

Análisis Numérico para Ingeniería

Clase Nro. 2

*“The best way to predict
the future is to invent it.”*

Alan Kay

Temas a tratar en esta clase :

- Estructuras de Control en FORTRAN
- Arreglos (Arrays) en FORTRAN
- Operaciones con Arreglos (Arrays)
- Lectura y Grabación de Archivos
- Subrutinas y Funciones
- Visualización de Datos con GNUPLOT
- Temas Avanzados de Programación (opcional)

Estructura básica de un programa

[**PROGRAM** nombre del programa]

[declaración de variables globales]

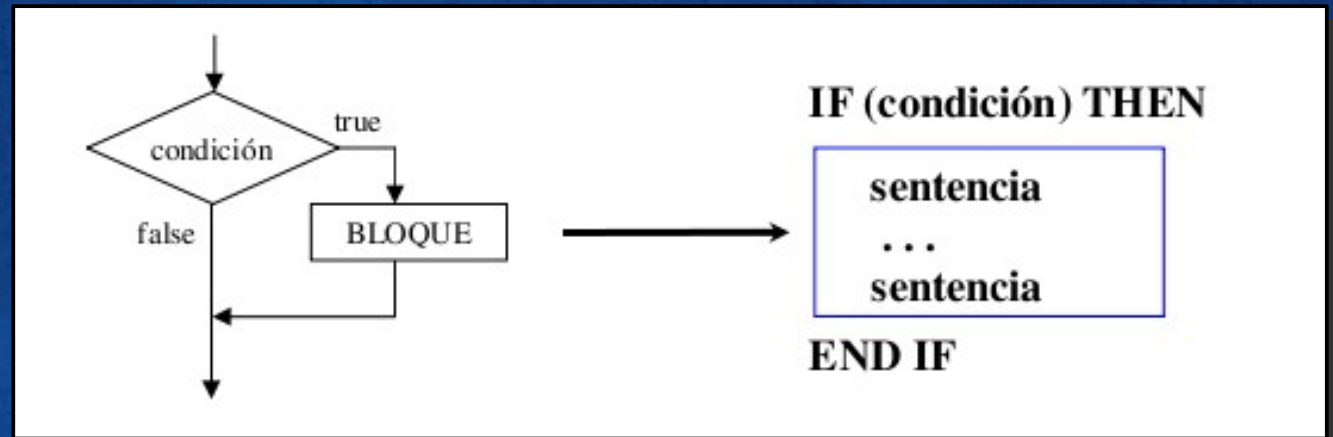
[sección ejecutable]

[sección de sub-programas internos]

END [PROGRAM [nombre de programa]]

IF

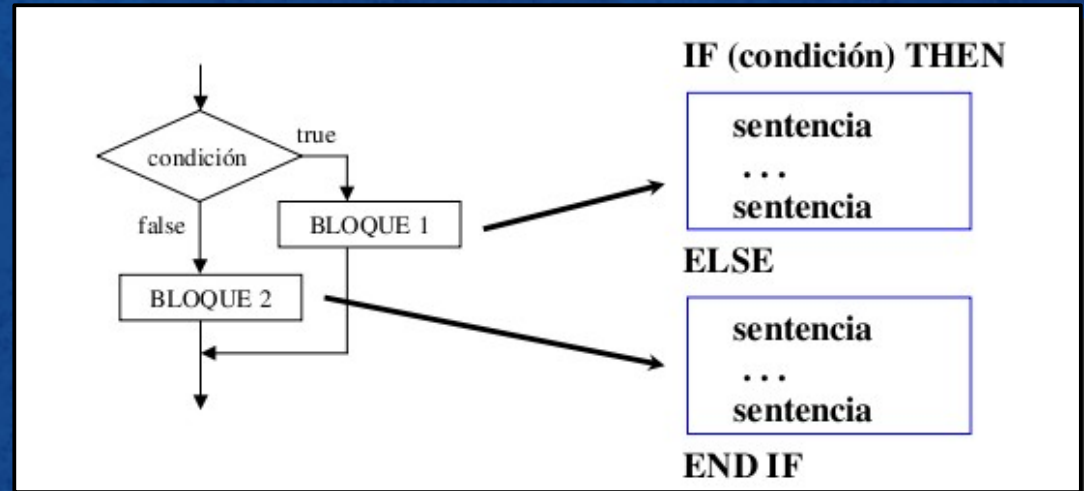
Ejemplo:



```
IF (a < b ) THEN
    aux = a
    a = b
    b = aux
END IF
```


IF THEN

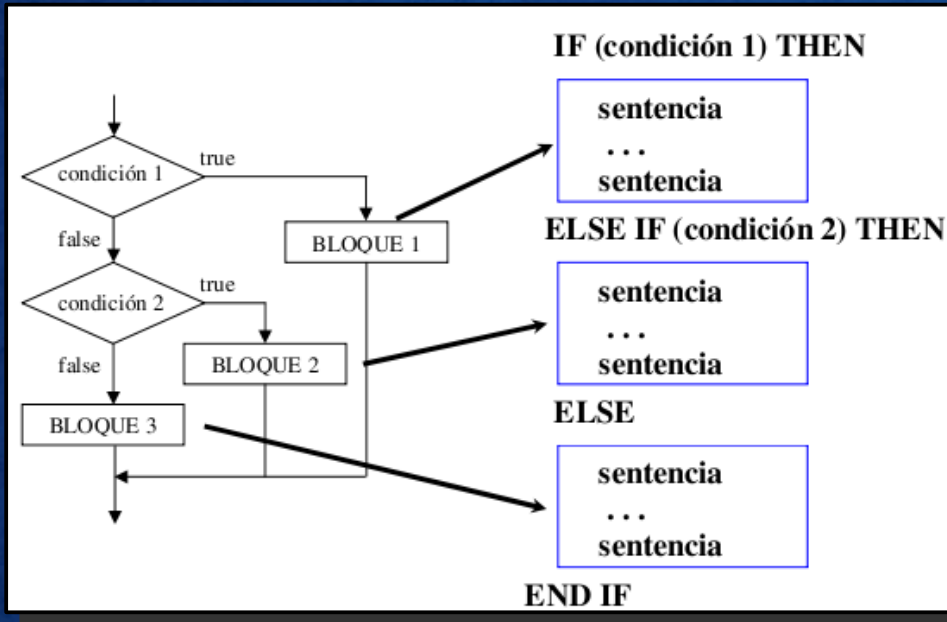
Ejemplo:



```
IF (leftCornerX < 0) THEN  
    leftCornerX = 0  
ELSE  
    aux = leftCornerX  
    leftCornerX = rightCornerX  
    rightCornerX = aux  
END IF
```

IF ELSE IF

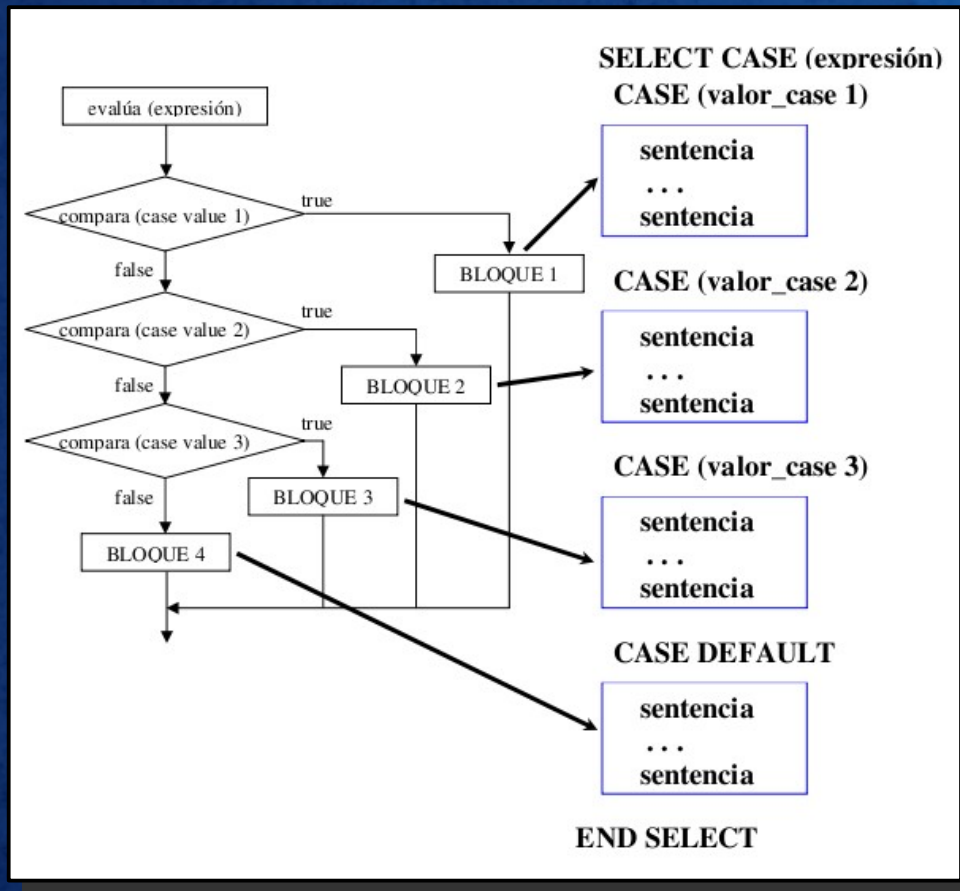
Ejemplo:



```
IF (kWatts < 50) THEN
    costo = 30
ELSE IF (kWatts < 100) THEN
    costo = 20+ 0.5*kWatts
ELSE IF (kWatts < 150) THEN
    costo = 15+ 0.3*kWatts
ELSE IF (kWatts < 200) THEN
    costo = 5+ 0.2*kWatts
ELSE
    costo = 0.15*kWatts
END IF
```


SELECT CASE

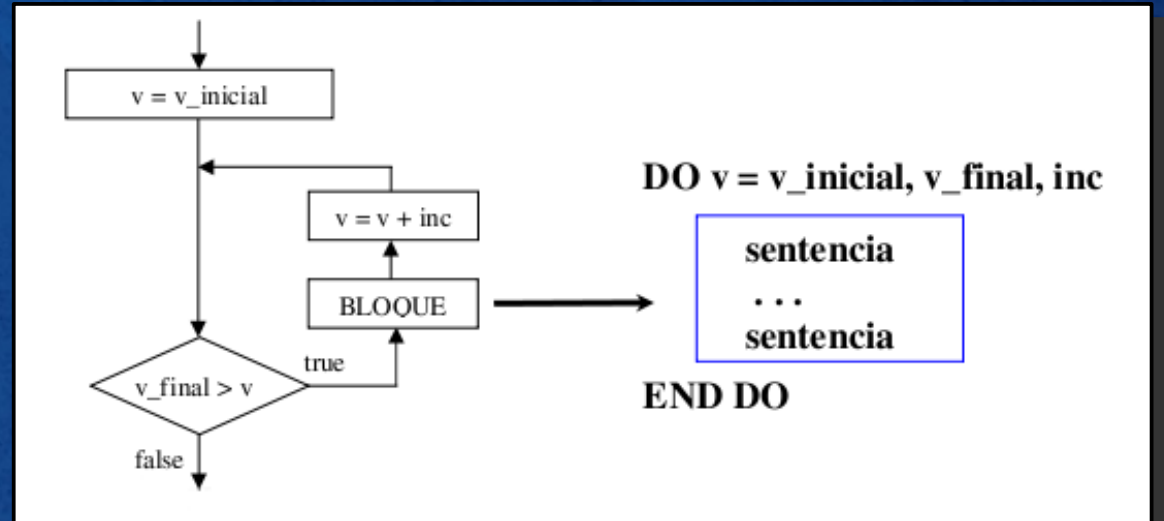
Ejemplo:



```
SELECT CASE (kWatts)
CASE (:49)
    costo = 30
CASE (50:99)
    costo = 20 + 0.5*kWatts
CASE (100:149)
    costo = 15 + 0.3*kWatts
CASE (150:199)
    costo = 5 + 0.2*kWatts
CASE DEFAULT
    costo = 0.15*kWatts
END SELECT
```

DO

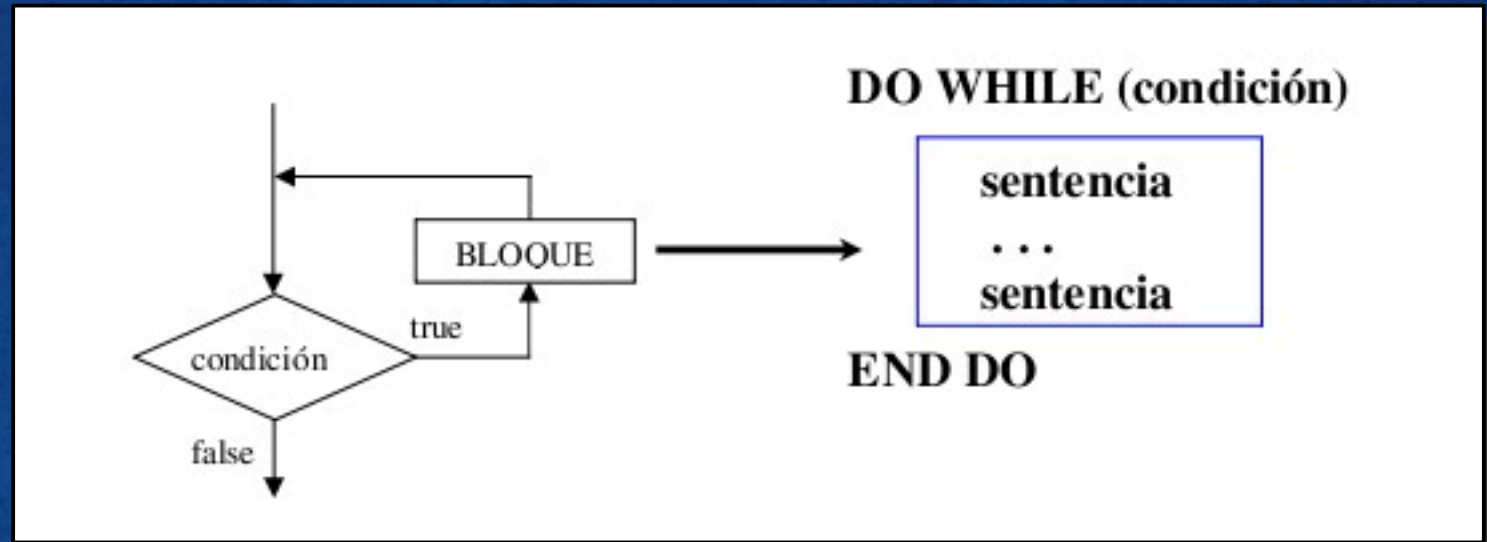
Ejemplo:



```
DO fila=1, maxFilas, 2
  DO col=1, maxCols, 3
    matriz(fila, col) = fila+2*col
  END DO
END DO
```


DO WHILE

Ejemplo:



```
DO WHILE (sigue != 'n')  
  WRITE (*, 'Desea continuar ?')  
  READ(*, ""), sigue  
END DO
```

Sugerencias de Programación

- Comenzar creando un proyecto para cada programa.
- Así cada programa poseerá su propia carpeta.
- Incluir en la misma carpeta los archivos de datos y los scripts para graficar con el GNUPLOT.
- Para ganar tiempo mientras prueban un programa o en un parcial, escriban el conjunto de datos (si no son muchos) dentro del mismo programa.
- Si los datos son muchos, generen un archivo de texto y leanlos desde el archivo. Las rutinas de ingreso de datos y validación son muy lindas pero pueden hacer perder mucho tiempo, si el conjunto de datos debe reingresarse muchas veces.

Al programar apliquen



Metodología K I S S

Kkeep

It

Simple &

Short

Arreglos

- Los **arreglos** (Arrays) se utilizan para almacenar un conjunto de variables del **mismo tipo**.
- El espacio de memoria para almacenar los arreglos **ESTATICOS** es fijo y se reserva en tiempo de compilación.
- **FORTRAN** permite la utilización de arreglos **DINÁMICOS**. Esto permite optimizar recursos y reservar sólo la cantidad de memoria necesaria en cada momento.

Arreglos Estáticos

- El número de dimensiones de un arreglo es denominado **rango** (rank) y el número total de elementos es el **tamaño** (size) del arreglo.
- La **forma** (shape) de un arreglo está determinada por su rango y la extensión de cada dimensión.

REAL matriz (10, 3, 2)

REAL arreglo (0:9, -1:1, 4:5)

Arreglos Dinámicos

```
REAL(8), ALLOCATABLE :: A(:, :), B(:)
```

```
INTEGER i, j, n
```

```
READ (*, *), n
```

```
ALLOCATE(A(n, n), B(n))
```

```
DO i=1, n
```

```
    B(i) = LOG(REAL(i))
```

```
    DO j=1, n
```

```
        A(i, j) = SQRT(REAL(i+j))
```

```
    END DO
```

```
END DO
```

```
DEALLOCATE (A, B)
```

Los Arreglos Dinámicos se especifican como **ALLOCATABLE**.

El espacio de memoria se reserva en tiempo de ejecución por medio de la instrucción **ALLOCATE**.

Dicho espacio se libera por medio de la instrucción **DEALLOCATE**.

Asignación de Valores

- Para referenciar a un arreglo se utiliza el **nombre** declarado en su definición.
- La asignación de un valor a dicho nombre significa que se asignará dicho valor a **todos** sus elementos.

```
REAL A( 1 0 ), B( 1 0 ), C( 1 0 )
```

```
A = 3.0
```

```
B = SQRT(A)
```

```
C = A + B
```


Operaciones con Arreglos

Es posible realizar ciertas **operaciones** sobre los elementos de un arreglo.

Por ejemplo:

REAL A (3,3), B(3,3), C(3,3)

A = 2.7

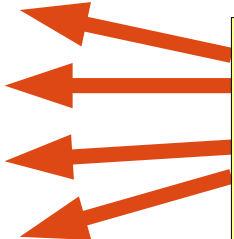
B = 4.2

C = A + B

C = A - B

C = A * B

C = A / B



Tener en cuenta que estas operaciones se realizan **sólo** elemento a elemento.

Referencias a un elemento

- Los **elementos** de un arreglo se referencian por medio de **subíndices**.
- Los valores de estos subíndices pueden ser **constantes, variables ó expresiones**.

```
REAL A ( 3,3 ), B( 3,3 ), C( 89 ), R
```

```
B(2, 2) = 4.5
```

```
R = 7.1
```

```
C( INT(R)*2 + 1 ) = 2.7
```

```
A(1, 2) = B( INT( C(15) ), INT(SQRT(R)))
```


Como referenciar varios elementos

- **FORTRAN** permite referenciar conjuntos de elementos de un arreglo.
- Esta característica, denominada **DO-IMPLICITO** simplifica enormemente el código.

```
REAL A ( 10, 10 )  
INTEGER i  
  
DO i=1, 10  
    A(i, :) = A(i, :)/A(i,i)  
    A(i, 4:8) = A(i, 4:8)*2  
    A(i, :4) = A(i, :8:2)/3.0  
END DO  
A(3, :) = A(3, 10:1:-1) + 2
```

Archivos en FORTRAN

- Los archivos permiten almacenar datos para que puedan ser utilizados por otros programas ó simplemente para resguardarlos una vez finalizado el programa que los generó y posteriormente recuperarlos fácilmente.
- Los **archivos de texto con formato**, son muy sencillos de generar y recuperar, y muchos programas son capaces de importarlos con facilidad. Además de brindarnos la posibilidad de crearlos, examinarlos y modificarlos con un simple **editor de textos**.

OPEN

```
OPEN ( [UNIT=] unidad [, ACCESS=acceso] [, ACTION=accion]  
[, DELIM=delim] [, ERR=err] [, FILE=archivo]  
[, STATUS=estado] )
```

- La sentencia **OPEN** nos permite abrir un archivo y vincularlo con un **nro. de unidad de entrada-salida**, para poder referenciarlo posteriormente en las operaciones de **lectura ó grabación**.
- Esta sentencia permite además especificar otras características como el **tipo de archivo**, la forma de **acceso**, etc.

UNIT (OPEN)

```
OPEN ( [UNIT=] unidad [, ACCESS=acceso] [, ACTION=accion]  
[, DELIM=delim] [, ERR=err] [, FILE=archivo]  
[, STATUS=estado] )
```

UNIT es el número de unidad que especifica el dispositivo lógico asociado (conectado) a un archivo externo físico (existente en un medio externo). Este especificador es un **número entero no negativo**.

Utilizar un * en las operaciones de lectura y/o grabación significa que se harán sobre los dispositivos **estándar** de entrada-salida, es decir, el teclado ó la pantalla.

ACCESS (OPEN)

```
OPEN ( [UNIT=] unidad [, ACCESS=acceso] [, ACTION=accion]  
[, DELIM=delim] [, ERR=err] [, FILE=archivo]  
[, STATUS=estado] )
```

Es posible acceder a los archivos en forma **secuencial**, **directa** ó en una modalidad que permite **agregar** datos al final del mismo, esto se especifica por medio del parámetro **ACCESS** y sus posibles valores son :

SEQUENTIAL, DIRECT y APPEND.

ACTION (OPEN)

```
OPEN ( [UNIT=] unidad [, ACCESS=acceso] [, ACTION=accion]  
[, DELIM=delim] [, ERR=err] [, FILE=archivo]  
[, STATUS=estado] )
```

Un archivo puede utilizarse para **lectura**, **grabación** ó para **lectura-grabación**, esto se especifica por medio del parámetro **ACTION**, siendo sus posibles valores:

READ, **WRITE** ó **READWRITE**.

STATUS (OPEN)

```
OPEN ( [UNIT=] unidad [, ACCESS=acceso] [, ACTION=accion]  
[, DELIM=delim] [, ERR=err] [, FILE=archivo]  
[, STATUS=estado] )
```

Si se utiliza **STATUS=REPLACE**, el archivo será creado si no existe y si existe se lo reemplazará por uno nuevo.

Los archivos temporarios se especifican por medio del parámetro **STATUS=SCRATCH**, de esta forma, se borrarán una vez finalizado el programa.

Instrucciones de Entrada-Salida

- Las instrucciones de **Entrada-Salida** de **FORTRAN** permiten la interacción con diversos dispositivos. Esta interacción se realiza principalmente por medio de las instrucciones **READ** y **WRITE**.
- Si bien es posible utilizar la salida standard por medio de la instrucción **PRINT**, las instrucciones **READ** y **WRITE**, permiten una mayor flexibilidad, sobre todo en lo que se refiere al formato de los datos.

WRITE

WRITE ([UNIT=] unidad [, **FMT=**] fmt]
[, ERR=err]) lista de salida

La instrucción **WRITE** permite la **especificación de formato** para los datos. Esto permite generar archivos de datos con formatos standard para que puedan ser leídos por otros programas, como por ejemplo, **GNU PLOT**.

Grabación de Archivos

```
PROGRAM grabaCoord  
  IMPLICIT NONE  
  INTEGER n, i  
  REAL(8) x, y, z
```

!Abre un archivo, si no existe lo crea, si existe lo reemplaza

```
OPEN(UNIT=2, FILE='coordenadas',STATUS='REPLACE')
```

```
WRITE (*, '(A28)', ADVANCE='NO'), 'Ingrese cantidad de puntos: '
```

```
READ *, n
```

```
DO i=1, n
```

```
  WRITE (*, '(A2, I3, A4)', ADVANCE='NO') 'X(', i, ') = '
```

```
  READ *, x
```

```
  WRITE (*, '(A2, I3, A4)', ADVANCE='NO') 'Y(', i, ') = '
```

```
  READ *, y
```

```
  WRITE (*, '(A2, I3, A4)', ADVANCE='NO') 'Z(', i, ') = '
```

```
  READ *, z
```

! Graba las coordenadas en el archivo

```
  WRITE (2, '(3F15.5)') x, y, z
```

```
END DO
```

! Cierra el archivo

```
CLOSE (2, STATUS='KEEP')
```

```
END PROGRAM
```


READ

READ ([UNIT=] unidad [, **FMT=**] fmt]
[, ERR=err][, END=endLabel]) lista de entrada

- Cuando se leen datos desde un archivo es necesario que el **formato** sea el adecuado.
- Una práctica común, es utilizar el **mismo** formato con el que fueron grabados.

Lectura de Archivos

```
PROGRAM leeCoord  
  IMPLICIT NONE
```

```
  INTEGER n, i, ioError  
  REAL(8) x, y, z
```

```
  !Abre un archivo, si no existe lo crea, si existe reemplaza el anterior
```

```
  OPEN(UNIT=2, FILE='coordenadas')
```

```
  DO WHILE (.TRUE.)
```

```
    ! Lee las coordenadas desde el archivo
```

```
    READ (2,'(3F15.5)', IOSTAT=ioError) x, y, z
```

```
    IF (ioError == 0) THEN
```

```
      ! Imprime las coordenadas en pantalla
```

```
      WRITE (*,'(3F15.5)') x, y, z
```

```
    ELSE
```

```
      EXIT      ! Sale fuera del ciclo DO
```

```
    ENDIF
```

```
  END DO
```

```
  ! Cierra el archivo
```

```
  CLOSE (2, STATUS='KEEP')
```

```
END PROGRAM
```


CLOSE

```
CLOSE( [UNIT=]unidad [, STATUS =stat]  
      [,ERR=errLabel][, IOSTAT=var] )
```

- Siempre es conveniente **cerrar** un archivo, al finalizar su uso, ya que esto **libera recursos** del sistema.
- Para **conservar** los datos del archivo se utiliza el especificador **STATUS="KEEP"**.

Especificación de Formato

Algunas de las especificaciones de formato más utilizadas son:

- A** - Texto.
- D** - Números Reales de doble precisión, notación exponencial.
- E** - Números Reales, notación exponencial.
- ES** - Números Reales, notación ingenieril.
- I** - Números Enteros.
- X** - Espacio (*space*).
- /** - Salto de Línea (*newline*).

Ejemplo:

```
WRITE(*, '(A, I5, ES10.4)' ADVANCE='NO'), "EJEMPLO: ", nro, valor
```


Especificación de Formato

Ejemplo :

```
INTEGER i
REAL(8) x, y

OPEN (2, FILE = "entrada", ACCESS = 'SEQUENTIAL')
OPEN (3, FILE = "salida", ACCESS = 'SEQUENTIAL', STATUS = 'REPLACE')

DO i=1, 10
  READ ( 2, '(2F10.4)' ) x, y
  WRITE (3, '(A6, F15.6, A2)' ) "z = [ ", sin(x)*cos(y), " ]"
END DO

CLOSE(2)
CLOSE(3)
```

Procedimientos en FORTRAN

Cuando en **FORTRAN** hablamos de procedimientos en forma genérica, nos estamos refiriendo tanto a **subrutinas** como a **funciones**. En este contexto, podemos diferenciar cuatro tipos de **procedimientos**:

- ***EXTERNOS***
- ***INTERNOS***
- ***INTRÍNSECOS***
- ***MÓDULOS***

Procedimientos Externos

Los **procedimientos externos** son **funciones** o **subrutinas** que escribe el programador y que se encuentran **fuera** del programa principal.

Los mismos pueden almacenarse en **archivos fuente separados** o pueden incluirse dentro del **mismo archivo** del programa principal, pero luego de la sentencia **END**.

Los **procedimientos externos** pueden contener a su vez funciones y procedimientos **internos**, los que se situarán luego de una sentencia **CONTAINS** y antes del final del **procedimiento**.

Procedimientos Internos

Los **procedimientos internos** son **funciones** o **subrutinas** que se encuentran dentro del programa principal, luego de la sentencia **CONTAINS**. Dichas subrutinas o funciones sólo pueden ser **invocadas** por el programa que las contiene.

Por otra parte los **procedimientos internos** son iguales a los **procedimientos externos** excepto por:

- *Los nombres de los procedimientos internos **son locales**. No globales.*
- *No se puede utilizar un **procedimiento interno** como **argumento actual** al invocar a otro procedimiento.*

Procedimientos Intrínsecos

Los procedimientos **intrínsecos** son **funciones** o **subrutinas** predefinidas por el lenguaje **FORTRAN** y por lo tanto son **automáticamente enlazados** al programa.

Entre otras cosas, los procedimientos intrínsecos realizan **conversiones de datos** y **devuelven información** sobre tipos de datos, ejecutan **operaciones sobre datos numéricos** y **caracteres**, controlan el final de los archivos, ejecutan **operaciones de bit**, etc.

```
HUGE(n)  
BIT_SIZE(i)  
ALLOCATED(A)  
CALL RANDOM_SEED
```

Módulos (ver en Programación Avanzada)

- Es posible agrupar **Subrutinas** y **Funciones** relacionadas, definiéndolas dentro de un **módulo**.
- El código se situará luego de la sentencia **MODULE** y dentro de sentencias **CONTAINS** y **END**.
- Cualquier programa que desee acceder a los **procedimientos públicos** definidos en el **módulo** deberá incluir la sentencia **USE** con el nombre del módulo correspondiente.

Subrutinas

```
SUBROUTINE nombresub [( [ lista de argumentos] )]  
    [sentencias de declaración]  
    [sentencias ejecutables]  
END [SUBROUTINE [nombresub]]
```

- Para invocar a una subrutina se utiliza la sentencia **CALL**.
- Una subrutina no devuelve directamente un valor.
- Cuando es invocada, una subrutina ejecuta el conjunto de acciones definido por sus sentencias ejecutables.

Subrutinas

Ejemplo:

```
SUBROUTINE Intercambio(a,b)
! Declaracion de argumentos
REAL a,b

! Declaracion de variables auxiliares
REAL aux

! Cuerpo de la subrutina
aux=a
a=b
b=aux

END SUBROUTINE Intercambio
```


Subrutinas

Ejemplo de Invocación:

```
PROGRAM MUESTRA  
INTEGER, PARAMETER:: n=3
```

```
CALL sub_A(n) ! llamada a la subrutina
```

```
PRINT *, n
```

```
CONTAINS
```

```
SUBROUTINE sub_A(a)
```

```
INTEGER a
```

```
...
```

```
END SUBROUTINE
```

```
END PROGRAM
```

Funciones

```
FUNCTION nombreFunc [( [ lista de argumentos] )]  
    [sentencias de declaración]  
    [sentencias ejecutables]  
END [FUNCTION [nombreFunc]]
```

- Una función se invoca directamente por su nombre y lista de argumentos.
- Una función devuelve directamente un valor.
- El valor de retorno de una función puede combinarse como parte de expresiones más complejas.

Funciones

Declaración:

```
FUNCTION factorial(n)  
INTEGER factorial, n, aux
```

```
aux = 1
```

```
DO i=2, n  
    aux = i*aux  
END DO
```

```
factorial = aux
```

```
END FUNCTION
```

Funciones

Retorno de arreglos:

```
FUNCTION hilbert(n)
INTEGER i,j,n
REAL, DIMENSION(n,n) :: hilbert

DO i=1, n
  DO j=1, n
    hilbert(i, j)=1.0/(i+j+1)
  ENDDO
ENDDO

END FUNCTION
```


INTENT

- **INTENT** especifica el uso intencional de los parámetros formales dentro de un procedimiento.
- El uso del atributo **INTENT** protege de acciones no deseadas.
- Si el argumento es **INTENT(IN)**, puede suministrarse una expresión o constante del tipo requerido en el parámetro actual.
- Si este es **INTENT(OUT)** o **INTENT(INOUT)**, se debe suministrar una variable en el parámetro actual.

INTENT

Ejemplo de declaración:

```
SUBROUTINE intercambio (a, b)  
  INTENT (INOUT) :: a, b  
  . . . . .
```

Otro ejemplo:

```
SUBROUTINE inversa(A,singular)  
  REAL, DIMENSION(:,:), INTENT (INOUT) :: A  
  LOGICAL; INTENT(OUT) :: singular  
  . . . . .
```


Invocación a programas externos

Para invocar a un programa externo, se utiliza la subrutina intrínseca **SYSTEM**.

Por ejemplo:

```
CALL SYSTEM ("clear")
```

Invoca al programa "clear" del **Sistema Operativo** para borrar la pantalla.

Gráficos con GNUPLOT

- **GNUPlot** es un programa de visualización de datos, el cuál puede utilizarse de forma **interactiva** desde una terminal ó desde un programa invocando a un **script**.
- Un script es un archivo con un **conjunto de instrucciones** que son interpretadas por el **GNUPlot** para realizar un **gráfico** determinado.
- Los **scripts** permiten generar **visualizaciones de datos** de gran calidad, en forma muy sencilla.

Ejemplo de Script (I)

```
set autoscale          # escala los ejes automaticamente
unset log              # quita la escala logaritmica
unset label            # quita los titulos anteriores
set xtic auto          # establece las divisiones del eje x
set ytic auto          # establece las divisiones del eje y
```

```
set title "Valores de la Función"
set xlabel "x"
set ylabel "f(x)"
```

```
plot "misDatos.dat" using 1:2 title 'Seno(x)' with lines
```

Invocar GNUPLOT desde FORTRAN

```
PROGRAM gnuplotTest
```

```
    IMPLICIT NONE
```

```
    REAL, PARAMETER:: pi=3.1415926537
```

```
    REAL(8) x
```

```
    INTEGER i
```

```
    OPEN (2, FILE = 'misDatos.dat')
```

```
    DO i=0, 99
```

```
        x=i*pi/100.0
```

```
        WRITE(2, '(3F10.6)') x, sin(x), cos(x)
```

```
    END DO
```

```
    CLOSE (2, STATUS='KEEP')
```

```
    CALL SYSTEM(" gnuplot -persist 'prueba.p' ")
```

```
    READ(*,*)
```

```
END
```


Ejemplo prueba.p

```
set autoscale          # escala los ejes automaticamente
unset log              # quita la escala logaritmica
unset label            # quita los titulos anteriores
set xtic auto          # establece las divisiones del eje x
set ytic auto          # establece las divisiones del eje y
```

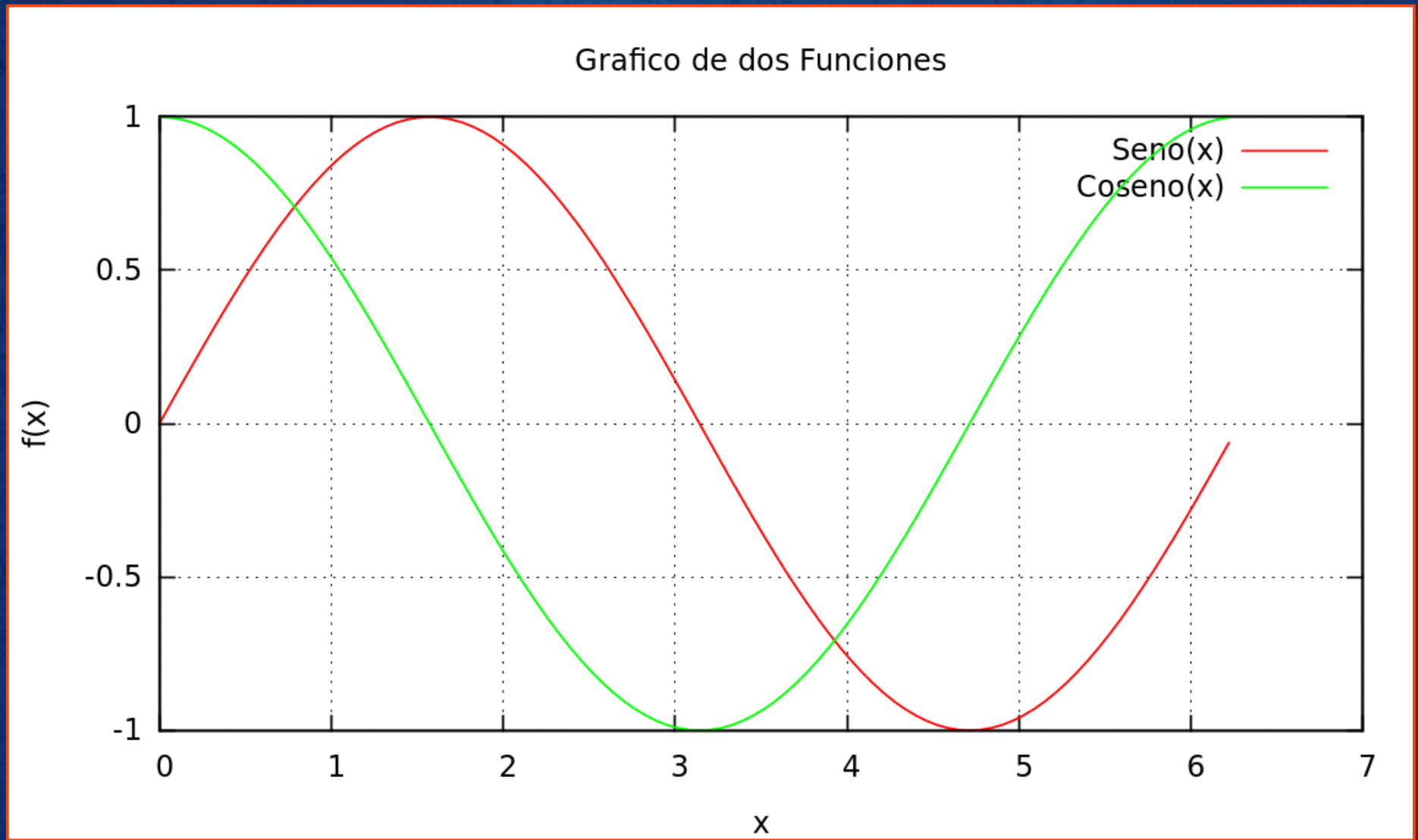
```
set title "Grafico de dos Funciones"
```

```
set xlabel "x"
```

```
set ylabel "f(x)"
```

```
plot "prueba.dat" using 1:2 title 'Seno(x)' with lines,\
      "prueba.dat" using 1:3 title 'Coseno(x)' with lines
```

Gráfico obtenido



Ejemplo de Script (II)

```
# Ejemplo de script Gnuplot, para graficar los datos
# que se encuentran en varios archivos"
# Este archivo se llama spline.p
set autoscale          # escala los ejes automaticamente
unset log              # quita la escala logaritmica (si la hubiera)
unset label            # quita los titulos anteriores
set xtic auto          # establece automaticamente las divisiones del eje x
set ytic auto          # establece automaticamente las divisiones del eje y
set grid
set title "Aproximación por Splines cúbicos"
set xlabel "valores de x"
set ylabel "valores de y"
```

```
plot "curva1.dat" using 1:2 title 'trazador_1' with lines lw 2, \
    "puntos1.dat" using 1:2 title " with points ls 2 lw 3 pt 7, \
    "curva2.dat" using 1:2 title 'trazador_2' with lines lw 2, \
    "puntos2.dat" using 1:2 title " with points ls 2 lw 3 pt 7, \
    "curva3.dat" using 1:2 title 'trazador_3' with lines ls 4 lw 2, \
    "puntos3.dat" using 1:2 title 'datos' with points ls 2 lw 3 pt 7
```

Gráfico combinando diferentes archivos

```
plot "curva1.dat" using 1:2 title 'trazador_1' with lines lw 2,\n  "puntos1.dat" using 1:2 title "" with points ls 2 lw 3 pt 7,\n  "curva2.dat" using 1:2 title 'trazador_2' with lines lw 2,\n  "puntos2.dat" using 1:2 title "" with points ls 2 lw 3 pt 7,\n  "curva3.dat" using 1:2 title 'trazador_3' with lines lw 2,\n  "puntos3.dat" using 1:2 title 'datos' with points ls 2 lw 3 pt 7
```

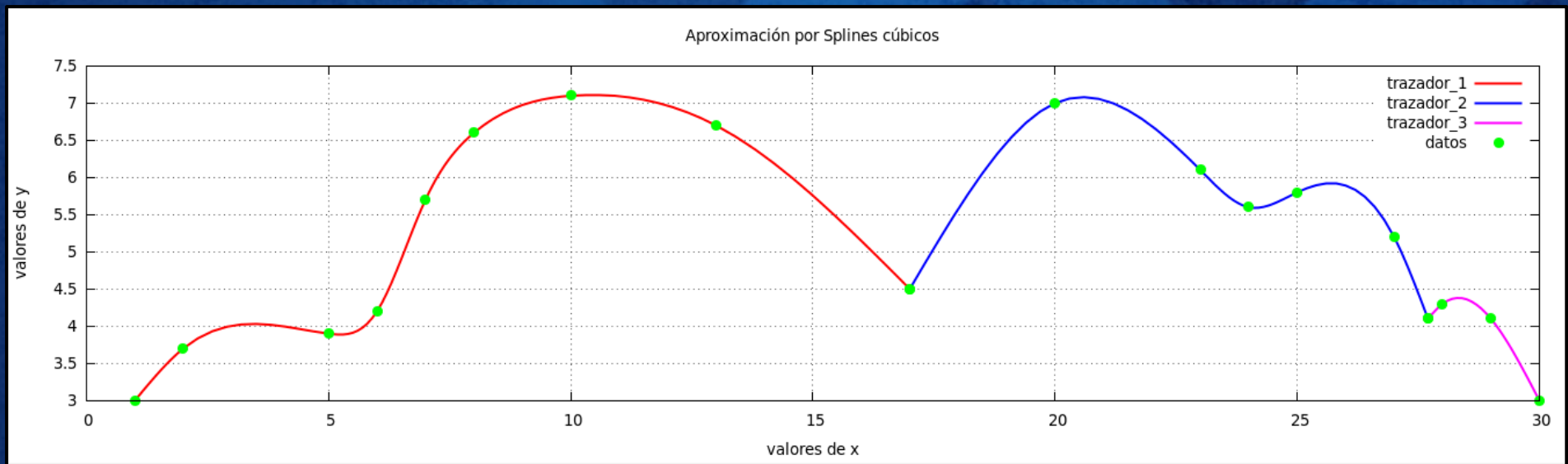
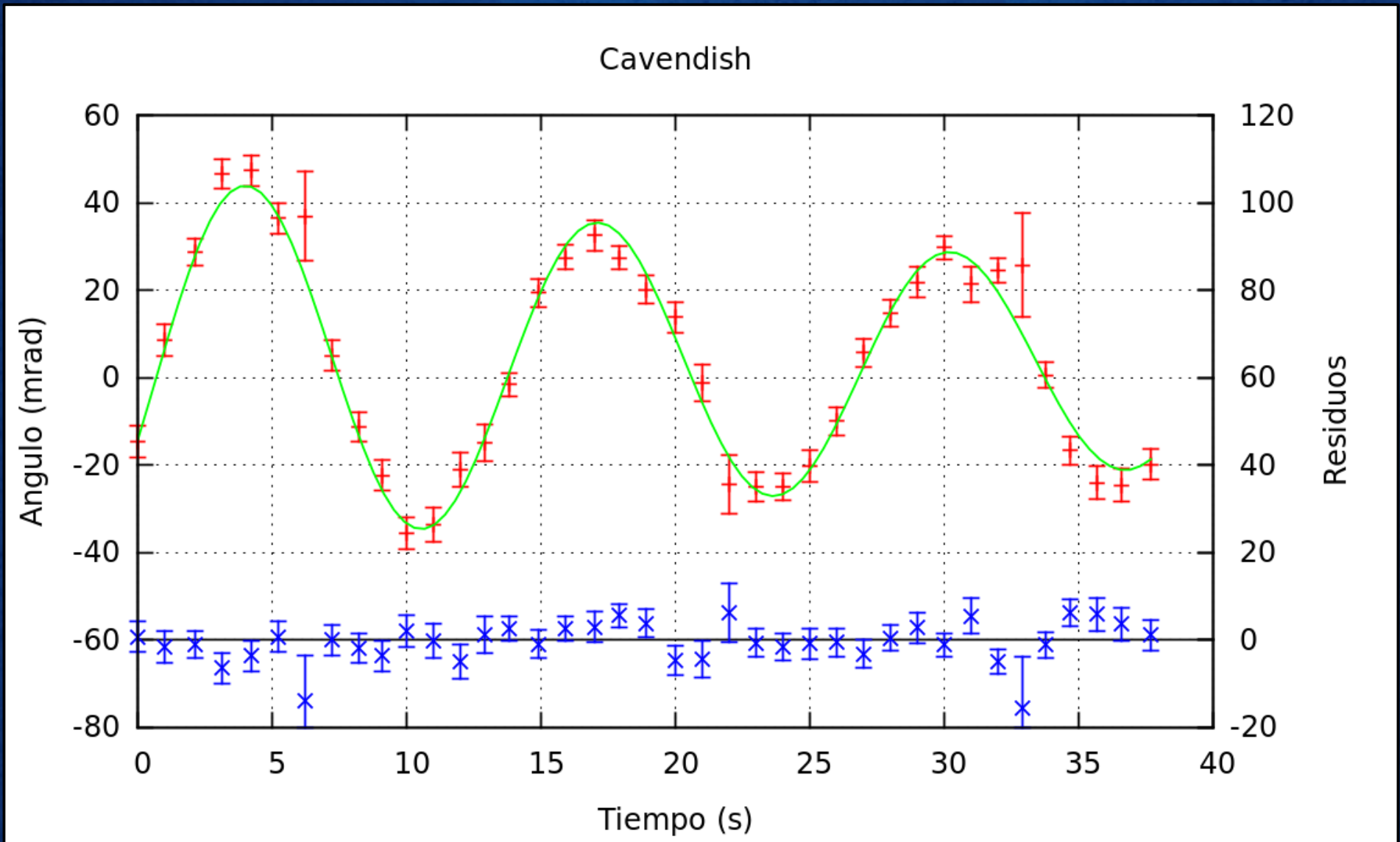


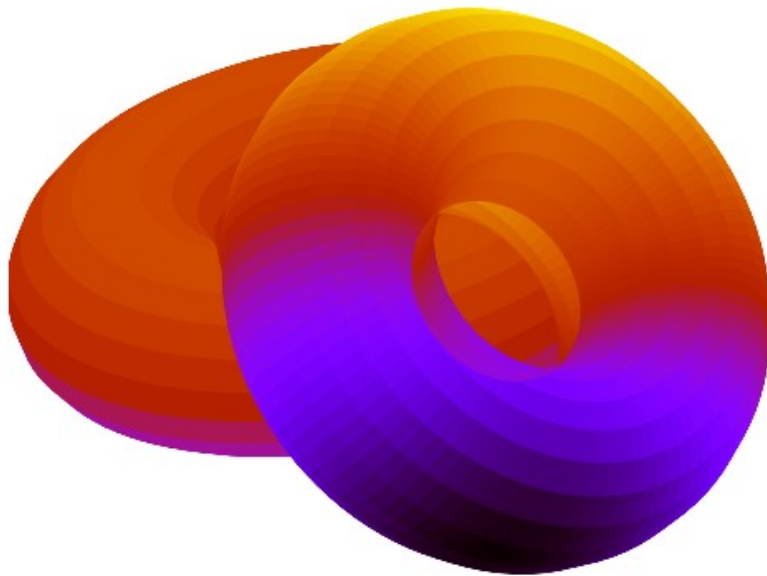
Grafico de Errores con Gnuplot



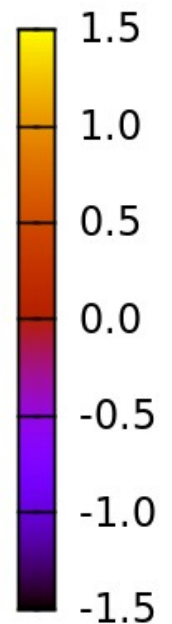
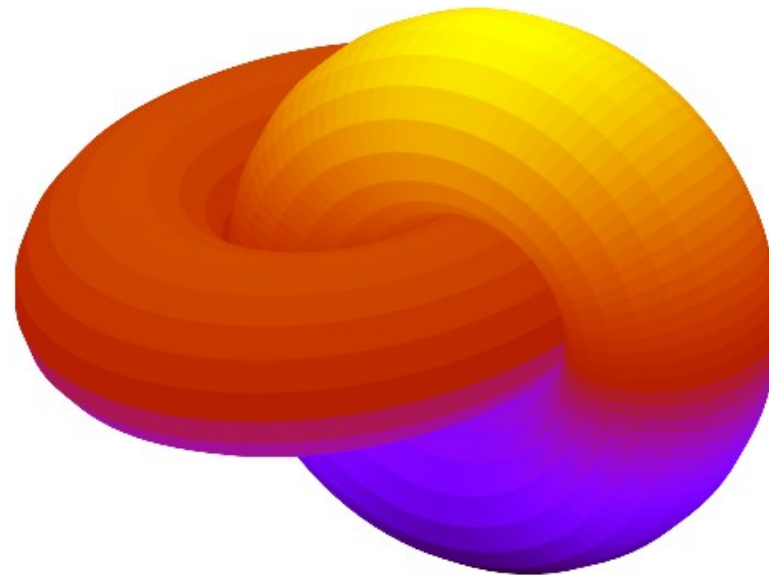
Graficos 3D con Gnuplot

Toroide Entrelazado

PM3D superficie
sin ordenamiento
por profundidad



PM3D superficie
ordenada
por profundidad



CUESTIONES PARA INVESTIGAR . . .

- ¿ Puede una función FORTRAN retornar más de un valor ? ¿ De qué forma ?
- ¿ Puede pasarse el nombre de un procedimiento como un parámetro de otro procedimiento ?
- ¿ Qué función GNUPLOT realiza un gráfico 3D a partir de un conjunto de puntos almacenados en un archivo ?
- ¿ Cómo deben estar grabados los datos en dicho archivo para que GNUPLOT pueda graficarlo adecuadamente ?

PROGRAMACIÓN AVANZADA

A encontrarán algunos temas de programación más avanzada.

Si bien es posible programar todos los algoritmos correspondientes a los métodos que analizamos en la asignatura sin la necesidad de utilizar las técnicas que se muestran a continuación, las mismas representan un conocimiento más avanzado de la programación.

Por lo tanto, es una sección sólo para espíritus inquietos, curiosos y valientes. Vamos, adelante !!!

Especificación de Formato Variable

Ejemplo 2 :

```
PROGRAM grabaVector
```

```
REAL(8), ALLOCATABLE :: v(:)
```

```
CHARACTER*15 formato, filename
```

```
INTEGER n=6
```

```
WRITE (*, '(A30)', ADVANCE = 'NO') "Ingrese el nombre del archivo: "
```

```
READ filename
```

```
WRITE (*, '(A30)', ADVANCE = 'NO') "Ingrese el orden del vector: "
```

```
READ n
```

```
! Reserva espacio para v
```

```
ALLOCATE(v(n))
```

```
v = 0
```

Sigue en la
próxima página

Especificación de Formato Variable

Ejemplo 2 : (Continuación)

! Especifica el formato de grabación en una variable

```
WRITE (formato,'(A4,I2,A2,I2,A1,I1,A1)' ) "(I5,", n+1, "F", 15, ".", 6, ")"
```

! Abre el archivo

```
OPEN (2, FILE = filename, ACCESS = 'SEQUENTIAL', STATUS = 'REPLACE')
```

```
DO i=1, 10
```

```
  v(:n-1) =v(2:n)+SQRT(i)
```

```
  WRITE (2, formato) i, v
```

```
END DO
```

```
CLOSE (2, STATUS='KEEP')
```

```
DEALLOCATE(v)
```

```
END PROGRAM
```


Módulos

- Es posible agrupar **Subrutinas** y **Funciones** relacionadas, definiéndolas dentro de un **módulo**.
- El código se situará luego de la sentencia **MODULE** y dentro de sentencias **CONTAINS** y **END**.
- Cualquier programa que desee acceder a los **procedimientos públicos** definidos en el **módulo** deberá incluir la sentencia **USE** con el nombre del módulo correspondiente.

Módulos (I)

```
MODULE matrixFunctions
```

```
  IMPLICIT NONE
```

```
  CONTAINS
```

```
  SUBROUTINE creaMatrizIdentidad(orden, matriz)
```

```
    !Crea una matriz identidad
```

```
    REAL(8), ALLOCATABLE :: matriz(:, :)
```

```
    INTEGER :: i, orden
```

```
    IF (ALLOCATED(matriz)) THEN
```

```
      DEALLOCATE(matriz)
```

```
    ENDIF
```

```
    ! Crea la matriz
```

```
    ALLOCATE(matriz(orden,orden))
```

```
    !Inicializa la matriz
```

```
    matriz = 0
```

```
    DO i=1, orden
```

```
      matriz(i,i) = 1
```

```
    ENDDO
```

```
  END SUBROUTINE creaMatrizIdentidad
```

Sigue en la
próxima página

Módulos (II)

```
SUBROUTINE imprimeMatriz(a)
!Imprime la Matriz
REAL(8) a(:, :)
INTEGER i,j,cantFilas, cantCols

cantFilas = SIZE(a,DIM=1)
cantCols = SIZE(a,DIM=2)
DO i=1,cantFilas
  DO j=1,cantCols
    WRITE (*,'(F12.4)', ADVANCE='NO') a(i,j)
  ENDDO
  WRITE (*,*)
ENDDO
END SUBROUTINE imprimeMatriz
END MODULE
```

Módulos (III)

Declaración de módulos en Programa Principal:

```
PROGRAM matrices
```

```
!Modulos utilizados
```

```
USE matrixFunctions
```

```
IMPLICIT NONE
```

```
! Código del Programa principal
```

```
...
```

```
END PROGRAM
```


Bibliotecas de código (Libraries)

- Una forma muy difundida de **reutilización de código**, es la utilización de bibliotecas de funciones y sub-rutinas **específicas**.
- **FORTRAN**, es tal vez, uno de los lenguajes que posee la **mayor cantidad de bibliotecas de funciones y sub-rutinas** para la resolución de problemas relacionados con múltiples áreas de la **matemática, la ciencia y la ingeniería**.
- Más información sobre las librerías LAPACK y BLAS en <http://www.netlib.org/lapack/explore-html/>

Biblioteca LAPACK

Linear Algebra *PACK*age

LAPACK: General M... x +

www.netlib.org/lapack/explore-html/d0/d86/group___g_e.html

Más visitados Getting Started

L A P A C K
L -A P -A C -K
L A P A -C -K
L -A P -A -C K
L A -P -A C K
L -A -P A C -K

LAPACK 3.7.0

LAPACK: Linear Algebra PACKage

Main Page Modules Data Types List Files

LAPACK

Modules

LAPACK

- General Matrices
- General Band Matrix
- Symmetric Matrix
- Hermitian Matrix
- Positive Definite Matrix
- General tridiagonal Matrix
- Positive Definite tridiagonal Matrix
- Eigenvalue
- Other Auxiliary Routines
- Other Computational Routines
- Other Solve Routines
- LAPACK Testing
- Reference BLAS
- Data Types List
- Files

General Matrices

LAPACK

Collaboration diagram for General Matrices:

```
graph LR; LAPACK --> GM[General Matrices]; GM --> LS[Linear Solve]; GM --> AR[Auxiliary routines]; GM --> VCR[Variants Computational routines]; GM --> EV[Eigenvalue]; GM --> SV[Singular Value]; GM --> CR[Computational routines];
```

Modules

- Linear Solve
- Eigenvalue
- Singular Value
- Computational routines
- Variants Computational routines
- Auxiliary routines

Detailed Description

This is the group of General Matrices routines

Generated on Fri Dec 23 2016 15:33:06 for LAPACK by [doxygen](#) 1.8.10

Biblioteca LAPACK

Linear Algebra *PACK*age

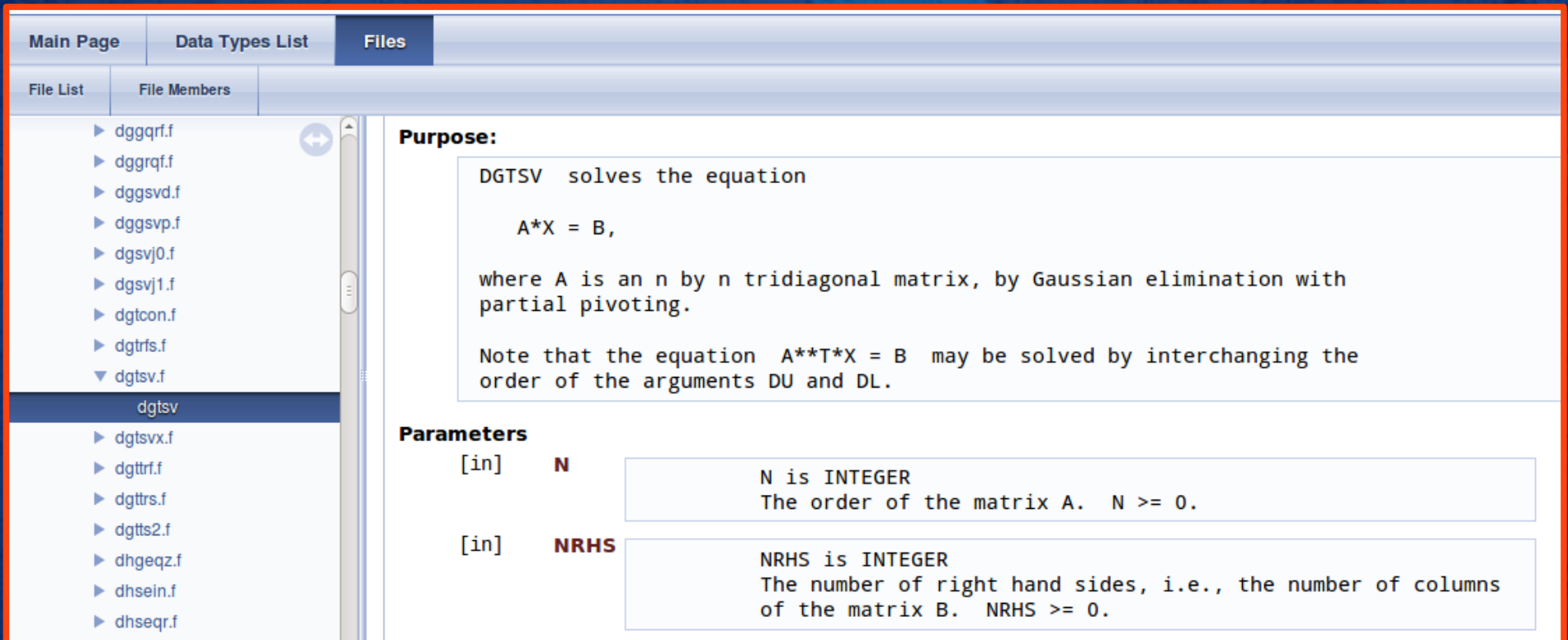
The screenshot shows a web browser displaying the LAPACK 3.5.0 source code for the `dgtsv` subroutine. The page is titled "LAPACK 3.5.0" and "LAPACK: Linear Algebra PACKage". The browser address bar shows the URL: `www.netlib.org/lapack/explore-html/d1/db3/dgtsv_8f.html#a2bf93f2ddefa5e671866eb2191dc19d4`. The page has a navigation bar with "Main Page", "Data Types List", and "Files". The "Files" tab is active, showing a list of files on the left and the source code for `dgtsv` on the right. The source code is as follows:

```
subroutine dgtsv ( integer                N,  
                  integer                NRHS,  
                  double precision, dimension( * ) DL,  
                  double precision, dimension( * ) D,  
                  double precision, dimension( * ) DU,  
                  double precision, dimension( ldb, * ) B,  
                  integer                LDB,  
                  integer                INFO  
)
```

Below the code, it states: "DGTSV computes the solution to system of linear equations $A * X = B$ for GT matrices". It provides download links for "Download DGTSV + dependencies [TGZ] [ZIP] [TXT]". The "Purpose:" section states: "DGTSV solves the equation $A * X = B$ ". The footer indicates the page was generated on Sat Nov 16 2013 17:29:02 for LAPACK by doxygen 1.8.3.1.

Subrutina DGTSV (LAPACK)

*La subrutina **DGTSV** resuelve un sistema de Ecuaciones Lineales Tri-Diagonal*



The screenshot shows the LAPACK website interface. The top navigation bar includes 'Main Page', 'Data Types List', and 'Files'. The 'Files' tab is selected. On the left, a 'File List' sidebar shows a directory of files, with 'dgtsv.f' expanded to show sub-files like 'dgtsvx.f', 'dgttrf.f', 'dgttrs.f', 'dgtts2.f', 'dhgeqz.f', 'dhsein.f', and 'dhseqr.f'. The main content area is titled 'Purpose:' and describes the DGTSV subroutine. It states that DGTSV solves the equation $A \cdot X = B$, where A is an n by n tridiagonal matrix, solved using Gaussian elimination with partial pivoting. A note mentions that the equation $A^T \cdot X = B$ can be solved by interchanging the arguments DU and DL . Below the purpose, the 'Parameters' section lists two input arguments: N (INTEGER, the order of the matrix A , $N \geq 0$) and $NRHS$ (INTEGER, the number of right hand sides, i.e., the number of columns of the matrix B , $NRHS \geq 0$).

Main Page | **Data Types List** | **Files**

File List | **File Members**

- ▶ dggqr.f
- ▶ dggqr.f
- ▶ dggsvd.f
- ▶ dggsvp.f
- ▶ dgsvj0.f
- ▶ dgsvj1.f
- ▶ dgtcon.f
- ▶ dgtrfs.f
- ▼ dgtsv.f
 - ▶ dgtsvx.f
 - ▶ dgttrf.f
 - ▶ dgttrs.f
 - ▶ dgtts2.f
 - ▶ dhgeqz.f
 - ▶ dhsein.f
 - ▶ dhseqr.f

Purpose:

DGTSV solves the equation

$$A \cdot X = B,$$

where A is an n by n tridiagonal matrix, by Gaussian elimination with partial pivoting.

Note that the equation $A^T \cdot X = B$ may be solved by interchanging the order of the arguments DU and DL .

Parameters

[in] **N** N is INTEGER
The order of the matrix A . $N \geq 0$.

[in] **NRHS** NRHS is INTEGER
The number of right hand sides, i.e., the number of columns of the matrix B . $NRHS \geq 0$.

Utilización de Librerías

```
PROGRAM triDiag
! Resuelve un sistema tri-diagonal

IMPLICIT NONE
INTEGER, PARAMETER :: n=3
REAL(8), DIMENSION(n) :: ID, dD, uD, x
INTEGER INFO


CALL ingVec(ID, "Diagonal Inferior  ")
CALL ingVec(dD, "Diagonal Principal  ")
CALL ingVec(uD, "Diagonal Superior  ")
CALL ingVec(x, "Terminos Independientes")

CALL DGTSV( n, 1, ID, dD, uD, x, n, INFO )

PRINT *, "La solucion es: ", x

CONTAINS
! Sigue en la próxima transparencia
```

Sigue en la
próxima página



Utilización de Librerías

```
SUBROUTINE ingVec(vec, title)
! Ingresar vector
REAL(8) vec(:)
INTEGER i,n
CHARACTER*22 title

CALL SYSTEM("clear")
PRINT *, "Ingrese ", title

n=SIZE(vec)
DO i=1, n
  WRITE(*,'(A10, I2, A4)', ADVANCE='NO') "Elemento (", i, " ): "
  READ (*,*) vec(i)
ENDDO
END SUBROUTINE

END PROGRAM
```


Preguntas . . .

