How to learn Applied Mathematics through modern FORTRAN

January 12, 2018

Contents

Chapter 1

How to write batch files or sh files

batch bsh

scripts

gfortran

make

Chapter 2

First examples: calculus and algebra

2.1 My first program: "Hello world"

2.2 sum of a numeric series

Dar el resultado de la suma de los 100 primeros términos de las siguientes series:

1. Serie de números naturales.

2. Serie de números naturales impares.

3. Serie numérica donde el término general de la serie es: $a_n = 1/n^2$ desde $n = 1$.

4. Serie numérica donde el término general de la serie es $a_n = 1/n!$ desde $n = 1$.

5. Serie numérica donde el término general de la serie es $a_n = (-1)^{n+1}/(2n-1)$ desde $n = 1$.

2.3 Operaciones con matrices y vectores

Considerar los vectores $V, W \in \mathbb{R}^N$ de componentes:

$$\{v_i = \frac{1}{i^2}, \quad i = 1 \ldots N\},$$

$$\{w_i = \frac{(-1)^{i+1}}{2i-1}, \quad i = 1 \ldots N\}.$$

Considerar la matriz $A \in \mathcal{M}_{N\times N}(\mathbb{R})$ donde su término genérico vale $a_{ij} = (i/N)^j$. Escribir un programa para calcular las operaciones siguientes con $N = 100$:

1. Suma de todas las componentes del vector V y del vector W.
2. Suma de todas las componentes de la matriz A.
3. Suma de las componentes del vector W mayores que cero.
4. Producto escalar de los vectores V y W.
5. Producto escalar del vector V y la columna N de la matriz A.
6. Suma de las componentes de vector que resulta de multiplicar la matriz A por el vector V.
7. Traza de la matriz A.

2.4 dynamic allocation of memory

Dada la matriz $A \in \mathcal{M}_{M\times M}(\mathbb{R})$ de término genérico

$$\{a_{ij} = (i/M)^j, \quad i = 0, \ldots M-1, \quad j = 0, \ldots M-1\}.$$

calcular las siguientes operaciones:

1. Calcular
$$\sum_{M=1}^{10} traza(A)$$
2. Calcular
$$\sum_{M=1}^{5} traza(A^2)$$
3. Calcular con $M = 4$
$$traza\left(\sum_{k=1}^{5} A^k\right)$$

2.5 Piecewise functions

Sean los vectores $X, F \in \mathbb{R}^{N+1}$. Las componentes de X almacenan los valores discretos del dominio de definición y F las imágenes correspondientes de la función $F : \mathbb{R} \to \mathbb{R}$ continua a trozos siguiente:

$$F(x) = \begin{cases} 1, & a \leq x \leq -\dfrac{\pi}{2}, \\ \cos(\pi x), & -\dfrac{\pi}{2} < x < \dfrac{\pi}{2}, \\ 0, & \dfrac{\pi}{2} \leq x \leq b. \end{cases}$$

Considerar una partición equiespaciada de la forma:

$$\{x_i = a + i\Delta x, \quad i = 0 \ldots N\}, \qquad \Delta x = \frac{b-a}{N}, \qquad a < -\frac{\pi}{2}, \qquad b > \frac{\pi}{2}.$$

Se pide calcular la suma;

$$S_N = \sum_{i=0}^{N} F_i \Delta x$$

1. con $N = 10$
2. con $N = 20$
3. con $N = 100$

2.6 Series of functions

Aproximar mediante un desarrollo en serie de potencias de la forma

$$f(x) = \sum_{k=0}^{M} a_k \; x^k, \qquad\qquad a_k = \frac{f^{(k)}(0)}{k!},$$

las funciones $F : \mathbb{R} \to \mathbb{R}$, siguientes:

1. $f(x) = e^x$ y calcular el valor $f(1)$ con $M = 5$.
2. $f(x) = \sin(x)$ y calcular el valor $f(\pi/2)$ con $M = 8$.
3. $f(x) = \cosh(x)$ y calcular el valor $f(1)$ con $M = 10$.
4. $f(x) = \dfrac{1}{1-x}$ y calcular el valor $f(0.9)$ con $M = 20$.
5. $f(x) = e^x$ y calcular el valor más preciso de $f(1)$ con doble precisión.
6. $f(x) = \sin(x)$ y calcular el valor más preciso $f(\pi/2)$ con doble precisión.
7. $f(x) = \cosh(x)$ y calcular el valor más preciso de $f(1)$ con doble precisión.
8. $f(x) = \dfrac{1}{1-x}$ y calcular el valor más preciso de $f(0.9)$ con doble precisión.

2.7 Reading and writing data files

Crear los ficheros de datos ForTran con nombres `input_1.dat` e `input_2.dat` con la información siguiente:

Contenido del fichero de entrada `input_1.dat` :

```
1      Datos de entrada 1
```

```
2
3       1.2       3.4      6.2     -14.0     0.1
4       -25.2     -8.6     5.1      9.9     17.0
5       -1.0      -2.0     -5.4     -8.6     0.0
6       3.14      -11.9    -7.0     -12.1    9.2
7       6.66      5.32     0.001    0.2      0.001
```

Contenido del fichero de entrada `input_2.dat` :

```
1       Datos de entrada 2
2
3       1.2       3.4      6.2     -14.0     0.1       4.89    7.54
4       -25.2     -8.6     5.1     12.0      9.9       12.24   17.0
5       0.0       34.5     -1.0    -2.0      -43.04    -8.6     0.0
6       3.14      -11.9    71.0    7.0       17.0      -12.1    9.2
7       6.66      5.32     0.001    0.2      0.001     0.008    -0.027
8       54.0      77.1     -9.002  -13.2     0.017     65.53    -0.021
9       23.04     -51.98   -34.2    9.99     5.34      8.87     3.22
```

Escribir un programa que gestione los datos de los ficheros anteriores siguiendo los pasos siguientes:

Declarar las matrices $A \in \mathcal{M}_{N\times N}(\mathbb{R})$, $B \in \mathcal{M}_{N\times 3}(\mathbb{R})$, $C \in \mathcal{M}_{N\times 2}(\mathbb{R})$ y los vectores $U, V, W, T \in \mathbb{R}^N$.

Leer el fichero de entrada (`input_1.dat` o `input_2.dat`) de la forma siguiente:

1. Cargar el fichero completo en la matriz A.
2. Cargar las cuatro primeras columnas del fichero en los vectores U, V , W y T.

3. Cargar la primera columna en el vector T y las tres últimas columnas en la matriz B.

4. Cargar la segunda columna en el vector U y las dos últimas columnas en la matriz C.

5. Cargar las columnas 1, 2 y 4 en la matriz B.

Además, el programa debe crear el fichero de salida (`output_1.dat` o `output_2.dat`), donde se irán escribiendo las matrices y vectores de los apartados anteriores. El formato de escritura debe ser el de números reales con cinco decimales.

Para el enunciado anterior, escribir los programas siguientes:

1. Programa 1 : Asignación estática de memoria.
   Ejecutar el programa por separado para los ficheros `input_1.dat` e `input_2.dat`. Para ello modificar las dimensiones y en nombre de los ficheros en el programa fuente.

2. Programa 2 : Asignación dinámica de memoria.
   Ejecutar el programa una única vez para gestionar los datos de los ficheros de entrada `input_1.dat` e `input_2.dat`.

2.8 Systems of linear equations

Implementar un módulo para la resolución de sistemas lineales de ecuaciones algebraicas. Los métodos de resolución propuestos son el de eliminación Gaussiana, factorización LU, factorización LU de la biblioteca *Numerical Recipes* y Jacobi.

Para cada método se pide:

- Validar los resultados con varios casos de prueba con dimensiones distintas.

- Evaluar tiempos de ejecución.

- Comparar resultados con los métodos restantes.

Aplicación : Estudiar el condicionamiento de sistemas lineales de ecuaciones para matrices aleatorias y de Vandermonde.

2.9 Systems of nonlinear equations

Implementar un módulo para la resolución numérica de ecuaciones no lineales. Para funciones $F : \mathbb{R} \to \mathbb{R}$, los métodos de resolución propuestos son el de la bisección y el de Newton-Raphson. Para funciones $F : \mathbb{R}^N \to \mathbb{R}^N$, se proponen el método de Newton-Raphson con matriz Jacobiana analítica y el método de Newton-Raphson con matriz Jacobiana numérica. Para la validación de los métodos propuestos, se pide implementar un módulo con al menos tres funciones $F : \mathbb{R} \to \mathbb{R}$ y al menos tres funciones $F : \mathbb{R}^N \to \mathbb{R}^N$. Este módulo debe contener las derivadas y matrices Jacobianas correspondientes de las funciones propuestas.

En el informe correspondiente, presentar tablas de soluciones numéricas en cada paso de iteración para las funciones de prueba propuestas.

2.10 Eigenvalues and eigenvectors

Implementar un módulo para el cálculo de autovalores y autovectores de una matriz. Los métodos de resolución propuestos son el método de la potencia y el método de la potencia inversa. Implementar el método de la potencia inversa a partir de la matriz inversa y resolviendo el sistema lineal correspondiente.

Para cada método se pide:

- Validar los resultados con varios casos de prueba con dimensiones distintas.
- Evaluar tiempos de ejecución. Comparar tiempos de ejecución del método de la potencia inversa mediante los dos algoritmos propuestos

: matriz inversa y solución del sistema lineal.

Aplicación : Estudiar el condicionamiento de sistemas lineales de ecuaciones para matrices aleatorias y de Vandermonde. Calcular la relación $\lambda_{max}/\lambda_{min}$ de los casos de prueba presentados en el hito 1 y relacionar y discutir los resultados.

```
subroutine   power_method

integer :: i, j, k
integer, parameter :: PI = 4 * atan(1d0)
integer, parameter :: N = 20
real :: x(0:N), Vandermonde(0:N, 0:N), sigma
real :: a=-1, b=1
real V(0:N), V0(0:N)


x = [ ( a + (b-a)*i/N, i=0, N) ]

forall(i=0:N, j=0:N) Vandermonde(i,j) = x(i)**j

V = 1
V0 = 0
do while( abs(norm2(V)-norm2(V0)) > 1d-5 )
V0 = V
V = matmul( Vandermonde, V ) / norm2(V)
write(*,*) maxval(V)
end do
sigma = dot_product( V, V )
write(*,*) "sigma = ", sigma


end subroutine
```

2.11 Finite differences

1. Obtener las fórmulas de las derivadas numéricas primeras descentradas, con tres puntos equiespaciados a una distancia Δx.
2. A partir de la función $f(x) = \mathrm{e}^x$ en el punto $x = 0$, representar gráficamente el error total de las derivadas numéricas frente al valor de Δx en precisión simple y doble. En particular, representar gráficamente las derivadas primeras adelantada (definición de derivada), centrada y descentradas y la derivada segunda, con tres puntos equiespaciados a una distancia Δx. Discutir los resultados obtenidos.
3. Resolver los problemas de contorno en ecuaciones diferenciales ordinarias siguientes:

   - **Problema 1:**

     $$u'' + u = 0, \quad x \in [-1, 1], \qquad u(-1) = 1, \quad u(1) = 0,$$

   - **Problema 2:**

     $$u''+u'-u = \sin(2\pi x), \quad x \in [-1, 1], \qquad u(-1) = 0, \quad u'(1) = 0.$$

   Para los problemas citados anteriormente se pide:

   (a) A partir de las derivadas numéricas con tres puntos equiespaciados escribir el sistema de ecuaciones resultante.
   (b) Obtener la solución numérica mediante la resolución de un sistema lineal de ecuaciones, con $N = 10$ y $N = 100$.
   (c) Representar gráficamente los resultados obtenidos.

2.12 Numerical integration

Implementar un módulo para la resolución numérica de integrales definidas de funciones $F : \mathbb{R} \to \mathbb{R}$. Los métodos de resolución propuestos son las reglas del rectángulo, punto medio, trapecio y Simpson. Implementar un módulo de funciones $F : \mathbb{R} \to \mathbb{R}$ de prueba para validar los métodos numéricos propuestos. Este módulo debe contener al menos tres funciones con funciones primitivas conocidas y una función cuya función primitiva sea desconocida.

Evaluar el error de las soluciones numéricas para cada método propuesto y para distintos valores del incremento de la partición.

2.13 to be included

elemental advance = no dummy versus actual assumed shape explicit shape

global, local and scope in modules

1 versus 1.

tab instead of blanks

brackets

mask in intrinsic functions

! and &

;

camel case versus underscore

overloading

forall parallel

enter matrices by row or columns

lower bound: upper bound

array operations C = A + B

public versus private

encapsulamiento y ocultaci

FORALL (I=1:N-1, J=1:N, J¿I) A(I,J) = A(J,I)

```
x   = 1.2345678912345678912345678900
xd  = 1.2345678912345678912345678900
xdd = 1.2345678912345678912345678900
```

```
write(*, '(ES)' ) x  ! scientific notation 1.234 (first digit should be greater
write(*, '(E)' ) x   ! exponential normalized  notation 0.1234 (first digit is z

write(*,*) " Single, double and quadruple precision "
write(*, '(E)' ) x
write(*, '(E)' ) xd
write(*, '(E)' ) xdd




!*****************************************************************************
!*
!*****************************************************************************
subroutine type_element



interface operator (+)
module procedure element
end interface

character(len=20) :: name
real (kind=4) :: x
real (kind=8) :: xd
real (kind=16) :: xdd

real :: a, b, c;
!  real :: a1, a2, a3;


type (person) :: father = person( "juan"), mother = person("cris")

associate ( name1 => father%name, name2=>mother%name )
name = trim(name1) // trim(name2)
end associate

a = 1; b = 1 ; c = 1;
```

```
associate ( a1 =>a, a2 =>b, a3=>c )
a3 = a1 + a2
write(*,*) " a3 = ", a3
end associate
write(*,*) " c = ", c
!    write(*,*) " a3 = ", a3

!   write(*,*) " father =", father % name
write(*,*) " element =", father + mother
write(*,*) " name =", name

x   = 1.23456789123456789123456789Q00
xd  = 1.23456789123456789123456789Q00
xdd = 1.23456789123456789123456789Q00
write(*, '(ES)' ) x  ! scientific notation 1.234 (first digit should be g
write(*, '(E)' ) x   ! exponential normalized  notation 0.1234 (first dig

write(*,*) " Single, double and quadruple precision "
write(*, '(E)' ) x
write(*, '(E)' ) xd
write(*, '(E)' ) xdd

write(*, '(3(E, :, ","))' ) x, xd, xdd

end subroutine



!**********************************************************************
!*
!**********************************************************************
subroutine Hello_world


write(*,'(a)', advance='no') " Hello world..... "
write(*,'(a)', advance='no') " press enter"
read(*,*)

end subroutine
```

```
!*****************************************************************************
!*
!*****************************************************************************
subroutine  commandline

character(len=256) :: line,  enval
integer :: i, iarg, stat, clen, len
integer :: estat, cstat

iarg = command_argument_count()
write(*,*) " iarg = ", iarg
do i=1,iarg
call get_command_argument(i,line,clen,stat)
write (*,'(I0,A,A)') i,': ',line(1:clen)
end do
call get_command(line,clen,stat)
write (*,'(A)') line(1:clen)

call get_environment_variable('HOSTNAME',enval,len,stat)
if (stat == 0) write (*,'(A,A)') 'Host=', enval(1:len)
call get_environment_variable('USER',enval,len,stat)
if (stat == 0) write (*,'(A,A)') 'User=', enval(1:len)


! call execute_command_line('ls -al', .TRUE., estat, cstat)
call execute_command_line('dir', .TRUE., estat, cstat)
if (estat==0) write (*,'(A)') "Command completed successfully"

end subroutine

!*****************************************************************************
!*
!*****************************************************************************
subroutine  allocate_characteristics

real, allocatable :: V(:), A(:, :)
character(:), allocatable :: S
integer :: i, j, N

real, pointer ::  B(:,:), Diagonal(:)
```

```
real, pointer :: memory(:)

real :: x, y, z
class(*), pointer :: p1(:)


N = 10
V = [ ( i/real(N), i=1, N ) ]  ! automatic allocation allocate( V(N) )
write(*,* ) " V = ", V

N = 2
A = reshape( [ (  ( (i/real(N))**j ,i=1, N ), j=1, N ) ], [N, N] )
do i=1, N
write(*,* ) " A = ", A(i,:)
end do

N = 4
A = reshape( [ (  ( (i/real(N))**j ,i=1, N ), j=1, N ) ], [N, N] )
do i=1, N
write(*,'(A, 100f6.2)' ) " A = ", A(i,:)
end do

S = "Hello world"
write(*,*) " S = ", S, len(S)

S = "Hello"
write(*,*) " S = ", S, len(S)


allocate( memory(1:N*N) )
B(1:N,1:N) => memory
memory = 0
forall(i=1:N) B(i,i) = 10.
do i=1, N
write(*,'(A, *(f6.2) )' ) " B = ", B(i,:)
end do
diagonal => memory(::N+1)
write(*,'(A, 100f6.2)' ) " diagonal = ", diagonal
write(*,'(A, 100f6.2)' ) " trace = ", sum(diagonal)
write(*,'(A, 100f6.2)' ) " trace = ", sum(memory(::N+1))
```

```
x = 1; y = 2; z = 3;

write(*,'("i=",I0,", REALs=",*(G0,1X),"....")') i, x, y, z  ! C++ style


allocate( integer :: p1(5) ) ! p1 is an array of integers

select type (p1)
type is (integer)
p1 = [( i, i=1, size(p1) ) ]
write(*,'(" p1 = ", *(I0, 1x) )')  p1

class default
stop 'Error in type selection'
end select

deallocate(p1)
allocate( real :: p1(3) ) ! now p1 is an array of reals

select type (p1)
type is (real)
p1 = [( i, i=1, size(p1) )  ]
write(*,'(" p1 = ", *(G0, 1x) )')  p1

class default
stop 'Error in type selection'
end select



end subroutine
```

Chapter 3

Interpolation techniques

Chapter 4

Three partial differential models

In this chapter, several partial differential equation models shall be considered. To start, a brief mathematical description will be given in order to clarify the types of equations that can be solved using the provided code. In particular, three models are presented: *Boundary Value Problems* (BVP), *Initial Value Boundary Problems* (IVBP) and *Mixed Problems*. Along with the mathematical explanation, examples for each type of problem, will be exposed, with its respective Fortran implementation and results.

4.1 Boundary Value Problems

In the study of many non-transient physical phenomenons, partial differential equations for the spatial distribution of the studied magnitudes appear. Along with these equations, several equations for the boundaries of the spatial domain must appear. The combination of these two sets of equations constitute a *Boundary Value Problem*. More rigurously it can be defined as follows:

Let us be $\Omega \subset \mathbb{R}^p$ an open and connected set, and $\partial\Omega$ its boundary set. The spatial domain D is defined as its closure, $D \equiv \{\Omega \cup \partial\Omega\}$. Each element of the spatial domain is called $\vec{x} \in D$.

It is defined a boundary problem for a vectorial function $\vec{u} : D \to \mathbb{R}^{N_v}$ of N_v variables, as:

$$\vec{\mathcal{L}}(\vec{x}, \vec{u}(\vec{x})) = 0, \quad \forall\ \vec{x} \in \Omega, \tag{4.1}$$

$$\vec{g}(\vec{x}, \vec{u}(\vec{x}))\big|_{\partial\Omega} = 0, \quad \forall\ \vec{x} \in \partial\Omega, \tag{4.2}$$

where $\vec{\mathcal{L}}$ is the spatial differential operator and $\vec{g}$ is the boundary conditions operator for the solution at the boundary points $\vec{u}\big|_{\partial\Omega}$.

In general, the differential operator is not linear and therefore we will distinguish between both cases as the numerical resolution of each one has its own particularities.

4.1.1 Linear Problems

1. **Poisson equation**

4.1.2 Non Linear Problems

1. **Von Karmann plate**

4.2 Initial Value Boundary Problems

Whenever in the considered physical model, the magnitudes vary not only along the spatial dimension but also along time, the equations that result are partial differential equations that involve temporal derivatives. In this case, not only boundary conditions are required for the resolution of the problem but also an initial value for the unknown variable shall be provided. The set of differential equation, boundary conditions and initial value is called *Initial Value Boundary Problem*, and more rigorously can be defined as follows:

Let us be $\Omega \subset \mathbb{R}^p$ an open and connected set, and $\partial\Omega$ its boundary set. The spatial domain D is defined as its closure, $D \equiv \{\Omega \cup \partial\Omega\}$. Each element of the spatial domain is called $\vec{x} \in D$. The temporal dimension is defined as $t \in \mathbb{R}$.

An Initial Value Boundary Problem for a vectorial function $\vec{u} : D \times \mathbb{R} \rightarrow \mathbb{R}^{N_v}$ of N_v variables, is defined as:

$$\frac{\partial \vec{u}}{\partial t}(\vec{x}, t) = \vec{\mathcal{L}}(\vec{x}, t, \vec{u}(\vec{x}, t)), \quad \forall \ \ \vec{x} \in \Omega, \tag{4.3}$$

$$\vec{g}(\vec{x}, t, \vec{u}(\vec{x}, t))\big|_{\partial\Omega} = 0, \quad \forall \ \ \vec{x} \in \partial\Omega, \tag{4.4}$$

$$\vec{u}(\vec{x}, t_0) = \vec{u}_0(\vec{x}), \tag{4.5}$$

where $\vec{\mathcal{L}}$ is the spatial differential operator, $\vec{u}_0(\vec{x})$ is the initial value and $\vec{g}$ is the boundary conditions operator for the solution at the boundary points $\vec{u}\big|_{\partial\Omega}$.

4.2.1 Parabollic Equations

Heat equation

4.2.2 Hyperbollic Equations

Waves equation

Biharmonic equation

4.3 Mixed Problems

Let us be $\Omega \subset \mathbb{R}^p$ an open and connected set, and $\partial\Omega$ its boundary set. The spatial domain D is defined as its closure, $D \equiv \{\Omega \cup \partial\Omega\}$. Each element of the spatial domain is called $\vec{x} \in D$. The temporal dimension is defined as $t \in \mathbb{R}$.

The intention of this section is to numerically solve a temporal evolution problem for two vectorial functions $\vec{u} : D \times \mathbb{R} \to \mathbb{R}^{N_u}$ of N_u variables and $\vec{v} : D \times \mathbb{R} \to \mathbb{R}^{N_v}$ of N_v variables, such as:

$$
\begin{aligned}
&\frac{\partial \vec{u}}{\partial t}(\vec{x},t) = \vec{\mathcal{L}}_u(\vec{x},t,\vec{u}(\vec{x},t),\vec{v}(\vec{x},t)), && \forall\ \vec{x} \in \Omega,\\
&\vec{g}(\vec{x},t,\vec{u}(\vec{x},t))\big|_{\partial\Omega} = 0, && \forall\ \vec{x} \in \partial\Omega,\\
&\vec{u}(\vec{x},t_0) = \vec{u}_0(\vec{x}), && \forall\ \vec{x} \in D,\\
\\
&\vec{\mathcal{L}}_v(\vec{x},t,\vec{v}(\vec{x},t),\vec{u}(\vec{x},t)) = 0, && \forall\ \vec{x} \in \Omega,\\
&\vec{h}(\vec{x},t,\vec{v}(\vec{x},t))\big|_{\partial\Omega} = 0, && \forall\ \vec{x} \in \partial\Omega,
\end{aligned}
$$

where $\vec{\mathcal{L}}_u$ is the spatial differential operator of the initial value problem of N_u equations, $\vec{u}_0(\vec{x})$ is the initial value, $\vec{g}$ is the boundary conditions operator for the solution at the boundary points $\vec{u}\big|_{\partial\Omega}$, $\vec{\mathcal{L}}_v$ is the spatial differential operator of the boundary value problem of N_v equations and $\vec{h}$ is the boundary conditions operator for $\vec{v}$ at the boundary points $\vec{v}\big|_{\partial\Omega}$. It can be seen that both problems are coupled as the differential operators are defined for both variables. The order in which appear in the differential operators u and v indicates its number of equations, for example: $\vec{\mathcal{L}}_v(\vec{x},t,\vec{v},\vec{u})$ and $\vec{v}$ are of the same size as it appears first in the list of variables from which the operator depends on.

4.3.1 Von Karmann equations

4.3.2 Diffusion-advection equation

Chapter 5

High Order Finite Differences

Bibliography

[1] Bishop, P. (1989) Conceptos de Informática. Anaya Multimedia.

[2] Borse, G. J. (1989) Programación en FORTRAN 77 con Aplicaciones de Cálculo Numérico en Ciencias e Ingeniería. Anaya Multimedia.

[3] Cuevas, G. (1991) Ingeniería del software: práctica de la programación. RA–MA.

[4] Dowd, K. (1993) High performance computing. O'Reilly & Associates, Inc.

[5] ESA.(1991) ESA Software Engineering Standards. PSS-05-0. Issue 2.

[6] Etter, Delores M. (1997) Structured FORTRAN 77 for engineers and scientists. Addison–Wesley.

[7] García Merayo, F.. (1990) Programación en FORTRAN 77. Paraninfo.

[8] Golden, James T. (1976) FORTRAN IV. Programación y cálculo. Urmo, S.A. de ediciones.

[9] Joyanes, L. (1987) Metodología de la programación; diagramas de flujo, algoritmos y programación estructurada. Mc Graw–Hill.

[10] Kernighan, B. W. y Ritchie, D. M. (1988) The C programming language. Prentice–Hall.

[11] Knuth, Donald E. (1997) The art of computing programming. Volume 1: Fundamental algorithms. Volume 2: Seminumerical algorithms. Addison–Wesley.

[12] Metcalf, M. y Reid, J. (1999) FORTRAN 90/95 explained. OXFORD. University Press.

[13] NASA. (1992) Recommended Approach to Software Development. Revision 3. Software Engineering Laboratory Series. SEL–81–305.

[14] Piattini, Mario G. y Daryanani, Sunil N. (1995) Elementos y herramientas en el desarrollo de sistemas de información. Una visión actual de la tecnología CASE. RA–MA.

[15] Pressman, Roger S. (1997) Ingeniería del software. Un enfoque práctico. Mc. Graw–Hill.

[16] Schofield, C. F. (1989) Optimising fortran programs. John Wiley & Sons.

[17] Wirth, N. (1980) Algoritmos + Estructuras de Datos = Programas. Ediciones del Castillo.

[18] Yourdon, E. (1989) Managing the structrured techniques. Prentice–Hall.