Universidad de Santiago de Compostela
Facultad de Matemáticas

Análisis Numérico Matricial. Curso 2020-2021
Grado en Matemáticas
Doble Grado en I. Informática y en Matemáticas
Doble Grado en Matemáticas y en Física

Práctica 1: Vectores y Matrices en FORTRAN 90

En esta práctica retomamos la programación en **FORTRAN 90** que ya utilizasteis en la materia del primer cuatrimestre del segundo curso **Cálculo Numérico en una Variable**. A partir de ahora, la mayor novedad consiste en la manipulación de vectores y matrices, es decir de **arreglos (arrays).**

1. Arreglos (arrays)

1.1. Declaración de arreglos

La **forma de un arreglo** queda definida por dos propiedades:

- **rango** (rank) **del arreglo:** número de dimensiones del arreglo (máximo 7).
- **extensión** (extent) **de cada dimensión:** número de elementos del que consta cada dimensión del arreglo.

Para conocer el valor de estas dos propiedades, FORTRAN 90 posee las funciones `SIZE` y `SHAPE`:

- `SIZE(a[,dim])`: calcula la extensión del arreglo `a` en la dimensión especificada por el valor `dim`. Si el argumento opcional `dim` se omite, esta función devuelve el número total de elementos del arreglo `a` (producto de las extensiones de sus dimensiones).
- `SHAPE(a)`: calcula un vector unidimensional, cuya longitud viene dada por el rango del arreglo `a` y cuyos coeficientes son las extensiones de las dimensiones del arreglo.

Diremos que dos arreglos son **conformes** si tienen la misma forma, es decir el mismo rango y las mismas extensiones; ten en cuenta que un escalar se considera conforme con cualquier arreglo.

1.2. Tipos de arreglos

En FORTRAN 90, existen diversos **tipos de arreglos** que recordamos a continuación:

Como primer ejemplo, para declarar un arreglo de números reales llamado `ar`, de rango 3, con extensiones 4, 8 y 5, añadimos el atributo `dimension` a la sentencia de declaración de tipo:

```
program principal

implicit none
...
real,dimension(4,8,5)::ar !equivalente a: real::ar(4,8,5)
...
end program principal
```

El anterior arreglo se conoce con el nombre de **arreglo de forma explícita (explicit-shape array).** También son de este tipo aquellos arreglos argumento cuyos límites se establecen en el momento de entrar en un procedimiento (por ejemplo, a través de otros argumentos); vemos un ejemplo:

```
subroutine eje(m,n,x,y)

implicit none
integer,intent(in)::m,n
real,intent(inout)::x(m,n),y(m,2*n) ...
```

Un procedimiento con argumentos ficticios que son arreglos cuyo tamaño varía de una llamada a otra, puede tener necesidad de arreglos locales cuyo tamaño también varía; son los **arreglos automáticos (automatic arrays),** que se crean cuando se llama al subprograma y desaparecen cuando se sale de él; en el siguiente ejemplo se crea el espacio `aux` con la misma forma que `a`:

```
subroutine intercambio(a,b)

implicit none
real,dimension(:),intent(inout)::a,b
real,dimension(size(a))::aux !"aux" tiene la misma forma que "a"

aux=a

if(size(a)==size(b)) then !"b" debe tener la misma forma que "a"
  a=b
  b=aux
endif

end subroutine intercambio
```

En este último ejemplo, los arreglos `a` y `b` son **arreglos de forma asumida (assumed-shape arrays),** es decir, arreglos argumento de un procedimiento que sólo conoce su rango pero no su tamaño. En estos casos, el procedimiento debe tener una **interfaz explícita** (mediante un bloque `interface`) en el lugar de llamada.

Algunas veces, el tamaño de un arreglo sólo se conoce después de leer algún dato o efectuar algún cálculo; en tal caso, podemos utilizar un **arreglo asignable de forma pospuesta o aplazada (allocatable o deferred-shape array),** que se define sólo como un nombre (y la información de su rango), pero sin memoria asociada (ya que no se conoce su tamaño). Sólo se podrá operar con este tipo de arreglos una vez que la instrucción `allocate` fija las extensiones de sus dimensiones; vemos un ejemplo:

```
program principal

implicit none
integer::n

real,allocatable::c(:,:),d(:,:) !"c" y "d" tienen rango 2
!y se van a utilizar para guardar dos matrices cuadradas

print*,"El orden de las matrices cuadradas c y d es:"
read*,n
print*,n

allocate(c(n,n),d(n,n))

print*,"Forma de los arreglos c y d: ",shape(c)

deallocate(c,d)

end program principal
```

Como se observa en el ejemplo, cuando el arreglo ya no es necesario, se puede liberar la memoria utilizada mediante la instrucción `deallocate`. Por último, debemos destacar que un arreglo declarado como `allocatable` no puede ser nunca un argumento ficticio en un procedimiento o el resultado de una función.

2. Procedimientos intrínsecos en FORTRAN 90

Parte del potencial de FORTRAN 90, tanto en lo que se refiere a programación paralela como a una expresión concisa de muchos algoritmos, se debe a su amplio conjunto de procedimientos intrínsecos.[1] Dos sencillos ejemplos de ellos son:

- `DOT_PRODUCT(vector_a,vector_b)`: efectúa el producto escalar de los vectores (arreglos de rango 1) del mismo tamaño `vector_a` y `vector_b`.
- `MATMUL(matrix_a,matrix_b)`: efectúa el producto de dos matrices compatibles para el producto `matrix_a` y `matrix_b`.

[1] https://gcc.gnu.org/onlinedocs/gfortran/Intrinsic-Procedures.html

3. Ejercicios

1. Escribe los siguientes **programas principales** y comprueba su buen funcionamiento con algunos ejemplos:

   1.1. Programa `prodesc_ppal` que lea y escriba el número y las componentes de dos vectores del mismo tamaño, y que efectúe y escriba su producto escalar; utiliza en primer lugar la propia definición de producto escalar y luego utiliza la función intrínseca `DOT_PRODUCT (vector_a,vector_b)`.

   1.2. Programa `matriz_ppal` que lea y escriba los números de filas, `m`, y de columnas, `n`, y los elementos de una matriz `a`. Utiliza para ello el código:

   ```
   print*
   print*,'La matriz A es:'
   do i=1,m
     read*,a(i,:)
     print*,a(i,:)
   end do
   ```

   Sustitúyelo por:

   ```
   print*
   print*,'La matriz A es:'
   do i=1,m
     read*,a(i,:)
   end do
   print*,a
   ```

   para determinar cómo se almacena la matriz `a` en la memoria del ordenador. Haz pruebas con `SIZE` y `SHAPE`.

   1.3. Programa `summat_ppal` que lea y escriba los números de filas y columnas y los elementos de dos matrices compatibles para la suma, y que efectúe y escriba su suma; ; utiliza en primer lugar la propia definición de suma de matrices y luego comprueba que basta utilizar el operador `+`.

   1.4. Programa `prodmat_ppal` que lea y escriba los números de filas y columnas y los elementos de dos matrices compatibles para la multiplicación, y que efectúe y escriba su producto; utiliza en primer lugar la propia definición de producto de matrices y luego utiliza la función intrínseca `MATMUL(matrix_a,matrix_b)`.

2. Escribe un **programa principal** `matdve_ppal` que lea y escriba el orden de una **matriz tridiagonal** A, sus tres diagonales, `ad`, `al` y `au`, y un vector v, y que calcule y escriba el **vector producto** $w = Av$.

3. Dados A una matriz $m \times n$, b un vector con m componentes y u un vector con n componentes, se quiere calcular el nuevo vector $r = Au - b$, denominado **vector residuo del sistema lineal** $Au = b$.

   3.1. Escribe un **programa principal** `residuo_ppal` que lea y escriba los datos `m`, `n`, `a`, `b` y `u`, y escriba `r` una vez obtenido mediante la correspondiente subrutina.

   3.2. **Detalla las fórmulas de cálculo del vector residuo.** Utilízalas para programar una **subrutina:**

   `residuo(m,n,a,b,u,r)`

   que calcule el vector `r`.

   3.3. El vector $r = Au - b$, en función de las **columnas de la matriz** A, es:

$$r = \left(\begin{array}{c|c|c|c} & & & \\ a^{(1)} & a^{(2)} & \dots & a^{(n)} \\ & & & \end{array} \right) \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} - b =$$

$$= a^{(1)}u_1 + a^{(2)}u_2 + \cdots + a^{(n)}u_n - b$$

   Dado que en FORTRAN una matriz se almacena por columnas, utiliza la anterior escritura del vector r con el fin de obtener una subrutina más eficiente en tiempo de cálculo.

   3.4. ¿Es posible eliminar el argumento `r` en la subrutina `residuo`?

   3.5. Crea ficheros de datos adecuados para validar las tareas propuestas.