0.1. Álgebra Vectorial - vectores y matrices

El uso de la sentencia DO en el caso de vectores y matrices es muy útil, ya que es la herramienta adecuada para controlar los subíndices de los elementos que forman estas estructuras. Veamos varios ejemplos:
Cargar una matriz en memoria y sumar los cuadrados de sus elementos de la diagonal.
Como la diagonal para una matriz con elementos $a_{i,j}$ son los elementos cuyos indices i y j son iguales, la suma de la diagonal sería:

$$S = \sum_{i=1}^{N} a_{i,i}$$

En el programa primero definiré el tamaño de la máximo matriz, luego leeré de pantalla cuál es su tamaño real (n). Con este valor de n puedo construir dos DOs. El primero genera todos lo casos posibles i y el segundo todos los casos de j. De esta manera puedo leer de pantalla elemento a elemento de la matriz. Una vez que leí todos los elementos, puedo pasar a realizar el cálculo.

```
program diagonal

real*4 a(100,100)
read(*,*) n. ! leo el tamaño de la matriz
do i=1,n
  do j=1,n
    read(*,*) a(i,j) ! leo los elementos
  enddo
enddo

tn=0.
do i=1,n
   tn=tn+a(i,i) ! Hago el cálculo
enddo
write(*,*) 'la suma de la diagonal es =', tn. ! imprimo el resultado
end
```

Probemos ahora sumar los elementos de la diagonal multiplicada por su antidiagonal, esta sería:

$$S = \sum_{i=1}^{N} a_{i,i}\ a_{i,n-i+1}$$

Y si tomo la parte que hace solo la operación (la lectura será igual que en el programa anterior)

```
program antidiagonal
⋮ ! Cargo los datos de la Matriz A
tn=0.
do i=1,n
  tn=tn+a(i,i)*a(i,n-i+1) ! hago el cálculo del subíndice (N-i+1) y lo utilizo
enddo
write(*,*)'la suma de la diagonal por la antidiagonal es=', tn
end
```

Suma de las filas de una matriz

Para sumar los elementos de las filas, tomo el índice de la fila y uso una sentencia DO que recorra toda la fila y un DO externo que recorra a su vez todas las filas de la Matriz. El resultado de esta operación un vector por lo cual tengo que definirlo.

```
PROGRAM FILA

REAL*4 A(100,100),R(100))
⋮ ! Cargo los datos de la Matriz A
⋮
DO I=1,N
R(I)=0.
  DO J=1,N
    R(I) = R(I) + A(I,J)
  ENDDO
ENDDO
⋮ ! Escribo el resultado del Vector R
END
```

0.1.1. Multiplicación de Matrices

En donde se puede ver la versatilidad de este tipo de comando es en la multiplicación de matrices Si tengo dos matrices A y B de igual tamaño) y las multiplico con resultado C, o sea:
AxB = C

Es decir

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n} \end{bmatrix} x \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & b_{2,3} & \cdots & b_{2,n} \\ b_{3,1} & b_{3,2} & b_{3,3} & \cdots & b_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{n,1} & b_{n,2} & b_{n,3} & \cdots & b_{n,n} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & c_{2,3} & \cdots & c_{2,n} \\ c_{3,1} & c_{3,2} & c_{3,3} & \cdots & c_{3,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{n,1} & c_{n,2} & c_{n,3} & \cdots & c_{n,n} \end{bmatrix} \tag{1}$$

Donde elemento de C se calcula como:

$$C_{i,j} = \sum_{k=1}^{N} A_{i,k} B_{k,j}$$

Esta formula es muy sencilla de programar tomando en cuenta que ahora tengo tres variables (i, j y k), las cuales tienen que tomar todos sus valores posibles dentro del programa para realizar el cálculo total. Una de ellas (**k**) tiene que hacer un ciclo completo para cada **i** y **j**. Veamos como sería el programa que resuelve este problema (obviamos la carga de datos en memoria de cada matriz para no hacer tan largo el código)

```
PROGRAM PRODUCTO

REAL*4 A(100,100),B(100,100),C(100,100)
```

```
⋮ ! cargamos los datos en las matrices A y B
⋮
DO I=1,N
 DO J=1,N

  C(I, J) = 0
  DO K=1,N
     C(I, J) = C(I, J) + A(I, K) * B(K, J)
  ENDDO

 ENDDO
ENDDO
⋮ ! Guardamos el resultado de la matriz C
⋮
END
```

Consideraciones sobre este programa:

- Es independiente del tamaño de las matrices salvo que sean más grandes que la dimensión de 100x100 con que se las definió al comienzo del programa. Con cambiar esos números, y re-compilando el problema es fácilmente resuelto. El algoritmo en si, no cambia con el tamaño de las matrices ("**N**").
- El tiempo que tarda puede ser **importante**, como son tres DOs uno dentro de otro el tiempo que tarda es proporcional a $\mathbf{t} \propto \mathbf{N^3}$. Por lo tanto, el tiempo aumenta muy rápido si se incrementa el tamaño de las matrices.
- En algoritmos como este cuyo tiempo depende de una potencia de **N**. Es muy fácil programar algo sencillo que exceda la capacidad de la computadora. Siempre hay que evitar que el tiempo que el programa tarda en correr y entregar un resultado no sea excesivo. Por lo cual, hay que estar atento a si el tiempo es proporcional a una potencia de la cantidad de valores a procesar y cuál es esta potencia. Es decir, se deben elegir cuando sea posible los algoritmos en los cuales esa potencia sea baja. Hay veces que se prefiere un cálculo aproximado a uno que da un resultado exacto pero que consume mucho tiempo en el cálculo.
- Reforzando el punto anterior, hay que considerar que realizar más operaciones matemáticas puede hacer que se pierdan más decimales y por lo tanto un cálculo largo tiene una mayor probabilidad a generar resultados que no son del todo correctos o son poco fiables.