INTRODUCCION AL CALCULO NUMERICO

A Q.1A Matemàtiques

INTRODUCCION AL CALCULO NUMERICO

1.- INTRODUCCION

La finalidad básica de la Informática consiste en almacenar, manipular y transmitir información.

Las características del medio utilizado para manipular la información determinan la necesidad de codificarla y escoger el tipo de codificación más adecuado. Codificar una información significa transformarla, es decir representarla de una manera distinta establecida previamente.

En nuestro caso la información deberá ser codificada en un sistema binario. Deberemos representar la información utilizando únicamente los simbolos 0 y 1 en bloques cuya longitud e interpretación estará determinada por las características del ordenador.

1.1.- UNIDADES

BIT: elemento de memoria que puede representar un valor 0 o 1

BYTE: conjunto de 8 bits

PALABRA: unidad elemental de tratamiento de datos en la memoria, su tamaño es variable de unos equipos a otros. Son normales las palabras de 16, 32 y 64 bits.

1.2.- REPRESENTACION INTERNA DE DATOS

Dentro del ordenador podemos almacenar datos numéricos y datos alfanuméricos, y tendremos diferentes tipos de representación para cada uno de estos dos tipos de datos.

1.2.1.- REPRESENTACION DE DATOS NUMERICOS

Un dato nimérico habitualmente se representa mediante una palabra de 32 bits.

1.2.1.1.- REPRESENTACION DE ENTEROS

Corresponde a la notación habitual de los numeros decimales, con un numero de cifras decimales fijo.

Binario puro: En una palabra se almacena el valor binario del numero. El numero maximo representable es 2**31-1. Los valores negativos se almacenan complementados, es decir, el resultado de restar de 0 en binario el valor positivo. Observese que en el

caso de un número positivo el primer bit de la izquierda es un 0 mientras que en el caso de un número negativo es un 1.

Ejemplos:

El numero 106 quedaria representado por

00000000	0000000	00000000	01101010

El numero -106 quedaria representado por

11111111	11111111	11111111	10010110

1.2.1.2.- REPRESENTACION DE NUMEROS DECIMALES

REPRESENTACION CON COMA FIJA

REPRESENTACION CON COMA FLOTANTE

Corresponde a la utilización de la notación cientifica o exponencial, en la forma

N = M * B ** E

Donde:
N es el numero
M fracción con signo (mantisa)
B base del sistema de numeración
E exponente con signo

El numero se almacena en una palabra en binario puro, asignando unas posiciones a la mantisa y otras al exponente.

s	mantisa	exp

El numero maximo representable es (2 ** 23 -1)* 10 ** 127

Ejemplo:

Numero 0.11835 * 10 ** 3

00000000	00101110	00111011	00000011

Numero - 0.11835 * 10 ** -3

11111111	11010001	11000101	11111101

1.2.2.- REPRESENTACION DE DATOS ALFANUMERICOS

Este tipo de datos o caracteres (las letras del alfabeto, las cifras árabes y otros simbolos como $,<,>,=,&, etc) se almacenan codificados. En cada byte se almacena el equivalente binario a un caracter.

La configuración de ceros y unos dependera del codigo utilizado, los codigos standar mas utilizados sobre bytes son

- ASCII
- EBCDIC

Algunos ejemplos de codigos ASCII

CODIGO DECIMAL	CODIGO HEXADECIMAL	CARACTER
33	21	!
48	30	0
60	3C	<
65	41	A
90	5A	Z
97	61	a
122	7A	z

2.- UTILIZACION DEL ORDENADOR

2.1.- LA HERRAMIENTA INFORMATICA VISTA DESDE EL USUARIO

PROBLEMA: Nos interesa presentar la informatica como herramienta en manos del usuario, utilizable en la resolución de un problema.

METODO: El primer paso a realizar es de hecho extrainformatico, buscar un metodo para la resolución del problema. Asi si se trata de determinar las raices de un polinomio, calcular una estructura u optimizar la trayectoria de un buque, deberemos comenzar basandonos en las Matematicas, la resistencia de los materiales o en los conocimientos de navegación. El metodo a utilizar esta asociado al experto en la materia.

ALGORITMO: El metodo a utilizar debera ser descrito en forma simple, clara y no ambiqua, de modo que pueda ser facilmente traducido a nivel informatico.

Los modelos mas usuales para la descripción de algoritmos son dos:

- pseudocodiqo
- diagramas de flujo.

PROGRAMACION: Podemos englobar aqui todas las actividades encaminadas a instruir a la maquina para que pueda realizar las funciones previstas en el algoritmo. Es decir obtener un programa en un lenquaje de programación determinado, por traducción de las especificaciones del algoritmo.

VERIFICACION: Una vez confeccionados los programas se pasara a ejecutarlo y a controlar los resultados que se obtienen, con las posibles fases de mantenimiento o remodelación de los pasos anteriores hasta obtener realmente el resultado buscado.

3.- LENGUAJES DE PROGRAMACION

3.1.- LENGUAJES.

Un lenguaje de programación queda definido por:

- Un alfabeto
- Una semantica
- Unas reglas de sintaxis

Una instrucción es un conjunto de simbolos de un alfabeto, encadenados de acuerdo con unas reglas sintacticas y que encierran un cierto significado.

Tipos de instrucciones:

Aritmeticas: sumar, restar multiplicar dividir
De transferencia de control, segun una condición.
De entrada y salida
De transferencia de datos internos
De descripción: de variables, ficheros...

3.2.- LENGUAJES DE BAJO NIVEL

Para que un ordenador pueda ejecutar un algoritmo ha de tener en memoria el programa adecuado, costituido por instrucciones que conozca la unidad de control.

El conjunto de las operaciones posibles junto con las reglas para componer instrucciones a partir de ellas constituyen lo que se denomina el LENGUAJE MAQUINA de un ordenador. Los programas en lenguaje maquina para ordenadores suelen estar codificados de manera muy poco comprensible. Y solo sirven para una maquina determinada.

Este tipo de programas suele prepararse en una forma simbolica, y convertirlos luego a lenguaje maquina con la ayuda de un programa llamado ENSAMBLADOR, que toma instrucciones simbolicas como datos y da como resultados su expresion en lenguaje maquina. A este tipo de lenguaje simbolico se le denomina lenguaje ENSAMBLADOR o ASSEMBLER. La programación en lenguaje maquina es extremadamente engorrosa, aunque sea en forma simbolica. Exige un conocimiento profundo del ordenador utilizado y en general el programa no sirve para otro tipo de ordenador. Actualmente este tipo de programación se utiliza unicamente para resolver problemas muy especializados, y casi nunca para resolver problemas comunes. Para esta ultima tarea se utilizan los denominados LENGUAJES DE ALTO NIVEL.

EJEMPLO : Programa para el VAX 780 que escribe los caracteres de codigo ASCII comprendidos entre 33 y 126

PROGRAMA EN LENGUAJE MAQUINA

```
                        0000:
                        0000:
                        0000:
                00000048  0000:
                1C000102  0004:
                00000048  0008:
                00000000  000C:
                00000001  0010:
                00000000  0014:
                00000000  0018:
                00000000  001C:
                00000000  0020:
                00000000  0024:
                00000000  0028:
                00000000  002C:
                00000001  0030:
                00000000  0034:
                00000060  0038:
                00000060  003C:
                00000000  0040:
                00000060  0044:
             4F 52 50 03  0048:
                00000004  004C:
                0016 000A  0050:
                0039 0014  0054:
                0068 001E  0058:
                0083 0028  005C:
                           0060:
                       0C  0060:
                           0061:

                           0000:
                           0000:
                      CFFC  0000:
              52 FB AF 9E  0002:
          50 00000000 GG 9E  0006:
                 51 50 D0  000D:
             00000000 GG 16  0010:
                           0016:
             FC AD FD AF 9E  0016:
       77 AB 00004304 8F 50  001B:
                77 AB 08 42  0023:

0002 77 AB 08 000043FC 8F 4F  0027:
                       4A 11  0032:
```

PROGRAMA EN ENSAMBLADOR

```
                .TITLE  PRO$MAIN
                .IDENT
                .PSECT  $PDATA
                    .LONG   72
                    .LONG   469762306
                    .LONG   72
                    .LONG   0
                    .LONG   1
                    .LONG   0
                    .LONG   0
                    .LONG   0
                    .LONG   0
                    .LONG   0
                    .LONG   0
                    .LONG   0
                    .LONG   1
                    .LONG   0
                    .LONG   96
                    .LONG   96
                    .LONG   0
                    .LONG   96
                    .ASCIC  "PRO"
                    .LONG   4
                    .WORD   10,22
                    .WORD   20,57
                    .WORD   30,104
                    .WORD   40,131
                ;       Decimal constants
                    .PACKED +0

                    .PSECT  $CODE
PRO$MAIN::
                    .WORD   ^M<R2,R3,R4,R5,R6,
                     R7,R8,R9,R10,R11,IV,DV>
                   MOVAB    .-3, R2
                   MOVAB    $PDATA, R0
                   MOVL     R0, R1
                   JSB      BAS$INIT R8
$L_10:             MOVAB    $L 10, -4(FP)
                       ; 0001
                   MOVF     #33., X(R11)
                   SUBF2    #8, X(R11)

                  ACBF 126.,#8,X(R11),$T 0034
                   BRB      $T 007E
```

```
        FC AD FD AF 9E  0034:  $T 0034:MOVAB   .-1, -4(FP)
                        0039:
        FC AD  FD AF 9E  0039:  $L 20: MOVAB    $L 20, -4(FP)
                                       ; 0002
                  00 DD  003E:          PUSHL    #0
      00000000 GG 01 FB  0040:          CALLS    #1, BAS$PRINT
            5C 77 AB 4A  0047:          CVTFL    X(R11), R12
               5C DD     004B:          PUSHL    R12
            00 A9 7F     004D:          PUSHAQ   $TMP0000(R9)
      00000000 GG 02 FB  0050:          CALLS    #2, BAS$CHR
            00 A9 7F     0057:          PUSHAQ   $TMP0000(R9)
      00000000 GG 01 FB  005A:          CALLS    #1, BAS$OUT_T_DX_S
      00000000 GG 00 FB  0061:          CALLS    #0, BAS$IO_END
         FC AD FD AF 9E  0068:  $L 30:  MOVAB    $L 30, -4(FP)
                                       ; 0003
                        0068:
        FC AD FD AF 9E  006D:  $T 006D:MOVAB    .-1, -4(FP)
FFBA 77 AB 08 B4 AF 4F  0072:          ACBF     #126., #8, X(R11),
                                      $T 0034
            77 AB 08 42  007A:          SUBF2    #8, X(R11)
        FC AD FD AF 9E  007E:  $T 007E:MOVAB    .-1, -4(FP)
                        0083:
        FC AD FD AF 9E  0083:  $L 40:  MOVAB    $L 40, -4(FP)
                                       ; 0004
    B50 00000000 0G 9E  0088:          MOVAB    $PDATA, R0
        00000000 GG 16  008F:          JSB      BAS$END_R8
              50 01 D0  0095:          MOVL     #1, R0
                 04     0098:          RET
                        0099:          .END
```

PROGRAMA EN BASIC

```
10    FOR X=33 TO 126
20    PRINT CHR$(X);
30    NEXT X
40    END
```

3.3.- LENGUAJES DE ALTO NIVEL (LAN)

Son lenguajes orientados a la programación de algoritmos con independencia de la maquina. Son inteligibles y estan orientados al problema y no a la maquina.

Una instrucción de estos lenguajes se traduce en varias sentencias de lenguaje maquina. Debido a que el ordenador solo puede ejecutar instrucciones de su propio lenguaje, los programas escritos en un lenguaje simbolico deberan traducirse a lenguaje maquina.

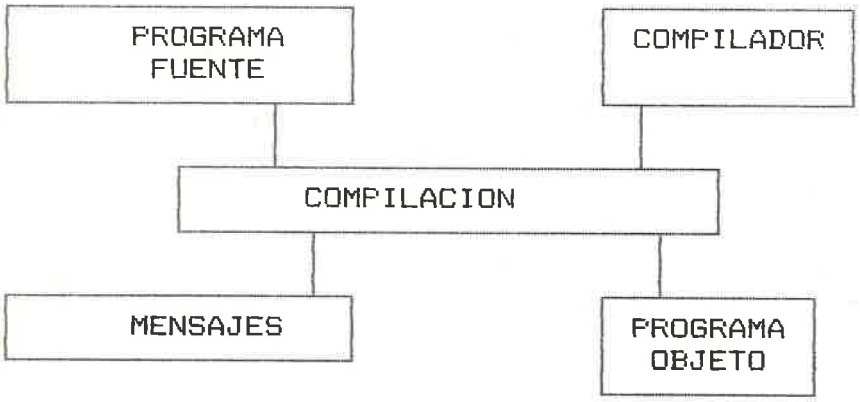
Existen dos mecanismos para la traducción a lenguaje maquina: la

compilación y la interpretación, que se efectua mediante la ejecución de un programa, compilador o interprete.

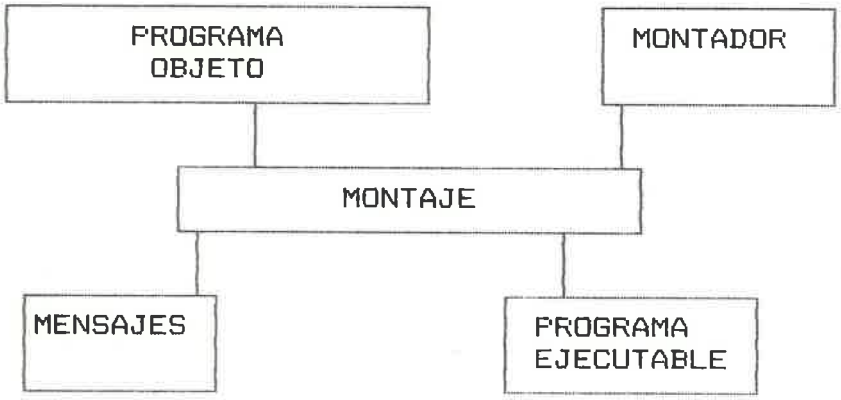
3.3.1.- COMPILACION.

El proceso de traducción se efectua en dos fases:

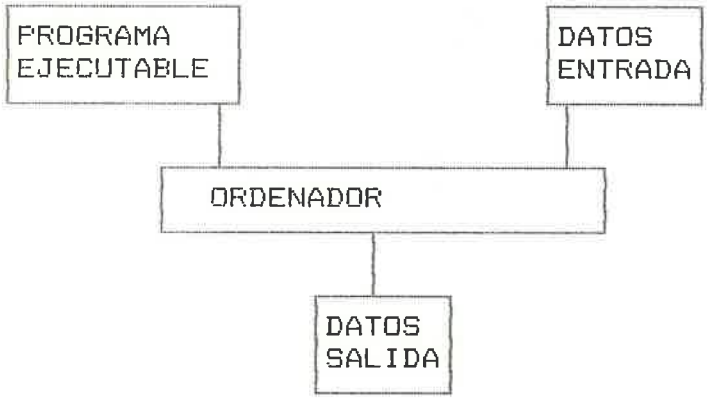
Fase de compilación: supone la ejecución de un programa, el compilador, que toma como datos de entrada las instrucciones del programa fuente (escrito en LAN) y las analiza comprobando los posibles errores sintacticos cometidos. Concluido este proceso, como información de salida se genera un listado de los errores detectados; si el proceso finaliza sin errores, se obtiene el programa objeto, es decir el programa fuente traducido a lenguaje maquina.



Fase de montaje: supone la ejecución de un programa, el montador (Linkeditor), que toma como datos de entrada la información del proograma objeto, al que incluye todo aquello que va a necesitar para su correcta ejecución. Tambien en esta fase se pueden producir errores, que quedaran reflejados en el correspondiente listado. Si no existen estos la salida sera un programa ejecutable.

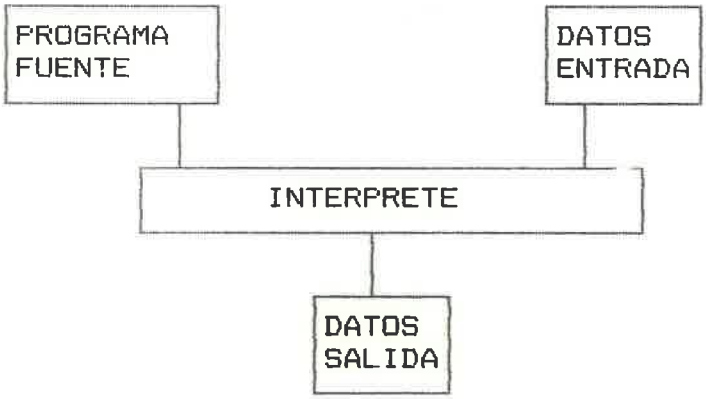


Ejecución: solo queda ejecutar el programa ejecutable.



3.3.2.-INTERPRETACION.

En este caso el proceso de traducción se realiza en una sola fase. Supone la ejecución de un programa, el interprete, que analiza las instrucciones una a una y las ejecuta a medida que las va analizando. En el momento en que detecte un error para la ejecución y escribe el mensaje correspondiente.



3.3.- LENGUAJES DE PROGRAMACION MAS COMUNES

FORTRAN: Siglas de FORmula TRANslation, introducido por IBM en 1954. Esta orientado a problemas de calculo numérico.

COBOL: Siglas de COmmon Business Oriented Lanquage, definido por una comision integrada por usuarios y fabricantes norteamericanos en 1959. Esta orientado a resolver problemas en los que se debe manejar un volumen de información considerable.

PL/I: Siglas de Programming Language number 1, definido por una comision de usuarios de IBM en 1965 y adoptado por este constructor con la ambición de ser un lenguaje util para cualquier tipo de problema. Sin embargo su amplia gama de posibilidades requiere compiladores muy complicados.

PASCAL: Esta definido para soportar programación estructurada, sus primeras versiones son del año 1970

BASIC: Siglas de Beginner's All-purpose Symbolic Instruction Code, diseñado en 1964 por Kemeny y Kurtz, con el fin de facilitar el empleo del ordenador a estudiantes y profanos. Fue creado para trabajar de forma interactiva, es decir para progamarlo y obtener resultados inmediatos. En 1952 aparece la primera versión de este lenguaje de programación junto con un compilador para el mismo.

4.- PROCESADOR Y ALGORITMOS

Llamaremos procesador a toda entidad capaz de entender y ejecutar un cierto número de enunciados dentro de un entorno o conjunto de utensilios que es capaz de manipular.

Una acción es un suceso de duración finita que modifica el entorno. Para un procesador dado distinguiremos dos tipos de acciones:

acción elemental - es aquella cuyo enunciado es suficiente para que el procesador pueda ejecutarla sin información suplementaria.

acción no elemental - es un enunciado que hace referencia a un grupo de acciones elementales y/o acciones no elementales.

Dado un tratamiento a ejecutar, un algoritmo del tratamiento está formado por una secuencia de acciones, que permiten al procesador efectuar el tratamiento deseado a traves de un lenguaje.

Dependiendo del tipo de procesador los enunciados podrán ser mas o menos complejos, por ejemplo un algoritmo para determinar si un año es bisiesto o no podria ser el siguiente:

1.- Tomar el año y ver si es divisible por 4.
2.- En caso de que no lo sea, la respuesta es NO y hemos acabado.
3.- Ver si el año termina en 100.
4.- En caso de que no, la respuesta es SI y hemos terminado.
5.- Ver si el año es divisible por 400
6.- En caso de serlo, la respuesta es SI y hemos terminado.
7.- En caso contrario la respuesta es NO y hemos terminado.

En este caso el procesador debe ser capaz de entender enunciados gramaticalmente complejos. Pero es evidente que podemos expresar este mismo algoritmo con una gramática más reducida:

```
acción
      a <-- "año"
      si a es divisible por 4
      entonces
           si a termina en 100
           entonces
                si a es divisible por 400
                entonces
                     escribir("SI")
                sino
                     escribir("NO")
                fsi
           sino
                escribir("SI")
           fsi
      sino
           escribir("NO")
      fsi
fin
```

5.- LENGUAJE PSEUDOCODIGO

Como trabajaremos en un entorno informatico, nuestro modelo de procesador requiere enunciados muy precisos, descritos en un lenguaje muy simple. Agruparemos las acciones de acuerdo con unas estructuras algorítmicas debidas a E.W.Dijsktra, que permiten una verificación formal del funcionamiento del algoritmo. El lenguaje que utilizaremos para describir algoritmos se denomina pseudocodigo.

5.1.- ESTRUCTURAS ALGORITMICAS.

Consideraremos las siguientes:

Secuencial:

```
acción 1
acción 2
   :
acción n
```

Alternativa:

```
si condición
entonces
     acción 1
sino
     acción 2
fsi
```

Iterativa:

```
mientras condición hacer
     acción
       :
     acción
fmientras
```

Se puede demostrar que con estos tres esquemas es posible describir la resolución de todos los problemas que admiten solución algoritmica. No obstante como veremos más adelante consideraremos algunas estructuras mas.

5.2.- DISEÑO DE ALGORITMOS

Para diseñar un algoritmo utilizaremos las técnicas de diseño descendente, cuyas caracteristicas generales se basan en:

- Ir de lo general a lo particular.
- Se diseña por niveles, dejando los detalles para los niveles mas bajos.
- En cada nivel se tiene que verificar que el esquema es correcto.

- A medida que descendemos vamos anotando los datos necesarios.
- Finalmente, realizamos un trabajo de recomposición del algoritmo completo.

Veamos un ejemplo de análisis descendente para un algoritmo que permita a un procesador cocinero servirnos una tortilla.

```
acción hacer una tortilla
inicio
     localizar ingredientes
     preparar ingredientes
     cocinar ingredientes
fin
```

Notemos que en este primer nivel el esquema obtenido es válido para casi todas las recetas de cocina. Continuemos el análisis descendente.

```
acción localizar ingredientes
inicio
     si hay suficientes huevos
     entonces
          cogerlos
     sino
          apuntarlo en la lista de compras
     fsi
     si hay suficiente aceite
     entonces
          cogerlo
     sino apuntarlo en la lista de compras
     fsi
     si lista de compras no vacia
     entonces
          ir a comprar
     fsi
facción

acción preparar ingredientes
inicio
     batir los huevos
facción

acción cocinar ingredientes
inicio
     calentar el aceite en una sarten
     verter los huevos en la sarten
     formar la tortilla
     colocarla en un plato
facción
```

Notemos que para poder especificar completamente un algoritmo, necesitamos especificar ademas de la secuencia de aciones el entorno con el que el procesador deberá trabajar. En el ejemplo anterior los utensilios necesarios serian una cocina, sarten, platos ..... En nuestro modelo informático de procesador este entorno estará formado por constantes y variables. Por tanto el esquema general de un algoritmo en pseudocodigo será el siguiente:

```
acción [nombre del algoritmo]
        ┌──────────────────────────┐
        │  descripción del entorno │
        └──────────────────────────┘
inicio
        ┌──────────────────────────┐
        │                          │
        │  secuencia de acciones   │
        │                          │
        └──────────────────────────┘
fin
```

5.3.- FORMALIZACION DEL ENTORNO DE UN PROBLEMA

5.3.1.- VARIABLES Y CONSTANTES.

Tenemos que definir sin ambigüedad y con precisión los objetos que componen el universo del problema. Los objetos manipulados por el procesador los clasificaremos en dos categorias constantes y variables.

Una variable es un objeto que tiene tres atributos: nombre, tipo y valor.

Nombre: es un atributo fijo, que sirve para identificarla, esta formado por caracteres alfabéticos y numéricos.
Tipo: es un atributo fijo, que indica el conjunto al que pertenece dicha variable (enteros, reales, complejos, caracteres) e implícitamente las operaciones que se pueden realizar con ellas.
Valor: es un atributo variable a lo largo del proceso,

Una constante es un objeto con los mismos atributos que una variable pero de valor fijo.

```
                NOMBRE    TIPO
var
                 total  : real
                     k  : entero
                 texto  : tabla(p,q,r) de caracter
                     a  : tabla(n,m) de real
const
            pi=3.14159  : real
           maximo=24.2  : real
```

5.3.2.- TIPOS DE DATOS ELEMENTALES

Distinguiremos dos clases de tipos de datos, tipos elementales o escalares que no son divisibles en componentes y tipos estructurados. En este capítulo nos referiremos únicamente a los tipos elementales de uso más frecuente que incluyen los valores lógicos, los números enteros y reales y el conjunto de caracteres utilizables por un ordenador.

Para describir un tipo tenemos que especificar cual es el rango de valores permitidos asi como las operaciones que se pueden efectuar con ellos.

Tipo lógico (boolean): El conjunto de valores es el correspondiente a los valores lógicos verdadero (true) o falso (false). Con las siguientes operaciones:

"O" disyunción (OR)
"Y" conjunción (AND)
"NO" negación (NOT)

Tipo real: El conjunto de valores son los numeros reales con la precision que permita el procesador junto con las operaciones:

+ suma
- resta
* producto
/ division
^ exponenciación

Tipo entero: El conjunto de valores son los numeros enteroscon la precision que permita el procesador junto con las operaciones:

+ suma
- resta
* producto
div division entera
mod resto de la division entera

Tipo caracter: El conjunto de valores son las cadenas de caracteres que se pueden formar a partir de los elementos del

conjunto de caracteres (letras , cifras y signos especiales) que permita el procesador junto con las operaciones:

+	concatenación
long	longitud de una cadena de caracteres
extraer	extraccion de una subcadena
pos	posicion relativa de un caracter dentro de una cadena

Ejemplos: "aaabbbbb" + "ccaaa" = "aaabbbbbccaaa"

long("abcdef")=6

extraer("aabcd",2,2)="ab"

pos("abcdef","bc")=2

Observación: Un valor caracter se escribe siempre entre comillas, para diferenciarlo de un nombre de variable.

5.4.- DEFINICION DEL ENTORNO

Tenemos que especificar todas las variables y constantes que aparecen en el universo del problema y el tipo al que pertenecen. Para ello utilizaremos la siguiente sintaxis: especificaremos un bloque precedido por la palabra const con todas las constantes que utilice el algoritmo, sequido de un bloque precedido por la palabra var con la especificación de todas las variables.

Ejemplo :

```
const
      pi=3.14159 : real
     maximo=24.2 : real
var
     x,total,n 1 : real
           i,j,k : entero
           texto : tabla(p,q,r) de caracter
inicio
     acción
     .
     .
     .
     acción
fin
```

Si un identificador (nombre) corresponde a una constante tenemos que especificar su valor y su tipo. Si corresponde a una variable sólo el tipo.

Cuando tenemos varias constantes o variables del mismo tipo las

agruparemos en una lista de nombres separada por "," seguida de ":" y del tipo.

5.5.- DOCUMENTACION DEL ENTORNO

Con las especificaciones anteriores el procesador tiene información suficiente para poder llevar a cabo el trabajo. Sin embargo no debemos olvidar que el algoritmo debe ser comprensible para cualquier persona y por tanto será conveniente introducir aclaraciones (comentarios) sobre el contenido o significado de las variables, asi como de algunos de los fragmentos de código. Para indicar que un texto es un comentario escribiremos este texto precedido por el símbolo ";".

6.- ACCIONES ELEMENTALES BASICAS

6.1.- ACCION DE ASIGNACION

El efecto es dotar de valor a una variable. Puede ser interna o externa dependiendo de que el valor que queramos asignar se pueda obtener o no de los valores de las variables del entorno o no.

6.1.1.- ASIGNACION INTERNA

var <-- exp

donde :

var es el nombre de la variable a la que el procesador debe asignarle un valor .

exp representa el valor a asignar y puede ser una constante,el nombre de otra variable que contenga el valor o una expresion describiendo un calculo a efectuar.

Ejemplos:
```
a <-- 25
v <-- v+1
d <-- b^2-4*a*c
```

Lo primero que hace el procesador es calcular el valor de la expresion y despues asignar este valor a la variable. En este caso dotamos de valor a la variable a partir de valores que el procesador tiene en su entorno.

6.2.- ASIGNACION EXTERNA

Tenemos tres tipos de entrada de datos , entrada por pantalla, desde el programa y desde archivo.

ENTRADAS DESDE PANTALLA

leer (lista de variables y textos)

Ejemplos:
```
leer(a)
leer(a,b,c,e)
leer("primer número",a,"segundo número,b)
```

ENTRADAS DESDE PROGRAMA

Los datos se especifican en sentencias lista del programa; con el formato:

lista nom lista

Los datos se asignan a las variables mediante:

leer nom lista

Ejemplo:

```
lista nombres juan,pedro,ignacio
leer lista nombres x,y,z
```

ENTRADAS DESDE ARCHIVO

En este caso los datos estarán en un archivo creado, como salida de otro programa o bien mediante el editor. Para especificar en que archivo estan los datos de entrada usamos la sentencia OPEN con dos formatos posibles:

abrir (nom fixero,nom registro,modo)

Donde:

nom archivo	es el nombre del archivo donde estan los datos.
registro	variable donde se descarga el contenido del archivo de datos.
modo	describe si sera para lectura o escritura

Para la lectura de datos utilizaremos la sentencia leer con el formato :

leer (nom archivo,registro)

Antes de acabar el programa tenemos que cerrar todos los archivos abiertos con la sentencia cerrar con el formato :

cerrrar nom archivo

6.3.- SALIDA DE DATOS

SALIDAS POR PANTALLA

escribir (lista de variables,expresiones y textos)

Con esta acción el procesador comunica al exterior los valores que ha calculado y que a nosotros nos interesa obtener como resultado.

Ejemplo:

```
escribir(a,b)
escribir("la media de",a,"y",b,"es",(a+b)/2)
```

SALIDAS EN ARCHIVO.

Tenemos que abrir el archivo mediante la sentencia abrir:

escribir (nom archivo,registro)

NOTA: A continuación vamos a describir las distintas acciones que se utilizan en la elaboración de un algoritmo. En todas ellas hay dos versiones. La primera opción se supone dentro de un entorno informático totalmente estructurado, mientras que la segunda es una simulación en otro entorno no estructurado.

6.4.- ACCIONES CONDICIONALES

Son acciones que el procesador ejecutara dependiendo de que una condición se cumpla o no, de acuerdo con los valores de las variables en este momento.

Una condición consiste en una o varias comparación entre variables y/o expresiones ligadas por conectivas logicas (y, o, no).

Observación : los operadores de comparación son :

```
<   menor
<=  menor o igual
>   mayor
>=  mayor o igual
=   igual
<>  distinto
```

ESTRUCTURA ALTERNATIVA SIMPLE

```
si [condición]                  linea1  si no (condición) ir a linea2
entonces
     acción                          acción
       .                               .
       .                               .
     acción                          acción
fsi                             linea2
```

Si se cumple la condición el procesador ejecutara las acciones que estan entre la palabra "entonces" y la palabra "fsi", si es falsa no hara nada y continuara con la siguiente acción.

Ejemplo : Supongamos que, en un paso determinado del algoritmo, queremos que una variable x de tipo numérico (real o entera), tome su valor absoluto. La manera de indicarlo seria la siguiente

```
si x<0
entonces
     x <-- -x
fsi
```

ESTRUCTURA ALTERNATIVA DOBLE

```
si [condición]                  ||   linea1 si no (condición ) ir a linea2
entonces                        ||
     acción                     ||            acción
       .                        ||              .
       .                        ||              .
     acción                     ||            acción
sino                            ||            ir a linea3
     acción                     ||   linea2  acción
       .                        ||              .
       .                        ||              .
     acción                     ||            acción
fsi                             ||   linea3  ......
```

Si se cumple la condición, se ejecutaran las acciones comprendidas entre "entonces" y "sino" y despues pasará a la acción siguiente al "fsi", si la condición es falsa ejecutara las acciones entre "sino" y "fsi".

Ejemplo : Calculo del maximo. Tenemos dos variables a,b y queremos que en una variable max quede el valor maximo de a y b

```
si a > b
entonces
     max <-- a
sino
     max <-- b
fsi
```

ESTRUCTURA ALTERNATIVA MULTIPLE

```
opción                          ||
     caso cond1                 ||   linea1 si no (cond1) ir a linea2
         acción                 ||       acción
           :                    ||         :
           :                    ||         :
         acción                 ||       acción
     fcaso                      ||   ir a lineafinal
     caso cond2                 ||   linea2 si no (cond2) ir a linea3
         acción                 ||       acción
           :                    ||         :
         acción                 ||       acción
     fcaso                      ||   ir a lineafinal
           :                    ||   linea3 si no (cond...) ....
           :                    ||         :
     caso condn                 ||   linean si no (condn) ir a lineafinal
          acción                ||       acción
           :                    ||         :
          acción                ||       acción
     fcaso                      ||
fopción                         ||   lineafinal
```

El procesador ejecutara las acciones del primer caso cuya condición se verifique, si no se verifica ninguna no ejecutara ninguna acción.

Ejemplo: Menu de operaciones : los datos de entrada son dos numeros y el codigo de la operación a realizar con el siguiente convenio :

1 suma
2 resta
3 maximo
4 minimo
5 valor medio

Podemos considerar el siguiente algoritmo que resuelve el problema :

```
acción menu de operaciones
var dato1,dato2 : real
     op : entero
inicio
     leer(dato1,dato2,op)
     opción
          caso op=1 hacer
               escribir("suma",dato1+dato2)
          fcaso
          caso op=2 hacer
               escribir("resta",dato1-dato2)
          fcaso
          caso op=3 hacer
               si dato1>dato2
               entonces
                    escribir("maximo",dato1)
               sino
                    escribir("maximo",dato2)
               fsi
          fcaso
          caso op=4 hacer
               si dato1<dato2
               entonces
                    escribir("minimo",dato1)
               sino
                    escribir("minimo",dato2)
               fsi
          fcaso
          caso op=5 hacer
               escribir("promedio",(dato1+dato2)/2)
          fcaso
fopción
```

6.5.- EJEMPLOS

1.- Cálculo del área y de la longitud de un círculo. En este caso el único dato a leer será el radio del círculo.

```
acción círculo
const     pi=3.14159:real
var       r,area,long:real
inicio
     leer(r)
     area <-- pi*r^2
     long <-- 2*pi*r
     escribir("area :",area)
     escribir("longitud :",long)
fin
```

2.- Cálculo del máximo de tres números

```
acción max de tres
var  a,b,c:real
inicio
     leer(a,b,c)
     si a >= b
     entonces
          si a >= c
          entonces
               escribir("maximo",a)
          sino
               escribir("maximo",c)
          fsi
     sino
          si b >= c
          entonces
               escribir("maximo",b)
          sino
               escribir("maximo",c)
          fsi
     fsi
fin
```

3.- Calculo del maximo comun divisor: El algoritmo de Euclides para hallar el M.C.D. de dos numeros consiste en lo siguiente: Se efectua la division entera entre el mayor y el menor. Si el resto es 0 , el M.C.D. es el divisor. Si no es el mismo que el del divisor y el resto. Se repiten los pasos hasta tener un resto 0.

```
acción mcd algoritmo Euclides
var  a,b,r : entero
inicio
     leer(a,b)
     si a<b
     entonces
          r <-- a
          a <-- b
          b <-- r
     fsi
     iterar
          r <-- a mod b
          salir si r = 0
          a <-- b
          b <-- r
     fiterar
     escribir (" maximo comun divisor ",b)
fin
```

4.- Resolución de una ecuación de segundo grado. Supondremos que la ecuación es $a\,x^2 + b\,x + c = 0$

```
acción ecuación de segundo grado
var  a,b,c,x,disc:real
inicio
     si a=0
     entonces escribir("ecuación de primer grado")
     sino
          disc <-- b^2-4*a*c
          opción
             caso disc <0
               escribir("raices complejas")
               escribir("parte real :",-b/2/a)
               escribir("parte imaginaria :",(-disc)^(1/2)/2/a)
             fcaso
             caso disc=0
               escribir("una raiz doble :", -b/2/a)
             fcaso
             caso disc > 0
               escribir("dos raices reales")
               disc <-- disc^(1/2)
               escribir((-b+disc)/2/a)
               escribir((-b-disc)/2/a)
             fcaso
          fopción
     fsi
fin
```

6.6.- ACCIONES ITERATIVAS

6.6.1.- ITERACION CONDICIONAL

MIENTRAS

```
mientras [condición] hacer
     acción
       .
       .
     acción
fmientras
```

```
linea1 si no condición ir a fin
     acción
       .
       .
     acción
     ir a linea1
fin
```

Si la condición se cumple se ejecuta el bloque de acciones comprencidas entre "mientras" y "fmientras". Una vez ejecutado el bloque la condición se vuelve a evaluar. El proceso se repetirá hasta que la condición no se cumpla, en cuyo caso se pasa a ejecutar la acción siguiente a fmientras. Por tanto si la primera vez la condición no se cumple no se ejecutará ninguna de las acciones.

Ejemplo : Calculo de la suma de los n primeros enteros

```
acción suma enteros
inicio
     numero<-- 1
     suma <-- 0
     mientras numero <= n hacer
          suma <-- suma + numero
          numero <-- numero + 1
     fmientras
fin
```

REPETIR

```
repetir
     acció
       .
       .
     acción
hasta que [condición]
```

```
linea1  acción
          .
          .
        acción
        si no  (condición) ir a linea1
```

Se ejecuta el bloque de acciones entre "repetir" y "hasta que". La condición se evalua despues de ejecutar el bloque. Cuando se detecte que no se cumple se pasa a la acción siguiente a "hasta que". Por tanto el bloque de acciones se ejecutan por lo menos una vez.

Ejemplo : Suma de los n orimeros enteros

```
acción suma enteros
inicio
      numero <-- 1
      suma <-- 0
      repetir
            suma <-- suma + numero
            numero <-- numero + 1
      hasta que numero > n
fin
```

ITERAR

```
iterar
      acción
        .
        .
      acción
      salir si [condiciónn]
      acción
        .
        .
      acción
fiterar
```

```
linea1  acción
          .
          .
          acción
          si condición ir a linea2
        acción
          .
          .
        acción
        ir a linea1
linea2  ..........
```

En este caso se evalua la condición en un punto intermedio de la secuencia , de donde se sale si la condición se verifica. Por lo tanto las acciones que estan antes del primer "salir si" se ejecutan al menos una vez .

Ejemplo : Calculo de n terminos de la sucesion de Fibonacci, n>2
($a_n = a_{n-1} + a_{n-2}$ con $a_0=0$ y $a_1=1$)

```
acción fibonacci
inicio
      anterior <-- 0
      actual <-- 1
      escribir (anterior)
      contador <-- 1
      iterar
            escribir(actual)
            contador <-- contador + 1
            salir si contador = n
            aux <-- anterior + actual
            anterior <-- actual
            actual <-- aux
      fiterar
fin
```

Observación : en cualquiera de los tres casos dentro de las acciones se deben modificar las variables que intervienen en la condición ya que sino se repetiria la ejecución de las acciones indefinidamente.

6.6.2.- ITERACION CON VARIABLE DE CONTROL

PARA

```
para [nom var=exp1] hasta exp2 incremento exp3 hacer
     acción
       .
       .
     acción
fpara
```

```
                              nom var <-- exp1
                    linea1    accion
                              .
                              .
                              accion
                              nom var <-- nom var + exp3
                              si nom var>exp2 ir a lineafin
                              ir a linea1
                    lineafin
```

donde :

- var: es un nombre de variable
- exp1: es una expresion numerica cuyo valor se asigna inicialmente a la variable.
- exp2: es una expresion numerica con cuyo valor se compara el valor de la variable. Cuando este sea mayor se finaliza la iteración.
- exp3: es una expresion numerica que se suma al valor de la variable cada vez que se ejecuta el grupo de acciones.

Ejemplo : suma de los primeros n enteros pares

```
acción suma pares
inicio
     suma <-- 0
     para i=2 hasta 2*n incremento 2 hacer
          suma <-- suma + i
     fpara
fin
```

7.- EQUIVALENCIAS ENTRE PSEUDOCODIGO Y DIAGRAMAS DE FLUJO

7.1.- DIAGRAMAS DE FLUJO.

Los diagramas de flujo, tambien llamados organigramas u ordinogramas, son metodos para expresar un algoritmo en forma grafica. Tienen el inconveniente de que su estructura rudimentaria obliga en ocasiones a complicaciones innecesarias en comparación con otros metodos de representación, como el pseudocodigo. Por otra parte al no permitir un diseño descendente, para algoritmos relativamente intrincados, resultan engorrosos y bastante incomprensibles.Para expresar un algoritmo mediante un diagrama de flujo se utilizan, de forma estandar, una serie de simbolos graficos. Veamos a continuación algunos de ellos, y las acciones elementales a las que se corresponden.

7.2.- SIMBOLOS GRAFICOS

SIMBOLO INICIAL Y TERMINAL

Todo diagrama empieza con el simbolo de inicio y acaba con el simbolo de final. Se corresponden a las palabras inicio y fin de los algoritmos descritos en pseudocodigo

```
acción
     Var
       :
       :

     inicio
       :
       :
      fin
```

Notemos que en los diagramas de flujo no se especifica la información relativa a las variables utilizadas, ni el tipo de las mismas.

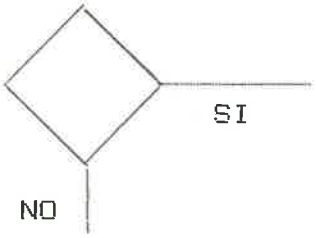
SIMBOLO DE ENTRADA DE DATOS

leer (lista de variables)

SIMBOLO DE SALIDA DE DATOS

escribir(lista de variables,
textos y expresiones)

SIMBOLO DE DECISION



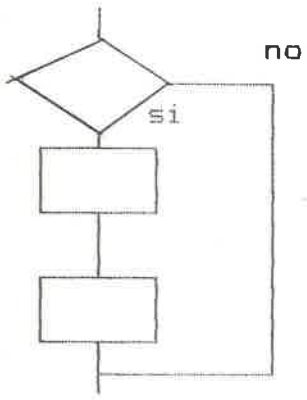
SIMBOLO DE ACCION A EJECUTAR

acción

7.3.- DIAGRAMAS DE FLUJO Y ACCIONES ELEMENTALES. EQUIVALENCIAS.

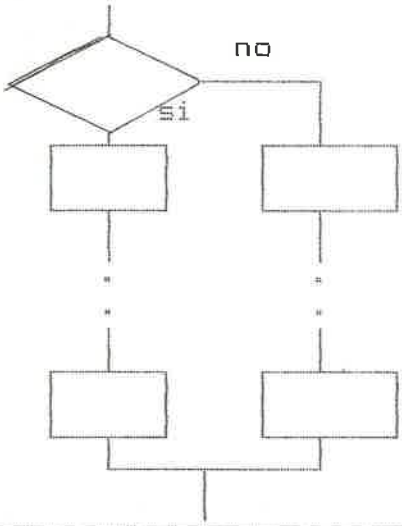
CONDICIONALL SIMPLE

```
si condición
entonces
      accion1
        :
      accionf
fsi
```



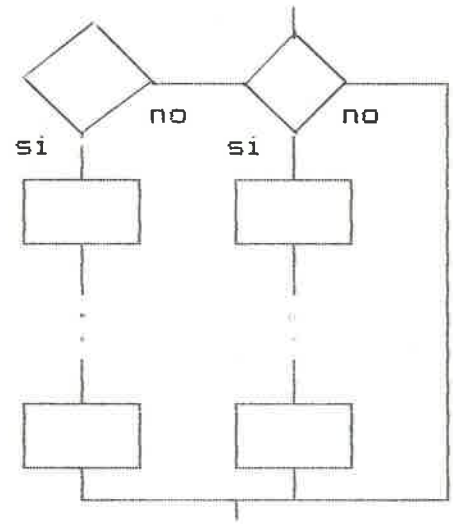
CONDICIONAL DOBLE

```
si condición
entonces
      accion11
        :
      accion1f
sino
      accion21
        :
      accion2f
fsi
```



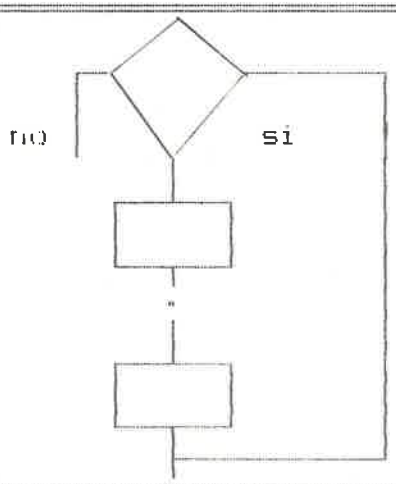
CONDICIONAL MULTIPLE

```
opción
      caso condición 1
            acción 1 1
              :
            acción 1 l
      fcaso
            :
            :
      caso condición f
            acción f 1
              :
            acción f l
      fcaso
fopción
```



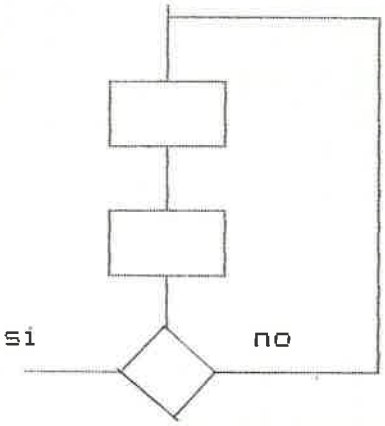
MIENTRAS

```
mientras condición hacer
      acción1
        :
      acciónl
fmientras
```



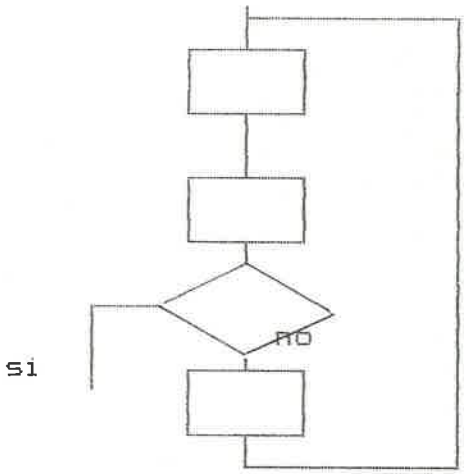
REPETIR

```
repetir
      acción1
        :
      acción1
hasta que condición
```



ITERAR

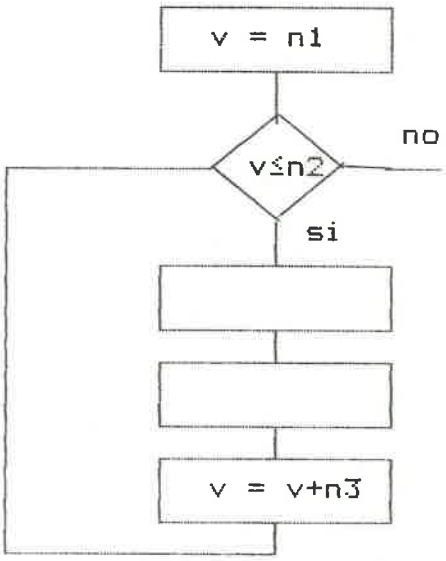
```
iterar
      acción1
        :
      acción1
salir si condición
        :
fiterar
```



ITERACION CON VARIABLE DE CONTROL

```
para v=n1 hasta n2
      incremento n3 hacer
      acción1
        :
      acción1

fpara
```



8.- TIPOS DE DE DATOS NO ELEMENTALES

La posibilidad de asociar un nombre colectivo a un conjunto de variables es muy importante en el procesamiento de la información. Un conjunto de variables reconocido por un nombre colectivo diremos que es una variable de un tipo no elemental de datos.

Para definir tipos no elementales tenemos que especificar:

- La forma en que está estructurado (definición de tipo)
- El método de acceso a las componentes individuales.
- El tipo de las componentes.
- En algunos casos las operaciones posibles.

Estudiaremos algunos tipos no elementales como son tablas y registros, aunque hay otros como listas, colas, pilas y archivos.

Evidentemente podremos definir nuevos tipos de datos utilizando definiciones previas de otros tipos no elementales y/o directamente a partir de los tipos elementales.

8.1.-TIPO TABLA

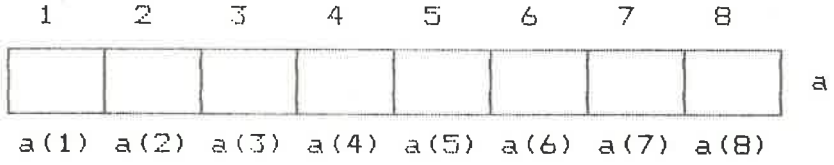
Consiste en un numero fijo de componentes, todas del mismo tipo base. Cada componente es accesible directamente indicando el nombre de la variable tabla sequida por uno o mas números llamados indices. Los rangos posibles de estos indices se denominan dimensiones de la tabla, y al número de índices número de entradas de la tabla.

Ejemplo: tipo v=tabla(8) de entero

Estamos definiendo un tipo tabla de una entrada de dimensión 8. Notemos que v es el nombre del tipo. Para definir una variable de este tipo tendriamos que especificar :

var a:v

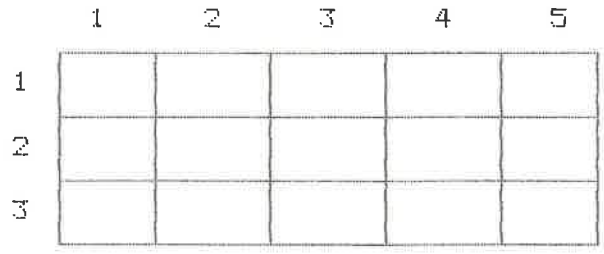
con lo que creamos una estructura de datos como la que sigue:



Es practica habitual omitir la definición de tipo y se incorpora a la definición de la variable.

```
var a:tabla (3,5) de real
```

corresponde a la estructura siguiente:



Para referenciar una casilla de la tabla utilizaremos el convenio a(i,j) donde i representa la fila y j la columna.

La representación geométrica no se corresponde con la representación real dentro de un ordenador. En el ordenador se almacena secuencialmente. En principio se pueden definir tablas de tantas entradas como nos interese.

Entenderemos habitualmente que los índices varian entre 1 y el valor especificado, a no ser que se indique lo contrario, por ejemplo:

```
var alfa: tabla(-1:5,0:27,-6:54,-1:1)
```

El primer iíndice podrá tomar valores entre -1 y 5 etc..

8.2.- TIPO REGISTRO

Consiste en un numero fijo de componentes llamados campos. Cada componente puede tener caracteristicas muy distintas. El acceso a un campo se indicara con el nombre de la variable, seguido de "." y del nombre generico del campo.

Ejemplo:

```
tipo registro=complejo
re,im : real
fregistro
var  z:complejo
```

Para especificar uno de los campos utilizaremos

```
z.re    z.im
```

Nota: En este caso las posibles operaciones sobre las variables, del tipo que hayamos definido, siempre las deberemos definir nosotros.

9.- ACCIONES NO ELEMENTALES

9.1.- ALGORITMOS Y FUNCIONES.

Son algoritmos (subrutinas o funciones) que realizan un cierto tratamiento, necesario en el algoritmo general, a partir de unos datos (parametros) obtenidos en el algoritmo general. Cada acción no elemental queda identificada por un nombre, y cualquier referencia a la acción tiene el formato :

[nombre acción] (lista parametros)

Si la acción unicamente calcula un valor, que utilizara el algoritmo general, utilizaremos el termino función en vez del de acción, por tanto una función solo podra ser referenciada al lado derecho de una asignación.

La definición de acciones tiene el siguiente formato :

```
acción  [nombre acción] ( lista de variables)
inicio
     acción
       .
       .
     acción
facción
```

La definición de funciones tiene el siguiente formato :

```
función [nombre función] ( lista de variables ) : tipo
inicio
     acción
       .
       .
     acción
ffunción
```

Donde tipo corresponde al tipo del valor que calcula la función. Dentro de la función tiene que existir una asignación al nombre de la función que sera el valor que calcula la función.

Observación : La lista de parametros y la lista de variables tienen que tener siempre el mismo numero de elementos, cuando se inicie el calculo de la acción se asignan a las variables de la lista los valores de los parametros en el orden en que estan escritos.

En la mayoria de los casos utilizaremos acciones sin parámetros como ocurre en el siguiente ejemplo.

Ejemplo: Tenemos datos de entrada formados por grupos de tres números. El primero corresponde al dia de la semana, el segundo corresponde a un producto (numerados de 1 a 8), y el tercero a la cantidad de este producto vendida en este dia de la semana.Queremos escribir un algoritmo para que la salida sea:

1) para cada dia de la semana el total vendido.
2) para cada producto la cantidad vendida a lo largo de la semana.

Supondremos que tenemos una tabla de 8 entradas donde almacenaremos los precios de los 8 productos. Y que el fin de datos lo detectaremos cuando el dia de la semana sea un 0.

Utilizaremos una tabla de dimensiones 6x8 para almacenar los datos leidos, antes de efectuar el cálculo.

```
acción ventas semanales almacen
var  dia producto:tabla(6,8) de real
     precio:tabla(8) de real
     i,j,m,n:entero
     p,t,total: real
inicio
     total <-- 0
     leer datos
     calcular y escribir total por dia
     calcular y escribir total por producto
     escribir("total ventas semana :",total)
fin


acción leer datos
inicio
     para i=1 hasta 8 hacer
          leer(precio(i))
     fpara
     para i=1 hasta 6 hacer
          para j=1 hasta 8 hacer
               dia producto(i,j) <-- 0
          fpara
     fpara
     iterar
          leer(m)
     salir si m=0
          leer(n,p)
          dia producto(m,n) <-- dia producto(m,n) + p
     fiterar
facción
```

```
acción calcular y escribir total por dia
inicio
     para i=1 hasta 6 hacer
          t <-- 0
          para j=1 hasta 8 hacer
               t <-- t + dia producto(i,j)*precio(j)
               escribir("dia",i,"total",t)
          fpara
          total <-- total + t
     fpara
facción
```

```
acción calcular y escribir total por producto
inicio
     para i=1 hasta 8 hacer
          t <-- 0
          para j=1 hasta 6 hacer
               t <-- t + dia producto(j,i)
               escribir("producto",i,"total",t)
          fpara
          total <-- total + t
     fpara
facción
```

Hasta este momento se han expuesto las bases de la programación en lenguaje pseudocódigo. A partir de ahora nuestro interes se centrará en el desarrollo de los algoritmos que permiten resolver ciertos problemas basicos del Cálculo y del Algebra Lineal. En la mayoría de los casos las bases teóricas de estos problemas ya estan tratadas en las correspondientes publicaciones del departamento. En caso contrario se hará una breve introducción teórica del tema correspondiente.

10.- CALCULO MATRICIAL

10.1.- SUMA DE MATRICES:

Si a y b son dos matrices nxm, A+B es la matriz en la que cada elemento es la suma de los elementos de A y B que están en la misma posición. Es decir, si C=A+B el elemento de la fila i y columna j de C se calcula como: c(i,j)=a(i,j)+b(i,j)

```
acción suma matrices
var      n,m,i,j : entero                    { nxm  dimensión matrices
           a,b,c : tabla(n,m) de real
inicio
      leer (n)
      leer matriz (a)
      leer matriz (b)
      para i=1 hasta n hacer
            para j=1 hasta m hacer
                  c(i,j) <-- a(i,j) + b(i,j)
            fpara
      fpara
      escribir matriz (c)
fin
```

```
10 REM ******************************************************
20 REM ***    SUMA DE MATRICES DE DIMENSIONES NXM ***
30 REM ******************************************************
40 REM
50 INPUT "DIMENSIONES",N,M
60 DIM A(N,M),B(N,M),C(N,M)
70 REM LECTURA DE LA MATRIZ A
80 PRINT "INTRODUZCA LA MATRIZ A POR FILAS"
90 FOR I=1 TO N
100        FOR J=1 TO M
120              INPUT A(I,J)
130        NEXT J
140 NEXT I
150 REM LECTURA DE LA MATRIZ B
160 PRINT "INTRODUZCA LA MATRIZ B POR FILAS"
170 FOR I=1 TO N
180        FOR J=1 TO M
190               INPUT B(I,J)
200        NEXT J
210 NEXT I
220 REM CALCULO DE LA MATRIZ C
230 FOR I=1 TO N
240        FOR J=1 TO M
250              C(I,J)=A(I,J)+B(I,J)
260        NEXT J
270 NEXT I
280 REM ESCRITURA DE LA MATRIZ C
290 FOR I=1 TO N
```

```
300         FOR J=1 TO M
310              PRINT C(I,J);
320         NEXT J
330         PRINT
340 NEXT I
350 END
```

10.2.- PRODUCTO MATRIZ Y UN ESCALAR:

Si A es una matriz nxm y p un número real p.A es la matriz mxn en la que cada elemento es el producto de p y el elemento de A que está en la misma posición. Si C=p.A el elemento de la fila i y columna j de C se calcula como: c(i,j)=p*a(i,j)

```
acción producto matriz y escalar
var   n,m,i,j : entero   ! nxm dimensión matriz
           esc : real
           a,c : tabla (n,n) de real
inicio
     leer (n)
     leer matriz (a)
     leer (esc)
     para i=1 hasta n hacer
          para  j=1 hasta m hacer
               c(i,j) <-- esc * a(i,j)
          fpara
     fpara
     escribir matriz (c)
fin
```

```
10 REM ******************************************************
20 REM ***       PRODUCTO MATRIZ (NXM) Y ESCALAR          ***
30 REM ******************************************************
40 REM
50 INPUT "DIMENSIONES".N,M
60 DIM A(N,M),C(N,M)
70 REM LECTURA DE LA MATRIZ A
80 PRINT "INTRODUZCA LA MATRIZ A POR FILAS"
90 FOR I=1 TO N
100        FOR J=1 TO M
110             INPUT A(I,J)
120        NEXT J
130 NEXT I
140 REM LECTURA DEL ESCALAR
150 INPUT "ESCALAR",ES
160 REM CALCULO DE LA MATRIZ C
170 FOR I=1 TO N
180        FOR J=1 TO M
190             C(I,J)=ES*A(I,J)
200        NEXT J
210 NEXT I
```

```
220 REM ESCRITURA DE LA MATRIZ C
230 FOR I=1 TO N
240       FOR J=1 TO M
250            PRINT C(I,J);
260       NEXT J
270       PRINT
280 NEXT I
290 END
```

10.3.- PRODUCTO DE DOS MATRICES

Si A es una matriz nxm y B es una matriz mxp, el elemento de la fila i y columna j de la matriz C=A.B se calcula como: c(i,j)==>a(i,k)*b(k,i), variando k desde 1 hasta el número máximo de columnas de la matriz, que debe coincidir con el número de filas la matriz B.

```
acción producto de matrices
var   n,m,p,i,j,k : entero
                a : tabla (n,m) de real
                b : tabla (m,p) de real
                c : tabla (n,p) de real
inicio
     leer (n,m,p)
     leer matriz (a)
     leer matriz (b)
     para i=1 hasta n hacer
          para j=1 hasta p hacer
             c(i,j) <-- 0
             para k=1 hasta m hacer
                 c(i,j) <-- c(i,j) + a(i,k) * b(k,j)
             fpara
          fpara
     fpara
     escribir matriz (c)
  fin
```

```
10 REM ******************************************************
20 REM ***  PRODUCTO DE MATRICES
30 REM ******************************************************
40 REM
50 INPUT "DIMENSIONES N,M,P",N,M,P
60 DIM A(N,M),B(M,P),C(N,P)
70 REM LECTURA DE LA MATRIZ A
80 PRINT "INTRODUZCA LA MATRIZ A POR FILAS"
90 FOR I=1 TO N
100       FOR J=1 TO M
110            INPUT A(I,J)
120       NEXT J
130 NEXT I
140 REM LECTURA DE LA MATRIZ B
```

```
150 PRINT "INTRODUZCA LA MATRIZ B POR FILAS"
160 FOR I=1 TO M
170       FOR J=1 TO P
180            INPUT B(I,J)
190       NEXT J
200 NEXT I
210 REM CALCULO DE LA MATRIZ C
220 FOR I=1 TO N
230       FOR J=1 TO P
240            C(I,J)=0
250            FOR K=1 TO M
260                 C(I,J)=C(I,J)+A(I,K)*B(K,J)
270            NEXT K
280       NEXT J
290 NEXT I
300 REM ESCRITURA DE LA MATRIZ C
310 FOR I=1 TO N
320       FOR J=1 TO M
330            PRINT C(I,J);
340       NEXT J
350       PRINT
360 NEXT I
370 END
```

11.- RESOLUCION DE SISTEMAS DE ECUACIONES. METODOS DIRECTOS

11.1.- SISTEMAS CON MATRIZ DE COEFICIENTES SIMPLE.

Caso 1: La matriz del sistema es una matriz diagonal. En este caso la resolución del sistema es inmediata.

$$\begin{aligned} a_{1,1}x_1 &= b_1 & x_1 &= b_1/a_{1,1} \\ a_{2,2}x_2 &= b_2 & x_2 &= b_2/a_{2,2} \\ &\vdots & &\vdots \\ a_{n,n}x_n &= b_n & x_n &= b_n/a_{n,n} \end{aligned}$$

Caso 2: El sistema es triangular inferior. En este caso podemos resolver el sistema por sustitución, empezando por calcular el valor de x_1 y terminando por el valor de x_n.

$$\begin{aligned} & a_{1,1}x_1 && && = b_1 \\ & a_{2,1}x_1 + a_{2,2}x_2 && && = b_2 \\ & a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 && && = b_3 \\ & \vdots && \vdots && \vdots \\ & a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,n-1}x_{n-1} + a_{n,n}x_n && && = b_n \end{aligned}$$

En donde la ecuación que nos calcula el valor de la incógnita x_k en función de los valores de las incógnitas $x_{k-1}, x_{k-2}, \ldots, x_1$ es:

$$x_k = (b_k - a_{k,1}x_1 - \ldots - a_{k,k-1}x_{k-1})/a_{k,k}$$

El algoritmo que realiza este proceso lo podemos estructurar como una acción no elemental, a partir de una tabla "a", matriz de coeficientes de nxn, y de una tabla "b", matriz de términos independientes de nx1.

```
acción resolver sistema triangular inferior
inicio
     para k=1 hasta n hacer
          x(k) <-- b(k)
          para i=1 hasta k-1 hacer
               x(k) <-- x(k) - a(k,i)*x(i)
          fpara
          x(k) <-- x(k)/a(k,k)
     fpara
fin
```

Caso 3: El sistema es triangular superior. En este caso podemos resolver el sistema también por sustitución, como en el caso anterior, empezando por calcular el valor de la incógnita x_n y terminando por x_1.

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n-1}x_{n-1} + a_{1,n}x_n &= b_1 \\ a_{2,2}x_2 + \ldots + a_{2,n-1}x_{n-1} + a_{2,n}x_n &= b_2 \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\qquad\qquad\qquad \vdots & \\ a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n &= b_{n-1} \\ a_{n,n}x_n &= b_n \end{aligned}$$

En donde la ecuación que nos calcula el valor de la incógnita x_k en función de los valores de las incógnitas $x_{k+1}, x_{k+2}, \ldots, x_n$ es:

$$x_k = (b_k - a_{k,n}x_n - \ldots - a_{k,k+1}x_{k+1})/a_{k,k}$$

El algoritmo que realiza este proceso lo podemos estructurar como una acción no elemental, a partir de una tabla "a", matriz de coeficientes de nxn, y de una tabla "b", matriz de términos independientes de nx1.

```
acción resolver sistema triangular superior
inicio
     para k=n hasta 1 incremento -1 hacer
          x(k) <-- b(k)
          para i=n hasta k+1 incremento -1 hacer
               x(k) <-- x(k) - a(k,i)*x(i)
          fpara
          x(k) <-- x(k)/a(k,k)
     fpara
fin
```

11.2.- METODO DE GAUSS.

El mmétodo de Gauss consiste en transformar la matriz ampliada del sistema, mediante las adecuadas transformaciones elementales de fila, hasta consequir una matriz con todos sus elementos situados por debajo de la diagonal principal nulos, es decir una matriz tianqular superior. En síntesis el método consiste en aplicar reiteradamente el proceso de eliminación a cada columna de la matriz. Para realizarlo se elige un elemento de la matriz, al que se denomina pivote, supondremos para abreviar que en cada paso del proceso el pivote está situado siempre en la posición $a_{i,i}$ para efectuar la eliminación de los elementos situados en la columna i por debajo del pivote. Veamos el proceso que se sigue en la eliminación de los elementos de la primera columna:

$$A=\left[\begin{array}{cccccc:c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & : & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} & : & b_2 \\ & & & & & : & \\ & & & & & : & \\ & & & & & : & \\ a_{n,1} & \cdots & \cdots & a_{n,n-1} & a_{n,n} & : & b_n \end{array}\right] =$$

tomando como pivote el elemento $a_{1,1}$ y efectuando las operaciones:

$$\text{fila } i - (a_{1,1}/a_{1,1}) * \text{fila } 1$$
$$b_1 - (a_{1,1}/a_{1,1}) * b_1$$

para todos los índices i desde 2 hasta n, obtenemos una matriz equivalente con la primera y que tiene la forma:

$$A=\left[\begin{array}{cccccc:c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} & : & b_1 \\ 0 & c_{2,2} & c_{2,3} & \cdots & c_{2,n} & : & d_2 \\ & & & & & : & \\ & & & & & : & \\ & & & & & : & \\ 0 & \cdots & \cdots & c_{n,n-1} & c_{n,n} & : & d_n \end{array}\right] =$$

Dada una matriz de dimensión nxm podemos expresar el proceso como una iteración considerando submatrices a las que tenemos que aplicar en cada paso el mismo tratamiento, estas submatrices vendran determinadas por una fila (i) y una columna (j) que inicialmente seran la fila 1 y la columna 1.

El tratamiento a realizar dada una submatriz i,j es el siquiente:

- Encontrar pivote, elegimos como pivote el primer elemento de la submatriz distinto de 0. Para ello procederemos buscando por columnas hasta encontrar un elemento no nulo que determinaremos por su indice de fila (f) y su indice de columna (c). Si este elemento no existe la submatriz está formada unicamente por ceros y por tanto hemos acabado el proceso. En caso contrario tendremos que efectuar los siquientes pasos:

- Si el pivote (a(f,c)) no está en la fila i tenemos que intercambiar la fila i con la fila f. Con lo que el pivote pasara a ocupar la posición (i,c).

- poner ceros por debajo de la fila i en la columna c

Una vez realizado el tratamiento la nueva submatriz queda determinada por la fila i+1 (nueva i) y la columna c+1 (nueva j).

Repetiremos el tratamiento hasta que i sea mayor que n o j mayor que m.

```
acción Gauss
var n,m: entero
      a:tabla(n,m) de real
inicio
     i <-- 1
     j <-- 1
     iterar
          buscar pivote          ¦ calculara la fila y columna en la
                                 ¦ que está el pivote.

          salir si c>m           ¦ no existe pivote
          si i<>c
          entonces
               intercambiar filas(i,f)
          fsi
          tratar columna
          j <-- c+1
          salir si j>m           ¦ todas las columnas tratadas
          i <-- i+1
          salir si i>n           ¦ todas las filas tratadas
     fiterar
facción
```

Buscarpivote: Esta acción calculara la fila y la columna que corresponde al primer elemento no nulo de la matriz.

```
acción buscar pivote(j)
inicio
     c <-- j
     f <-- i
     mientras (a(f,c)=0 y c<= m ) hacer
          f <-- f+1
          si f>n
          entonces
               c <-- c+1
               f <-- i
          fsi
     fmientras
facción
```

Intercambiar filas: Utilizaremos una variable auxiliar para efectuar el intercambio.

```
acción intercambiar filas(i,i)
var   q: real          ¦ auxiliar para realizar el intercambio
inicio
```

```
     para k=1 hasta m hacer
          q <-- a(i,k)
          a(i,k) <-- a(f,k)
          a(f,k) <-- q
     fpara
facción
```

Tratar columna: Esta acción reduce a cero los elementos situados por debajo de la diagonal principal.

```
acción tratar columna
var q:real
inicio

     para k=i+1 hasta n hacer
          q <-- a(k,c)/a(i,c)

          para l=c hasta m hacer
               a(k,l) <-- a(k,l) - p*a(i,l)
          fpara
     fpara
facción
```

Este método solo puede llevarse a cabo si en el paso "k-ésimo" el elemento que ocupa la posición (k,k) es distinto de cero. Teniendo en cuenta que el orden de las ecuaciones y el orden de las incognitas dentro de las ecuaciones es irrelevante, es posible que ocurra el hecho de que con la disposición inicial no se pueda realizar el proceso pero que si pueda hacerse con otra reordenación, a este proceso de reordenación se le llama de pivotage.

Si la matriz A es regular entonces resuslta que siempre existe una determinada reordenación con la que poder realizar este proceso. De todas las reordenaciones posibles algunas son preferibles a otras en función de la rapidez y perecisión de los resultados que se quieran obtener. La determinación de la reordenación más conveniente depende, en cada paso del proceso, del pivote.

```
10 REM ******************************************************
20 REM    METODO DE GAUSS MATRIZ nxm
30 REM ******************************************************
40 REMINICIO DEL ALGORITMO
50 INPUT "DIMENSIONES DE LA MATRIZ";N,M
60 DIM A(N,M)
70 REM LECTURA DE LA MATRIZ POR FILAS
80 PRINT "MATRIZ A POR FILAS"
90 FOR I=1 TO N
100       FOR J=1 TO M
110            INPUT A(I,J)
```

```
120         NEXT J
130 NEXT I
140 REM INICIO DEL PROCESO
150 I=1
160 J=1
170 REM ITERAR
180 REM BUSQUEDA DEL PIVOTE
190 C=J
200 F=I
210 WHILE (A(F,C)=0 AND C<=M)
220         F=F+1
230         IF F>N THEN C=C+1\F=I
240 NEXT
250 IF C>M THEN 450
260 IF I=F THEN 340
270 REM INTERCAMBIAR FILAS
280 FOR K=C TO M
290         AUX=A(F,K)
300         A(F,K)=A(I,K)
310         A(I,K)=AUX
320 NEXT K
330 REM TRATAR COLUMNA
340 FOR K=I+1 TO N
350         AUX=A(K,C)/A(I,C)
360         FOR L=C TO N
370               A(K,L)=A(K,L)-AUX*A(I,L)
380         NEXT L
390 NEXT K
400 I=I+1
410 IF I=N THEN 450
420 J=C+1
430 IF J>M THEN 450
440 GOTO 170
450 REM ESCRITURA DE LA MATRIZ FINAL
460 FOR I=1 TO N
470         FOR J=1 TO M
480               PRINT A(I,J);
490         NEXT J
500         PRINT
510 NEXT I
520 END
```

11.3.- METODO DE GAUSS-JORDAN

Este método es una modificación al método de Gauss, consistente en efectuar las necesarias transformaciones elementales de fila a la matriz ampliada del sistema hasta llegar a una matriz diagonal equivalente con la primera. Basicamente el algoritmo es parecido al de Gauss con la salvedad de que ahora el proceso de eliminación de los elemtos de la columna "i-ésima" se extiende tanto a los elementos situados por encima de la diagonal principal como a los situados por debajo.

```
acción gauss jordan
inicio
     leer a,b
     para i=1 hasta n hacer
          eliminar2(i)
     fpara
     para i=1 hasta n hacer
          x(i) <-- b(i)/a(i,i)
          escribir x(i)
     fpara
fin

acción eliminar2(i)
inicio
     para k=i-1 hasta 1 incremento -1 hacer
          m <-- a(k,i)/a(i,i)
          a(k,i) <-- 0
          para l=i+1 hasta n hacer
               a(k,l) <-- a(k,l) - m*a(i,l)
          fpara
          b(k) <-- b(k) - m*b(i)
     fpara
     para k=i+1 hasta n incremento +1 hacer
          m <-- a(k,i)/a(i,i)
          a(k,i) <-- 0
          para l=i+1 hasta n hacer
               a(k,l) <-- a(k,l) - m*a(i,l)
          fpara
          b(k) <-- b(k) - m*b(i)
     fpara
```

11.4.- DISCUSION DE UN SISTEMA DE n ECUACIONES CON m INCOGNITAS

El proceso a seguir es el mismo que el del mmétodo de Gauss. Consideraremos la matriz ampliada del sistema, una vez aplicado el proceso de Gauss, tenemos que calcular el rango de la matriz y el de la ampliada.

```
acción discusión sistema
inicio
     leer(n,m)                 ¦ dimensiones de la matriz ampliada
     leer matriz(a)            ¦ incluido el termino independiente
     Gauss
     calcular rango de A       ¦ en la variable rga
     calcular rango de A¦b     ¦ en la variable rgab
     si rga=rgab
     entonces
```

```
            si rga=m
            entonces
                 escribir("sistema compatible determinado")
            sino
                 escribir("sistema compatible indeterminado")
            fsi
      sino
            escribir("sistema incompatible")
      fsi
fin
```

Calcular rango de A: para calcular el rango de A es suficiente encontrar la fila que ocupa el primer elemento no nulo por encima de la diagonal, empezando la busqueda en la fila n, columna m-1. Una vez encontrado esta fila el rango sera valor.

```
acción calcular rango de A
inicio
      i<--n
      j<--m-1
      mientras (a(i,j)=0 y i>0) hacer
            j<--j-1

            si j<i
            entonces
                 i<--i-1
                 j<--m-1
            fsi
      fmientras
      rga <-- i
facción
```

Calcular rango de A¦b: para calcular el rango de la matriz ampliada, calcularemos la fila en la que se encuentra el primer elemento no nulo de la columna n, empezando por la fila n. Una vez obtenido este valor el rango de la matriz ampliada sera este valor si este número es mayor que el rango de la matriz o el rango de la matriz sin ampliar.

```
acción calcular rango de A¦b
inicio
      i<--n
      j<--m+1
      mientras (a(i,j)=0 y i>0) hacer
            i<--i-1
      fmientras
      rgab <-- i
      si rgab<rga
      entonces
            rgab<--rga
      fsi
facción
```

```
10 REM ******************************************************
20 REM    DISCUSION SISTEMA DE ECUACIONES
30 REM    MATRIZ AMPLIADA DIMENSIONES nxm
40 REM ******************************************************
50 REM INICIO DEL ALGORITMO
60 INPUT "DIMENSIONES DE LA MATRIZ AMPLIADA";N,M
70 DIM A(N,M)
80 REM LECTURA DE LA MATRIZ POR FILAS
90 PRINT "MATRIZ A POR FILAS"
100 FOR I=1 TO N
110       FOR J=1 TO M-1
120            INPUT A(I,J)
130       NEXT J
140 NEXT I
150 REM LECTURA TERMINO INDEPENDIENTE
160 PRINT "TERMINO INDEPENDIENTE"
170 FOR I=1 TO N
180       INPUT A(I,M)
190 NEXT I
200 REM GAUSS
210 I=1
220 J=1
230 REM ITERAR
240 REM BUSQUEDA DEL PIVOTE
250 C=J
260 F=I
270 WHILE (A(F,C)=0 AND C<=M)
280       F=F+1
290       IF F>N THEN C=C+1\F=I
300 NEXT
310 IF C>M THEN 510
320 IF I=F THEN 400
330 REM INTERCAMBIAR FILAS
340 FOR K=C TO M
350       AUX=A(F,K)
360       A(F,K)=A(I,K)
370       A(I,K)=AUX
380 NEXT K
390 REM TRATAR COLUMNA
400 FOR K=I+1 TO N
410       AUX=A(K,C)/A(I,C)
420       FOR L=C TO N
430            A(K,L)=A(K,L)-AUX*A(I,L)
440       NEXT L
450 NEXT K
460 I=I+1
470 IF I=N THEN 510
480 J=C+1
490 IF J>M THEN 510
500 GOTO 230
510 REM ESCRITURA DE LA MATRIZ FINAL
```

```
520 FOR I=1 TO N
530       FOR J=1 TO M
540            PRINT A(I,J);
550       NEXT J
560       PRINT
570 NEXT I
580 REM CALCULO DEL RANGO DE LA MATRIZ DE COEFICIENTES
590 I=N
600 J=M-1
610 WHILE (A(I,J)=0 AND I>0)
620       J=J-1
630       IF J<I THEN I=I-1\J=M-1
640 NEXT
650 RGA=I
660 PRINT "RANGO MATRIZ DEL SISTEMA: ";RGA
670 REM CALCULO DEL RANGO DE LA MATRIZ AMPLIADA
680 I=N
690 WHILE (A(I,M)=0 AND I>0)
700       I=I-1
710 NEXT
720 RGAB=I
730 IF RGAB<RGA THEN RGAB=RGA
740 PRINT "RANGO DE LA MATRIZ AMPLIADA: ";RGAB
750 REM DISCUSION
760 IF RGA<>RGAB THEN PRINT "SISTEMA INCOMPATIBLE"\GOTO 790
770 IF RGA=M-1 THEN PRINT "COMPATIBLE DETERMINADO"\GOTO 790
780 PRINT "SISTEMA COMPATIBLE INDETERMINADO":END
```

11.5.-RESOLUCION DE UN SISTEMA COMPATIBLE DETERMINADO CON n INCOGNITAS

A partir de la matriz ampliada del sistema (nx(n+1)) una vez efectuado el proceso de Gauss, si la matriz es regular, obtenemos una matriz triangular superior, por tanto un algoritmo que resuleve el problema es el siguiente;

```
acción resolución sistema
var       x:tabla(n) de real
          a:tabla(n,m) de real
        n,m: entero
inicio
     leer(n)
     m=n+1
     leer matriz(a)          ¦ incluido el termino independiente
     Gauss
     calcular rango de A     ¦ en la variable rga
     si rga=n
     entonces
          resolver sistema triangular superior
          escribir solución
     sino
```

```
            escribir("no es un sistema compatible determinado")
      fsi
fin
```

Resolver sistema triangular superior: Dado un sistema de este tipo podemos calcular el valor de x(i) a partir de los valores de x(i+1) ..... x(n) y los coeficientes de la matriz

$$x(i) = (a(i,n+1)-a(i+1)*x(i+1)-\ldots-a(n)*x(n))/a(i,i)$$

calcularemos la solución en el orden x(n), ... , x(1)

```
acción resolver sistema triangular superior
inicio
      para i=n hasta 1 incremento -1 hacer
            x(i) <-- a(i,n+1)
            para k=i+1 hasta n hacer
                  x(i)<-- x(i)-a(i,k)*x(k)
            fpara
            x(i)<-- x(i)/a(i,i)
      fpara
facción

acción escribir solución
inicio
      para i=1 hasta n hacer
            escribir("x",i,"=",x(i))
      fpara
fin
```

```
10 REM ******************************************************
20 REM RESOLUCION SISTEMA DE ECUACIONES
30 REM MATRIZ AMPLIADA DIMENSIONES nx(n+1)
40 REM ******************************************************
50 REM INICIO DEL ALGORITMO
60 INPUT "DIMENSIONES DE LA MATRIZ ";N
70 M=N+1
80 DIM A(N,M),X(N)
90 REM LECTURA DE LA MATRIZ POR FILAS
100 PRINT "MATRIZ A POR FILAS"
110 FOR I=1 TO N
120       FOR J=1 TO M-1
130             INPUT A(I,J)
140       NEXT J
150 NEXT I
160 REM LECTURA TERMINO INDEPENDIENTE
170 PRINT "TERMINO INDEPENDIENTE"
180 FOR I=1 TO N
190       INPUT A(I,M)
200 NEXT I
210 REM GAUSS
```

```
220 I=1
230 J=1
240 REM ITERAR
250 REM BUSQUEDA DEL PIVOTE
260 C=J
270 F=I
280 WHILE (A(F,C)=0 AND C<=M)
290       F=F+1
300       IF F>N THEN C=C+1\F=I
310 NEXT
320 IF C>M THEN 520
330 IF I=F THEN 410
340 REM INTERCAMBIAR FILAS
350 FOR K=C TO M
360       AUX=A(F,K)
370       A(F,K)=A(I,K)
380       A(I,K)=AUX
390 NEXT K
400 REM TRATAR COLUMNA
410 FOR K=I+1 TO N
420       AUX=A(K,C)/A(I,C)
430       FOR L=C TO N
440            A(K,L)=A(K,L)-AUX*A(I,L)
450       NEXT L
460 NEXT K
470 I=I+1
480 IF I=N THEN 520
490 J=C+1
500 IF J>M THEN 520
510 GOTO 240
520 REM ESCRITURA DE LA MATRIZ FINAL
530 FOR I=1 TO N
540       FOR J=1 TO M
550            PRINT A(I,J);
560       NEXT J
570       PRINT
580 NEXT I
590 REM CALCULO DEL RANGO DE LA MATRIZ DE COEFICIENTES
600 I=N
610 J=M-1
620 WHILE (A(I,J)=0 AND I>0)
630       J=J-1
640       IF J<I THEN I=I-1\J=M-1
650 NEXT
660 RGA=I
670 PRINT "RANGO MATRIZ DEL SISTEMA: ";RGA
680 IF RGA<>N THEN PRINT "SISTEMA INCOMPATIBLE"\GOTO 810
690 REM RESOLUCION SISTEMA TRIANGULAR SUPERIOR
700 FOR I=N TO 1 STEP -1
710       X(I)=A(I,N+1)
720       FOR J= I+1 TO N
730            X(I)=X(I)-A(I,J)*X(J)
```

```
740         NEXT J
750         X(I)=X(I)/A(I,I)
760 NEXT I
770 REM ESCRITURA SOLUCION
780 FOR I=1 TO N
790         PRINT "X";I;" = ";X(I)
800 NEXT I
810 END
```

11.6.- RESOLUCION DE UN SISTEMA DE n ECUACIONES CON m INCOGNITAS

Veamos como podemos resolver este tipo de sistemas en el caso en que sean compatibles. Una vez finalizado el proceso de Gauss tendremos una matriz con rga filas distintas de cero, aun cuando no esten ordenadas correctamente (caso SCI). Por tanto en primer lugar tendremos que reordenar las columnas para tener un conjunto de sistemas de rga ecuaciones con rga incognitas y como terminos independientes los de las variables independientes mas el termino independiente inicial, por ejemplo supongamos que la matriz despues del proceso de Gauss nos queda:

x1	x2	x3	x4	x5	x6	x7	x8	ti
0	1	1	1	0	0	0	0	1
0	0	0	1	1	1	0	0	1
0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

En este caso el rango es 3, si reordenamos la matriz obtenemos:

x2	x4	x6	ti	x1	x3	x5	x7	x8
1	1	0	1	0	1	0	0	0
0	1	1	1	0	0	1	0	0
0	0	1	1	0	0	0	1	1

La resolucion del sistema conlleva la resolucion de n-rga+1 sistemas de ecuaciones, que podemos resolver simultaneamente, aplicando el mmétodo de Gauss al triangulo superior, en el ejemplo:

x2	x4	x6	ti	x1	x3	x5	x7	x8
1	0	0	1	0	1	-1	1	1
0	1	0	0	0	0	1	-1	-1
0	0	1	1	0	0	0	1	1

De donde tendriamos:

```
x2 = 1+x3-x5+x7+x8
x4 = x5-x7-x8
x6 = 1+x7+x8
```

Por tanto un algoritmo para resolver el problema es:

```
acción resolución sistema
inicio
      leer(n,m)
      leer matriz(a)              ¦ incluido el termino independiente
      Gauss
      calcular rango de A  ¦ en la variable rqa
      calcular rango de A¦b¦ en la variable rqáb
      si rqa=rqab
      entonces
            si rqa=m
            entonces
                  escribir("sistema compatible determinado")
            sino
                  escribir("sistema compatible indeterminado")
            fsi
            reordenar matriz
            Gauss2
            escribir solución
      sino
            escribir("sistema incompatible")
      fsi
fin
```

Reordenar matriz: Al reordenar la matriz tenemos que mantener información en cada momento sobre la variable representada en cada columna, para ello utilizaremos una tabla (ord) tal que ord(i)=número de la variable a la que corresponde la columna i. Inicialmente la tabla tendra los valores 1....n.n+1.

```
acción reordenar matriz
var         aux1:tabla (n) de real
            ord:tabla(m) de real
inicio
      para i=1 hasta m hacer  ¦ ordenación inicial de variables
            ord(i) <-- i
      fpara
      para i=1 hasta rqa hacer
            mientras a(i,i)=0 hacer
                  para i=1 hasta n hacer
                        aux1(j) <-- a(j,i)
                  fpara
                  para k=i hasta m-1 hacer
                        para j=1 hasta n hacer
                              a(j,k) <-- a(i,k+1)
                        fpara
```

```
                    ord(k) <-- ord(k+1)
               fpara
               ord(m) <-- i
               para j=1 hasta n hacer
                    a(j,m) <-- aux1(j)
               fpara
          fmientras
     fpara
facción

acción Gauss2                    ¦ Analogo a Gauss pero solo
                                 ¦ triangulo superior.
inicio
     para i=2 hasta rga hacer
          para i=1 hasta j-1 hacer
               p <-- a(i,j)/a(j,j)
               para k=j hasta m+1 hacer
                    a(i,k) <-- a(i,k) - p*a(j,k)
               fpara
          fpara
     fpara
     para i=1 hasta rga hacer
          para j=rga+1 hasta m hacer
               a(i,j) <-- a(i,j)/a(i,i)
          fpara
          a(i,i) <-- 1
     fpara
facción

acción escribir solución
inicio
     para i=1 hasta rga hacer
          escribir ("x",ord(i),"=",a(i,rga+1))
          para k=rga+2 hasta m hacer
               escribir ("+ x",ord(k),"* (",a(i,k),")")
          fpara
     fpara
facción

10 REM ******************************************************
20 REM  RESOLUCION SISTEMA DE ECUACIONES
30 REM MATRIZ AMPLIADA DIMENSIONES nxm
40 REM ******************************************************
50 REM  INICIO DEL ALGORITMO
60 INPUT "DIMENSIONES DE LA MATRIZ AMPLIADA";N,M
70 DIM A(N,M),AUX1(N),ORD(M)
80 REM LECTURA DE LA MATRIZ POR FILAS
90 PRINT "MATRIZ A POR FILAS"
100 FOR I=1 TO N
110       FOR J=1 TO M-1
120            INPUT A(I,J)
130       NEXT J
```

```
140  NEXT I
150 REM LECTURA TERMINO INDEPENDIENTE
160 PRINT "TERMINO INDEPENDIENTE"
170 FOR I=1 TO N
180       INPUT A(I,M)
190 NEXT I
200 REM GAUSS
210 I=1
220 J=1
230 REM ITERAR
240 REM BUSQUEDA DEL PIVOTE
250 C=J
260 F=I
270 WHILE (A(F,C)=0 AND C<=M)
280       F=F+1
290       IF F>N THEN C=C+1\F=I
300 NEXT
310 IF C>M THEN 510
320 IF I=F THEN 400
330 REM INTERCAMBIAR FILAS
340 FOR K=C TO M
350       AUX=A(F,K)
360       A(F,K)=A(I,K)
370       A(I,K)=AUX
380 NEXT K
390 REM TRATAR COLUMNA
400 FOR K=I+1 TO N
410       AUX=A(K,C)/A(I,C)
420       FOR L=C TO N
430            A(K,L)=A(K,L)-AUX*A(I,L)
440       NEXT L
450 NEXT K
460 I=I+1
470 IF I=N THEN 510
480 J=C+1
490 IF J>M THEN 510
500 GOTO 230
510 REM ESCRITURA DE LA MATRIZ FINAL
520 FOR I=1 TO N
530       FOR J=1 TO M
540            PRINT A(I,J);
550       NEXT J
560       PRINT
570 NEXT I
580 REM CALCULO DEL RANGO DE LA MATRIZ DE COEFICIENTES
590 I=N
600 J=M-1
610 WHILE (A(I,J)=0 AND I>0)
620       J=J-1
630       IF J<I THEN I=I-1\J=M-1
640 NEXT
650 RGA=I
```

```
660 PRINT "RANGO MATRIZ DEL SISTEMA: ";RGA
670 REM CALCULO DEL RANGO DE LA MATRIZ AMPLIADA
680 I=N
690 WHILE (A(I,M)=0 AND I>0)
700       I=I-1
710 NEXT
720 RGAB=I
730 IF RGAB<RGA THEN RGAB=RGA
740 PRINT "RANGO DE LA MATRIZ AMPLIADA: ";RGAB
750 REM DISCUSION
760 IF RGA<>RGAB THEN PRINT "SISTEMA INCOMPATIBLE"\GOTO 1220
770 IF RGA=M-1 THEN PRINT"SISTEMA COMPATIBLE DETERMINADO"\GOTO790
780 PRINT "SISTEMA COMPATIBLE INDETERMINADO"
790 REM REORDENAR MATRIZ
800 FOR I=1 TO M
810       ORD(I)=I
820 NEXT I
830 FOR I=1 TO RGA
840       WHILE A(I,I)=0
850             FOR J=1 TO N
860                   AUX1(J)=A(J,I)
870             NEXT J
880             FOR K=I TO M-1
890                   FOR J=1 TO N
900                         A(J,K)=A(J,K+1)
910                   NEXT J
920                   ORD(K)=ORD(K+1)
930             NEXT K
940             ORD(M)=I
950             FOR J=1 TO N
960                   A(J,M)=AUX1(J)
970             NEXT J
980       NEXT
990 NEXT I
1000 REM GAUSS 2
1010 FOR J=2 TO RGA
1020       FOR I=1 TO J-1
1030             AUX=A(I,J)/A(J,J)
1040             FOR K=J TO M
1050                   A(I,K)=A(I,K)-AUX*A(J,K)
1060             NEXT K
1070       NEXT I
1080 NEXT J
1090 FOR I=1 TO RGA
1100       FOR J=RGA+1 TO M
1110             A(I,J)=A(I,J)/A(I,I)
1120       NEXT J
1130 NEXT I
1140 REM ESCRITURA SOLUCION
1150 FOR I=1 TO RGA
1160       PRINT "X";ORD(I);" = ";A(I,RGA+1);
1170       FOR K=RGA+2 TO M
```

```
1180            PRINT " + X";ORD(K);" * (";A(I,K);"
1190        NEXT K
1200        PRINT
1210    NEXT I
1220    END
```

11.6.- DESCOMPOSICION LU.

Este método consiste en descomponer la matriz asociada al sistema de ecuaciones

$$A=\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & & & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & & & a_{1,2} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & & & a_{1,3} \\ & & & & & & \\ a_{i,1} & \cdots & a_{i,i-1} & a_{i,i} & a_{i,i+1} & \cdots & a_{i,n} \\ & & & & & & \\ a_{n,1} & \cdots & & \cdots & a_{n,n-1} & & a_{n,n} \end{bmatrix} =$$

en producto de dos matrices, una triangular inferior L y otra triangular superior U, es decir este método da la manera de poder descomponer A=LU.

$$\begin{bmatrix} \alpha_{1,1} & 0 & \cdots & 0 \\ \alpha_{2,1} & \alpha_{2,2} & 0 & \cdots & 0 \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & 0 & \cdots & 0 \\ & & & & & \\ \alpha_{n,1} & \cdots & \alpha_{n,n-1} & \alpha_{n,n} \end{bmatrix} \begin{bmatrix} \tau_{1,1} & \tau_{1,2} & \tau_{1,3} & \cdots & \tau_{1,n} \\ 0 & \tau_{2,2} & \tau_{2,3} & \cdots & \tau_{2,n} \\ 0 & 0 & \tau_{3,3} & \cdots & \tau_{3,n} \\ & & & & \\ 0 & & \cdots & 0 & \tau_{n,n} \end{bmatrix}$$

de manera que resolver el sistema Ax=T se reduce a resolver sucesivamente el sistema triangular inferior Ly=T y el sistema triangular superior Ux=y.

Debemos observar que en este método la descomposición que se hace de la matriz asociada al sistema es independiente de la matriz de los términos independientes. Por tanto este método será valido para resolver aquellos sistemas que tengan la misma matriz asociada y distintos términos independientes.

Esta descomposición nos lleva al siguiente sistema de equaciones:

$$a_{1,1} = \alpha_{1,1}\ \tau_{1,1}$$
$$a_{1,j} = \alpha_{1,1}\ \tau_{1,j} \quad \text{para } j=1,2,\ldots,n$$
$$a_{i,1} = \alpha_{i,1}\ \tau_{1,1} \quad \text{para } i=1,2,\ldots,n$$

y para k=1,2,...,n tenemos los siguientes tipos de equaciones

$$a_{k,k} = \sum_{p=1}^{k} \alpha_{k,p}\ \tau_{p,k}$$

$$a_{k,j} = \sum_{p=1}^{k} \alpha_{k,p}\ \tau_{p,j}$$

$$a_{i,k} = \sum_{p=1}^{k} \alpha_{i,p}\ \tau_{p,k}$$

sistema del que hay que determinar los coeficientes $\alpha_{i,j}$, $\tau_{i,j}$ para i,j=1,2,...,n. Despejando se obtiene:

$$\alpha_{1,1}\ \tau_{1,1} = a_{1,1}$$
$$\alpha_{i,1} = a_{i,1}/\tau_{1,1} \quad \text{para } i=1,2,\ldots,n$$
$$\tau_{1,j} = a_{1,j}/\alpha_{1,1} \quad \text{para } j=1,2,\ldots,n$$

y para k=1,2,...,n tenemos las siguientes relaciones:

$$\alpha k,k\ \tau_{k,k} = a_{k,k} - \sum_{p=1}^{k-1} \alpha_{k,p}\ \tau_{p,k}$$

$$\tau_{k,j} = (a_{k,1} - \sum_{p=1}^{k-1} \alpha_{k,p}\ \tau_{p,j})/\alpha_{k,k} \quad \text{para } j=k+1,\ldots,n$$

$$\alpha_{i,k} = (a_{i,k} - \sum_{p=1}^{k-1} \alpha_{i,p}\ \tau_{p,k})/\tau k,k \quad \text{para } i=k+1,\ldots,n$$

Observación: En todo este esquema de calculo quedan patentes dos hechos fundamentales que determinan los distintos métodos que existen:

- En primer lugar se calculan los elementos situados en la diagonal principal, es evidente la multiplicidad de elecciones a hacer, lo que da lugar a los distintos métodos, como ya apuntábamos anteriormente.

- En segundo lugar se calculan los coeficientes de la fila "k-ésima de U" y posteriormente los coeficientes de la columna "k-ésima" de L y .

Estudiaremos en particular dos métodos, Mmétodo de Crout y Método de Cholesky, que utilizan este tipo de descomposición.

11.6.1.- METODO DE CROUT.

Este método consiste en la descomposición LU, de la matriz asociada al sistema de equaciones, en donde se elige $\alpha_{i,i}=1$ para $i=1,2,\ldots,n$. Quedando entonces las siguientes relaciones:

$$\tau_{1,1} = a_{1,1}$$
$$\alpha_{1,1} = a_{1,1}/\tau_{1,1} \quad \text{para } i=1,2,\ldots,n$$
$$\tau_{1,1} = a_{1,j} \text{ para } j=1,2,\ldots,n$$

y para $k=1.2,\ldots,n$ tenemos las siguientes relaciones:

$$\tau k,k = a_{k,k} - \sum_{p=1}^{k-1} \alpha_{k,p}\ \tau_{p,k}$$

$$\tau_{k,j} = a_{k,j} - \sum_{p=1}^{k-1} \alpha_{k,p}\ \tau_{p,j} \qquad \text{para } j=k+1,\ldots,n$$

$$\alpha_{i,k} = (a_{i,k} - \sum_{p=1}^{k-1} \alpha_{i,p}\ \tau_{p,k})/\tau k,k \qquad \text{para } i=k+1,\ldots,n$$

```
acción crout
inicio
     para k=1 hasta n hacer
          α(k,k) <-- 1
          τl(k,k) <-- a(k,k)
          para p=1 hasta k-1 hacer
               τ(k,k) <-- τ(k,k)-α(k,p)*τ(p,k)
          fpara
          para i=k+1 hasta n hacer
               α(i,k) <-- a(i,k)
               para p=1 hasta k-1 hacer
                    α(i,k) <-- α(i,k)-α(i,p)*τ(p,k)
               fpara
          α(i,k) <-- α(i,k)/τ(k,k)
          fpara
          para j=k+1 hasta n hacer
               τ(k,j) <-- a(k,j)
               para p=1 hasta k-1 hacer
                    τ(k,j) <-- τ(k,i)-α(k,p)*τ(p,i)
               fpara
          fpara
     fpara
fin
```

11.6.2.- METODO DE CHOLESKY.

Este es un método particular de descomposición LU para sistemas de ecuaciones lineales cuya matriz asociada es una matriz simétrica, en cuyo caso la matriz asociada A descompone como $A=LL^T$. Es evidente que este método solo precisa del cálculo de los coeficientes de una de las matrices puesto que la otra es su traspuesta.

Las relaciones que obtenemos al aplicar la descomposición son:

$$\alpha^2_{1,1} = a_{1,1}$$

$$\alpha_{i,1} = a_{i,1}/\alpha_{1,2} \quad \text{para } i=1,2,\ldots,n$$

$$\alpha^2_{k,k} = \alpha_{k,k} - \sum_{p=1}^{k-1} \alpha^2_{k,p} \quad \text{para } k=1,2,\ldots,n$$

$$\alpha_{i,k} = (a_{i,k} - \sum_{p=1}^{k-1} \alpha_{i,p}\ \alpha_{k,p})/\alpha_{k,k} \quad \text{para } i=k+1,\ldots,n$$

```
acción cholesky
inicio
     para k=1 hasta n hacer
          α(k,k) <-- a(k,k)
          para p=1 hasta k-1 hacer
               α(k,k) <-- α(k,k)-α(k,p)*α(k,p)
          fpara
          α(k,k) <-- √ 1(k,k)
          para i=k+1 hasta n hacer
               α(i,k) <-- a(i,k)
               para p=1 hasta k-1 hacer
                    α(i,k) <-- α(i,k)-α(i,p)*α(k,p)
               fpara
               α(i,k) <-- α(i,k)/α(k,k)
          fpara
     fpara
fin
```

12.- RESOLUCION DE SISTEMAS DE ECUACIONES METODOS ITERATIVOS

12.1.- Métodos iterativos.

Los métodos iterativos de cálculo de las soluciones de un sistema de ecuaciones, se basan en el cálculo de una sucesión de aproximaciones de la solución del sistema. La solución así calculada tiene por límite la solución buscada.

El método de construcción de la solución es siempre el mismo, dado el sistema Ax=b, en el que la matriz asociada es regular, consiste básicamente en los siguientes pasos:

i) Se descompone la matriz A=N-P, donde N y P son dos matrices del mismo orden que A, de manera que el sistema puede escribirse de la forma Nx=Px+b

ii) A partir de un valor inicial x_0 se construye una sucesión $(x_n)_{n\in N}$ mediante la recurrencia:

$$Nx_1 = Px_0 + b$$
$$\vdots$$
$$Nx_n = Px_{n-1} + b$$

iii) Si la sucesión es convergente, es decir si $x_n \dashrightarrow x$, entonces

$$Nx = Px + b \iff Ax = b$$

Conviene observar que el cálculo de x_n implica la resolución del sistema $Nx_n=c$, por lo tanto es muy importante elegir la mejor descomposición de A=N-P de manera que este sistema sea facilmente resoluble. Por otro lado es evidente que dependiendo de como se descomponga A tendremos un método u otro.

Tambien hay que observar que en todo método iterativo se ha de controlar en todo momento su convergencia asi como el numero de iteraciones que se efectuan.

Estudiaremos concretamente dos métodos iterativos: el Método de Jacobi y el Método de Gauss-Seidel.

12.2- Método de Jacobi.

El método de Jacobi es una aplicación de esta técnica general en la que se toma la siguiente descomposición de la matriz del sistema:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ & & & \\ a_{n,1} & & \cdots & a_{n,n} \end{bmatrix} =$$

$$\begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ & & & \\ 0 & 0 & \cdots & a_{n,n} \end{bmatrix} + \begin{bmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & 0 & \cdots & a_{2,n} \\ & & & \\ a_{n,1} & & \cdots & 0 \end{bmatrix}$$

En la "k-ésima" iteración la solución aproximada es:

$$a_{1,1}x_1^k = b_1 - (a_{1,2}x_2^{k-1} + a_{1,3}x_3^{k-1} + \ldots + a_{1,n}x_n^{k-1})$$

$$\vdots$$

$$a_{i,i}x_i^k = b_i - (a_{i,2}x_2^{k-1} + \ldots + a_{i,i-1}x_{i-1}^{k-1} + a_{i,i+1}x_{i+1}^{k-1} + \ldots + a_{i,n}x_n^{k-1})$$

$$\vdots$$

$$a_{n,n}x_n^k = b_n - (a_{n,1}x_1^{k-1} + \ldots + a_{n,n-1}x_{n-1}^{k-1})$$

que escrita en forma compacta es:

$$x_i^k = \Big(b_i - \sum_{\substack{j=1 \\ j<>i}}^{n} a_{i,j}\, x_i^{k-1} \Big) / a_{i,i} \qquad \text{para } i=1,2,\ldots,n$$

```
acción jacobi
var
                  a : tabla (n,n) de real
          b,x0,x1 : tabla (n) de real
            eps,z : real
     max,i,j,n,nit : entero
inicio
     leer a,x0.max,eps
     nit <-- 0
     repetir
          para i=1 hasta n hacer
```

```
                x1(i) <-- b(i)
                para j=1 hasta i-1 hacer
                     x1(i) <-- x1(i) - a(i,j)*xO(j)
                fpara
                para i=i+1 hasta n hacer
                     x1(i) <-- x1(i) - a(i,j)*xO(j)
                fpara
                x1(i) <-- x1(i)/a(i,i)
           fpara
           z <-- norma(xO-x1)
           nit <-- nit + 1
           xo <-- x1
      hasta que ( eps<O o nit>max)
      escribir ("número de iteraciones";nit)
      escribir ("solución: ";xO)
      escribir ("norma:";z)
fin
```

12.3.- Método de Gauss-Seidel

En este método la descomposición de la matriz del sistema es de la siguiente manera:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ & & & \\ a_{n,1} & & \cdots & a_{n,n} \end{bmatrix} =$$

$$\begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & 0 \\ & & & \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} + \begin{bmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ 0 & 0 & \cdots & a_{2,n} \\ & & & a_{n-1,n} \\ 0 & & \cdots & 0 \end{bmatrix}$$

En la "k-ésima" iteración la solución aproximada es:

$$a_{1,1}x_1^k = b_1 - (a_{1,2}x_2^{k-1} + a_{1,3}x_3^{k-1} + \ldots + a_{1,n}x_n^{k-1})$$

$$\vdots$$

$$a_{i,1}x_1^k + \ldots + a_{i,i}x_i^k = b_i - (a_{i,i+1}x_{i+1}^{k-1} + \ldots + a_{i,n}x_n^{k-1})$$

$$\vdots$$

$$a_{n,1}x_1^k + \ldots + a_{n,n}x_n^k = b_n$$

que escrita en forma compacta es:

$$x_i^k = \left(b_i - \sum_{j=1}^{i-1} a_{i,j}\, x_j^k - \sum_{j=i+1}^{n} a_{i,j} x_j^{k-1} \right) / a_{i,i}$$

para i=1,2....,n

```
acción gauss seidel
var
                  a : tabla (n,n) de real
            b,x0,x1 : tabla (n) de real
              eps,z : real
      max,i,j,n,nit : entero
inicio
      leer a,x0,max,eps
      nit <-- 0
      repetir
            para i=1 hasta n hacer
                  x1(i) <-- b(i)
                  para j=1 hasta i-1 hacer
                        x1(i) <-- x1(i) - a(i,j)*x1(j)
                  fpara
                  para j=i+1 hasta n hacer
                        x1(i) <-- x1(i) - a(i,j)*x0(j)
                  fpara
                  x1(i) <-- x1(i)/a(i,i)
            fpara
            z <-- norma(x0-x1)
            nit <-- nit + 1
            xo <-- x1
      hasta que ( eps<0 o nit>max)
      escribir ("número de iteraciones";nit)
      escribir ("solución: ";x0)
      escribir ("norma:";z)
fin
```

Observemos cual es la diferencia fundamental entre ambos métodos. En el método de Jacobi la solución aproximada en la "k-ésima" iteración se calcula mediante la solución aproximada de la iteración anterior, mientras que en el método de Gauss-Seidel la componente "i-ésima" de la solución aproximada en la "k-ésima" iteración se calcula con las soluciones aproximadas obtenidas en la iteración anterior junto con las "i-1" primeras soluciones obtenidas en esta misma iteración.

13.- CALCULO DE LA MATRIZ INVERSA. METODO DE GAUSS

Para calcular la matriz inversa nos basta con utilizar el mmétodo de Gauss para la matriz ampliada por la derecha con una matriz identidad, efectuando los pasos necesarios hasta obtener una matriz diagonal. Si el rango de la matriz no es n, la matriz no es regular y por tanto no tiene inversa.

```
acción inversa
inicio
     inicialización
     Gauss
     Calcular rango de A
     si rga=n
     entonces
          Gauss2
          escribir resultados
     sino
          escribir ("la matriz no es regular")
     fsi
fin
```

```
10 REM ******************************************************
20 REM  CALCULO DE LA MATRIZ INVERSA
30 REM (METODO DE GAUSS CON M=2*N)
40 REM ******************************************************
50 REM  INICIO DEL ALGORITMO
60 INPUT "DIMENSIONES DE LA MATRIZ ";N
70 M=2*N
80 DIM A(N,M)
90 REM LECTURA DE LA MATRIZ POR FILAS
100 PRINT "MATRIZ A POR FILAS"
110 FOR I=1 TO N
120       FOR J=1 TO N
130            INPUT A(I,J)
140       NEXT J
150 NEXT I
160 REM AMPLIACION DE LA MATRIZ
170 FOR I=1 TO N
180       FOR J=N+1 TO M
190            A(I,J)=0
200       NEXT J
210       A(I,N+I)=1
220 NEXT I
230 REM GAUSS
240 I=1
250 J=1
260 REM ITERAR
270 REM BUSQUEDA DEL PIVOTE
280 C=J
290 F=I
```

```
300 WHILE (A(F,C)=0 AND C<=M)
310       F=F+1
320       IF F>N THEN C=C+1\F=I
330 NEXT
340 IF C>M THEN 540
350 IF I=F THEN 430
360 REM INTERCAMBIAR FILAS
370 FOR K=C TO M
380       AUX=A(F,K)
390       A(F,K)=A(I,K)
400       A(I,K)=AUX
410 NEXT K
420 REM TRATAR COLUMNA
430 FOR K=I+1 TO N
440       AUX=A(K,C)/A(I,C)
450       FOR L=C TO N
460             A(K,L)=A(K,L)-AUX*A(I,L)
470       NEXT L
480 NEXT K
490 I=I+1
500 IF I=N THEN 540
510 J=C+1
520 IF J>M THEN 540
530 GOTO 260
540 REM CALCULO DEL RANGO DE LA MATRIZ
550 I=N
560 J=M-1
570 WHILE (A(I,J)=0 AND I>0)
580       J=J-1
590       IF J<I THEN I=I-1\J=M-1
600 NEXT
610 RGA=I
620 IF RGA<>N THEN PRINT "MATRIZ NO REGULAR"\GOTO 840
630 REM GAUSS 2
640 FOR J=2 TO RGA
650       FOR I=1 TO J-1
660             AUX=A(I,J)/A(J,J)
670             FOR K=J TO M
680                   A(I,K)=A(I,K)-AUX*A(J,K)
690             NEXT K
700       NEXT I
710 NEXT J
720 FOR I=1 TO RGA
730       FOR J=RGA+1 TO M
740             A(I,J)=A(I,J)/A(I,I)
750       NEXT J
760 NEXT I
770 REM ESCRITURA MATRIZ INVERSA
780 FOR I=1 TO N
790       FOR J=N+1 TO M
800             PRINT A(I,J);
810       NEXT J
```

```
820         PRINT
830 NEXT I
840 END
```

14.- CALCULO DE DETERMINANTES, METODO DE GAUSS

Para calcular el determinante de una matriz, utilizando el mmétodo de Gauss, unicamente tenemos que tener en cuenta que cada vez que intercambiamos dos filas el determinante cambia de signo; y que el determinante de una matriz trianoular superior es el producto de los elementos de la diagonal.

```
acción determinante
inicio
     sign<--1
     j <-- 1
     iterar
          buscar pivote(j)     ¦ calculara la fila y columna en la
                               ¦ que está el pivote.
          salir si j>n         ¦ no existe pivote
          si i<>j
          entonces
               intercambiar_filas(i,j)
               sign <-- -sign
          fsi
          tratar columna
          j <-- j+1
          salir si j>n              ¦ todas las columnas tratadas
     fiterar
     det<-- sign
     para i=1 hasta n hacer         ¦ calculo del determinante
          det<-- det*a(i,i)
     fpara
facción
```

```
10 REM ******************************************************
20 REM  CALCULO DEL DETERMINANTE
30 REM  METODO DE GAUSS MATRIZ NxN
40 REM ******************************************************
50 REM  INICIO DEL ALGORITMO
60 INPUT "DIMENSIONES DE LA MATRIZ";N
70 DIM A(N,N)
80 M=N
90 REM LECTURA DE LA MATRIZ POR FILAS
100 PRINT "MATRIZ A POR FILAS"
110 FOR I=1 TO N
120       FOR J=1 TO M
130            INPUT A(I,J)
140       NEXT J
150 NEXT I
160 REM INICIO DEL PROCESO
170 SIGN=1
180 I=1
190 J=1
200 REM ITERAR
```

```
210 REM BUSQUEDA DEL PIVOTE
220 C=J
230 F=I
240 WHILE (A(F,C)=0 AND C<=M)
250       F=F+1
260       IF F>N THEN C=C+1\F=I
270 NEXT
280 IF C>M THEN 490
290 IF I=F THEN 380
300 REM INTERCAMBIAR FILAS
310 FOR K=C TO M
320       AUX=A(F,K)
330       A(F,K)=A(I,K)
340       A(I,K)=AUX
350 NEXT K
360 SIGN= - SIGN
370 REM TRATAR COLUMNA
380 FOR K=I+1 TO N
390       AUX=A(K,C)/A(I,C)
400       FOR L=C TO N
410            A(K,L)=A(K,L)-AUX*A(I,L)
420       NEXT L
430 NEXT K
440 I=I+1
450 IF I=N THEN 490
460 J=C+1
470 IF J>M THEN 490
480 GOTO 200
490 REM CALCULO DEL DETERMINANT
500 DT=SIGN
510 FOR I=1 TO N
520       DT=DT*A(I,I)
530 NEXT I
540 PRINT "DETERMINANTE :";DT
550 END
```