, valor que ha sido definido en el programa que llama a la subrutina. Para hacer esto, utilizamos el signo asterisco (*) en la declaración del arreglo unidimensional. Este tipo de codificación es conocido como *método de arreglos de tamaño asumido*. Con este método la subrutina es escrita como sigue.

```
      SUBROUTINE IMPRIMIR_VECTOR(N,V)
C     -------------------------------
C     Tercera version
C     -------------------------------
      IMPLICIT NONE
      INTEGER N
      REAL V(*)
      INTEGER I
C     -------------------------------
      DO I=1,N
        WRITE (*,*) V(I)
      ENDDO
C     -------------------------------
      END
```

Cualquiera de las dos versiones presentadas de la subrutina cumplen su cometido. Solo debe tenerse presente que es responsabilidad del programador (o sea, nuestra) de que el arreglo sea definido con una dimensión mayor o igual que N en el programa que realiza la llamada.

Ahora que hemos discutido como codificar en forma eficiente el uso de arreglos unidimensionales como argumentos en subrutinas, pasemos al caso de arreglos bidimensionales y veamos si podemos aplicar los métodos anteriores.

Nos interesa ahora diseñar una subrutina para imprimir las *n* primeras filas y *m* primeras columnas de una matriz dada. Para fijar el problema consideraremos un arreglo bidimensional de dimensiones máximas NMAX = MMAX = 3, es decir, estamos asumiendo una matriz de trabajo de tres filas por tres columnas, dando un total de nueve elementos. Asignemos los valores de estos elementos a los numeros enteros consecutivos comenzando desde la unidad para el primer elemento y recorriendo la matriz por filas. En otras palabras, tenemos la matriz:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Esto puede hacerse facilmente con el siguiente programa

```
        PROGRAM TEST
C       ------------------------------------------------
        IMPLICIT NONE
```

```
      INTEGER NMAX,MMAX
      PARAMETER (NMAX=3,MMAX=3)
      REAL A(NMAX,MMAX)
      INTEGER N,M,I,J,COUNT
C     -----------------------------------------------
C     Asignamos valores a nuestra matriz de trabajo
C     -----------------------------------------------
      N=3
      M=3
      COUNT=1
      DO I=1,3
          DO J=1,3
              A(I,J) = REAL(COUNT)
              COUNT  = COUNT + 1
          END DO
      END DO
C     -----------------------------------------------
C     Testeamos que nuestra matriz de trabajo es correcta
C     -----------------------------------------------
      WRITE (*,*) 'MATRIZ DE TRABAJO:'
      WRITE (*,*)
      DO I=1,N
          WRITE (*,*) (A(I,J), J=1,M)
      ENDDO
      WRITE (*,*)
C     -----------------------------------------------
C     Aqui vendrá la llamada a la subrutina que imprime
C     las primeras n filas y m columnas
C     -----------------------------------------------
      ...
      CALL ...
      END
```

La subrutina que queremos llamar en el espacio reservado para tal fin en este programa es tal que, por ejemplo, para el caso $n = m = 2$ tiene que imprimir la submatriz 2×2

$$\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$$

Habiendo entendido lo que hicimos para el caso de arreglos de tamaño ajustable para una dimensión, resulta evidente que la forma en que intentamos inmediatamente de aplicar para el caso presente de dos dimensiones es algo similar a lo que sigue.

```
      SUBROUTINE IMPRIMIR_SUBMATRIZ(N,M,A)
C     -----------------------------------------------
C     Primera version (no funcionará!)
C     -----------------------------------------------
      IMPLICIT NONE
      INTEGER N,M,A(N,M)
      INTEGER i,j
C     -----------------------------------------------
      WRITE (*,*) 'SUBMATRIZ: ', N, ' X ', M
      WRITE (*,*)
      DO I=1,N
```

```
            WRITE (*,*) (A(I,J), J=1,M)
        ENDDO
        RETURN
        END
```

Entonces en el programa principal escribimos la llamada a la subrutina como sigue

```
        N = 2
        M = 2
        CALL IMPRIMIR_SUBMATRIZ(N,M,A)
```

Despues de compilar el programa, al ejecutarlo obtenemos la siguiente salida por pantalla:

```
 Matriz de trabajo:

  1 2 3
  4 5 6
  7 8 9

  Submatriz:  2 x  2

  1 7
  4 2
```

¡Esto no está nada bien! ¿Qué es lo qué está ocurriendo? Por favor, ¡devuélvanme mi dinero! Evidentemente, pasar el arreglo bidimensional con tamaños ajustables para *ambos* índices no está dando el resultado buscado. Probemos entonces utilizar el método de arreglos de tamaño asumido. La subrutina sería ahora como sigue.

```
        SUBROUTINE IMPRIMIR_SUBMATRIZ(N,M,A)
C       ---------------------------------------------
C       Segunda version (no compilará!)
C       ---------------------------------------------
        IMPLICIT NONE
        INTEGER N,M,A(*,*)
        INTEGER I,J
C       ---------------------------------------------
        WRITE (*,*) 'SUBMATRIZ: ', N, ' X ', M
        WRITE (*,*)
        DO I=1,N
            WRITE (*,*) (A(I,J), J=1,M)
        ENDDO
        RETURN
        END
```

¡Esta versión es aún peor! ¡El programa ni siguiera compila! Evidentemente considerar *ambos* índices del arreglo bidimensional como dimensiones asumidas tampoco conduce a nada.

En este punto seguramente ya estamos desilusionados y entonces, volviendo sobre la idea en la codificación original de la subrutina para un arreglo unidimensional, consideramos angustiados la posibilidad de definir las dimensiones del arreglo en la subrutina igual a las del programa principal. Entonces escribimos la subrutina como sigue:

```
        SUBROUTINE IMPRIMIR_SUBMATRIZ(N,M,A)
```

```
C       ---------------------------------------------
C       Tercer versión (no flexible)
C       ---------------------------------------------
        IMPLICIT NONE
        INTEGER N,M
        INTEGER NMAX,MMAX
        PARAMETER (NMAX=3, MMAX=3)
        REAL A(NMAX,MMAX)
        INTEGER I,J
C       ---------------------------------------------
        WRITE (*,*) 'SUBMATRIZ: ', N, ' X ', M
        WRITE (*,*)
        DO I=1,N
            WRITE (*,*) (A(I,J), J=1,M)
        ENDDO
        RETURN
        END
```

Esta vez el programa compila, y cuando lo ejecutamos obtenemos lo siguiente:

```
 Matriz de trabajo:

  1 2 3
  4 5 6
  7 8 9

 Submatriz:  2 x  2

  1 2
  4 5
```

¡Funcionó! Si, es cierto, funciona. Pero el costo es que volvimos a diseñar una subrutina que no es flexible. Si necesitamos almacenar una matriz de mayores dimensiones tendremos que alterar todas las sentencias `PARAMETER` en todas las subrutinas que implementemos.

Como ya he dicho, intentamos ser buenos programadores y, entonces, esta solución no nos satisface. Más aún queremos saber porque fracasaron nuestros dos primeros intentos, y como podemos escribir subrutinas que pasen matrices como argumentos de la manera más flexible posible. Por tal motivo continuamos leyendo.

2. Como guarda FORTRAN 77 los arreglos en memoria

Para entender (y solucionar) nuestro problema no alcanza con conocer la sintaxis para operar con arreglos. Necesitamos también conocer como FORTRAN 77 almacena estos objetos en memoria.

Cuando un arreglo es declarado en FORTRAN 77, el compilador sabrá cuanta memoria consecutiva debe reservar para todos los elementos del arreglo en base a la declaración de tipo (`INTEGER`, `REAL`, `DOUBLE PRECISION`, etc) y de los números que se asignan a los tamaños de las dimensiones del arreglo. Los elementos del arreglo son almacenados en esta porción de memoria uno después de otro. Pero la forma en que se disponen depende de la dimensionalidad del arreglo.

Para arreglos unidimensionales, la situación es simple. Un arreglo unidimensional es asignado en la memoria recorriendo el índice del arreglo de menor a mayor. Así, si tenemos la siguiente declaración

```
INTEGER NMAX
PARAMETER (NMAX=3)
REAL V(NMAX)
```

la cual declara un conjunto de tres elementos (todos ellos reales), los mismos son destribuidos en la memoria en el orden

V(1)	V(2)	V(3)

De esta manera, si, para simplificar, consideramos que la posición de memoria del primer elemento es la 1, entonces el elemento *i*-ésimo de un arreglo unidimensional ocupará la posición de memoria `K = I`.

La situación se complica para arreglos multidimensionales. En FORTRAN 77 un arreglo bidimensional es almacenado en memoria variando el primer indice más rápidamente. Esto implica que *la matriz que guardamos en el arreglo bidimensional es almacenada en memoria por columnas*. Para ejemplificar, consideremos nuevamente la matriz 3×3,

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Esta matriz la guardamos en un arreglo bidimensional que declaramos así

```
INTEGER NMAX,MMAX
PARAMETER (NMAX=3,MMAX=3)
REAL A(NMAX,MMAX)
```

Entonces, en la memoria de la computadora los elementos del arreglo son distribuidos a partir de la posición de memoria del elemento `A(1,1)` como sigue

A(1,1)	A(2,1)	A(3,1)	A(1,2)	A(2,2)	A(3,2)	A(1,3)	A(2,3)	A(3,3)

Es fácil ver que si, para simplificar, consideramos que la posición de memoria del primer elemento es la 1, entonces la posición de memoria que ocupa el elemento `A(I,J)` de índices `I`, `J`, es

```
K = I + NMAX*(J-1)
```

donde `NMAX` es el límite superior de la primera dimensión del arreglo (es decir el número de filas de la matriz).

En nuestro ejemplo particular, vemos que la submatriz formada con las primeras $n = 2$ filas y $m = 2$ columnas, tiene sus elementos guardados en la primera, cuarta, segunda y quinta posiciones de memoria. Claramente estas posiciones no son consecutivas en la memoria.

Incidentalmente, notemos que, debido al proceso de guardar por columnas, *la fórmula que da la posición de memoria no involucra el límite superior* `MMAX` *de la segunda dimensión*. Este límite superior solo es necesario para determinar el máximo valor permitido del segundo índice y, en consecuencia, el tamaño del arreglo.

3. Solución al problema

Para terminar de comprender (y solucionar) nuestro problema tenemos que saber que en FORTRAN 77 los argumentos en la llamada de un subprograma (ya sea una `SUBROUTINE` o una `FUNCTION`) son *pasados por referencia*. Esto significa que las variables no son pasadas en otras copias para ser usadas en el subprograma, sino que lo que se pasa son las posiciones de memoria de las mismas. Así el subprograma

opera sobre las mismas posiciones de memoria de las variables involucradas en la llamada. En particular, *cuando se pasa un arreglo, lo que realmente se pasa en la llamada de la subrutina es la posición de memoria del primer elemento del arreglo*. Esta información, *junto* con la declaración dimensional del arreglo que se hace *en* la subrutina, resulta suficiente para que la subrutina pueda determinar la posición de memoria de cualquier elemento del arreglo y obtener así su valor. El punto clave es realizar una declaración dimensional correcta en la subrutina para que todo funcione.

En el caso de un arreglo unidimensional la situación es simple. En este caso tanto el método de arreglo ajustable como el método de arreglo asumido funciona porque, conociendo la posición de memoria del primer elemento del arreglo, el valor de un elemento *i*-ésimo cualquiera es obtenido por el programa simplemente leyendo *i*-ésima posición de memoria (por simplicidad, tomamos a la posición de memoria del primer elemento como 1).

Para un arreglo bidimensional, según el esquema que hemos visto, no le alcanzará a la subrutina conocer solo la posición de memoria del primer elemento (nuevamente, por simplicidad tomamos como 1 a tal posición), sino que será necesario conocer el límite superior `NMAX` del primer índice (es decir el número total de filas de la matriz). Ahora resulta claro porque falló la compilación al tratar de considerar en la subrutina el arreglo bidimensional de tamaño asumido. ¡El primer índice necesariamente requiere un tamaño, no podemos poner un asterisco!

Por otro lado nótese que `NMAX` es el límite superior del primer índice utilizado para dimensionar la matriz en el programa que llama a la subrutina, el cual, en general, será mayor que el valor que queremos utilizar como índice en las operaciones sobre el arreglo en la subrutina. Sin embargo, en la declaración de dimensional de la subrutina éste tiene que ser `NMAX` y no otro valor, porque sino la fórmula que utiliza el compilador para obtener las posiciones de memoria conducirá a posiciones incorrectas y por lo tanto a valores incorrectos. Esto es precisamente lo que nos pasó cuando en la subrutina declaramos al arreglo como

```
REAL A(N,M)
```

con `N` (y `M`) pasados como argumentos. Con `N=2` (en lugar de `NMAX = 3`), los valores que obtenemos deducidos a partir de las posiciones de memoria para la submatriz 2×2, conducen a un resultado incorrecto. En efecto, supongamos que queremos la componente `I=1, J=2`, la posición de memoria será `K = 1 + 2*(2-1) = 3`, la cual contiene el número 7. Así pues, en nuestra subrutina tenemos, erroneamente, `A(1,2) = 7` .

A esta altura nos debe quedar claro que *en la declaración de dimensionalidad del arreglo bidimensional en la subrutina el valor del límite superior del primer índice debe ser igual al valor del límite superior correspondiente a la sentencia que define dicho arreglo en el programa que llama a la subrutina*. Por esto funcionó nuestra última subrutina. Entonces resulta evidente que si queremos escribir una subrutina flexible para pasar arreglos bidimensionales como argumentos tenemos que pasar *también* como argumento el límite superior del primer índice. ¿Y qué pasa con el segundo índice? Pues bien, dado que éste no interviene en la determinación de las posiciones de memoria podemos considerarlo como de valor asumido dentro de la subrutina, y utilizar entonces para dicho índice el asterisco.

Así pues, nuestra subrutina para escribir las *n* primeras fila y *m* primeras columnas debe ser escrita como sigue.

```
      SUBROUTINE IMPRIMIR_SUBMATRIZ(N,M,NMAX,A)
C     ------------------------------------------------
C     Versión definitiva
C     ------------------------------------------------
      IMPLICIT NONE
      INTEGER N,M,NMAX
      REAL A(NMAX,*)
      INTEGER I,J
C     ------------------------------------------------
```

```
WRITE (*,*) 'SUBMATRIZ: ', N, ' X ', M
WRITE (*,*)
DO I=1,N
    WRITE (*,*) (A(I,J), J=1,M)
ENDDO
RETURN
END
```

y la llamada a la misma en el programa principal debe ser como sigue.

```
N=2
M=2
CALL IMPRIMIR_SUBMATRIZ(N,M,NMAX,A)
```

Si ahora compilamos el programa y lo ejectuamos obtendremos el resultado deseado. ¡Hemos cumplido nuestro objetivo!

4. Conclusión

Habiendo comprendido como FORTRAN 77 guarda en memoria los arreglos bidimensionales, ya tenemos una forma de construir, en forma general y flexible, subrutinas que pasen matrices como argumentos. El procedimiento aquí contado es utilizado en muchos paquetes numéricos de subrutinas para problemas de álgebra lineal (tal es el caso de LAPACK). Así el conocimiento de este método no solo es beneficioso para nuestras propias subrutinas sino también para poder usar en forma correcta tales paquetes.

Para finalizar les cuento que existe una segunda alternativa para diseñar subrutinas que trabajen con matrices. Esta consiste básicamente en guardar la matriz en un arreglo *unidimensional*. Esto no será tratado aquí, pero será, seguramente, el tema de un próximo apunte.

Espero que este apunte les haya resultado útil. Cualquier comentario o sugerencia diríganla a pablo@fcaglp.unlp.edu.ar