UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA CIVIL

“ MÉTODOS PARA SOLUCIONAR SISTEMAS DE ECUACIONES LINEALES ”

TESINA PROFESIONAL

PRESENTA:

PAUL ALEJANDRO SANDOVAL HUERTA

PARA OBTENER EL TÍTULO DE:

INGENIERO CIVIL

ASESOR DE TESINA:

M. I. ALMA ROSA SÁNCHEZ IBARRA

MORELIA, MICH. MAYO, 2012

ÍNDICE

INTRODUCCIÓN

Son numerosas las aplicaciones prácticas de los sistemas de ecuaciones: el cálculo de estructuras reticulares definidas por barras, modelos económicos, cálculo de tensiones en los nodos de una red de tensión continua, etc., cuya resolución práctica pasa por plantear y resolver un sistema de ecuaciones lineales.

En las ciencias aplicadas gran parte de los problemas que surgen se resuelven mediante algoritmos que involucran, en alguna de las etapas, la resolución de un sistema lineal, desde la cuantificación de materias primas para la creación de un producto hasta la intensidad de corriente que pasa dentro de un circuito.

Se abordará en este tema el estudio de 3 métodos directos y un método iterativo de resolución de sistemas de ecuaciones. Los métodos directos considerados son el método de *Gauss-Jordan,* el método de la *Matriz Inversa* y el método de *Cholesky* (la factorización *LU* de matrices) y el método iterativo que se analizará es el método de *Gauss-Seidel.*

En cada tema se estudian los fundamentos teóricos suficientes, se desarrolla el pseudocódigo del algoritmo numérico, y se acompaña con ejemplos que tiendan a clarificar e ilustrar la aplicación de los métodos numéricos objeto de análisis.

El aprendizaje y estudio del análisis numérico es más didáctico con el empleo de los lenguajes de programación numérico y simbólico, dado que los estudiantes pueden comprobar los resultados con el empleo de la computadora. Cuando los alumnos se enfrentan a la programación de los métodos son capaces de comprender en toda su magnitud el problema de la aproximación numérica.

OBJETIVO

Se ha preparado la presente tesina con el propósito de ofrecer y presentar los fundamentos teóricos y las aplicaciones prácticas de la Solución de Sistemas de Ecuaciones Lineales. Sentar las bases del estudio del análisis numérico y del cálculo científico proporcionando algoritmos y métodos que permitan calcular una aproximación de la solución hallada y la real, dando una estimación del error cometido.

Se buscan soluciones aproximadas mediante algoritmos numéricos a los problemas de solución de sistemas de ecuaciones.

Así mismo, desarrollar un programa en FORTRAN 90 que soluciona sistemas de ecuaciones lineales por los métodos antes mencionados (el método de *Gauss-Jordan*, de *la Matriz Inversa*, de *Cholesky* y de *Gauss-Seidel*) con la finalidad de contar con una herramienta más veloz para la solución de los sistemas y poder comparar su efectividad ante los procesos tradicionales de análisis numérico.

CAPÍTULO 1.- SOLUCIÓN DE SISTEMAS DE ECUACIONES LINEALES

1.1 DEFINICIÓN DE SISTEMAS DE ECUACIONES

Considere las siguientes ecuaciones como un sistema de ecuaciones lineales.

$$\begin{array}{lllllll}
a_{11}x_1 + & a_{12}x_2 + & a_{13}x_3 + & \ldots\ldots + & a_{1n}x_n & = c_1 \\
a_{21}x_1 + & a_{22}x_2 + & a_{23}x_3 + & \ldots\ldots + & a_{2n}x_n & = c_2 \\
a_{31}x_1 + & a_{32}x_2 + & a_{33}x_3 + & \ldots\ldots + & a_{3n}x_n & = c_3 \\
\ldots\ldots\ldots & \ldots\ldots\ldots & \ldots\ldots\ldots & \ldots\ldots & \ldots\ldots\ldots & \ldots\ldots\ldots \\
a_{n1}x_1 + & a_{n2}x_2 + & a_{n3}x_3 + & \ldots\ldots + & a_{nn}x_n & = c_n
\end{array}$$

Este es un sistema de *n* ecuaciones con *n* incógnitas que son:

$x_1, x_2, x_3, \ldots, x_n$.

Los coeficientes a_{ij} y los términos independientes c_i son números reales determinados. El sistema lineal n x n se puede representar mediante una notación matricial del siguiente modo:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & & a_{3n} \\ & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \ldots \\ x_n \end{bmatrix} \qquad C = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \ldots \\ c_n \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & & a_{3n} \\ & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \ldots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \ldots \\ c_n \end{bmatrix}$$

Entonces, si a la matriz de coeficientes del sistema se le llama A, al vector solución se le llama x, y a el vector de términos independientes o términos constantes se le llama C, el sistema lineal de ecuaciones se reduce a la expresión

$$A\,x = C$$

Otra manera de escribir el sistema de ecuaciones lineales es representando en una sola matriz, la matriz de coeficientes y la de los términos constantes:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & & a_{1n} & c_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & c_2 \\ a_{31} & a_{32} & a_{33} & & a_{3n} & c_3 \\ & \vdots & & \ddots & \vdots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & c_n \end{bmatrix}$$

Resolver un sistema de ecuaciones lineales consiste en localizar todas las soluciones del mismo, si es que existen. Cuando un sistema de ecuaciones lineales no tiene solución se dice que es incompatible.

Si tiene al menos una solución se dice que el sistema es compatible. Si es un sistema compatible y solamente tiene una solución, entonces diremos que el sistema es determinado y si hay más de una solución es indeterminado.

Los sistemas de ecuaciones lineales homogéneos son todos compatibles ya que tienen al menos una solución, a esta solución se le conoce como solución trivial.

Para los sistemas de ecuaciones lineales no homogéneos, una forma de localizar la solución particular es realizando transformaciones elementales en las filas de la matriz ampliada, hasta conseguir otro sistema que contenga una matriz triangular. De este modo, el nuevo sistema resultante tendrá la misma solución que el sistema de partida pero es más sencillo de resolver.

1.2 MÉTODO DE GAUSS-JORDAN Ó DE LA MATRIZ AUMENTADA

El método de eliminación de *Gauss-Jordan* para resolver un sistema de ecuaciones lineales de la forma $A\,x = C$ combina las etapas de eliminación progresiva y sustitución regresiva del método de *Gauss*, pero obteniendo una matriz de coeficientes transformada en una matriz identidad.

En este algoritmo, en el k-ésimo paso principal se restan múltiplos de la fila k de las demás filas, de tal modo que el coeficiente x_k es cero en todas las filas menos en la k-énesima fila. Al final la matriz será diagonal, en lugar de triangular superior. Una vez obtenida la matriz identidad, el vector solución se calcula directamente.

Se usa la k-énesima ecuación para eliminar la variable x_k en las ecuaciones o filas F_{k+1}, F_{k+2},..., F_n y las ecuaciones F_1, F_2, ...F_{k-1}. Por lo tanto la ecuación $A\,x = C$ quedaría reducida a:

$$\begin{bmatrix} a_{11}^{(1)} & 0 & 0 & & 0 \\ 0 & a_{22}^{(2)} & 0 & \cdots & 0 \\ 0 & 0 & a_{33}^{(3)} & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} c_1^{(1)} \\ c_2^{(2)} \\ c_3^{(3)} \\ \dots \\ c_4^{(4)} \end{bmatrix}$$

Donde cada elemento $a_{jj}^{(1)}$ es el obtenido en la i-ésima transformación del sistema lineal en su equivalente.

La solución se obtiene usando la ecuación

$$x_k = \frac{c_1^{(i)}}{a_{ii}^{(i)}} \quad \text{para } i = 1, 2, \dots, n.$$

1.3 MÉTODO DE LA MATRIZ INVERSA

Se dice que la matriz A de dimensión *n* x *n* es no singular, si existe una matriz A^{-1} que verifica $A\,A^{-1} = A^{-1}A = I$. A la matriz A^{-1} se le llama inversa de A. A una matriz que no tiene inversa se le denomina *singular*.

Una matriz A no singular tiene las siguientes propiedades:

1. A^{-1} es única
2. $(A^{-1})^{-1} = A$
3. Si *B* es también una matriz no singular n x n, entonces $(A\,B)^{-1} = A^{-1}B^{-1}$

Si se conoce el valor de A^{-1} es fácil resolver el sistema lineal $A\,x = C$ aunque no es suficiente resolver A^{-1}, dado los cálculos que requiere.

Si $A\,x = C$

premultiplicando por A^{-1}

$$A^{-1}A\,x = A^{-1}C$$

pero

$$A^{-1}A = I$$

entonces

$$I\,x = A^{-1}C$$

como

$$I\,x = x$$

entonces

$$x = A^{-1}C$$

de esta manera, conociendo la matriz inversa de A, se llega a determinar el vector solución x.

1.4 MÉTODO DE CHOLESKY

Existen otros métodos para resolver un sistema lineal de la forma $A\,x = C$. Uno de ellos es la factorización de una matriz en un producto matricial. Esta factorización es muy útil cuando es de la forma $A = LU$, donde L es una matriz triangular inferior y U es una triangular superior. Esta descomposición en dos matrices triangulares recibe el nombre de factorización LU de una matriz o descomposición LU.

No todas las matrices pueden factorizarse de esta manera, pero en la mayoría de los casos si es posible hacerlo.

Dada la matriz A se buscan las matrices L y U tales que $A = LU$, donde la expresión de L y U son:

$$L = \begin{bmatrix} l_{11} & 0 & 0 & & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & & 0 \\ & \vdots & & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix}$$

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & & u_{3n} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Si la matriz A se puede factorizar de la forma triangular $A = LU$, entonces para resolver el sistema $A\,x = C$ se puede calcular el vector solución x, empleando un proceso de dos pasos.

Primero se resuelve el sistema $Ly = C$. Como L es triangular se puede emplear el método de eliminación de Gauss. Segundo, conocido el vector y, se determina el vector x que verifica $Ux = y$.

La factorización LU de Cholesky consiste en obtener una matriz triangular superior U unitaria (u_{ii} = 1 para $1 \leq i \leq n$) y una matriz triangular inferior con $l_{ii} \neq 1$ en general. Por ejemplo, para una matriz de 4 x 4 se tendrían las matrices siguientes para una factorización de Cholesky:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = L\,U$$

$$= \begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Los valores de los elementos de las matrices L y U se obtienen realizando la operación matricial LU.

Las fórmulas generales que definen los elementos de las matrices L y U en función de los de la matriz *A* vienen dadas por las expresiones siguientes:

$l_{i\,1} = a_{i1}$ para j=1; i=1, 2, 3, …, n

$u_{1\,j} = a_{1j} / a_{11}$ para i=1; j=1, 2, 3, …, n

$$u_{j\,j} = \left(aij - \sum_{k=1}^{i-1} l_{ik}.\,u_{kj}\right) \Big/ l_{ii} \quad \text{si } 1 \leq i \leq j\ ;$$

$$l_{i\,j} = \left(aij - \sum_{k=1}^{j-1} l_{ik}.\,u_{kj}\right) \quad \text{si } i \leq j \leq 1\ ;$$

1.5 MÉTODO DE GAUSS-SEIDEL

Los métodos iterativos para la resolución de sistema de ecuaciones proporcionan una solución del sistema lineal $A\ x = C$ como límite de una sucesión de vectores aproximantes, en general no se obtiene una solución de forma exacta sino de manera aproximada. A pesar de esto, su uso está muy extendido, debido a las limitaciones y complejidad que presentan los métodos directos. Entre otras:

a) Cuando el tamaño de la matriz A del sistema es muy grande (n >> 100) la propagación del error de redondeo es también grande, y los resultados obtenidos puede diferir bastante de los reales.
b) Las matrices que aparecen en los sistemas lineales de diferentes aplicaciones son muchas de ellas de gran tamaño (n ≅ 100) y con la particularidad de que la mayoría de los elementos son nulos. Este tipo de matrices se denominan *vacías* o *huecas*. Si los elementos no nulos están distribuidos alrededor de la diagonal principal, se les puede aplicar los métodos directos. En cambio, cuando la matriz es *dispersa*, al aplicar los métodos directos se produce el efecto conocido como *rellenado*: con el método de eliminación de *Gauss* se anulan los elementos de una columna por debajo de la diagonal principal, pero se convierten en elementos no nulos algunos que previamente lo eran.

Los métodos iterativos solucionan estos problemas ya que se basan en la resolución de sistemas diagonales o triangulares. La premisa fundamental en estos métodos es que el valor de cada iteración no sea mayor que el de la multiplicación de una matriz por un vector. Este objetivo se consigue en cada paso al resolver un sistema diagonal o triangular.

El método de *Gauss* y sus variantes son conocidos como *métodos directos* para resolver el sistema de ecuaciones lineales $A\,x = C$. Estos métodos se ejecutan a través de un número finito de pasos y generan un vector solución x que sería exacto si no fuera por los errores de redondeo.

En contraste, un *método iterativo* da lugar a una sucesión de vectores que idealmente convergen a la solución. El cálculo se detiene cuando se tiene una solución aproximada con cierto grado de precisión especificado con antelación, o bien después de un número dado de iteraciones. Los métodos indirectos son casi siempre *iterativos* por su naturaleza, puesto que la solución aproximada se calcula con la repetición de un determinado proceso.

Para sistemas lineales grandes, los métodos iterativos muestran ventajas decisivas respecto de los métodos directos, en cuanto a velocidad y requerimientos de memoria. Otra ventaja de los métodos iterativos es que usualmente son estables y amortiguan errores, debidos al redondeo o a errores pequeños, conforme se lleva a cabo el proceso.

Teorema

Sea el sistema lineal $x = Bx + D$, siendo la matriz B y el vector D tales que la matriz $I - B$ es inversible. El método iterativo asociado a la matriz es convergente si existe una norma matricial tal que:

$$\|B\| < 1$$

El criterio fundamental de convergencia de los métodos iterativos, sólo involucra a la matriz B del método iterativo considerado.

A la hora de resolver un sistema lineal aplicando un método iterativo se debe asegurar su convergencia encontrando alguna norma para la cual $\|B\| < 1$.

Se van a introducir uno de los métodos iterativos más clásicos en la resolución de los sistemas lineales: el método de *Gauss-Seidel*.

Este comparte en su diseño la descomposición de la matriz del sistema en la suma de otras dos:

$$A = M - N$$

Donde M va a ser una matriz fácil de calcular su inversa (diagonal o triangular). Con esta descomposición se tiene:

$$Ax = C \Leftrightarrow (M-N)x = C \Leftrightarrow Mx = Nx + C \Leftrightarrow$$
$$x = Bx + D,$$

Siendo

$$B = M^{-1} N,\ D = M^{-1} C$$

Entonces se puede considerar el método iterativo

$$x^{(0)} = \text{ arbitrario}$$
$$x^{(i+1)} = Bx^{(i)} + D,\ i = 1, 2, \ldots$$

Como $N = M - A$, se tiene que $B = M^{-1} N = M^{-1}(M - A) = I - M^{-1} A$, y la diferencia $I - B = M^{-1}$, es una matriz inversible, por lo que el sistema $(I - B)x = D$ tiene solución única.

Notación

Dada una matriz $A = (aij)_{i,j=1}^{n}$ inversible con los elementos $aii \neq 0$ para $i = 1, 2, \ldots, n$, se considera la siguiente descomposición de la matriz

$$A = \begin{bmatrix} . & . & -F \\ . & D & . \\ -E & . & . \end{bmatrix}$$

Donde:

$$D = \text{diagonal}\,(a11, a22, \dots, ann) = \begin{bmatrix} a_{11} & 0 & 0 & & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ 0 & 0 & a_{33} & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix}$$

$$E = (eij)_{i,j=1}^{n} = \begin{bmatrix} 0 & 0 & 0 & & 0 \\ -a_{21} & 0 & 0 & \cdots & 0 \\ -a_{31} & -a_{32} & 0 & & 0 \\ & \vdots & & \ddots & \vdots \\ -a_{n1} & -a_{n2} & -a_{n3} & \cdots & 0 \end{bmatrix}, \qquad e_{ij} = \begin{cases} -aij & si\ i > j \\ 0 & si\ i \le j \end{cases}$$

$$F = (fij)_{i,j=1}^{n} = \begin{bmatrix} 0 & -a_{12} & -a_{13} & & -a_{1n} \\ 0 & 0 & -a_{23} & \cdots & -a_{2n} \\ 0 & 0 & 0 & & -a_{3n} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}, \qquad f_{ij} = \begin{cases} -aij & si\ i < j \\ 0 & si\ i \ge j \end{cases}$$

De esta forma A queda expresada de la forma $A = D - (E + F)$. Esta descomposición de la matriz se denomina descomposición $D - E - F$ por puntos.

En este método iterativo se mejora la convergencia obteniendo la componente k del paso $i + 1$, es decir, $x_k^{(i+1)}$ a partir de las $(k - 1)$ primeras componentes del paso $(i + 1)$ ya calculadas, o sea de:

$$\{ x_1^{(i+1)},\ x_2^{(i+1)}, \dots, x_{k-1}^{(i+1)} \},$$

En lugar de usar las anteriores $\{ x_1^{(i+1)},\ x_2^{(i+1)}, \dots, x_{k-1}^{(i+1)} \}$. Considerando las componentes ya calculadas se obtienen las ecuaciones iterativas siguientes:

$$x_k^{(i+1)} = \frac{1}{a_{kk}} \left(c_k - \sum_{j=1}^{k-1} a_{kj} x_j^{(i+1)} - \sum_{j=k+1}^{n} a_{kj} x_j^{(i)} \right),$$

para $k = 1, 2, \dots n$, donde

$$x^{(i)} = (x_1^{(i)}\ x_2^{(i)}\ \dots\ x_n^{(i)})^{\mathrm{T}} \quad \text{y} \quad x^{(i+1)} = (x_1^{(i+1)}\ x_2^{(i+1)}\ \dots\ x_n^{(i+1)})^{\mathrm{T}}$$

Este nuevo método iterativo se puede deducir tomando las matrices siguientes:

$$M = D - E,\ N = F$$

De esta forma, la ecuación lineal se convierte en:

$$A\,x = C \Leftrightarrow (M - N)x = C$$
$$(D - E)\,x = Fx + C$$

y despejando el vector x se tiene

$$x = (D - E)^{-1}\,Fx = (D - E)^{-1}\,C$$

Esta ecuación da lugar al método iterativo de *Gauss-Seidel*

$x^{(0)}$ = arbitrario

$$x^{(i+1)} = (D - E)^{-1}\,Fx^{(i+1)} + (D - E)^{-1}\,C,\ i = 0, 1, 2, \ldots$$

o su equivalente,

$x^{(0)}$ = arbitrario

$$(D - E)^{-1}\,x^{(i+1)} = Fx^{(i)} + C,\ i = 0, 1, 2, \ldots$$

La matriz de este método es la siguiente:

$$G = (D - E)^{-1}\,F = I - (D - E)^{-1}\,A,$$

que se denomina matriz de *Gauss-Seidel* por puntos.

Conocidos los valores de las matrices D, E y F, la ley de recurrencia

$x^{(0)}$ = arbitrario

$$(D - E)^{-1}\,x^{(i+1)} = Fx^{(i)} + C,\ i = 0, 1, 2, \ldots$$

se convierte en la expresión matricial siguiente:

$$D - E = \begin{bmatrix} a_{11} & 0 & 0 & & 0 \\ 0 & a_{22} & 0 & \cdots & 0 \\ 0 & 0 & a_{33} & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & & 0 \\ -a_{21} & 0 & 0 & \cdots & 0 \\ -a_{31} & -a_{32} & 0 & & 0 \\ & \vdots & & \ddots & \vdots \\ -a_{n1} & -a_{n2} & -a_{n3} & \cdots & 0 \end{bmatrix} =$$

$$\begin{bmatrix} a_{11} & 0 & 0 & & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & & 0 \\ & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

$$F = \begin{bmatrix} 0 & -a_{12} & -a_{13} & & -a_{1n} \\ 0 & 0 & -a_{23} & \cdots & -a_{2n} \\ 0 & 0 & 0 & & -a_{3n} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & 0 & 0 & & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & & 0 \\ & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1^{(i+1)} \\ x_2^{(i+1)} \\ x_3^{(i+1)} \\ \cdots \\ x_n^{(i+1)} \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -a_{12} & -a_{13} & & -a_{1n} \\ 0 & 0 & -a_{23} & \cdots & -a_{2n} \\ 0 & 0 & 0 & & -a_{3n} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \\ \cdots \\ x_n^{(i)} \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \cdots \\ c_n \end{bmatrix}$$

Operando matricialmente, se obtienen las ecuaciones del algoritmo iterativo de *Gauss-Seidel*:

$$a_{11}x_1^{(i+1)} = c_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)} - a_{14}x_4^{(i)} - \cdots - a_{1n}x_n^{(i)}$$

$$a_{21}x_1^{(i+1)} + a_{22}x_2^{(i+1)} =$$

$$c_2 - a_{23}x_3^{(i)} - a_{24}x_4^{(i)} - \cdots - a_{2n}x_n^{(i)}$$

$$a_{31}x_1^{(i+1)} + a_{32}x_2^{(i+1)} + a_{33}x_3^{(i+1)} =$$

$$c_3 - a_{34}x_4^{(i)} - \cdots - a_{3n}x_n^{(i)}$$

……..…

$$a_{n1}x_1^{(i+1)} + a_{n2}x_2^{(i+1)} + a_{n3}x_3^{(i+1)} + \cdots + \; = \; a_{nn-1}x_{n-1}^{(i+1)} + \; a_{nn}x_n^{(i+1)} = \; c_n$$

Despejando el vector solución en la iteración (i +1) se tiene el siguiente sistema equivalente:

$$x_1^{(i+1)} = \frac{1}{a_{11}}\left(c_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)} - a_{14}x_4^{(i)} - \cdots - a_{1n}x_n^{(i)}\right)$$

$$x_2^{(i+1)} = \frac{1}{a_{22}}\left(c_2 - a_{21}x_1^{(i)} - a_{23}x_3^{(i)} - a_{24}x_4^{(i)} - \cdots - a_{2n}x_n^{(i)}\right)$$

$$x_3^{(i+1)} = \frac{1}{a_{33}}\left(c_3 - a_{31}x_1^{(i)} - a_{32}x_2^{(i)} - a_{34}x_4^{(i)} - \cdots - a_{3n}x_n^{(i)}\right)$$

……

$$x_n^{(i+1)} = \frac{1}{a_{nn}}\left(c_n - a_{n1}x_1^{(i)} - a_{n2}x_2^{(i)} - a_{n3}x_3^{(i)} - \cdots - a_{nn-1}x_{n-1}^{(i+1)}\right)$$

De forma general

$$x_k^{(i+1)} = \frac{1}{a_{kk}}\left(c_k - \sum_{j=1}^{k-1} a_{kj}x_j^{(i+1)} - \sum_{j=k+1}^{n} a_{kj}x_j^{(i)}\right)$$

para k = 1, 2, …, n,

$$x^{(i)} = (x_1^{(i)}\ x_2^{(i)}\ \ldots\ x_n^{(i)})^{\mathrm{T}} \quad \text{y} \quad x^{(i+1)} = (x_1^{(i+1)}\ x_2^{(i+1)}\ \ldots\ x_n^{(i+1)})^{\mathrm{T}}$$

En cada iteración las n componentes del vector $x^{(i+1)}$ se obtienen de forma sucesiva a partir de las componentes ya calculadas de $x^{(i+1)}$ y de las restantes del vector $x^{(i)}$. Por esta razón, a este método se le conoce como método de las *iteraciones sucesivas*.

Este método iterativo, al igual que el de *Gauss-Jacobi* se puede finalizar cuando se cumpla que la norma de la diferencia entre dos soluciones aproximadas $x^{(i+1)}$ y $x^{(i)}$ sea menor que una cantidad prefijada k,

$$\left\| x^{(i+1)} - x^{(i)} \right\| < k$$

El método de *Gauss-Seidel* también es un método *auto-corrector*, lo que indica que los errores cometidos en el cálculo, no afectan a la solución final, pero sí afectan a la rapidez de la convergencia del proceso.

Teóricamente, el método de Gauss-Seidel puede ser un proceso infinito. En la práctica el proceso se acaba cuando de x^k a x^{k+n} los cambios son muy pequeños. Esto quiere decir que el valor de x es casi la solución real.

Como el método no siempre converge, otra detención del proceso es determinada cuando el número de iteraciones realizadas es igual a un número máximo de iteraciones previsto.

Algunos criterios garantizan la convergencia del método de Gauss-Seidel. Aunque una matriz A puede no cumplir estos requisitos y sin embargo el método puede ser convergente. En la práctica, es muy difícil poder aplicar estos criterios.

Una matriz cuadrada es de diagonal estrictamente dominante si el valor absoluto del elemento diagonal es mayor que la suma de los valores absolutos de los otros elementos de la fila,

$$|a_{ii}| > > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad \forall i$$

Teorema 0.1. Si A es de diagonal estrictamente dominante por filas, entonces el método de Gauss-Seidel converge para cualquier x^0 inicial.

Teorema 0.2. Si A es definida positiva, entonces el método de Gauss-Seidel converge para cualquier x^0 inicial.

CAPÍTULO 2.- ASPECTOS BÁSICOS DE PROGRAMACIÓN EN FORTRAN 90

2.1 INTRODUCCIÓN A FORTRAN 90

La palabra FORTRAN es la abreviación de FORmula TRANslation. Fortran fue el primer lenguaje científico de alto nivel utilizado en la historia de las computadoras. La primera versión fue desarrollada para IBM 704 por John Backus y colaboradores entre 1954 y 1957, pocos meses después apareció la versión llamada Fortran II. Poco a poco se empezaron a desarrollar versiones más o menos similares de Fortran para diversas computadoras. En 1962 se presentó Fortran IV, que era casi por completo independiente de la computadora en el que se había de ejecutar. En 1962 se estableció un comité de ANSI (America Nacional Standard Institute) para definir un Fortran estándar, que estuvo basado en Fortran IV. Este estándar fue ratificado en 1966, y se le conoce como Fortran 66.

En 1977 se publicó el borrador de un nuevo estándar que incorporaba los avances alcanzados en aquellos años. Este nuevo estándar fue publicado en 1978 con el nombre de Fortran 77. Finalmente, en 1991 se publicó un nuevo estándar, esta vez aprobado por la ISO (Internacional Standards Organization).

Fortran 90 es una actualización de Fortran 77 que moderniza este lenguaje, incorporando algunas de las características comunes de los lenguajes más modernos (C, C++, Pascal, etc.). Con el nuevo lenguaje se persiguen los siguientes objetivos:

- Modernizar la sintaxis.
- Incluir aspectos de programación modular, recursividad.
- Mejorar la “habilidad” para trabajar con las matemáticas.
- Incorporación de estructuras de datos y de punteros.

Fortran nació y se ha desarrollado como un lenguaje especializado en cálculos técnicos y científicos. Aunque las librerías matemáticas y numéricas existentes para Fortran son probablemente las más completas y eficientes, y aunque los compiladores de Fortran suelen producir el código ejecutable más rápido de todos, lo cierto es que el lenguaje Fortran ha ido perdiendo peso frente a lenguajes de propósito general como C/C++ que son hoy día muchísimo más utilizados. Fortran mantiene sin embargo una cierta importancia en ingeniería y métodos numéricos, y en muchos casos es la opción preferible. Fortran es más fácil de aprender que C/C++ y en las últimas versiones ha ido incorporando ideas de otros lenguajes más modernos.

2.1.1 ESTRUCTURA GENERAL DE UN PROGRAMA EN FORTRAN 90

La primera y la última línea de un programa son respectivamente:

```
PROGRAM nombre_del_programa

...

END PROGRAM nombre_del_programa
```

Donde `nombre_del_programa` debe ser una cadena de caracteres que inicie con un alfa-numérico y que no contenga espacios en blanco ni operadores o símbolos.

2.1.2 CARACTERES PERMITIDOS

Los caracteres permitidos en Fortran 90 son los siguientes:

0, 1, 2, 3 ,4, 5, 6, 7, 8, 9

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

', ", (,), +, -, *, /, :, =, !, &, $, <, >, %, ?, (incluye espacio, punto, coma, y punto y coma)

Es importante tener en cuenta que Fortran no distingue entre mayúsculas y minúsculas.

2.1.3 PALABRAS RESERVADAS

Como todos los lenguajes de programación, Fortran 90 tiene algunas palabras reservadas (comandos), de las cuales se debe tener conciencia en su uso:

ALLOCATE, ALLOCATABLE
CALL, CASE, CHARACTER, COMPLEX, CONTAINS, CYCLE,
DEALLOCATE, DEFAULT, DIMENSION, DO,
END, ELSE, ELSEWHERE, EXIT, EXTERNAL
FUNCTION,
IF, IMPLICIT, IN, INOUT, INTEGER, INTENT, INTERFACE, INTRINSIC
KIND,
LOGICAL,
MODULE

NONE,
ONLY, OPEN, OUT,
PARAMETER, POINTER, PRINT, PROGRAM,
READ, REAL, RECURSIVE, RESULT, RETURN
SAVE, SELECT, SIZE, STAT, STOP, SUBROUTINE
TARGET, THEN, TYPE
UNIT, USE
WHERE, WRITE

2.1.4 CARACTERÍSTICAS GENERALES

Los identificadores son nombres para identificar programas, variables, constantes (simbólicas), y otras identidades. Los identificadores deben siempre empezar por una letra, y pueden ir seguidos por hasta 30 letras, dígitos y guión bajo.

En los identificadores no se distingue entre mayúsculas y minúsculas. Es bastante habitual escribir con mayúsculas las palabras clave del lenguaje (IF, DO, END, …). Con frecuencia se unen varias palabras poniendo la primera letra en mayúscula o separándolas con un guión bajo.

Las *líneas* pueden tener hasta 132 caracteres. Una línea puede contener varias sentencias, separadas por el carácter punto y coma (;).

Las sentencias pueden llevar una *etiqueta*, que debe ir al comienzo de la línea y que es una constante entera entre 1 y 99999. La etiqueta debe ir separada del resto de la sentencia por uno o más espacios.

Las *líneas de comentario* empiezan por el carácter (!). Se admiten comentarios al final de una sentencia ejecutable, pues el compilador ignora todo lo que aparece en una línea a continuación del carácter (!), excepto si este carácter aparece en el interior de una cadena de caracteres.

Para *continuar* una sentencia en la línea siguiente se termina con el carácter *&*. Se permiten hasta 39 líneas de continuación (40 líneas por sentencia). El carácter de continuación puede cortar una cadena de caracteres, pero en este caso es necesario poner otro carácter *&* al comienzo de la línea siguiente.

La sentencia STOP *termina inmediatamente* la ejecución de un programa. Opcionalmente se le puede añadir una cadena de caracteres que se imprime al ejecutar esta sentencia, o una cadena de hasta seis dígitos que sirve para identificar el error.

2.1.5 TIPOS DE VARIABLES

Fortran 90 admite los siguientes 5 tipos de variables o constantes: INTEGER, REAL, COMPLEX, CHARACTER y LOGICAL.

Las variables se declaran en la forma:

```
INTEGER :: a, b, inercia, metro = 100

REAL :: gravedad = 9.81, tiempo, media = 0.5
```

Es posible inicializar las variables en el momento de declararlas. Si no se inicializan contienen basura informática y su uso resulta imprevisible.

Las *constantes reales* siempre deben llevar un punto decimal (.), aunque su valor sea entero.

Las *cadenas de caracteres* van entre "—" ó '—'. El carácter de cierre debe ser el mismo que el de apertura. En una cadena definida con "-" el apóstrofo ' se considera como tal, es decir como un carácter más de la cadena.

Para las variables CHARACTER se puede declarar la *longitud* de la cadena que contendrán.

```
CHARACTER (LEN = 20) :: nombre, primerApellido, segundoApellido
```

que es lo mismo que:

```
CHARACTER (20) :: nombre, primerApellido, segundoApellido
```

o que:

```
CHARACTER :: nombre*20, primerApellido*20, segundoApellido*20
```

También es posible definir una *longitud general* y establecer excepciones:

```
CHARACTER (20) :: nombre*30, primerApellido, segundoApellido,
iniciales*3
```

Fortran 90 no obliga a declarar las variables que se van a utilizar. Existe desde la primera versión una *convención de nombre implícita*, que establece que las variables que empiezan pos las letras I, J, K, L, M, N (ó i, j, k, l, m, n) son INTEGER. Si comienzan por cualquiera de las letras restantes son REAL. Sin embargo, para evitar errores es recomendable no utilizar esta característica del lenguaje, introduciendo la sentencia IMPLICIT NONE.

También permite trabajar con constantes y variables de tipo INTEGER o REAL con distinta precisión, es decir, con distinto número de cifras o rango. Por ejemplo, si se

utilizan variables INTEGER con 16 bits se pueden representar números entre -32768 y 32767; con 32 bits se pueden representar enteros entre -2147483648 y 2147483647. Las variables REAL se pueden representar en computadoras con 32 ó 64 bits; en el primer caso se tienen unas 7 cifras decimales equivalentes y en el segundo caso 16.

Las constantes o variables complejas se definen mediantes dos constantes reales entre paréntesis, separadas por una coma:

```
(1.23, -3.12), (1e-02, .3243-02)
```

Las variables complejas se declaran mediante la sentencia COMPLEX, que es similar a la REAL:

```
COMPLEX :: a, b

COMPLEX (KIND = 8), DIMENSION(100,100) :: X, Y
```

En Fortran hay dos constantes lógicas: TRUE y FALSE (los puntos antes y después de la correspondiente palabra son obligatorios).

Las variables lógicas se declaran con la palabra LOGICAL y se les asigna valor por medio de expresiones lógicas. Las variables lógicas pueden imprimirse con la sentencia PRINT sin formato.

2.1.6 CONSTANTES SIMBÓLICAS

Las constantes simbólicas (constantes con nombre) se declaran de modo análogo a las constantes, introduciendo la palabra PARAMETER a continuación del tipo, separados por una coma:

```
INTEGER, PARAMETER :: Max = 1000
```

En las constantes de tipo CHARACTER se puede definir la longitud implícitamente por medio del carácter (*):

```
CHARACTER (*), PARAMETER :: saludo = 'Hola', despedida = 'Adios'
```

Las constantes simbólicas se emplean luego en las expresiones aritméticas exactamente igual que las variables.

2.1.7 OPERADORES ARITMÉTICOS

Los operadores aritméticos binarios son:

Suma (+), Resta (-), Multiplicación (*), División (/) y Potenciación (**)

Operadores elementales: Signo positivo (+) y signo negativo (-).

No se pueden poner nunca dos operadores consecutivos. Si es necesario su uso, se utilizan los paréntesis para separarlos.

2.1.8 EXPRESIONES ARITMÉTICAS

Las expresiones aritméticas se definen mediante una combinación de variables, constantes numéricas y simbólicas, paréntesis, llamadas a función y operadores aritméticos.

El *orden de las operaciones aritméticas* en una expresión es el siguiente:

1. Se evalúa en primer lugar el operador potencia (**). Si hay varios, se ejecutan de derecha a izquierda.

2. Se evalúan a continuación las multiplicaciones (*) y divisiones (/). Si hay varios operadores de este tipo se evalúan de izquierda a derecha.

3. Finalmente se evalúan las sumas (+) y restas (-), también de izquierda a derecha.

Los *paréntesis* cambian el orden de realización de las operaciones: en primer lugar se realizan las operaciones del paréntesis más interior.

Las operaciones se realizan con el tipo de datos de los operandos. Por ejemplo, la suma de INTEGER es un INTEGER; la división de dos INTEGER es otro INTEGER, lo que quiere decir que se trunca el cociente.

Las *operaciones mixtas* (u operaciones en modo mixto) son operaciones entre variables o constantes de distinto tipo. Lo más frecuente es combinar INTEGER con REAL. En las operaciones mixtas el dato INTEGER se convierte antes en REAL, y el resultado es de tipo REAL. Por ejemplo, 5/2 es 2, mientras que 5/2.0 es 2.5.

Un caso particular interesante es el *operador potencia* (**). Si el exponente es INTEGER el resultado se obtiene por multiplicación repetida. Por el contrario, si el exponente es de tipo REAL el resultado se obtiene mediante la aplicación de logaritmos; esto quiere decir que la base no puede ser un número negativo, porque los números negativos no tienen logaritmo.

La prioridad de los operadores elementales es la misma que la de los operadores binarios correspondientes.

2.1.9 FUNCIONES NUMÉRICAS

Fortran 90 dispone de funciones numéricas para evaluar las funciones matemáticas más habituales. Algunas de estas funciones se muestran en la *Tabla 1 de la referencia*, que muestra también el tipo del argumento (o de los argumentos) y del resultado o valor de retorno.

Hay otras funciones numéricas en Fortran, como por ejemplo las funciones hiperbólicas (SINH, COSH, TANH).

Todas las funciones numéricas se utilizan incluyendo su nombre, seguido de los argumentos entre paréntesis y separados por comas, en alguna expresión aritmética. La función se evalúa y el resultado se incluye en dicha expresión aritmética.

Función	Significado matemático	Tipo argumento	Tipo resultado
ABS (x)	Valor absoluto de x	INTEGER ó REAL	Como el argumento
COS (x)	Coseno de x (en radianes)	REAL	REAL
EXP (x)	Función exponencial	REAL	REAL
INT (x)	Parte entera de x	REAL	INTEGER
FLOOR (x)	Mayor entero ≤ x	REAL	INTEGER
FRACTION (x)	Fracción de x	REAL	REAL
LOG (x)	Logaritmo natural de x	REAL	REAL
MAX (x1,…,xn)	Máximo de x1, x2, …xn	INTEGER ó REAL	Como el argumento
MIN (x1,…,xn)	Mínimo de x1, x2, …xn	INTEGER ó REAL	Como el argumento
MOD (x,y)	Resto de la división: x-INT(x/y)	REAL	INTEGER
NINT (x)	x redondeado al entero más proximo	INTEGER	REAL
REAL (x)	Se convierte x a Real	REAL	REAL
SIN (x)	Seno de x (en radianes?	REAL	REAL
RANDOM_NUMBER (x)	Subrutina que genera n° aleatorio 0≤x<1	REAL	REAL
SQRT (x)	Raíz cuadrada de x	REAL	REAL
TAN (x)	Tangente de x (en radianes)	REAL	REAL
ACON (x)	Arco coseno de x	REAL	REAL
ASIN (x)	Arco seno de x	REAL	REAL
ATAN (x)	Arco tangente de x	REAL	REAL
ATAN (x,y)	Arco tangente x/y (considera signos)	REAL	REAL

Tabla 1. Funciones intrínsecas para operaciones numéricas.

2.1.10 SENTENCIAS DE ASIGNACIÓN

La forma general de las sentencias de asignación hace uso del operador (=) en la forma:

```
Variable = expresión
```

Si la *variable* es de tipo REAL y la expresión es de tipo INTEGER, el resultado de la expresión se convierte a REAL antes de hacer la asignación.

En el caso contrario de que el resultado de una expresión de tipo REAL deba ser asignado a una variable INTEGER, se trunca la parte decimal o fraccional de la expresión y la parte entera se asigna a la variable INTEGER.

Estas sentencias de asignación entre distintos tipos de variables, con cambios de tipo implícitos, se consideran una metodología de programación peligrosa. Se considera preferible incluir funciones de cambio de tipo explícito, tales como REAL () ó INTEGER ().

También se puede utilizar sentencias de asignación con variables o constantes que representan cadenas de caracteres, por ejemplo en la forma:

```
Nombre = 'Alejandro'
```

2.1.11 SENTENCIAS DE ENTRADA/SALIDA SENCILLAS

Fortran utiliza sentencias de entrada/salida muy sencillas, en las que casi todas las opciones son las de defecto. Las primeras están controladas por la lista de variables a leer o escribir. Las segundas tienen formatos específicos que controlan todos los detalles.

2.1.11.1 Sentencias PRINT y WRITE

La sentencia PRINT permite imprimir variables y constantes en la salida estándar, lo habitual es la plataforma desde la que se está ejecutando el programa (por ejemplo, una ventana MS-DOS). A continuación se muestra la forma general y algunos ejemplos de sentencia PRINT sin formatos:

```
PRINT *, lista_de_variables_y_constantes_separadas_por_comas

PRINT *, 'Mi nombre es ', nombre, ' '

PRINT *, 'Velocidad = ', velocidad, 'm/s'

PRINT *
```

Cada sentencia PRINT empieza en una nueva línea. Si se utiliza sin ninguna variable o constante a imprimir, simplemente produce una línea en blanco.

La sentencia WRITE es similar a PRINT, con la posibilidad adicional de elegir otras unidades de salida tales como impresoras y archivos. También se puede utilizar WRITE sin variables, en cuyo caso imprime una línea en blanco.

```
WRITE (*, *) lista_de_variables_y_constantes_a_imprimir

WRITE (*, *)
```

2.1.11.2 Sentencia READ

La sentencia READ hace que se adquieran ciertos valores (normalmente desde el teclado) desde una o más *líneas de entrada* y que se asignen a la lista de variables de la sentencia. Cuando el programa llega a una sentencia READ espera a que el usuario introduzca los valores de las variables que deben ser leídas. Por este motivo siempre es conveniente, antes de llamar a la sentencia READ, imprimir un mensaje por la pantalla solicitando los datos. A continuación se muestra una forma general y un ejemplo de sentencia READ:

```
READ *, lista_de_variables_separadas_por_comas

READ *, velocidad, tiempo
```

Una línea de entrada es un conjunto de valores o constantes de distinto tipo. Una línea de entrada se puede introducir desde el teclado, o se puede leer desde un archivo. Se pueden leer cadenas de caracteres poniéndolas entre comillas simples o dobles.

Las constantes numéricas de la línea de entrada deben ser del mismo tipo que las variables de la lista, aunque a las variables reales es posible asignarles constantes enteras.

2.1.12 EJEMPLO COMPLETO DE PROGRAMA EN FORTRAN 90

A continuación se muestra un ejemplo sencillo pero completo para un programa en Fortran 90. Este programa, que se guarda en un archivo llamado *ecugrado2.f90*, resuelve la ecuación de segundo grado.

Los coeficientes A, B y C se leen de un archivo de datos llamado *datos.d*, que se supone están en el mismo directorio que el programa. A continuación se muestra el listado del programa numerando las sentencias para poderlas comentar más fácilmente:

```
1.   PROGRAM ecugrado2
2.      IMPLICIT NONE
3.      INTEGER :: error
4.      REAL (KIND=8) :: A, B, C, X, X1, X2, DISCR
5.      REAL (KIND=8), PARAMETER :: TOL=1e-10
6.      COMPLEX (KIND=8) :: XA, XB, PIM
7.
8.      ! Los datos se leen en el archivo datos.d
9.     OPEN(UNIT=10, FILE='datos.d', STATUS='OLD',
            ACTION='READ',& IOSTAT=error)
10.     IF (error /= 0) THEN
11.         PRINT *, 'Error de lectura del archivo'
12.         STOP 123456
13.     END IF
14.     READ (10,*) A, B, C
```

```
15.      PRINT *, ' ', 'A=', A, ' ', 'B=', B, ' ', 'C=', C
16.      DISCR = (B**2-4.0*A*C)
17.
18.      IF (DISCR>TOL) THEN
19.         X1 = (-B+SQRT(DISCR))/(2.0*A)
20.         X2 = (-B-SQRT(DISCR))/(2.0*A)
21.         PRINT *, 'Las soluciones son reales: X1=',X1,' X2=',X2
22.      ELSE IF (DISCR<-TOL) THEN
23.         PIM = SQRT(CMPLX(DISCR))
24.         XA = (-B+PIM)/(2.0*A)
25.         XB = (-B-PIM)/(2.0*A)
26.         PRINT *, 'Las soluciones son complejas: XA=',XA,' XB=',XB
27.      ELSE
28.         X = -B/(2.0*A)
29.         PRINT *, 'La raiz doble es: X=', X
30.      END IF
31.   END PROGRAM
```

A propósito de este programa se pueden hacer los siguientes comentarios:

1. Las sentencias 1 y 31 definen el comienzo y el final del programa, que se llama como se indica en la línea 1: ecugrado2.
2. Las sentencias 3-6 definen las variables que se van a utilizar. Es imprescindible definirlas, porque en la sentencia 2 se ha establecido así con IMPLICIT NONE. El parámetro KIND = 8 indica que las variables REAL y COMPLEX van a ser de 8 bytes (64 bits).
3. La palabra PARAMETER en la línea 5 indica que TOL es una constante simbólica, cuyo valor por tanto no puede cambiar a lo largo de la ejecución del programa.
4. La sentencia 9 abre para lectura (READ), un archivo ya existente (OLD), llamado *datos.d* (FILE) como unidad 10 (UNIT). Si la lectura es correcta, la variable INTEGER error tomará valor 0, pero si se produce un error tomará un valor positivo. En la sentencia 10 se comprueba que no ha habido error; si lo hay, la sentencia STOP detiene el programa imprimiendo el número que va a continuación.
5. En la sentencia 23 se calcula la raíz cuadrada de un número negativo. Para evitar un error es necesario convertir el argumento de REAL a COMPLEX con la función CMPLX().

2.2 ESTRUCTURAS

Al igual que en otros lenguajes de programación, las *estructuras* son agrupaciones de datos de naturaleza diferente. A cada uno de los datos que se agrupan bajo un mismo nombre se les llama *componentes* o *campos*. Por ejemplo, la estructura persona puede tener dos componentes CHARACTER que sean el nombre y los apellidos; un componente

INTEGER que sea el número de años, un componente REAL que sea la estatura en metros, etc.

Las estructuras se tratan como nuevos tipos de variables, con los que pueden formarse vectores y matrices, pasarse como argumentos a funciones y subrutinas, etc. Una estructura se define de la siguiente forma:

```
TYPE nombre_estructura
    declaracion_de_variables_1
    declaracion_de_variables_2
    declaracion_de_variables_3
    ...
END TYPE nombre_estructura
```

Esta definición se puede realizar después de IMPLICIT NONE y antes de la declaración de las variables. El nombre de la estructura debe de satisfacer las reglas de los identificadores de Fortran 90.

Una vez definida una estructura, pueden crearse constantes simbólicas o variables de esa estructura con algunas de las sentencias siguientes:

```
TYPE(nombre_estructura) :: lista_de_variables
TYPE(nombre_estructura), DIMENSION(10) :: A
TYPE(nombre_estructura) :: A(10)
```

Para inicializar una variable de este tipo se utiliza el nombre de la estructura seguido, entre paréntesis, por los valores adecuados para las distintas componentes, en el orden en que se definieron:

```
unaPersona = persona("Raúl", "Sandoval", 60, 1.70)
```

Por supuesto, el valor de una variable de una estructura puede ser asignado a otra variable del mismo tipo de estructura:

```
otraPersona = unaPersona
```

También se puede *pasar como argumento* a un subprograma una variable de tipo estructura, aunque es necesario en este caso que el argumento actual y el formal sean estructuras del mismo tipo.

Las estructuras, sin embargo, no se pueden *leer* o *imprimir* de un modo global, sino que hay que hacerlo componente a componente. Se *accede a una componente* particular de una estructura por medio del carácter *tanto por ciento* (%), como en el siguiente ejemplo:

```
PRINT *, unaPersona%nombre, ' ', unaPersona%apellidos
```

Esta forma de acceder a los componentes permite utilizarlos o modificarlos como se desee.

2.2.1 PUNTEROS

Los punteros son variables relacionadas con el contenido de una determinada dirección de memoria. El tipo de puntero está relacionado con el tipo de información almacenada en esa zona de la memoria.

2.2.1.1 Declaración y definición de punteros

Las variables puntero se declaran con el atributo POINTER, como por ejemplo:

```
TYPE fecha
INTEGER :: dia, mes, año
END TYPE fecha
REAL, POINTER :: xpuntero, ypuntero, zpuntero
TYPE (fecha), POINTER :: fechapuntero
```

Aunque se haya creado una variable puntero, dicha variable *no está asociada* todavía (no apunta) a una zona concreta de memoria. Se dice que está sin definir. Para reservar una zona de memoria a la que apunte esa variable puntero se utiliza la sentencia ALLOCATE, en una de las dos formas siguientes:

```
ALLOCATE (lista_de_variables_puntero)

ALLOCATE (lista_de_variables_puntero, STAT = var)
```

En la segunda de estas formas la variable *var* es cero si la reserva de memoria se realiza sin dificultades y adquiere un valor positivo si se produce una situación de error, como por ejemplo el que no haya memoria suficiente o que ya se haya reservado memoria previamente.

Además del estado *inicial* o *sin definir*, los punteros pueden estar *asociados* o *no-asociados* (a este estado también se llama *nulo*). Un puntero *asociado* es el que está relacionado con una zona de memoria concreta y *no-asociado* es el que ha perdido dicha asociación. La asociación se pierde por medio de la función NULLIFY:

```
NULLIFY(lista_de_variables_puntero)
```

Por otra parte, la función ASSOCIATED(varpuntero) devuelve TRUE o FALSE según el estado de *varpuntero* sea *asociado* o *no-asociado*.

La función DEALLOCATE se utiliza para *liberar la memoria* reservada con ALLOCATE. Se puede utilizar en cualquiera de las formas siguientes:

```
DEALLOCATE (lista_de_variables_puntero)
```

```
DEALLOCATE (lista_de_variables_puntero, STAT = var)
```

2.2.1.2 Asignación de punteros

Hay que distinguir entre “asignación de punteros” y “asignación de las variables asociadas a los punteros”.

La *asignación de punteros* significa que un puntero pasa a estar asignado a la zona de memoria asociada a otro puntero del mismo tipo. Se realiza mediante los caracteres (=>) en la forma:

```
puntero2 => puntero1
```

Después de ejecutarse esta sentencia ambos punteros están *asociados a la misma variable* en la memoria. La asociación que tuviera *puntero2* con otra variable se pierde al ejecutar esta sentencia.

Fortran 90 permite crear variables con el atributo TARGET. Estas variables tienen como finalidad el asociar punteros a ellas, como por ejemplo:

```
REAL, TARGET :: xpuntero
REAL, POINTER :: ypuntero

ALLOCATE(xpuntero)
ypuntero => xpuntero
```

La función ASSOCIATED (xpuntero, ypuntero) devuelve .TRUE. si ambos punteros están asociados con la misma zona de memoria. Análogamente, ASSOCIATED (pointer, target) devuelve .TRUE. si *pointer* está asociado con *target*.

2.2.1.3 Utilización de punteros en expresiones y en sentencias

Cuando una variable puntero se utiliza en una expresión aritmética o lógica, o bien en una sentencia de entrada/salida, automáticamente se sustituye por el contenido de la memoria asociada con dicho puntero1. La sentencia,

```
xpuntero = xpuntero + w*m -3
```

Utiliza el valor asociado con el puntero *xpuntero* para evaluar la expresión aritmética del miembro derecho y el resultado se almacena en la memoria asociada con *xpuntero*. En resumen, *xpuntero* se refiere a la misma zona de memoria antes y después de ejecutar esa sentencia, pero cambia el valor almacenado en esa zona de memoria.

En este sentido, *los operadores (=) y (=>) se comportan de modo diferente con punteros*. Considérense dos punteros *xpuntero* e *ypuntero* del mismo tipo pero apuntando o asociados con zonas de memoria diferentes. La sentencia,

```
ypuntero => xpuntero
```

hace que *ypuntero* deje de apuntar a donde apuntaba antes y apunte a la misma zona de memoria que *xpuntero* (a partir de esta sentencia ambos nombres representan el mismo valor en cualquier expresión en que se utilicen, porque están asociados con la misma zona de memoria). Si se ejecuta la sentencia *xpuntero = xpuntero−1* también cambia el valor representado por *ypuntero* en cualquier otra expresión. Sin embargo, la sentencia,

```
ypuntero = xpuntero
```

copia el contenido de memoria apuntado por *xpuntero* a la zona de memoria apuntada por *ypuntero.* Si uno de ellos cambia su valor, el otro permanece con el valor anterior.

2.2.1.4 Paso de punteros como argumentos a subprogramas

Los punteros se pueden pasar como argumentos a funciones y subrutinas, pero con algunas condiciones:

1) Los argumentos actual y formal correspondientes deben ser punteros del mismo tipo. 2) Si un puntero se utiliza como argumento formal, no puede tener atributo INTENT. 3) Si un argumento formal es un puntero (o un target), el correspondiente subprograma debe tener definida una *interface* de modo explícito.

El *valor de retorno* de una función puede ser un puntero definido localmente, pero en este caso hay que utilizar un RETURN para devolver el valor asociado con dicho puntero.

2.3 SENTENCIAS DE CONTROL: SENTENCIAS Y CICLOS

En ausencia de otros tipos de indicaciones, las sentencias de un programa Fortran se ejecutan de modo secuencial, es decir cada una a continuación de la anterior. Las sentencias de control permiten modificar esta ejecución secuencial. Las principales sentencias de control corresponden a estas dos categorías:

1. *Sentencias*: Dependiendo de que se cumpla o no cierta condición, permiten ejecutar o no una parte del código, o bien permiten ejecutar alternativamente unas u otras sentencias.
2. *Ciclos*: Los ciclos ejecutan repetidamente un conjunto de sentencias. Algunos ciclos utilizan un contador para ejecutarlas un número determinado de veces y otros utilizan una condición de salida o de terminación.

En ambas sentencias de control las expresiones lógicas constituyen una parte importante, porque permiten controlar la ejecución de dichas sentencias.

2.3.1 EXPRESIONES LÓGICAS Y OPERADORES RELACIONALES

Las expresiones lógicas pueden ser simples o compuestas. Se llaman expresiones lógicas simples las que constan de una constante lógica (TRUE y FALSE) o de una variable lógica, o bien de una expresión relacional en la forma:

```
expresión1 operador_relacional expresión2
```

donde tanto *expresión1* como *expresión2* son expresiones aritméticas, de caracteres o lógicas. Los operadores relacionales de Fortran se muestran en la Tabla 2.

Símbolo	Significado
< o .LT.	Menor que
> o .GT.	Mayor que
== o .EQ.	Igual que
<= o .LE.	Menor o igual que
>= o .GE.	Mayor o igual que
/= o .NE.	Distinto de

Tabla 2. Operadores relacionales.

Cada uno de los operadores relacionales tiene dos formas, que se pueden usar indistintamente. Para comparar cadenas de caracteres se utiliza el código ASCII de cada carácter; si es necesario, la cadena más corta se completa con blancos por la derecha.

Las *expresiones lógicas compuestas* constan de una o más expresiones lógicas combinadas mediante los *operadores lógicos*.

2.3.2 OPERADORES LÓGICOS

Los operadores lógicos de Fortran se muestran en la Tabla 3.

Operador	Significado
.NOT.	Negación
.AND.	Conjunción lógica
.OR.	Disyunción lógica
.EQV.	Equivalencia (.TRUE. si coinciden)
.NEQV.	Negación de equivalencia (.TRUE. si no coinciden)

Tabla 3. Operadores lógicos.

2.3.3 SENTENCIAS IF

Existen diversas formas de la sentencia IF para realizar bifurcaciones en la ejecución. Estas formas se presentan a continuación.

2.3.3.1 Sentencia IF simple

La forma más sencilla de la bifurcación IF es la siguiente:

```
IF (expresión_lógica) THEN
    sentencias

END IF
```

Las *sentencias* que están dentro del bloque se ejecutan sólo si la *expresión_lógica* es verdadera.

Existe todavía un IF más sencillo para cuando se trata de ejecutar o no una sola sentencia:

```
IF (expresión_lógica) sentencia
```

2.3.3.2 Sentencia IF compuesta

La forma general de la sentencia IF es la siguiente:

```
IF (expresión_lógica) THEN
    sentencias1
ELSE
    sentencias2
END IF
```

Si la *expresión_lógica* es verdadera se ejecutan *sentencias1* y si no lo es se ejecutan *sentencias2*.

2.3.3.3 Sentencia IF-ELSE IF

Esta es la forma más general de la sentencia IF, que permite encadenar varias condiciones lógicas.

```
IF (expresión_lógica1) THEN
    sentencias1
ELSE IF (expresión_lógica2) THEN
    sentencias2
ELSE IF (expresión_lógica3) THEN
```

```
    sentencias3
ELSE
    sentencias4
END IF
```

Pueden encadenarse tantas condiciones lógicas como se desee. Si ninguna de las expresiones lógicas es verdadera se ejecutan las sentencias que siguen a ELSE. Esta parte de la construcción es opcional; si no está presente y no se cumple ninguna de las condiciones, se ejecutan las sentencias que siguen al END IF.

2.3.3.4 Sentencias IF con nombre

Las sentencias IF pueden *anidarse*, es decir, dentro de las sentencias de un IF puede haber otras sentencias IF. En estos casos puede ser útil poner un nombre a cada IF, nombre que aparece al principio y final del bloque:

```
unNombre: IF (expresión_lógica) THEN
    sentencias1
ELSE
    sentencias2
END IF unNombre
```

En las sentencias IF-ELSE IF el nombre se puede poner también después de cada THEN o detrás del ELSE:

```
otroNombre: IF (expresión_lógica1) THEN
    sentencias1
ELSE IF (expresión_lógica2) THEN otroNombre
    sentencias2
ELSE IF (expresión_lógica3) THEN otroNombre
    sentencias3
ELSE otroNombre
    sentencias4
END IF otroNombre
```

2.3.3.5 Sentencia CASE

La sentencia CASE equivale en cierta forma a una sentencia IF-ELSE IF compleja. Su forma general es la siguiente:

```
SELECT CASE (selector)
    CASE (lista_de_valores1)
        Sentencias1
    CASE (lista_de_valores2)
        Sentencias2
```

```
    CASE DEFAULT
        Sentencias3
END SELECT
```

En este caso *selector* es una expresión entera, de caracteres o lógica. Cada *lista_de_valores* es un conjunto de posibles valores para el selector, entre paréntesis y separados por comas.

Cuando la ejecución llega a la sentencia SELECT CASE se evalúa la expresión *selector*, y a continuación las sentencias que siguen a la *lista_de_valores* que contiene el resultado de dicha evaluación. Si ninguna *lista_de_valores* contiene dicho resultado se evalúan las sentencias que siguen a DEFAULT.

2.3.3.6 Sentencia CASE con nombre

La sentencia CASE también puede tener nombres, que se utilizan en la forma:

```
unNombre: SELECT CASE (selector)
    CASE (lista_de_valores1)
        Sentencias1
    CASE (lista_de_valores2)
        Sentencias2
    CASE DEFAULT
       Sentencias3
END SELECT unNombre
```

2.3.4 CICLOS

Los ciclos se utilizan para *repetir la ejecución* de un conjunto de sentencias un determinado número de veces o hasta que se cumpla una determinada condición. Es posible referirse a estos casos como *ciclos controlados por contador* o *ciclos controlados por expresión lógica*, respectivamente.

2.3.4.1 Ciclos DO controlados por contador

La primera forma de este ciclo es la siguiente:

```
DO variable_de_control = valor_inicial, valor_final, incremento
    Sentencias
END DO
```

El modo de ejecutarse este ciclo es el siguiente: la *variable_de_control* toma el *valor_inicial* y se compara con *valor_final*. Las Sentencias se ejecutan si dicha

variable_de_control es menor o igual que *valor_final*, si el incremento es positivo. En caso contrario, se ejecuta la siguiente línea después del END DO. El incremento puede ser negativo.

Los valores inicial, final y el incremento de la variable de control se determinan antes de que el ciclo comience a ejecutarse, y no deben ser modificados durante la ejecución del ciclo. La *variable_ de_control* puede ser utilizada en las *Sentencias*, pero no debe ser modificada en ellas. Si el valor del incremento se omite, este se tomará igual a 1 por defecto.

2.3.4.2 Ciclos DO generales (controlados por expresión lógica)

La forma general del ciclo DO controlado mediante una expresión lógica es la siguiente:

```
DO
    Sentencias1
    IF (expresión_lógica) EXIT
    Sentencias2
END DO
```

La ejecución del ciclo se termina cuando *expresión_lógica* se hace verdadera. Con esta construcción es posible que *Sentencias1* se ejecute sólo una vez y que *Sentencias2* no se ejecuten ninguna. Según las sentencias que se pongan delante o detrás del IF se puede hacer que el control se realice al principio del ciclo, al final o en un punto intermedio.

También es posible bloquear la computadora si la condición de salida no se cumple nunca. Para evitar esto es conveniente introducir un *contador de iteraciones* e incluir en la condición de salida el que el contador haya alcanzado un valor máximo previamente fijado.

2.3.4.3 Sentencia CYCLE

La sentencia CYCLE introducida entre las demás sentencias del ciclo hace que se *termine la iteración actual* (saltando las sentencias que están entre ella y el END DO) y se comience una nueva. Esta sentencia se puede utilizar con cualquiera de los ciclos DO.

2.3.4.4 Ciclos DO con nombre

Los ciclos con nombre no solamente son más fáciles de identificar, sino que también presentan otras importantes ventajas prácticas en el caso de los *ciclos anidados*, pues el nombre puede también asociarse a las sentencias EXIT y CYCLE.

```
ciclo_1: DO
    Sentencias1
    ciclo_2: DO
        Sentencias2
        IF (expresión_lógica1) THEN
            EXIT ciclo_2
        ELSE IF (expresión_lógica2) THEN
            EXIT ciclo_1
        END IF
    END DO ciclo_2
END DO ciclo_1
```

Con esta construcción, si *expresión_lógica1* es verdadera se abandona el *ciclo_2*, mientras que si *expresión_lógica2* es la que se hace verdadera se abandonaría el *ciclo_1*.

2.4 SENTENCIAS DE ENTRADA/SALIDA

Fortran tiene dos tipos de sentencias de E/S (Entrada/Salida): con *formato fijo* y con *formato libre*, también llamadas *controladas por formato*, o *controladas por lista de constantes y variables*.

Existen unas unidades de entrada y de salida estándar, que suelen ser el teclado y la pantalla, respectivamente. Ya se verá cómo estas unidades pueden ser especificadas en el momento de leerlo y/o escribir.

2.4.1 SENTENCIA PRINT

La forma general de la sentencia PRINT es la siguiente:

```
PRINT formato, lista_de_constantes_y_variables
```

Donde *formato* es una de las opciones siguientes:

a) Un asterisco (*): En este caso se trata de un formato libre dependiente del compilador utilizado.
b) Una constante o variable de tipo CHARACTER (o una expresión de este tipo, o bien un vector de caracteres), que especifica los formatos con los que se imprimirán las variables de la lista.
c) La etiqueta numérica (de 1 a 99999) de una sentencia FORMAT que especifica el formato.

Por lo general, cada sentencia PRINT da comienzo a una nueva línea, de tal forma que si se ejecuta una sentencia PRINT vacía, simplemente se deja una línea en blanco. A continuación se ofrecen las dos formas generales de sentencias PRINT con formato:

```
PRINT 'descriptores_de_formato', lista_de_constantes_y_variables
PRINT label, lista_de_constantes_y_variables
label FORMAT(descriptores_de_formato)
```

2.4.1.1 Espaciados y saltos de página

El primer carácter de cada línea no se imprime y es considerado como un carácter de control que determina el espaciado vertical. Las posibles opciones son:

Como la opción más frecuente es la primera, se suele introducir un carácter en blanco al comienzo de cada lista de variables:

```
PRINT 'A1,F12.4,A4', ' ', Velocidad, ' m/s'
```

2.4.1.2 Descriptores de formato

Fortran dispone de numerosos descriptores de formato adaptados a los distintos tipos de variable y a las distintas formas de representarlos.

Formatos		Significado
Iw	Iw.m	Constantes y variables INTEGER
Bw	Bw.m	Datos INTEGER en formato binario
Ow	Ow.m	Datos INTEGER en formato octal
Zw	Zw.m	Datos INTEGER en formato hexadecimal
Fw.d		Datos Real con notación decimal
Ew.d	Ew.dEe	Datos Real con notación exponencial
ESw.d	ESw.dEe	Datos Real con notación científica
ENw.d	ENw.dEe	Datos Real con notación ingenieril
Gw.d	Gw.dEe	Datos Real con formato adaptable
A	Aw	Caracteres
"*…*"	'*…*'	Cadenas de caracteres
Lw		Datos LOGICAL
Tc	TLn TRn	Tabuladores
nX		Espacios en horizontal
/		Espacio Vertical
:		Control para lectura de caracteres

Tabla 4. Distintos especificadores de formato de la sentencia PRINT.

El significado de los "campos" que aparecen en la Tabla 4 es el siguiente:

w número entero positivo que indica la anchura del campo (por ejemplo, 6 espacios)
m número entero igual o mayor que cero que indica un nº mínimo de dígitos a leer o imprimir
d número entero positivo que indica el número de cifras a la derecha del punto decimal
e número entero igual o mayor que cero que indica el nº de cifras en un exponente
* un carácter
c número entero positivo que representa la posición de un carácter

n número entero positivo que indica un nº de caracteres

2.4.1.3 Formato para números enteros

Por ejemplo, el formato I5 indica que se dispone de cinco espacios para imprimir un número entero; si el número es positivo y tiene menos de cinco cifras se ajusta por la derecha y se completa con blancos por la izquierda. Si es lo primero que se imprime en una línea el primer carácter por la izquierda se toma como carácter de control y no se imprime. En principio el signo se imprime sólo si es negativo. Si el número a imprimir precisa más de cinco espacios se imprimen cinco asteriscos (*****), como forma de advertir que se ha producido un error.

2.4.1.4 Formatos para números reales

Los formatos reales −descriptores F, E, ES y EN− también ajustan el contenido por la derecha. Por ejemplo, el formato F10.4 indica que se reserva un total de 10 espacios para imprimir el número y que aparecerán 4 cifras a la derecha del punto decimal. Si tiene menos de cuatro cifras decimales se completa con ceros por la derecha. Si necesita más de 10 espacios se imprimen asteriscos (*) en toda la anchura del campo.

El formato E12.6 imprime el número en forma exponencial normalizada: un signo − opcional, un cero, un punto decimal seguido de 6 cifras, la letra E y el exponente de 10 con 4 cifras (con 2 cifras si se hubiera definido como E12.6e2). También se completa con blancos por la izquierda y con asteriscos si la anchura del campo es insuficiente para representar el número a imprimir.

La notación científica ES es similar a la E, pero la mantisa se normaliza de otra forma, de modo que mayor o igual que 1 y menor que 10 (a no ser que sea cero). La notación ingenieril EN es también similar, pero en este caso la mantisa es mayor o igual que 1 y menor que 1000 (salvo que sea cero) y el exponente siempre ha de ser múltiplo de tres.

2.4.1.5 Formatos para cadenas de caracteres

Las cadenas de caracteres se imprimen mediante el descriptor A, que tiene dos formas: Si se especifica A se imprime la cadena ocupando los espacios que necesita; si se especifica A20 la salida impresa tendrá 20 caracteres. Si la cadena tiene menos caracteres se ajusta por la derecha y se rellena con blancos por la izquierda. Si la cadena tiene más de 20 caracteres, se imprimen los 20 caracteres que están más a la izquierda.

2.4.1.6 Formatos de espaciado horizontal (X y T)

El formato 12X deja 12 espacios en blanco en la línea de salida. Por otra parte, el formato T32 indica que el siguiente dato se imprimirá en esa línea a partir de la posición 32.

2.4.1.7 Factor de repetición de formatos

El factor de repetición se antepone a uno de los formatos de enteros o reales vistos previamente. Por ejemplo, los formatos 3I6 y 5F10.3 permiten respectivamente imprimir tres enteros con seis espacios para cada uno, y cinco números reales con una anchura de 10 posiciones y 3 cifras decimales.

Es posible también repetir un conjunto de descriptores encerrándolos entre paréntesis y poniendo el factor de repetición antes del paréntesis, como por ejemplo: 3(I5,2X,F12.4).

2.4.1.8 Descriptor de cambio de línea /

El descriptor barra / hace que se produzca un salto a la línea siguiente. Se puede poner varias veces seguidas (///), o se puede utilizar un factor de repetición (3/).

2.4.1.9 Correspondencia entre la lista de variables y los descriptores de formato

Fortran establece una correspondencia entre la lista de variables a imprimir y la lista de descriptores de formato. Esta correspondencia sigue las siguientes reglas:

1. Si la lista de variables se agota antes que la lista de formatos, la lista de descriptores se continúa ejecutando, imprimiéndose las constantes, los espaciados y los tabuladores, hasta que se produce una de las condiciones siguientes:

    a. Se llega al cierre de paréntesis que determina el fin de la lista de descriptores
    b. Se encuentra un descriptor de uno de los tipos I, F, E, ES, EN, A, L, G, B, O ó Z.
    c. Se encuentra una coma (,)

2. Si la lista de descriptores de formato se agota antes que la lista de variables, se comienza una nueva línea de salida y se vuelve a empezar con la lista de descriptores desde el principio si ésta no tiene paréntesis. Si hay paréntesis internos se repiten los descriptores contenidos en el paréntesis interno más a la derecha, incluyendo el factor de repetición si lo tuviera. Considérese el siguiente ejemplo:

```
PRINT 100, I, A, I+1, A+1.0, I+2, A+2.0, I+3, A+3.0
100 FORMAT(1X, I10,F14.4, / (1X, I6, F10.2))
```

Si I = −100 y A = 3.141592654, la salida que se obtiene es la siguiente:

```
-100    3.142
-99     4.14
-98     5.14
-97     6.14
```

2.4.2 ENTRADA DE DATOS CON FORMATO (SENTENCIA READ)

La lectura de datos se realiza con la sentencia READ. Su forma general, con formato, es:

```
READ especificadores_de_formatos, lista_de_variables
```

En este caso los especificadores o descriptores de formatos son muy similares a los vistos anteriormente. La principal diferencia es que en los formatos de entrada no pueden aparecer constantes con cadenas de caracteres, y que la presencia de comas (,) como separador es ignorada.

2.4.2.1 Lectura de variables INTEGER

La lectura de variables INTEGER se hace mediante campos del tipo rIw, como por ejemplo:

```
READ '(I5,I10,I5)', I, J, K
```

que es equivalente a:

```
READ 100, I, J, K
100 FORMAT(I5,I10,I5)
```

Los datos pueden disponerse separados por más o menos espacios en blanco, pues los espacios por defecto se ignoran en las líneas de entrada. Los datos pueden prepararse también sin ningún tipo de separadores (ni espacios, ni comas).

2.4.2.2 Lectura de variables REAL

Los valores correspondientes a variables REAL pueden prepararse con o sin puntos decimales. Si no se introducen puntos decimales, los puntos se suponen exactamente donde lo indican los formatos F, E, ES ó EN.

2.4.2.3 Lectura de variables CHARACTER

Cuando la lista de variables de una sentencia READ contiene variables CHARACTER, se leen tantos caracteres de la línea de entrada como longitud o número de caracteres tienen las variables correspondientes.

Por ejemplo, considérense las variables frase1 y frase2 y la siguiente sentencia READ:

```
CHARACTER (8) :: frase1, frase2
READ 100, frase1, frase2
100 FORMAT(2A)
```

2.4.3 SENTENCIA WRITE

La sentencia WRITE es más complicada y tiene más posibilidades que la sentencia PRINT. Su forma general es la siguiente:

```
WRITE (lista_de_control) lista_de_constantes_y_variables
```

La lista_de_control puede contener los elementos siguientes:

1. Una *unidad de escritura*, que es una expresión entera que indica un dispositivo de salida. Si es un *asterisco* (*) se utiliza la unidad de salida estándar (la pantalla o una impresora). Esta unidad se puede poner también en la forma UNIT=expresion_entera, lo que hace que el contenido quede más claro.
2. Un conjunto de especificadores de formatos, tales como los presentados para PRINT. Pueden escribirse sin más, como en PRINT, pero en ese caso deben ser el 2º item de la *lista_de_control*; también pueden incluirse en la forma FMT=especificadores_formato.

3. Una especificación ADVANCE en la forma ADVANCE=conjunto_de_caracteres, donde *conjunto_de_caracteres* es "NO" o "YES" después de eliminar espacios en blanco y pasar a mayúsculas. Esta especificación se utiliza para indicar si al final de la actual operación de lectura hay que avanzar o no a la siguiente línea. La opción por defecto es 'YES'.
4. Otros elementos de utilidad para operaciones de lectura/escritura de archivos.

A continuación se muestran algunas de las sentencias WRITE:

```
WRITE (*, *) A, B, C ! equivalente a PRINT *; A, B, C

WRITE (6, FMT='3F10.4') A, B, C

WRITE (UNIT=unidadSalida, *) A, B, C
```

2.4.4 SENTENCIAS PARA LECTURA/ESCRITURA DE ARCHIVOS

Uno de los principales usos de las formas generales de READ y WRITE es para operaciones de lectura y/o escritura de archivos.

La primera operación que se debe realizar es la de abrir el archivo, asociándolo con un determinado número de unidad. Los archivos se abren por medio de la sentencia OPEN, que tiene la forma:

```
OPEN (lista_de_apertura)
```

donde *lista_de_apertura* incluye los siguientes especificadores:

1. Un número especificador de unidad, que será luego la unidad a la que se hará referencia en las sentencias READ y WRITE.
2. Una cláusula FILE = nombre_de_archivo.
3. Una cláusula STATUS = *cadena_de_caracteres*, donde cadena_de_caracteres es una de estas posibilidades: 'OLD', 'NEW', 'REPLACE' ó 'UNKNOWN'. La primera se utiliza si el archivo ya existe; la segunda cuando el archivo debe ser creado por el programa por primera vez, en cuyo caso OPEN crea un archivo vacío y cambia el estado a 'OLD'; con 'REPLACE' se crea un nuevo archivo, borrando uno anterior con el mismo nombre si existe, y cambiando el estado a 'OLD'.
4. Una cláusula ACTION = operacion, donde *operacion* es una cadena de caracteres cuyos valores pueden ser 'READ', 'WRITE' ó 'READWRITE', lo que determina si el archivo se abre sólo para lectura, sólo para escritura, o para ambas operaciones. Si se omite esta cláusula, lo normal es que el archivo se abra para lectura y escritura.
5. Una cláusula POSITION = cadena_de_caracteres, donde *cadena_de_caracteres* tiene como posibles los valores siguientes: 'REWIND', 'APPEND' ó 'ASIS'. En el primer caso el cursor se situaría al comienzo del archivo, en el segundo al final y

en el tercero se quedaría donde estaba. Si se omite esta cláusula el cursor sigue donde está si el archivo estaba ya abierto, o se crea el archivo y el cursor se coloca al principio si el archivo no existía.

6. Una cláusula IOSTAT = variable_de_estado, donde *variable_de_estado* es una variable entera a la que se asigna un valor según haya transcurrido la operación. De ordinario se asigna un cero si el archivo se abre sin problemas, un número positivo si se produce alguna condición de error, y un número negativo si se ha llegado al fin del archivo sin que se produzca un error.

Por ejemplo, a continuación se indica cómo se abre un archivo llamado *datos.d* para lectura:

```
OPEN(UNIT=10, FILE='datos.d', STATUS='OLD', ACTION='READ', &
     POSITION='REWIND', IOSTAT=error)
```

Otro ejemplo de apertura de archivos es el siguiente:

```
OPEN(UNIT=10, FILE=miArchivo, STATUS='NEW', ACTION='WRITE', &
     POSITION='REWIND', IOSTAT=error)
```

Una vez terminadas las operaciones de lectura y/o escritura, se debe proceder a cerrar el archivo, lo cual se realiza con la sentencia CLOSE, cuya forma general es la siguiente:

```
CLOSE (lista_de_clausura)
```

donde *lista_de_clausura* incluye necesariamente el número de unidad, y opcionalmente puede incluir las cláusulas IOSTAT, ERR y STATUS. Los archivos no cerrados explícitamente se cierran de modo automático al llegar a una sentencia STOP ó END.

2.5 FUNCIONES Y SUBRUTINAS

A diferencia de otros lenguajes de programación como C/C++ y Java, Fortran dispone de dos tipos de subprogramas: las *funciones* y las *subrutinas*. La principal diferencia es que las funciones tienen un *valor de retorno* y pueden por tanto ser utilizadas en expresiones aritméticas o de otro tipo. Por el contrario, las subrutinas no devuelven ningún valor y todos sus resultados los transmiten a través de *modificaciones de sus argumentos*.

2.5.1 FUNCIONES

No obstante que el lenguaje Fortran proporciona un gran número de funciones de librería, el usuario también puede programar sus propias funciones.

Las funciones constan de las partes siguientes:

1. Encabezamiento o sentencia FUNCTION.
2. Sentencias de especificación.
3. Sentencias ejecutables.
4. Sentencia END FUNCTION.

La sentencia FUNCTION es la primera sentencia de cualquier función y tiene la forma siguiente:

```
[tipo_retorno] FUNCTION nombre_funcion(lista_de_argumentos)
```

donde *tipo_retorno* es un identificador de tipo opcional (REAL, INTEGER, …), *nombre_funcion* es un identificador válido de Fortran, y *lista_de_argumentos* es una lista de variables que son utilizadas para pasar información a la función. A estos argumentos que aparecen en el encabezamiento de la función se les llama *argumentos formales*, para distinguirlos de los *argumentos actuales*, que son los que aparecen en la llamada a la función desde otro programa.

En la parte de las sentencias de especificación deben determinarse:

1. El tipo del valor de retorno, si no se ha especificado en la sentencia FUNCTION.
2. Los tipos de todos y cada uno de los argumentos formales. En esta especificación puede aparecer el calificativo INTENT, que sirve para indicar cómo se usa ese argumento. Por ejemplo, INTENT(IN) indica que esa variable es sólo *de entrada* y que no se deberá modificar en la función. Por ejemplo:

```
REAL FUNCTION EspacioRecorrido(velocidad, tiempo)
REAL, INTENT(IN) :: velocidad, tiempo
```

Las sentencias ejecutables son distintas sentencias de Fortran que realizarán diversas operaciones, pero que siempre deberán calcular el valor de retorno antes de devolver el control. Esto se hace *asignando valor* a una variable cuyo nombre y tipo es el de la función.

La última sentencia de la función tiene la forma:

```
END FUNCTION nombre_funcion
```

o bien, para las *funciones externas* (las definidas después de la sentencia END PROGRAM):

```
END
```

Téngase en cuenta que *la forma de llamar a la función* en el programa principal es por medio del *nombre seguido de los argumentos actuales* entre paréntesis, en una expresión que sea compatible con el tipo del valor de retorno de la función.

2.5.2 SUBRUTINAS

2.5.2.1 Analogías y diferencias entre funciones y subrutinas

Las *subrutinas* son subprogramas con muchas cosas en común y algunas diferencias con las *funciones*. De modo análogo que las funciones, las subrutinas:

− Son llamadas desde un *programa principal* o desde otra subrutina para realizar una tarea determinada.

− Constan de un *encabezamiento* con la palabra SUBROUTINE y una lista de argumentos entre paréntesis, unas declaraciones, un cuerpo de sentencias ejecutables y la sentencia END.

− Pueden ser *internas*, formar parte de un *módulo* o ser *externas*.

− Las reglas de *visibilidad* y *permanencia* de variables son las mismas que para las funciones.

− Su nombre puede ser pasado como argumento a otro subprograma y pueden llamarse a sí mismas, es decir pueden ser *recursivas*.

Sin embargo, las subrutinas se diferencian de las funciones en que:

− Las funciones tienen siempre un valor de retorno, que toma valor a través de una variable con el mismo nombre que la función. Las subrutinas *no tienen valor de retorno* propiamente dicho y siempre devuelven los resultados a través de *modificaciones de los argumentos formales* que se transmiten a los argumentos actuales. En ocasiones las subrutinas no devuelven nada y se limitan por ejemplo a imprimir ciertos valores.

− Las funciones se llaman incluyendo su nombre seguido de los argumentos en una expresión. Las subrutinas se llaman siempre a través de una sentencia CALL.

2.6 VECTORES Y MATRICES

Una característica fundamental de cualquier lenguaje de programación es la de poder trabajar con vectores y matrices, a los que se hace referencia con un nombre genérico seguido de uno o dos subíndices.

2.6.1 DECLARACIÓN DE UNA VARIABLE COMO VECTOR

Para declarar un vector de nombre *x* con 1000 elementos de tipo REAL se utiliza el atributo DIMENSION seguido del número de elementos entre paréntesis, en alguna de las formas siguientes:

```
REAL, DIMENSION(1000) :: x

REAL :: y(1000)

INTEGER DIMENSION(50:50) :: peso

INTEGER :: peso(50:50)
```

La definición de matrices es similar, utilizando dos subíndices separados por comas:

```
REAL, DIMENSION(100,100) :: A

INTEGER, DIMENSION(100,4) :: matriz
```

La definición de las dimensiones de los vectores y matrices se puede hacer por medio de constantes simbólicas, como por ejemplo:

```
INTEGER, PARAMETER :: ini=-10, fin=10, n=40

REAL, DIMENSION(ini:fin) :: x

REAL, DIMENSION(n, n) :: A, B
```

Fortran 90 admite un máximo de 7 índices, es decir, permite trabajar con hipermatrices hasta de orden 7.

2.6.2 MEMORIA DINÁMICA

En muchas ocasiones se sabe ya en *tiempo de compilación* el tamaño que deben tener los vectores y matrices que se van a utilizar en un determinado programa. En otras muchas ocasiones sólo en *tiempo de ejecución*, después de leer los datos del problema, se sabe de qué tamaño deben ser los vectores y matrices para los que hay que reservar memoria. Una posibilidad es reservar memoria para el caso más grande que se pueda presentar y que pueda ser estudiado por la computadora disponible.

Para declarar uno o más vectores y/o matrices dejando para más adelante la definición de su dimensión se utiliza el calificador ALLOCATABLE, como en las sentencias siguientes:

```
REAL, DIMENSION(:), ALLOCATABLE :: x, y
REAL, DIMENSION(:,:), ALLOCATABLE :: matrizA, matrizB
```

Después, en tiempo de ejecución, se da valor a la dimensión con la sentencia ALLOCATE. Las variables que se utilizan para asignar la dimensión deben ser INTEGER. Considérese el ejemplo siguiente:

```
READ *, n
ALLOCATE(x(n), y(n:n))
ALLOCATE(matrizA(n,n), matrizB(n:n, n:n)
```

2.6.3 LIBERACIÓN DE LA MEMORIA DINÁMICA

Cuando una variable para la que se ha reservado memoria dinámicamente ya no se va a utilizar, es conveniente liberar esa memoria con objeto de que quede disponible para otros posibles usos que puedan surgir. Para liberar la memoria se utiliza la sentencia DEALLOCATE, como por ejemplo:

```
DEALLOCATE(x, y, matrizA, matrizB)
```

2.6.4 OPERACIONES CON VECTORES Y MATRICES

Las operaciones permitidas son las siguientes:

1. Asignación de valor a partir de un escalar.
2. Asignación de valor a partir de otro vector, matriz o expresión del mismo tamaño.
3. Multiplicación por un escalar.
4. Combinación lineal de vectores o matrices.

No se permiten otras operaciones, tales como:

1. Producto escalar de vectores (se puede hacer con la función DOT_PRODUCT( )).
2. Producto de matrices (se puede hacer con la función MATMUL( )).
3. Otras operaciones matriciales (como la inversión, las diversas factorizaciones, etc.) que deben realizarse con funciones o subrutinas programadas por el usuario o importadas de distintas librerías.

Considérense los siguientes ejemplos:

```
z = x + y
w = x + ABS(y)*2
v = (x < 0) .AND. (MOD(y,3) == 1)
```

Es posible asignar a un vector o matriz el resultado de una operación vectorial o matricial o un escalar. Véanse los ejemplos siguientes:

```
vector = expresion_con_vectores
vector = 6
matrizA = 10
```

A la hora de operar con vectores y matrices, tanto a la derecha como a la izquierda del operador de asignación (=) se pueden utilizar partes del vector o *subvectores* y partes de la matriz o *submatrices*. Para ello se pueden utilizar los tres números o variables típicos de direccionamiento de Fortran: *valor_inicial:valor_final:salto*.

```
x = y(2:20:2)
matrizB = matrizA(2:20:2, 1:15)
```

Los subvectores y submatrices también se pueden utilizar en las sentencias de entrada/salida. En cualquier caso las dimensiones de los distintos operandos y del resultado deben ser las correctas.

2.6.5 VECTORES Y MATRICES EN FUNCIONES Y SUBRUTINAS

Existen algunas funciones intrínsecas que admiten vectores o matrices como argumentos. Algunas de las más utilizadas son las mostradas en la Tabla 5.

Función	**Significado matemático**
ALLOCATED (x)	Devuelve .TRUE. si se ha reservado memoria para x
DOT_PRODUCT(x,y)	Devuelve el producto escalar de los vectores x e y
MAXVAL(x)	Devuelve el valor máximo del vector x
MAXLOC(x)	Devuelve un vector con un elemento cuyo valor indica la posición de la 1ª vez que aparece el valor máximo de x
MINVAL(x)	Devuelve el valor mínimo del vector x
MINLOC(x)	Devuelve un vector con un elemento cuyo valor indica la posición de la 1ª vez que aparece el valor mínimo de x
PRODUCT(x)	Devuelve el valor del producto de los elementos x
SIZE(x)	Devuelve el número de elementos de x
SUM(x)	Devuelve el valor de la suma de los elementos de x
MATMUL(A,B)	Devuelve el resultado del producto matricial de matrices y/o vectores

Tabla 5. Funciones intrínsecas que admiten argumentos vectoriales o matriciales.

Las funciones de usuario también pueden tener vectores o matrices como argumentos formales. En este caso los argumentos vectores y/o matrices deben ser declarados tanto en el programa principal como en la función o subrutina. También los valores de retorno pueden ser un vector o una matriz.

Se permite pasar como argumento, además de vectores y matrices, las d*imensiones* de estos vectores y matrices. Por ejemplo, la función *normaVect(x,n)* puede recibir como argumentos el vector cuya norma tiene que calcular y el número de elementos de dicho vector:

```
FUNCTION normaVect(x,n)
    INTEGER, INTENT(IN) :: n
    REAL DIMENSION(n), INTENT(IN) :: x
    REAL :: normaVect=0.0
```

```
    INTEGER :: i
    DO i=1,n
        normaVect = normaVect + x(i)*x(i)
    END DO
    normaVect = SQRT(normaVect)
END FUNCTION normaVect
```

Los vectores y matrices que se pasan como argumentos pueden haber sido creados estática o dinámicamente, por medio de la sentencia ALLOCATE. Sin embargo, dentro de la función o subrutina, nunca se utiliza memoria dinámica para los argumentos.

El paso de vectores y matrices como argumentos es tan frecuente, que Fortran 90 dispone de una forma sencilla para hacer que los vectores y matrices tomen automáticamente dentro de la función o subrutina el mismo tamaño que tenían los argumentos actuales en el programa principal. Para ello basta utilizar el carácter *dos puntos* (:) en la sentencia DIMENSION, como por ejemplo:

```
SUBROUTINE ElimGauss (matrizA, vectorb, x)
    REAL DIMENSION(:,:) :: matrizA
    REAL DIMENSION(:) :: vectorb, x
    ...
END SUBROUTINE ElimGauss
```

Si en el programa principal se define la INTERFACE de la función o subrutina, es posible utilizar también este sistema de definición automática de la DIMENSION.

Para definir dentro de una función o subrutina vectores o matrices locales que tengan el mismo tamaño que otros vectores o matrices que hayan llegado como argumentos, es posible utilizar la sentencia SIZE, como por ejemplo:

```
SUBROUTINE ElimGauss(matrizA, vectorb, x)
    REAL DIMENSION(:,:) :: matrizA
    REAL DIMENSION(SIZE(matrizA)) :: matrizB
    REAL DIMENSION(:) :: vectorb, x
    ...
END SUBROUTINE ElimGauss
```

2.6.6 SENTENCIAS DE ENTRADA/SALIDA CON VECTORES Y MATRICES

Se pueden utilizar expresiones abreviadas en la lectura o escritura de un vector o de una matriz:

```
READ (10,*) (x(i), i=1,n)
PRINT (100, *), x(1:n)
READ *, ((matrizA(i,j), i=1,m), j=1,n)
DO i=1,m
    READ *, (matrizB(i,j), j=1,n)
```

```
END DO
```

CAPÍTULO 3.- DESARROLLO DE UN PROGRAMA PARA LA SOLUCIÓN DE SISTEMAS DE ECUACIONES LINEALES

3.1 PRESENTACIÓN

Con la finalidad de proporcionar una herramienta para optimizar el proceso de solución de sistemas de ecuaciones lineales, se desarrolló un programa que resuelve este tipo de sistemas de ecuaciones por diversos métodos.

El programa se llama SOLECLIN y fue elaborado en lenguaje FORTRAN 90. Los métodos para solucionar sistemas de ecuaciones que se incluyen son:

- Método de Gauss-Jordan
- Método de la Inversa
- Método de Cholesky
- Método de Gauss-Seidel

3.2 CÓDIGO FUENTE

```
PROGRAM TESINA

IMPLICIT NONE
REAL:: M, TOL
INTEGER:: I, J, R, C, N, P, O, ITER
REAL, POINTER:: A(:,:), B(:,:), MI(:,:), L(:,:), U(:,:), X(:,:), XO(:,:)
CHARACTER(2):: OTRA, OP

100 FORMAT (/,3X,A)
200 FORMAT (/,10X,A,I2,A,I2,A,\)
300 FORMAT (/,10X,A,I2,A,I2,A,\)
400 FORMAT (3/,25X,A,A,\)
500 FORMAT (2/,10X,A,/)
600 FORMAT (10X,A,I3,A,2/)
700 FORMAT (10X,F12.5,\)


PRINT*, '-------------------------------------------------------------'
PRINT '(/,10X, A, 2/)', 'PROGRAMA PARA SOLUCIONAR SISTEMAS DE ECUACIONES
LINEALES'
PRINT '(8X, A, 4/)',   'PROGRAMA ELABORADO POR:   >> PAUL ALEJANDRO
SANDOVAL HUERTA <<'

1000 PRINT*, '-------------------------------------------------------'
```

```
PRINT '(3X,A)', 'ELIJA EL METODO:'
PRINT 100, ' (1) METODO DE GAUSS-JORDAN '
PRINT 100, ' (2) METODO DE LA INVERSA   '
PRINT 100, ' (3) METODO CHOLESKY        '
PRINT 100, ' (4) METODO GAUSS-SEIDEL    '
PRINT '(2/,5X,A,\)', '* OPCION: '
READ*, OP


!METODO DE GAUSS-JORDAN  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PRINT '(2/,A)', '-------------------------------------------------------'
IF (OP=='1') THEN
      PRINT '(3/,25X,A,3/)', '>>  METODO DE GAUSS-JORDAN  <<'

2000 PRINT*, '-------------------------------------------------------'


! DATOS DEL SISTEMA
PRINT '(2/,10X,A,/)', '*  ELEMENTOS DEL SISTEMA  *'
PRINT '(/,10X,A,\)', ' CUANTAS ECUACIONES TIENE EL SISTEMA:      '
READ*, N
PRINT '(/,10X,A,\)', ' CUANTAS INCOGNITAS "X" TIENE EL SISTEMA:  '
READ*, O

!  CONDICION
IF (N/=O) THEN
      PRINT '(2/,5X,A)', '** EL NUMERO DE ECUACIONES DEBE SER IGUAL AL
NUMERO DE INCOGNITAS'
      PRINT '(8X,A)', 'PARA OBTENER UNA SOLUCION UNICA Y REAL **'
      GOTO 2000
END IF

! ACTIVAR LAS MATRICES
ALLOCATE (A(N,N), B(N,1))
A=0
B=0

!INTRODUCIR DATOS DE LA MATRIZ [A]
     PRINT '(4/,10X,A,/)', '*  ELEMENTOS DEL SISTEMA DE LA MATRIZ [A]  *'
DO I=1,N
      DO J=1,N
            PRINT 200, ' INTRODUCE EL ELEMENTO (',I,',',J,')= '
            READ*, A(I,J)
      END DO
END DO

! INTRODUCIR DATOS DEL VECTOR DE TERMINOS INDEPENDIENTES {B}
PRINT '(2/,10X,A,/)', '*  ELEMENTOS DEL VECTOR {B}  *'
DO I=1,N
```

```
      J=1
      PRINT 300, ' INTRODUCE EL ELEMENTO (',I,',',J,')= '
      READ*, B(I,J)
END DO

! IMPRESION MATRIZ A
PRINT '(4/,10X,A)', '**  FORMA MATRICIAL  **'
PRINT '(3/,10X,A,I3,A,I3,A,2/)', ' LA MATRIZ [A] DE ORDEN',  N, ' X', N,'
ES: '
DO I=1, N
  DO J=1, N
      PRINT '(3X,F10.5,\)', A(I,J)
  END DO
  PRINT*
END DO

! IMPRESION VECTOR B
PRINT '(3/,10X,A,I3,A,2/)', ' EL VECTOR [B] DE ORDEN', N,' X 1 ES: '
DO I=1, N
   J=1
      PRINT '(13X,F10.5,\)', B(I,J)
      PRINT*
END DO

!OPERACIONES
DO P=1,N
   M= A(P,P)
            DO C=1,N
                  A(P,C)= (A(P,C))/M
            END DO
                  B(P,1)= (B(P,1))/M
            DO R=1,N
                  IF (R==P) CYCLE
                        M= A(R,P)
                        DO C=1,N
                              A(R,C)= A(R,C)-M*A(P,C)
                        END DO
                              B(R,1)= B(R,1)-M*B(P,1)
                  END DO
      END DO
!IMPRESION DE LAS INCOGNITAS X
PRINT 500, '>> SOLUCIONES <<'
PRINT 600, 'EL VECTOR DE INCOGNITAS {X} DE ORDEN',  N, ' X 1 ES: '
DO I=1, N
      J=1
      PRINT 700, B(I,J)
      PRINT*
END DO
```

```
!! DESACTIVAR MATRICES
DEALLOCATE (A, B)

! METODO DE LA INVERSA !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PRINT '(2/,A)', '-------------------------------------------------------'
ELSE IF (OP=='2') THEN
      PRINT '(3/,25X,A,3/)', '>>  METODO DE LA INVERSA  <<'

3000 PRINT*, '-------------------------------------------------------'


! DATOS DEL SISTEMA
PRINT '(2/,10X,A,/)', '*  ELEMENTOS DEL SISTEMA  *'
PRINT '(/,10X,A,\)', ' CUANTAS ECUACIONES TIENE EL SISTEMA:      '
READ*, N
PRINT '(/,10X,A,\)', ' CUANTAS INCOGNITAS "X" TIENE EL SISTEMA:  '
READ*, O

! CONDICION
IF (N/=O) THEN
      PRINT '(2/,5X,A)', '** EL NUMERO DE ECUACIONES DEBE SER IGUAL AL
NUMERO DE INCOGNITAS'
      PRINT '(8X,A)', 'PARA OBTENER UNA SOLUCION UNICA Y REAL **'
      GOTO 3000
END IF

! ACTIVAR LAS MATRICES
ALLOCATE (A(N,N), MI(N,N), B(N,1))
A=0
MI=0
B=0

!INTRODUCIR DATOS DE LA MATRIZ [A]
PRINT '(4/,10X,A,/)', '*  ELEMENTOS DEL SISTEMA  *'
DO I=1,N
      DO J=1,N
            PRINT 200, ' INTRODUCE EL ELEMENTO (',I,',',J,'): '
            READ*, A(I,J)
      END DO
END DO
! INTRODUCIR DATOS DEL VECTOR DE TERMINOS INDEPENDIENTES {B}
PRINT '(2/,10X,A,/)', '*  ELEMENTOS DEL VECTOR {B}  *'
DO I=1,N
   J=1
      PRINT 300, ' INTRODUCE EL ELEMENTO (',I,',',J,')= '
      READ*, B(I,J)
END DO

! IMPRESION DE LA MATRIZ A
```

```
PRINT '(4/,10X,A)', '**  FORMA MATRICIAL  **'
PRINT '(3/,10X,A,I3,A,I3,A,2/)', ' LA MATRIZ [A] DE ORDEN',  N, ' X', N,'
ES: '
DO I=1, N
      DO J=1, N
            PRINT '(3X,F10.5,\)', A(I,J)
      END DO
    PRINT*
END DO

! IMPRESION VECTOR B
PRINT '(3/,10X,A,I3,A,2/)', ' EL VECTOR [B] DE ORDEN', N,' X 1 ES: '
DO I=1, N
   J=1
      PRINT '(13X,F10.5,\)', B(I,J)
      PRINT*
END DO

! MATRIZ IDENTIDAD
DO I=1, N
      DO J=1, N
            MI(I,J)=0
            MI(I,I) = 1
      END DO
            PRINT*
END DO

!OPERACIONES
DO P=1,N
      M= A(P,P)
            DO C=1,N
                  A(P,C)= (A(P,C))/M
                  MI(P,C)= (MI(P,C))/M
            END DO
                  B(P,1)= (B(P,1))/M
            DO R=1,N
                  IF (R==P) CYCLE
                        M= A(R,P)
                        DO C=1,N
                              A(R,C)= A(R,C)-M*A(P,C)
                              MI(R,C)= MI(R,C)-M*MI(P,C)
                        END DO
                              B(R,1)= B(R,1)-M*B(P,1)
            END DO
END DO

! IMPRESION DE LAS INCOGNITAS X
PRINT 500, '>>>  SOLUCIONES <<<'
PRINT 600, 'EL VECTOR DE INCOGNITAS {X} DE ORDEN',  N, ' X 1 ES: '
```

```
DO I=1, N
   J=1
      PRINT 700, B(I,J)
      PRINT*
END DO
!! DESACTIVAR MATRICES
DEALLOCATE (A, MI, B)

! METODO DE CHOLESKY !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PRINT '(2/,A)', '-------------------------------------------------------'
ELSE IF (OP=='3') THEN
      PRINT '(3/,25X,A,3/)', '>>  METODO DE CHOLESKY  <<'

4000 PRINT '(/,A)', '-------------------------------------------------'

!DATOS DEL SISTEMA
PRINT '(2/,10X,A,/)', '*  ELEMENTOS DEL SISTEMA  *'
PRINT '(/,10X,A,\)', ' CUANTAS ECUACIONES TIENE EL SISTEMA:  '
READ*, N
PRINT '(/,10X,A,\)', ' CUANTAS INCOGNITAS "X" TIENE EL SISTEMA:  '
READ*, O

!CONDICION
IF (N/=O) THEN
      PRINT '(2/,5X,A)', '** EL NUMERO DE ECUACIONES DEBE SER IGUAL AL
NUMERO DE INCOGNITAS'
      PRINT '(8X,A)', 'PARA OBTENER UNA SOLUCION UNICA Y REAL **'
      GOTO 4000
END IF

!ACTIVAR MATRICES
ALLOCATE (A(N,N+1),L(N,N+1),U(N,N+1),X(N,1),B(N,1))

!INTRODUCIR DATOS DE LA MATRIZ [A]
PRINT '(3/,10X,A,/)', '*  ELEMENTOS DEL SISTEMA DE LA MATRIZ [A]  *'
DO I=1,N
      DO J=1,N
      PRINT 200, ' INTRODUCE EL ELEMENTO (',I,',',J,'): '
      READ*, A(I,J)
      END DO
END DO

!INTRODUCIR DATOS DEL VECTOR DE TERMINOS INDEPENDIENTES {B}
PRINT '(3/,10X,A,/)', '*  ELEMENTOS DEL VECTOR {B}  *'
DO I=1,N
   J=1
      PRINT 300, ' INTRODUCE EL ELEMENTO (',I,',',J,'): '
      READ*, B(I,J)
END DO
```

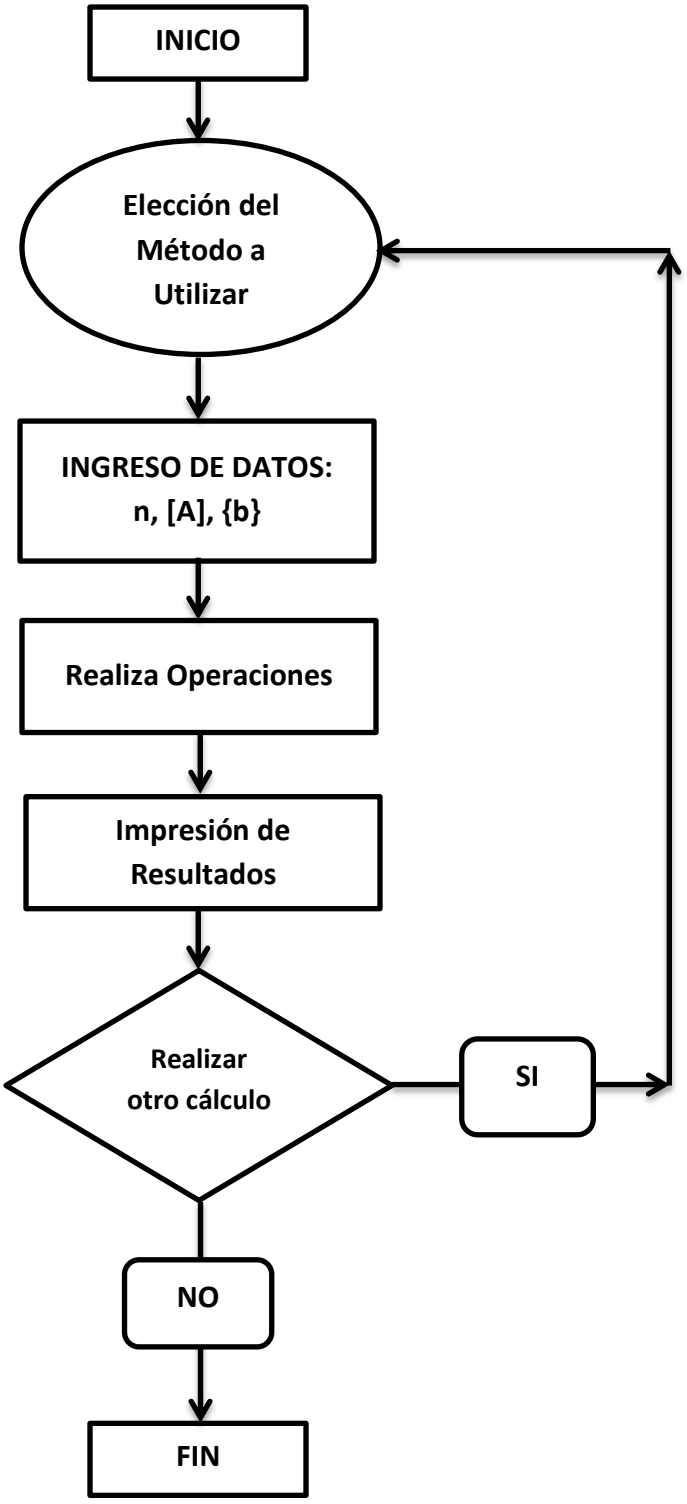
```
! PASAR VECTOR B A MATRIZ A
DO I=1, N
   J=1
      A(I,O+1)=B(I,J)
END DO
!IMPRESION MATRIZ A
PRINT '(3/,10X,A)', '**  FORMA MATRICIAL  **'
PRINT '(2/,10X,A,I3,A,I3,A,2/)', ' LA MATRIZ [A] DE ORDEN',  N, ' X', N,'
ES: '
DO I=1, O
      DO J=1, O
            PRINT '(3X,F10.5,\)', A(I,J)
      END DO
            PRINT*
END DO

!IMPRESION VECTOR B
PRINT '(2/,10X,A,I3,A,2/)', ' EL VECTOR [B] DE ORDEN', N,' X 1 ES: '
DO I=1, N
   J=1
            PRINT '(13X,F10.5,\)', B(I,J)
            PRINT*
END DO

! OPERACIONES
DO J=1,N
      DO I=J,N
            L(I,J)=0
            DO R=1,J-1
                  L(I,J)=L(I,J)-L(I,R)*U(R,J)
            END DO
            L(I,J)=L(I,J)+A(I,J)
      END DO
      DO I=J,N+1
            U(J,I)=0
            DO R=1,J-1
                  U(J,I)=U(J,I)-L(J,R)*U(R,I)
            END DO
            U(J,I)= (A(J,I)+U(J,I))/L(J,J)
      END DO
END DO

DO I=N,1,-1
      X(I,1)=0
      DO R=I+1,N
            X(I,1)=X(I,1)-U(I,R)*X(R,1)
      END DO
            X(I,1)= U(I,N+1)+X(I,1)
```

```
END DO

!IMPRESION DE LAS INCOGNITAS X
PRINT  500,'>>> SOLUCIONES <<<'
PRINT  600, ' EL VECTOR DE INCOGNITAS {X} DE ORDEN',  N, ' X 1 ES: '
DO I=1, N
   J=1
            PRINT 700, X(I,J)
            PRINT*
END DO

!DESACTIVAR MATRICES
DEALLOCATE (A,L,U,X,B)


! METODO DE GAUSS-SEIDEL !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PRINT '(2/,A)', '-------------------------------------------------------'
ELSE IF (OP=='4') THEN
      PRINT '(3/,25X,A,3/)', '>>  METODO DE GAUSS-SEIDEL  <<'

5000 PRINT '(/,A)', '-------------------------------------------------'


! DATOS DEL SISTEMA
PRINT '(2/,10X,A,/)', '*  ELEMENTOS DEL SISTEMA  *'
PRINT '(/,10X,A,\)', 'CUANTAS ECUACIONES TIENE EL SISTEMA:      '
READ*, N
PRINT '(/,10X,A,\)', 'CUANTAS INCOGNITAS "X" TIENE EL SISTEMA:  '
READ*, O

!CONDICION
IF (N/=O) THEN
      PRINT '(2/,5X,A)', '** EL NUMERO DE ECUACIONES DEBE SER IGUAL AL
NUMERO DE INCOGNITAS'
      PRINT '(8X,A)', 'PARA OBTENER UNA SOLUCION UNICA Y REAL **'
      GOTO 5000
END IF

ALLOCATE (A(N,N), B(N,1), X(N,1), XO(N,1))
A=0
B=0
X=0
XO=0

!INTRODUCIR DATOS DE LA MATRIZ [A]
PRINT '(2/,10X,A,/)', '*  ELEMENTOS DEL SISTEMA DE LA MATRIZ [A]  *'
DO I=1,N
      DO J=1,N
            PRINT 200, ' INTRODUCE EL ELEMENTO (',I,',',J,')= '
```

```
            READ*, A(I,J)
      END DO
END DO

! INTRODUCIR DATOS DEL VECTOR DE TERMINOS INDEPENDIENTES {B}
PRINT '(4/,10X,A,/)', '*  ELEMENTOS DEL VECTOR {B}  *'
DO I=1,N
J=1
      PRINT 300, ' INTRODUCE EL ELEMENTO (',I,',',J,')= '
      READ*, B(I,J)
END DO

! IMPRESION MATRIZ A
PRINT '(4/,10X,A)', '**  FORMA MATRICIAL  **'
PRINT '(3/,10X,A,I3,A,I3,A,2/)', 'LA MATRIZ [A] DE ORDEN',  N, ' X', N,'
ES: '
DO I=1, N
  DO J=1, N
      PRINT '(3X,F10.5,\)', A(I,J)
  END DO
  PRINT*
END DO

! IMPRESION VECTOR B
PRINT '(3/,10X,A,I3,A,2/)', 'EL VECTOR [B] DE ORDEN', N,' X 1 ES: '
DO I=1, N
    J=1
      PRINT '(13X,F10.5,\)', B(I,J)
      PRINT*
END DO

! INTRODUCIR DATOS DEL VECTOR X, LA ESTIMA DE LAS SOLUCIONES
PRINT '(2/,10X,A,/)', '*  ELEMENTOS DEL VECTOR {X}  *'
DO I=1,N
      J=1
      PRINT 300, 'INTRODUCE EL ELEMENTO (',I,',',J,')= '
      READ*, X(I,J)
END DO

PRINT '(3/,3X,A,2/)', '   -ITERACION - TOLERANCIA -                    -
SOLUCIONES -   '


!OPERACIONES
TOL=0.0
DO ITER=1,2000,1
      DO I=1,N,1
            XO(I,1)=X(I,1)
            X(I,1) =B(I,1)
```

```
            DO J=1,N,1
                  IF(J==I) CYCLE
                  X(I,1)=X(I,1)-(A(I,J)*X(J,1))
            END DO
            X(I,1)=X(I,1)/A(I,I)
            TOL=TOL+ABS(XO(I,1)-X(I,1))
      END DO

      PRINT*, ITER, TOL, (X(I,1),I=1,N)
      IF (TOL<=1.0E-5) EXIT
      TOL   =0.0
END DO

!! IMPRESION DE RESULTADOS
PRINT '(2/,10X,A,I4,)', ' TOTAL DE ITERACIONES: ', ITER
PRINT '(10X,A,8X,F10.5)', ' TOLERANCIA: ', TOL

PRINT  500,'>>> SOLUCIONES <<<'
PRINT  600, ' EL VECTOR DE INCOGNITAS {X} DE ORDEN',  N, ' X 1 ES: '
DO I=1, N
      PRINT '(13X,F10.5,\)', X(I,1)
    PRINT*
END DO

DEALLOCATE (A, B, X, XO)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
PRINT '(3/,A)', '-------------------------------------------------------'

END IF

PRINT '(3/,5X,A,X,A)', 'DESEAS INGRESAR OTRA MATRIZ??','(SI/NO)'
READ*, OTRA
      IF (OTRA=='SI') GOTO 1000

END PROGRAM
```

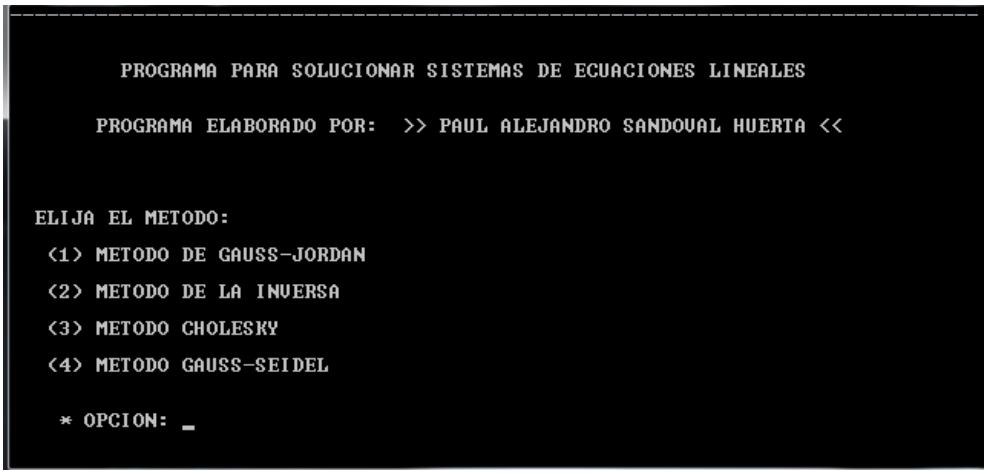
3.3 DIAGRAMA DE BLOQUES



3.4 MANUAL DE USUARIO

El presente manual para el programa Solución de Sistemas Lineales diseñado en FORTRAN 90, se ha pensado con la finalidad de explicar de forma detallada todos los pasos que el usuario debe seguir para su utilización.

- Hace doble clic sobre el icono con el nombre “SOLECLIN.exe”. Inmediatamente se abrirá una ventana de tipo DOS con la siguiente apariencia.

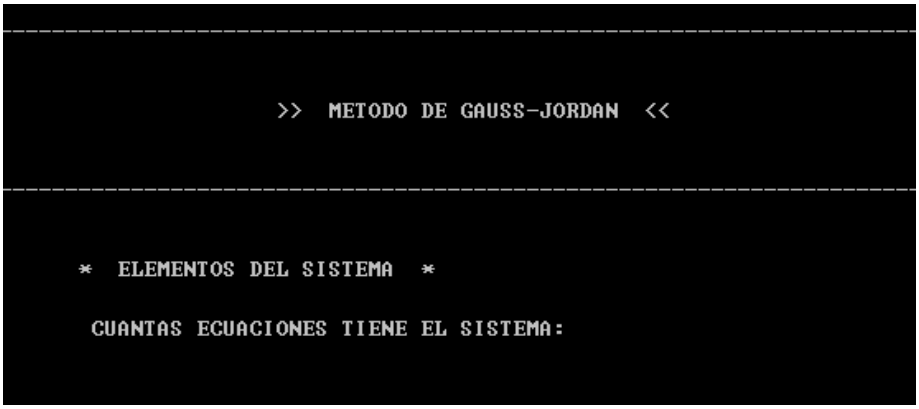


En la ventana de tipo DOS podemos ver los diferentes métodos que se pueden utilizar, además de que la ubicación del cursor será un guion bajo intermitente, para acceder a uno de ellos solo se debe de teclear el número que lo precede.

Como ejemplo se usará el método de Gauss-Jordan (la forma de introducir los valores es idéntica para los demás métodos). Se introduce el número “1” desde el teclado referenciado para el “Método de Gauss-Jordan” y se presionará la tecla ENTER, en cada paso donde se introduzca un dato será necesario presionar la tecla ENTER.

- A continuación aparecerá el nombre del método, seguido de los siguientes enunciados “ELEMENTOS DEL SISTEMA “ y “ CUANTAS ECUACIONES TIENE

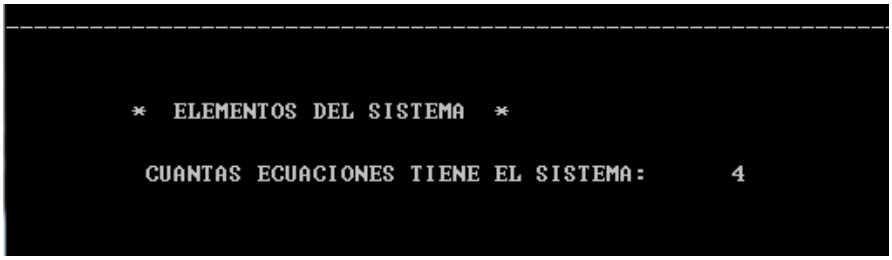
EL SISTEMA" se debe de ingresar con números enteros la cantidad de ecuaciones que tenga el sistema que se desea solucionar.



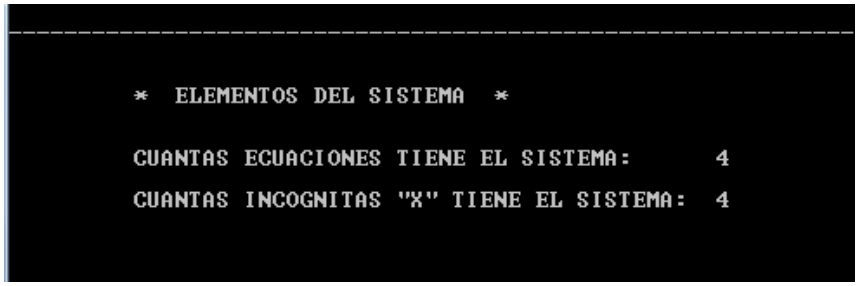
Para comparar la efectividad del programa el ejemplo será el usado en cada uno de los métodos vistos en el CAPÍTULO 1.

$$A\,x = C = \underset{(\ 4\,x\,4\ )}{\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}} \underset{(\ 4\,x\,1\ )}{\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

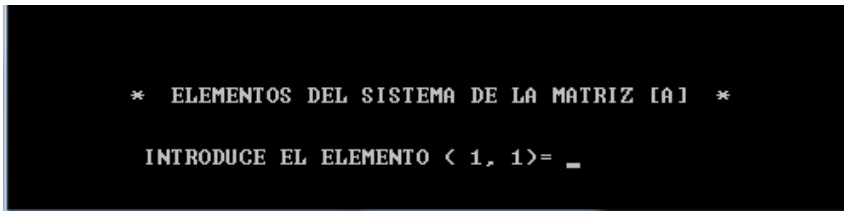
- Nuestro sistema cuenta con 4 ecuaciones y 4 incógnitas. Así que se teclea el número 4 y la tecla ENTER.



- Después aparece el enunciado "CUANTAS INCOGNITAS 'X' TIENE EL SISTEMA" y de la misma forma se ingresará el número 4 y la tecla ENTER, dado que una de las reglas para la solución de sistemas de ecuaciones lineales es que el número de incógnitas sea igual al número de ecuaciones.

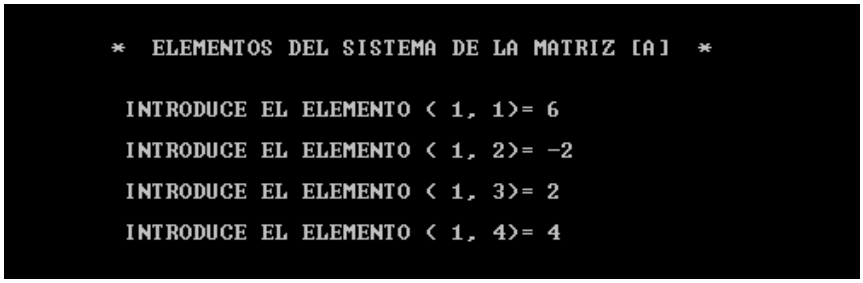


- Aparecerá el enunciado “ELEMENTOS DEL SISTEMA DE LA MATRIZ [A]” seguido del enunciado “INTRODUCE EL ELEMENTO (1, 1)=” .



El segundo enunciado indica la posición del elemento por fila y columna, el cual se ira modificando al introducir cada número, el orden para introducir los elementos es por fila con respecto a la matriz. Si la cantidad es negativa se añadirá al inicio de la cifra un guion intermedio (-) y si contiene decimales solo se deberá de agregar un punto (.) en la posición que le corresponda, una vez introducida la cantidad se presiona la tecla ENTER. Siguiendo el mismo procedimiento para cada elemento de cada fila.

$$\begin{matrix} F1 \rightarrow \\ F2 \rightarrow \\ F3 \rightarrow \\ F4 \rightarrow \end{matrix} \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}$$



- Cuando se han indicado todos los elementos de la matriz de coeficientes, el programa pedirá que se introduzcan los elementos del vector de términos independientes. El procedimiento es idéntico al anterior.

```
*  ELEMENTOS DEL VECTOR (B)  *

 INTRODUCE EL ELEMENTO ( 1, 1)=
```

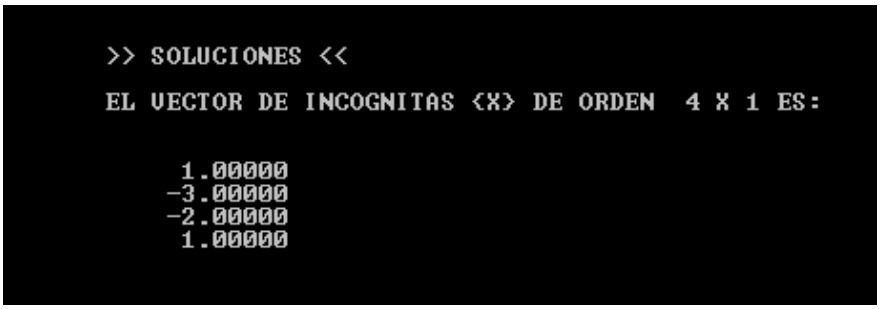
$$\begin{matrix} \text{F1} \rightarrow \\ \text{F2} \rightarrow \\ \text{F3} \rightarrow \\ \text{F4} \rightarrow \end{matrix} \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

```
*  ELEMENTOS DEL VECTOR (B)  *

 INTRODUCE EL ELEMENTO ( 1, 1)= 12
 INTRODUCE EL ELEMENTO ( 2, 1)= 34
 INTRODUCE EL ELEMENTO ( 3, 1)= 27
 INTRODUCE EL ELEMENTO ( 4, 1)= -38
```

- Automáticamente el programa ordena los datos en forma matricial separándolos por la Matriz [A] y el Vector {B} e indicando sus respectivos tamaños.

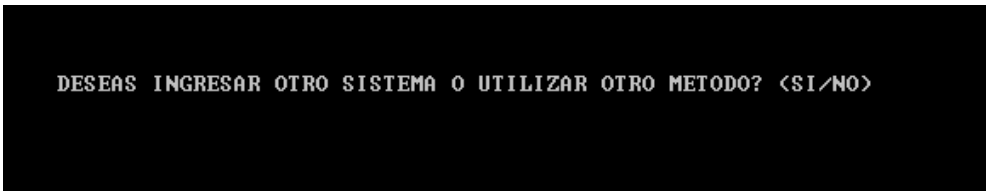
```
**  FORMA MATRICIAL  **

  LA MATRIZ [A] DE ORDEN  4 X  4 ES:

  6.00000     -2.00000      2.00000      4.00000
 12.00000     -8.00000      6.00000     10.00000
  3.00000    -13.00000      9.00000      3.00000
 -6.00000      4.00000      1.00000    -18.00000

  EL VECTOR [B] DE ORDEN  4 X 1 ES:

      12.00000
      34.00000
      27.00000
     -38.00000
```

- Y dando solución al sistema, aparece el Vector de Incógnitas el cual en su enunciado se indica su tamaño.



- NOTA: Si en alguno de los pasos anteriores, durante la introducción de los datos, se comete el error de ingresar caracteres distintos a; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, ., el programa se cerrara de inmediato y se tendrá que volver a comenzar.

- Por último, si se desea utilizar otro método o conocer la solución de otro sistema, sólo se debe de teclear la palabra “SI” con letras mayúsculas y presionar ENTER, y en caso de querer salir del programa se tecle la palabra “NO” en mayúsculas y se presiona ENTER, esto hará que el programa se cierre.



CAPÍTULO 4.- EJEMPLOS DE APLICACIÓN

En este capítulo se mostrará de manera detallada el proceso para la solución de un sistema de ecuaciones utilizando los diferentes métodos mencionados y explicados en el capítulo 1.

4.1 MÉTODO DE GAUSS-JORDAN

Aplicando el método de *Gauss-Jordan* se buscarán los valores del vector de incógnitas que den solución al siguiente sistema de ecuaciones lineales.

$$A\,x = C = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

Solución

$$\begin{matrix} \text{F1} \rightarrow \\ \text{F2} \rightarrow \\ \text{F3} \rightarrow \\ \text{F4} \rightarrow \end{matrix} \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

Fila pivote: $i = 1$; elemento pivote: $a_{11} = 6$

$F_1 \leftarrow (1/6) * F_1$

$$\begin{bmatrix} 1 & -1/3 & 1/3 & 2/3 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

Fila pivote: $i = 1$; elemento pivote: $a_{11} = 1$

$F_2 \leftarrow F_2 - (12 * F_1)$

$F_3 \leftarrow F_3 - (3 * F_1)$

$F_4 \leftarrow F_4 - (-6 * F_1)$

$$\begin{bmatrix} 1 & -1/3 & 1/3 & 2/3 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 10 \\ 21 \\ -26 \end{bmatrix}$$

Fila pivote: i = 2; elemento pivote: a_{22} = -4

$F_2 \leftarrow (1 / -4) * F_2$

$$\begin{bmatrix} 1 & -1/3 & 1/3 & 2/3 \\ 0 & 1 & -1/2 & -1/2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 2 \\ -5/2 \\ 21 \\ -26 \end{bmatrix}$$

Fila pivote: i = 2; elemento pivote: a_{22}= 1

$F_1 \leftarrow F_1 - (-1/3 * F_2)$

$F_3 \leftarrow F_3 - (-12 * F_2)$

$F_4 \leftarrow F_4 - (2 * F_2)$

$$\begin{bmatrix} 1 & 0 & 1/6 & 1/2 \\ 0 & 1 & -1/2 & -1/2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 7/6 \\ -5/2 \\ -9 \\ -21 \end{bmatrix}$$

Fila pivote: i = 3; elemento pivote: a_{33} =2

$F_3 \leftarrow (1/2) * F_3$

$$\begin{bmatrix} 1 & 0 & 1/6 & 1/2 \\ 0 & 1 & -1/2 & -1/2 \\ 0 & 0 & 1 & -5/2 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 7/6 \\ -5/2 \\ -9/2 \\ -21 \end{bmatrix}$$

Fila pivote: i = 3; elemento pivote: a_{33}= 1

$F_1 \leftarrow F_1 - (1/6 * F_3)$

$F_2 \leftarrow F_2 - (-1/2 * F_3)$

$F_4 \leftarrow F_4 - (4 * F_3)$

$$\begin{bmatrix} 1 & 0 & 0 & 11/12 \\ 0 & 1 & 0 & -7/4 \\ 0 & 0 & 1 & -5/2 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 23/12 \\ -19/4 \\ -9/2 \\ -3 \end{bmatrix}$$

Fila pivote: i = 4; elemento pivote: a_{44}= -3

$F_4 \leftarrow (1/-3) = * F_4$

$$\begin{bmatrix} 1 & 0 & 0 & 11/12 \\ 0 & 1 & 0 & -7/4 \\ 0 & 0 & 1 & -5/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 23/12 \\ -19/4 \\ -9/2 \\ 1 \end{bmatrix}$$

Fila pivote: i = 4; elemento pivote: a_{44}= 1

$F_1 \leftarrow F_1 - (11/12 * F_4)$

$F_2 \leftarrow F_2 - (-7/4 * F_4)$

$F_3 \leftarrow F_3 - (-5/2 * F_4)$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \\ -2 \\ 1 \end{bmatrix}$$

El vector solución del sistema es:

$$X = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \\ -2 \\ 1 \end{bmatrix}$$

4.2 MÉTODO DE LA MATRIZ INVERSA

Aplicando el método de *la Matriz Inversa* se buscará resolver el siguiente sistema de ecuaciones lineales, encontrando la matriz inversa de la matriz A y multiplicando por el vector de términos independientes.

$$A\,x = C = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

Solución

Se aplica el método de la matriz inversa al sistema de ecuaciones.

$$\begin{matrix} F1 \rightarrow \\ F2 \rightarrow \\ F3 \rightarrow \\ F4 \rightarrow \end{matrix} \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fila pivote: $i = 1$; elemento pivote: $a_{11} = 6$

$F_1 \leftarrow (1/6) * F_1$

$$\begin{bmatrix} 1 & -1/3 & 1/3 & 2/3 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 1/6 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fila pivote: $i = 1$; elemento pivote: $a_{11} = 1$

$F_2 \leftarrow F_2 - (12 * F_1)$

$F_3 \leftarrow F_3 - (3 * F_1)$

$F_4 \leftarrow F_4 - (-6 * F_1)$

$$\begin{bmatrix} 1 & -1/3 & 1/3 & 2/3 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 1/6 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -1/2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Fila pivote: $i = 2$; elemento pivote: $a_{22} = -4$

$F_2 \leftarrow (1/-4) * F_2$

$$\begin{bmatrix} 1 & -1/3 & 1/3 & 2/3 \\ 0 & 1 & -1/2 & -1/2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 1/6 & 0 & 0 & 0 \\ 1/2 & -1/4 & 0 & 0 \\ -1/2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Fila pivote: i = 2; elemento pivote: a_{22}= 1

$F_1 \leftarrow F_1 - (-1/3 * F_2)$

$F_3 \leftarrow F_3 - (-12 * F_2)$

$F_4 \leftarrow F_4 - (2 * F_2)$

$$\begin{bmatrix} 1 & 0 & 1/6 & 1/2 \\ 0 & 1 & -1/2 & -1/2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 1/3 & -1/12 & 0 & 0 \\ 1/2 & -1/4 & 0 & 0 \\ 11/2 & -3 & 1 & 0 \\ 0 & 1/2 & 0 & 1 \end{bmatrix}$$

Fila pivote: i = 3; elemento pivote: a_{33} = 2

$F_3 \leftarrow (1/2) * F_3$

$$\begin{bmatrix} 1 & 0 & 1/6 & 1/2 \\ 0 & 1 & -1/2 & -1/2 \\ 0 & 0 & 1 & -5/2 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 1/3 & -1/12 & 0 & 0 \\ 1/2 & -1/4 & 0 & 0 \\ 11/4 & -3/2 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1 \end{bmatrix}$$

Fila pivote: i = 3; elemento pivote: a_{33}= 1

$F_1 \leftarrow F_1 - (1/6 * F_3)$

$F_2 \leftarrow F_2 - (-1/2 * F_3)$

$F_4 \leftarrow F_4 - (4 * F_3)$

$$\begin{bmatrix} 1 & 0 & 0 & 11/12 \\ 0 & 1 & 0 & -7/4 \\ 0 & 0 & 1 & -5/2 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} -1/8 & 1/6 & -1/12 & 0 \\ 15/8 & -1 & 1/4 & 0 \\ 11/4 & -3/2 & 1/2 & 0 \\ -11 & 13/2 & -2 & 1 \end{bmatrix}$$

Fila pivote: i = 4; elemento pivote: a_{44}= -3

$F_4 \leftarrow (1/-3) * F_4$

$$\begin{bmatrix} 1 & 0 & 0 & 11/12 \\ 0 & 1 & 0 & -7/4 \\ 0 & 0 & 1 & -5/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} -1/8 & 1/6 & -1/12 & 0 \\ 15/8 & -1 & 1/4 & 0 \\ 11/4 & -3/2 & 1/2 & 0 \\ 11/3 & -13/6 & 2/3 & -1/3 \end{bmatrix}$$

Fila pivote: i = 4; elemento pivote: a_{44}= 1

$F_1 \leftarrow F_1 - (11/12 * F_4)$

$F_2 \leftarrow F_2 - (-7/4 * F_4)$

$F_3 \leftarrow F_3 - (-5/2 * F_4)$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} -251/72 & 155/72 & -25/36 & 11/36 \\ 199/24 & -115/24 & 17/12 & -7/12 \\ 143/12 & -83/12 & 13/6 & -5/6 \\ 11/3 & -13/6 & 2/3 & -1/3 \end{bmatrix}$$

Por lo tanto, la matriz inversa es:

$$A^{-1} = \begin{bmatrix} -251/72 & 155/72 & -25/36 & 11/36 \\ 199/24 & -115/24 & 17/12 & -7/12 \\ 143/12 & -83/12 & 13/6 & -5/6 \\ 11/3 & -13/6 & 2/3 & -1/3 \end{bmatrix}$$

El vector solución es la multiplicación de la matriz inversa A^{-1} y el vector de términos independientes:

$$X = \begin{bmatrix} -251/72 & 155/72 & -25/36 & 11/36 \\ 199/24 & -115/24 & 17/12 & -7/12 \\ 143/12 & -83/12 & 13/6 & -5/6 \\ 11/3 & -13/6 & 2/3 & -1/3 \end{bmatrix} \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

$$X = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \begin{bmatrix} -\frac{251}{6} & +\frac{2635}{36} & -\frac{75}{4} & -\frac{209}{18} \\ \frac{199}{2} & -\frac{1955}{12} & +\frac{153}{4} & +\frac{133}{6} \\ \frac{143}{1} & -\frac{1411}{6} & +\frac{117}{2} & +\frac{95}{3} \\ \frac{44}{1} & -\frac{221}{3} & +\frac{18}{1} & +\frac{38}{3} \end{bmatrix}$$

$$X = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \\ -2 \\ 1 \end{bmatrix}$$

4.3 MÉTODO DE CHOLESKY

Aplicando el método de *Cholesky* se buscarán los valores del vector de incógnitas que den solución al siguiente sistema de ecuaciones lineales.

$$A\,x = C = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

Solución

El sistema de ecuaciones lineales se resuelve con el método de descomposición LU, mediante los sistemas:

$$Ax = C;\ (LU)\,x = C;\ L\,(Ux) = C$$

El sistema $Ux = y$ está formado por las incógnitas auxiliares y.
El sistema $Ly = \mathrm{C}$ representa un sistema triangular en y.

Se descompone la matriz A en las matrices triangular inferior (L) y superior (U), según el método de Cholesky:

$$A = L\,U$$

$$A = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}$$

$$L = \begin{bmatrix} l11 & 0 & 0 & 0 \\ l21 & l22 & 0 & 0 \\ l31 & l32 & l33 & 0 \\ l41 & l42 & l43 & l44 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & u12 & u13 & u14 \\ 0 & 1 & u23 & u24 \\ 0 & 0 & 1 & u34 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se tiene que cumplir que $LU = A$. Operando matricialmente, se tiene:

$$\begin{bmatrix} l11 & 0 & 0 & 0 \\ l21 & l22 & 0 & 0 \\ l31 & l32 & l33 & 0 \\ l41 & l42 & l43 & l44 \end{bmatrix} \begin{bmatrix} 1 & u12 & u13 & u14 \\ 0 & 1 & u23 & u24 \\ 0 & 0 & 1 & u34 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}$$

$$\begin{bmatrix} l11 & l11u12 & l11u13 & l11u14 \\ l21 & l21u12 + l22 & l21u13 + l22u23 & l21u14 + l22u24 \\ l31 & l31u12 + l32 & l31u13 + l32u23 + l33 & l31u14 + l32u24 + l33u34 \\ l41 & l41u12 + l42 & l41u13 + l42u23 + l43 & l41u14 + l42u24 + l43u34 + l44 \end{bmatrix}$$

$$= \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}$$

Fila n°1 de la matriz de coeficientes:

$$\begin{bmatrix} 6 \\ -2 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} l11 \\ l11u12 \\ l11u13 \\ l11u14 \end{bmatrix}$$

Fila n°2 de la matriz de coeficientes:

$$\begin{bmatrix} 12 \\ -8 \\ 6 \\ 10 \end{bmatrix} = \begin{bmatrix} l21 \\ l21u12 + l22 \\ l21u13 + l22u23 \\ l21u14 + l22u24 \end{bmatrix}$$

Fila n°3 de la matriz de coeficientes:

$$\begin{bmatrix} 3 \\ -13 \\ 9 \\ 3 \end{bmatrix} = \begin{bmatrix} l31 \\ l31u12 + l32 \\ l31u13 + l32u23 + l33 \\ l31u14 + l32u24 + l33u34 \end{bmatrix}$$

Fila n°4 de la matriz de coeficientes:

$$\begin{bmatrix} -6 \\ 4 \\ 1 \\ -18 \end{bmatrix} = \begin{bmatrix} l41 \\ l41u12 + l42 \\ l41u13 + l42u23 + l43 \\ l41u14 + l42u24 + l43u34 + l44 \end{bmatrix}$$

Sustituyendo los valores se obtienen los coeficientes L y U:

6 = $l11$				$\rightarrow$	$l11$ = 6
-2 = $l11u12$	$\rightarrow$	-2 = $(6)u12$	$\rightarrow$	$u12$ = -2/6	
2 = $l11u13$	$\rightarrow$	2 = $(6)u13$	$\rightarrow$	$u13$ = 2/6	
4 = $l11u14$	$\rightarrow$	4 = $(6)u14$	$\rightarrow$	$u14$ = 4/6	

$12 = l21$ $\rightarrow$ $l21 = 12$

$-8 = l21u12 + l22$ $\rightarrow$ $-8 = (12)\left(-\frac{2}{6}\right) + l22$ $\rightarrow$ $l22 = -4$

$6 = l21u13 + l22u23$ $\rightarrow$ $6 = (12)\left(\frac{2}{6}\right) + (4)u23$ $\rightarrow$ $u23 = -1/2$

$10 = l21u14 + l22u24$ $\rightarrow$ $10 = (12)\left(\frac{4}{6}\right) + (-4)u24$ $\rightarrow$ $u24 = -1/2$

$3 = l31$ $\rightarrow$ $l31 = 3$

$-13 = l31u12 + l32$ $\rightarrow$ $-13 = (3)\left(-\frac{2}{6}\right) + l32$ $\rightarrow$ $l32 = -12$

$9 = l31u13 + l32u23 + l33$

$\rightarrow$ $9 = (3)\left(\frac{2}{6}\right) + (-12)\left(-\frac{1}{2}\right) + l33$ $\rightarrow$ $l33 = 2$

$3 = l31u14 + l32u24 + l33u34$

$\rightarrow$ $3 = (3)\left(\frac{4}{6}\right) + (-12)\left(-\frac{1}{2}\right) + (2)u34$ $\rightarrow$ $u24 = -5/2$

$-6 = l41$ $\rightarrow$ $l41 = -6$

$4 = l41u12 + l42$ $\rightarrow$ $4 = (-6)\left(-\frac{2}{6}\right) + l42$ $\rightarrow$ $l42 = 2$

$1 = l41u13 + l42u23 + l43$

$\rightarrow$ $1 = (-6)\left(\frac{2}{6}\right) + (2)\left(-\frac{1}{2}\right) + l43$ $\rightarrow$ $l43 = 4$

$-18 = l41u14 + l42u24 + l43u34 + l44$

$\rightarrow$ $-18 = (-6)\left(\frac{4}{6}\right) + (2)\left(-\frac{1}{2}\right) + (4)\left(-\frac{5}{2}\right) + l44$ $\rightarrow$ $l44 = -3$

$$L = \begin{bmatrix} 6 & 0 & 0 & 0 \\ 12 & -4 & 0 & 0 \\ 3 & -12 & 2 & 0 \\ -6 & 2 & 4 & -3 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & -\frac{2}{6} & \frac{2}{6} & \frac{4}{6} \\ 0 & 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 1 & -\frac{5}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se resuelve el sistema $Ly = C$

$$\begin{bmatrix} 6 & 0 & 0 & 0 \\ 12 & -4 & 0 & 0 \\ 3 & -12 & 2 & 0 \\ -6 & 2 & 4 & -3 \end{bmatrix} \begin{bmatrix} y1 \\ y2 \\ y3 \\ y4 \end{bmatrix} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

$6y1 = 12 \rightarrow\ y1 = 12/\ 6 = 2$

$12y1 - 4y2 = 34 \rightarrow 12(2) - 4y2 = 34 \rightarrow\ y2 = (\ 34 - 28)/\ -4 = -5/2$

$3y1 - 12y2 + 2y3 = 27 \rightarrow 3(2) - 12(-\frac{5}{2}) + 2y3 = 27 \rightarrow y3 = (27 - 6 - 30)/\ 2 = -9/2$

$-6y1 + 2y2 + 4y3 - 3y4 = -38 \rightarrow -6(2) + 2(-\frac{5}{2}) + 4(-\frac{9}{2}) - 3(-\frac{9}{2}) = -38$
$\rightarrow y4 = (-38 + 12 + 5 + 18)/\ -3 = 1$

$$\begin{aligned} y1 &= 2 \\ y2 &= -\frac{5}{2} \\ y3 &= -\frac{9}{2} \\ y4 &= 1 \end{aligned}$$

$$Ly = C = \begin{bmatrix} 6 & 0 & 0 & 0 \\ 12 & -4 & 0 & 0 \\ 3 & -12 & 2 & 0 \\ -6 & 2 & 4 & -3 \end{bmatrix} \begin{bmatrix} 2 \\ -\frac{5}{2} \\ -\frac{9}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

Se resuelve el sistema $Ux = y$

$$\begin{bmatrix} 1 & -\frac{2}{6} & \frac{2}{6} & \frac{4}{6} \\ 0 & 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 1 & -\frac{5}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \begin{bmatrix} 2 \\ -\frac{5}{2} \\ -\frac{9}{2} \\ 1 \end{bmatrix}$$

$1x4 = 1 \rightarrow\ x4 = 1$

$1x3 - \left(\frac{5}{2}\right)x4 = -\frac{9}{2} \quad \rightarrow 1x3 - \left(\frac{5}{2}\right)(\ 1) = -\frac{9}{2} \quad \rightarrow\ x3 = -\frac{9}{2} + \frac{5}{2} = -\ 2$

$1x2 - \frac{1}{2}x3 - \frac{1}{2}x4 = -\frac{1}{2} \rightarrow 1x2 - \frac{1}{2}(-2\ ) - \frac{1}{2}(\ 1) = -\frac{5}{2} \rightarrow\ x2 = -\frac{5}{2} - 1 + \frac{1}{2} = -3$

$1x1 - \frac{2}{6}x2 + \frac{2}{6}x3 + \frac{4}{6}x4 = 2 \rightarrow\ 1x1 - \frac{2}{6}(-3) + \frac{2}{6}(-2) + \frac{4}{6}(\ 1\ ) = 2$
$\rightarrow\ x1 = 2 - \ 1\ + \frac{4}{6} - \frac{4}{6} = 1$

$$U\,x = \mathrm{y} = \begin{bmatrix} 1 & -\frac{2}{6} & \frac{2}{6} & \frac{4}{6} \\ 0 & 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 1 & -\frac{5}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -3 \\ -2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ -\frac{5}{2} \\ -\frac{9}{2} \\ 1 \end{bmatrix}$$

La solución del sistema de ecuaciones es el vector x:

$$x = \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \\ -2 \\ 1 \end{bmatrix}$$

4.4 MÉTODO DE GAUSS-SEIDEL

En el siguiente ejemplo de un sistema de ecuaciones lineales se intentará determinar el vector de incógnitas mediante el método iterativo de *Gauss-Seidel*.

$$A\,x = C = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

Tomando como vector inicial el vector

$$x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Y aplicando el método iterativo hasta que se verifique la condición

$$\left\| x^{(i+1)} - x^{(i)} \right\|_\infty < 0.0001$$

Solución

Dado el sistema de ecuaciones y el vector de términos independientes.

$$a = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \qquad c = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

Las ecuaciones iterativas del método de *Gauss-Seidel* son:

$$x_1^{(i+1)} = \frac{1}{a_{11}}\left(c_1 - a_{12}x_2^{(i)} - a_{13}x_3^{(i)} - a_{14}x_4^{(i)}\right)$$

$$x_2^{(i+1)} = \frac{1}{a_{22}}\left(c_2 - a_{21}x_1^{(i+1)} - a_{23}x_3^{(i)} - a_{24}x_4^{(i)}\right)$$

$$x_3^{(i+1)} = \frac{1}{a_{33}}\left(c_3 - a_{31}x_1^{(i+1)} - a_{32}x_2^{(i+1)} - a_{34}x_4^{(i)}\right)$$

$$x_4^{(i+1)} = \frac{1}{a_{44}}\left(c_4 - a_{41}x_1^{(i+1)} - a_{42}x_2^{(i+1)} - a_{43}x_3^{(i+1)}\right)$$

Se resuelve aplicando el método de *Gauss-Seidel*, comenzando con el vector solución inicial:

$$x_1^{(0)} = 0 \qquad x_2^{(0)} = 0 \qquad x_3^{(0)} = 0 \qquad x_4^{(0)} = 0$$

Sustituyendo en $i = 0$:

$$x_1^{(1)} = \frac{1}{6}\left(12 + 2x_2^{(0)} - 2x_3^{(0)} - 4x_4^{(0)}\right) = 2.000000$$

$$x_2^{(1)} = \frac{1}{-8}\left(34 - 12x_1^{(1)} - 6x_3^{(0)} - 10x_4^{(0)}\right) = -1.250000$$

$$x_3^{(1)} = \frac{1}{9}\left(27 - 3x_1^{(1)} + 13x_2^{(1)} - 3x_4^{(0)}\right) = 0.527778$$

$$x_4^{(1)} = \frac{1}{-18}\left(-38 + 6x_1^{(1)} - 4x_2^{(1)} - 1x_3^{(1)}\right) = 1.195988$$

$$x_1^{(1)} = 2.000000 \qquad x_2^{(1)} = -1.250000 \qquad x_3^{(1)} = 0.527778 \qquad x_4^{(0)} = 1.195988$$

Sustituyendo en $i = 1$:

$$x_1^{(2)} = \frac{1}{6}\left(12 + 2x_2^{(1)} - 2x_3^{(1)} - 4x_4^{(1)}\right) = 0.610082$$

$$x_2^{(2)} = \frac{1}{-8}\left(34 - 12x_1^{(2)} - 6x_3^{(1)} - 10x_4^{(1)}\right) = -1.444059$$

$$x_3^{(2)} = \frac{1}{9}\left(27 - 3x_1^{(2)} + 13x_2^{(2)} - 3x_4^{(1)}\right) = 0.312114$$

$$x_4^{(2)} = \frac{1}{-18}\left(-38 + 6x_1^{(2)} - 4x_2^{(2)} - 1x_3^{(2)}\right) = 1.604188$$

$$x_1^{(2)} = 0.610082 \qquad x_2^{(2)} = -1.444059 \qquad x_3^{(2)} = 0.312114 \qquad x_4^{(2)} = 1.604188$$

Sustituyendo en $i = 2$:

$$x_1^{(3)} = \frac{1}{6}\left(12 + 2x_2^{(2)} - 2x_3^{(2)} - 4x_4^{(2)}\right) = 0.345150$$

$$x_2^{(3)} = \frac{1}{-8}\left(34 - 12x_1^{(3)} - 6x_3^{(2)} - 10x_4^{(2)}\right) = -1.492954$$

$$x_3^{(3)} = \frac{1}{9}\left(27 - 3x_1^{(3)} + 13x_2^{(3)} - 3x_4^{(2)}\right) = 0.193732$$

$$x_4^{(3)} = \frac{1}{-18}\left(-38 + 6x_1^{(3)} - 4x_2^{(3)} - 1x_3^{(3)}\right) = 1.675056$$

Se prosigue la iteración hasta que la tolerancia sea la indicada.

Tabla de datos

i	$x_1^{(i)}$	$x_2^{(i)}$	$x_3^{(i)}$	$x_4^{(i)}$
0	2.000000	-1.250000	0.527778	1.195988
1	0.610082	-1.444059	0.312114	1.604188
2	0.345150	-1.492954	0.193732	1.675056
3	0.321067	-1.529280	0.125666	1.671230
4	0.334198	-1.565416	0.070368	1.655751
5	0.350905	-1.601178	0.018302	1.639342
…	…	…	…	…
235	0.998098	-2.995900	-1.994085	1.001874
236	0.998146	-2.996003	-1.994233	1.001827

$$\left\| x_1^{(236)} - x_1^{(235)} \right\| < 0.0000$$

$$\left\| x_2^{(236)} - x_2^{(235)} \right\| < 0.0001$$

$$\left\| x_3^{(236)} - x_3^{(235)} \right\| < 0.0001$$

$$\left\| x_3^{(236)} - x_3^{(235)} \right\| < 0.0000$$

La solución aproximada del sistema de ecuaciones es:

$$x^{(236)} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} = \begin{bmatrix} 0.998098 \\ -2.996003 \\ -1.994233 \\ 1.001827 \end{bmatrix} \approx \begin{bmatrix} 1 \\ -3 \\ -2 \\ 1 \end{bmatrix}$$

CONCLUSIONES

- Los métodos directos para resolver sistemas de ecuaciones lineales descritos en esta tesina presentan ventajas y desventajas entre ellos pero en general presentan grandes inconvenientes para su solución si se trata de sistemas grandes debido a que se debe de mantener mucho cuidado durante el proceso de solución, estos implican diversos procesos matemáticos en los cuales si se llega a tener un error la solución que se encuentre ya no será la exacta para el sistema implicado. Los errores que se presentan también pueden ser de redondeo. Esto hace que su solución resulte tediosa y compleja.

- El método iterativo de Gauss-Seidel aunque es un método que ofrece resolver matrices de sistemas de n>100 con notables ventajas ante los problemas con los errores numéricos gracias a que sólo es necesario identificar una vez las ecuaciones que darán solución al sistema, también presenta otras limitaciones, para que el método converja y el error de aproximación sea menor, debe de presentar en su matriz del sistema una diagonal dominante, sino podrá presentarse que en cada iteración empiecen a alejarse más y más de la solución y jamás converger.

- Se desarrollo el programa SOLECLIN (Solución de Ecuaciones Lineales) el cual permite solucionar sistemas de ecuaciones lineales sin tener la incertidumbre de que se haya cometido algún error durante el procedimiento para su solución. El correcto uso de los algoritmos matemáticos permitieron la adecuada programación dando como resultado una eficaz y rápida herramienta de análisis numérico. Adaptándose a los requerimientos del usuario y presentando diferentes métodos de solución, de esta manera ofrece diferentes alternativas para corroborar los resultados obtenidos.

GLOSARIO

A = Matriz de un sistema.

A^{-1} = Matriz inversa de A.

I = Matriz identidad.

x = Vector de incógnitas.

C = Vector de términos independientes.

$\forall$ = Para todo; para cualquier, para cada.

$\Rightarrow$ = Implica; si … entonces; por lo tanto.

$\Leftrightarrow$ = Si y sólo si.

$\rightarrow$ = De …a ...

[ ] = Agrupamiento de elementos.

{ } = Agrupamiento de elementos.

{ : } = El conjunto de todos los elementos … para los cuales … es verdadero.

{ | } = … tal que …

$\|$ = Valor absoluto.

$\approx$ = Casi igual a … (asintótico a…).

Σ = Suma sobre ... desde ... hasta ... de …

BIBLIOGRAFÍA

(1) Rodríguez Gómez Francisco Javier, “Cálculo y Métodos Numéricos (Teoría, Algoritmos y Problemas Resueltos)”. Edit. Publicaciones de la Universidad Pontifica Comillas Madrid, España 2012.

(2) Burden Richard L. y Douglas Faires J., “Análisis Numérico”. Edit. Thomson-Learning, Estados Unidos de América 2003.

(3) Becerril Espino José, Benítez Morales Lorenzo, Rivera Valladares Irene, Zubieta Badillo Carlos, “Solución de Sistemas de Ecuaciones Lineales Mediante el Método de Gauss-Jordan. Universidad Autónoma Metropolitana de Azcapotzalco. México 2004.

(4) García de Jalón, de Asís de Ribera, “Aprenda Fortran 8.0 como si estuviera en primero”. Escuela Técnica Superior de Ingenieros Industriales. España 2005.

(5) Enríquez Palma Pedro Alberto y Puyuelo García María Pilar, “Introducción a la Programación en Fortran 90 para Químicos”. Universidad de la Rioja. España 2007.

(6) García Castellano Francisco Javier, “Cuaderno de Prácticas. Fundamentos de Informática para la Ingeniería con Fortran 90”. Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada. España 2010.