1. INTRODUCCIÓN A RPN

En esta materia los programas serán elaborados para la calculadora HP (en sus modelos 48 y 49) pues es la calculadora que más ampliamente se emplea en las diferentes carreras de la Facultad de Tecnología.

Esta calculadora emplea la notación RPN (Notación Polaca Inversa) siendo sus lenguajes de programación ***HP Basic*** (o modo Algebraico), ***User RPN***, ***System RPN*** y el ***lenguaje de máquina***. La facilidad con la que se puede elaborar un programa, al menos en primera instancia, sigue este mismo orden, es decir es más fácil elaborar un programa en ***HP Basic***, le sigue ***User RPN***, luego ***System RPN*** y finalmente el ***lenguaje de máquina***, sin embargo, la eficiencia y velocidad de un programa sigue el orden inverso, es decir es mucho más eficiente y rápido un programa elaborado en lenguaje de máquina que uno elaborado en ***HP Basic***.

Puesto que la eficiencia (uso de memoria) y la velocidad, son factores muy importantes cuando se trabaja con los limitados recursos de una calculadora, no resulta práctico elaborar programas en ***HP Basic (modo algebraico)*** pues consumen demasiada memoria y genera programas muy lentos. Por otra parte el ***lenguaje de máquina*** tampoco es muy práctico pues elaborar programas en el mismo requiere mucho tiempo.

Por estas razones los programas de uso frecuente serán elaborados en ***System RPN***, mientras que los programas de uso específico en ***User RPN*** (aunque en ocasiones también en ***System***).

1.1. User y System RPN

User RPN es el lenguaje que viene por defecto en la calculadora, por lo que puede ser programado directamente en la misma. Se trata de un subconjunto de ***System RPN*** y cuenta con alrededor de 800 comandos, mientras que ***System RPN*** cuenta con alrededor de 5000 comandos (incluidos los comandos de ***User***).

Pero si ***User*** es un subconjunto de ***System*** ¿Por qué es más lento? La respuesta se encuentra en la forma en que se ejecutan los comandos de ***User***. Todos los comandos de ***User*** son "seguros" pues antes de ejecutarse verifican que existan los datos y tipos de datos adecuados, de esta manera impiden la escritura de datos erróneos en memoria.

Por el contrario, los comandos de ***System*** no verifican nada y realizan las operaciones existan o no los datos adecuado. Al no perder tiempo en verificar la existencia de datos y tipos de datos adecuados, estos comandos se ejecutan con mayor rapidez. La verificación de los datos y tipos de datos queda en manos del programador, razón por la cual la programación en ***System RPN*** requiere de mayor cuidado por parte del programador, pues un error generalmente ocasiona un fallo del sistema y la pérdida de la información almacenada en memoria.

Puesto que ***System RPN*** no es el lenguaje por defecto de las calculadoras no puede ser programado directamente en las mismas. Para su programación se requieren de herramientas adicionales, las cuales son gratuitas y pueden ser bajadas de ***www.hpcalc.org***.

Para instalar cualquier librería en la calculadora, una vez bajada de ***hpcalc.org***, se la pasa a la calculadora (con el programa de conexión: HP connectivity kit, que también puede ser bajado de ***hpcalc.org***), entonces se almacena en uno de los puertos (0 STO, 1 STO o 2 STO) y se reinicia la calculadora (presionando ON+C) para que la librería sea instalada.

Para la HP48 todas las herramientas necesarias se encuentran en la librería *JAZZ*. Para que *JAZZ* funcione se requiere instalar también la librería de fuentes *UFL3* (que viene conjuntamente la librería *JAZZ*). Puesto que la HP48 no cuenta con un administrador de archivos (un *filer*) es recomendable instalar una de las librerías *FILER48*, la versión más completa *(F48F)* requiere alrededor de 19 kb y la más compacta *(F48U)* alrededor de 10 kb, con las versiones más completa es posible examinar todos los puertos de la calculadora.

Para la HP49 se requieren las librerías *EMACS*, *SDIAG* y *EXTABLE* (las cuales vienen conjuntamente *eMacs*), adicionalmente, en la HP49 es conveniente instalar las librerías *NOSY*, *OT49* y *KEYMAN*. Estas herramientas permiten elaborar o editar programas tanto en *System RPN* como en *lenguaje de máquina* (más propiamente en ensamblador).

La principal limitante de la HP48 es su memoria, pues sólo dispone de 128 kb y la librería JAZZ ocupa alrededor de 70 kb. Aunque es posible instalar una versión más limitada de JAZZ, de alrededor de 50 kb, sigue siendo una librería muy grande para la memoria disponible en la HP48.

Por otra parte la principal dificultad en la HP49 es su depurador (DBUG), el cual provoca fallos del sistema al depurar algunos comandos de *System* o *Ensamblador*, ello impide en la práctica la depuración de programas escritos en estos lenguajes. Para solucionar este problema se puede instalar la librería JAZZ en su versión HP49, pero esta librería sólo puede ser instalada en el puerto 0, lo que consume 70 kb de los 256 kb disponibles en el mismo, adicionalmente se requieren otros 40 kb para la tabla de direcciones de memoria (y 4 Kb para *UFL*), la cual sin embargo puede ser instalada en los puertos 1 o 2. No obstante, aún con JAZZ existen todavía algunas limitaciones pues no tiene soporte para los comandos *FLASH*, por lo que dichos comandos no pueden ser depurado paso a paso.

A pesar de las dificultades mencionadas, es conveniente contar con estas librerías (sobre todo en la HP49) principalmente para elaborar programas no muy largos o modificar programas existentes.

En general, sin embargo, es más cómodo y seguro elaborar y probar los programas en una computadora y pasarlos luego a la calculadora. Para ello existen varias herramientas siendo la más completa *DEBUG4X*, que también puede ser bajada de *hpcalc.org*. Emplearemos esta herramienta para elaborar y depurar los programas por lo que deberá bajarla e instalarla en una computadora. *DEBUG4X* incluye emuladores para los modelos HP48 y HP49 (incluido HP49G+), los cuales incorporan todas las funciones de las calculadoras reales lo que nos permite utilizarlas aún sin contar realmente con las mismas (es una buena idea instalar estos emuladores y emplearlos en lugar de la calculadora estándar de Windows).

1.2. Notación RPN

Como se dijo la calculadora HP emplea la Notación Polaca Inversa, este es probablemente el aspecto de la calculadora al que más cuesta acostumbrarse. En RPN se escribe primero él o los datos y luego el comando u operación que actúa sobre los mismos, por ejemplo para sumar 4.5 y 6.7 se escribe:

```
4.5 6.7 +
```

En lugar de 4.5+6.7 como ocurre con otras calculadoras (y lenguajes). Para calcular el seno de 45.6 escribimos:

```
45.6 SIN
```

En lugar de SIN(45.6). En RPN no existen paréntesis pues no se necesitan. Por ejemplo para calcular el valor de la siguiente expresión:

$$\sqrt{\frac{(3)(4.5)+(7.2)(8^{2.2})}{6.2+9.5e^{4.65}}}$$

Escribimos:

3 4.5 * 7.2 8 2.2 ^ * + 6.2 9.5 4.65 EXP * + / √

En lugar de: SQRT((3*4.5+7.2*8^2.2)/(6.2+9.5*EXP(4.65)), como ocurre en otras calculadoras y lenguajes.

Entonces siempre que se escribe una expresión matemática en RPN se deben tomar en cuenta dos aspectos: a) Escribir él o los datos antes del comando u operación y b) No emplear paréntesis (pues no existen).

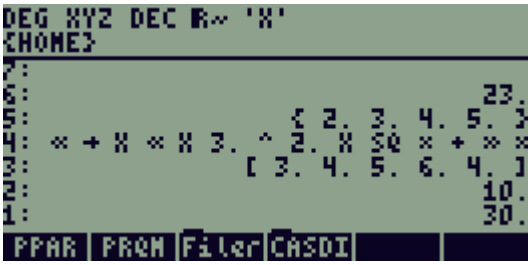
La única forma de acostumbrarse a la notación RPN es la práctica, mientras más se emplee la calculadora (o su emulador) en la resolución de problemas más fácil y natural resulta. Con la práctica la notación RPN no sólo resulta tan natural como la notación directa sino que en muchos aspectos es inclusive más sencilla.

1.3. La pila (Stack)

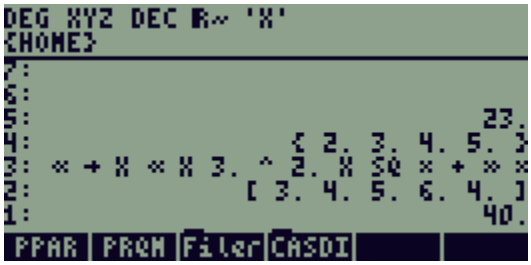
Otro aspecto que siempre se debe tener en mente al trabajar con la calculadora HP es la ***pila***. La pila es el sector de memoria donde se almacena temporalmente la información. Visualmente la pila se muestra en el sector medio de la pantalla y tiene números al lado izquierdo, tal como se muestra en la figura:

Se denomina pila (o montón) porque la información que se guarda en la misma se va apilando (o amontando). El último dato guardado es siempre el número uno y este es también el primer dato que se puede sacar de la pila, se trata entonces de una pila ***LIFO*** (Last In First Out).

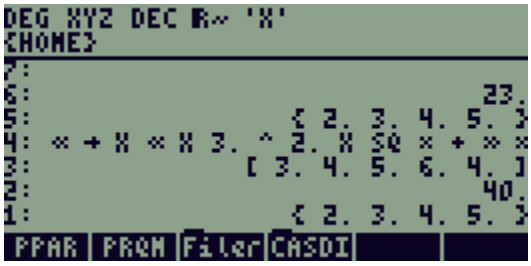
La pila es muy importante porque todos los comandos y operadores de la HP toman sus datos y guardan sus resultados en la misma. Así por ejemplo si queremos sumar los números 10 y 30 dichos números deben ser colocados primero en la pila, para ello simplemente escribimos 10 30 y presionamos la tecla ***Enter*** con lo que la pila queda en la forma:



Entonces al presionar la tecla +, se ejecuta el operador de la suma el cual toma los dos primeros datos de la pila (los números 30 y 10), los suma y coloca el resultado en la pila, con lo que la misma queda en la forma:



Cada número identifica un elemento de la pila y existen varios comandos que nos permiten trabajar directamente con la pila, algunos de ellos emplean estos números como referencia, así por ejemplo para hacer una copia de cualquier elemento de la pila se coloca el número o nivel en el que se encuentra seguido del comando PICK, por ejemplo para copiar la lista que se encuentra en el nivel 4 de la pila, escribimos 4 PICK, con lo que la pila queda en la forma.



Las operaciones y comandos se ejecutan más rápidamente cuando se trabaja directamente con los elementos de la pila (en lugar de trabajar con variables). Por esta razón es frecuente el uso de los comandos que manejan la pila en los programas, sin embargo, no es recomendable abusar de ellos pues los programas se tornan ilegibles y consecuentemente resulta extremadamente difícil modificarlos o corregirlos, actividades que casi siempre se realizan tarde o temprano cuando se elabora un programa.

En la pila se puede guardar cualquier tipo de información, desde un simple número hasta una librería, en realidad sin embargo, lo único que se almacena la pila son direcciones de memoria (cada una de las cuales ocupan 2.5 bytes o lo que es lo mismo 20 bits). Puesto que una dirección de memoria es sólo un número no sería de mucha utilidad ver en pantalla una serie de números, por esta razón el sistema operativo de la calculadora muestra en pantalla el contenido de las direcciones de memoria que se encuentran en la pila, no obstante esto es sólo para efectos de presentación, pues como se dijo en la pila únicamente existen direcciones de memoria.

1.4. Algoritmos

Los algoritmos describen la secuencia lógica de acciones que deben llevarse a cabo para resolver un determinado problema. Existen muchas formas en las que se puede expresar un algoritmo, de ellas emplearemos en esta materia los ***diagramas de actividades***, los cuales son parte de UML (Lenguaje de Modelado Unificado) que es el lenguaje gráfico más utilizado actualmente para el modelado de software.

El inicio de un diagrama de actividades se representa con un círculo relleno:

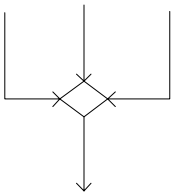
El final del diagrama de actividades se representa con un círculo que contiene un círculo relleno en su interior:

Una acción (o sentencia) se representa con un rectángulo con sus bordes redondeados:

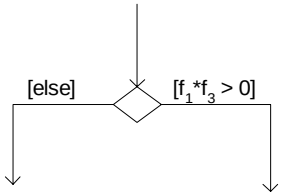
Un flujo, que es el símbolo empleado para unir los elementos del diagrama y señalar la dirección en que deben seguir las acciones, se representa por una flecha continua abierta:

Una unión o bifurcación se representa por un pequeño rombo:

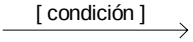
Cuando se emplea como unión llegan dos o más flujos y sale uno.



Cuando se emplea como bifurcación ingresan uno o más flujos y salen dos o más flujos acompañados de alguna condición.

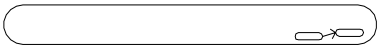


Una condición, tal como se puede observar en la anterior figura, se representa como texto encerrado entre corchetes y siempre está relacionado a algún flujo:



El texto de la condición puede ser una expresión matemática, lógica o literal. Para la condición por defecto se puede emplear ***[else]*** (caso contrario).

Una actividad representa un conjunto de acciones (o subprograma) que se detalla luego por separado empleando otro diagrama de actividades. Se representa como una acción con un icono de una acción llamando a otra en la parte inferior derecha:



Las notas se emplean para comentar y documentar los diagramas de actividades. Se representan como una hoja con una esquina doblada y asociada a cualquiera de los elementos de un diagrama de actividades mediante una línea discontinua:

Al elaborar un diagrama de actividades se debe tener presente que se trata de un lenguaje y como tal es necesario respetar su sintaxis (es decir los símbolos) pues cada uno de ellos tiene su propio significado y si se cambia, cambia también la lógica del algoritmo. Por ejemplo, para representar un flujo siempre se debe emplear la flecha continua abierta, no una flecha cerrada y rellena: ——▶ , una flecha cerrada no rellena: ——▷ , o una flecha discontinua abierta: ⋯⋯> pues cada una de ellas tienen un significado muy diferente (mensaje, herencia y dependencia).

1.5. Programación en User y System RPN

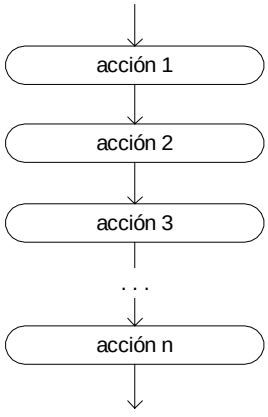
La forma más sencilla y rápida de aprender un nuevo lenguaje de programación es por comparación con otro lenguaje conocido. Puesto que en la Facultad de Tecnología todas las carreras de ingeniería estudian, en Informática I, el lenguaje de programación PASCAL, emplearemos este lenguaje como base para el estudio de los lenguajes *User RPN* y *System RPN*.

Como ya se dijo *User RPN* es un subconjunto de *System RPN*, de manera que todos los comandos y sentencias de *User* puede ser empleados también en *System* (para ello simplemente se antepone una *x* minúscula al nombre del comando *User*). No obstante como los programas elaborados con los comandos y sentencias exclusivos de *System* son más rápidos, son muy raras las ocasiones donde se emplee un comando de *User* en un programa elaborado en *System*.

Al contar tanto *User* como *System* con sentencias y comandos estructurados, abordaremos su estudio describiendo la manera en que se implementan las estructuras estándar de PASCAL en estos lenguajes.

1.5.1. Secuencia

La estructura básica para la elaboración de cualquier programa es la *secuencia*. Una secuencia es una serie de sentencias que se ejecutan una a continuación de otra. El diagrama de actividades de una secuencia se ve aproximadamente como se muestra en la siguiente figura:



Donde las acciones pueden ser acciones simples o acciones compuestas correspondientes a otras estructuras. Recordemos que en PASCAL una secuencia se implementa encerrando las sentencias entre las palabras *begin* y *end* y separando las sentencias con puntos y comas, es decir:

```
begin
  acción 1;
  acción 2;
```

```
  acción 3;
  ...
  acción n;
end;
```

Como también se recordará todo programa, procedimiento o función en PASCAL es una secuencia, razón por la cual siempre comienzan con ***begin*** y terminan con ***end***.

En ***User RPN*** igualmente ***todo programa o subprograma es una secuencia***, pero en ***User*** en lugar de emplear ***begin*** y ***end*** se emplean comillas francesas: **« »**, además los comandos se separan con saltos de línea o espacios en blanco:

```
«
  acción 1
  acción 2
  acción 3
  ...
  acción n
»
```

En ***System RPN*** igualmente ***todo programa o subprograma es una secuencia***, pero en lugar de emplear ***begin*** y ***end*** se emplean cuatro puntos (::) y punto y comma (;). Al igual que en ***User*** los comandos se separan con saltos de línea o espacios en blanco:

```
::
  acción 1
  acción 2
  acción 3
  ...
  acción n
;
```

1.5.2. Variables

Al igual que en PASCAL, en ***User*** y ***System RPN*** se pueden declarar variables tanto globales como locales, sin embargo, en ***User*** y ***System*** la declaración de una variable es una sentencia, por lo que debe formar parte de una secuencia, es decir estar dentro de comillas francesas (para ***User***) o entre :: y ; (para ***System***).

Como sucede con PASCAL, en ***User*** y ***System*** se prefiere el uso de variables locales, pues no interfieren con las variables declaradas en otros programas o módulos, además en ***User*** y ***System RPN*** podemos trabajar también con la pila (stack).

A diferencia de PASCAL, donde las variables pueden almacenar diferentes tipos de datos, en ***User*** y ***System RPN*** todas las variables almacenan únicamente direcciones de memoria, las cuales como sabemos ocupan 2.5 bytes. Se trata en consecuencia de variables tipo puntero. Afortunadamente la asignación y liberación de la memoria ocupada por estas variables es realizada automáticamente por el sistema operativo, lo que facilita considerablemente su uso.

Contrariamente a lo que se podría pensar, el que las variables almacenen únicamente direcciones de memoria no limita sus posibilidades de almacenamiento, sino que por el contrario las amplía, pues la dirección que se guarda en una variable puede apuntar a cualquier tipo de dato. En consecuencia desde el punto de vista práctico en una variable de ***User*** o ***System RPN*** se puede guardar cualquier tipo de información, desde un simple número hasta un programa completo.

Lo anterior es posible debido a la forma en que maneja la información la HP. Todo dato en la HP, desde un número hasta un programa, es almacenado en memoria empleando una estructura denominada ***objeto***. Un ***objeto*** en la HP está conformado por un prólogo (de 20 bits) que identifica al objeto, seguido por el cuerpo del objeto (que son los datos del objeto propiamente) y terminando con un epílogo que puede ser nulo o una dirección de memoria (de 20 bits). De esa manera para acceder a cualquier tipo de dato sólo es necesario conocer su dirección, pues en esa dirección se encuentra toda la información necesaria para trabajar con el mismo.

Debido a esta característica de las variables, todos los programas o subprogramas en la HP, al igual que cualquier otro tipo de dato, son guardados en variables simples (o en librerías).

En cuanto a los nombres, las variables pueden tener nombres conformados por cualquier combinación de letras y números. Se permiten también algunos símbolos especiales como el de subrayado (_).

Algo que se debe tomar muy en cuenta al momento de programar y utilizar programas es que ***User*** y ***System RPN*** (a diferencia de PASCAL) diferencian entre mayúsculas y minúsculas, así ***Var1*** es una variable diferente a ***VAR1***, ***var1***, ***vAr1***, ***vaR1***, ***VAr1***, ***vAR1***, etc.

1.5.2.1. Variables locales

1.5.2.1.1. User RPN

Para declarar variables locales en ***User RPN*** se emplea el carácter correspondiente a la flecha derecha: →. Los datos que se asignan a las variables se escriben a la izquierda de la flecha (o deben encontrarse en la pila) y los nombres de las variables a la derecha. Como ya se dijo, en ***User*** y en ***System*** la declaración de variables es una sentencia, por lo que debe encontrarse dentro de una secuencia.

Por ejemplo para crear 5 variables locales: ***a, b, c, d*** y ***e***, con los valores 10, 20, 30, 40 y 50, escribimos lo siguiente:

```
10 20 30 40 50 → a b c d e
```

De esta manera ***"a"*** apuntará al número 10, ***"b"*** al número 20, ***"c"*** al 30, ***"d"*** al 40 y ***"e"*** al 50.

Para crear 4 variables locales: ***var1, var2, var3*** y ***var4***, donde los valores de ***var1*** y ***var2*** se encuentran en la pila, el valor de ***var3*** es [1 2 3] y el de ***var4*** { 4.5 6.7 8.9 }, escribimos:

```
[ 1 2 3 ] { 4.5 6.7 8.9 } → var1 var2 var3 var4
```

Como se puede observar en este caso no sólo existen datos para dos de las variables (***var3*** y ***var4***) los dos datos faltantes deben encontrarse en la pila. En realidad cuando en un programa se escribe un dato (de cualquier tipo) el mismo (más propiamente su dirección) es colocado en la pila. Por lo tanto en los anteriores ejemplos los valores que se encuentran a la izquierda de la flecha son colocados primero en la pila y luego recién son asignados a las variables respectivas. Por consiguiente en todos los casos los datos que se asignan a las variables deben encontrarse de una u otra forma en la pila.

Para familiarizarnos con el uso de las variables locales así como la elaboración de programas, crearemos algunos de ellos. Con ese fin debe instalar primero el programa DEBUG4X. Una vez instalado, en la listas de programas (en Inicio) encontrará la carpeta: "HP 48 & 49 Development Kit" y dentro de esta carpeta los emuladores: emu48 (para la HP48), emu49 (para la HP49G) y emu49G+ (para la HP49G+). Si cuenta con una calculadora real elija el emula-

dor correspondiente a su modelo, caso contrario elija emu49. Entonces aparecerá en pantalla la calculadora y podrá trabajar con ella de la misma forma que con la calculadora real.

Por defecto el emulador funciona a la misma velocidad que la calculadora real, si desea trabajar a la velocidad de la computadora entonces debe ir al menú ***File*** -> ***Settings*** y en la ventana que aparece desmarcar la opción correspondiente a ***Authentic Calculator Speed***. Es conveniente también que estén desmarcadas las opciones: ***Show Load Object Warning*** y ***Always Show KML Compilation Result*** y que por el contrario estén marcadas las opciones: ***Automatically Save Files*** y ***Automatically Save Files on Exit.***

Toda información que se guarda en la memoria de la calculadora (en el emulador) queda almacenada en disco duro en archivos que tienen la extensión .E49 (o .E48 para la HP48) y que por defecto tienen el nombre ***Default.E49*** (o ***Default.E48***). Se pueden guardar tantas copias de la memoria de la calculadora como se quiera. Para ello debe ir al menú ***File -> Save As…***, ingresar al directorio donde desea guardar la copia, escribir un nombre para la copia (por ejemplo la sigla de la materia) y hacer clic en guardar. Cuando quiera trabajar con una de las copias guardadas debe ir al menú ***File -> Open,*** ingresar al directorio donde guardó la copia y hacer doble clic en la misma.

Estas copias (que ocupan 1 MB) son el modo más cómodo y seguro de llevar la información del emulador de una computadora a otra y es la forma que deben emplear para presentar los trabajos que elaboren en ***User RPN***. Estas copias no sólo sirven para transportar la información, sino que constituyen también un respaldo, pues en ocasiones, al igual que sucede con la calculadora real, la memoria del emulador puede quedar corrompida. Por ello es conveniente tener al menos dos copias de la memoria de la calculadora.

En cuanto a su manejo, todos los comandos y funciones que se ven en el teclado de la calculadora (tanto en el emulador como en la calculadora real) pueden ser utilizados de la siguiente manera: Si el nombre del comando está sobre el teclado, es suficiente presionar la tecla, así para calcular la raíz cuadrada de 2 se escribe el número 2 y luego se presiona la tecla que tiene el símbolo de la raíz cuadrada, obteniéndose: 1.41421356237 (haga la prueba).

Si el nombre del comando o función está en azul (verde para la HP48) o rojo, entonces se debe pulsar primero la tecla de cambio azul o roja:

Denominadas teclas de cambio izquierda y derecha respectivamente, luego se pulsa la tecla donde se encuentra el comando. Así por ejemplo para calcular el logaritmo natural de 4.56, se escribe el número 4.56, se pulsa la tecla de cambio derecha y luego la tecla donde se encuentra la función correspondiente la logaritmo natural (LN), obteniéndose: 1.51732262353 (haga la prueba).

Para escribir las letras que se encuentran en el teclado se pulsa primero la tecla ALPHA:

Y luego la tecla donde se encuentra la letra. Por defecto, el pulsar una vez la tecla ALPHA permite escribir sólo una letra, para escribir más de una letra se debe pulsar la tecla ALPHA dos veces seguidas, con lo que el teclado queda en el modo alfanumérico y en este modo se pueden escribir letras hasta que se pulse otra vez la tecla ALPHA.

Todos los comandos y funciones que se ven en el teclado, así como aquellos que no están en el teclado, pueden ser utilizados también escribiéndolos directamente en la pila y pulsando luego la tecla ENTER:

Así por ejemplo para calcular el logaritmo natural de 4.56 podemos escribir en la pila:

```
4.56 LN
```

Y luego pulsar ENTER (haga la prueba). Si desea que el teclado quede en el modo alfanumérico presionando una sola vez la tecla ALPHA, se debe activar la bandera N° 60. Para activar (o desactivar) cualquier bandera se ingresar a MODE, luego a FLAGS, se busca el número de la bandera y se la activa (o desactiva) pulsando la opción CHK. Alternativamente se puede activar cualquier bandera escribiendo su número (como un número negativo) y a continuación el comando SF (Set Flag), así por ejemplo para activar la bandera N° 60 podemos escribir lo siguiente:

```
-60 SF
```

Como de costumbre se debe pulsar luego ENTER. Para desactivar cualquier bandera se escribe su número (como número negativo) y a continuación el comando CF (Clear Flag), así para desactivar la bandera N° 60 podemos escribir lo siguiente:

```
-60 CF
```

Los programas en ***User RPN*** se escriben directamente en la pila, para ello colocamos primero las comillas francesas (« ») y en su interior escribimos las instrucciones del programa.

Como primer ejemplo elaboremos un programa que devuelva el valor de la función:

$$f(x) = x^3 + 2x^2 + 3x + 4$$

Puesto que ***"x"*** es una variable (y no una constante) su valor debe encontrarse en la pila y en consecuencia sólo debemos declarar la variable "x" (→ x). Luego con la variable se realizan las operaciones correspondientes a la función (x 3 ^ 2 x 2 ^ * + 3 x * + 4 +). Sin embargo, antes de escribir estas operaciones debemos abrir una nueva secuencia, pues en ***User RPN*** siempre que se declara una o más variables es necesario iniciar otra secuencia. Entonces el programa queda en la forma:

```
« → x
  « x 3 ^ 2 x 2 ^ * + 3 x * + 4 + »
»
```

Escriba este programa en el emulador y luego cree el directorio 'PRAC1' escribiendo en la pila lo siguiente:

```
'PRAC1' CRDIR
```

Ingrese al directorio creado (pulsando la tecla correspondiente al mismo) y guarde el programa en la variable global 'F1', escribiendo:

```
'F1' STO
```

Por supuesto en lugar de escribir STO (y luego pulsar ENTER) puede pulsar también la tecla correspondiente a dicho comando. Con ello el programa queda guardado en la variable global 'F1' dentro del directorio 'PRAC1'.

Ahora con el programa creado puede calcular valores de la función para diferentes valores de "x". Así por ejemplo para calcular el valor de la función para x=1, escribimos:

1 F1

Por supuesto en lugar de escribir el nombre de programa puede pulsar la tecla correspondiente al programa. Entonces obtendrá como resultado el número 10. Para calcular el valor de la función para x=4.56, escribimos:

4.56 F1

Con lo que obtenemos el resultado: 154.086016.

Supongamos ahora que queremos modificar este programa pues tenemos la función:

$$f(x) = x^3 + 2x^2 + 3x - 4$$

Que como vemos sólo difiere de la anterior en el último signo. Para modificar un programa debemos colocarlo primero en la pila escribiendo:

'F1' RCL

Como ya se ha señalado reiteradamente, en lugar de escribir el comando RCL (y luego pulsar ENTER), se puede pulsar la tecla equivalente. Alternativamente, se puede pulsar la tecla de cambio derecha y luego la tecla del programa. Siguiendo cualquiera de estos procedimientos tendrá el programa en la pila.

Una vez que se tiene el programa en la pila, se puede editar el mismo pulsando la tecla de cursor hacia abajo:

Entonces se puede corregir o modificar el programa, en este caso se cambia el último signo de "+" a "-", quedando el programa en la forma:

```
« → x
  « x 3 ^ 2 x 2 ^ * + 3 x * + 4 - »
»
```

Guarde este programa con el nombre "F2" ('F2' STO). Entonces calcule el valor de la función para x=2.12 (2.12 F2), con lo que deberá obtener el resultado: 20.876928.

Con frecuencia es conveniente hacer correr un programa paso a paso ya sea para corregir errores como para analizar la forma en que funciona el mismo. Cada vez que se elabora un nuevo programa es una buena práctica hacerlo correr por primera vez paso a paso, de esa manera se detectan los errores y es posible optimizar el código.

Para hacer correr un programa paso a paso, se coloca el programa en la pila (o se escribe el nombre del programa entre apóstrofes), entonces se ingresa al menú PRG, dentro al submenú RUN y dentro se elige la opción DBUG.

Por ejemplo para hacer correr paso a paso el programa F1 para un valor de x=1.5, escribimos 1.5 'F1' y seguimos la secuencia PRG -> RUN -> DBUG (haga la prueba). Entonces en la parte superior de la pantalla aparecerá el mensaje HLT (Halt), indicando que la ejecución del programa ha sido detenida. Para continuar la ejecución con el siguiente comando (o paso) se pulsa la tecla correspondiente a la opción SST o a SST↓. Tanto SST como SST↓ ejecutan el siguiente comando, pero si dicho comando es a su vez un programa SST↓ salta al mismo y la depuración continúa al interior de dicho programa. SST

por el contrario, no ingresa a ningún otro programa y trata los programas (o subprogramas) como si fueran comandos simples.

En nuestro pequeño programa no existen subprogramas, razón por la cual da lo mismo si se emplea SST o SST↓. A medida que vaya depurando el programa (pulsando SST o SST↓), podrá ver en la parte superior de la pantalla el nombre del comando o variable que esté ejecutando, mientras que en la parte inferior de podrá ver como el programa va colocando o quitando valores de la pila. En cualquier momento puede detener la depuración de un programa pulsando la tecla correspondiente a la opción KILL (o escribiendo KILL y pulsando ENTER). Igualmente puede hacer que el programa continúe su ejecución normal pulsando la tecla CONT (o escribiendo CONT y pulsando ENTER).

Como segundo ejemplo elaboraremos un programa que permita evaluar la función:

$$f(k) = \theta_c - \frac{T_0}{k}\left(1 - \cos\left(\sqrt{\frac{k}{I_0}}t\right)\right)$$

Donde:

$T_0 = T_c + \theta_k k$

$T_c = 0.625$

$\theta_c = 2.97$

$\theta_k = 3.27$

$t = 0.0191$

$I_0 = 0.0006$

Como vemos en esta función existen varias constantes. Desde el punto de vista de ***User RPN*** estas constantes son variables locales estáticas con valores iniciales definidos. Se tiene también una expresión intermedia cuyo resultado se guarda en ***T₀***, esta es igualmente una variable local estática pero como el valor de la misma se calcula, puede ser inicializada con un valor arbitrario cualquiera, normalmente 0. Finalmente tenemos la variable: ***k***, que no tiene un valor definido, por lo tanto su valor deberá estar en la pila.

En consecuencia la declaración de variables para esta función es: 0.625 2.97 3.27 0.0191 0.0006 0 → k Tc Thc Thk t Io To; donde el cero se coloca para la variable To. Una vez declaradas las variables se evalúa la expresión intermedia (Tc Thk k * +), se guarda ese valor en To ('To' STO) y con este valor se calcula el valor de la función (Thc To k / 1 k Io / √ COS t * - * -). El programa resultante es:

```
« 0.625 2.97 3.27 0.0191 0.0006 0
  → k Tc Thc Thk t Io To
  « Tc Thk k * + 'To' STO
    Thc To k / 1 k Io / √ t * COS - * -
  »
»
```

Guarde este programa en el directorio PRAC1 con el nombre F3 ('F3' STO). Para evaluar esta función la unidad de medida de los ángulos deben estar en radianes. Puede cambiar la unidad de medida ingresando a MODE, bajando a Angle Measure y seleccionando (con CHOOS) ***radians.*** Alternativamente puede escribir RAD y pulsar ENTER.

Ahora escriba en la pila 3.36 y haga correr el programa paso a paso. Si encuentra algún error, detenga la depuración (con KILL), corrija el error (editando el programa), vuelva a guardar el programa (con 'F3' STO) y vuelva

a hacer correr el programa paso a paso hasta que el mismo finalice sin errores. Si todo va bien al final deberá tener el resultado: .00132605009.

1.5.2.1.2. System RPN

En ***System RPN***, existen dos tipos de variables locales: las variables locales sin nombre (NULLLAM) y las variables locales con nombre (LAM). Las variables locales sin nombre, son referenciadas por el orden en que son creadas, son más rápidas que las variables locales con nombre, por lo que es más eficiente trabajar con ellas, sin embargo, no se aconseja su uso cuando el programa llama a otros programas (o subprogramas) que a su vez crean otras variables locales. Cuando ello sucede las posiciones relativas de las variables en el programa son afectadas, generando errores y dificultando considerablemente la depuración del programa.

Las variables locales con nombres (LAM) se declaran de forma similar a las variables locales de ***User RPN***. Si tienen valores iniciales se colocan primero dichos valores (caso contrario los valores deberán estar en la pila), luego se escribe la lista de variables entre llaves (precedidas por la palabra LAM) y se crean las variables con el comando BIND. A diferencia de ***User***, las variables locales de ***System*** deben ser eliminadas manualmente por el programador (cuando ya no son necesarias), con el comando ABND. La forma general en la que se declaran variables locales con nombre en ***System*** es:

Valores iniciales { lista de variables } BIND

(sentencias donde se utilizan las variables)

ABND

Como ya se mencionó anteriormente, los comandos de ***System*** no verifican nada, por consiguiente el comando ***BIND*** crea las variables existan o no los datos para las mismas, sin embargo, si no existen se escribirá en sectores erróneos de la memoria provocando un fallo del sistema. Por ello en ***System*** se debe verificar previamente que exista el número de datos adecuado, para ello se emplean los comandos CK0NOLASTWD, CK1NOLASTWD, CK2NOLASTWD, CK3NOLASTWD, CK4NOLASTWD, CK5NOLASTWD y CKNNOLASTWD (o CK0, CK1, CK2, CK3, CK4, CK5 y CKN si se trata de una librería), los primeros 6 verifican que existan 0, 1, 2, 3, 4 o 5 datos en la pila, el último permite verificar la existencia de un número arbitrario de datos y debe estar precedido por un número entero binario (BINT).

Los números enteros binarios (BINT) son de uso frecuente en ***System*** debido a que almacenan 20 bits y como sabemos ese es el tamaño de una dirección de memoria. Los BINT de uso más frecuente (hasta el 64 en la HP48 y hasta el 130 en la HP49) están predefinidos y pueden ser colocados en la pila escribiendo el número en inglés (o BINT seguido del número en la HP49). Así para colocar en la pila el entero binario 25 escribimos:

```
TWENTYFIVE
```

O alternativamente BINT25 en la HP49. Otros BINT no predefinidos pueden ser colocados en la pila escribiendo el símbolo "#" seguido del número en notación hexadecimal, así por ejemplo para colocar en la pila el entero binario 205 escribimos:

```
# CD
```

Para verificar la existencia de 30 datos en la pila escribimos:

```
THIRTY CKNNOLASTWD
```

O THIRTY CKN si estamos trabajando en una librería.

Por otra parte es importante comprobar que los datos sean del tipo correcto, pues a diferencia de ***User***, todas las funciones y comandos en ***System*** son específicos para un determinado tipo de dato, así por ejemplo para sumar dos números reales el comando es %+, para sumar dos números enteros binarios es #+, para sumar (o concatenar) dos cadenas (strings) es &$, mientras que en ***User*** para todos ellos es "+". En ***System*** la mayoría de los comandos, operadores y funciones que trabajan con números enteros binarios (BINT) están precedidos por el signo "#", los que trabajan con números reales por "%", los que trabajan con números reales de doble precisión (que no existen en ***User***) por "%%", los que trabajan con números complejos por "C%" y los que trabajan con números complejos de doble precisión por "C%%".

Para comprobar los tipos de datos se emplea el comando CK&DISPATCH0 o CK&DISPATCH1. Ambos tienen el siguiente formato general:

```
CK&DISPATCH1
  #tipos1 acción1
  #tipos2 acción2
  . . .
  #tiposn acciónn
;
```

Donde #tipos1, #tipos2, etc., son números enteros binarios (BINT) que tienen el formato "# nnnnn", donde cada "n" es un número hexadecimal que identifica a un tipo de dato en particular. Por lo tanto cada BINT puede verificar los tipos de hasta 5 datos de la pila (menos para aquellos tipos de datos que se identifican con dos números hexadecimales). La forma en que funcionan estos comandos es la siguiente: primero comparan los tipos de datos de la pila con el BINT ***#tipos1*** y si son iguales ejecutan la ***acción1***, luego de lo cual el programa continúa después del ";", caso contrario comparan con el BINT ***#tipos2*** y si son iguales ejecuta la ***acción2***, luego de lo cual el programa continúa después del ";" y así sucesivamente. Si ninguno de los BINTS concuerda con los tipos de datos de la pila se genera el error "Bad Argument Type" (tipo de argumento erróneo).

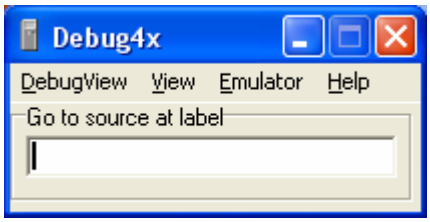
Como se mencionó cada tipo de dato se identifica por un número. Los números para los datos de uso más frecuente son los siguientes:

0	:	Cualquier tipo de dato
1	:	Número real
2	:	Número complejo
3	:	Cadena (String)
4	:	Vector o matriz (Array)
5	:	Lista
6	:	Variable global (ID)
7	:	Variable local (LAM)
8	:	Programa
9	:	Simbólico
C	:	Objeto gráfico (GROB)
D	:	Objeto etiquetado (TAG)
E	:	Objeto de unidad
1F	:	Entero binario (BINT)
2F	:	Directorio
3F	:	Real extendido (%%)
4F	:	Complejo extendido (C%%)

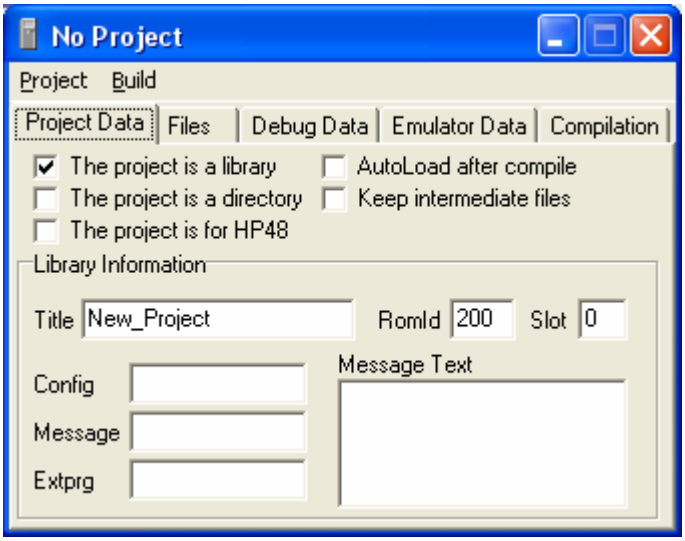
Así por ejemplo para verificar que en la pila existe un programa en el quinto nivel, una lista en el cuarto, un vector o matriz en el tercero, una cadena en el segundo y un número real en el primero, el número entero binario sería: # 85431.

La diferencia entre CK&DISPATCH0 y CK&DISPATCH1 es que el segundo si no encuentra concordancia con ninguno de los BINT, convierte los enteros largos (ZINT) y los objetos etiquetados (TAG) a números reales y realiza una segunda pasada.

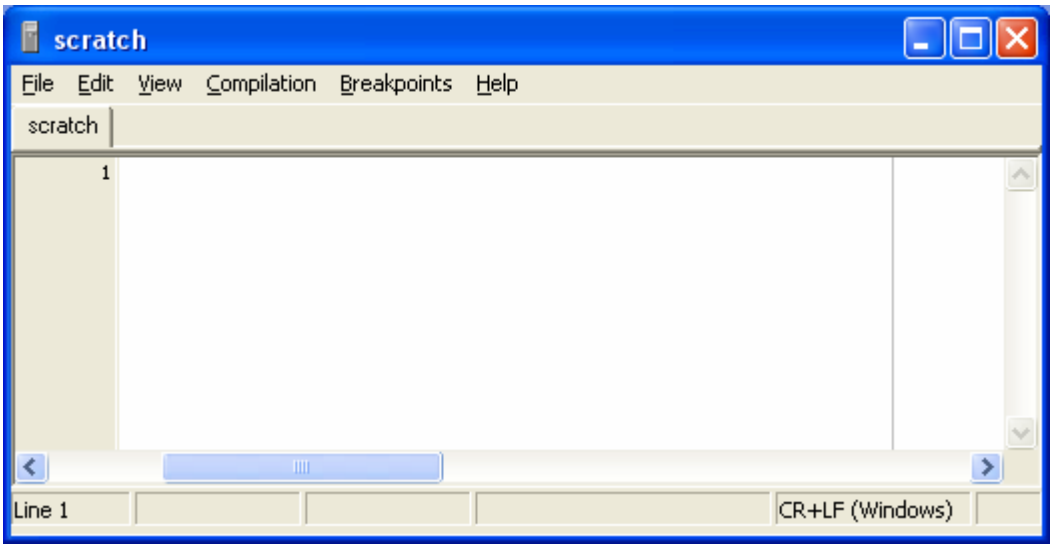
Los programas en ***System RPN*** serán elaborados con Debug4x, que como los emuladores, se encuentra dentro de la carpeta "HP 48 & 49 Development Kit". Al ingresar al programa tendrá una ventana como la siguiente:



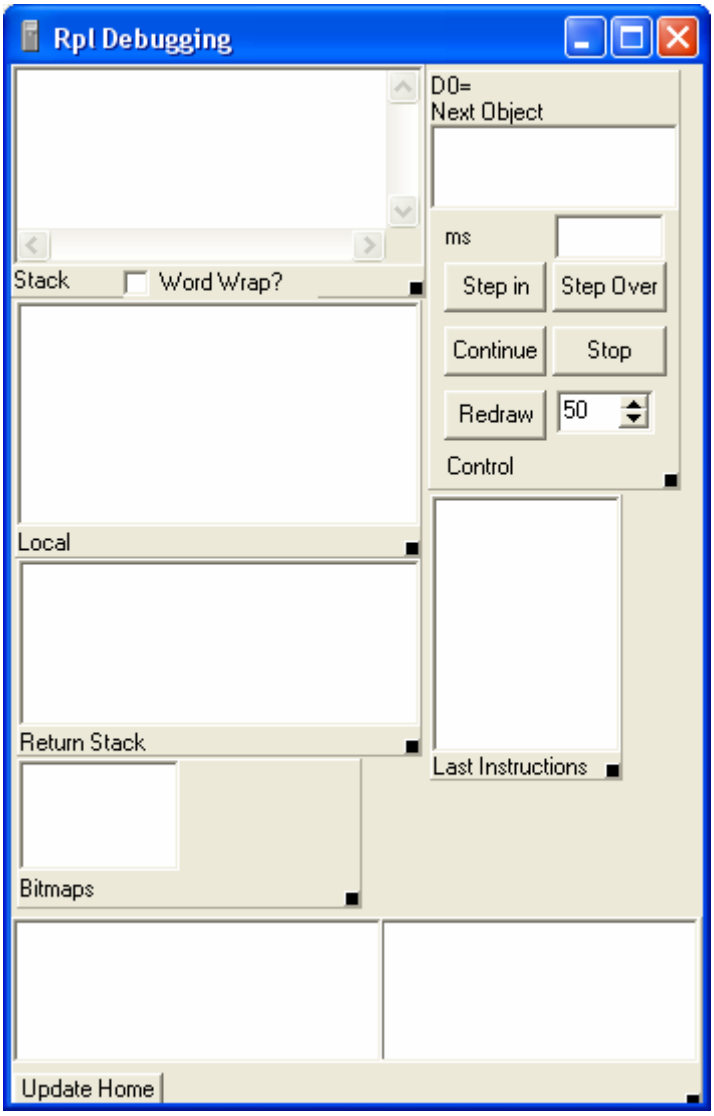
Si no está visible la ventana de proyectos vaya al menú ***View*** -> Project, con lo que deberá aparecer la siguiente ventana:



De la misma manera, si no está visible la ventana de edición vaya al menú ***View -> Editor,*** con lo que aparecerá una ventana similar a la siguiente:



Si no está abierta la ventana del depurador vaya al menú DebugView -> Debugger RPL, con lo que aparecerá una ventana similar a la siguiente:



Finalmente para ver el emulador de la calculadora vaya al menú ***Emulator -> Emulate***.

Como primer ejemplo elaboraremos un programa en ***System RPN*** para evaluar la función que programamos en ***User:***

$$f(x) = x^3 + 2x^2 + 3x + 4$$

En la ventana de proyectos vaya al menú ***Project -> New Project***. En la ventana que aparece cree en el directorio raíz del disco duro el directorio RPN, dentro de este el directorio PRAC1 y dentro guarde el proyecto con el nombre ***Proyecto1***.

Como en este caso crearemos un programa (no una librería) desmarque la opción "The project es a library" y asegúrese que esté marcada la opción "Autoload after compile" para que el programa, una vez compilado, aparezca en el emulador. Si está utilizando una calculadora HP48, entonces marque también la opción "The project is for HP48". En ***Path*** escriba: "PRAC1 f1", que es el nombre del directorio y de la variable donde se guardará el programa en el emulador, observe que "f1" está en minúsculas.

Ahora, en la ventana de proyectos haga clic en la pestaña *"Files"* y vaya al menú ***Project -> New Source File***. En la ventana que aparece escriba como nombre del archivo *"f1"* y haga clic en el botón *"Guardar"*.

Entonces en la ventana de edición aparece un encabezado y se puede comenzar a escribir el programa debajo del mismo. Las consideraciones para este programa son esencialmente las mismas que para su equivalente en ***User***, excepto que ahora se debe verificar el número de datos (1) y el tipo de los mismos. Puesto que las funciones normalmente se evalúan para números reales, este será el tipo de dato que reciba el programa, pero para permitir además números enteros (ZINT) y objetos etiquetados (TAG) se verificará el tipo con CK&DISPATCH1. Tomando en cuentas estas consideraciones se ha escrito el siguiente programa (transcriba el mismo en la ventana de edición):

```
::
  CK1NOLASTWD
  CK&DISPATCH1
    ONE
    ::
      { LAM x }
      BIND
      LAM x %3 %^ %2 LAM x %2 %^ %* %+ %3 LAM x %* %+ %4 %+
      ABND
    ;
;
```

Una vez escrito el programa guárdelo ***(File -> Save)*** y compile el proyecto ***(Compilation -> Build Project)***, entonces, al cabo de algunos segundos aparecerá en el emulador, en el directorio "PRAC1", el archivo *"f1"*. Para hacer correr este programa paso a paso, que es lo recomendable la primera vez que se hace correr un programa, haga clic a la izquierda del número que aparece en la línea donde se encuentra { LAM x }, entonces verá que aparece un punto rojo, este es un punto de ruptura (breakpoint) y se coloca de esta manera en él o los lugares donde uno quiere detener la ejecución de un programa.

Ahora en el emulador escriba el número 1 y pulse la tecla correspondiente al programa "f1". Entonces el programa comenzará a correr y se detendrá en la línea donde se puso el punto de ruptura (quedando remarcada con un fondo celeste). Para continuar la ejecución paso a paso haga clic en el botón "Step over" (equivalente a SST) en la ventana de depuración, si quiere ingresar a los subprogramas deberá hacer clic en "Step in" (equivalente a SST↓). A medida que vaya ejecutando el programa paso a paso (haciendo clic en "Step over" o "Step in") podrá ver el contenido de la pila (en Stack), las variables locales (en Local), el siguiente comando (en Next Object) e información adicional en los otros recuadros. Finalmente cuando se ejecute el comando ABND termine el programa haciendo clic en el botón "***Continue***". Si no ha cometido ningún error obtendrá (en el emulador) el resultado 10, que por supuesto es el mismo resultado calculado con el programa equivalente en ***User***.

Una vez probado el programa, elimine el punto de ruptura (volviendo a hacer clic en el mismo) y haga correr el programa para x=4.56, escribiendo en el emulador:

```
4.56 f1
```

Con lo que obtendrá el resultado: 154.086016.

El modificar o corregir un programa con ***Debug4x*** es incluso más sencillo que en ***User***. Por ejemplo si se desea modificar el proyecto (sin guardar el proyecto original) para evaluar la función:

$$f(x) = x^3 + 2x^2 + 3x - 4$$

Que difiere de la anterior sólo en el último signo, debe proceder de la siguiente manera: en la ventana de proyectos (en Path) cambie "PRAC1 f1" por "PRAC1 f2" de manera que el nuevo programa se guarde en la variable "f2". En la ventana de edición modifique el programa (cambie el último signo + por -) y compile el proyecto ***(Compilation -> Build project)***. Entonces en el emulador aparecerá un nuevo programa "f2" que se el programa modificado. Al hacer correr el programa con x=2.12 ***(2.12 f2)***, obtendrá el resultado 20.876928.

Como segundo ejemplo elaboremos un programa en ***System RPN*** para evaluar la función que programamos como segundo ejemplo en ***User:***

$$f(k) = \theta_c - \frac{T_0}{k}\left(1 - \cos\left(\sqrt{\frac{k}{I_0}}t\right)\right)$$

Donde:

$T_0 = T_c + \theta_k k$

$T_c = 0.625$

$\theta_c = 2.97$

$\theta_k = 3.27$

$t = 0.0191$

$I_0 = 0.0006$

Para crear este programa elaboraremos un nuevo proyecto ***(Project -> New Project)*** con el nombre ***Proyecto2***. Recuerde desmarcar la opción ***"The project is a library"*** y marcar la opción ***"Autoload after compile"***. El programa será guardado con el nombre "f3" (en minúscula), por lo tanto en ***Path***, en la ventana de proyectos escriba: "PRAC f3" y en la pestaña ***"Files"*** añada el archivo "F3.S" ***(Project -> New Source File).***

Las consideraciones para este programa son las mismas que en ***User***, con el cuidado adicional de controlar el número (1) y tipo de datos (real). El programa, que debe ser escrito en la ventana de edición, es el siguiente:

```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  ::
    % 0.625 % 2.97 % 3.27 % 0.0191 % 0.0006 %0
    { LAM k LAM Tc LAM Thc LAM Thk LAM t LAM Io LAM To }
    BIND
    LAM Tc LAM Thk LAM k %* %+
    ' LAM To STO
    LAM Thc LAM To LAM k %/ %1 LAM k LAM Io %/
    %SQRT LAM t %* %COS %- %* %-
    ABND
  ;
;
```

Observe que todos los números, operadores y funciones están precedidos por el símbolo "%" porque se trabaja con números reales (la raíz cuadrada se calcula con %SQRT). Observe también que para guardar el resultado en la variable "To", se escribe un apóstrofe delante de la variable, esto permite colocar la variable (y no su contenido) en la pila. Esto es algo que se debe hacer siempre que sea necesario colocar un objeto en la pila sin evaluarlo.

Guarde el programa ***(File -> Save All)***, compile el proyecto ***(Compilation -> Build project)*** y si existen errores corríjalos, luego coloque un punto de ruptura en la línea donde se encuentran las constantes. En el emulador escriba el número 3.36 y haga correr el programa paso a paso. Si todo está correcto, al final obtendrá el resultado: .00132605009.

Finalmente quite el punto de ruptura y haga correr el programa con k=3.5 (3.5 f3) con lo que deberá obtener el resultado: -.09310507507.

Como se dijo, en ***System*** se pueden declarar también variables locales sin nombre (NULLLAM). Estas variables se declaran de manera similar a las variables locales con nombre, excepto que todas las variables tienen el nombre "NULLLAM". Así por ejemplo para declarar 7 variables locales sin nombre (todas con valores iniciales) es escribe:

```
%1 %2 %3 %4 %5 %6 %7
{ NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM NULLLAM }
BIND
```

Se puede evitar escribir "n" veces NULLLAM, si se coloca un "NULLLAM" en la pila con " ' NULLLAM " y se duplica n veces este valor escribiendo el número como entero binario seguido del comando "NDUPN", luego se crean las variables con el comando DOBIND (no BIND porque no se trata de una lista). Así las 7 variables anteriores pueden ser creadas de la siguiente manera:

```
%1 %2 %3 %4 %5 %6 %7
' NULLLAM SEVEN NDUPN
DOBIND
```

En ***System RPN,*** muchos comandos son sólo combinaciones de otros y su único fin es el ahorro de memoria. Por ejemplo para crear una variable local se puede emplear el comando "1LAMBIND" en lugar de "{ NULLLAM } BIND" o "'NULLLAM ONE DOBIND".

Una vez creada una variable local se accede a la misma en base al orden de creación, por ejemplo para acceder a la primera variable local se escribe 1GETLAM, para la cuarta 4GETLAM y de la misma manera para cualquier otra variable. Para guardar nuevos valores en las variables locales se emplea el mismo procedimiento, así para guardar un nuevo valor en la primera variable se escribe 1PUTLAM, para guardar uno en la cuarta 4PUTLAM, etc.

Las variables locales sin nombre se enlazan a sus valores de manera diferente a las variables locales con nombre, la primera variable local (1GETLAM) toma el primer valor de la pila, la segunda (2GETLAM) el segundo y así sucesivamente. Por lo tanto, para el ejemplo 1GETLAM tiene el número %7, 2GETLAM el número %6, 3GETLAM el número %5 y así sucesivamente.

Como primer ejemplo del uso de variables locales sin nombre volveremos a programar la función: $f(x) = x^3 + 2x^2 + 3x + 4$.

Para ello creamos un nuevo proyecto con el nombre ***Proyecto4 (Project -> New Project),*** recordando desmarcar la opción ***"The project is a library"*** y marcar la opción ***"Autoload after compile"***. El programa se llamará ***fun1***, por lo que en ***Path*** se escribe: ***"PRAC1 fun1"***. En ***Files*** añada el archivo ***FUN1.S (Project -> New Source File),*** recuerde también visualizar el emulador ***(Emulator -> Emulate).*** Entonces en la ventana de edición escriba el siguiente programa:

```
::
  CK1NOLASTWD
  CK&DISPATCH1
    ONE
    ::
```

```
      1LAMBIND
      1GETLAM %3 %^ %2 1GETLAM %2 %^ %* %+ %3 1GETLAM %* %+ %4 %+
      ABND
    ;
;
```

Guarde las modificaciones ***(File -> Save All)***, compile el proyecto ***(Compilation -> Build Project)***, coloque un punto de ruptura en la línea donde se encuentra el comando 1LAMBIND y haga correr el programa paso a paso con x=4.56 ***(4.56 fun1)***. Si todo va bien obtendrá el resultado 154.086016.

Como segundo ejemplo del uso de variables locales sin nombre volveremos a programar la función:

$$f(k) = \theta_c - \frac{T_0}{k}\left(1 - \cos\left(\sqrt{\frac{k}{I_0}}t\right)\right)$$

Donde:

$$T_0 = T_c + \theta_k k$$
$$T_c = 0.625$$
$$\theta_c = 2.97$$
$$\theta_k = 3.27$$
$$t = 0.0191$$
$$I_0 = 0.0006$$

Para ello creamos un nuevo proyecto con el nombre ***Proyecto5 (Project -> New Project)***, recordando desmarcar la opción ***"The project is a library"*** y marcar la opción ***"Autoload after compile"***. El programa se llamará ***fun3***, por lo que en ***Path*** se escribe: ***"PRAC1 fun3"***. En ***Files*** añada el archivo ***FUN3.S (Project -> New Source File)***, recuerde también visualizar el emulador ***(Emulator -> Emulate)***. Entonces en la ventana de edición escriba el siguiente programa:

```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  :: ( k=7 Tc=6 Thc=5 Thk=4 t=3 Io=2 To=1 )
    % 0.625 % 2.97 % 3.27 % 0.0191 % 0.0006 %0
    ' NULLLAM SEVEN NDUPN
    DOBIND
    6GETLAM 4GETLAM 7GETLAM %* %+ 1PUTLAM ( To=Tc+Thk*k )
    5GETLAM 1GETLAM 7GETLAM %/
    %1 7GETLAM 2GETLAM %/ %SQRT
    3GETLAM %* %COS %- %* %- ( Thc-To/k*[1-COS[[k/Io]^0.5*t]] )
    ABND
  ;
;
```

Donde las expresiones que están entre paréntesis son comentarios, los cuales se emplean para hacer el código más entendible pero que no forman parte realmente del programa, por lo que en realidad pueden ser omitidos. Sin embargo se recomienda su uso, sobre todo cuando se trabaja con variables locales sin nombre, pues sólo de esa manera el código es entendible y es posible corregir y modificar estos programas. Por ejemplo en el primer comentario se colocan los números que son equivalentes a las variables, así la variable "t" está representada por el número 3, por lo tanto para obtener su

valor se emplea 3GETLAM y para guardar un valor en la misma se emplea 3PUTLAM.

Una vez escrito el programa, guarde las modificaciones ***(File -> Save All)***, compile el proyecto ***(Compilation -> Build Project)***, coloque un punto de ruptura en la línea donde se encuentran los números y haga correr el programa paso a paso con ***k=3.36 (3.36 fun3).*** Si todo va bien obtendrá el resultado: .00132605009.

Alternativamente es posible darles nombres temporales a las variables sin nombre. Ello permite trabajar con las variables sin nombre de manera similar a como se trabaja con las variables locales de ***User***, sin embargo se debe recordar que estos nombres son sólo temporales pero que no existen realmente en el programa.

Para darles nombres temporales a las variables sin nombre, se declaran colocando los nombres entre dobles llaves:

```
{{ Nombres temporales }}
```

Para utilizar una de estas variables simplemente se escribe el nombre temporal y para guardar un valor se escribe el nombre temporal precedido del signo de admiración (!).

Como ejemplo programaremos nuevamente la función $f(x)=x^3+2x^2+3x+4$, pero empleando nombres temporales para las variables locales sin nombre.

Con ese fin crearemos un nuevo proyecto con el nombre ***"Proyecto7" (Project -> New Project)***, recordando desmarcar la opción ***"The project is a library"*** y marcar la opción ***"Autoload after compile"***. El programa se llamará ***fn3***, por lo que en ***Path*** se debe escribir: ***"PRAC1 fn3"***. En ***Files*** añada el archivo ***FN3.S (Project -> New Source File)***, recuerde también visualizar el emulador ***(Emulator -> Emulate).*** Entonces en la ventana de edición escriba el siguiente programa:

```
 ::
  CK1NOLASTWD
  CK&DISPATCH1
    ONE
    ::
      {{ x }}
      x %3 %^ %2 x %2 %^ %* %+ %3 x %* %+ %4 %+
      ABND
    ;
;
```

Guarde las modificaciones ***(File -> Save All)***, compile el proyecto ***(Compilation -> Build Project)***, coloque un punto de ruptura en la línea donde se encuentra el nombre temporal de la variable y haga correr el programa paso a paso con x=4.56 ***(4.56 fn1)***. Entonces podrá observar que no existe realmente la variable "x", pues en ***"Next object"*** verá "1GETLAM" en lugar de "x". Al final obtendrá el resultado 154.086016.

Como se puede observar la elaboración de programas es más sencilla cuando se emplean nombres temporales, lamentablemente, debido a un error en DEBUG4X, sólo es posible emplear nombres temporales en programas individuales, como los que hemos estado elaborando hasta el momento, pero no en directorios o librerías, por lo que en las mismas nos vemos obligados a trabajar (hasta que se corrija ese error) con los comandos GETLAM y PUTLAM.

Como segundo ejemplo de la aplicación de nombres temporales volveremos a programar la función:

$$f(k) = \theta_c - \frac{T_0}{k}\left(1 - \cos\left(\sqrt{\frac{k}{I_0}}t\right)\right)$$

$Donde:$

$T_0 = T_c + \theta_k k$

$T_c = 0.625$

$\theta_c = 2.97$

$\theta_k = 3.27$

$t = 0.0191$

$I_0 = 0.0006$

Para ello creamos un nuevo proyecto con el nombre ***Proyecto8 (Project -> New Project),*** recordando desmarcar la opción ***"The project is a library"*** y marcar la opción ***"Autoload after compile".*** El programa se llamará ***fn3,*** por lo que en ***Path*** se escribe: ***"PRAC1 fn3".*** En ***Files*** añada el archivo ***FN3.S (Project -> New Source File),*** recuerde también visualizar el emulador ***(Emulator -> Emulate).*** Entonces en la ventana de edición escriba el siguiente programa:

```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  ::
    % 0.625 % 2.97 % 3.27 % 0.0191 % 0.0006 %0
    {{ k Tc Thc Thk t Io To }}
    Tc Thk k %* %+
    !To
    Thc To k %/ %1 k Io %/
    %SQRT t %* %COS %- %* %-
    ABND
  ;
;
```

Una vez escrito el programa, guarde las modificaciones ***(File -> Save All),*** compile el proyecto ***(Compilation -> Build Project),*** coloque un punto de ruptura en la línea donde se encuentran los números y haga correr el programa paso a paso con ***k=3.36 (3.36 fun3).*** Si todo va bien obtendrá el resultado: .00132605009.

Al elaborar un programa en ***System RPN*** no se deben combinar las variables locales con nombre (LAM) con las variables locales sin nombre (NULLLAM), pues la nomenclatura empleada en las variables locales sin nombre (GETLAM y PUTLAM) funciona también con las variables locales con nombre, así el siguiente programa coloca el número %5 en la pila (haga la prueba):

```
::
  %3 %2 %0
  { LAM a LAM b LAM c }
  BIND
  LAM a LAM b %+ ' LAM c STO
  1GETLAM
  ABND
;
```

De manera que el combinar ambos tipos de variables puede dar lugar a errores difíciles de corregir.

1.5.2.2. Variables globales

Las variables globales se emplean principalmente para guardar programas y algunos datos de importancia en la memoria del usuario (puerto 0). Como ya se dijo en los programas se prefiere trabajar con variables locales porque no interfieren con otros programas y son por lo tanto más seguras.

1.5.2.2.1. User RPN

En *User RPN* ya hemos empleado variables globales para guardar los programas que hemos elaborado. El formato general para guardar un objeto de cualquier tipo, en una variable global (tanto en un programa como en la pila) es el siguiente:

'Nombre de la variable' STO

Donde el objeto a guardar debe encontrarse en la pila. Este comando debe ser utilizado con cierto cuidado, pues no verifica previamente si existe otra variable con el mismo nombre. Por lo tanto si se crea una variable global con un nombre ya existente, se pierde para siempre el contenido de la variable original.

Para eliminar una variable global se emplea el comando *PURGE* de acuerdo al siguiente formato:

'Nombre de la variable' PURGE

O si se quiere eliminar más de una variable, se escriben los nombres de las variables en una lista:

{ Nombres de las variables } PURGE

Igualmente este comando debe ser utilizado con precaución pues por defecto no pide confirmación para eliminar la o las variables.

1.5.2.2.2. System RPN

Las variables globales en *System RPN (IDs),* se crean de la misma forma en que se guardan nuevos valores en una variable local *(con STO)*, sólo que el nombre de la variable debe estar precedido por *ID* en lugar de *LAM*. El formato general para crear (o guardar un nuevo valor) en una variable global en *System* es el siguiente:

' ID Nombre de la variable STO

Donde el objeto a guardar debe encontrarse en la pila.

Para eliminar una variable global se emplea el comando *PURGE* de acuerdo al siguiente formato:

ID Nombre de la variable PURGE

Al trabajar con variables globales en *System* se debe tener incluso más cuidado que en *User*, pues los comandos de *System* suelen tener un alcance más general. Por ejemplo el comando *PURGE* de *System* busca la variable a eliminar en el directorio actual y si no la encuentra busca la variable en el directorio padre y así sucesivamente hasta llegar al directorio raíz *(HOME)*. Por lo tanto si en *System* se escribe mal un nombre es posible que se esté eliminando una variable (que no quería eliminarse) en otro directorio.

1.5.3. Preguntas y Ejercicios

Tanto los ejemplos como los ejercicios que se dan al final de las preguntas, deben ser presentados en un CD. Para este fin, una vez realizados los

ejemplos y ejercicios, guarde una copia de la memoria del emulador con el nombre "PR1nn_nnn"; donde "nn_nnn" es el número de su carnet universitario. Entonces, en el CD cree el directorio PRAC1 y dentro de él copie el archivo PR1nn_nnn así como todos los archivos que se encuentran en el directorio RPN\PRAC1 de su disco duro.

1. ¿Con qué lenguaje de programación se pueden lograr los programas más rápidos y eficientes?
2. ¿Por qué no es práctico elaborar programas en el modo algebraico?
3. ¿Por qué los programas escritos en System RPN son más rápidos que los escritos en User RPN?
4. ¿Por qué la programación en System requiere mayor cuidado por parte del programador?
5. ¿Qué aspectos se deben tomar en cuenta al emplear la notación RPN?
6. ¿Qué es la pila?
7. ¿Por qué es importante la pila?
8. Si en la pila es encuentran un programa y un número ¿Cuál de ellos ocupa más memoria en la pila?
9. ¿Por qué en los programas se suele emplear comandos que manejan directamente la pila?
10. ¿Qué son los algoritmos?
11. ¿Por qué al elaborar un diagrama de actividades se debe respetar la sintaxis?
12. ¿Qué es una secuencia?
13. ¿Cómo se implementa una secuencia en User RPN?
14. ¿Cómo se implementa una secuencia en System RPN?
15. ¿Qué tipo de información almacena una variable en User y System RPN?
16. ¿Cuándo se depura un programa cuál es la diferencia entre SST y SST↓?
17. Elabore un programa en User RPN para evaluar la función que se presenta a continuación. El programa debe ser guardado con el nombre F4 y probado con f=0.009 (siendo el resultado: .00851274051)

$$f(f)=\frac{1}{\sqrt{f}}-\left(\frac{1}{k}\right)\ln\left(\mathrm{R_e}\sqrt{f}\right)+\left(14-\frac{5.6}{k}\right)$$

donde:

$k=0.28$

$\mathrm{R_e}=37.50$

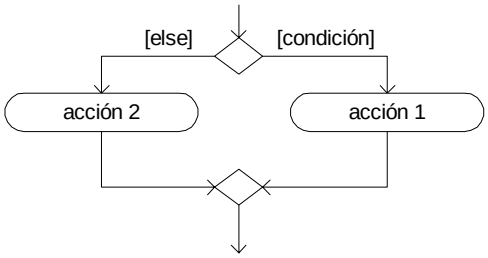
18. Repita el ejercicio anterior pero elaborando el programa en ***System RPN*** y guardando el programa en la variable "f4".
19. Repita el ejercicio anterior pero empleando variables locales sin nombre y guardando el programa en la variable "fun4".
20. Repita el ejercicio anterior pero empleando nombres temporales para las variables locales sin nombre y guardando el programa en la variable ***"fn4".***

1.5.4. Selección

Sólo los problemas más sencillos, como los estudiados hasta ahora, pueden ser resueltos empleando únicamente la secuencia, en general la mayoría de los problemas prácticos tienen dos o más alternativas que se siguen dependiendo de ciertas condiciones que deben cumplirse. Para este fin se recurre a las estructuras selectivas y tanto en *User* como en *System RPN* se pueden implementar las estructuras selectivas *IF-THEN-ELSE* y *CASE* de PASCAL.

1.5.4.1. Estructura IF-THEN-ELSE

La estructura *IF-THEN-ELSE* tiene el siguiente diagrama de actividades:



Y recordemos que en PASCAL se implementa de la siguiente manera:

if *condición* **then** *acción 1* **else** *acción 2;*

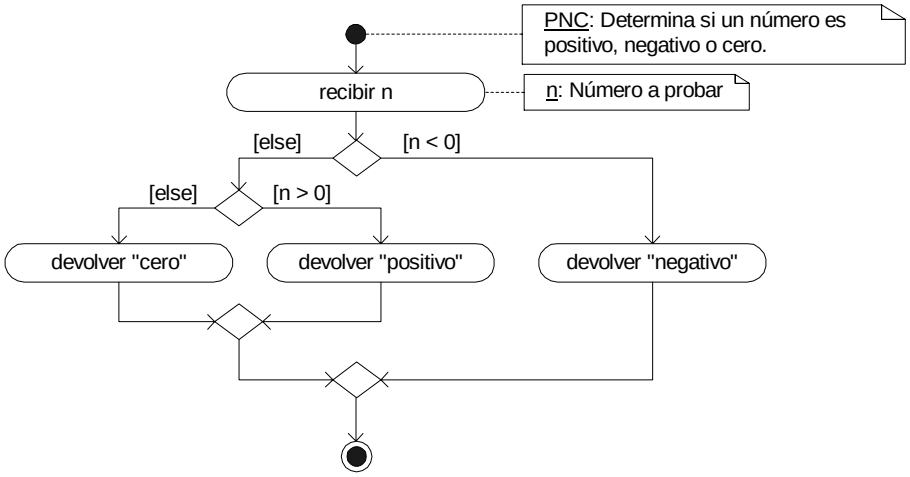
Donde *condición* es cualquier expresión lógica que devuelve un resultado falso o verdadero. Si *acción 1* y/o *acción 2* involucran más de una sentencia se trata como una secuencia, es decir encerrando las sentencias entre *Begin* y *End*.

En *User RPN* esta estructura se implementa de la siguiente manera:

IF** condición **THEN** acción 1 **ELSE** acción 2 **END;

Si *acción 1* y *acción 2* constan de más de una sentencia no se requiere una secuencia (« »). Al igual que en PASCAL el caso contrario *(ELSE)* es opcional, es decir que puede existir o no. En User el valor verdadero está representado por cualquier número diferente de cero (generalmente 1), mientras que el valor falso está representado por el número 0.

Como ejemplo elaboraremos un programa que determine si un número es positivo, negativo o cero. El algoritmo a seguir es el siguiente:



Como se puede observar en el algoritmo existen dos estructuras ***IF-THEN-ELSE*** (una anidada). La implementación en ***User RPN*** es la siguiente (recuerde trabajar en el directorio PRAC1):

```
« → n
 « IF n 0 <
   THEN "NEGATIVO"
   ELSE
     IF n 0 >
     THEN "POSITIVO"
     ELSE "CERO"
     END
   END
 »
»
```

Guarde el programa con el nombre ***"PNC" ('PNC' STO)*** y haga correr el mismo (paso a paso) con un número positivo, uno negativo y cero.

En ***System RPN*** la estructura ***IF-THEN-ELSE*** se implementa de la siguiente manera:

condición ***ITE*** *acción 1 acción 2*

En ***System***, al igual que en PASCAL, si ***acción 1*** o ***acción 2*** constan de más de una sentencia deben ser escritas como secuencias (es decir deben estar entre "::" y ";"). Cuando no se quiere el caso contrario (ELSE), en ***System*** se escribe:

condición ***IT*** *acción 1*

En ***System RPN*** (al igual que en PASCAL) el valor lógico ***verdadero*** es ***TRUE*** y el valor lógico ***falso*** es ***FALSE***.

Como ejemplo implementemos el algoritmo que determina si un número es positivo negativo o cero en ***System RPN***. Con ese fin y siguiendo el procedimiento explicado en los anteriores ejemplos, cree el programa ***"pnc"*** en el proyecto ***"PNC"***, y escriba el siguiente programa en el archivo ***PNC.S***:

```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  ::
    1LAMBIND
    1GETLAM %0 %<
    ITE "NEGATIVO"
    ::
      1GETLAM %0 %>
      ITE "POSITIVO" "CERO"
    ;
    ABND
  ;
;
```

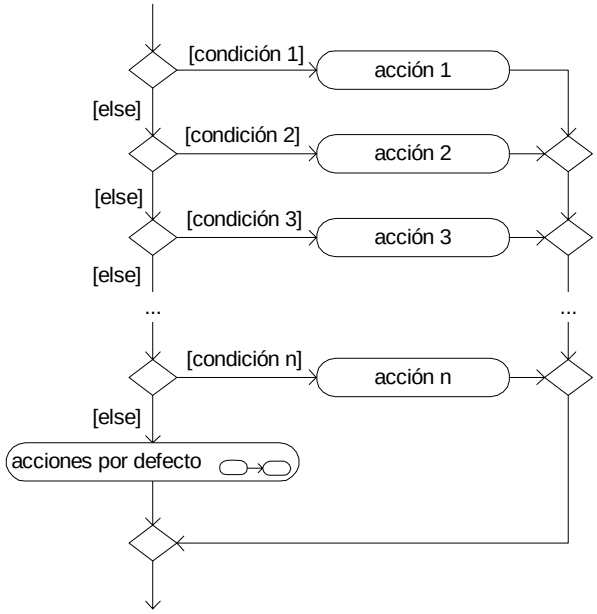
Finalmente compile el proyecto y haga correr el programa (paso a paso) con un número positivo, otro negativo y cero.

1.5.4.2. Estructura CASE

La estructura CASE tiene el diagrama de actividades que se muestra en la siguiente página. Recordemos que la forma de implementar esta estructura en PASCAL es la siguiente:

```
case valor o expresión ordinal of
  caso 1: acción 1;
  caso 2: acción 2;
  caso 3: acción 3;
  . . .
  caso n: acción n;
else
  acciones por defecto;
end;
```

Que está limitada a valores o expresiones ordinales y donde los casos (o condiciones) sólo pueden ser igualdades o intervalos.



En *User RPN* la estructura *CASE* es más completa que en PASCAL pues las condiciones pueden ser cualquier expresión lógica y no están limitadas únicamente a igualdades, intervalos y valores ordinales. La forma de implementar esta estructura en *User RPN* es la siguiente:

```
CASE
  condición 1 THEN acción 1 END
  condición 2 THEN acción 2 END
  condición 3 THEN acción 3 END
  . . .
  condición n THEN acción n END
  acciones por defecto
END
```

Donde *acción 1*, *acción 2*, etc., pueden ser una o más sentencias y no necesitan estar en una secuencia (« »). Al igual que en PASCAL, las acciones por defecto son opcionales.

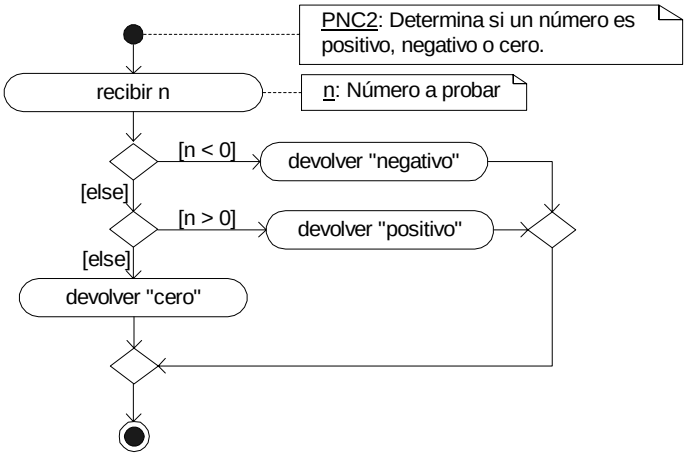
Como ejemplo volvamos a resolver el problema de determinar si un número es positivo, negativo o cero, pero ahora empleando la estructura *CASE*. El diagrama de actividades (PNC2), que en cuanto a su lógica es igual al elaborado para la estructura IF-THEN-ELSE, ha sido modificado de manera que se identifique mejor la estructura CASE y se presenta en la siguiente página.

La implementación de este diagrama en *User RPN* es la siguiente:

```
« → n
```

```
  « CASE
      n 0 < THEN "NEGATIVO" END
      n 0 > THEN "POSITIVO" END
      "CERO"
    END
  »
»
```

Guarde este programa con el nombre *"PNC2"* (en el directorio PRAC1) y haga correr el mismo (paso a paso) con un número positivo, otro negativo y cero.



En *System RPN*, la estructura *CASE* debe ser implementada dentro de una secuencia de la siguiente manera:

```
::
  condición 1 case acción 1
  condición 2 case acción 2
  condición 3 case acción 3
  . . .
  condición n case acción n
  acciones por defecto
;
```

Donde si *acción 1, acción 2, etc.*, constan de más de una sentencia deben estar encerradas dentro de una secuencia.

Resolvamos el problema de determinar si un número es positivo, negativo o cero empleando la estructura *case* de *System*. Para ello cree un proyecto con el nombre *"PNC2"*, dé al programa el nombre *"pnc2"*, cree el archivo *"PNC2.S"* y escriba el siguiente código en la ventana de edición.

```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  ::
    1LAMBIND
    ::
      1GETLAM %0< case "NEGATIVO"
      1GETLAM %0> case "POSITIVO"
      "CERO"
    ;
    ABND
  ;
;
```

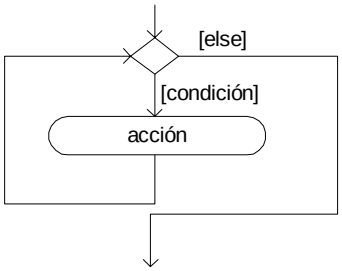
Compile el programa y haga correr el mismo (paso a paso) con un número positivo, uno negativo y cero. Al hacer correr el programa paso a paso, para ingresar a la estructura ***case*** debe emplear el botón ***"Step in"***.

1.6. Iteración

Tanto en ***User*** como en ***System*** se pueden implementar las tres estructuras iterativas estándar de PASCAL: While, Until y For. Además en ***System*** existe un comando especializado que permite crear una estructura infinita.

1.6.1. Estructura While

La estructura ***While (mientras)*** tiene el siguiente diagrama de actividades:



Recordemos que en esta estructura la acción se repite ***mientras*** la condición sea verdadera y termina cuando es falsa. En PASCAL esta estructura se implementa de la siguiente manera:

while *condición* ***do*** *acción;*

Donde si la ***acción*** involucra más de una sentencia se emplea una secuencia *(begin – end)*.

En ***User RPN*** la estructura ***while*** se implementa de la siguiente manera:

```
WHILE condición
REPEAT
  acción
END
```

Donde la acción puede ser una o más sentencias y no se requiere una secuencia (pues el final está señalado por la palabra ***END***).

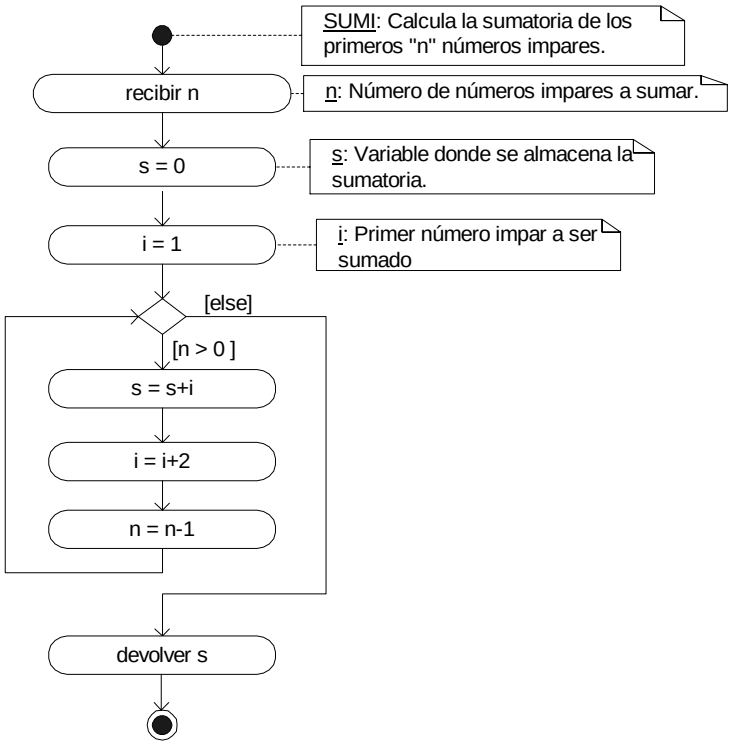
Como ejemplo elaboraremos un programa que calcule la sumatoria de los primeros "n" números impares. El diagrama de actividades (algoritmo) para resolver este problema se presenta en la siguiente página (SUMI).

El código que implementa el algoritmo en ***User RPN*** es el siguiente:

```
« 0 1 → n s i
 « WHILE n 0 >
   REPEAT
     i 's' STO+
     2 'i' STO+
     'n' 1 STO-
   END
   s
 »
»
```

Donde los comandos STO+, STO- (así como STO* y STO/) realizan la operación asociada y almacenan el resultado en la variable respectiva. Así por

ejemplo "i 's' STO+" es equivalente a " i s + 's' STO", "'n' 1 STO-" es equivalente a " n 1 - 'n' STO". Guarde el programa con el nombres ***"SUMI"*** (en el directorio PRAC1) y haga correr el mismo (paso a paso) con el número 5, como resultado obtendrá el número 25.



En ***System RPN*** la estructura ***while*** se implementa de la siguiente manera:

BEGIN
condición
WHILE
acción
REPEAT

Donde si la ***acción*** involucra más de una sentencia, las mismas deben encontrarse en una secuencia (:: ;). DEBUG4X coloca automáticamente el secundario, de manera que cuando se programa en DEBUG4X no se debe escribir manualmente la secuencia.

Como ejemplo implementaremos el algoritmo que suma números impares (SUMI) en ***System RPN***. Para ello cree el proyecto ***"SUMI"***, dé al programa el nombre ***"sumi"***, cree el archivo ***"SUMI.S"*** y escriba el siguiente código en la ventana de edición.

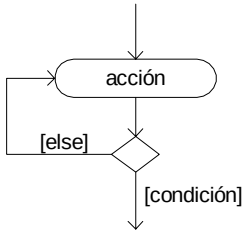
```
::
  CK1NOLASTWD
  CK&DISPATCH0
  ONE
  ::
    %0 %1
    {{ n s i }}
    BEGIN
      n %0 %>
    WHILE
      s i %+ !s
      i %2 %+ !i
```

```
      n %1 %- !n
    REPEAT
    s
    ABND
  ;
;
```

Compile el proyecto y haga correr el programa (paso a paso) con el número 5. Si todo está bien obtendrá como resultado el número 25.

1.6.2. Estructura UNTIL

La estructura ***Until (hasta)*** tiene el siguiente diagrama de actividades:



Recordemos que en esta estructura la ***acción*** se repite ***hasta*** que la ***condición*** es verdadera y que se prefiere esta estructura (en lugar de ***WHILE***) cuando el proceso debe repetirse al menos una vez. En PASCAL esta estructura se implementa de la siguiente manera:

```
repeat
  acción
until condición;
```

Donde la ***acción*** puede constar de una o más sentencias sin que sea necesario emplear una secuencia ***(begin - end)***.

En ***User RPN*** esta estructura se implementa de la siguiente manera:

```
DO
  acción
UNTIL
  Condición
END
```

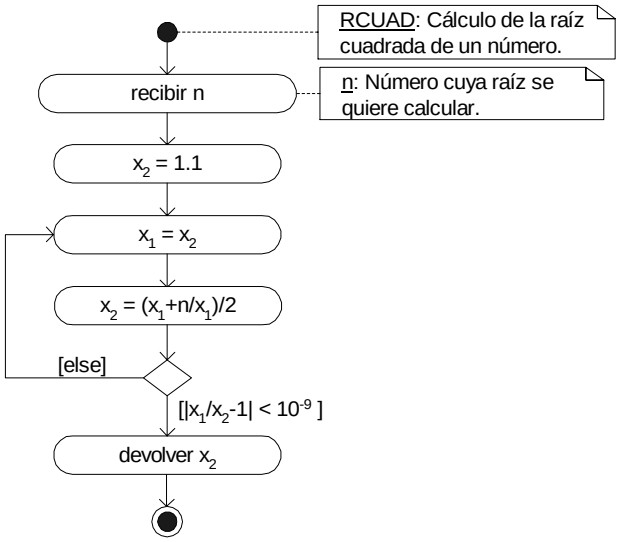
Donde ***acción***, al igual que en PASCAL puede constar de una o más sentencias y no se requiere una secuencia (« »).

Como ejemplo elaboraremos un programa que calcule la raíz cuadrada de un número "n" empleando la ecuación de ***Newton***:

$$x_2 = \frac{1}{2}\left(x_1 + \frac{n}{x_1} \right)$$

Para resolver esta ecuación se asume un valor inicial de x_1, usualmente 1.1 y con ese valor se calcula x_2, si x_1 y x_2 son aproximadamente iguales el proceso concluye, siendo la raíz cuadrada valor x_2, caso contrario x_1 toma el valor de x_2 y el proceso se repite ***hasta*** que los dos valores son aproximadamente iguales.

El diagrama de actividades (el algoritmo) para la resolución de este problema se presenta en la siguiente página y en el mismo la condición: $|x_1/x_2-1|<10^{-9}]$, que se explicará posteriormente, es verdadera si x_1 y x_2 son iguales en los primeros 9 dígitos.



La implementación del algoritmo en ***User RPN*** es la siguiente:

```
« 1.1 0 → n x2 x1
  « DO
      x2 'x1' STO
      x1 n x1 / + 2 / 'x2' STO
    UNTIL
      x1 x2 / 1 - ABS 1E-9 <
    END
    x2
  »
»
```

Guarde el programa con el nombre ***RCUAD*** (en el directorio ***PRAC1***) y haga correr el mismo (paso a paso) con el número 2. Entonces si no ha cometido ningún error obtendrá el resultado: 1.41421356238, que es la raíz cuadrada del número 2 con 9 dígitos de exactitud. Cabe aclarar que los programas elaborados hasta el momento son ideales, pues no toman en cuenta aspectos que si deben ser considerados en un programa real, por ejemplo en el cálculo de la raíz cuadrada se debe verificar que el número sea positivo y en caso contrario generar un error, asimismo no se deben realizar los cálculos si el número es cero o uno, pues para los mismos ya se conoce la respuesta.

En ***System RPN*** la estructura ***UNTIL*** se implementa de la siguiente manera:

```
BEGIN
  acción
  condición
UNTIL
```

Donde, al igual que en PASCAL, ***acción*** puede constar de una o más sentencias y no requiere estar dentro de una secuencia (:: ;). Como ejemplo implementemos el algoritmo que calcula la raíz cuadrada en ***System RPN***. Para ello cree el proyecto ***"RCUAD"***, dé al programa el nombre ***"rcuad"*** (en minúsculas), cree el archivo ***"RCUAD.S"*** y en la ventana de edición escriba el siguiente código:

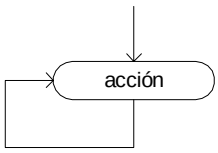
```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  ::
```

```
    % 1.1 %0
    {{ n x2 x1 }}
    BEGIN
      x2 !x1
      x1 n x1 %/ %+ %2 %/ !x2
      x1 x2 %/ %1- %ABS % 1E-9 %<
    UNTIL
    x2
    ABND
  ;
;
```

Compile el proyecto y haga correr el programa (paso a paso) con el número 2, entonces deberá obtener el resultado: 1.41421356238.

1.6.3. Ciclo infinito

Un ciclo infinito es aquel que se repite por siempre. El diagrama de actividades de un ciclo infinito es el siguiente:



Por supuesto en programación un ciclo infinito como tal no tiene utilidad práctica, por el contrario es uno de los errores que se debe evitar al momento de escribir un programa. Para que tenga utilidad práctica un ciclo infinito debe contar con alguna condición de finalización, pero la misma no está al principio (como en ***WHILE***) ni al final (como en ***UNTIL***) sino en alguna parte intermedia del ciclo.

En PASCAL no existe un comando específico para crear ciclos infinitos, por lo que los mismos se implementan con la estructura ***WHILE*** o ***UNTIL*** escribiendo en lugar de la condición un valor lógico verdadero (true) o falso (false) de manera que el ciclo se repita siempre. Así con la estructura ***WHILE*** el ciclo infinito se crea de la siguiente manera:

***while** TRUE **do** acción;*

Y con la estructura ***UNTIL*** de la siguiente manera:

***repeat** acción **until** FALSE;*

Para salir de un ciclo infinito en PASCAL se utilizan los comandos ***break***, ***exit*** o ***goto***.

En ***User RPN***, al igual que en PASCAL, no existe un comando para crear ciclos infinitos, por lo que se emplean las estructuras ***WHILE*** o ***UNTIL***, sin embargo como en ***User RPN*** no existen comandos equivalentes a ***break, exit*** o ***goto***, en lugar de la condición se coloca en la pila el valor lógico verdadero (1) o falso (0). El valor lógico se coloca generalmente mediante una estructura ***IF-THEN-ELSE*** ubicada en el interior del ciclo. Así con la estructura ***WHILE*** el ciclo infinito se crea de la siguiente manera:

```
1
WHILE
REPEAT
  acciones1
  IF condición
  THEN acciones2 0
```

```
  ELSE acciones3 1
  END
END
```

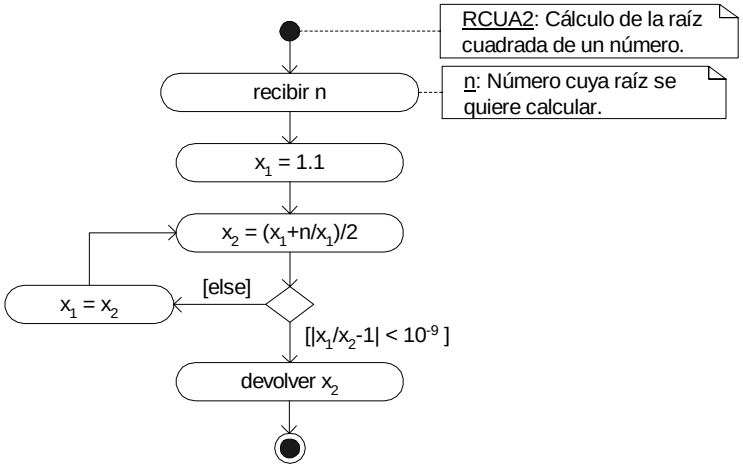
Donde el número 1 (verdadero) antes del ciclo se coloca para ingresar por primera vez al ciclo. Si la condición es verdadera se ejecutan las ***acciones2*** y para salir del ciclo se coloca en la pila el número 0 (falso), caso contrario se ejecutan las ***acciones3*** y para repetir el ciclo se coloca en la pila el número 1 (verdadero).

Con la estructura ***UNTIL*** el ciclo infinito se crea de la siguiente manera:

```
DO
  acciones1
  IF condición
  THEN acciones2 1
  ELSE acciones3 0
  END
UNTIL
END
```

En este caso si la condición es verdadera se ejecutan las ***acciones2*** y para salir del ciclo se coloca en la pila el número 1 (verdadero), caso contrario se ejecutan las ***acciones3*** y para repetir el ciclo se coloca en la pila el número 0 (falso).

Como ejemplo volveremos a resolver el problema de calcular la raíz cuadrada de un número real, pero en este caso seguiremos la lógica natural o directa que se muestra en el siguiente diagrama de actividades:



Este algoritmo sigue el razonamiento que naturalmente se seguiría para resolver el problema: se asume un valor para x_1, se calcula x_2, se compara x_1 con x_2 y si son aproximadamente iguales el proceso concluye, caso contrario x_1 toma el valor de x_2 y el proceso se repite.

La implementación de este algoritmo en ***User RPN*** empleando la estructura ***WHILE*** es la siguiente:

```
« 1.1 0 → n x1 x2
  « 1
    WHILE
    REPEAT
      x1 n x1 / + 2 / 'x2' STO
      IF x1 x2 / 1 - ABS 1E-9 <
      THEN x2 0
      ELSE x2 'x1' STO 1
```

```
      END
    END
  »
»
```

Guarde este programa con el nombre *"RCUA2"* y haga correr el mismo (paso a paso) con el número 2, si todo está correcto obtendrá el resultado: 1.41421356238.

La implementación de este programa empleando la estructura ***UNTIL*** es la siguiente:

```
« 1.1 0 → n x1 x2
  « DO
      x1 n x1 / + 2 / 'x2' STO
      IF x1 x2 / 1 - ABS 1E-9 <
      THEN x2 1
      ELSE x2 'x1' STO 0
      END
    UNTIL
    END
  »
»
```

Guarde este programa con el nombre *"RCUA3"* y haga correr el mismo (paso a paso) con el número 2 (resultado: 1.41421356238).

En ***System RPN***, se pueden implementar ciclos infinitos con las estructuras ***WHILE*** o ***UNTIL***, al igual que en ***User***, pero además en ***System*** existe una estructura específica, la estructura ***AGAIN***, que tiene la siguiente sintaxis:

```
BEGIN
  acciones
AGAIN
```

Lamentablemente no existe un comando específico para salir de este ciclo, por lo que para salir del mismo se debe manipular directamente la pila. Debido a esto, generalmente esta estructura se coloca dentro de una secuencia, y para salir de la misma se emplea un formato similar al siguiente:

```
::
BEGIN
  acciones1
  condición IT :: acciones2 2RDROP;
  acciones3
AGAIN
;
```

Donde si la ***condición*** es verdadera se ejecutan las ***acciones2*** y el programa continúa después de la secuencia (después del ;).

Como ejemplo implementaremos el programa que calcula la raíz cuadrada. Para ello cree el proyecto *"RCUA2"*, dé al programa el nombre *"rcua2"* y cree el archivo *"RCUA2.S"*, entonces en la ventana de edición escriba el siguiente código:

```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  ::
    % 1.1 %0
    {{ n x1 x2 }}
```

```
    ::
    BEGIN
      x1 n x1 %/ %+ %2 %/ !x2
      x1 x2 %/ %1- %ABS % 1E-9 %<
      IT :: x2 2RDROP ;
      x2 !x1
    AGAIN
    ;
    ABND
  ;
;
```

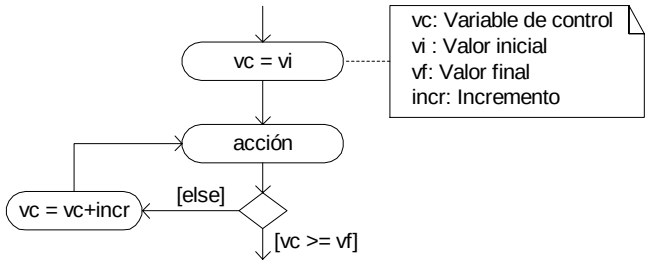
Compile el proyecto y haga correr el mismo (paso a paso) con el número 2, (resultado: 1.41421356238). Como casi siempre ocurre existe más de una forma de elaborar un programa, así el anterior programa también puede ser escrito de la siguiente manera:

```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  ::
    % 1.1 %0
    {{ n x1 x2 }}
    ::
    BEGIN
      x1 n x1 %/ %+ %2 %/ !x2 x2
      x1 x2 %/ %1- %ABS % 1E-9 %<
      IT :: 2RDROP ;
      !x1
    AGAIN
    ;
    ABND
  ;
;
```

Donde simplemente se ahorra algo de memoria, al colocar *x2* en la pila, antes de la condición (haga la prueba).

1.7. Estructura FOR

La estructura FOR suele tener lógicas ligeramente diferentes según el lenguaje de programación. La lógica más frecuente es la que se muestra en el siguiente diagrama de actividades:



Donde el ciclo se repite, incrementando en cada repetición el valor de la variable de control, hasta que es mayor o igual a un determinado valor final. Generalmente existen dos maneras de implementar este ciclo, una incrementando el valor de la variable de control y otra disminuyendo el valor de la variable de control.

Como se sabe en PASCAL la variable de control (vc) sólo puede ser incrementada (o diminuída) de uno en uno y el ciclo termina cuando su valor es igual al valor final (vf). Además la estructura ***FOR*** de PASCAL realiza una comprobación previa, de manera que si el valor inicial de la variable de control es mayor al valor final el programa continúa después del ciclo. La implementación de esta estructura en PASCAL, cuando la variable se incrementa de uno en uno es:

for *vc:=vi* ***to*** *vf* ***do*** *acción;*

Y cuando disminuye de uno en uno:

for *vc:=vi* ***downto*** *vf* ***do*** *acción;*

La estructura ***FOR*** de ***User RPN,*** a diferencia de PASCAL, no realiza una comprobación previa al ciclo, de manera que el mismo se ejecuta al menos una vez. La implementación de esta estructura cuando la variable de control se incrementa de uno en uno es la siguiente:

vi vf ***FOR*** *vc*
  acción
NEXT

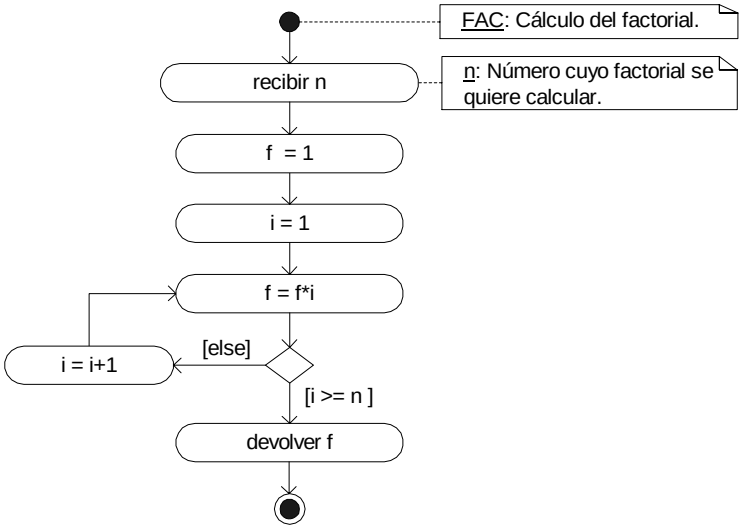
Donde ***acción*** puede ser una o más sentencias y no se requiere de una secuencia (pues termina con ***NEXT***). A diferencia de PASCAL la variable de control en ***User RPN*** es una variable real y en consecuencia puede tomar valores fraccionarios.

Cuando la variable de control incrementa con valores diferentes a uno, o disminuye en lugar de incrementar, se implementa de la siguiente forma:

vi vf ***FOR*** *vc*
  acción
  incr
STEP

Donde ***incr*** es un número real positivo o negativo.

Como ejemplo elaboraremos un programa que calcule el factorial ***(n!=1*2*3*…*n).*** La lógica para resolver este problema se presenta en el siguiente diagrama de actividades:



La implementación de este algoritmo en ***User RPN*** es la siguiente:

« 1 → n f

```
  « 1 n FOR i
      'f' i STO*
    NEXT
    f
  »
»
```

Guarde este programa con el nombre ***"FAC"*** y haga correr el mismo con el número 5 (paso a paso). Obtendrá como resultado el número 120.

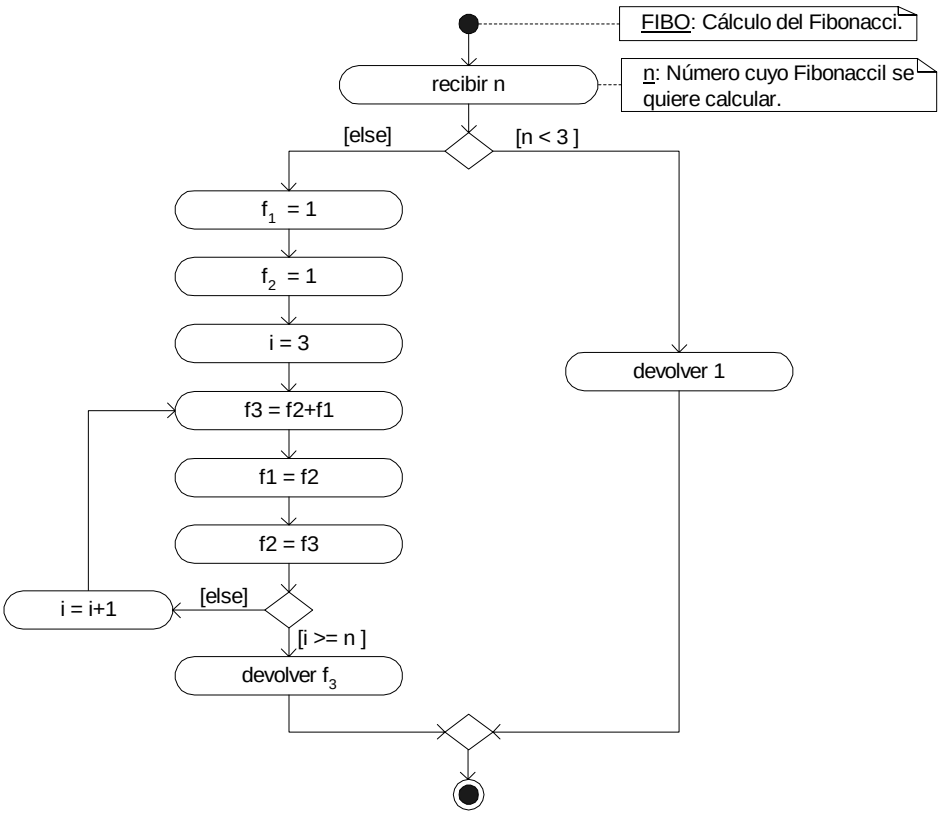
Cuando no se requiere el valor de la variable de control dentro del ciclo, la estructura ***FOR*** puede ser implementada de la siguiente manera:

```
vi vf START
  acción
NEXT
```

Y cuando el incremento es diferente a uno con:

```
vi vf START
  acción
  incr
STEP
```

Como ejemplo elaboraremos un programa para calcular el Fibonacci. El Fibonacci de un número "n" se calcula con la ecuación: $F_n = F_{n-1}+F_{n-2}$, donde por definición $F_1 = F_2 = 1$. El algoritmo para resolver este problema es el siguiente:



Donde como se puede ver el valor de la variable de control ***(i)*** no se utiliza realmente dentro del ciclo, por lo tanto resulta más eficiente emplear ***START*** en lugar de ***FOR***:

```
« 1 1 0 → n f1 f2 f3
```

```
  « IF n 3 <
    THEN 1
    ELSE
      3 n START
        f2 f1 + 'f3' STO
        f2 'f1' STO
        f3 'f2' STO
      NEXT
      f3
    END
  »
»
```

Guarde este programa con el nombre *"FIBO"* y haga correr el mismo con el número 8. Si no ha cometido ningún error obtendrá el resultado 21.

En *System RPN* la implementación de la estructura *FOR,* cuando la variable de control *(vc)* incrementa de uno en uno es la siguiente:

```
vf+1 vi DO
  acción
LOOP
```

Donde *acción* puede ser una o más sentencias y no se requiere una secuencia (:: ;). Observe que al valor final *(vf)* se le debe sumar 1, pues en *System* el ciclo se repite desde el valor inicial *(vi)* hasta el valor final menos uno *(vf-1)*. Tome en cuenta también que en *System* primero se escribe el valor final y luego el valor inicial (al revés de la implementación en *User*).

En la implementación de *System* la variable de control *(vc)* es creada automáticamente y es una variable de tipo entero binario (BINT). El valor de la variable de control se recupera con *INDEX@* y se le puede asignar un nuevo valor con *INDEXSTO*. Es posible también recuperar el valor final del ciclo *(vf)* con *ISTOP@* y asignarle un nuevo valor con *ISTOPSTO*.

Cuando existen dos estructuras *FOR* anidadas, la variable de control del ciclo externo se recupera con *JINDEX@* y se le puede asignar un nuevo valor con *JINDEXSTO.* Igualmente el valor final del ciclo externo se recupera con *JSTOP@* y se puede guardar un nuevo valor final con *JSTOPSTO*.

Cuando el incremento *(incr)* de la variable de control es diferente de uno, se implementa de la siguiente forma:

```
vf+1 vi DO
  acción
  incr
+LOOP
```

Donde el incremento *(incr*) debe ser un número entero binario *(BINT).*

Como primer ejemplo implementaremos el algoritmo para calcular el factorial *(FAC).* Para ello cree el proyecto *FAC*, dé al programa el nombre *fact*, cree el archivo *FAC.S* y en la ventana de edición escriba el siguiente código:

```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  ::
    %1
    {{ n f }}
```

```
    n COERCE #1+
    ONE_DO
      f INDEX@ UNCOERCE %* !f
    LOOP
    f
    ABND
  ;
;
```

En este programa ***COERCE*** convierte un número real ***(n)*** en un entero binario, #1+ suma uno al resultado de ***COERCE, ONE_DO*** coloca el entero binario ***ONE*** y luego inicia el ciclo con ***DO***, finalmente ***UNCOERCE*** convierte un entero binario ***(INDEX@)*** en un número real. Como ya se dijo en ***System RPN*** muchos de los comandos son simplemente combinaciones de dos o más comandos cuyo único propósito es el de ahorrar memoria, así en este caso en lugar de emplear: ONE #+ ONE DO, que son cuatro comandos, se puede emplear #1+ ONE_DO, que son dos comandos, o inclusive: #1+_ONE_DO, que es un solo comando y que en consecuencia ocupa menos memoria (haga la prueba).

Al compilar el proyecto, debido a un error de ***Debug4x***, aparecerá un mensaje de error indicándole que no existe el ***DO*** para ***LOOP***. En este caso ignore el error y haga correr el programa, paso a paso, con el número 5, entonces deberá obtener el resultado 120.

Como segundo ejemplo implementaremos el algoritmo para calcular el Fibonacci ***(FIBO).*** Para ello cree el proyecto ***FIBO***, dé al programa el nombre ***fibo,*** cree el archivo ***FIBO.S*** y en la ventana de edición escriba el siguiente código:

```
::
  CK1NOLASTWD
  CK&DISPATCH1
  ONE
  ::
    %1 %1 %0
    {{ n f1 f2 f3 }}
    n 3 %<
    ITE
    %1
    ::
      n COERCE #1+ THREE DO
        f2 f1 %+ !f3
        f2 !f1
        f3 !f2
      LOOP
      f3
    ;
    ABND
  ;
;
```

Compile el proyecto y haga correr el programa con el número 8, entonces deberá obtener el resultado 21.

1.7.1. Preguntas y ejercicios

Al igual que antes, los ejemplos y los ejercicios que se dan al final de las preguntas deben ser presentados en un CD. Para este fin, una vez realizados los ejemplos y ejercicios, guarde una copia de la memoria del emulador con el nombre “PR1Bnn_nnn”; donde “nn_nnn” es el número de su carnet uni-

versitario. Entonces, en el CD cree el directorio PRAC1 y dentro de él copie el archivo PR1nn_nnn así como todos los archivos que se encuentran en el directorio RPN\PRAC1 de su disco duro.

21. En la estructura IF-THEN-ELSE de ***User RPN*** ¿Las sentencias deben estar dentro de una secuencia?

22. En la estructura IF-THEN-ELSE de ***System RPN*** ¿Las sentencias deben estar dentro de una secuencia?

23. En la estructura CASE de ***User RPN*** ¿Las condiciones tienen alguna limitante?

24. En la estructura CASE de ***System RPN*** ¿Las acciones deben estar dentro de una secuencia?

25. ¿La estructura ***WHILE*** se repite mientras la condición es falsa?

26. ¿La estructura ***HASTA*** se repite hasta que la condición sea falsa?

27. ¿En ***User RPN*** existe un comando específico para crear ciclos infinitos?

28. ¿Cómo se sale de un ciclo infinito en ***System RPN?***

29. En la estructura ***FOR*** de ***User RPN*** ¿Si el valor inicial es mayor al valor final se ejecuta el ciclo?

30. En la estructura ***FOR*** de ***User RPN*** ¿Cuándo se emplea la palabra ***START*** en lugar de ***FOR?***

31. En la estructura ***FOR*** de ***System RPN*** ¿La variable de control puede ser un número real?

32. Elabore un programa en ***User RPN*** que empleando la estructura ***IF-THEN-ELSE*** determine si un número es par, impar o cero. Guarde el programa con el nombre PIC.

33. Resuelva el ejercicio anterior en ***System RPN***. Para ello elabore un proyecto con el nombre PIC y dé al programa el nombre pic.

34. Elabore un programa en ***User RPN*** que empleando la estructura ***CASE*** determine si un número es par, impar o cero. Guarde el programa con el nombre ***PIC2***.

35. Resuelva el ejercicio anterior en ***System RPN***. Para ello elabore un proyecto con el nombre ***PIC2*** y dé al programa el nombre ***pic2***.

36. Elabore un programa en ***User RPN*** que empleando la estructura ***WHILE*** calcule la productoria de los primeros "n" números pares. Guarde el programa con el nombre ***PROP.***

37. Resuelva el ejercicio anterior en ***System RPN.*** Para ello elabore un proyecto con el nombre ***PROP*** y dé al programa el nombre ***prop***.

38. Elabore un programa en ***User RPN*** que empleando la estructura ***UNTIL*** calcule la raíz cúbica de un número "n" empleando la ecuación de Newton: (guarde el programa con el nombre ***RCUB***)

$$x_2 = \frac{1}{3}\left(2x_1 + \frac{n}{x_1^2}\right)$$

39. Resuelva el ejercicio anterior en ***System RPN.*** Para ello elabore un proyecto con el nombre ***RCUB*** y dé al programa el nombre ***rcub***.

40. Elabore un programa en ***User RPN*** que resuelva el ejercicio 38, empleando un ciclo infinito implementado con la estructura ***WHILE*** (guarde el programa con el nombre ***RCUB2***).

41. Elabore un programa en ***User RPN*** que resuelva el ejercicio 38, empleando un ciclo infinito implementado con la estructura ***UNTIL*** (guarde el programa con el nombre ***RCUB3***).

42. Elabore un programa en ***System RPN*** que resuelva el ejercicio 38, empleando un ciclo infinito (cree el proyecto ***RCUB2*** y guarde el programa con el nombre ***rcub2***).

43. Elabore un programa en ***User RPN*** que empleando la estructura ***FOR*** calcule la sumatoria de los primeros "n" números enteros: $\sum_{i=1}^{n} i$ (guarde el programa con el nombre SUMN).

44. Elabore un programa en ***User RPN*** que empleando la estructura ***FOR*** (implementada con ***START***), calcule la potencia entera de un número real: $x^n = \prod_{i=1}^{n} x = x * x * \ldots * x \quad (n \text{ veces})$. Guarde el programa con el nombre ***POTE***.

45. Resuelva el ejercicio 43 empleando la estructura ***FOR*** de ***System RPN*** (cree el proyecto ***SUMN*** y guarde el programa con el nombre ***sumn***).

46. Resuelva el ejercicio 44 empleando la estructura ***FOR*** de ***System RPN*** (cree el proyecto POTE y guarde el programa con el nombre ***pote)***.

10. INTERPOLACIÓN

Cuando se tiene una serie de datos tabulados sobre un cierto rango como los siguientes:

x	0	0.2	0.4	0.6	0.8	1.0
y	0	0.199	0.389	0.565	0.717	0.841

Donde en la fila *"x"* están los valores correspondientes a la variable independiente y en la fila *"y"* los valores de la variable dependiente (es decir *"y"* es una función de *"x"*), con frecuencia es necesario estimar los valores de *"y"* para valores conocidos de *"x"*. En general este problema puede ser resuelto mediante uno de los siguientes procedimientos: a) Asumir que todos los datos pueden ser representados por una sola ecuación analítica, calcular entonces (con los datos conocidos) las constantes y coeficientes de dicha ecuación y emplear la ecuación resultante para predecir los valores de la variable independiente y b) Asumir que cada cierto número de puntos se conectan entre si mediante una determinada forma geométrica (o ecuación analítica), calcular las constantes y coeficientes de esas ecuaciones y emplear dichas ecuaciones para predecir los valores de la variable independiente.

La primera forma corresponde al *ajuste de curvas* que estudiaremos en el siguiente capítulo y la segunda corresponde a la *interpolación* del cual estudiaremos tres métodos en el presente capítulo. Se emplea la *interpolación* cuando es necesario encontrar un determinado valor de *"y"* para algún valor intermedio de *"x"* no incluido en la tabla. El valor conocido de "x" debe estar comprendido en el rango de los valores tabulados, si no es así el problema de encontrar el valor de la función se conoce como *extrapolación* (para la cual existen otros métodos).

La interpolación se emplea cuando los datos con los que se cuenta son confiables. Si los datos no son confiables, entonces se debe optar por el ajuste de curvas.

La mayoría de los métodos de interpolación asumen que los datos se interconectan mediante una ecuación polinomial de grado N-1 (siendo N el número de pares de datos (x-y) conocidos):

$$y = a_1x^{n-1} + a_2x^{n-2} + a_3x^{n-3} + \ldots + a_{n-1}x + a_n$$

Los diferentes métodos existentes se diferencian unos de otros por la forma en que calculan los coeficientes de este polinomio.

10.1. INTERPOLACIÓN LINEAL

El método de interpolación lineal es el método de interpolación más sencillo conocido. A pesar de su simplicidad, es uno de los métodos más útiles en la práctica. Este método proporciona resultados satisfactorios cuando los datos con los que se cuenta están distribuidos en segmentos cercanos, o cuando los datos a pesar de estar espaciados describen aproximadamente una línea recta.

Otra de las ventajas de la interpolación lineal radica en que se emplea la misma ecuación (la ecuación de la línea recta) tanto para interpolar como para extrapolar.

En el método de interpolación lineal se busca el intervalo donde se encuentra el valor conocido de *"x"* y empleando los dos puntos de dicho intervalo se calculan los coeficientes de la ecuación de una línea recta $y = a + b*x$. Con los coeficientes *(a* y *b)* calculados se emplea la ecuación de la

línea recta y el valor conocido de “x” para calcular el valor desconocido de “y”.

Si el valor conocido de "x" es mayor al mayor valor tabulado de "x" (o menor al menor valor tabulado), entonces se puede emplear el último par de puntos (o el primer par de puntos) para efectuar una extrapolación lineal.

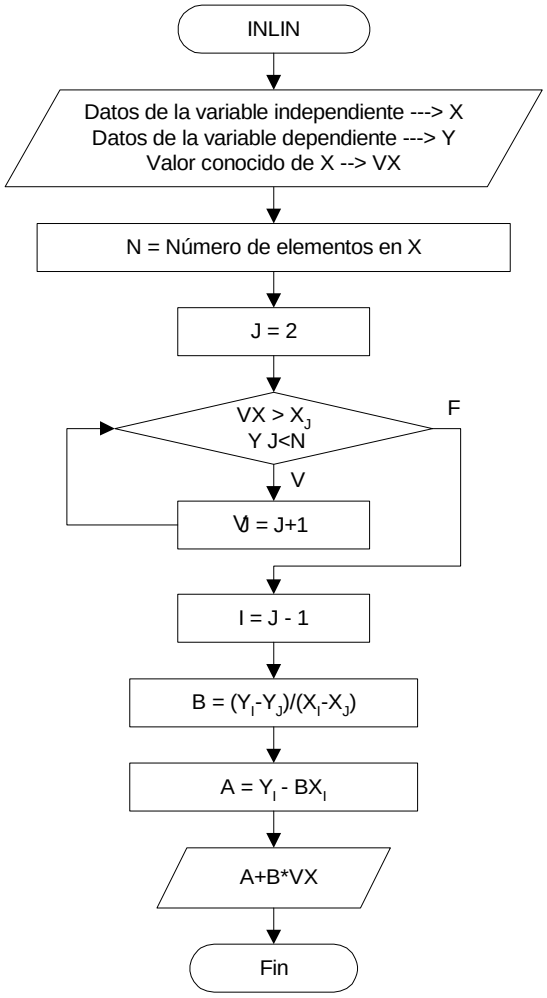
Si denominamos (x_1,y_1) y (x_2,y_2) a los dos puntos conocidos del intervalo, entonces los coeficientes de la línea recta pueden ser calculados con las ecuaciones:

$$b = \frac{y_1 - y_2}{x_1 - x_2} \quad (1)$$

$$a = y_1 - bx_1 \quad (2)$$

10.1.1. Algoritmo

El algoritmo elaborado se presente en la siguiente figura:



En el mismo en primer lugar se determina cual es el intervalo (o par de puntos) entre los cuales se encuentra el valor a interpolar, para ello en primera instancia se compara el valor a interpolar con el segundo valor conocido de *“x” (x_2)*. Si el valor a interpolar es menor o igual a este valor, entonces se emplea el primer par de puntos para efectuar la interpolación (o extrapolación), esto debido a que aún en el caso de que el valor a interpo-

lar sea menor al menor dato conocido *(x_1)*, de todas formas se tendría que emplear el primer par de puntos para efectuar la extrapolación.

Si el valor a interpolar es mayor a x_2, entonces se compara con x_3, luego con x_4, etc., si es menor o igual a alguno de estos datos, entonces el intervalo está comprendido entre ese punto y el punto anterior, así por ejemplo si es menor a x_5 se deberá interpolar entre (x_4,y_4) y (x_5,y_5). Como puede observar en el diagrama, sólo se compara hasta el dato *n-1*, pues si el valor a interpolar es mayor al penúltimo dato conocido, entonces de todas formas se tienen que emplear el último par de puntos (ya sea para interpolar o para extrapolar).

Una vez ubicado el intervalo (con la variable J), se calculan las constantes (A y B) y a con las mismas se calcula el valor interpolado *(y)*. Es importante notar que los valores de la variable independiente deben estar ordenados ascendentemente para que el proceso de búsqueda funcione.

El código elaborado en base al algoritmo es el siguiente:

```
« 0 0 2 0 0 → X Y VX N I J A B
  « X SIZE 1 GET 'N' STO
    WHILE VX X J GET > J N < AND
    REPEAT
      1 'J' STO+
    END
    J 1. - 'I' STO
    Y I GET Y J GET -
    X I GET X J GET - / 'B' STO
    Y I GET B X I GET * - 'A' STO
    A B VX * +
  »
»
```

Puesto que este método suele ser de uso frecuente puede ser de utilidad ganar algo de velocidad, por esta razón se ha elaborado la versión del programa que emplea sólo la pila (no variables).

```
« SWAP ARRY→ 1 GET 2 + ROLL ARRY→
  1 GET DUP 2 * 2 + ROLL 2
  WHILE
    DUP 4 PICK < 3 PICK 5 PICK 4 PICK -
    6 + PICK > AND
  REPEAT
    1 +
  END
  3 PICK 2 * 2 PICK - 6 + DUP
  PICK DUP 3 ROLL PICK -
  5 PICK 4 PICK - 8 + DUP PICK DUP 3 ROLL PICK -
  3 ROLL SWAP / DUP 4 ROLLD * - SWAP 4 ROLL * +
  3 ROLLD DROP 2 * DUP 2 + 3 ROLL SWAP ROLLD DROPN
»
```

Para que no se sobreponga al anterior programa, guarde este código con el nombre *INLIP* (Interpolación Lineal utilizando la Pila).

10.1.2. Ejemplo

Considerando los datos presentados al principio del capítulo calcule los valores de *"y"* para valores de *"x"* iguales a 0.3, 0.5, 0.7, 0.9 y 1.2 (Extrapolación en el último caso).

El programa que resuelve el problema es:

```
« 0 0 → X Y
  « [ 0 .2 .4 .6 .8 1 ] 'X' STO
    [ 0 .199 .389 .565 .717 .841 ] 'Y' STO
    X Y .3 INLIN
    X Y .5 INLIN
    X Y .7 INLIN
    X Y .9 INLIN
    X Y 1.2 INLIN
  »
»
```

Haciendo correr el programa se obtienen los resultados: 0.294, 0.477, 0.641, 0.779 y 0.965 que son los valores de "y" para los valores de "x": 0.3, 0.5, 0.7, 0.9 y 1.2 respectivamente.

10.1.3. Ejercicio

1. Con los siguientes datos:

x	3.5	4	4.5	5	5.5	6	6.2	6.5	6.8
y	0.002	0.030	0.155	0.396	0.658	0.903	0.933	0.975	0.993

   Calcule los valores de "y" para "x" igual a 3.9, 4.8, 6.1, 6.57, 6.7 y 7.1.

10.2. INTERPOLACIÓN DE LAGRANGE

El método de Lagrange es un método de interpolación polinomial, donde el valor interpolado se calcula mediante la siguiente ecuación:

$$y = \sum_{k=1}^{N} y_k L_k(x) \tag{3}$$

donde:

$$L_k = \frac{P_k(x)}{Q_k(x)} \qquad \{k = 1 \rightarrow N \tag{4}$$

$$P_k(x) = \prod_{\substack{i=1 \\ i \neq k}}^{N} \left(x - x_i\right) \tag{5}$$

$$Q_k(x_k) = \prod_{\substack{i=1 \\ i \neq k}}^{N} \left(x_k - x_i\right) \tag{6}$$

"N" es el número de datos conocidos, *"x"* es el valor a interpolar, *"x_i"* y *"y_i"* son los valores conocidos de las variables independiente y dependiente respectivamente.

Todos los valores presentados en las ecuaciones 1 a 4 tienen que ser calculados cada vez que se quiere iterar para un valor conocido de la variable independiente. Si bien las ecuaciones del método pueden ser programadas directamente, es más eficiente dividir el proceso en dos, calculando por separado los valores del vector "Q", pues estos valores sólo dependen de los valores tabulados para "x" y "y" y emplear luego estos valores para efectuar las interpolaciones que fueran necesarias. De esta manera es posible se re-

duce el número de cálculos cuando es necesario interpolar para varios valores de la variable independiente.

Es importante tomar en cuenta que el método de Lagrange no permite realizar interpolaciones cuando existen 2 o más valores repetidos de la variable independiente *(x)*, pues como se puede observar en la ecuación la productoria sería cero y en consecuencia se produciría una división entre cero. Al igual que en el método de interpolación lineal, los valores de la variable independiente deben estar ordenados ascendentemente.

10.2.1. Ejemplo manual

Para comprender mejor el método obtendremos por interpolación, con los datos de la siguiente tabla, los valores de la variable dependiente *"y"* para *"x=2"* y posteriormente para *"x= 3"*:

x	0	1	4	6
y	1	-1	1	-1

Empleando la ecuación 6.2.4, podemos calcular los valores de "Q" del método de Langrange:

$Q_1 = (x_1-x_2)*(x_1-x_3)*(x_1-x_4)=(0-1)*(0-4)*(0-6)= -24$
$Q_2 = (x_2-x_1)*(x_2-x_3)*(x_2-x_4)=(1-0)*(1-4)*(1-6)= 15$
$Q_3 = (x_3-x_1)*(x_3-x_2)*(x_3-x_4)=(4-0)*(4-1)*(4-6)= -24$
$Q_4 = (x_4-x_1)*(x_4-x_2)*(x_4-x_3)=(6-0)*(6-1)*(6-4)= 60$

Entonces los valores de *"P"* para *"x=2"* (ecuación 6.2.3) son:

$P_1 = (x-x_2)*(x-x_3)*(x-x_4)=(2-1)*(2-4)*(2-6)= 8$
$P_2 = (x-x_1)*(x-x_3)*(x-x_4)=(2-0)*(2-4)*(2-6)= 16$
$P_3 = (x-x_1)*(x-x_2)*(x-x_4)=(2-0)*(2-1)*(2-6)= -8$
$P_4 = (x-x_1)*(x-x_2)*(x-x_3)=(2-0)*(2-1)*(2-4)= -4$

Con los valores de *Q* y *P* se pueden calcular los valores de *"L"* (ecuación 6.2.2):

$L_1=P_1/Q_1= 8/(-24) = -0.333333333333$
$L_2=P_2/Q_2= 16/15 = 1.06666666667$
$L_3=P_3/Q_3= -8/(-24) = 0.333333333333$
$L_4=P_4/Q_4= -4/60 = -6.66666666667E-2$

Y con los valores de *"L"* obtenemos el valor de *"y"* para *"x=2"* aplicando la ecuación de Lagrange (ecuación 6.2.1):

$y = y_1*L_1+y_1*L_2+y_3*L_3+y_4*L_4$
$= 1*(-0.333333333333)+(-1)*(1.06666666667)+$
$1*(0.333333333333)+(-1)*(-6.66666666667E-2)$
$= \mathbf{-1}$

Ahora para *x=3*, sólo cambian los valores de *"P"*, pues los datos conocidos siguen siendo los mismos. Los nuevos valores de *"P"* son:

$P_1 = (x-x_2)*(x-x_3)*(x-x_4)=(3-1)*(3-4)*(3-6)= 6$
$P_2 = (x-x_1)*(x-x_3)*(x-x_4)=(3-0)*(3-4)*(3-6)= 9$
$P_3 = (x-x_1)*(x-x_2)*(x-x_4)=(3-0)*(3-1)*(3-6)= -18$
$P_4 = (x-x_1)*(x-x_2)*(x-x_3)=(3-0)*(3-1)*(3-4)= -6$

Y los nuevos valores de "L":

$L_1=P_1/Q_1= 6/(-24) = -0.25$
$L_2=P_2/Q_2= 9/15 = 0.6$
$L_3=P_3/Q_3= -18/(-24) = 0.75$
$L_4=P_4/Q_4= -6/60 = -0.1$

Por consiguiente el valor interpolado es:

$y = y_1*L_1+y_1*L_2+y_3*L_3+y_4*L_4$
$=1*(-0.25)+(-1)*(0.6)+1*(0.75)+(-1)*(-0.1)$
$= \mathbf{0}$

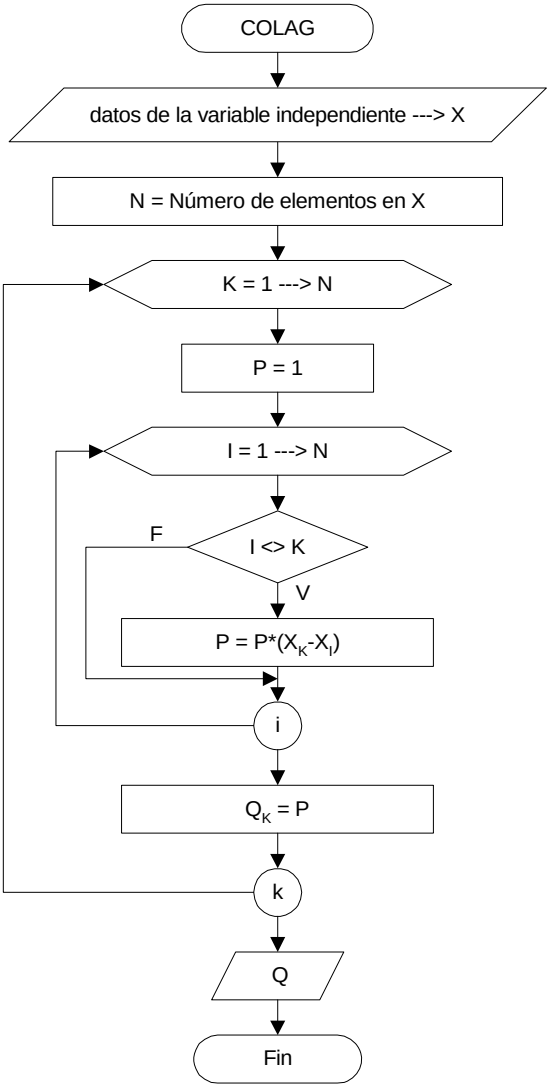
Como se puede observar en este ejemplo, mientras no se cambien los datos conocidos, los valores de *"Q"* permanecen constantes.

10.2.2. Algoritmo y código

Por las razones antes explicadas el proceso ha sido dividido en dos módulos: a) cálculo de los coeficientes *"Q"* y b) el módulo principal donde se efectúa interpolación propiamente.

10.2.2.1. Cálculo de los coeficientes Q

El algoritmo para el cálculo de los coeficientes *"Q"*, es el siguiente:



Y el código elaborado en base al mismo es:

```
« 0 0 → X P Q
  « X SIZE 1. GET 'N' STO
```

```
    X 'Q' STO
    1 N FOR K
      1 'P' STO
      1 N FOR I
        IF I K ‹ THEN
          X K GET X I GET - 'P' STO*
        END
      NEXT
      Q K P PUT 'Q' STO
    NEXT Q
  »
»
```

Haciendo correr este módulo con los datos del ejemplo manual:

[0 1 4 6]

Se obtiene el vector con los coeficientes “Q”:

[-24 15 -24 60]

Que son los mismos valores calculados manualmente.

10.2.2.2. Módulo principal

El algoritmo del módulo principal se presenta en el diagrama de flujo de la siguiente página y el código elaborado en base al mismo es:

```
« 1 0 0 → X Y Q VX P S N
  « X SIZE 1. GET 'N' STO
    1 N FOR K
      1 'P' STO
      1 N FOR I
        IF I K ‹THEN VX X I GET - 'P' STO*
        END
      NEXT
      Y K GET P * Q K GET / 'S' STO+
    NEXT
    S
  »
»
```

Haciendo correr el programa con los datos del ejemplo manual:

[0 1 4 6] [1 -1 1 -1] [-24 15 -24 60] 2

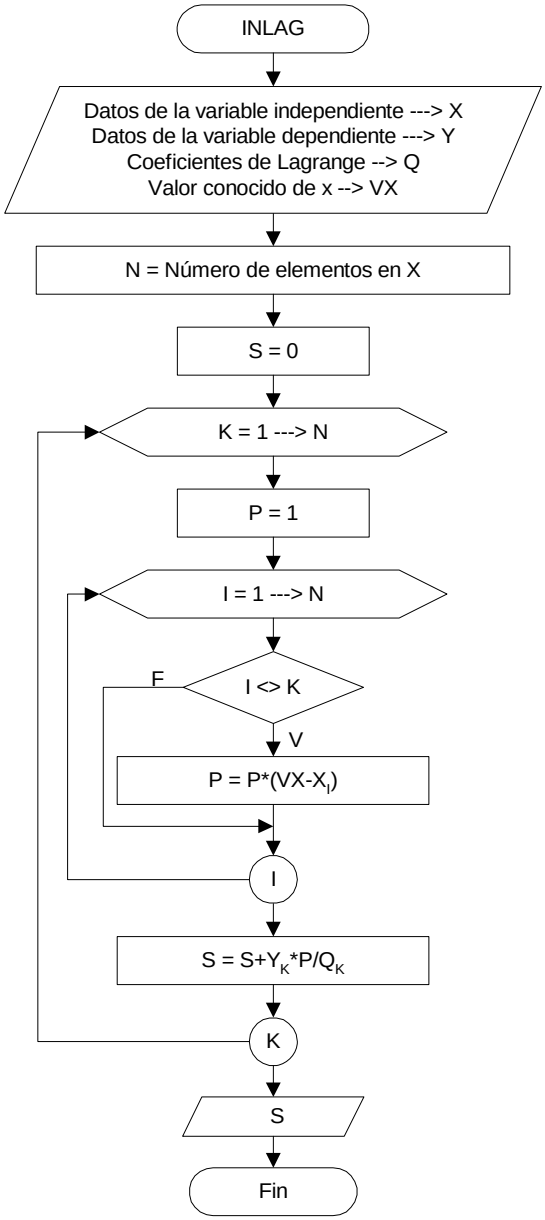
Se obtiene el resultado: -1, que es el mismo resultado obtenido en el ejemplo.

10.2.3. Ejemplo

Resuelva el ejemplo del método de interpolación lineal empleando el método de Lagrange.

El programa que resuelve el problema es:

```
« 0 0 0 → X Y Q
  « [ 0 .2 .4 .6 .8 1 ] 'X' STO
    [ 0 .199 .389 .565 .717 .841 ] 'Y' STO
    X COLAG 'Q' STO
    X Y Q .3 INLAG
    X Y Q .5 INLAG
```




```
    X Y Q .7 INLAG
    X Y Q .9 INLAG
    X Y Q 1.2 INLAG
  »
»
```

Haciendo correr el programa se obtiene los resultado: 0.29519140625 0.47938671875 0.64451953125 0.78221484375 y 0.95 que son los valores de "y" para los valores de "x": 0.3, 0.5, 0.7, 0.9 y 1.2 respectivamente.

10.2.4. Ejercicio

2. Resuelva el ejercicio 1 del método de interpolación lineal empleando el método de Lagrange.

10.3. INTERPOLACIÓN DE NEWTON

El método de Newton es también un método polinomial, en este caso el valor interpolado se calcula empleando la siguiente ecuación:

$$y = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2) + \ldots + b_n(x - x_1)(x - x_2)..(x - x_{n-1}) \quad (7)$$

Donde:

$b_1 = y_1$
$b_2 = f(x_1, x_2)$
$b_3 = f(x_1, x_2, x_3)$;
$b_4 = f(x_1, x_2, x_3, x_4)$;
...
$b_n = f(x_1, x_2, ..., x_n)$; (8)

$f(x_i, x_j) = (y_j - y_i)/(x_j - x_i)$
$f(x_i, x_j, x_k) = (f(x_j, x_k) - f(x_i, x_j))/(x_k - x_i)$
…
$f(x_i, x_j, \ldots, x_n) = (f(x_j, x_k, \ldots, x_n) - f(x_i, x_j, \ldots, x_{n-1}))/(x_n - x_i)$ (9)

"n" es el número de datos conocidos, *"x_i"* y *"y_i"* son los datos conocidos para la variable independiente y dependiente respectivamente y *"x"* es el valor a interpolar.

Como los coeficientes "b" sólo dependen de los valores conocidos pueden ser calculados por separado (tal como sucedió con el método de Lagrange). De esta manera es posible reducir el número de cálculos cuando es necesario interpolar para varios valores de la variable independiente.

Al igual que sucede con el método de Lagrange, el método de Newton no puede ser empleado cuando existen 2 o más valores repetidos de la variable independiente, pues como se puede observar en las ecuaciones 3, se produciría división entre cero. Una vez más es necesario que los valores conocidos de la variable independiente estén ordenados ascendentemente.

10.3.1. Ejemplo manual

Para comprender mejor el método interpolaremos (calcularemos valores de *"y"*), con los datos de la siguiente tabla, para *"x"* igual a 2 y luego para *"x"* igual a 5:

x	1	4	5	6
y	0	1.3862944	1.6094379	1.7917595

En el método de Newton debemos calcular primero los valores de las funciones (ecuación 6.3.3).

$f(x_1, x_2) = (y_2 - y_1)/(x_2 - x_1) = (1.3862944 - 0)/(4 - 1) = 0.462098133333$
$f(x_2, x_3) = (y_3 - y_2)/(x_3 - x_2) = (1.6094379 - 1.3862944)/(5 - 4) = 0.2231435$
$f(x_3, x_4) = (y_4 - y_3)/(x_4 - x_3) = (1.7917595 - 1.6094379)/(6 - 5) = 0.1823216$
$f(x_1, x_2, x_3) = (f(x_2, x_3) - f(x_1, x_2))/(x_3 - x_1) = (0.2231435 - 0.462098133333)/(5 - 1)$
$= -5.97386583332E\text{-}2$
$f(x_2, x_3, x_4) = (f(x_3, x_4) - f(x_2, x_3))/(x_4 - x_2) = (0.1823216 - 0.2231435)/(6 - 4)$
$= -0.02041095$
$f(x_1, x_2, x_3, x_4) = (f(x_2, x_3, x_4) - f(x_1, x_2, x_3))/(x_4 - x_1)$
$= (-0.02041095 + 5.97386583332E\text{-}2)/(6 - 1) = 7.86554166664E\text{-}3$

Con estos valores podemos calcular ahora los coeficientes *"b"* (ecuación 6.3.2):

$b_1 = y_1 = 0$;

$b_2 = f(x_1,x_2) = 0.462098133333;$
$b_3 = f(x_1,x_2,x_3) = -5.97386583332E-2;$
$b_4 = f(x_1,x_2,x_3,x_4) = 7.86554166664E-3$

Y con los coeficientes *"b"* y la ecuación de *Newton* (ecuación 6.3.1), obtenemos el valor (interpolado) para *x=2:*

$y = b_1+b_2*(2-1)+b_3*(2-1)*(2-4)+b_4*(2-1)*(2-4)*(2-5) = 0.628768699999$

Ahora para *x=5.5*, lo único que tenemos que hacer es volver a emplear la ecuación de *Newton*, pues los coeficientes no cambian mientras no cambien los datos conocidos.

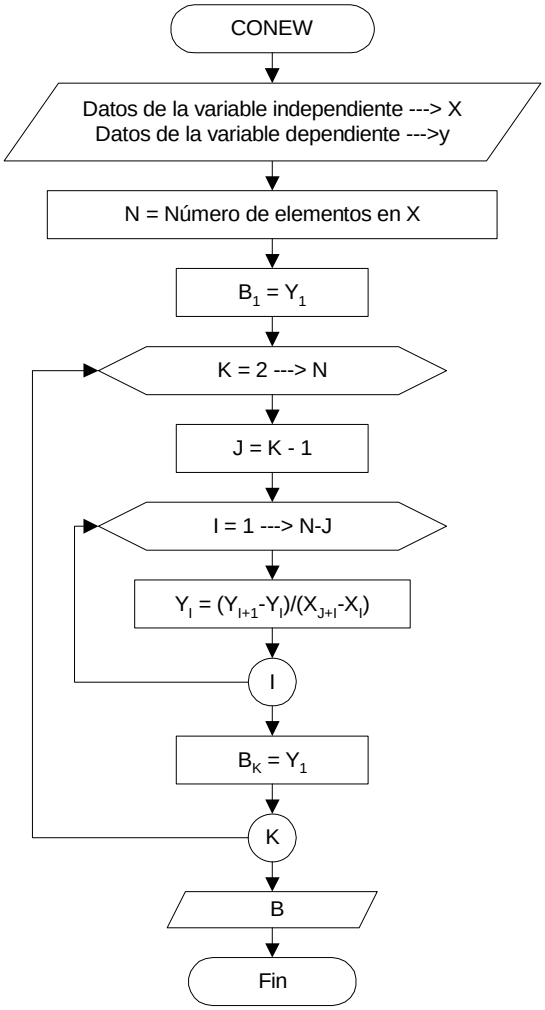
$y = b_1+b_2*(5.5-1)+b_3*(5.5-1)*(5.5-4)+b_4*(5.5-1)*(5.5-4)*(5.5-5)$
$= 1.70275185937$

10.3.2. Algoritmo y código

Por las razones antes explicadas, el algoritmo ha sido dividido en dos módulos: a) un módulo para el cálculo de los coeficientes *"b"* y b) el módulo principal donde se lleva a cabo la interpolación.

10.3.2.1. Cálculo de los coeficientes "b" del método de interpolación de Newton

El algoritmo que permite el cálculo de los coeficientes *"b"* es:



En este algoritmo, para ahorrar memoria y tiempo, se almacenan los valores de las funciones en el vector con los datos *"y"*. Esto es posible porque (como puede observar en el ejemplo manual) una vez que se emplea *"y_1"* para calcular *"f_1"*, no se vuelve a emplear más este valor de *"y_1"* en ninguno de los cálculos, lo mismo sucede con *"y_2"*, pues una vez calculado *"f2"*, no se vuelve a emplear más *"y_2"* y lo mismo es válido para *"y_3"* con *"f_3"*, *"y_4"* con *"f_4"*, etc. Por consiguiente en el algoritmo elaborado todos los valores de *"f_i"* quedan almacenados finalmente en *"y_i"*. Dado que *"b_1"* es igual a *"y_1"*, esta asignación se la efectúa al principio del programa y la función que contiene el valor de *"b_k"*, para cada iteración, es almacenada en esta variable.

El código elaborado en base al algoritmo es:

```
« 0 0 0 → X Y N B J
  « X SIZE 1 GET 'N' STO
    X 'B' STO
    B 1 Y 1 GET PUT 'B' STO
    2 N FOR K
      K 1 - 'J' STO
      Y
      1 N J - FOR I
        I
        Y I 1 + GET Y I GET -
        X J I + GET X I GET - / PUT
      NEXT
      'Y' STO
      B K Y 1 GET PUT 'B' STO
    NEXT
    B
  »
»
```

Haciendo correr este programa con los datos del ejemplo manual:

[1 4 5 6] [0 1.3862944 1.6094379 1.7917595]

Se obtiene el vector con los coeficientes *"b"*:

[0 0.462098133333 -5.97386583332E-2 7.86554166664E-3]

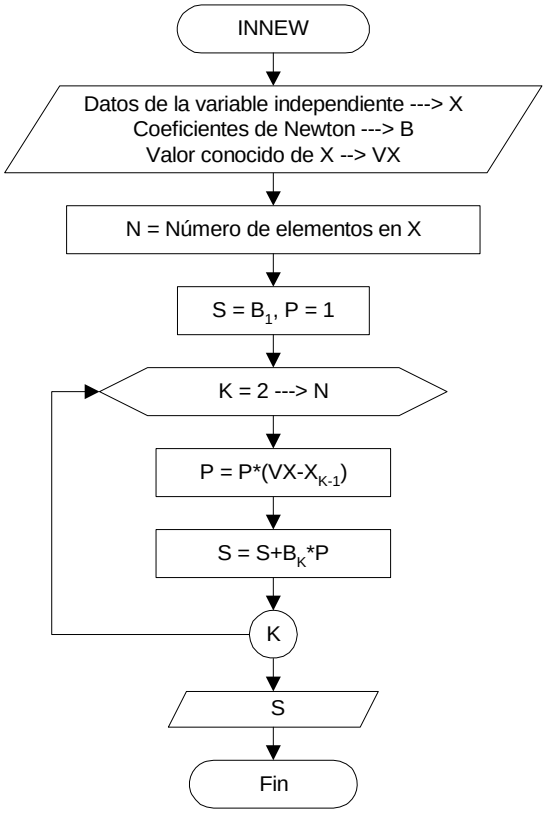
Que son los mismos resultados obtenidos manualmente.

1.1.1.1. Módulo principal

El algoritmo del módulo principal, donde se efectúa propiamente la interpolación, se presenta en el diagrama de flujo de la siguiente página y el código elaborado en base al mismo es el siguiente:

```
« 0 1 0 → X B VX S P N
  « X SIZE 1 GET 'N' STO
    B 1 GET 'S' STO
    2 N FOR K
      VX X K 1. - GET - 'P' STO*
      B K GET P * 'S' STO+
    NEXT
    S
  »
»
```

Haciendo correr este programa con los datos del ejemplo manual (para x=2) y los resultados obtenidos con el programa *CONEW:*




```
[1 4 5 6] [0 0.462098133333 -5.97386583332E-2 7.86554166664E-3] 2
```

Obtenemos el resultado: *0.628768699999*, que es el mismo resultado obtenido manualmente.

10.3.3. Ejemplo

Resuelva el ejemplo resuelto para el método de interpolación lineal empleando el método de Newton.

El programa que resuelve el problema es:

```
« 0 0 0 → X Y B
  « [ 0 .2 .4 .6 .8 1 ] 'X' STO
    [ 0 .199 .389 .565 .717 .841 ] 'Y' STO
    X Y CONEW 'B' STO
    X B .3 INNEW X B .5 INNEW
    X B .7 INNEW X B .9 INNEW
    X B 1.2 INNEW
  »
»
```

Haciendo correr el programa se obtiene los resultado: 0.29519140625 0.47938671875 0.64451953125 0.78221484375 y 0.950000000001 que son los valores de "y" para los valores de "x": 0.3, 0.5, 0.7, 0.9 y 1.2 respectivamente y que como se puede observar son esencialmente los mismos resultados obtenidos por el método de Lagrange.

10.3.4. Ejercicio

3. Resuelva los ejercicios 1 del método de interpolación lineal empleando el método de interpolación de Newton.

11. INTEGRACIÓN NUMÉRICA

La integración numérica se emplea cuando las funciones a integrar se encuentran en forma tabular, gráfica o son demasiado complejas como para integrarlas analíticamente. Sin embargo, aún en situaciones donde las funciones no son muy complejas se recurre a la integración numérica, porque inclusive en esos casos es más sencillo encontrar la solución numéricamente.

Existen muchos métodos que nos permiten integrar funciones o tablas numéricamente. En este capítulo estudiaremos los métodos del *Trapecio*, *Romberg* y *Simpson*, debido a que son los de mayor aplicación práctica en el campo de la ingeniería.

11.1. Método del trapecio

La integral se define como el área bajo la curva de la función (ver Figura 10.1).

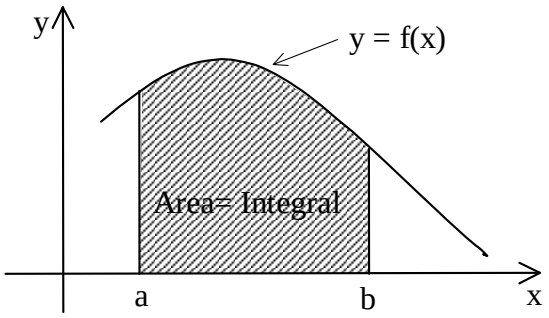


Figura 10.1. Representación gráfica de la integral

Todos los métodos numéricos tratan de calcular esta área empleando para ello diferentes aproximaciones. Si se logra calcular esta área se tiene entonces el valor de la integral:

$$\int_a^b f(x) * dx = \text{Area bajo la curva} \qquad (1)$$

Con el fin de facilitar los cálculos, el área bajo la curva puede ser dividido en *"n"* segmentos, tal como se muestra en la Figura 10.2.

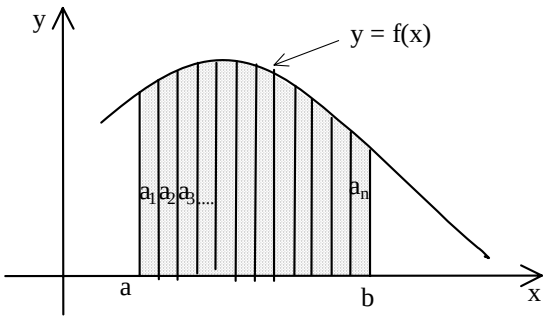


Figura 10.2. La integral (área bajo la curva) dividida en "n" segmentos

Entonces la integral (área bajo la curva) se calcula sumamos las áreas de cada uno de los segmentos:

$$\int_a^b f(x)*dx = Area\ bajo\ la\ curva = \sum_{i=1}^{n} a_i \qquad (2)$$

En el método del trapecio se asume que cada uno de estos segmentos tiene la forma de un trapecio. En otras palabras se asume que la curva entre dos puntos consecutivos es una línea recta.

Como se puede observar en los segmentos ampliados de la Figura 10.3, el asumir que los segmentos están unidos por una línea recta ocasiona un error (por exceso o por defecto) en el cálculo de las áreas de los segmentos. Este error es más grande cuanto más grande es el ancho del segmento y menor cuanto más pequeño es el ancho (dentro de los límites de precisión que nos permite la computadora o calculadora con la que estemos trabajando).

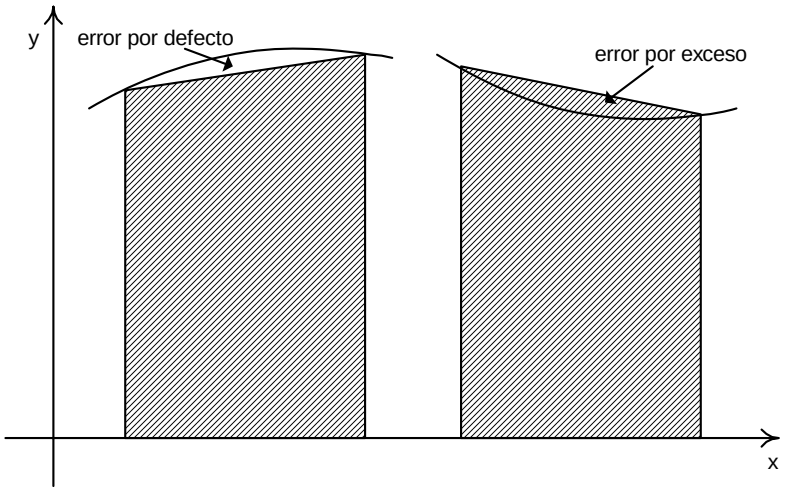


Figura 10.3. Vista amplificada de dos segmentos.

Como se ha asumido que los segmentos son trapecios, el área de cualquier segmento *i*, puede ser calculada con la siguiente ecuación:

$$a_i = \frac{(x_{i+1} - x_i)(y_{i+1} + y_i)}{2} \qquad (3)$$

Y si llamamos "h_i" al ancho del segmento:

$$h_i = x_{i+1} - x_i \qquad (4)$$

La ecuación (4) se transforma en:

$$a_i = \frac{h_i}{2}(y_i + y_{i+1}) \qquad (5)$$

Aplicando esta ecuación (o la ecuación 3) podemos calcular el área de cualquier segmento y con estas áreas y la ecuación (2) la integral (el área bajo la curva).

11.1.1. Integración de datos tabulados

Para aplicar la ecuación (3) o (4), es necesario conocer los valores de x_i y y_i de todos los segmentos y esto es algo que se conoce cuando los datos están tabulados, pues son los datos de la tabla. Por consiguiente para integrar datos tabulados sólo tenemos que aplicar la ecuación (3) a cada par de datos consecutivos (un segmento). Entonces si se tienen "*n*" datos se calculará el área de "*n-1*" segmentos.

Para comprender mejor el proceso calcularemos la integral: $\int_{2.5}^{7.2} f(x)dx = \int_{2.5}^{7.2} ydx$, con los datos de la tabla:

X	*Y*
2.0	4.800
2.5	6.625
2.7	7.509
3.0	9.00
3.3	10.689
3.8	13.944
4.2	16.944
4.5	19.425
5.1	24.981
5.7	31.329
6.0	34.800
6.4	39.736
6.9	46.401
7.2	50.664
7.5	55.125

Los límites de la integral están comprendidos entre 2.5 y 7.2, entonces tomamos todos los puntos comprendidos entres estos límites, esto es un total de 13 puntos, por lo que debemos calcular el área de 12 segmentos (aplicando la ecuación 3):

$$a_1 = \frac{(2.7-2.5)(7.509+6.625)}{2} = 1.4134$$

$$a_2 = \frac{(3.0-2.7)(9.0+7.509)}{2} = 2.47635$$

$$a_3 = \frac{(3.3-3.0)(10.689+9.0)}{2} = 2.95335$$

$$a_4 = \frac{(3.8-3.3)(13.944+10.689)}{2} = 6.15825$$

$$a_5 = \frac{(4.2-3.8)(16.944+13.944)}{2} = 6.1776$$

$$a_6 = \frac{(4.5-4.2)(19.425+16.944)}{2} = 5.45535$$

$$a_7 = \frac{(5.1-4.5)(24.981+16.944)}{2} = 13.3215$$

$$a_8 = \frac{(5.7-5.1)(31.329+24.981)}{2} = 16.893$$

$$a_9 = \frac{(6.0-5.7)(34.8+31.329)}{2} = 9.9135$$

$$a_{10} = \frac{(6.4-6.0)(39.736+34.8)}{2} = 14.9072$$

$$a_{11} = \frac{(6.9 - 6.4)(46.401 + 39.736)}{2} = 21.53425$$

$$a_{12} = \frac{(7.2 - 6.9)(50.664 + 46.401)}{2} = 14.55975$$

Ahora podemos emplear la ecuación (2) para calcular el área total (la integral):

$$\int_{2.5}^{7.2} f(x) * dx = \sum_{i=1}^{12} a_i = 1.4134 + 2.47635 + 2.95335 + 6.15825 + 6.1776 + 5.45535 + 13.3215 + 16.893 + 9.9135 +$$
$$14.9072 + 21.53425 + 14.55975 = 115.76965$$

Con lo que obtenemos el resultado buscado.

En ocasiones puede suceder que los límites de la integral no correspondan exactamente con los datos tabulados. En esos casos se calculan los puntos correspondientes a estos límites mediante una interpolación (o extrapolación) lineal.

Por ejemplo, para calcular el valor de la integral: $\int_{2.2}^{7.4} f(x)dx = \int_{2.2}^{7.4} ydx$, empleando la tabla del anterior ejemplo, vemos que los límites no corresponden exactamente a ningún punto de la tabla, por lo que es necesario realizar una interpolación lineal para calcular los puntos límite.

Para el límite inferior realizamos la siguiente interpolación:

```
[2.0 2.5] [4.8 6.625] 2.2 INLIN
```

Con lo que obtenemos el valor: 5.53, por lo tanto el primer punto es: (2.2, 5.53).

Para el límite superior realizamos la siguiente interpolación:

```
[7.2 7.5] [50.664 55.125] 7.4 INLIN
```

Con lo que obtenemos el valor: 53.638, por lo tanto el último punto es: (7.4, 53.638).

Con estos dos nuevos puntos tenemos un total de 15 puntos entre los límites inferior y superior, por lo tanto para calcular la integral debemos calcular el área de 14 segmentos. Sin embargo, los puntos intermedios son los mismos del ejemplo anterior, por lo tanto las áreas de los 12 segmentos intermedios son las mismas (sólo que ahora serán los segmentos 2 a 13 en lugar de 1 a 12).

En consecuencia nos resta calcular dos áreas más.

El área del primer segmento es:

$$a_1 = \frac{(2.5 - 2.2)(6.625 + 5.53)}{2} = 1.82325$$

Y el área del último:

$$a_{14} = \frac{(7.4 - 7.2)(53.638 + 50.664)}{2} = 10.4302$$

Entonces el área total (la integral) es:

$$\int_{2.2}^{7.4} f(x)*dx = \sum_{i=1}^{14} a_i = 1.82325 + 1.4134 + 2.47635 + 2.95335 + 6.15825 + 6.1776 + 5.45535 + 13.3215 + 16.893 +$$
$$9.9135 + 14.9072 + 21.53425 + 14.55975 + 10.4302 = 128.0231$$

Que es el valor buscado.

La forma tabular del método del trapecio puede ser empleada también para calcular el área de funciones analíticas, en cuyo caso se construye una tabla reemplazando valores de la variable independiente (comprendidos entre los límites inferior y superior) en la función a integrar. Lo más conveniente en estos casos es fijar un número de segmentos de manera que los puntos intermedios queden igualmente espaciados. El espacio que separa cada par de puntos se calcula dividiendo la diferencia de los límites superior e inferior entre el número de segmentos:

$$h = \frac{\textit{Límite superior} - \textit{Límite inferior}}{\textit{Número de segmentos}} \qquad (6)$$

Siendo *h* el espacio que separa los puntos (o ancho del segmento).

Por ejemplo para calcular el valor de la integral:

$$\int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1})dx$$

Fijamos un número de segmentos, por ejemplo 10, con lo que el espacio entre puntos (o incremento) es:

$$h = \frac{8.4 - 5.1}{10} = 0.33$$

Como hemos fijado 10 segmentos tenemos que calcular 11 puntos. El primer punto se calcula con el límite inferior *(x_1):*

$$x_1 = 5.1$$
$$y_1 = 3 - 6x_1^2 + 1.1x_1^{3.1} = 3 - 6*5.1^2 + 1.1*5.1^{3.1} = 18.675560187$$

Para el segundo punto el valor de *x (x_2)* se calcula sumando al valor anterior *(x_1)* el incremento *(h)*.

$$x_2 = x_1 + h = 5.1 + 0.33 = 5.43$$
$$y_2 = 3 - 6x_2^2 + 1.1x_2^{3.1} = 3 - 6*5.43^2 + 1.1*5.43^{3.1} = 34.670356868$$

De la misma forma calculamos los puntos restantes:

$$x_3 = x_2 + h = 5.43 + 0.33 = 5.76$$
$$y_3 = 3 - 6x_3^2 + 1.1x_3^{3.1} = 3 - 6*5.76^2 + 1.1*5.76^{3.1} = 54.373643803$$

$$x_4 = x_3 + h = 5.76 + 0.33 = 6.09$$
$$y_4 = 3 - 6x_4^2 + 1.1x_4^{3.1} = 3 - 6*6.09^2 + 1.1*6.09^{3.1} = 78.121677860$$

$$x_5 = x_4 + h = 6.09 + 0.33 = 6.42$$
$$y_5 = 3 - 6x_5^2 + 1.1x_5^{3.1} = 3 - 6*6.42^2 + 1.1*6.42^{3.1} = 106.252649908$$

$$x_6 = x_5 + h = 6.42 + 0.33 = 6.75$$
$$y_6 = 3 - 6x_6^2 + 1.1x_6^{3.1} = 3 - 6*6.75^2 + 1.1*6.75^{3.1} = 139.10658997$$

$$x_7 = x_6 + h = 6.75 + 0.33 = 7.08$$
$$y_7 = 3 - 6x_7^2 + 1.1x_7^{3.1} = 3 - 6*7.08^2 + 1.1*7.08^{3.1} = 177.025281858$$

$$x_8 = x_7 + h = 7.08 + 0.33 = 7.41$$
$$y_8 = 3 - 6x_8^{\,2} + 1.1x_8^{\,3.1} = 3 - 6*7.41^2 + 1.1*7.41^{3.1} = 220.352185648$$

$$x_9 = x_8 + h = 7.41 + 0.33 = 7.74$$
$$y_9 = 3 - 6x_9^{\,2} + 1.1x_9^{\,3.1} = 3 - 6*7.74^2 + 1.1*7.74^{3.1} = 269.432367184$$

$$x_{10} = x_9 + h = 7.74 + 0.33 = 8.07$$
$$y_{10} = 3 - 6x_{10}^{\,2} + 1.1x_{10}^{\,3.1} = 3 - 6*8.07^2 + 1.1*8.07^{3.1} = 324.612433097$$

$$x_{11} = x_{10} + h = 8.07 + 0.33 = 8.4$$
$$y_{11} = 3 - 6x_{10}^{\,2} + 1.1x_{10}^{\,3.1} = 3 - 6*8.07^2 + 1.1*8.07^{3.1} = 386.240472262$$

De esta manera tenemos los 11 puntos que se requieren para calcular las 10 áreas. Los cálculos de estas áreas (con la ecuación 3) se simplifican un poco porque al estar los segmentos igualmente espaciados, la diferencia $x_{i+1}-x_i$ es constante e igual a 0.33:

$$a_1 = \frac{(0.33)(34.670356868 + 18.675560187)}{2} = 8.80207631408$$

$$a_2 = \frac{(0.33)(54.373643803 + 34.670356868)}{2} = 14.6922601107$$

$$a_3 = \frac{(0.33)(78.121677860 + 54.373643803)}{2} = 21.8617280759$$

$$a_4 = \frac{(0.33)(106.252649908 + 78.121677860)}{2} = 30.42127640832$$

$$a_5 = \frac{(0.33)(139.10658997 + 106.252649908)}{2} = 40.4842745794$$

$$a_6 = \frac{(0.33)(177.025281858 + 139.10658997)}{2} = 52.1617588516$$

$$a_7 = \frac{(0.33)(220.352185648 + 177.025281858)}{2} = 65.5672821385$$

$$a_8 = \frac{(0.33)(269.432367184 + 220.352185648)}{2} = 80.1844512041$$

$$a_9 = \frac{(0.33)(324.612433097 + 269.432367184)}{2} = 98.017392033$$

$$a_{10} = \frac{(0.33)(386.240472262 + 324.612433097)}{2} = 117.290729384$$

Finalmente calculamos el área bajo la curva con la ecuación (2):

$$\int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1})dx = \sum_{i=1}^{10} a_i = 8.80207631408 + 14.6922601107 + 21.8617280759 + 30.42127640832 +$$
$$40.4842745794 + 52.1617588516 + 65.5672821385 + 80.1844512041 + 98.017392033 + 117.290729384$$
$$= 530.113716775$$

Que es el resultado buscado. Este resultado tiene un error relativo igual a 0.26%, error que es aceptable en la mayoría de los cálculos ingenieriles (el resultado exacto es 528.719026846).

11.1.2. Algoritmo para la forma tabular

El algoritmo que automatiza el cálculo de las integrales a partir de datos tabulados, se presenta en la Figura 10.4.

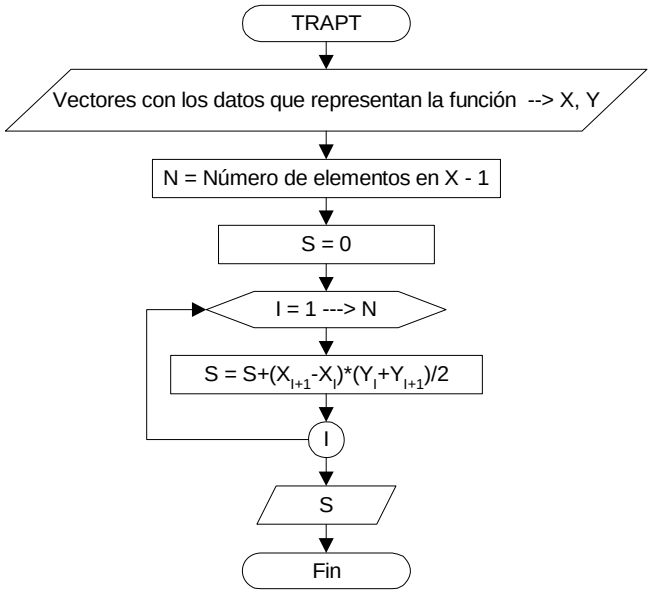


Figura 10.4. Algoritmo para integrar datos tabulados con el método del trapecio.

Como se puede observar básicamente se trata de un ciclo *FOR* donde en cada repetición se calcula el área de un segmento y se suma dicha área en una variable.

11.1.3. Programa para la forma tabular

Siguiendo el algoritmo de la Figura 10.4, se ha elaborado el siguiente programa:

```
« 0 0 □ X Y N S
 « X SIZE 1 GET 1 - 'N' STO
   1 N FOR I
     X I 1 + GET
     X I GET -
     Y I 1 + GET
     Y I GET +
     2 / * 'S' STO+
   NEXT
   S
 »
»
```

En lo sucesivo se asumirá que este programa ha sido guardado en la variable *"TRAPT"*.

11.1.4. Ejemplos para la forma tabular

Ejemplo 1

Como primer ejemplo calcularemos la integral resuelta en el primer ejemplo del acápite 11.1.1:

$$\int_{2.5}^{7.2} f(x)dx = \int_{2.5}^{7.2} ydx$$

Esta integral se resuelve con el siguiente programa:

```
« [ 2.5 2.7 3.0 3.3 3.8 4.2 4.5 5.1 5.7 6.0 6.4 6.9 7.2 ]
  [ 6.625 7.509 9 10.689 13.944 16.944 19.425 24.981 31.329 34.8
    39.736 46.401 50.664]
  TRAPT "Integral" →TAG
»
```

Que nos devuelve el resultado:

Integral: 115.76965

Que como era de esperar es el mismo resultado calculado manualmente.

Ejemplo 2

Como segundo ejemplo calcularemos la segunda integral resuelta en el segundo ejemplo del acápite 11.1.1:

$$\int_{2.2}^{7.4} f(x)dx = \int_{2.2}^{7.4} ydx$$

Esta integral se calcula con el siguiente programa:

```
« [ 2.2 2.5 2.7 3.0 3.3 3.8 4.2 4.5 5.1 5.7 6.0 6.4 6.9 7.2 7.4 ]
  [ 2.0 2.5 ] [ 4.8 6.625 ] 2.2 INLIN
  6.625 7.509 9 10.689 13.944 16.944 19.425 24.981 31.329 34.8
  39.736 46.401 50.664
  [ 7.2 7.5 ] [ 50.664 55.125 ] 7.4 INLIN 15 →ARRY
  TRAPT "Integral" →TAG
»
```

Donde como se puede observar, el primer y último elemento del vector *"y"*, se calculan por interpolación lineal. El vector *"y"* se arma a partir de sus elementos con 15 →ARRY. Haciendo correr el programa se obtiene el resultado:

Integral: 128.0231

Que es el mismo valor calculado manualmente.

Ejemplo 3

Como tercer ejemplo resolveremos el tercer ejemplo resuelto en el acápite 11.1.1:

$$\int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1})dx$$

Para calcular esta integral elaboramos el siguiente programa:

```
« 5.1 8.4 10 0 « → x « 3 6 x SQ * - 1.1 x 3.1 ^ * + » »
  → a b n h fx
  « b a - n / 'h' STO
    a b FOR x x h STEP n 1 + →ARRY
    a b FOR x x fx EVAL h STEP n 1 + →ARRY
    TRAPT "Integral" →TAG
  »
»
```

Donde el vector *"x"* se calcula en el primer ciclo *FOR* el cual va desde el límite inferior *(a)* hasta el límite superior *(b)*, incrementando en *h* en cada

repetición. El vector *"y"* se calcula en el segundo ciclo *FOR* que tiene los mismos límites e incremento que el primero, pero donde en cada repetición se calcula un elemento de *y* evaluando la función *fx*. En ambos casos una vez calculados los elementos se arma el vector con *n 1 + →ARRY*. Haciendo correr el programa se obtiene el resultado:

```
Integral: 530.113716775
```

Que es el mismo valor calculado manualmente. Con el programa sin embargo es muy fácil mejorar la precisión del resultado incrementando el número de segmentos. Así si cambiamos de 10 a 100 el número de segmentos obtenemos el resultado:

```
Integral: 528.73297382
```

Cuyo error relativo es sólo 0.0026%.

11.1.5. Integración de funciones analíticas

Como se demostró anteriormente, el método del trapecio en su forma tabular puede ser empleado también para resolver funciones analíticas. Sin embargo, puesto que en estos casos el espaciamiento entre puntos *(h)* es uniforme, puede ser factorizado dando lugar a una forma más compacta para el cálculo del área:

$$\int_a^b f(x)*dx = \sum_{i=1}^{n} a_i = \frac{h}{2}(y_1+y_2)+\frac{h}{2}(y_2+y_3)+\frac{h}{2}(y_3+y_4)+\ldots+\frac{h}{2}(y_{n-1}+y_n)+\frac{h}{2}(y_n+y_{n+1})$$

$$\int_a^b f(x)*dx = \frac{h}{2}(y_1+y_2+y_2+y_3+y_3+y_4+\ldots+y_{n-1}+y_n+y_n+y_{n+1})$$

$$\int_a^b f(x)*dx = \frac{h}{2}(y_1+2y_2+2y_3+2y_4+\ldots+2y_{n-1}+2y_n+y_{n+1})$$

$$\int_a^b f(x)*dx = \frac{h}{2}\left(y_1+2\sum_{i=2}^{n} y_i + y_{n+1}\right) \qquad (7)$$

Esta ecuación es conocida como la *regla del trapecio* y es la forma que se emplea para integrar funciones analíticas.

11.1.6. Algoritmo para la forma analítica

Al resolver la ecuación (7) los valores de y_i no se calculan y almacenan en un vector, como se hace cuando se resuelven funciones analíticas con la forma tabular, sino que son calculados y sumados directamente en un ciclo *FOR* donde la variable independiente *(x)* va desde el límite inferior más *h*, hasta el límite inferior menos *h*. El primer (y_1) y último punto (y_{n+1}) se calculan luego empleando los límites inferior *(a)* y superior *(b)* respectivamente.

El algoritmo elaborado para esta forma se presenta en la Figura 10.5.

11.1.7. Programa para la forma analítica

Siguiendo el algoritmo de la Figura 10.5, se ha elaborado el siguiente programa:

```
« 0 0 □ F A B N H S
 « B A - N / 'H' STO
   A H + B H - FOR X
     X F EVAL 'S' STO+
```

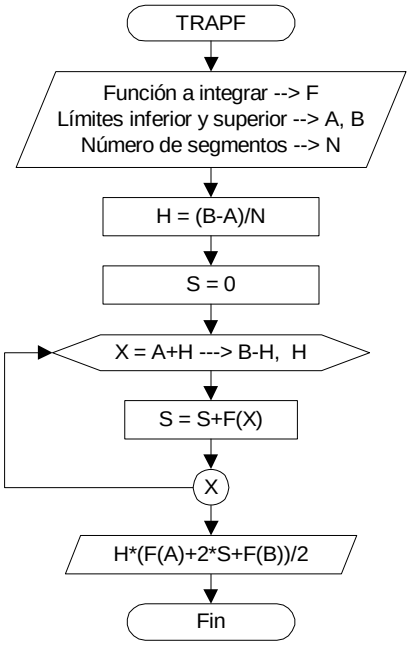


Figura 10.5. Algoritmo para integrar funciones analíticas con el método del trapecio

```
  H STEP
   A F EVAL 2 S * + B F EVAL + H * 2 /
 »
»
```

En lo sucesivo se asumirá que este programa ha sido guardado en la variable *"TRAPF"*.

11.1.8. Ejemplos para la forma analítica

Ejemplo 1

Como primer ejemplo resolveremos la integral de la función analítica calculada con la forma tabular:

$$\int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1})dx$$

El programa elaborado para calcular esta integral (con 100 segmentos) es:

```
« « → x « 3 6 x SQ * - 1.1 x 3.1 ^ * + » » 5.1 8.4 100
  TRAPF
  "Integral" →TAG
»
```

Haciendo correr el programa se obtiene el resultado:

Integral: 528.73297382

Que es la solución buscada y que como era de esperar, concuerda con el resultado calculado con la forma tabular.

Ejemplo 2

Como segundo ejemplo, para demostrar que la forma analítica nos permite integrar también datos tabulados, resolveremos la integral resuelta en el primer ejemplo de la forma tabular:

$$\int_{2.5}^{7.2} f(x)dx = \int_{2.5}^{7.2} ydx$$

En este caso los datos tabulados se introducen en una función que recibe el valor de la variable independiente y emplea un método de interpolación para devolver el correspondiente valor de *"y"*. El programa elaborado para resolver esta integral, donde la función emplea el método de interpolación lineal y se consideran 20 segmentos es el siguiente:

```
« « [ 2.5 2.7 3.0 3.3 3.8 4.2 4.5 5.1 5.7 6.0 6.4 6.9 7.2 ]
    [ 6.625 7.509 9 10.689 13.944 16.944 19.425 24.981 31.329 34.8
    39.736 46.401 50.664] ROT INLIN
  » 2.5 7.2 20
  TRAPF "Integral" →TAG
»
```

Donde el comando *ROT*, toma la variable independiente (que debe encontrarse en el tercer nivel de la pila) y lo coloca en el primer nivel, pues ese es el orden en que se deben mandar los datos al método de interpolación *(INLIN)*. Haciendo correr el programa se obtiene:

```
Integral: 115.81900975
```

Resultado que concuerda bastante bien con el calculado con la forma tabular.

Por supuesto es posible emplear otro método de interpolación, por ejemplo si en lugar del método de interpolación lineal empleamos el método de *Newton* el programa cambia a la siguiente forma:

```
« [ 2.5 2.7 3.0 3.3 3.8 4.2 4.5 5.1 5.7 6.0 6.4 6.9 7.2 ]
  [ 6.625 7.509 9 10.689 13.944 16.944 19.425 24.981 31.329 34.8
    39.736 46.401 50.664] 0
  → vx vy vc
  « vx vy CONEW 'vc' STO
    « vx vc ROT INNEW » 2.5 7.2 20
    TRAPF "Integral" →TAG
  »
»
```

Haciendo correr el programa obtenemos la solución:

```
Integral: 115.642518875
```

Que igualmente concuerda bastante bien con el resultado obtenido con la forma tabular.

11.1.9. Preguntas y Ejercicios

1. ¿Cuál es la definición de una integral?
2. En el método del trapecio ¿Qué forma se asume para cada uno de los segmentos en los que se divide el área bajo la curva?
3. En el método del trapecio ¿Qué tipo línea une los puntos de cada segmento?
4. En el método del trapecio, para calcular la integral con mayor precisión ¿es conveniente que los segmentos sean grandes o pequeños?

5. Si se toman los datos de la tabla del ejemplo 1 del método del trapecio en su forma tabular ¿De cuántos segmentos deberemos calcular el área para resolver la integral: $\int_{3.0}^{6.4} f(x)dx = \int_{3.0}^{6.4} ydx$?

6. Si se toman los datos de la tabla del ejemplo 1 del método del trapecio en su forma tabular ¿De cuantos segmentos deberemos calcular el área para resolver la integral: $\int_{1.8}^{8.3} f(x)dx = \int_{1.8}^{8.3} ydx$?

7. Para resolver la integral de la anterior pregunta, ¿Qué operación adicional deberemos efectuar?

8. ¿Qué se debe hacer para calcular una función analítica empleando la forma tabular del método del trapecio?

9. ¿Por qué se simplifican los cálculos cuando se integran funciones analíticas con el método del trapecio?

10. En la forma analítica del método del trapecio ¿los valores de y_i deben ser calculados y almacenados en un vector?

11. Cuando se integra funciones analíticas ¿Cómo se calcula el espacio (o incremento) entre los puntos de cada segmento?

12. ¿Es posible integrar datos tabulados empleando el método del trapecio en su forma analítica?

13. Si en el método del trapecio se quiere calcular una integral con mayor precisión ¿Se deberá incrementar o disminuir el número de segmentos?

14. Con los datos de la siguiente tabla elabore un programa que calcule la integral: $\int_{3.6}^{6.9} f(x)dx$, empleando el método de trapecio en su forma tabular.

x	3.5	4	4.5	5	5.5	6	6.2	6.5	6.8
y	0.002	0.030	0.155	0.396	0.658	0.903	0.933	0.975	0.993

15. Repita el ejercicio 14 empleando el método del trapecio en su forma analítica y el método de interpolación lineal.

16. Repita el ejercicio 14 empleando el método del trapecio en su forma analítica y el método de interpolación de Lagrange.

17. Repita el ejercicio 14 empleando el método del trapecio en su forma analítica y el método de interpolación de Newton.

18. Empleando los datos del ejercicio 14 calcule la integral: $\int_{4}^{6.2} f(x)dx$, empleando el método del trapecio en su forma tabular.

19. Elabore un programa que resuelva la siguiente integral (con 100 segmentos): $\int_{0}^{12.4} \frac{xdx}{\sqrt{x^2+2x+5}}$, empleando el método del trapecio en su forma analítica.

20. Repita el ejercicio 19 empleando el método del trapecio en su forma tabular.

21. Repita el ejercicio 19 empleando el método del trapecio en su forma analítica pero tomando 200 segmentos en lugar de 100. Compare los resultados obtenidos y determine si son suficientes 100, 200 o se requiere un mayor número de segmentos.

11.2. Método de Simpson

En el método del trapecio, se asume que los puntos están unidos por líneas rectas, en el método de Simpson se mejora esta aproximación pues se asume que cada tres puntos (2 segmentos) están unidos por una ecuación cuadrática (ver Figura 10.6).

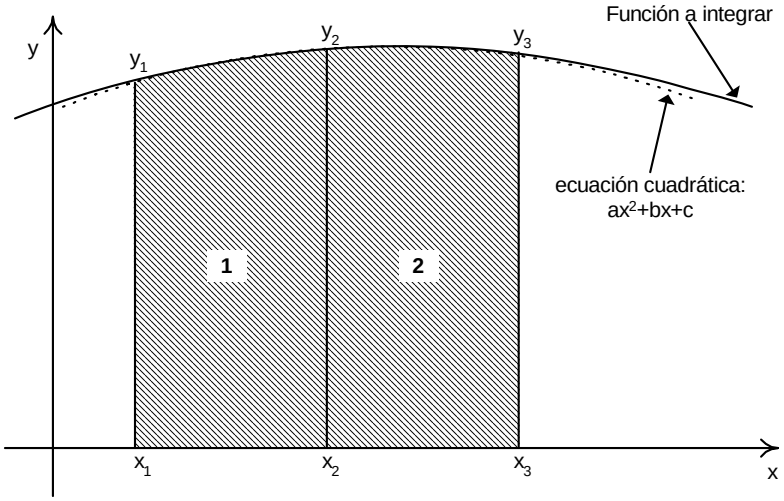


Figura 10.6. Área de dos segmentos en el método de Simpson

Entonces el área de 2 segmentos puede ser calculada integrando la ecuación cuadrática ax^2+bx+c:

$$\text{Área de dos segmentos} = \int_{x_1}^{x_3} (ax^2+bx+c)dx = \frac{a}{3}\left(x_3^3 - x_1^3\right) + \frac{b}{2}\left(x_3^2 - x_1^2\right) + c\left(x_3 - x_1\right) \qquad (8)$$

Si reemplazamos los tres puntos del segmento (x_1,y_1), (x_2,y_2) y (x_3,y_3) en esta ecuación, se forma un sistema de 3 ecuaciones con 3 incógnitas:

$$\begin{aligned} y_1 &= ax_1^2 + bx_1 + c \\ y_2 &= ax_2^2 + bx_2 + c \\ y_3 &= ax_3^2 + bx_3 + c \end{aligned} \qquad (9)$$

De este sistema podemos despejar los coeficientes a, b y c de la ecuación cuadrática:

$$\begin{aligned} a &= \frac{(y_1 - y_3)(x_1 - x_2) - (y_1 - y_2)(x_1 - x_3)}{(x_1^2 - x_3^2)(x_1 - x_2) - (x_1^2 - x_2^2)(x_1 - x_3)} \\ b &= \frac{(y_1 - y_2) - a(x_1^2 - x_2^2)}{(x_1 - x_2)} \\ c &= y_1 - ax_1^2 - bx_1 \end{aligned} \qquad (10)$$

Si en lugar de los puntos *1*, *2* y *3* tomamos en cuenta tres puntos consecutivos cualesquiera: *i*, *j* y *k*, la ecuación (8) toma la forma general:

$$\text{Area de los segmentos } i, j = \int_{x_i}^{x_k} (ax^2+bx+c)dx = \frac{a}{3}\left(x_k^3 - x_i^3\right) + \frac{b}{2}\left(x_k^2 - x_i^2\right) + c\left(x_k - x_i\right) \qquad (11)$$

Igualmente las ecuaciones (10) toman la forma general:

$$a = \frac{(y_i - y_k)(x_i - x_j) - (y_i - y_j)(x_i - x_k)}{(x_i^2 - x_k^2)(x_i - x_j) - (x_i^2 - x_j^2)(x_i - x_k)}$$

$$b = \frac{(y_i - y_j) - a(x_i^2 - x_j^2)}{(x_i - x_j)} \qquad (12)$$

$$c = y_i - ax_i^2 - bx_i$$

Entonces el área total bajo la curva (la integral) puede ser calculada sumando las áreas de los *"n"* segmentos existentes entre el límite inferior *(a)* y superior *(b)*:

$$Area\ bajo\ la\ curva = \int_a^b f(x)dx = \sum_{i=1,3,5}^{n-1} a_{ij} \qquad (13)$$

Para aplicar la ecuación (13) el número de segmentos *"n"* debe ser par, o lo que es lo mismo, el número de puntos debe ser impar.

Las ecuaciones (11) a (13) son las ecuaciones generales del método de Simpson. Estas ecuaciones permiten obtener resultados más exactos que el método del trapecio, porque, tal como se pude ver en la Figura 10.6, la ecuación cuadrática tiene una forma mucho más cercana a la forma real que la ecuación lineal.

11.2.1. Integración de datos tabulados

Cuando se integran datos tabulados, estos datos son los puntos *(x,y)* necesarios para resolver las ecuaciones (11) a (13). No obstante puede suceder que el número de puntos comprendidos entre los límites de integración sea par, o lo que es lo mismo el número de segmentos sea impar. Entonces no se pueden aplicar directamente las ecuaciones (11) y (13) pues las mismas han sido deducidas para un número par de segmentos.

En estos casos lo que se hace es calcular los coeficientes de la ecuación cuadrática con los tres primeros puntos, aplicando las ecuaciones (10) o lo que es lo mismo las ecuaciones (12). Los coeficientes calculados se emplean entonces para calcular el área del primer segmento con la ecuación (8) (o la ecuación 11), pero modificando los límites de manera que involucren sólo al primer segmento:

$$a_1 = \int_{x_1}^{x_2} (ax^2 + bx + c)dx = \frac{a}{3}\left(x_2^3 - x_1^3\right) + \frac{b}{2}\left(x_2^2 - x_1^2\right) + c\left(x_2 - x_1\right) \qquad (14)$$

Una vez calculada el área del primer segmento queda un número par de segmentos y como ya es par se pueden emplear las ecuaciones (11) a (13) para calcular el área de los segmentos restantes (comenzando con el segundo punto). Por lo tanto el área total bajo la curva (integral) es:

$$Area\ bajo\ la\ curva\ (n\ impar) = \int_a^b f(x)dx = a_1 + \sum_{i=2,4,6...}^{n-1} a_{ij} \qquad (15)$$

Para comprender mejor el procedimiento de cálculo del método Simpson, resolveremos la primera integral resuelta con el método del Trapecio:

$$\int_{2.5}^{7.2} f(x)dx$$

Siendo los datos tabulados los mismos del mencionado ejemplo.

Entre los límites (2.5 y 7.2) existe un total de 13 puntos en la tabla, por lo tanto se tiene un número par de segmentos, en consecuencia podemos aplicar directamente las ecuaciones (11) a (13).

Aplicando las ecuaciones (12) calculamos los coeficientes de la ecuación cuadrática que une los tres primeros puntos (los dos primeros segmentos) (*i=1, j=2* y *k=3)*:

$$a=\frac{(y_1-y_3)(x_1-x_2)-(y_1-y_2)(x_1-x_3)}{(x_1^2-x_3^2)(x_1-x_2)-(x_1^2-x_2^2)(x_1-x_3)}=\frac{(6.625-9)(2.5-2.7)-(6.625-7.509)(2.5-3.0)}{(2.5^2-3.0^2)(2.5-2.7)-(2.5^2-2.7^2)(2.5-3.0)}=1.1$$

$$b=\frac{(y_1-y_2)-a(x_1^2-x_2^2)}{(x_1-x_2)}=\frac{(6.625-7.509)-1.1(2.5^2-2.7^2)}{(2.5-2.7)}=-1.3$$

$$c=y_1-ax_1^2-bx_1=6.625-(1.1)2.5^2-(-1.3)2.5=3$$

Con estos coeficientes calculamos el área de los dos primeros segmentos aplicando la ecuación (12):

$$a_{1,2}=\frac{a}{3}\left(x_3^3-x_1^3\right)+\frac{b}{2}\left(x_3^2-x_1^2\right)+c\left(x_3-x_1\right)=\frac{1.1}{3}\left(3^3-2.5^3\right)+\frac{-1.3}{2}\left(3^2-2.5^2\right)+3\left(3-2.5\right)=3.883333333334$$

Ahora calculamos el área del tercer y cuarto segmento empleando los puntos 3 a 5:

$$a=\frac{(y_3-y_5)(x_3-x_4)-(y_3-y_4)(x_3-x_5)}{(x_3^2-x_5^2)(x_3-x_4)-(x_3^2-x_4^2)(x_3-x_5)}=\frac{(9-13.944)(3-3.8)-(9-10.689)(3-3.8)}{(3^2-3.8^2)(3-3.3)-(3^2-3.3^2)(3-3.8)}=1.1$$

$$b=\frac{(y_3-y_4)-a(x_3^2-x_4^2)}{(x_3-x_4)}=\frac{(9-10.689)-1.1(3^2-3.3^2)}{(3-3.3)}=-1.3$$

$$c=y_3-ax_3^2-bx_3=9-(1.1)3^2-(-1.3)3=3$$

Que como vemos devuelve los mismos coeficientes que en los dos primeros segmentos. Entonces el área de estos dos segmentos es:

$$a_{3,4}=\frac{a}{3}\left(x_5^3-x_3^3\right)+\frac{b}{2}\left(x_5^2-x_3^2\right)+c\left(x_5-x_3\right)=\frac{1.1}{3}\left(3.8^3-3^3\right)+\frac{-1.3}{2}\left(3.8^2-3^2\right)+3\left(3.8-3\right)=9.0837333333$$

De la misma manera calculamos las áreas de los pares de segmentos restantes (5-6,7-8,9-10,11-12), obteniéndose los siguientes valores:

$$a_{5,6}=11.6162666667$$

$$a_{7,8}=30.1356$$

$$a_{9,10}=24.809866667$$

$$a_{11,12}=36.066133334$$

Con estos valores y la ecuación (13) calculamos el área total bajo la curva:

$$\int_{2.5}^{7.2} f(x)dx=\sum_{i=1,3,5,\ldots}^{12-1} a_{ij}=a_{1,2}+a_{3,4}+a_{5,6}+a_{7,8}+a_{9,10}+a_{11,12}$$

$$=3.883333333334+9.0837333333+11.6162666667+30.1356+24.809866667+36.066133334$$

$$=115.5949333333$$

Que es el valor de la integral. Este resultado es el valor exacto de la integral. En esta ocasión el método de Simpson calcula el resultado exacto porque los datos tabulados se ajustan perfectamente a una ecuación cuadráti-

ca, como lo revelan los coeficientes de la ecuación cuadrática que para todos los pares de segmentos son exactamente iguales.

Como se ha podido apreciar en este ejemplo, el método de Simpson requiere más cálculos que el método del trapecio, razón por la cual se justifica aún más su automatización.

Para ejemplificar el procedimiento que se sigue cuando el número de segmentos es impar, calcularemos la integral:

$$\int_{2.0}^{7.2} f(x)dx$$

En este caso el número de puntos comprendidos entre el límite inferior y superior es 14, por lo tanto se tiene un número impar de segmentos (13). Siguiendo el procedimiento antes explicado calculamos primero los coeficientes de la ecuación cuadrática empleando los tres primeros puntos:

$$a = \frac{(y_1 - y_3)(x_1 - x_2) - (y_1 - y_2)(x_1 - x_3)}{(x_1^2 - x_3^2)(x_1 - x_2) - (x_1^2 - x_2^2)(x_1 - x_3)} = \frac{(4.8 - 7.509)(2 - 2.5) - (4.8 - 6.625)(2 - 2.7)}{(2^2 - 2.7^2)(2 - 2.5) - (2^2 - 2.5^2)(2 - 2.7)} = 1.1$$

$$b = \frac{(y_1 - y_2) - a(x_1^2 - x_2^2)}{(x_1 - x_2)} = \frac{(4.8 - 6.625) - 1.1(2^2 - 2.5^2)}{(2 - 2.5)} = -1.3$$

$$c = y_1 - ax_1^2 - bx_1 = 4.8 - (1.1)2^2 - (-1.3)2 = 3$$

Y con estos coeficientes calculamos el área del primer segmento (ecuación 14):

$$a_1 = \frac{1.1}{3}\left(2.5^3 - 2^3\right) + \frac{-1.3}{2}\left(2.5^2 - 2^2\right) + 3\left(2.5 - 2\right) = 2.83333333334$$

Los doce segmentos restantes se calculan con las ecuaciones (11) y (12) y en este ejemplo son los doce segmentos calculados en el ejemplo anterior, por consiguiente el área total es:

$$\int_{2}^{7.2} f(x)dx = a_1 + \sum_{i=2,4,6,\ldots}^{13-1} a_{i,j} = a_1 + a_{2,3} + a_{4,5} + a_{6,7} + a_{8,9} + a_{10,11} + a_{12,13}$$

$$= 2.83333333334 + 3.883333333334 + 9.0837333333 + 11.6162666667 + 30.1356 + 24.809866667 + 36.066133334$$

$$= 118.428266667$$

Que es la integral buscada.

Al igual que con el método del trapecio, el método de Simpson en su forma tabular puede ser empleado para integrar funciones analíticas. El procedimiento es mismo que se siguió con el método del trapecio: empleando la ecuación analítica se calculan puntos para valores de la variable independiente igualmente espaciados, luego se emplean esos puntos para calcular el área. En este caso por supuesto se establece un número par de segmentos, de manera que sea posible aplicar directamente las ecuaciones (11) a (13).

11.2.2. Algoritmo para la forma tabular

El algoritmo para integrar datos tabulados con el método de Simpson se presenta en la Figura 10.7. Este algoritmo toma en cuenta los casos donde el número de segmentos es impar, calculando de ser así el área del primer segmento con los tres primeros puntos, incrementando los contadores y disminuyendo el número de segmentos en 1 (para que quede un número par de segmentos).

El cálculo de los coeficientes se lleva a cabo en un submódulo porque dichos coeficientes deben ser calculados en dos partes: una cuando el número de segmentos es impar y otra en el ciclo donde se calculan las áreas de cada par de segmentos. Como se puede observar en el submódulo se calculan por separado varias diferencias (DXIJ, DXIJ2, etc), se procedido de esta manera principalmente para evitar la repetición de código.

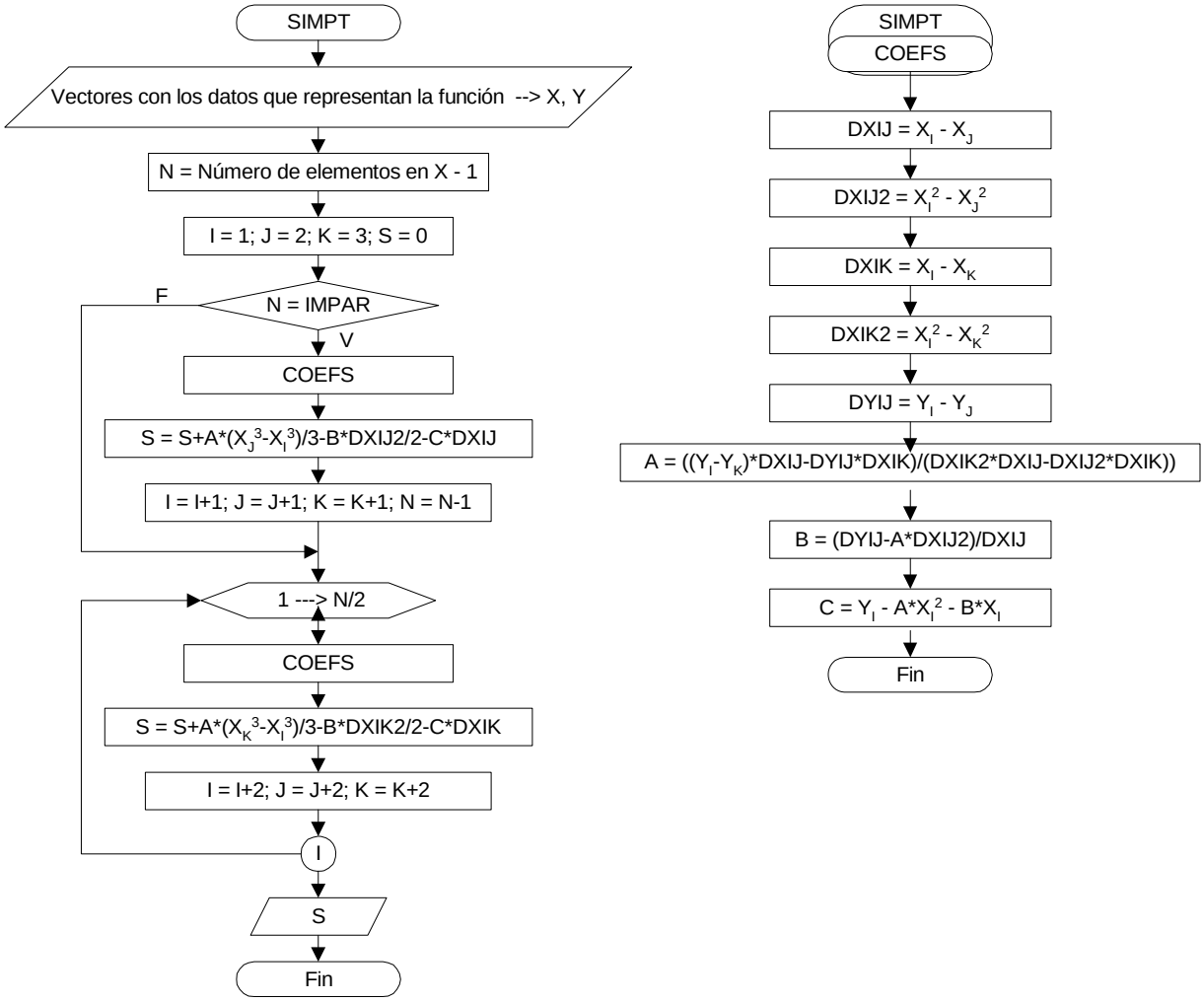


Figura 10.7. Algoritmo para integrar datos tabulados con el método de Simpson

11.2.3. Programa para la forma tabular

Siguiendo los diagramas de flujo presentados en la Figura 10.7, se ha elaborado el siguiente programa, que automatiza el proceso de cálculo:

```
« 1 2 3 0 11 NDUPN DROP □ X Y I J
  K N A B C S DXIJ DXIJ2 DXIK
  DXIK2 DYIJ COEFS
 «
   «
     X I GET X J GET - 'DXIJ' STO
     X I GET SQ X J GET SQ -
      'DXIJ2' STO
     X I GET X K GET - 'DXIK' STO
     X I GET SQ X K GET SQ -
```

```
      'DXIK2' STO
     Y I GET Y J GET - 'DYIJ' STO
     Y I GET Y K GET - DXIJ *
      DYIJ DXIK * - DXIK2 DXIJ *
      DXIJ2 DXIK * - / 'A' STO
     DYIJ A DXIJ2 * - DXIJ / 'B'
      STO
     Y I GET A X I GET SQ * - B
      X I GET * - 'C' STO
  » 'COEFS' STO
  X SIZE LIST→ - 'N' STO
  IF N 2 MOD 0 ≠ THEN
    COEFS EVAL
    A 3 / X J GET 3 ^ X I GET
     3 ^ - * B 2 / DXIJ2 * -
     C DXIJ * - 'S' STO
   1 'I' STO+ 1 'J' STO+ 1 'K'
    STO+ 'N' 1 STO-
  END
  1 N 2 / START
    COEFS EVAL
    A 3 / X K GET 3 ^ X I GET 3
     ^ - * B 2 / DXIK2 * - C
     DXIK * - 'S' STO+
    2 'I' STO+ 2 'J' STO+ 2 'K'
     STO+
  NEXT
  S
»
»
```

En lo sucesivo se asumirá que este programa ha sido guardado en la variable *SIMPT.*

11.2.4. Ejemplos para la forma tabular

Ejemplo 1

Como primer ejemplo emplearemos el programa *SIMPT* para calcular la integral resuelta manualmente al ejemplificar el método de Simpson:

$$\int_{2.5}^{7.2} f(x)dx$$

Esta integral fue resuelta también con el programa *TRAPT* (del método del trapecio) y el programa para resolverla con *SIMPT* es esencialmente el mismo, con la única diferencia de que ahora se llama al programa *SIMPT* en lugar de *TRAPT*. Esta es una de las ventajas de automatizar las soluciones un problema puede ser resuelto con varios métodos empleando prácticamente el mismo programa.

```
« [ 2.5 2.7 3.0 3.3 3.8 4.2 4.5 5.1 5.7 6.0 6.4 6.9 7.2 ]
  [ 6.625 7.509 9 10.689 13.944 16.944 19.425 24.981 31.329 34.8
    39.736 46.401 50.664]
  SIMPT "Integral" →TAG
»
```

Que nos devuelve:

```
Integral: 115.594933333
```

Y como era de esperara es el mismo resultado calculado manualmente.

Ejemplo 2

Como segundo ejemplo resolveremos la segunda integral calculada manualmente al ejemplificar el método de Simpson:

$$\int_{2}^{7.2} f(x)dx$$

El programa para resolver este ejemplo es básicamente el mismo que el del ejemplo anterior, pues sólo se añade un punto a los datos, lo que da lugar a un número impar de segmentos, pero como ese caso está considerado en el programa *SIMPT*, lo único que se debe hacer es añadir el punto adicional al principio de los vectores:

```
« [ 2.0 2.5 2.7 3.0 3.3 3.8 4.2 4.5 5.1 5.7 6.0 6.4 6.9 7.2 ]
  [ 4.8 6.625 7.509 9 10.689 13.944 16.944 19.425 24.981 31.329 34.8
    39.736 46.401 50.664]
  SIMPT "Integral" →TAG
»
```

Programa que devuelve el resultado:

Integral: 118.428266667

Que es el mismo resultado calculado manualmente.

Ejemplo 3

Como tercer ejemplo resolveremos la siguiente integral, de una ecuación analítica, empleando el método de Simpson en su forma tabular *(SIMPT)*:

$$\int_{5.1}^{8.4} (3-6x^2+1.1x^{3.1})dx$$

Esta integral también fue resuelta con el método del trapecio *(TRAPT)* y una vez más (con excepción del método empleado), el programa es esencialmente el mismo: Se fija un número de segmentos (en este ejemplo 10), se calcula el espaciamiento o incremento (h), con ese incremento se calculan los valores del vector *"x"*, para cada valor de *"x"* se evalúa la función analítica calculándose así los elementos del vector *"y"*, finalmente se mandan estos dos vectores al método de Simpson en su forma tabular *(SIMPT)*. En el método de Simpson es conveniente que el número de segmentos fijado sea un número par:

```
« 5.1 8.4 10 0 « → x « 3 6 x SQ * - 1.1 x 3.1 ^ * + » »
  → a b n h fx
  « b a - n / 'h' STO
    a b FOR x x h STEP n 1 + →ARRY
    a b FOR x x fx EVAL h STEP n 1 + →ARRY
    SIMPT "Integral" →TAG
  »
»
```

Ejecutando el programa se obtiene el resultado:

Integral: 528.719058066

Que tiene un error relativo igual a 0.000006% con relación al resultado exacto (528.719026846), mientras que el resultado obtenido con el método del trapecio para igual número de segmentos tenía un error relativo igual a 0.26%, lo que demuestra la mayor exactitud del método de Simpson con relación al método del trapecio.

11.2.5. Integración de funciones analíticas

Como ya se demostró en el tercer ejemplo del anterior acápite, el método de Simpson en su forma tabular puede ser empleado también para resolver funciones analíticas. Sin embargo, cuando se tiene una función analítica se puede establecer un espaciamiento uniforme entre segmentos, lo que permite simplificar las ecuaciones. Así el cálculo de los coeficientes (para tres puntos consecutivos *i, j, k*) se simplifica a:

$$a = \frac{y_i - 2y_j + y_k - y_j}{2h^2}$$
$$b = \frac{y_k - y_i}{2h} \qquad (16)$$
$$c = y_j$$

Donde *h* es el espaciamiento (o incremento de la variable independiente), tal como fue definida en la ecuación (6). Entonces, el cálculo del área de dos segmentos consecutivos se reduce a:

$$a_{i,j} = \frac{2a}{3}h^3 + 2ch = \frac{h}{3}(y_i + 4y_j + y_k) \qquad (17)$$

Y reemplazando la ecuación (17) en la ecuación (13):

$$\int_a^b f(x)*dx = \sum_{i=1,3,5,...}^{n-1} a_{ij} = \frac{h}{3}(y_1 + 4y_2 + y_3 + y_3 + 4y_4 + y_5 + y_5 + 4y_6 + y_7 + ... + y_{n-1} + 4y_n + y_{n+1})$$

$$\int_a^b f(x)*dx = \frac{h}{3}(y_1 + +4(y_2 + y_4 + y_6 + ... + y_n) + 2(y_3 + y_5 + y_7 + ... + y_{n-1}) + y_{n+1})$$

La ecuación (13) se reduce a

$$\int_a^b f(x)*dx = \frac{h}{3}(y_1 + 4\sum_{i=2,4,6,}^{n} y_i + 2\sum_{i=1,3,5,}^{n-1} y_i + y_{n+1}) \qquad (18)$$

Que es la ecuación conocida como la *regla de Simpson* y es la que se emplea cuando se integran ecuaciones analíticas. En esta ecuación el número de segmentos siempre tiene que ser par, pero ello no es un problema cuando se integran ecuaciones analíticas pues con el valor adecuado de *"h"* siempre es posible calcular de un número impar de puntos.

En esta ecuación y_1 es el valor de la ecuación analítica para el límite inferior *(a)*, y_2 el valor de la ecuación para *x+h*, y_3 el valor de la función para *x+2h* y así sucesivamente siendo y_{n+1} el valor de la ecuación para el límite superior *(b)*.

11.2.6. Algoritmo para la forma analítica

El algoritmo del método de Simpson para integrar ecuaciones analíticas se presenta en la Figura 10.8. En el mismo para asegurar que el número de segmentos sea siempre par, se le suma 1 en caso de ser impar. Por lo tanto si se manda al algoritmo un número impar de segmentos este se convierte en un número par.

Al igual que con el método del trapecio, los valores de la función (y_i) no se almacenan en un vector, sino que se calculan y suman directamente. Los valores correspondientes a los puntos impares se suman en la variable *SI*, mientras que los valores correspondientes a los puntos pares se suman en la variable *SP*.

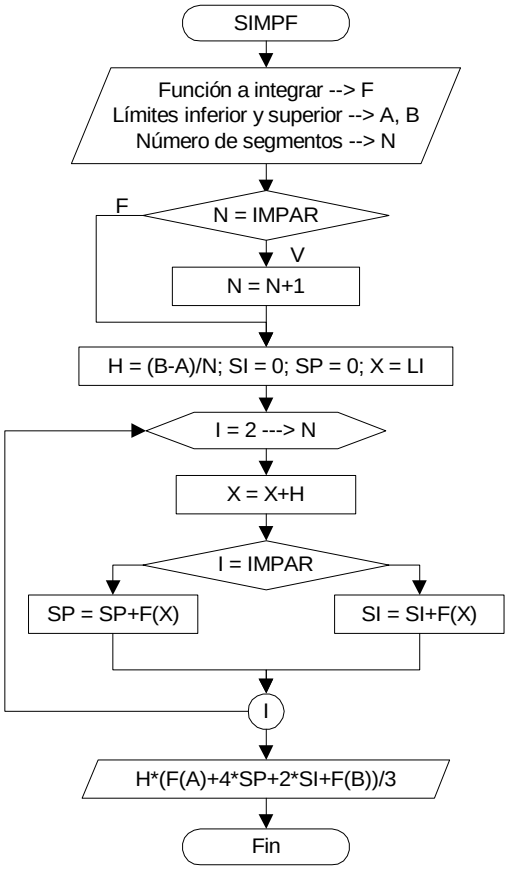


Figura 10.8. Algoritmo para integrar ecuaciones analíticas (funciones) con el método de Simpson.

11.2.7. Programa para la forma analítica

El programa para integrar ecuaciones analíticas (funciones), elaborado siguiendo el algoritmo de la Figura 10.8, es el siguiente:

```
« 0 0 0 0
  □ F A B N H SI SP X
 « IF N 2 MOD 0 ≠ THEN
     1 'N' STO+
   END
   B A - N / 'H' STO
   A 'X' STO
   2 N FOR I
     H 'X' STO+
     X F EVAL
     IF I 2 MOD 0 ≠ THEN
       'SI'
     ELSE
       'SP'
     END
     STO+
   NEXT
   H 3 / A F EVAL 4 SP * + 2 SI
    * + B F EVAL + *
 »
»
```

En lo sucesivo se asumirá que este programa ha sido almacenado en la variable *'SIMPF'*.

11.2.8. Ejemplos para la forma analítica

Ejemplo 1

Como primer ejemplo resolveremos la integral resuelta en el último ejemplo de la forma tabular:

$$\int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1})dx$$

Este ejemplo también fue resuelto con el método del trapecio y el programa para resolverlo con el método de Simpson es el mismo (excepto por el nombre del método):

```
«
  « → x « 3 6 x SQ * - 1.1 x 3.1 ^ * + » »
  5.1 8.4 100
  SIMPF
  "Integral" →TAG
»
```

Evaluando el programa se obtiene:

```
Integral: 528.719026846
```

Que es el resultado exacto, mientras que con el método del trapecio, para el mismo número de segmentos, el error relativo fue: 0.0026%.

Ejemplo 2

Como segundo ejemplo, para demostrar que es posible integrar datos tabulados empleando el método de Simpson en su forma analítica, calcularemos la primera integral resuelta con la forma tabular:

$$\int_{2.5}^{7.2} f(x)dx = \int_{2.5}^{7.2} ydx$$

Como en el método del trapecio, para integrar datos tabulados con la forma analítica se requiere una función que reciba la variable independiente (usualmente x) y devuelva el valor de la variable dependiente (usualmente y) empleando un método de interpolación.

Para resolver este ejemplo emplearemos el método de interpolación de Lagrange fijando el número de segmentos en 10.

```
« [ 2.5 2.7 3.0 3.3 3.8 4.2 4.5 5.1 5.7 6.0 6.4 6.9 7.2 ]
  [ 6.625 7.509 9 10.689 13.944 16.944 19.425 24.981 31.329 34.8
    39.736 46.401 50.664] 0
  → vx vy vc
  « vx COLAG 'vc' STO
    « vx vy vc 4 ROLL INLAG » 2.5 7.2 10
    SIMPF "Integral" →TAG
  »
»
```

Evaluando el programa se obtiene:

```
Integral: 115.594933333
```

Que es el valor exacto de la integral.

11.2.9. Preguntas y ejercicios

1. ¿Qué tipo de curva une los puntos de dos segmentos en el método de Simpson?
2. ¿Cómo se calcula, en el método de Simpson, el área de cada par de segmentos?
3. ¿Por qué se requieren tres puntos, en el método de Simpson, para calcular el área de dos segmentos?
4. ¿Cómo debe ser el número de segmentos en los que se divide el área bajo la curva en el método de Simpson?
5. ¿Por qué el método de Simpson es más exacto que el método del Trapecio?
6. ¿Qué se hace en el método de Simpson, cuando el número de puntos comprendidos entre los límites de integración es par?
7. ¿Es posible integrar funciones analíticas con el método de Simpson en su forma tabular?
8. ¿Por qué cuando se resuelven funciones analíticas con el método de Simpson en su forma tabular, es conveniente que el número de segmentos fijado sea par?
9. ¿Qué hace el programa *SIMPF* cuando el número de segmentos es impar?
10. ¿Al resolver la ecuación de la regla de Simpson, los valores de *"y"* se almacenan en un vector?
11. ¿Qué se debe hacer para integrar datos tabulados con la forma analítica?
12. Con los datos de la siguiente tabla, elabore un programa que calcule la integral: $\int_{2}^{42} f(x)dx$, empleando el método de Simpson en su forma tabular.

X	1	2	5	7	10	13	15	18	22	26	30	35	42
Y	0.3125	0.3333	0.3775	0.3901	0.4005	0.4065	0.4093	0.4124	0.4152	0.4172	0.4187	0.4200	0.4214

13. Repita el ejercicio 12 empleando el método de Simpson en su forma analítica conjuntamente el método de interpolación lineal.
14. Repita el ejercicio 12 empleando el método de Simpson en su forma analítica conjuntamente el método de interpolación de Newton.
15. Empleando los datos del ejercicio 12, elabore un programa que calcule la integral $\int_{1.5}^{37.2} f(x)dx$, empleando el método de Simpson en su forma tabular.
16. Elabore un programa que resuelve la integral: $\int_{0}^{2\pi} \frac{xdx}{1+\cos^2 x}$, empleando el método de Simpson en su forma analítica con 14 segmentos.
17. Repita el ejercicio 16 empleando el método de Simpson en su forma tabular.
18. Elabore un programa que resuelva la integral: $\int_{2}^{7} (3x^2+2x-5)dx$, empleando el método de Simpson, en su forma analítica, con 10 segmentos.

11.3. Método de Romberg

El método de Romberg es básicamente la regla del trapecio a la cual se aplica el método de extrapolación de Richardson.

En este método se aplica el método del trapecio para calcular inicialmente dos áreas: una tomando toda el área como un solo segmento $(I_{0,1})$ y la segunda dividiendo el área en dos segmentos $(I_{0,2})$. Entonces se aplica la fórmula de Richardson para extrapolar un valor a partir de estos dos valores calculados.

La fórmula de extrapolación de Richardson es la siguiente:

$$I_{n,k} = \frac{4^n I_{n-1,k+1} - I_{n-1,k}}{4^n - 1} \qquad (19)$$

Donde "n" es el nivel de extrapolación, "k" es el k-ésimo valor dentro del nivel al que pertenece $(n \text{ o } n-1)$, $I_{n-1,k+1}$ es el valor de la integral en el nivel de extrapolación $n-1$ y en la posición $k+1$, $I_{n-1,k}$, es el valor de la integral en el nivel de extrapolación $n-1$ y en la posición k.

Aplicando la ecuación (19) a los dos valores calculados con el método del trapecio (que como no son extrapolados pertenecen al nivel 0), se tiene:

$$I_{1,1} = \frac{4^1 I_{0,2} - I_{0,1}}{4^1 - 1}$$

Si este valor extrapolado $(I_{1,1})$ es igual, en un determinado número de dígitos, al último valor del nivel anterior $(I_{0,2})$, entonces el proceso concluye, siendo el valor de la integral el valor extrapolado $(I_{1,1})$. Caso contrario se vuelve a aplicar la regla del trapecio para calcular el valor de la integral $(I_{0,3})$, pero con un número de segmentos igual al doble del número de segmentos del cálculo anterior $(I_{0,2})$. Como en el último cálculo se emplearon 2 segmentos, en este nuevo cálculo se emplean 4.

Con el nuevo valor de la integral y el penúltimo valor de este nivel $(I_{0,2})$ se realiza otra extrapolación:

$$I_{1,2} = \frac{4^1 I_{0,3} - I_{0,2}}{4^1 - 1}$$

Con este valor tenemos ahora dos valores en el nivel de extrapolación 1 $(I_{1,1}$ e $I_{1,2})$, por lo que podemos efectuar otra extrapolación con los mismos:

$$I_{2,1} = \frac{4^2 I_{1,2} - I_{1,1}}{4^2 - 1}$$

Se comprueba entonces si este valor es igual, en un determinado número de dígitos, al último valor del nivel anterior $(I_{1,2})$ y de ser así el proceso concluye, siendo el valor de la integral el último valor extrapolado $(I_{2,1})$. Caso contrario se vuelve a calcular un nuevo valor de la integral con la regla del Trapecio $(I_{0,4})$, tomando un número de segmentos igual al doble del número de segmentos del cálculo anterior. Como en el cálculo anterior se emplearon 4 segmentos, ahora se emplean 8.

Con el nuevo valor de la integral y el penúltimo valor de este nivel $(I_{0,3})$ se realiza una nueva extrapolación:

$$I_{1,3} = \frac{4^1 I_{0,4} - I_{0,3}}{4^1 - 1}$$

Ahora en el nivel 1 tenemos tres valores, entonces con los dos últimos valores de este nivel *($I_{1,2}$ e $I_{1,3}$)* realizamos una nueva extrapolación:

$$I_{2,2} = \frac{4^2 I_{1,3} - I_{1,2}}{4^2 - 1}$$

Con lo que tenemos dos valores en el nivel 2 *($I_{2,1}$ e $I_{2,2}$)*. Entonces podemos llevar a cabo una nueva extrapolación:

$$I_{3,1} = \frac{4^3 I_{2,2} - I_{2,1}}{4^3 - 1}$$

Ahora comparamos este valor con el último valor del nivel anterior *($I_{2,2}$)* y si son iguales en un determinado número de dígitos el proceso concluye, siendo el resultado el último valor extrapolado, caso contrario se vuelve a repetir el proceso antes descrito, calculando la integral con un número de segmentos igual al doble del número de segmentos del cálculo anterior.

Por la descripción efectuada se deduce que el método de Romberg no puede ser aplicado directamente a datos tabulados a menos que los mismos estén igualmente espaciados y tengan un número de segmentos igual a una potencia de 2. (2,4,8,16, etc.), algo que ocurre muy raras veces en la práctica.

Sin embargo, es posible integrar datos tabulados con este método si se crea una función que reciba la variable independiente y empleando algún método de interpolación, devuelva el valor de la variable dependiente. Tal como se ha hecho con las formas analíticas de los métodos del Trapecio y Simpson.

Para comprender mejor el método de Romberg calcularemos la siguiente integral con 7 dígitos de exactitud:

$$\int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1})dx$$

Primero calculamos un valor de la integral con la *regla del trapecio* empleando un segmento. Entonces el incremento es igual a todo el intervalo de integración: *$h_1 = 8.4-5.1 = 3.3$*:

$$I_{0,1} = \frac{3.3}{2}(f(5.1) + f(8.4)) = \frac{3.3}{2}(18.675560187 + 386.240472262) = 668.111453539$$

Ahora calculamos un segundo valor empleando dos segmentos, es decir con un incremento igual a la mitad del incremento anterior: *$h_2 = h_1/2 = 3.3/2 = 1.65$*:

$$I_{0,2} = \frac{1.65}{2}(f(5.1) + 2f(6.75) + f(8.4)) = \frac{1.65}{2}(18.675560187 + 2*139.106589971 + 386.240472262)$$
$$= 563.58160022$$

Entonces con estos dos valores realizamos la primera extrapolación:

$$I_{1,1} = \frac{4^1 I_{0,2} - I_{0,1}}{4^1 - 1} = \frac{4*563.58160022 - 668.111453539}{3} = 528.73831578$$

Ahora comparamos este valor con el valor calculado previamente y vemos si son iguales en un determinado número de dígitos. Para este ejemplo compararemos los 7 primeros dígitos.

Para esta primera extrapolación, los dos últimos valores son iguales sólo en el primer dígito. Por lo tanto debemos calcular un nuevo valor de la in-

tegral con la *regla del trapecio*, empleando esta vez 4 segmentos, es decir empleando un incremento igual a la mitad del incremento anterior: $h_3 = h_2/2 = 1.65/2 = 0.825$:

$$I_{0,3} = \frac{0.825}{2}(f(5.1) + 2(f(5.925) + f(6.75) + f(7.575)) + f(8.4))$$
$$= \frac{1.65}{2}(18.675560187 + 2*(65.720974685 + 139.106589971 + 244.151489139) + 386.240472262)$$
$$= 537.435582765$$

Con los dos últimos valores de este nivel realizamos una nueva extrapolación:

$$I_{1,2} = \frac{4^1 I_{0,3} - I_{0,2}}{4^1 - 1} = \frac{4*537.435582768 - 563.58166022}{3} = 528.720243613$$

Y como ahora tenemos dos valores extrapolados en el nivel 1 ($I_{1,1}$ e $I_{1,2}$), podemos efectuar una segunda extrapolación:

$$I_{2,1} = \frac{4^2 I_{1,2} - I_{1,1}}{4^2 - 1} = \frac{16*528.720243613 - 528.73831578}{16 - 1} = 528.719038802$$

Entonces comparamos los dos últimos valores calculados ($I_{2,1}$ con $I_{1,2}$) y observamos que estos dos valores son iguales en los 4 primeros dígitos. Estamos en consecuencia más cerca del número de dígitos fijado, pero todavía es necesario repetir el proceso por lo menos una vez más.

Una vez más calculamos un nuevo valor de la integral empleando la *regla del trapecio* con 8 segmentos, es decir con un incremento igual a la mitad del incremento empleado en la anterior iteración: $h_4 = h_3/2 = 0.825/2 = 0.4125$:

$$I_{0,4} = \frac{0.4125}{2}(f(5.1) + 2(f(5.5125) + f(5.925) + f(6.3375) + f(6.75) + f(7.1625) + f(7.575) + f(7.9875)) + f(8.4))$$
$$= \frac{1.65}{2}(18.675560187 + 2*(39.235406609 + 65.720974685 + 98.790441668 + 139.106589971 + 187.336535733 +$$
$$+ 244.151489139 + 310.226541121) + 386.240472262)$$
$$= 530.898223$$

Ahora Con los dos últimos valores de este nivel realizamos una nueva extrapolación:

$$I_{1,3} = \frac{4^1 I_{0,4} - I_{0,3}}{4^1 - 1} = \frac{4*530.898223 - 537.435582765}{3} = 528.71910308$$

Igualmente con los dos últimos valores del nivel 1 se realiza una segunda extrapolación:

$$I_{2,2} = \frac{4^2 I_{1,3} - I_{1,2}}{4^2 - 1} = \frac{16*528.71910308 - 528.720243613}{15} = 528.719027045$$

Y con los dos últimos valores de este nivel realizamos una tercera extrapolación:

$$I_{3,1} = \frac{4^3 I_{2,2} - I_{2,1}}{4^3 - 1} = \frac{64*528.719027045 - 528.719038802}{63} = 528.719026859$$

Comparamos ahora estos dos últimos valores ($I_{3,1}$ con $I_{2,2}$) y podemos ver que los mismos son iguales en los 8 primeros dígitos, por lo que no sólo

hemos logrado la precisión deseada sino que la hemos superado. En consecuencia podemos afirmar que la integral buscada es 528.719026865 con 8 dígitos de exactitud. El resultado exacto es 528.719026846, el cual concuerda con el valor calculado en los 10 primeros dígitos.

11.3.1. Algoritmo

El algoritmo propuesto para el método de Romberg se presenta en la Figura 10.9.

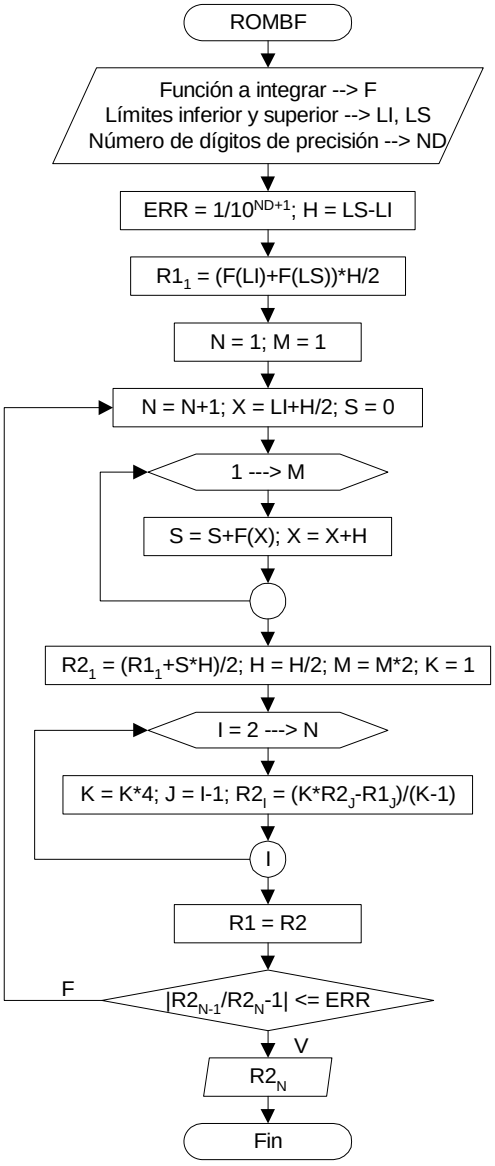


Figura 10.9. Algoritmo del método de Integración de Romberg.

Como se ha hecho notar en la explicación y se ha demostrado en el ejemplo, en el método de *Romberg* sólo se emplean los dos últimos valores de cada nivel, razón por la cual no es necesario almacenar todos los valores de un nivel, sino sólo los dos últimos. En el algoritmo se ha tomado en cuenta este hecho y por esa razón sólo se almacenan los dos últimos valores en los vectores: R_1 y R_2.

Además, como se puede ver en el ejemplo, cada vez que se calcula un nuevo valor de la integral (empleando la *regla del trapecio*), se vuelven a calcu-

lar valores que ya habían sido calculados en la iteración anterior. Así por ejemplo al calcular la cuarta integral *($I_{0,4}$)*, se vuelve a evaluar la función para valores de "x" iguales a: *5.1*, *5.925*, *6.75*, *7.575* y *8.4*, valores para los cuales la función ya fue evaluada en la iteración anterior (al calcular el valor *$I_{0,3}$*). Este hecho ha sido también tomado en cuenta en el algoritmo donde no se hace una llamada directa al método del trapecio *(TRAPF)*, sino que sólo se calculan los nuevos valores (en el primer ciclo *For*) y se reutilizan los valores calculados en la iteración anterior (los cuales están guardados en el primer elemento del vector *R1)*.

11.3.2. Programa

El programa elaborado siguiendo el algoritmo de la Figura 10.9, es el siguiente:

```
« 1 1 0 0 0 0 0 0 { 15 } 0. CON
  DUP □ F LI LS ND N M ERR H X
  S K J R1 R2
  « ND 1 + ALOG INV 'ERR' STO
    LS LI - 'H' STO
    R1 1 LI F EVAL LS F EVAL +
     H * 2 / PUT 'R1' STO
    DO
      1 'N' STO+
      LI H 2 / + 'X' STO
      0 'S' STO
      1 M START
        X F EVAL 'S' STO+
        H 'X' STO+
      NEXT
      R2 1 R1 1 GET S H * + 2 /
       PUT 'R2' STO
      'H' 2 STO/
      'M' 2 STO*
      1 'K' STO
      2 N FOR I
        'K' 4. STO*
        I 1. - 'J' STO
        R2 I K R2 J GET * R1 J
         GET - K 1 - / PUT 'R2'
         STO
      NEXT
      R2 'R1' STO
    UNTIL
      R2 N 1 - GET R2 N GET
      / 1. - ABS ERR ≤
    END
    R2 N GET
  »
»
```

En lo sucesivo se asumirá que este programa ha sido guardado en la variable *"ROMBF"*.

Como se ha comprobado en la práctica que muy rara vez se requieren más de 10 extrapolaciones, en el programa sólo se reserva espacio para 14 extrapolaciones. Es por esta razón que los vectores R1 y R2 son creados con 15 elementos cada uno (uno más para el nivel de extrapolación 0).

11.3.3. Ejemplos

Ejemplo 1

Como primer ejemplo resolveremos la integral resuelta manualmente:

$$\int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1})dx$$

Empleando en esta ocasión el programa *ROMBF* y calculando la solución con 9 dígitos de exactitud.

Para ello escribimos el siguiente programa:

```
«
  « → x « 3 6 x SQ * - 1.1 x 3.1 ^ * + » »
  5.1 8.4 9
  ROMBF
  "Integral" →TAG
»
```

Que como vemos es muy similar a los programas elaborados para los métodos de Simpson y del Trapecio. Haciendo correr el programa se obtiene:

Integral: 528.719026839

Que prácticamente es el resultado exacto.

Ejemplo 2

Como segundo ejemplo calcularemos la siguiente integral, empleando el método del trapecio conjuntamente el método de Lagrange.

$$\int_{2.5}^{7.2} f(x)dx = \int_{2.5}^{7.2} ydx$$

Esta integral fue resuelta previamente con los métodos de Simpson y del Trapecio. El programa para el método de Romberg es prácticamente el mismo (con excepción de los dígitos de exactitud y el nombre):

```
« [ 2.5 2.7 3.0 3.3 3.8 4.2 4.5 5.1 5.7 6.0 6.4 6.9 7.2 ]
  [ 6.625 7.509 9 10.689 13.944 16.944 19.425 24.981 31.329 34.8
    39.736 46.401 50.664] 0
  → vx vy vc
  « vx COLAG 'vc' STO
    « vx vy vc 4 ROLL INLAG » 2.5 7.2 9
    ROMBF "Integral" →TAG
  »
»
```

Haciendo correr el programa se obtiene:

Integral: 115.594933333

Que es el resultado buscado y que concuerda exactamente con el valor calculado con el método de Simpson.

11.3.4. Preguntas y ejercicios

1. ¿De qué métodos consta el método de Romberg?
2. En la fórmula de extrapolación de Richardson ¿Qué són los índices *"n"* y *"k"*?

3. En el método de Romberg, cada vez que se calcula el valor de una nueva integral con la regla del trapecio ¿Cuantos segmentos se toman para efectuar dicho cálculo?

4. Cada vez que se emplea la fórmula de extrapolación de Richardson ¿Cuáles son los valores que se extrapolan?

5. ¿Cuándo concluye el proceso de extrapolación en el método de Romberg?

6. ¿Cuál es el valor que se toma como resultado (valor de la integral) en el método de Romberg?

7. ¿El método de Romberg puede ser aplicado directamente en la integración de datos tabulados?

8. ¿Es posible integrar datos tabulados con el método de Romberg?

9. En el método de Romberg, ¿Es necesario almacenar todos los valores de todos los niveles de extrapolación?

10. En el método de Romberg, ¿Cada vez que se calcula un nuevo valor de la integral con la *regla del trapecio* es necesario evaluar la función para todos los puntos de los segmentos?

11. Elabore un programa que resuelve la integral: $\int_0^{2\pi} \frac{x dx}{1+\cos^2 x}$, empleando el método de Romberg en su forma analítica con 10 dígitos de exactitud.

12. Elabore un programa que resuelva la integral: $\int_2^7 (3x^2+2x-5)dx$, empleando el método de Romberg con 8 dígitos de exactitud.

13. Elabore un programa que calcule la integral: $\int_{0.1}^{0.9} f(x)dx$, empleando el método de Romberg con 9 dígitos de exactitud conjuntamente el método de interpolación de Newton. Los datos que definen esta función son los siguientes.

x	0	0.2	0.4	0.6	0.8	1.0
y	0	0.199	0.389	0.565	0.717	0.841

14. Con los datos del anterior ejercicio calcule la integral: $\int_{0.24}^{0.98} f(x)dx$, empleando el método de Romberg con 8 dígitos de exactitud, conjuntamente el método de interpolación lineal.

12. ECUACIONES DIFERENCIALES ORDINARIAS

Con frecuencia en la solución de problemas en el campo de la ingeniería es necesario resolver una o más ecuaciones diferenciales. En este capítulo estudiaremos algunos de los métodos que nos permiten resolver dichas ecuaciones.

Una ecuación diferencial ordinaria de enésimo grado puede ser representada en la siguiente forma:

$$y^n = f(x, y, y', y'', y''', ..., y^{n-1}) \tag{1}$$

Donde *"x"* es la variable independiente, *"y"* la variable dependiente y *y', y'' a y'''*, son las derivadas: $y'=dy/dx$; $y''=d^2y/dx^2$; $y'''=d^3y/dx^3$, *etc.*

Una ecuación diferencial es resuelta cuando se calcula el valor de la variable dependiente *"y"* para un determinado valor de la variable independiente *"x"*.

En las ecuaciones diferenciales, el cálculo de la variable dependiente no es directo pues se desconoce la forma que tiene la función $y = f(x)$. Es esta función la que constituye la solución cuando una ecuación diferencial es resuelta analíticamente.

Puesto que existe un número infinito de funciones y cada una de ellas puede dar lugar a un número infinito de ecuaciones diferenciales, el número de posibles casos es infinito.

Analíticamente es posible encontrar la solución de algunas ecuaciones diferenciales típicas, pero no existe un método analítico general para resolver ecuaciones diferenciales. Esto sumado a lo tediosas que resultan las operaciones involucradas, ha hecho que la solución de ecuaciones diferenciales en el campo de la ingeniería, sea llevada a cabo casi exclusivamente con métodos numéricos implementados en una computadora.

Para ilustrar como una función puede dar lugar a un número infinito de ecuaciones diferenciales tomemos por ejemplo la siguiente función:

$$y = f(x) = 3x^3 + 2x^2 + 5x \tag{2}$$

Las derivadas primera, segunda y tercera de esta función son:

$$y^i = 9x^2 + 4x + 5 \tag{3}$$

$$y^{ii} = 18x + 4 \tag{4}$$

$$y^{iii} = 18 \tag{5}$$

Con estas derivadas y la función original podemos generar por ejemplo las siguientes ecuaciones diferenciales:

$$\begin{aligned} &2xy^i + 3y^{ii} = 2x(9x^2 + 4x) + 3(18x + 4) = 18x^3 + 8x^2 + 10x + 54x + 12 \\ &2xy^i + 3y^{ii} - 18x^3 - 64x - 12 = 0 \end{aligned} \tag{6}$$

$$\begin{aligned} &3yy^{iii} - 2y^{ii} = 3y(8) - 2(8x + 4) = 54y - 36x - 8 \\ &3yy^{iii} - 2y^{ii} - 54y + 36x + 8 = 0 \end{aligned} \tag{7}$$

La solución de estas dos ecuaciones es por supuesto la ecuación (2). De manera similar se pueden generar una infinidad de ecuaciones diferenciales y de todas ellas la solución será la ecuación (2).

En un caso real no se conoce la función (la ecuación 2 en el ejemplo), sino sólo las ecuaciones diferenciales (ecuaciones 6 y 7) y con las mismas se debe calcular el valor de la variable dependiente *"y"* para un determinado valor de la variable independiente *"x"*.

El procedimiento antes ilustrado es útil para deducir ecuaciones diferenciales y con ellas comprobar la exactitud de un determinado método.

12.1. Problemas del valor inicial y problemas del valor límite

En general al resolver ecuaciones diferenciales ordinarias se presentan dos tipos de problemas:

Problemas del valor inicial: cuando se conoce el valor de la variable dependiente y de todas sus derivadas, para un valor *inicial* dado de la variable independiente.

Problemas del valor límite: Cuando algunos valores de la variable dependiente y de sus derivadas se conocen para el valor inicial y otros para el valor final de la variable independiente.

Los métodos que estudiemos en este capítulo permiten resolver problemas del valor inicial. Estos métodos combinados con otros métodos iterativos (como el de Newton – Raphson) permiten resolver también los problemas del valor límite.

De los muchos métodos disponibles para resolver los problemas del valor inicial estudiaremos los métodos de *Euler* y *Runke-Kutta*. El primero porque dada su sencillez permite comprender el procedimiento general que se sigue en la resolución de ecuaciones diferenciales y el segundo porque es uno de los métodos más empleado en la práctica.

12.2. Método de Euler

El método de Euler, como todos los métodos numéricos existentes sólo permite resolver ecuaciones diferenciales de primer orden, pues como demostraremos posteriormente, una ecuación diferencial de enésimo orden puede ser transformada en un sistema de *"n"* ecuaciones diferenciales de primer orden.

Por esta razón estudiaremos en primer lugar la solución de ecuaciones diferenciales de primer orden.

12.2.1. Ecuaciones diferenciales de primer orden

Consideremos la siguiente ecuación diferencial de primer orden:

$$y^i = f(x,y) \tag{8}$$

Para los problemas del valor inicial, se conoce el valor de la variable dependiente (y_0) para un valor inicial de la variable independiente (x_0).

El problema radica en calcular el valor de la variable dependiente (y_n) para un determinado valor de la variable independiente (x_n). Si se conoce la forma de la curva *y'* versus *f(x,y)* la solución del problema se encuentra calculando el área bajo la curva comprendida entre el valor inicial (x_0) y el valor final (x_n), con uno de los métodos de integración estudiados en el anterior capítulo:

$$\int_{y_0}^{y_n} dy = y_n - y_0 = \int_{x_0}^{x_n} f(x,y)dx \;\Rightarrow\; y_n = \int_{x_0}^{x_n} f(x,y)dx + y_0 \tag{9}$$

Como en la práctica no se conoce la forma de la curva, lo que hacen los diferentes métodos es asumir una forma para la curva y emplearla para calcular el área. La forma de curva asumida es lo que diferencia a un método del otro.

Así en el método de *Euler* se divide el área comprendida entre x_0 y x_n en un número de segmentos igualmente espaciados y se asume que el área de cada uno de los segmentos es un rectángulo (ver Figura 11.1).

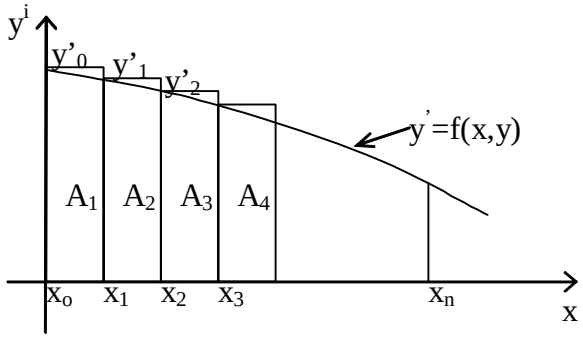


Figura 11.1. Cálculo del área con el método de Euler

Como el ancho de los segmentos es conocido $(h=(x_n-x_0)/n)$ el área del primer segmento (A_1) se calcula multiplicando la altura del rectángulo (y'_0) por su ancho (h). La altura (y'_0) se calcula sustituyendo los valores iniciales (x_0, y_0) en la ecuación (8):

$$y'_0 = f(x_0, y_0) \tag{10}$$

Entonces el valor de la variable dependiente en el segundo punto (y_1), se calcula con:

$$\int_{y_0}^{y_1} dy = y_1 - y_0 \approx A_1 = y'_0 * h \quad \Rightarrow \quad y_1 = y_0 + y'_0 * h \tag{10}$$

El valor de la variable independiente en este punto (x_1) es $x_1=x_0+h$. Ahora con este nuevo punto y la ecuación (8), calculamos el valor de y'_1:

$$y'_1 = f(x_1, y_1) \tag{11}$$

Entonces podemos calcular el área del segundo segmento (A_2) y con este el valor de y_2:

$$\int_{y_1}^{y_2} dy = y_2 - y_1 \approx A_2 = y'_1 * h \quad \Rightarrow \quad y_2 = y_1 + y'_1 * h \tag{12}$$

Siendo el valor de $x_2 = x_1+h$. Con este nuevo punto podemos calcular A_3 y con este valor y_3 prosiguiendo de esta manera hasta llegar al segmento A_n y consiguientemente el valor de y_n, que es la solución buscada.

Entonces para cualquier punto conocido (x_i, y_i), podemos calcular los valores de un nuevo punto (x_{i+1}, y_{i+1}) con:

$$\begin{aligned} y'_i &= f(x_i, y_i) \\ y_{i+1} &= y_i + y'_i\, h \\ x_{i+1} &= x_i + h \end{aligned} \tag{13}$$

Que son las ecuaciones del método de Euler.

Puesto que en el método de Euler se asumen áreas rectangulares para los segmentos se comete un error apreciable en los cálculos. Por esta razón, el

método de Euler no se emplea normalmente en la solución de problemas reales, sin embargo, el procedimiento general que se sigue en este método es la base de otros métodos más elaborados.

Si en lugar de asumir áreas rectangulares para los segmentos se asumen áreas trapezoidales, la exactitud del método de Euler mejora considerablemente. Cuando se lleva a cabo esta modificación, las ecuaciones resultantes se conocen con el nombre de ***método de Euler modificado.***

12.2.1.1. Ejemplo manual

Para comprender mejor el método de ***Euler*** resolveremos manualmente la ecuación diferencial de primer orden:

$$3y'+4xy-8x^3+4x=0 \quad \begin{cases} y=4 \; para \; x=2 \\ y=? \; para \; x=6 \end{cases}$$

Donde, como se puede ver los valores iniciales son $x_0=2$, $y_0=4$ y se quiere calcular el valor de *"y"* para *"x=6"*.

Fijamos primero el número de segmentos en 10 (n=10), por lo tanto el incremento (o espacio entre segmentos) es:

$$h=\frac{6-2}{10}=0.4$$

Con los valores iniciales $(x_0=2, y_0=4)$ aplicamos las ecuaciones (13) al primer segmento:

$$y'_0=f(x_0,y_0)=\frac{8x_0^3-4x_0-4x_0y_0}{3}=\frac{8(2)^3-4(2)-4(2)(4)}{3}=8$$
$$y_1=y_0+y'_0\,h=4+8(0.4)=7.2$$
$$x_1=x_0+h=2+0.4=2.4$$

Con el nuevo punto $(x_1=2.4, y_1=7.2)$ aplicamos la ecuación (13) al segundo segmento:

$$y'_1=f(x_1,y_1)=\frac{8x_1^3-4x_1-4x_1y_1}{3}=\frac{8(2.4)^3-4(2.4)-4(2.4)(7.2)}{3}=10.624$$
$$y_2=y_1+y'_1\,h=7.2+10.624(0.4)=11.4496$$
$$x_2=x_1+h=2.4+0.4=2.8$$

Entonces con este nuevo punto $(x_2=2.8, y_2=11.4496)$ aplicamos la ecuación (13) al tercer segmento:

$$y'_2=f(x_2,y_2)=\frac{8x_2^3-4x_2-4x_2y_2}{3}=\frac{8(2.8)^3-4(2.8)-4(2.8)(11.4496)}{3}=12.06016$$
$$y_3=y_2+y'_2\,h=11.4496+12.06016(0.4)=16.273664$$
$$x_3=x_2+h=2.8+0.4=3.2$$

Procediendo de la misma forma con los segmentos restantes obtenemos:

$$y'_3=\frac{8x_3^3-4x_3-4x_3y_3}{3}=\frac{8(3.2)^3-4(3.2)-4(3.2)(16.273664)}{3}=13.6803669333$$
$$y_4=y_3+y'_3\,h=16.273664+13.6803669333(0.4)=21.7458107733$$
$$x_4=x_3+h=3.2+0.4=3.6$$

$$y'_4 = \frac{8x_4^3 - 4x_4 - 4x_4y_4}{3} = \frac{8(3.6)^3 - 4(3.6) - 4(3.6)(21.7458107733)}{3} = 15.236108288$$
$$y_5 = y_4 + y'_4\,h = 21.7458107733 + 15.236108288(0.4) = 27.8402540885$$
$$x_5 = x_4 + h = 3.6 + 0.4 = 4$$

$$y'_5 = \frac{8x_5^3 - 4x_5 - 4x_5y_5}{3} = \frac{8(4)^3 - 4(4) - 4(4)(27.8402540885)}{3} = 16.8519781947$$
$$y_6 = y_5 + y'_5\,h = 27.8402540885 + 16.8519781947(0.4) = 34.5810453664$$
$$x_6 = x_5 + h = 4 + 0.4 = 4.4$$

$$y'_6 = \frac{8x_6^3 - 4x_6 - 4x_6y_6}{3} = \frac{8(4.4)^3 - 4(4.4) - 4(4.4)(34.5810453664)}{3} = 18.415200517$$
$$y_7 = y_6 + y'_6\,h = 27.8402540885 + 18.415200517(0.4) = 41.9471255732$$
$$x_7 = x_6 + h = 4.4 + 0.4 = 4.8$$

$$y'_7 = \frac{8x_7^3 - 4x_7 - 4x_7y_7}{3} = \frac{8(4.8)^3 - 4(4.8) - 4(4.8)(41.9471255732)}{3} = 20.0503963317$$
$$y_8 = y_7 + y'_7\,h = 41.9471255732 + 20.0503963317(0.4) = 49.9672841059$$
$$x_8 = x_7 + h = 4.8 + 0.4 = 5.2$$

$$y'_8 = \frac{8x_8^3 - 4x_8 - 4x_8y_8}{3} = \frac{8(5.2)^3 - 4(5.2) - 4(5.2)(49.9672841059)}{3} = 21.5814968667$$
$$y_9 = y_8 + y'_8\,h = 49.9672841059 + 21.5814968667(0.4) = 58.5998828526$$
$$x_9 = x_8 + h = 5.2 + 0.4 = 5.6$$

$$y'_9 = \frac{8x_9^3 - 4x_9 - 4x_9y_9}{3} = \frac{8(5.6)^3 - 4(5.6) - 4(5.6)(58.5998828526)}{3} = 23.2968747$$
$$y_{10} = y_9 + y'_9\,h = 58.5998828526 + 23.2968747(0.4) = 67.9186327326$$
$$x_{10} = x_9 + h = 5.6 + 0.4 = 6$$

Por consiguiente el valor de *"y"* para *"x=6"* es 67.9186327326. El resultado exacto es 68, por consiguiente el error relativo cometido con el método de Euler es del 0.12%.

12.2.1.2. Algoritmo

El algoritmo del método de Euler, que básicamente consiste en repetir los cálculos de la ecuación (13) en un ciclo FOR, se presenta en la Figura 11.2.

12.2.1.3. Código

El código elaborado siguiendo el algoritmo de la Figura 11.2 es el siguiente:

```
« 0 □ F Y X XN N H
 « XN X - N / 'H' STO
   1 N START
     X Y F EVAL H * 'Y' STO+
     H 'X' STO+
   NEXT
   Y
 »
»
```

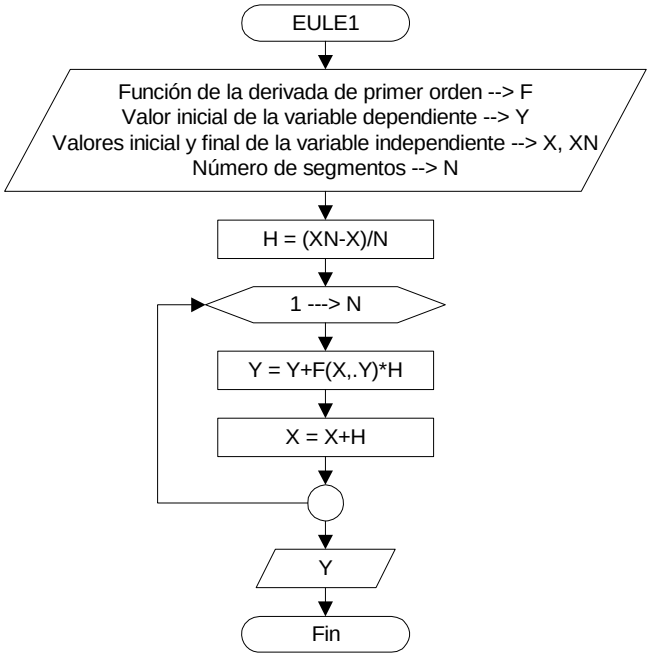


Figura 11.2. Algoritmo del método de Euler para ecuaciones diferenciales de primer orden.

En lo sucesivo se asumirá que el programa del método ha sido guardado en la variable ***"EULE1"***. Este programa requiere como datos (y en ese orden): la función de dos variables que corresponde a la derivada primera ***(y')***, el valor inicial de la variable dependiente, los valores inicial y final de la variable independiente y el número de segmentos. Devuelve como resultado el valor de la variable dependiente correspondiente al valor final de la variable independiente.

Para ilustrar la aplicación este programa resolveremos el ejemplo manual. Para ello elaboramos el siguiente programa:

```
«
  « → x y « 8 x 3 ^ * 4 x * - 4 x * y * - 3 / » »
  4 2 6 10 EULE1 "y" →TAG
»
```

Haciendo correr el programa se obtiene el resultado:

```
y: 67.9186327326
```

Que es el mismo valor calculado manualmente. Sin embargo, como ahora resolvemos el problema con la ayuda de un programa podemos mejorar la exactitud del resultado simplemente incrementando el número de segmentos. Así si en lugar de 10 segmentos empleamos 50:

```
«
  « □ x y « 8 x 3 ^ * 4 x * - 4 x * y * - 3 / » »
  4 2 6 50 EULE1 "y" □TAG
»
```

El resultado obtenido es:

```
y: 67.9795610066
```

Cuyo error relativo es 0.03%. Cuanto más alejado se encuentre el valor inicial del valor final, mayor deberá ser el número de segmentos a emplear. Así si se quiere calcular el valor de ***"y"*** para ***"x=15"*** y se toman 10 segmentos:

```
«
  « → x y « 8 x 3 ^ * 4 x * - 4 x * y * - 3 / » »
  4 2 15 10 EULE1 "y" →TAG
»
```

Se obtiene el resultado:

y: 17938851982.2

Que ni siquiera se parece al resultado correcto: 446. Pero si en lugar de 10 segmentos se emplean 100:

```
«
  « → x y « 8 x 3 ^ * 4 x * - 4 x * y * - 3 / » »
  4 2 15 100 EULE1 "y" →TAG
»
```

Se obtiene el resultado

y: 445.98696035

Que tiene un error relativo igual a 0.003%.

Cuando en un problema es necesario resolver varias veces la misma ecuación diferencial para diferentes valores de la variable independiente, resulta más eficiente aplicar el método una sola vez (para el mayor valor posible de la variable independiente) y guardar los valores intermedios en dos vectores. Entonces se emplean estos vectores, conjuntamente un método de interpolación, para calcular los valores de la variable dependiente correspondientes a cualquier valor intermedio de la variable independiente.

Por supuesto ello implica modificar el programa del método de Euler de manera que devuelva estos vectores en lugar del último valor de la variable independiente. Un programa modificado para este fin es el siguiente:

```
« 0 1 □ F Y X XN N H I
  « XN X - N / 'H' STO
    N 1 + 1 □LIST 0 CON DUP
    I X PUT SWAP I Y PUT
    1 N START
      X Y F EVAL H * 'Y' STO+
      H 'X' STO+
      1 'I' STO+
      I Y PUT
      SWAP I X PUT
      SWAP
    NEXT
  »
»
```

Que en lo sucesivo se asumirá ha sido guardado en la variable "EULE2". Resolviendo el ejemplo manual con este programa:

```
«
  « □ x y « 8 x 3 ^ * 4 x * - 4 x * y * - 3 / » »
  4 2 6 10 EULE2 "y" □TAG SWAP "x" □TAG
»
```

Se obtiene:

y: [4. 7.2 11.4496 16.273664 21.7458107733 27.8402540885 34.5810453664 41.9471255732 49.9672841059 58.5998828526 67.9186327326]

x: [2. 2.4 2.8 3.2 3.6 4. 4.4 4.8 5.2 5.6 6.]

Que son los resultados intermedios obtenidos en el ejemplo manual. Si en un problema dado, el máximo valor de *"x"* es 6, entonces se pueden emplear estos vectores para calcular por interpolación, otros valores de la variable dependiente.

Por ejemplo, en el siguiente programa se calculan valores de *"y"* para valores de *"x"* iguales a 2.5, 3, 4, 4.7, 5.1, 5.3 y 5.9 empleando los vectores obtenidos con el programa *EULE2* y el método de interpolación de Newton:

```
« [ 2.5 3 4 4.7 5.1 5.3 5.9 ] 0 0 0
  □ vcx vx vy vc
  «
    « □ x y « 8 x 3 ^ * 4 x * - 4 x * y * - 3 / » »
    4 2 6 10 EULE2 'vy' STO 'vx' STO
    vx vy CONEW 'vc' STO
    1 7 FOR i
      vx vc vcx i GET INNEW
    NEXT
    7 □ARRY "y" □TAG
  »
»
```

Haciendo correr el programa se obtiene:

y: [8.2228655641 13.7778568316 27.8402540885 40.0472814476 47.8974048064 52.0781141721 65.4091922268]

Que son los valores de la variable dependiente para x = 2.5, 3, 4, 4.7, 5.1, 5.3 y 5.9 respectivamente.

12.2.2. Ecuaciones diferenciales de enésimo orden

Como se mencionó previamente, una ecuación de enésimo orden puede ser transformada en un sistema de ecuaciones diferenciales de primer orden. Para ilustrar el procedimiento consideremos la siguiente ecuación diferencial:

$$3x\frac{d^4y}{dx^4}+(x-y)\frac{d^3y}{dx^3}+5\frac{d^2y}{dx^2}-\frac{dy}{dx}-\frac{x^2-y^2}{xy}=3xy''''+(x-y)y'''+5y''-y'-\frac{x^2-y^2}{xy}0 \qquad (14)$$

Esta ecuación puede ser transformada en un sistema de 4 ecuaciones diferenciales de primer orden realizando los siguientes cambios de variable:

$$\begin{aligned} y &= y_1 \\ \frac{dy}{dx} &= \frac{dy_1}{dx} = y'_1 = y_2 \\ \frac{d^2y}{dx^2} &= \frac{d}{dx}\left(\frac{dy}{dx}\right) = \frac{d}{dx}(y_2) = y'_2 = y_3 \\ \frac{d^3y}{dx^3} &= \frac{d}{dx}\left(\frac{d^2y}{dx^2}\right) = \frac{d}{dx}(y_3) = y'_3 = y_4 \\ \frac{d^4y}{dx^4} &= \frac{d}{dx}\left(\frac{d^3y}{dx^3}\right) = \frac{d}{dx}(y_4) = y'_4 = y_5 \end{aligned} \qquad (15)$$

Donde del primer cambio de variable se hace simplemente para diferenciar la variable dependiente *(y)* del vector con las nuevas variables (y = y_1, y_2, y_3, y_4, y_5). Por consiguiente en lugar de la ecuación (14) podemos escribir el siguiente sistema de ecuaciones diferenciales de primer orden:

$$
\begin{aligned}
y'_1 &= y_2 \\
y'_2 &= y_3 \\
y'_3 &= y_4 \\
y'_4 &= f_4(x, y_1, y_2, y_3, y_4) = \frac{(y_1 - x)y_4 - 5y_3 + y_2 + (x^2 - y_1^{\,2})/xy_1}{3x} = y_5
\end{aligned}
\tag{16}
$$

Como estamos resolviendo los problemas del valor inicial, son datos los valores iniciales de y_1, y_2, y_3 y y_4 para un valor inicial dado de *"x"*.

Las ecuaciones de Euler (ecuaciones 13) pueden ser aplicadas a cada una de las ecuaciones de este sistema tomando en cuenta las nuevas variables (ecuaciones 15 y 16) y empleando en todos los casos el mismo número de segmentos. Entonces aplicando las ecuaciones de Euler a un punto *"j"* cualquiera se tiene:

$$
\begin{aligned}
y_{1,j+1} &= y_{1,j} + y'_{1,j} * h = y_{1,j} + y_{2,j} * h \\
y_{2,j+1} &= y_{2,j} + y'_{2,j} * h = y_{2,j} + y_{3,j} * h \\
y_{3,j+1} &= y_{3,j} + y'_{3,j} * h = y_{3,j} + y_{4,j} * h \\
y_{4,j+1} &= y_{4,j} + y'_{4,j} * h = y_{4,j} + y_{5,j} * h \\
y_{5,j} &= f(x_j, y_{1,j}, y_{2,j}, y_{3,j}, y_{4,j}) = \frac{(y_{1,j} - x_j)y_{4,j} - 5y_{3,j} + y_{2,j} + (x_j^{\,2} - y_{1,j}^{\,2})/x_j y_{1,j}}{3x} \\
x_{j+1} &= x_j + h
\end{aligned}
\tag{17}
$$

Como se puede observar, la única función que realmente necesita ser evaluada es la correspondiente a la derivada de enésimo grado (y_5 en el ejemplo), pues las otras funciones son simplemente los valores de las derivadas, las mismas que se conocen del punto anterior (o son los valores iniciales del problema).

Generalizando el procedimiento para una ecuación de enésimo grado, en un punto *"j"* cualquiera se tiene:

$$
\begin{aligned}
y_{i,j+1} &= y_{i,j} + y'_{i,j} * h = y_{i,j} + y_{i+1,j} * h \quad \{i = 1 \to n \\
y_{n+1,j} &= f(x_j, y_{1,j}, y_{2,j}, y_{3,j}, \ldots, y_{n,j}) \\
x_{j+1} &= x_j + h
\end{aligned}
\tag{18}
$$

Donde las variables y_i corresponden a las siguientes igualdades:

$$
\begin{aligned}
y &= y_1 \\
\frac{d^i y}{dx^i} &= y_{i+1} \\
y'_i &= y_{i+1}
\end{aligned}
\tag{19}
$$

12.2.2.1. Ejemplo manual

Par comprender mejor el procedimiento que se sigue al resolver una ecuación diferencial de enésimo grado, resolveremos la siguiente ecuación diferencial de cuarto grado:

$$
\begin{aligned}
& 3y'''' - 3xy'' + y' + 1.8711x^{1.9} + 14.49x^{1.1} - 5 = 0 \\
& \begin{cases} y = 4,\ y' = 11.3,\ y'' = 6.93,\ y''' = 0.693 \ para\ x = 1 \\ y = ?,\ y' = ?,\ y'' = ?,\ y''' = ?\ para\ x = 4 \end{cases}
\end{aligned}
$$

La solución analítica de esta ecuación es: $y=3x^{2.1}+5x-4$.

Primero efectuamos los cambios de variables y transformamos esta ecuación en un sistema de ecuaciones diferenciales de primer orden:

$$y = y_1$$
$$\frac{dy}{dx} = \frac{dy_1}{dx} = y'_1 = y_2$$
$$\frac{d^2y}{dx^2} = \frac{dy_2}{dx} = y'_2 = y_3$$
$$\frac{d^3y}{dx^3} = \frac{dy_3}{dx} = y'_3 = y_4$$
$$\frac{d^4y}{dx^4} = \frac{dy_4}{dx} = y'_4 = y_5 = \frac{1}{3}(5 - 14.49x^{1.1} - 1.8711x^{-1.9} - y_2 + 3xy_3)$$
$$\begin{cases} y_1 = 4,\ y_2 = 11.3,\ y_3 = 6.93,\ y_4 = 0.693\ para\ x = 1 \\ y_1 = ?,\ y_2 = ?,\ y_3 = ?,\ y_4 = ?\ para\ x = 1.43 \end{cases}$$

Para este ejemplo fijaremos el número de segmentos en 10, entonces el incremento es:

$$h = \frac{1.43 - 1}{10} = 0.043$$

Aplicamos ahora la ecuación (18) con ***n=4,*** comenzando con los valores iniciales (punto cero):

$$y_{1,1} = y_{1,0} + y'_{1,0} * h = y_{1,0} + y_{2,0} * h = 4 + 11.3 * 0.043 = 4.4859$$
$$y_{2,1} = y_{2,0} + y'_{2,0} * h = y_{2,0} + y_{3,0} * h = 11.3 + 6.93 * 0.043 = 11.59799$$
$$y_{3,1} = y_{3,0} + y'_{3,0} * h = y_{3,0} + y_{4,0} * h = 6.93 + 0.693 * 0.043 = 6.959799$$
$$y'_{4,0} = y_{5,0} = \frac{1}{3}\left(5 - 14.49x_0^{1.1} - 1.8711x_0^{-1.9} - y_{2,0} + 3x_0y_{3,0}\right)$$
$$= \frac{1}{3}\left(5 - 14.49(1)^{1.1} - 1.8711(1)^{-1.9} - 11.3 + 3(1)(6.93)\right) = -0.6237$$
$$y_{4,1} = y_{4,0} + y'_{4,0} * h = y_{4,0} + y_{5,0} * h = 0.693 + (-0.6237) * 0.043 = 0.6661809$$
$$x_1 = x_0 + h = 1 + 0.043 = 1.043$$

Contamos ahora con todos los valores del punto 1, por lo que podemos pasar al siguiente punto:

$$y_{1,2} = y_{1,1} + y'_{1,1} * h = y_{1,1} + y_{2,1} * h = 4.4859 + 11.59799 * 0.043 = 4.98461357$$
$$y_{2,2} = y_{2,1} + y'_{2,1} * h = y_{2,1} + y_{3,1} * h = 11.5979 + 6.959799 * 0.043 = 11.897261357$$
$$y_{3,2} = y_{3,1} + y'_{3,1} * h = y_{3,1} + y_{4,1} * h = 6.959799 + 0.6661809 * 0.043 = 6.9884447787$$
$$y'_{4,1} = y_{5,1} = \frac{1}{3}\left(5 - 14.49x_1^{1.1} - 1.8711x_1^{-1.9} - y_{2,1} + 3x_1y_{3,1}\right)$$
$$= \frac{1}{3}\left(5 - 14.49(1.043)^{1.1} - 1.8711(1.043)^{-1.9} - 11.59799 + 3(1.043)(6.959799)\right) = -0.5749557561$$
$$y_{4,2} = y_{4,1} + y'_{4,1} * h = y_{4,1} + y_{5,1} * h = 0.6661809 + (-0.5749557561) * 0.043 = 0.641457802488$$
$$x_2 = x_1 + h = 1.043 + 0.043 = 1.086$$

Ahora contamos con todos los valores del punto 2, por lo que podemos pasar al siguiente punto. Prosiguiendo de esta manera: calculando los valores del siguiente punto con los valores calculados en el punto anterior, obtenemos los siguientes resultados:

$$y_{1,3} = y_{1,2} + y'_{1,2} * h = y_{1,2} + y_{2,2} * h = 5.49619580835$$
$$y_{2,3} = y_{2,2} + y'_{2,2} * h = y_{2,2} + y_{3,2} * h = 12.1977644825$$
$$y_{3,3} = y_{3,2} + y'_{3,2} * h = y_{3,2} + y_{4,2} * h = 7.01602746421$$
$$y'_{4,2} = y_{5,2} = \frac{1}{3}\left(5 - 14.49 x_2^{1.1} - 1.8711 x_2^{-1.9} - y_{2,2} + 3 x_2 y_{3,2}\right) = -0.5316810895$$
$$y_{4,3} = y_{4,2} + y'_{4,2} * h = y_{4,2} + y_{5,2} * h = 0.61859551564$$
$$x_3 = x_2 + h = 1.086 + 0.043 = 1.129$$

$$y_{1,4} = y_{1,3} + y'_{1,3} * h = y_{1,3} + y_{2,3} * h = 6.0206996811$$
$$y_{2,4} = y_{2,3} + y'_{2,3} * h = y_{2,3} + y_{3,3} * h = 12.4994536635$$
$$y_{3,4} = y_{3,3} + y'_{3,3} * h = y_{3,3} + y_{4,3} * h = 7.04262707138$$
$$y'_{4,3} = y_{5,3} = \frac{1}{3}\left(5 - 14.49 x_3^{1.1} - 1.8711 x_3^{-1.9} - y_{2,3} + 3 x_3 y_{3,3}\right) = -0.4930835517$$
$$y_{4,4} = y_{4,3} + y'_{4,3} * h = y_{4,3} + y_{5,3} * h = 0.597392922917$$
$$x_4 = x_3 + h = 1.129 + 0.043 = 1.172$$

$$y_{1,5} = y_{1,4} + y'_{1,4} * h = y_{1,4} + y_{2,4} * h = 6.55817618863$$
$$y_{2,5} = y_{2,4} + y'_{2,4} * h = y_{2,4} + y_{3,4} * h = 12.8022866276$$
$$y_{3,5} = y_{3,4} + y'_{3,4} * h = y_{3,4} + y_{4,4} * h = 7.06831496707$$
$$y'_{4,4} = y_{5,4} = \frac{1}{3}\left(5 - 14.49 x_4^{1.1} - 1.8711 x_4^{-1.9} - y_{2,4} + 3 x_4 y_{3,4}\right) = -0.4585102651$$
$$y_{4,5} = y_{4,4} + y'_{4,4} * h = y_{4,4} + y_{5,4} * h = 0.577676981518$$
$$x_5 = x_4 + h = 1.172 + 0.043 = 1.215$$

$$y_{1,6} = y_{1,5} + y'_{1,5} * h = y_{1,5} + y_{2,5} * h = 7.10867451362$$
$$y_{2,6} = y_{2,5} + y'_{2,5} * h = y_{2,5} + y_{3,5} * h = 13.1062241712$$
$$y_{3,6} = y_{3,5} + y'_{3,5} * h = y_{3,5} + y_{4,5} * h = 7.09315307728$$
$$y'_{4,5} = y_{5,5} = \frac{1}{3}\left(5 - 14.49 x_5^{1.1} - 1.8711 x_5^{-1.9} - y_{2,5} + 3 x_5 y_{3,5}\right) = -0.427419263567$$
$$y_{4,6} = y_{4,5} + y'_{4,5} * h = y_{4,5} + y_{5,5} * h = 0.55929795318$$
$$x_6 = x_5 + h = 1.215 + 0.043 = 1.258$$

$$y_{1,7} = y_{1,6} + y'_{1,6} * h = y_{1,6} + y_{2,6} * h = 7.67224215298$$
$$y_{2,7} = y_{2,6} + y'_{2,6} * h = y_{2,6} + y_{3,6} * h = 13.4112298395$$
$$y_{3,7} = y_{3,6} + y'_{3,6} * h = y_{3,6} + y_{4,6} * h = 7.11720488927$$
$$y'_{4,6} = y_{5,6} = \frac{1}{3}\left(5 - 14.49 x_6^{1.1} - 1.8711 x_6^{-1.9} - y_{2,6} + 3 x_6 y_{3,6}\right) = -0.399357485667$$
$$y_{4,7} = y_{4,6} + y'_{4,6} * h = y_{4,6} + y_{5,6} * h = 0.542125581301$$
$$x_7 = x_6 + h = 1.258 + 0.043 = 1.301$$

$$y_{1,8} = y_{1,7} + y'_{1,7} * h = y_{1,7} + y_{2,7} * h = 8.24892503608$$
$$y_{2,8} = y_{2,7} + y'_{2,7} * h = y_{2,7} + y_{3,7} * h = 13.7172696497$$
$$y_{3,8} = y_{3,7} + y'_{3,7} * h = y_{3,7} + y_{4,7} * h = 7.14051628927$$
$$y'_{4,7} = y_{5,7} = \frac{1}{3}\left(5 - 14.49x_7^{1.1} - 1.8711x_7^{-1.9} - y_{2,7} + 3x_7 y_{3,7}\right) = -0.373973711133$$
$$y_{4,8} = y_{4,7} + y'_{4,7} * h = y_{4,7} + y_{5,7} * h = 0.526046001722$$
$$x_8 = x_7 + h = 1.301 + 0.043 = 1.344$$

$$y_{1,9} = y_{1,8} + y'_{1,8} * h = y_{1,8} + y_{2,8} * h = 8.83876763102$$
$$y_{2,9} = y_{2,8} + y'_{2,8} * h = y_{2,8} + y_{3,8} * h = 14.0243118501$$
$$y_{3,9} = y_{3,8} + y'_{3,8} * h = y_{3,8} + y_{4,8} * h = 7.16313626734$$
$$y'_{4,8} = y_{5,8} = \frac{1}{3}\left(5 - 14.49x_8^{1.1} - 1.8711x_8^{-1.9} - y_{2,8} + 3x_8 y_{3,8}\right) = -0.350855207967$$
$$y_{4,9} = y_{4,8} + y'_{4,8} * h = y_{4,8} + y_{5,8} * h = 0.510959227779$$
$$x_9 = x_8 + h = 1.344 + 0.043 = 1.387$$

$$y_{1,10} = y_{1,9} + y'_{1,9} * h = y_{1,9} + y_{2,9} * h = 9.44181304057$$
$$y_{2,10} = y_{2,9} + y'_{2,9} * h = y_{2,9} + y_{3,9} * h = 14.3323267096$$
$$y_{3,10} = y_{3,9} + y'_{3,9} * h = y_{3,9} + y_{4,9} * h = 7.18516751413$$
$$y'_{4,9} = y_{5,9} = \frac{1}{3}\left(5 - 14.49x_9^{1.1} - 1.8711x_9^{-1.9} - y_{2,9} + 3x_9 y_{3,9}\right) = -0.329817196333$$
$$y_{4,10} = y_{4,9} + y'_{4,9} * h = y_{4,9} + y_{5,9} * h = 0.496777088337$$
$$x_{10} = x_9 + h = 1.387 + 0.043 = 1.430$$

En consecuencia los valores de la variable dependiente y de las derivadas para x = 1.430 son:

$$y = y_1 = 9.44181304057$$
$$\frac{dy}{dx} = y' = y_2 = 14.3323267096$$
$$\frac{d^2y}{dx^2} = y'' = y_3 = 7.18516751413$$
$$\frac{d^3y}{dx^3} = y''' = y_4 = 0.496777088337$$

Como queda claro en el ejemplo, aún cuando se toma un número pequeño de segmentos (10), la cantidad de cálculos necesarios es muy elevado, razón por la cual estos métodos no pueden aplicarse en la práctica sino es con la ayuda de un programa de computadora.

12.2.2.2. Algoritmo

El algoritmo que automatiza el procedimiento se presenta en la Figura 11.3.

El algoritmo consiste básicamente en aplicar la ecuación (18) en un ciclo ***FOR*** que se repite el número de segmentos establecido.

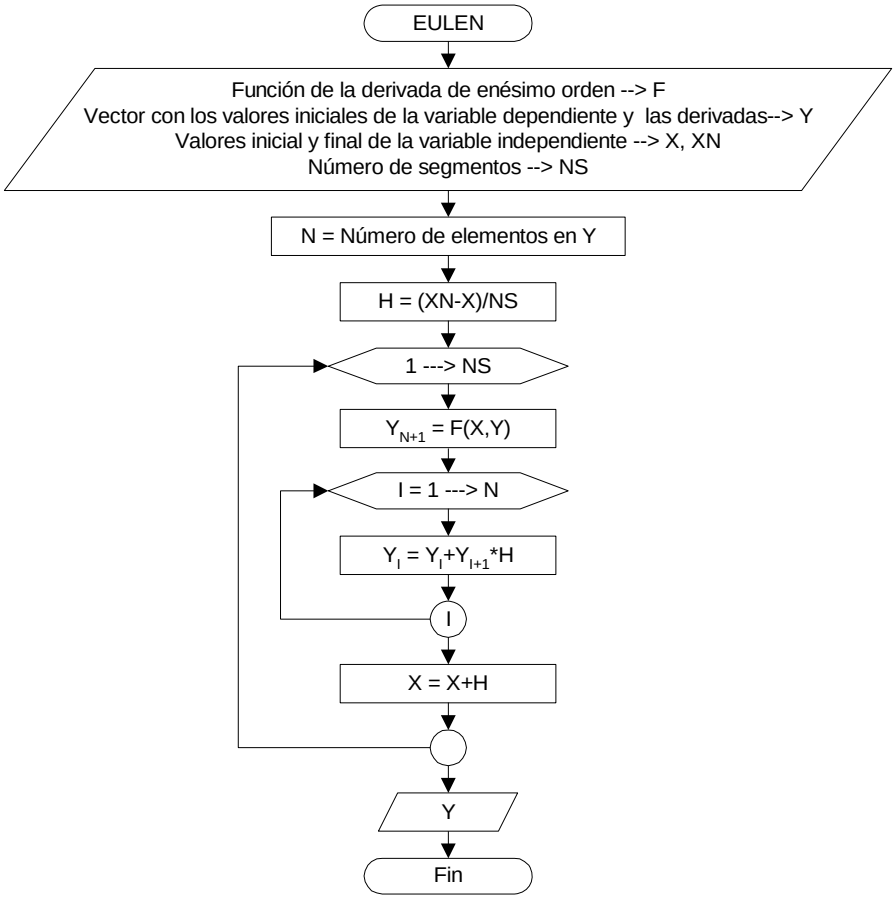


Figura 11.3. Algoritmo para resolver ecuaciones diferenciales de enésimo grado con el método de Euler

12.2.2.3. Programa

El programa elaborado siguiendo el algoritmo de la Figura 11.3, es el siguiente:

```
« 0 0 □ F Y X XN NS N H
  « Y Y SIZE DUP 1 GET 'N' STO
    1 ADD RDM
    XN X - NS / 'H' STO
    1 NS START
      N 1 + X 3 PICK F EVAL PUT
      1 N FOR I
        I OVER I GET 3 PICK I 1 + GET H
         * + PUT
      NEXT
      H 'X' STO+
    NEXT
  »
»
```

En lo sucesivo se asumirá que este programa ha sido guardado en la variable global ***'EULEN'***.

Para resolver el ejemplo manual empleando ***EULEN***, podemos escribir el siguiente programa:

```
«
```

```
  «  □  X Y
    « 5 14.49 X 1.1 ^ * - 1.8711 X 1.9 NEG ^ * - Y 2 GET -
      3 X * Y 3 GET * +  3 /  »
  »
  [4 11.3 6.93 0.693] 1 1.43 10
  EULEN "Y" □TAG
»
```

Haciendo corre el programa se obtiene el resultado:

```
Y: [9.44181304057 14.3323267096 7.18510751413 .496777088337 -.329817196333]
```

Que son los mismos valores ***(y, y', y'', y''', y'''')***, calculados en el ejemplo manual.

Al igual que sucede con las ecuaciones diferenciales de primer orden, cuando en la solución de un problema es necesario resolver varias veces una ecuación de enésimo orden, una mejor alternativa consiste en resolver la ecuación una sola vez, tomando el mayor valor posible de la variable independiente, guardar los resultados intermedios en vectores (o una matriz) y emplear luego estos vectores (o la matriz) conjuntamente un método de interpolación para calcular otros valores de la variable dependiente y/o sus derivadas.

12.2.3. Sistemas de ecuaciones diferenciales

Cuando en lugar de una ecuación diferencial se debe resolver un sistema de ecuaciones diferenciales y en dicho sistema existen ecuaciones de segundo o mayor orden, las mismas deben ser transformadas previamente en ecuaciones diferenciales de primer orden (siguiendo el procedimiento descrito en el anterior acápite).

De esta manera es siempre posible transformar un sistema de ecuaciones diferenciales de cualquier orden en un sistema de ecuaciones diferenciales de primer orden de la forma:

$$\begin{aligned} y'_1 &= f_1(x, y_1, y_2, y_3, \ldots, y_n) \\ y'_2 &= f_2(x, y_1, y_2, y_3, \ldots, y_n) \\ y'_3 &= f_3(x, y_1, y_2, y_3, \ldots, y_n) \\ &\ldots \\ y'_n &= f_n(x, y_1, y_2, y_3, \ldots, y_n) \end{aligned} \qquad (20)$$

En este sistema ***"x"*** es la variable independiente; y_1, y_2, y_3, …, y_n son las variables dependientes o las variables auxiliares empleadas al transformar ecuaciones diferenciales de orden superior en ecuaciones de primer orden; y'_1, y'_2, y'_3, …, y'_n son las derivadas de dichas variables y f_1, f_2, f_3, ..., f_n las funciones de dichas derivadas.

Aplicando la ecuación de ***Euler (13)*** a cada una de las ecuaciones del sistema, en un punto ***"j"*** cualquiera, se tiene:

$$\begin{aligned} y_{1,j+1} &= y_{1,j} + y'_{1,j} * h = y_{1,j} + f_1(x_j, y_{1,j}, y_{2,j}, y_{3,j}, \ldots, y_{n,j}) * h \\ y_{2,j+1} &= y_{2,j} + y'_{2,j} * h = y_{2,j} + f_2(x_j, y_{1,j}, y_{2,j}, y_{3,j}, \ldots, y_{n,j}) * h \\ y_{3,j+1} &= y_{3,j} + y'_{3,j} * h = y_{3,j} + f_3(x_j, y_{1,j}, y_{2,j}, y_{3,j}, \ldots, y_{n,j}) * h \\ &\ldots \\ y_{n,j+1} &= y_{n,j} + y'_{n,j} * h = y_{n,j} + f_n(x_j, y_{1,j}, y_{2,j}, y_{3,j}, \ldots, y_{n,j}) * h \\ x_{j+1} &= x_j + h \end{aligned} \qquad (21)$$

En estas ecuaciones, las funciones (f_1 a f_n) son en su mayoría funciones reales y no simplemente variables auxiliares, razón por la cual es necesario evaluarlas en cada punto.

Las ecuaciones (21) pueden ser escritas también como:

$$\begin{aligned} y_{i,j+1} &= y_{i,j} + y'_{i,j} * h = y_{i,j} + f_i(x_j, y_{1,j}, y_{2,j}, y_{3,j}, ..., y_{n,j}) * h \quad \{i = 1 \rightarrow n \\ x_{j+1} &= x_j + h \end{aligned} \tag{21.a}$$

Por supuesto estas ecuaciones pueden ser empleadas también para resolver una ecuación diferencial de enésimo orden (transformándola en un sistema de ecuaciones diferenciales de primer orden), sin embargo, es más eficiente en estos casos emplear el procedimiento descrito en el anterior acápite.

12.2.3.1. Ejemplo manual

Para comprender mejor la resolución de sistemas de ecuaciones diferenciales empleando el método de *Euler* calcularemos los valores de *u, v, w* para *t=0.35*, en base al siguiente sistema de ecuaciones diferenciales:

$$\begin{aligned} &u' - v + 2w + t^2 - 2t^{1.5} - 1.2t^{0.2} + 10 = 0 \\ &v' + 5w - 5t^{1.5} - 2t + 35 = 0 \\ &w' - 2u + v - t^2 + 2t^{1.2} - 1.5t^{0.5} + 10 = 0 \\ &\begin{cases} u = 3,\ v = -4,\ w = -7 \ para\ t = 0 \\ u = ?,\ v = ?,\ w = ?\ para\ t = 0.35 \end{cases} \end{aligned}$$

Donde *u, v* y *w* son las variables dependientes y *t* la variable independiente. Las soluciones analíticas de este sistema son: u = $t^{1.2}+3$; $v = t^2-4$ y w = $t^{1.5}-7$.

Primero efectuamos un cambio de variables y reordenamos las ecuaciones de manera que tengan la forma de la ecuación (20):

$$\begin{aligned} y'_1 &= f_1(x, y_1, y_2, y_3) = y_2 - 2y_3 - x^2 + 2x^{1.5} + 1.2x^{0.2} - 10 \\ y'_2 &= f_2(x, y_1, y_2, y_3) = 5x^{1.5} + 2x - 5y_3 - 35 \\ y'_3 &= f_3(x, y_1, y_2, y_3) = 2y_1 - y_2 + x^2 - 2x^{1.2} + 1.5x^{0.5} - 10 \\ &\begin{cases} y_1 = 3,\ y_2 = -4,\ y_3 = -7 \ para\ x = 0 \\ y_1 = ?,\ y_2 = ?,\ y_3 = ?\ para\ x = 0.35 \end{cases} \end{aligned}$$

Con estas nuevas variables las soluciones analíticas son: $y_1 = x^{1.2}+3$; $y_2 = x^2-4$ y $y_3 = x^{1.5}-7$. Para este ejemplo fijaremos el número de segmentos en 10, entonces el incremento *(h)* a emplear es:

$$h = \frac{0.35 - 0}{10} = 0.035$$

Comenzando con los valores iniciales conocidos (valores del punto cero) calculamos los valores del primer punto empleando las ecuaciones (21):

$$\begin{aligned} y_{1,1} &= y_{1,0} + f_1(x_0, y_{1,0}, y_{2,0}, y_{3,0})h = y_{1,0} + \left(y_{2,0} - 2y_{3,0} - x_0^{\,2} + 2x_0^{\,1.5} + 1.2x_0^{\,0.2} - 10\right)h \\ &= 3 + \left((-4) - 2(-7) - 0^2 + 2(0)^{1.5} + 1.2(0)^{0.2} - 10\right)0.035 = 3 \\ y_{2,1} &= y_{2,0} + f_2(x_0, y_{1,0}, y_{2,0}, y_{3,0}) = y_{2,0} + \left(5x_0^{\,1.5} + 2x_0 - 5y_{3,0} - 35\right)h \\ &= -4 + \left(5(0)^{1.5} + 2(0) - 5(-7) - 35\right)0.035 = -4 \end{aligned}$$

$$y_{3,1} = y_{3,0} + f_3(x_0, y_{1,0}, y_{2,0}, y_{3,0}) = y_{3,0} + \left(2y_{1,0} - y_{2,0} + x_0^{\,2} - 2x_0^{\,1.2} + 1.5x_0^{\,0.5} - 10\right)h$$
$$= -7 + \left(2(3) - (-4) + (0)^2 - 2(0)^{1.2} + 1.5(0)^{0.5} - 10\right)0.035 = -7$$
$$x_1 = x_0 + h = 0 + 0.035 = 0.035$$

Con los valores del punto 1 calculamos ahora los valores del siguiente punto:

$$y_{1,2} = y_{1,1} + f_1(x_1, y_{1,1}, y_{2,1}, y_{3,1})h = y_{1,1} + \left(y_{2,1} - 2y_{3,1} - x_1^{\,2} + 2x_1^{\,1.5} + 1.2x_1^{\,0.2} - 10\right)h$$
$$= 3 + ((-4) - 2(-7) - 0.035^2 + 2(0.035)^{1.5} + 1.2(0.035)^{0.2} - 10)0.035 = 3.02189689372$$
$$y_{2,2} = y_{2,1} + f_2(x_1, y_{1,1}, y_{2,1}, y_{3,1})h = y_{2,1} + \left(5x_1^{\,1.5} + 2x_1 - 5y_{3,1} - 35\right)h$$
$$= -4 + \left(5(0.035)^{1.5} + 2(0.035) - 5(-7) - 35\right)0.035 = -3.99640411743$$
$$y_{3,2} = y_{3,1} + f_3(x_1, y_{1,1}, y_{2,1}, y_{3,1})h = y_{3,1} + \left(2y_{1,1} - y_{2,1} + x_1^{\,2} - 2x_1^{\,1.2} + 1.5x_1^{\,0.5} - 10\right)h$$
$$= -7 + \left(2(3) - (-4) + (0.035)^2 - 2(0.035)^{1.2} + 1.5(0.035)^{0.5} - 10\right)0.035 = -6.99138835694$$
$$x_2 = x_1 + h = 0.035 + 0.035 = 0.070$$

Entonces con los valores del punto 2 calculamos los del punto 3 y así sucesivamente obteniéndose así los siguientes resultados:

$$y_{1,3} = y_{1,2} + \left(y_{2,2} - 2y_{3,2} - x_2^{\,2} + 2x_2^{\,1.5} + 1.2x_2^{\,0.2} - 10\right)h = 3.0472205196$$
$$y_{2,3} = y_{2,2} + \left(5x_2^{\,1.5} + 2x_2 - 5y_{3,2} - 35\right)h = -3.8977010961$$
$$y_{3,3} = y_{3,2} + \left(2y_{1,2} - y_{2,2} + x_2^{\,2} - 2x_2^{\,1.2} + 1.5x_2^{\,0.5} - 10\right)h = -6.97879856369$$
$$x_3 = x_2 + h = 0.105$$

$$y_{1,4} = y_{1,3} + \left(y_{2,3} - 2y_{3,3} - x_3^{\,2} + 2x_3^{\,1.5} + 1.2x_3^{\,0.2} - 10\right)h = 3.07485032658$$
$$y_{2,4} = y_{2,3} + \left(5x_3^{\,1.5} + 2x_3 - 5y_{3,3} - 35\right)h = -3.98017618045$$
$$y_{3,4} = y_{3,3} + \left(2y_{1,3} - y_{2,3} + x_3^{\,2} - 2x_3^{\,1.2} + 1.5x_3^{\,0.5} - 10\right)h = -6.96313636538$$
$$x_4 = x_3 + h = 0.140$$

$$y_{1,5} = y_{1,4} + \left(y_{2,4} - 2y_{3,4} - x_4^{\,2} + 2x_4^{\,1.5} + 1.2x_4^{\,0.2} - 10\right)h = 3.10428942802$$
$$y_{2,5} = y_{2,4} + \left(5x_4^{\,1.5} + 2x_4 - 5y_{3,4} - 35\right)h = -3.96766025591$$
$$y_{3,5} = y_{3,4} + \left(2y_{1,4} - y_{2,4} + x_4^{\,2} - 2x_4^{\,1.2} + 1.5x_4^{\,0.5} - 10\right)h = -6.94487478444$$
$$x_5 = x_4 + h = 0.175$$

$$y_{1,6} = y_{1,5} + \left(y_{2,5} - 2y_{3,5} - x_5^{\,2} + 2x_5^{\,1.5} + 1.2x_5^{\,0.2} - 10\right)h = 3.13525376803$$
$$y_{2,6} = y_{2,5} + \left(5x_5^{\,1.5} + 2x_5 - 5y_{3,5} - 35\right)h = -3.95224581198$$
$$y_{3,6} = y_{3,5} + \left(2y_{1,5} - y_{2,5} + x_5^{\,2} - 2x_5^{\,1.2} + 1.5x_5^{\,0.5} - 10\right)h = -6.92431679085$$
$$x_6 = x_5 + h = 0.210$$

$$y_{1,7} = y_{1,6} + \left(y_{2,6} - 2y_{3,6} - x_6^{\,2} + 2x_6^{\,1.5} + 1.2x_6^{\,0.2} - 10\right)h = 3.16755946791$$
$$y_{2,7} = y_{2,6} + \left(5x_6^{\,1.5} + 2x_6 - 5y_{3,6} - 35\right)h = -3.9339494079$$
$$y_{3,7} = y_{3,6} + \left(2y_{1,6} - y_{2,6} + x_6^{\,2} - 2x_6^{\,1.2} + 1.5x_6^{\,0.5} - 10\right)h = -6.90167713586$$
$$x_7 = x_6 + h = 0.245$$

$$y_{1,8} = y_{1,7} + \left(y_{2,7} - 2y_{3,7} - x_7^{\,2} + 2x_7^{\,1.5} + 1.2x_7^{\,0.2} - 10\right)h = 3.2010782768$$
$$y_{2,8} = y_{2,7} + \left(5x_7^{\,1.5} + 2x_7 - 5y_{3,7} - 35\right)h = -3.91278386685$$
$$y_{3,8} = y_{3,7} + \left(2y_{1,7} - y_{2,7} + x_7^{\,2} - 2x_7^{\,1.2} + 1.5x_7^{\,0.5} - 10\right)h = -6.87711755413$$
$$x_8 = x_7 + h = 0.280$$

$$y_{1,9} = y_{1,8} + \left(y_{2,8} - 2y_{3,8} - x_8^{\,2} + 2x_8^{\,1.5} + 1.2x_8^{\,0.2} - 10\right)h = 3.23571615302$$
$$y_{2,9} = y_{2,8} + \left(5x_8^{\,1.5} + 2x_8 - 5y_{3,8} - 35\right)h = -3.88875993203$$
$$y_{3,9} = y_{3,8} + \left(2y_{1,8} - y_{2,8} + x_8^{\,2} - 2x_8^{\,1.2} + 1.5x_8^{\,0.5} - 10\right)h = -6.85076479488$$
$$x_9 = x_8 + h = 0.315$$

$$y_{1,10} = y_{1,9} + \left(y_{2,9} - 2y_{3,9} - x_9^{\,2} + 2x_9^{\,1.5} + 1.2x_9^{\,0.2} - 10\right)h = 3.27146158771$$
$$y_{2,10} = y_{2,9} + \left(5x_9^{\,1.5} + 2x_9 - 5y_{3,9} - 35\right)h = -3.86188726341$$
$$y_{3,10} = y_{3,9} + \left(2y_{1,9} - y_{2,9} + x_9^{\,2} - 2x_9^{\,1.2} + 1.5x_9^{\,0.5} - 10\right)h = -6.82272095555$$
$$x_{10} = x_9 + h = 0.350$$

Por lo tanto los valores de las variables independientes para ***t = 0.35*** son: ***u = 3.27146158771, v = -3.86188726341, w = -6.82272095555***, siendo los resultados exactos: ***u = 3.28371457908, v = -3.8775, w = -6.79293720759***, por lo tanto los errores relativos cometidos son: 0.37% para ***u***, 0.40% para ***v*** y 0.44% para ***w***.

12.2.3.2. Algoritmo

El algoritmo que automatiza el procedimiento seguido en el ejemplo manual se presenta en la Figura 11.4.

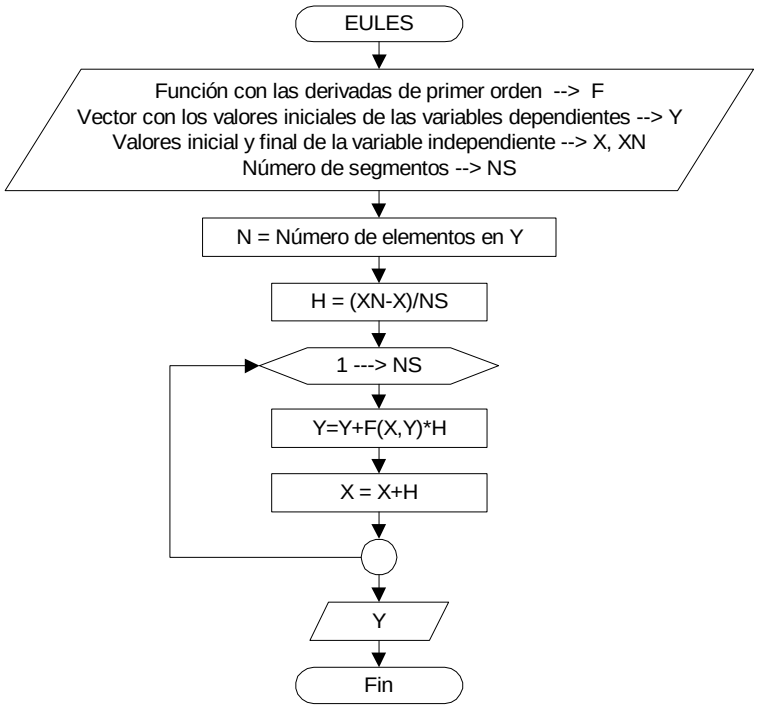


Figura 11.4. Algoritmo para resolver sistemas de ecuaciones diferenciales con el método de Euler.

12.2.3.3. Programa

El programa elaborado siguiendo el algoritmo de la Figura 11.4 es el siguiente:

```
« 0 0 □ F Y X XN NS N H
 « Y SIZE 1 GET 'N' STO
   XN X - NS / 'H' STO
   1 NS START
     Y X Y V□ F EVAL N □ARRY H * + 'Y' STO
     H 'X' STO+
   NEXT
   Y
 »
»
```

En lo futuro se asumirá que este programa ha sido guardado en la variable ***"EULES"***. El programa recibe como datos: a) una función con las ecuaciones correspondientes a las derivadas primeras de las variables del sistema; b) el vector con los valores iniciales conocidos (y_1 a y_n); c) los valores inicial y final la variable independiente (x_0 y x_n) y d) el número de segmentos. Devuelve como resultado un vector con los valores de las variables dependientes correspondientes a x_n.

La función que se manda al programa debe recibir la variable independiente y los valores conocidos de las variables dependientes (valores del punto anterior). Debe devolver los valores de las derivadas primeras. Así para resolver el ejemplo manual con ***EULES***, escribimos el siguiente programa:

```
« « □ x y1 y2 y3
    « y2 2 y3 * - x SQ - 2 x 1.5 ^ * + 1.2 x .2 ^ * + 10 -
      5 x 1.5 ^ * 2 x * + 5 y3 * - 35 -
      2 y1 * y2 - x SQ + 2 x 1.2 ^ * - 1.5 x √ * + 10 -
    »
  » [ 3 -4 -7 ] 0 .35 10
  EULES
  "y" □TAG
»
```

Haciendo correr el programa se obtiene:

```
y: [ 3.27140158771 -3.86188726341 -6.82272095555 ]
```

Que son los mismos valores calculados en el ejemplo manual. Ahora que contamos con un programa podemos mejorar la exactitud incrementando el número de segmentos. Así si en lugar de 10 segmentos empleamos 30:

```
« « → x y1 y2 y3
    « y2 2 y3 * - x SQ - 2 x 1.5 ^ * + 1.2 x .2 ^ * + 10 -
      5 x 1.5 ^ * 2 x * + 5 y3 * - 35 -
      2 y1 * y2 - x SQ + 2 x 1.2 ^ * - 1.5 x √ * + 10 -
    »
  » [ 3 -4 -7 ] 0 .35 30
  EULES
  "y" →TAG
»
```

Obtenemos los resultados:

```
y: [ 3.28186384245 -3.87414367781 -6.79860476109 ]
```

Cuyos errores relativos son 0.056%, 0.087% y 0.083% respectivamente.

Como en todos los otros casos, cuando en la resolución de un problema es necesario resolver varias veces un mismo sistema de ecuaciones diferenciales, es más eficiente resolver el sistema una sola vez, tomando para esta solución el mayor valor posible de la variable independiente y almacenando los resultados intermedios en vectores o en una matriz. Entonces, para otros valores de la variable independiente, se calculan los valores de las variables dependientes con los vectores (o matriz) y un método de interpolación.

12.2.4. Preguntas y ejercicios

1. ¿Cuándo se dice que una ecuación diferencial ha sido resuelta?
2. ¿Por qué en una ecuación diferencial no es posible calcular directamente el valor de la variable dependiente?
3. ¿Cuál es la solución cuando se resuelve analíticamente una ecuación diferencial?
4. ¿Es posible resolver analíticamente cualquier ecuación diferencial?
5. Cuándo se generan ecuaciones diferenciales a partir de una función conocida y=f(x) ¿Cuál es la solución de las ecuaciones generadas?
6. ¿Cuándo se tiene un problema del valor inicial?
7. ¿Cuándo se tiene un problema del valor límite?
8. ¿Por qué todos los métodos numéricos sólo permiten resolver ecuaciones diferenciales de primer orden?
9. Si se conoce la forma de la curva y^i versus $f(x,y)$ ¿Cómo se calcula el valor de y_n?
10. ¿Cómo se asume que es la forma de la curva en el método de Euler?
11. ¿En qué cosiste el método de Euler modificado?
12. Si al aplicar el método de Euler el número de segmentos es 15, ¿Cuántas veces deberá repetirse el procedimiento para obtener el valor de la variable dependiente?
13. ¿El número de segmentos que se emplea en el método de Euler debe incrementar o disminuir a medida que incrementa la distancia entre el valor inicial y final?
14. ¿Qué procedimiento resulta más eficiente cuando es necesario resolver varias veces una misma ecuación diferencial?
15. ¿Qué se debe hacer para resolver una ecuación diferencial de enésimo orden con el método de Euler?
16. Para resolver una ecuación diferencial de séptimo orden, ¿Cuántas variables auxiliares deben emplearse?
17. ¿Qué se debe hacer para resolver un sistema de ecuaciones diferenciales de cualquier orden con el método de Euler?
18. ¿Cuál sería la función que se emplearía con el programa EULES para resolver la ecuación de cuarto grado del ejemplo manual?
19. Elabore un programa que empleando el método de Euler, resuelva la siguiente ecuación diferencial de primer orden:

$$y' = y\left(x^2 - 1\right) \quad \begin{cases} y = 1 \; para \; x = 0 \\ y = ? \; para \; x = 2 \end{cases}$$

20. Elabore un programa que empleando el método de Euler, resuelva la siguiente ecuación diferencial de primer orden:

$$y' = 4e^{0.8x} - 0.5y \quad \begin{cases} y = 2 \; para \; x = 0 \\ y = ? \; para \; x = 2 \end{cases}$$

21. Elabore un programa que empleando el método de Euler, resuelva la siguiente ecuación diferencial de primer orden:

$$y' = \frac{5x}{y} - xy \quad \begin{cases} y = 2 \; para \; x = 0 \\ y = ? \; para \; x = 0.5 \end{cases}$$

22. Elabore un programa que empleando el método de Euler, resuelva la siguiente ecuación diferencial de primer orden:

$$y' = x^3 + y^2 \quad \begin{cases} y = 0 \; para \; x = 0 \\ y = ? \; para \; x = 1.4 \end{cases}$$

23. Elabore un programa que empleando el método de Euler, resuelva la siguiente ecuación diferencial de primer orden:

$$y' = y^2 + t^2 \quad \begin{cases} y = 2 \; para \; t = 1 \\ y = ? \; para \; t = 2 \end{cases}$$

24. Elabore un programa que empleando el método de Euler, resuelva la siguiente ecuación diferencial de segundo orden:

$$\frac{1}{2}q'' + 6q' + 50q = 24sen(10t) \quad \begin{cases} q = 0; q' = 0 \; para \; t = 0 \\ q = ?; q' = ? \; para \; t = 2.1 \end{cases}$$

25. Elabore un programa que empleando el método de Euler, resuelva la siguiente ecuación diferencial de segundo orden:

$$x'' + 64x = 16cos(8t) \quad \begin{cases} x = 0; x' = 0 \; para \; t = 0 \\ x = ?; x' = ? \; para \; t = 0.8 \end{cases}$$

26. Elabore un programa que empleando el método de Euler, resuelva la siguiente ecuación diferencial de tercer orden:

$$y''' + ty'' - ty' - 2y = t \quad \begin{cases} y = 0; y' = 0; y'' = 0 \; para \; t = 0 \\ y = ?; y' = ?; y'' = ? \; para \; t = 0.2, 0.4, 0.8 \end{cases}$$

27. Elabore un programa que empleando el método de Euler, resuelva el siguiente sistema de ecuaciones diferenciales:

$$\frac{dx}{dt} = x' = 2x + 3y$$

$$\frac{dy}{dt} = y' = 2x + y$$

$$\begin{cases} x = 2.7; y = 2 \; para \; t = 0 \\ x = ?; y = ? \; para \; t = 1.3 \end{cases}$$

28. Elabore un programa que empleando el programa EULES, resuelva el ejercicio 25.

29. Elabore un programa que empleando el programa EULES, resuelva el ejercicio 26.

12.3. Métodos de Runge – Kutta

Los métodos de ***Runge-Kutta***, son aquellos que tienen la forma general:

$$y_{i+1} = y_i + a_1 k_1 + a_2 k_2 + a_3 k_3 + ... + a_n k_n \tag{22}$$

Donde los valores de k tienen la forma:

$$\begin{aligned}
k_1 &= hf(x_i, y_i) \\
k_2 &= hf(x_i + p_j h, y_i + q_{1,1} k_1) \\
k_3 &= hf(x_i + p_2 h, y_i + q_{2,1} k_1 + q_{2,2} k_2) \\
k_4 &= hf(x_i + p_3 h, y_i + q_{3,1} k_1 + q_{3,2} k_2 + q_{3,3} k_3) \\
&... \\
k_n &= hf(x_i + p_{n-1} h, y_i + q_{n-1,1} k_1 + q_{n-1,2} k_2 + ... + q_{n-1,n-1} k_{n-1})
\end{aligned} \tag{23}$$

Donde $f(x,y)$ es la función correspondiente a la derivada primera de la variable dependiente, x_i, y_i son los valores conocidos de las variables independiente y dependiente para un punto dado i; y_{i+1} es el valor desconocido de la variable dependiente en el siguiente punto y h es el incremento de la variable independiente. Para todas las formas, el valor de la variable independiente del siguiente punto se calcula como en el método de ***Euler***:

$$x_{i+1} = x_i + h \tag{24}$$

En base a la ecuación (22), estableciendo el número de evaluaciones funcionales (valores de k) y calculando los parámetros *"a", "p"* y *"q"*, se puede deducir un número prácticamente infinito de ecuaciones de ***Runke-Kutta***.

No todas las formas de Runge – Kutta predicen correctamente los valores de la variable dependiente, razón por la cual una vez deducidas deben ser puestas para determinar su confiabilidad.

Muchos investigadores han deducido y puesto a prueba varias formas de ***Runge-Kutta***. Estas ecuaciones se clasifican generalmente como métodos ***(m,n)***, donde *"m"* es el número de parámetros *"a_i"* y *"n"* es el número de valores *"k"* que deben ser calculados.

Algunas de las ecuaciones de las formas de ***Runke-Kutta*** que más se utilizan en la práctica son las siguientes:

La ecuación (2,2) de ***Runge-Kutta*** con una exactitud comparable con la del método modificado de ***Euler***:

$$\begin{aligned}
y_{i+1} &= y_i + \frac{1}{2}(k_1 + k_2) \\
k_1 &= hf(x_i, y_i) \\
k_2 &= hf(x_i + h, y_i + k_1)
\end{aligned} \tag{25}$$

La ecuación (3,3) de ***Runge-Kutta*** con un error de orden h^4:

$$\begin{aligned}
y_{i+1} &= y_i + \frac{1}{6}(k_1 + 4k_2 + k_3) \\
k_1 &= hf(x_i, y_i) \\
k_2 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \\
k_3 &= hf(x_i + h, y_i - k_1 + 2k_2)
\end{aligned} \tag{26}$$

La ecuación (4,4) de ***Runge-Kutta,*** conocida como la forma ***clásica***:

$$
\begin{aligned}
y_{i+1} &= y_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\
k_1 &= hf\left(x_i, y_i\right) \\
k_2 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \\
k_3 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right) \\
k_4 &= hf\left(x_i + h, y_i + k_3\right)
\end{aligned}
\tag{27}
$$

La ecuación (4,4) de ***Runge-Kutta*** conocida como la forma de ***Gil,*** la cual minimiza los errores debido a redondeos:

$$
\begin{aligned}
y_{i+1} &= y_i + \frac{1}{6}\left(k_1 + \left(2-\sqrt{2}\right)k_2 + \left(2+\sqrt{2}\right)k_3 + k_4\right) \\
k_1 &= hf\left(x_i, y_i\right) \\
k_2 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \\
k_3 &= hf\left(x_i + \frac{h}{2}, y_i + \left(\sqrt{2}-1\right)\frac{k_1}{2} + \left(2-\sqrt{2}\right)\frac{k_2}{2}\right) \\
k_4 &= hf\left(x_i + h, y_i - \sqrt{2}\frac{k_2}{2} + \left(1+\frac{\sqrt{2}}{2}\right)k_2\right)
\end{aligned}
\tag{28}
$$

Y una ecuación (5,6) conocida como la forma de ***Butcher,*** que suele ser empleada cuando se requiere gran exactitud en los cálculos, es la siguiente:

$$
\begin{aligned}
y_{i+1} &= y_i + \frac{1}{90}\left(7k_1 + 32k_3 + 12k_4 + 35k_5 + 7k_6\right) \\
k_1 &= hf\left(x_i, y_i\right) \\
k_2 &= hf\left(x_i + \frac{h}{4}, y_i + \frac{k_1}{4}\right) \\
k_3 &= hf\left(x_i + \frac{h}{4}, y_i + \frac{k_1}{8} + \frac{k_2}{8}\right) \\
k_4 &= hf\left(x_i + \frac{h}{2}, y_i - \frac{k_2}{2} + k_3\right) \\
k_5 &= hf\left(x_i + \frac{3}{4}h, y_i + \frac{3}{16}k_1 + \frac{9}{16}k_4\right) \\
k_6 &= hf\left(x_i + h, y_i - \frac{3}{7}k_1 + \frac{2}{7}k_2 + \frac{12}{7}k_3 - \frac{12}{7}k_4 + \frac{8}{7}k_5\right)
\end{aligned}
\tag{29}
$$

En este capítulo emplearemos la forma clásica (ecuación 27), por ser la forma que más se emplea en la práctica. Por lo tanto cuando hagamos referencia al método de ***Runge Kutta*** estaremos haciendo referencia a esta ecuación.

12.3.1. Resolución de una ecuación diferencial de primer orden

Todas las ecuaciones de ***Runge Kutta*** permiten resolver sólo ecuaciones diferenciales de primer orden y el procedimiento de cálculo es esencialmente el mismo que en el método de ***Euler,*** sólo que en lugar de calcular el valor

de la variable dependiente con la ecuación 13 ($y_{i+1} = y_i + y'_i\, h$) se emplea la ecuación 27.

12.3.1.1. Ejemplo manual

Para comprender mejor el procedimiento de cálculo resolveremos manualmente la ecuación diferencial de primer orden:

$$3y' + 4xy - 8x^3 + 4x = 0 \quad \begin{cases} y = 4 \text{ para } x = 2 \\ y = ? \text{ para } x = 6 \end{cases}$$

Fijaremos en 10 el número de segmentos, por lo tanto el valor de h es:

$$h = \frac{6-2}{10} = 0.4$$

La función $f(x,y)$, correspondiente a la derivada primera de la variable dependiente, obtenida despejando y' de la ecuación diferencial es:

$$y' = f(x,y) = \frac{8x^3 - 4x - 4xy}{3}$$

Ahora aplicamos la ecuación 27 comenzando con el primer punto conocido $(x_0=2,\ y_0=4)$:

$$k_1 = hf(x_0, y_0) = hf(2,4) = 0.4\left(\frac{8(2)^2 - 4(2) - 4(2)(4)}{3}\right) = 3.2$$

$$k_2 = hf\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right) = 0.4\, f\left(2 + \frac{0.4}{2}, 4 + \frac{3.2}{2}\right) = 0.4\, f(2.2, 5.6)$$

$$= 0.4\left(\frac{8(2.2)^2 - 4(2.2) - 4(2.2)(5.6)}{3}\right) = 3.61386666667$$

$$k_3 = hf\left(x_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}\right) = 0.4\, f\left(2 + \frac{0.4}{2}, 4 + \frac{3.61386666667}{2}\right) = 0.4\, f(2.2, 5.80693333334)$$

$$= 0.4\left(\frac{8(2.2)^2 - 4(2.2) - 4(2.2)(5.80693333334)}{3}\right) = 3.37106488888$$

$$k_4 = hf(x_0 + h, y_0 + k_3) = hf(2 + 0.4, 4 + 3.37106488888) = hf(2.4, 7.37106488888)$$

$$= 0.4\left(\frac{8(2.4)^2 - 4(2.4) - 4(2.4)(7.37106488888)}{3}\right) = 4.03063694224$$

$$y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$= 4 + \frac{1}{6}(3.2 + 2(3.61386666667) + 2(3.37106488888) + 4.03063694224) = 7.53341667555$$

El valor de x en el siguiente punto (x_1) es:

$$x_1 = x_0 + h = 2 + 0.4 = 2.4$$

Ahora que conocemos los valores de "x" y "y" para el punto 1, aplicamos la ecuación 27 para este punto:

$$k_1 = hf(x_1, y_1) = hf(2.4, 7.53341667555) = 3.82282665529$$

$$k_2 = hf\left(x_1+\frac{h}{2}, y_1+\frac{k_1}{2}\right) = 0.4\,f\left(2.4+\frac{0.4}{2}, 7.53341667555+\frac{3.8228665529}{2}\right)$$
$$= 0.4\,f\left(2.6, 9.44483000319\right) = 4.26423572892$$
$$k_3 = hf\left(x_1+\frac{h}{2}, y_1+\frac{k_2}{2}\right) = 0.4\,f\left(2.4+\frac{0.4}{2}, 7.53341667555+\frac{4.26423572892}{2}\right) = 0.4\,f\left(2.2, 5.80693333334\right)$$
$$= 0.4\,f\left(2.6, 9.66553454001\right) = 3.95819210453$$
$$k_4 = hf\left(x_1+h, y_1+k_3\right) = hf\left(2.4+0.4, 7.53341667555+3.95819210453\right)$$
$$= hf\,(2.8, 11.4916087801) = 4.76133088840$$
$$y_2 = y_1+\frac{1}{6}\left(k_1+2k_2+2k_3+k_4\right)$$
$$= 7.53341667555+\frac{1}{6}\left(3.82282665529+2\left(4.26423572892\right)+2\left(3.95819210453\right)+4.76133088840\right)$$
$$= 11.7049188773$$
$$x_2 = x_1+h = 2.4+0.4 = 2.8$$

Ahora aplicamos la ecuación 27 con los valores del punto 2 calculados y continuamos así sucesivamente hasta el décimo punto obteniéndose los siguientes resultados para los otros puntos:

$$k_1 = hf\left(x_2, y_2\right) = 4.44278780988$$
$$k_2 = hf\left(x_2+\frac{h}{2}, y_2+\frac{k_1}{2}\right) = 4.91789954852$$
$$k_3 = hf\left(x_2+\frac{h}{2}, y_2+\frac{k_2}{2}\right) = 4.53781015748$$
$$k_4 = hf\left(x_2+h, y_2+k_3\right) = 5.52494244732$$
$$y_3 = y_2+\frac{1}{6}\left(k_1+2k_2+2k_3+k_4\right) = 16.5181104888$$
$$x_3 = x_2+h = 3.2$$

$$k_1 = hf\left(x_3, y_3\right) = 5.05495809908$$
$$k_2 = hf\left(x_3+\frac{h}{2}, y_3+\frac{k_1}{2}\right) = 5.57493097052$$
$$k_3 = hf\left(x_2+\frac{h}{2}, y_2+\frac{k_2}{2}\right) = 5.10348890028$$
$$k_4 = hf\left(x_3+h, y_3+k_3\right) = 6.33291729200$$
$$y_4 = y_3+\frac{1}{6}\left(k_1+2k_2+2k_3+k_4\right) = 21.9755649911$$
$$x_4 = x_3+h = 3.6$$

$$k_1 = hf\left(x_4, y_4\right) = 5.65331521708$$
$$k_2 = hf\left(x_4+\frac{h}{2}, y_4+\frac{k_1}{2}\right) = 6.23762886480$$
$$k_3 = hf\left(x_4+\frac{h}{2}, y_4+\frac{k_2}{2}\right) = 5.64552436840$$

$$k_4 = hf\left(x_4 + h, y_4 + k_3\right) = 7.20834269972$$

$$y_5 = y_4 + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) = 28.0802257216$$

$$x_5 = x_4 + h = 4.0$$

$$k_1 = hf\left(x_5, y_5\right) = 6.22865179388$$

$$k_2 = hf\left(x_5 + \frac{h}{2}, y_5 + \frac{k_1}{2}\right) = 6.91118037452$$

$$k_3 = hf\left(x_5 + \frac{h}{2}, y_5 + \frac{k_2}{2}\right) = 6.14697236400$$

$$k_4 = hf\left(x_5 + h, y_5 + k_3\right) = 8.19644182572$$

$$y_6 = y_5 + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) = 34.8371589044$$

$$x_6 = x_5 + h = 4.4$$

$$k_1 = hf\left(x_6, y_6\right) = 6.76506710440$$

$$k_2 = hf\left(x_6 + \frac{h}{2}, y_6 + \frac{k_1}{2}\right) = 7.60608783988$$

$$k_3 = hf\left(x_6 + \frac{h}{2}, y_6 + \frac{k_2}{2}\right) = 6.57443573772$$

$$k_4 = hf\left(x_6 + h, y_6 + k_3\right) = 9.39111771628$$

$$y_7 = y_6 + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) = 42.2566975670$$

$$x_7 = x_6 + h = 4.8$$

$$k_1 = hf\left(x_7, y_7\right) = 7.22765422852$$

$$k_2 = hf\left(x_7 + \frac{h}{2}, y_7 + \frac{k_1}{2}\right) = 8.34526751652$$

$$k_3 = hf\left(x_7 + \frac{h}{2}, y_7 + \frac{k_2}{2}\right) = 6.85516465880$$

$$k_4 = hf\left(x_7 + h, y_7 + k_3\right) = 11.0051024160$$

$$y_8 = y_7 + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) = 50.3622850019$$

$$x_8 = x_7 + h = 5.2$$

$$k_1 = hf\left(x_8, y_8\right) = 7.53712959468$$

$$k_2 = hf\left(x_8 + \frac{h}{2}, y_8 + \frac{k_1}{2}\right) = 9.18475257868$$

$$k_3 = hf\left(x_8 + \frac{h}{2}, y_8 + \frac{k_2}{2}\right) = 6.81217548132$$

$$k_4 = hf\left(x_8 + h, y_8 + k_3\right) = 13.5760113573$$

$$y_9 = y_8 + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) = 59.2134511806$$

$$x_9 = x_8 + h = 5.6$$

$$k_1 = hf\left(x_9, y_9\right) = 7.48622580068$$
$$k_2 = hf\left(x_9 + \frac{h}{2}, y_9 + \frac{k_1}{2}\right) = 10.28049510000$$
$$k_3 = hf\left(x_9 + \frac{h}{2}, y_9 + \frac{k_2}{2}\right) = 5.95869192608$$
$$k_4 = hf\left(x_9 + h, y_9 + k_3\right) = 18.6491420560$$
$$y_{10} = y_9 + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) = 68.9824081666$$
$$x_{10} = x_9 + h = 6.0$$

Por lo tanto el valor de *y* para *x=6* es 68.9824081666, resultado que tiene un error relativo del 1.44%.

12.3.1.2. Algoritmo

El algoritmo que automatiza el procedimiento de cálculo se presenta en la Figura 11.5.

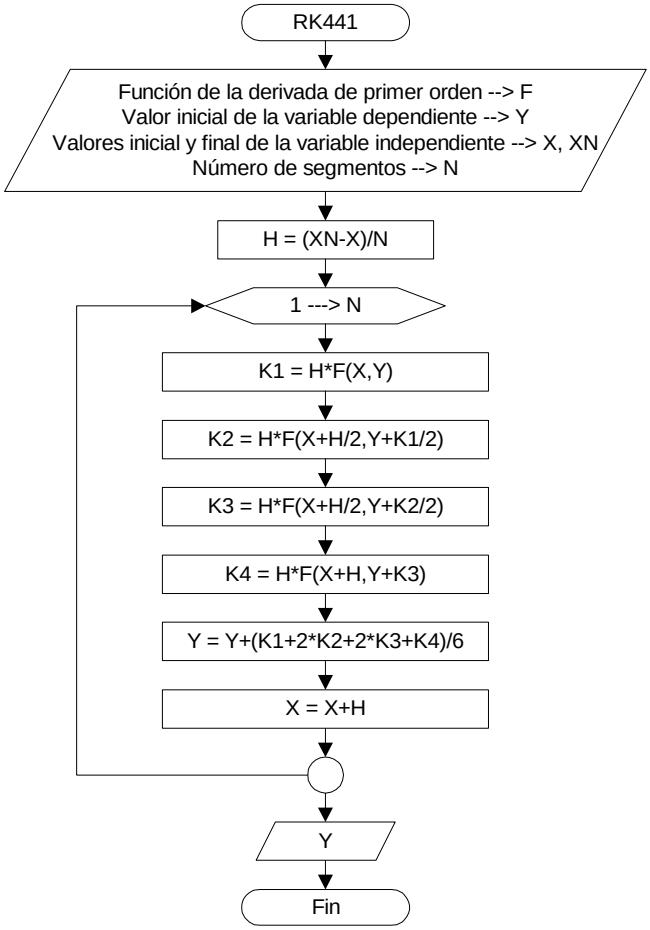


Figura 11.5. Algoritmo para resolver ecuaciones diferenciales de primer orden con el método de Runge – Kutta (forma clásica)

12.3.1.3. Código

El código elaborado siguiendo al algoritmo de la Figura 11.5, es el siguiente:

```
« 0 0 0 0 0 → F Y X XN N H K1 K2 K3 K4
 « XN X - N / 'H' STO
   1 N START
     H X Y F EVAL * 'K1' STO
     H X H 2 / + Y K1 2 / + F
      EVAL * 'K2' STO
     H X H 2 / + Y K2 2 / + F
      EVAL * 'K3' STO
     H X H + Y K3 + F EVAL *
      'K4' STO
     K1 2 K2 * + 2 K3 * + K4 + 6
      / 'Y' STO+
     H 'X' STO+
   NEXT
   Y
 »
»
```

Para ilustrar la utilización de este programa resolveremos el ejemplo manual. Para ello elaboramos el siguiente programa:

```
«
  « □ x y « 8 x 3 ^ * 4 x * - 4 x * y * - 3 / » »
  4 2 6 10 RK441 "y" □TAG
»
```

Con lo que obtenemos:

```
y: 68.9824081666
```

Que es el valor de ***y*** para ***x=6*** y que es el mismo resultado obtenido manualmente. Como se puede observar en este ejemplo, para resolver el problema con ***RK441*** se elabora el mismo programa que para ***EULE1***, excepto que en lugar de llamar a ***EULE1*** se llama a ***RK441***.

Al igual que el método de Euler, incrementando el número de segmentos (disminuyendo el valor de ***h)*** disminuye el error relativo, así haciendo correr el programa con 30 segmentos:

```
«
  « → x y « 8 x 3 ^ * 4 x * - 4 x * y * - 3 / » »
  4 2 6 30 RK441 "y" →TAG
»
```

Se obtiene:

```
y: 68.001355316
```

Cuyo error relativo es 0.002%.

12.3.2. Ecuaciones diferenciales de enésimo orden

Al igual que con el método de ***Euler***, los métodos de ***Runge-Kutta*** pueden ser empleados para resolver ecuaciones diferenciales de enésimo orden. Para ello se sigue el mismo procedimiento que en el método de ***Euler***, es decir se transforma la ecuación diferencial en un sistema de ecuaciones diferenciales de primer orden y se aplica la ecuación de ***Runge-Kutta*** a cada una de las ecuaciones resultantes.

Puesto que las primeras ***"n-1"*** ecuaciones del sistema son sólo igualdades, resultantes de los cambios de variable y no funciones propiamente (vea el acápite 12.2.2), el cálculo de los parámetros ***"k"*** así como los valores en el nuevo punto se simplifica para estas primeras ***n-1*** ecuaciones. Aplicando en-

tonces la ecuación de ***Runge Kutta*** a estas primeras ***n-1*** ecuaciones en un punto ***j*** para el cual se conocen los valores de ***x, y_1, y_2,*** hasta y_n se tiene:

$$\left.\begin{aligned}
&y_{i,j+1} = y_{i,j} + \frac{1}{6}\left(k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i}\right)\\
&k_{1,i} = hf_i\left(x_j, y_{1,j}, y_{2,j}, ..., y_{n,j}\right) = hy_{i+1,j}\\
&k_{2,i} = hf_i\left(x_j + \frac{h}{2}, y_{1,j} + \frac{k_{1,1}}{2}, y_{2,j} + \frac{k_{1,2}}{2}, ..., y_{n,j} + \frac{k_{1,n}}{2}\right) = h\left(y_{i+1,j} + \frac{k_{1,i+1}}{2}\right)\\
&k_{3,i} = hf_i\left(x_j + \frac{h}{2}, y_{1,j} + \frac{k_{2,1}}{2}, y_{2,j} + \frac{k_{2,2}}{2}, ..., y_{n,j} + \frac{k_{2,n}}{2}\right) = h\left(y_{i+1,j} + \frac{k_{2,i+1}}{2}\right)\\
&k_{4,i} = hf_i\left(x_j + h, y_{1,j} + k_{3,1}, y_{2,j} + k_{3,2}, ..., y_{n,j} + k_{3,n}\right) = h\left(y_{i+1,j} + k_{3,i+1}\right)
\end{aligned}\right\} i = 1 \rightarrow n-1 \qquad (30)$$

Donde $f_i(x, y_1, y_2, \ldots, y_n)$, son las funciones que representan las derivadas primeras de las variables y_i (las cuales sabemos son iguales a y_{i+1} para las primeras ***n-1*** ecuaciones); $k_{1,i}$, $k_{2,i}$, $k_{3,i}$ y $k_{4,i}$ son los valores de los parámetros k_1, k_2, k_3 y k_4 para la ecuación diferencial ***"y'_i"***, así $k_{1,2}$ es el valor del parámetro k_1 para la ecuación y'_2, $k_{4,5}$ es el valor del parámetro k_4 para la ecuación y'_5, etc.

Como ya se dijo el subíndice ***"j"*** representa al punto para el cual se conocen los valores de las variables: ***x, y_1, y_2, y_3, . . ., y_n*** y ***"j+1"*** es el punto consecutivo para el cual se calculan los valores de estas variables.

Aplicamos ahora la ecuación de ***Runge Kutta*** (en su forma clásica) a la derivada primera de la variable y_n, que como sabemos es la única función del sistema que no es simplemente una igualdad:

$$\begin{aligned}
&y_{n,j+1} = y_{n,j} + \frac{1}{6}\left(k_{1,n} + 2k_{2,n} + 2k_{3,n} + k_{4,n}\right)\\
&k_{1,n} = hf_n\left(x_j, y_{1,j}, y_{2,j}, ..., y_{n,j}\right)\\
&k_{2,n} = hf_n\left(x_j + \frac{h}{2}, y_{1,j} + \frac{k_{1,1}}{2}, y_{2,j} + \frac{k_{1,2}}{2}, ..., y_{n,j} + \frac{k_{1,n}}{2}\right) \qquad (31)\\
&k_{3,n} = hf_n\left(x_j + \frac{h}{2}, y_{1,j} + \frac{k_{2,1}}{2}, y_{2,j} + \frac{k_{2,2}}{2}, ..., y_{n,j} + \frac{k_{2,n}}{2}\right)\\
&k_{4,n} = hf_n\left(x_j + h, y_{1,j} + k_{3,1}, y_{2,j} + k_{3,2}, ..., y_{n,j} + k_{3,n}\right)
\end{aligned}$$

Al resolver estas ecuaciones se deben calcular primero todos los valores de ***"k_1"***, luego con estos valores todos los valores de ***"k_2"***, con estos los de ***"k_3"*** y finalmente los de ***"k_4"***. Como siempre el valor de la variable de pendiente se calcula con:

$$x_{j+1} = x_j + h \qquad (32)$$

12.3.2.1. Ejemplo manual

Para comprender mejor el procedimiento de cálculo resolveremos manualmente la ecuación diferencial:

$$3y''' - 3xy'' + y' + 1.8711x^{1.9} + 14.49x^{1.1} - 5 = 0$$

$$\begin{cases} y = 4,\ y' = 11.3,\ y'' = 6.93,\ y''' = 0.693 \ para\ x = 1 \\ y = ?,\ y' = ?,\ y'' = ?,\ y''' = ?\ para\ x = 1.43 \end{cases}$$

Que se transforma en el siguiente sistema de ecuaciones diferenciales de primer orden:

$$\frac{dy}{dx}=\frac{dy_1}{dx}=y'_1=y_2$$
$$\frac{d^2y}{dx^2}=\frac{dy_2}{dx}=y'_2=y_3$$
$$\frac{d^3y}{dx^3}=\frac{dy_3}{dx}=y'_3=y_4$$
$$\frac{d^4y}{dx^4}=\frac{dy_4}{dx}=y'_4=\frac{1}{3}(5-14.49x^{1.1}-1.8711x^{-1.9}-y_2+3xy_3)=y_5$$
$$\begin{cases} y_1=4,\ y_2=11.3,\ y_3=6.93,\ y_4=0.693\ para\ x=1 \\ y_1=?,\ y_2=?,\ y_3=?,\ y_4=?\ para\ x=1.43 \end{cases}$$

Donde y_1 es el valor de la variable dependiente "y". Fijando el número de segmentos en 10, el valor de h es:

$$h=\frac{1.43-1}{10}=0.043$$

Entonces calculamos los valores de k_1 aplicando la ecuación 30 para las 3 primeras ecuaciones (n-1) y la ecuación 31 para la cuarta (n):

$$k_{1,1}=hy_{1+1,0}=hy_{2,0}=0.043(11.3)=0.4859$$
$$k_{1,2}=hy_{2+1,0}=hy_{3,0}=0.043(6.93)=0.29799$$
$$k_{1,3}=hy_{3+1,0}=hy_{4,0}=0.043(0.693)=0.029799$$
$$k_{1,4}=hf_4\left(x_0,y_{1,0},y_{2,0},y_{3,0},y_{4,0}\right)=h\frac{1}{3}\left(5-14.49x^{1.1}-1.8711x^{-1.9}-y_2+3xy_3\right)$$
$$=(0.043)\frac{1}{3}\left(5-14.49(1)^{1.1}-1.8711(1)^{-1.9}-(11.3)+3(1)(6.93)\right)=-0.0268191$$

Ahora calculamos los valores de k_2 igualmente aplicando la ecuación 30 para las 3 primeras ecuaciones y la ecuación 31 para la cuarta:

$$k_{2,1}=h\left(y_{1+1,0}+\frac{k_{1,1+1}}{2}\right)=h\left(y_{2,0}+\frac{k_{1,2}}{2}\right)=0.043\left(11.3+\frac{0.29799}{2}\right)=0.492306785$$
$$k_{2,2}=h\left(y_{2+1,0}+\frac{k_{1,2+1}}{2}\right)=h\left(y_{3,0}+\frac{k_{1,3}}{2}\right)=0.043\left(6.93+\frac{0.029799}{2}\right)=0.2986306785$$
$$k_{2,3}=h\left(y_{3+1,0}+\frac{k_{1,3+1}}{2}\right)=h\left(y_{4,0}+\frac{k_{1,4}}{2}\right)=0.043\left(0.693+\frac{-0.0268191}{2}\right)=0.02922238935$$
$$k_{2,4}=hf_4\left(x_0+\frac{h}{2},y_{1,0}+\frac{k_{1,1}}{2},y_{2,0}+\frac{k_{1,2}}{2},y_{3,0}+\frac{k_{1,3}}{2},y_{4,0}+\frac{k_{1,4}}{2}\right)$$
$$=(0.043)\,f_4\left(1+\frac{0.043}{2},4+\frac{0.4859}{2},11.3+\frac{0.29799}{2},6.93+\frac{0.029799}{2},0.693+\frac{-0.0268191}{2}\right)$$
$$=(0.043)\,f_4\left(1.0215,4.24295,11.448995,6.9448995,0.67959045\right)$$
$$=(0.043)\frac{1}{3}\left(5-14.49(1.0215)^{1.1}-1.8711(1.0215)^{-1.9}-(11.448995)+3(1.0215)(6.9448995)\right)$$
$$=\text{-2.57482357242E-2}$$

Con estos valores calculamos los valores de k_3:

$$k_{3,1} = h\left(y_{1+1,0} + \frac{k_{2,1+1}}{2}\right) = h\left(y_{2,0} + \frac{k_{2,2}}{2}\right) = 0.043\left(11.3 + \frac{0.2986306785}{2}\right) = 0.49232055959$$

$$k_{3,2} = h\left(y_{2+1,0} + \frac{k_{2,2+1}}{2}\right) = h\left(y_{3,0} + \frac{k_{2,3}}{2}\right) = 0.043\left(6.93 + \frac{0.02922238935}{2}\right) = 0.298618281371$$

$$k_{3,3} = h\left(y_{3+1,0} + \frac{k_{2,3+1}}{2}\right) = h\left(y_{4,0} + \frac{k_{2,4}}{2}\right) = 0.043\left(0.693 + \frac{\text{-2.57482357242E-2}}{2}\right) = 2.92454129319\text{E-2}$$

$$k_{3,4} = hf_4\left(x_0 + \frac{h}{2}, y_{1,0} + \frac{k_{2,1}}{2}, y_{2,0} + \frac{k_{2,2}}{2}, y_{3,0} + \frac{k_{2,3}}{2}, y_{4,0} + \frac{k_{2,4}}{2}\right)$$

$$= (0.043)\, f_4\left(\begin{matrix} 1 + \frac{0.043}{2}, 4 + \frac{0.492306785}{2}, 11.3 + \frac{0.2986306785}{2}, 6.93 + \\ \frac{0.02922238935}{2}, 0.693 + \frac{\text{-2.57482357242E-2}}{2} \end{matrix}\right)$$

$$= (0.043)\, f_4\left(1.0215, 4.2461533925, 11.4493153392, 6.94461119468, 0.680125882138\right)$$

$$= (0.043)\frac{1}{3}\left(\begin{matrix} 5 - 14.49\left(1.2461533925\right)^{1.1} - 1.8711\left(1.2461533925\right)^{-1.9} - \left(11.3146111947\right) \\ +3\left(1.2461533925\right)\left(6.94461119468\right) \end{matrix}\right)$$

$$= \text{-2.57654909219E-2}$$

Ahora con los valores de k_3 calculamos los de k_4:

$$k_{4,1} = h\left(y_{1+1,0} + k_{3,1+1}\right) = h\left(y_{2,0} + k_{3,2}\right) = 0.043\left(11.3 + 0.298618281371\right) = 0.4987405961$$

$$k_{4,2} = h\left(y_{2+1,0} + k_{3,2+1}\right) = h\left(y_{3,0} + k_{3,3}\right) = 0.043\left(6.93 + 2.92454129319\text{E-2}\right) = 0.299247552756$$

$$k_{4,3} = h\left(y_{3+1,0} + k_{3,3+1}\right) = h\left(y_{4,0} + k_{3,4}\right) = 0.043\left(0.693 + \text{-2.57654909219E-2}\right) = 2.86910838904\text{E-2}$$

$$k_{4,4} = hf_4\left(x_0 + h, y_{1,0} + k_{3,1}, y_{2,0} + k_{3,2}, y_{3,0} + k_{3,3}, y_{4,0} + k_{3,4}\right)$$

$$= (0.043)\, f_4\left(\begin{matrix} 1 + 0.043, 4 + 0.49232055959, 11.3 + 0.298618281371, 6.93 + \\ 2.92454129319\text{E-2}, 0.693 + \left(\text{-2.57654909219E-2}\right) \end{matrix}\right)$$

$$= (0.043)\, f_4\left(1.043, 4.49232055959, 11.598618281371, 6.95924541293, 0.66723450978\right)$$

$$= (0.043)\frac{1}{3}\left(\begin{matrix} 5 - 14.49\left(1.043\right)^{1.1} - 1.8711\left(1.043\right)^{-1.9} - \left(11.598618281371\right) \\ +3\left(1.043\right)\left(6.93292454129\right) \end{matrix}\right)$$

$$= \text{-2.47569307049E-2}$$

Ahora contamos con todos los parámetros para calcular los nuevos valores de y_1, y_2, y_3 y y_4 en el nuevo punto:

$$y_{1,1} = y_{1,0} + \frac{1}{6}\left(k_{1,1} + 2k_{2,1} + 2k_{3,1} + k_{4,1}\right)$$

$$= 4 + \frac{1}{6}\left(0.4859 + 2\left(0.492306785\right) + 2\left(0.49232055959\right) + 0.4987405961\right) = 4.49231587921$$

$$y_{2,1} = y_{2,0} + \frac{1}{6}\left(k_{1,2} + 2k_{2,2} + 2k_{3,2} + k_{4,2}\right)$$

$$= 4 + \frac{1}{6}\left(0.29799 + 2\left(0.2986306785\right) + 2\left(0.298618281371\right) + 0.299247552756\right) = 11.5986225788$$

$$y_{3,1} = y_{3,0} + \frac{1}{6}\left(k_{1,3} + 2k_{2,3} + 2k_{3,3} + k_{4,3}\right)$$

$$= 4 + \frac{1}{6}\left(0.029799 + 2\left(0.02922238935\right) + 2\left(2.92454129319\text{E-2}\right) + 2.86910838904\text{E-2}\right) = 6.95923761474$$

$$y_{4,1} = y_{4,0} + \frac{1}{6}\left(k_{1,4} + 2k_{2,4} + 2k_{3,4} + k_{4,4}\right)$$
$$= 4 + \frac{1}{6}\left(\text{-0.0268191} + 2\left(\text{-2.57482357242E-2}\right) + 2\left(\text{-2.57654909219E-2}\right) + \text{-2.47569307049E-2}\right)$$
$$= 0.667232752667$$

Siendo el valor de la variable independiente en este nuevo punto:

$$x_1 = x_0 + h = 1 + 0.043 = 1.043$$

Con este nuevo punto conocido ($x_1, y_{1,1}, y_{1,2}, y_{1,3}, y_{1,4}$) se repite el proceso, obteniéndose así los valores del punto 2, luego con estos los del punto 3 y así sucesivamente hasta llegar al décimo punto. Puesto que el proceso manual ocupa demasiado espacio (se requerirían unas 20 páginas para mostrar todo el proceso), para los otros puntos se presenta simplemente un resumen de los resultados obtenidos:

$$y_{1,2} = 4.9974922487$$
$$y_{2,2} = 11.8984791698$$
$$y_{3,2} = 6.98740983259$$
$$y_{4,2} = 0.643407902994$$
$$x_2 = 1.086$$

$$y_{1,3} = 5.51560213951$$
$$y_{2,3} = 12.1995256879$$
$$y_{3,3} = 7.01459543337$$
$$y_{4,3} = 0.621310489594$$
$$x_3 = 1.129$$

$$y_{1,4} = 6.0466749001$$
$$y_{2,4} = 12.5017212467$$
$$y_{3,4} = 7.04086463763$$
$$y_{4,4} = 0.600756368019$$
$$x_4 = 1.172$$

$$y_{1,5} = 6.59076608807$$
$$y_{2,5} = 12.8050278171$$
$$y_{3,5} = 7.06628031223$$
$$y_{4,5} = 0.581586856771$$
$$x_5 = 1.215$$

$$y_{1,6} = 7.14792270569$$
$$y_{2,6} = 13.1094099376$$
$$y_{3,6} = 7.09089897001$$
$$y_{4,6} = 0.563664465$$
$$x_6 = 1.258$$

$$y_{1,7} = 7.7181902807$$
$$y_{2,7} = 13.4148344392$$
$$y_{3,7} = 7.11477160446$$
$$y_{4,7} = 0.546869454732$$
$$x_7 = 1.301$$

$$y_{1,8} = 8.3016129607$$
$$y_{2,8} = 13.7212702638$$
$$y_{3,8} = 7.13794439109$$
$$y_{4,8} = 0.53109705313$$
$$x_8 = 1.344$$

$$y_{1,9} = 8.89823359863$$
$$y_{2,9} = 14.0286882301$$
$$y_{3,9} = 7.16045928035$$
$$y_{4,9} = 0.516255175479$$
$$x_9 = 1.387$$

$$y_{1,10} = 9.50809383039$$
$$y_{2,10} = 14.337060883$$
$$y_{3,10} = 7.18235450167$$
$$y_{4,10} = 0.502267552779$$
$$x_{10} = 1.430$$

Que son los valores de la variable dependiente *(y_1)*, la derivada primera *(y_2)*, la derivada segunda *(y_3)* y la derivada tercera *(y_4)* para *x* igual a 1.43:

$$y = y_1 = 9.50809383039$$
$$\frac{dy}{dx} = y' = y_2 = 14.337060883$$
$$\frac{d^2y}{dx^2} = y'' = y_3 = 7.18235450167$$
$$\frac{d^3y}{dx^3} = y''' = y_4 = 0.502267552779$$

12.3.2.2. Algoritmo

El algoritmo que automatiza el procedimiento que se sigue al emplear la ecuación clásica de ***Runge Kutta*** en la resolución de ecuaciones diferencial de enésimo orden se presenta en la Figura 11.6.

Al observar este algoritmo se debe tomar en cuenta que las variables *"Y, K1, K2, K3* y *K4"* son vectores y que por lo tanto todas las operaciones que se realizan con ellas son operaciones vectoriales.

12.3.2.3. Código

El código elaborado en base al algoritmo de la Figura 11.6, es el siguiente:

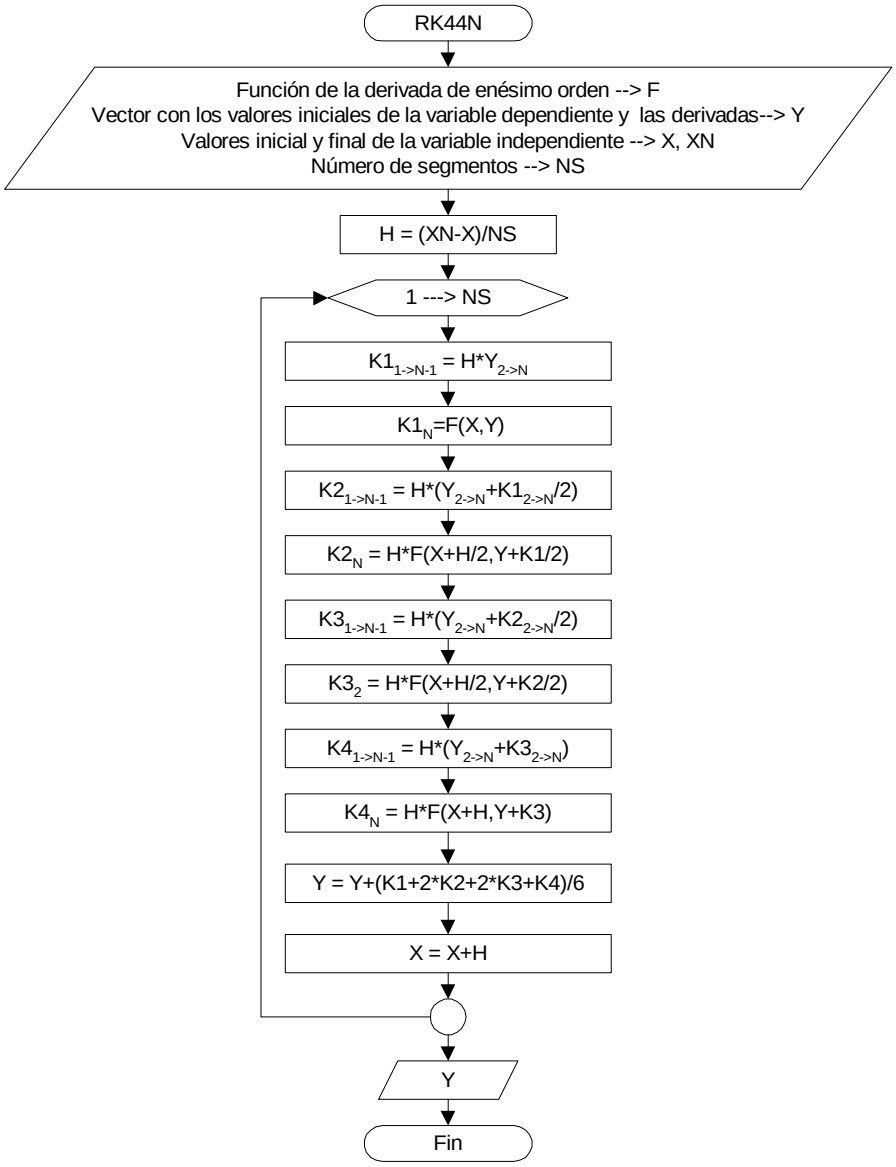


Figura 11.6. Algoritmo para resolver ecuaciones diferenciales de enésimo orden con la ecuación de Runge Kutta (forma clásica).

```
« 0 0 0 0 0 0
  □ F Y X XN NS N H K1 K2 K3 K4
 « XN X - NS / 'H' STO
   Y SIZE 1 GET 'N' STO
   1 NS START
     Y V□ N ROLL DROP
     X Y V□ F EVAL N □ARRY H * 'K1' STO
     Y K1 2 / + V□ N ROLL DROP
     X H 2 / + Y K1 2 / + V□ F EVAL N □ARRY H * 'K2' STO
     Y K2 2 / + V□ N ROLL DROP
     X H 2 / + Y K2 2 / + V□ F EVAL N □ARRY H * 'K3' STO
     Y K3 + V□ N ROLL DROP
     X H + Y K3 + V□ F EVAL N □ARRY H * 'K4' STO
     Y K1 2 K2 * + 2 K3 * + K4 + 6 / + 'Y' STO
     H 'X' STO+
   NEXT
   Y
```

```
 »
»
```

En lo sucesivo se asumirá que este programa ha sido guardado en la variable global *"RK44N"*. A diferencia de ***EULEN***, la función correspondiente a la derivada enésima no recibe "*Y*" como un vector, por lo tanto debe recibir un número de variables equivalente al orden de la ecuación (a más de la variable independiente). Así si se resuelve una ecuación de séptimo orden deberá recibir: x, y_1, y_2, y_3, y_4, y_5, y_6 y y_7.

Para ilustrar el uso de este programa resolveremos el ejemplo manual. Para ello elaboramos el siguiente programa:

```
«
  « □ X Y1 Y2 Y3 Y4
    « 5 14.49 X 1.1 ^ * - 1.8711 X -1.9 ^ * - Y2 - 3 X * Y3 * + 3 /
    »
  » [ 4 11.3 6.93 0.693 ] 1 1.43 10
  RK44N "Y" □TAG
»
```

Y haciendo correr el programa se obtiene:

```
Y: [ 9.50809383039 14.337060883 7.18235450167 .50226255278 ]
```

Que son los mismos resultados obtenidos en el ejemplo manual.

12.3.3. Sistemas de ecuaciones diferenciales

Al igual que el método de ***Euler***, los métodos de ***Runge-Kutta*** pueden ser empleados para resolver sistemas de ecuaciones diferenciales. En caso de que en el sistema existan ecuaciones diferenciales de segundo orden o superior, dichas ecuaciones deben ser transformadas en ecuaciones diferenciales de primer orden (tal como se procede cuando se resuelve una ecuación diferencial de enésimo orden). De esta manera siempre es posible transformar un sistema de ecuaciones diferenciales con ecuaciones de cualquier orden en un sistema de ecuaciones diferenciales de primer orden:

$$\begin{aligned} y'_1 &= f_1(x, y_1, y_2, y_3, ..., y_n) \\ y'_2 &= f_2(x, y_1, y_2, y_3, ..., y_n) \\ y'_3 &= f_3(x, y_1, y_2, y_3, ..., y_n) \\ &... \\ y'_n &= f_n(x, y_1, y_2, y_3, ..., y_n) \end{aligned} \qquad (20)$$

Entonces se aplica la ecuación de ***Runge Kutta*** a cada una de las ecuaciones de sistema. Así para un punto j cualquiera se tiene:

$$\left.\begin{aligned} y_{i,j+1} &= y_{i,j} + \frac{1}{6}\left(k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i}\right) \\ k_{1,i} &= h * f_i\left(x_j, y_{1,j}, y_{2,j}, ..., y_{n,j}\right) \\ k_{2,i} &= h * f_i\left(x_j + \frac{h}{2}, y_{1,j} + \frac{k_{1,1}}{2}, y_{2,j} + \frac{k_{1,2}}{2}, ..., y_{n,j} + \frac{k_{1,n}}{2}\right) \\ k_{3,i} &= h * f_i\left(x_j + \frac{h}{2}, y_{1,j} + \frac{k_{2,1}}{2}, y_{2,j} + \frac{k_{2,2}}{2}, ..., y_{n,j} + \frac{k_{2,n}}{2}\right) \\ k_{4,i} &= h * f_i\left(x_j + h, y_{1,j} + k_{3,1}, y_{2,j} + k_{3,2}, ..., y_{n,j} + k_{3,n}\right) \end{aligned}\right\} \quad i = 1 \rightarrow n \qquad (33)$$

Donde f_i es la función correspondiente a la derivadas primera de la variable y_i (tal como se define en la ecuación 20); $k_{1,i}$, $k_{2,i}$, $k_{3,i}$ y $k_{4,i}$ son los parámetros de la ecuación para la variable y_i; h es el incremento de la variable independiente (x). El subíndices j identifica los valores conocidos y $j+1$ los valores a calcular.

Al resolver este sistema, al igual que cuando se resuelve una ecuación de enésimo orden, se calculan primero para todas las ecuaciones los parámetros k_1, con estos se calculan los parámetros k_2, con estos k_3 y finalmente k_4.

12.3.3.1. Ejemplo manual

Para comprender mejor el procedimiento de cálculo se resolverá manualmente el siguiente sistema de ecuaciones diferenciales:

$$\begin{aligned}
&u' - v + 2w + t^2 - 2t^{1.5} - 1.2t^{0.2} + 10 = 0\\
&v' + 5w - 5t^{1.5} - 2t + 35 = 0\\
&w' - 2u + v - t^2 + 2t^{1.2} - 1.5t^{0.5} + 10 = 0\\
&\begin{cases} u = 3,\ v = -4,\ w = -7 \ para\ t = 0 \\ u = ?,\ v = ?,\ w = ?\ para\ t = 0.35 \end{cases}
\end{aligned}$$

Primero efectuamos un cambio de variables y reordenamos las ecuaciones de manera que estén en la forma de la ecuación (20):

$$\begin{aligned}
&y'_1 = f_1(x, y_1, y_2, y_3) = y_2 - 2y_3 - x^2 + 2x^{1.5} + 1.2x^{0.2} - 10\\
&y'_2 = f_2(x, y_1, y_2, y_3) = 5x^{1.5} + 2x - 5y_3 - 35\\
&y'_3 = f_3(x, y_1, y_2, y_3) = 2y_1 - y_2 + x^2 - 2x^{1.2} + 1.5x^{0.5} - 10\\
&\begin{cases} y_1 = 3,\ y_2 = -4,\ y_3 = -7 \ para\ x = 0 \\ y_1 = ?,\ y_2 = ?,\ y_3 = ?\ para\ x = 0.35 \end{cases}
\end{aligned}$$

En este ejemplo (al igual que con el método de Euler) fijaremos el número de segmentos en 10, entonces el incremento (h) es:

$$h = \frac{0.35 - 0}{10} = 0.035$$

Entonces empleando los valores del punto conocido (el punto cero) comenzamos calculando los valores $k1$:

$$\begin{aligned}
k_{1,1} &= h * f_1\left(x_0, y_{1,0}, y_{2,0}, y_{3,0}\right) = (0.035) * f_1(0, 3, -4, -7)\\
&= (0.035)\left((-4) - 2(-7) - (0)^2 + 2(0)^{1.5} + 1.2(0)^{0.2} - 10\right) = 0\\
k_{1,2} &= h * f_2\left(x_0, y_{1,0}, y_{2,0}, y_{3,0}\right) = (0.035) * f_2(0, 3, -4, -7)\\
&= (0.035)\left(5(0)^{1.5} + 2(0) - 5(-7) - 35\right) = 0\\
k_{1,3} &= h * f_3\left(x_0, y_{1,0}, y_{2,0}, y_{3,0}\right) = (0.035) * f_3(0, 3, -4, -7)\\
&= (0.035)\left(2(3) - (-4) + (0)^2 - 2(0)^{1.2} + 1.5(0)^{0.5} - 10\right) = 0
\end{aligned}$$

Ahora con los valores $k1$ calculamos los valores $k2$:

$$\begin{aligned}
k_{2,1} &= h * f_1\left(x_0 + \frac{h}{2}, y_{1,0} + \frac{k_{1,1}}{2}, y_{2,0} + \frac{k_{1,2}}{2}, y_{3,0} + \frac{k_{1,3}}{2}\right) = (0.035) * f_1\left(0 + \frac{0.035}{2}, 3 + \frac{0}{2}, -4 + \frac{0}{2}, -7 + \frac{0}{2}\right)\\
&= (0.035) * f_1(0.0175, 3, -4, -7)
\end{aligned}$$

$$= (0.035)\left((-4) - 2(-7) - (0.0175)^2 + 2(0.0175)^{1.5} + 1.2(0.0175)^{0.2} - 10\right) = 0.018851992047$$

$$k_{2,2} = h * f_2\left(x_0 + \frac{h}{2}, y_{1,0} + \frac{k_{1,1}}{2}, y_{2,0} + \frac{k_{1,2}}{2}, y_{3,0} + \frac{k_{1,3}}{2}\right) = (0.035) * f_2(0.0175, 3, -4, -7)$$

$$= (0.035)\left(5(0.0175)^{1.5} + 2(0.0175) - 5(-7) - 35\right) = 0.00163013067$$

$$k_{2,3} = h * f_3\left(x_0 + \frac{h}{2}, y_{1,0} + \frac{k_{1,1}}{2}, y_{2,0} + \frac{k_{1,2}}{2}, y_{3,0} + \frac{k_{1,3}}{2}\right) = (0.035) * f_3(0.0175, 3, -4, -7)$$

$$= (0.035)\left(2(3) - (-4) + (0.0175)^2 - 2(0.0175)^{1.2} + 1.5(0.0175)^{0.5} - 10\right) = 0.006410380067$$

Entonces con los valores *k2* calculamos los valores *k3*:

$$k_{3,1} = h * f_1\left(x_0 + \frac{h}{2}, y_{1,0} + \frac{k_{2,1}}{2}, y_{2,0} + \frac{k_{2,2}}{2}, y_{3,0} + \frac{k_{2,3}}{2}\right)$$

$$= (0.035) * f_1\left(0 + \frac{0.035}{2}, 3 + \frac{0.018851992047}{2}, -4 + \frac{0.00163013067}{2}, -7 + \frac{0.006410380067}{2}\right)$$

$$= (0.035) * f_1(0.0175, 3.00942599602, -3.99918493466, -6.99679480997)$$

$$= 0.018656156029$$

$$k_{3,2} = h * f_2\left(x_0 + \frac{h}{2}, y_{1,0} + \frac{k_{2,1}}{2}, y_{2,0} + \frac{k_{2,2}}{2}, y_{3,0} + \frac{k_{2,3}}{2}\right)$$

$$= (0.035) * f_2(0.0175, 3.00942599602, -3.99918493466, -6.99679480997)$$

$$= 0.001069222413$$

$$k_{3,3} = h * f_3\left(x_0 + \frac{h}{2}, y_{1,0} + \frac{k_{2,1}}{2}, y_{2,0} + \frac{k_{2,2}}{2}, y_{3,0} + \frac{k_{2,3}}{2}\right)$$

$$= (0.035) * f_3(0.0175, 3.00942599602, -3.99918493466, -6.99679480997)$$

$$= 7.0416725015\text{E-}3$$

Finalmente con los valores *k3* calculamos los valores *k4*:

$$k_{4,1} = h * f_1\left(x_0 + h, y_{1,0} + k_{3,1}, y_{2,0} + k_{3,2}, y_{3,0} + k_{3,3}\right)$$

$$= (0.035) * f_1(0 + 0.035, 3 + 0.018656156029, -4 + 0.001069222413, -7 + 7.0416725015\text{E-}3)$$

$$= (0.035) * f_1(0.035, 3.018656156029, -3.99893077759, -6.9929583275)$$

$$= 0.021441399427$$

$$k_{4,2} = h * f_2\left(x_0 + h, y_{1,0} + k_{3,1}, y_{2,0} + k_{3,2}, y_{3,0} + k_{3,3}\right)$$

$$= (0.035) * f_2(0.035, 3.018656156029, -3.99893077759, -6.9929583275)$$

$$= 0.002363589886$$

$$k_{4,3} = h * f_3\left(x_0 + h, y_{1,0} + k_{3,1}, y_{2,0} + k_{3,2}, y_{3,0} + k_{3,3}\right)$$

$$= (0.035) * f_3(0.035, 3.018656156029, -3.99893077759, -6.9929583275)$$

$$= 9.8801511935\text{E-}3$$

Ahora se tienen todos los parámetros *k*, de manera que se pueden calcular los valores de las variables dependientes en el nuevo punto:

$$y_{1,1} = y_{1,0} + \frac{1}{6}\left(k_{1,1} + 2k_{2,1} + 2k_{3,1} + k_{4,1}\right)$$

$$= 3 + \frac{1}{6}\left(0 + 2(0.018851992047) + 2(0.018656156029) + 0.021441399427\right)$$

$$= 3.0160762826$$

$$y_{2,1} = y_{2,0} + \frac{1}{6}\left(k_{1,2} + 2k_{2,2} + 2k_{3,2} + k_{4,2}\right) =$$

$$= -4 + \frac{1}{6}\left(0 + 2\left(0.00163013067\right) + 2\left(0.001069222413\right) + 0.002363589886\right)$$

$$= \text{-}3.99870628399$$

$$y_{3,1} = y_{3,0} + \frac{1}{6}\left(k_{1,3} + 2k_{2,3} + 2k_{3,3} + k_{4,3}\right) =$$

$$= -7 + \frac{1}{6}\left(0 + 2\left(0.006410380067\right) + 2\left(7.0416725015\text{E-}3\right) + 9.8801511935\text{E-}3\right)$$

$$= \text{-}6.99386929061$$

Siendo el valor de la variable independiente en el nuevo punto:

$$x_1 = x_0 + h = 0 + 0.035 = 0.035$$

Con el nuevo punto calculado (x1,y1,1,y2,1,y3,1) se repite el proceso y se calculan los valores del punto 2, con estos los del 3 y así sucesivamente hasta llegar al décimo punto (pues hemos elegido 10 segmentos). Una vez más debido al espacio que ocupan los cálculos manuales para los otros puntos sólo mostramos un resumen de los resultados obtenidos:

$$y_{1,2} = 3.0393376684$$
$$y_{2,2} = \text{-}3.99494624844$$
$$y_{3,2} = \text{-}6.98202839108$$
$$x_2 = 0.07$$

$$y_{1,3} = 3.06516179195$$
$$y_{2,3} = \text{-}3.9887136213$$
$$y_{3,3} = \text{-}6.96665565553$$
$$x_3 = 0.105$$

$$y_{1,4} = 3.09280813054$$
$$y_{2,4} = \text{-}3.9800081608$$
$$y_{3,4} = \text{-}6.94842730175$$
$$x_4 = 0.140$$

$$y_{1,5} = 3.12189673812$$
$$y_{2,5} = \text{-}3.96882980942$$
$$y_{3,5} = \text{-}6.92773370973$$
$$x_5 = 0.175$$

$$y_{1,6} = 3.15219159275$$
$$y_{2,6} = \text{-}3.95517856839$$
$$y_{3,6} = \text{-}6.9048381305$$
$$x_6 = 0.21$$

$$y_{1,7} = 3.18353024508$$
$$y_{2,7} = \text{-}3.93905448856$$
$$y_{3,7} = \text{-}6.87993370138$$
$$x_7 = 0.245$$

$$y_{1,8} = 3.21579335479$$
$$y_{2,8} = -3.92045767574$$
$$y_{3,8} = -6.85316994593$$
$$x_8 = 0.280$$

$$y_{1,9} = 3.24888917738$$
$$y_{2,9} = -3.89938829918$$
$$y_{3,9} = -6.82466703999$$
$$x_9 = 0.315$$

$$y_{1,10} = 3.28274477202$$
$$y_{2,10} = -3.87584660093$$
$$y_{3,10} = -6.79452427755$$
$$x_{10} = 0.350$$

Siendo estos últimos valores las soluciones buscadas: $y_{1,10} = u = 3.28274477202$, $y_{2,10} = v = -3.87584660093$; $y_{3,10} = w = -6.79452427755$. Los resultados exactos: $u = 3.28371457908$, $v = -3.8775$, $w = -6.79293720759$, por lo tanto los errores relativos son 0.030% para u, 0.043% para v y 0.023% para w.

12.3.3.2. Algoritmo

El algoritmo que automatiza el proceso de cálculo se presenta en la figura Figura 11.7.

Como se puede observar este algoritmo es prácticamente el mismo que permite resolver una ecuación diferencial de primer orden, sin embargo se debe tomar en cuenta que ahora *Y, K1, K2, K3* y *K4* son vectores y no escalares, por lo tanto todas las operaciones en las que intervienen son operaciones vectoriales.

12.3.3.3. Código

El código elaborado en base al algoritmo de la Figura 11.7, es el siguiente:

```
« 0 0 0 0 0 0
  □ F Y X XN NS N H K1 K2 K3 K4
 « Y SIZE 1 GET 'N' STO
   XN X - NS / 'H' STO
   1 NS START
     H X Y V□ F EVAL N □ARRY * 'K1' STO
     H X H 2 / + Y K1 2 / + V□ F
      EVAL N □ARRY * 'K2' STO
     H X H 2 / + Y K2 2 / + V□ F
      EVAL N □ARRY * 'K3' STO
     H X H + Y K3 + V□ F EVAL N □ARRY *
      'K4' STO
     Y K1 2 K2 * + 2 K3 * + K4 + 6
      / + 'Y' STO
     H 'X' STO+
   NEXT
   Y
 »
»
```

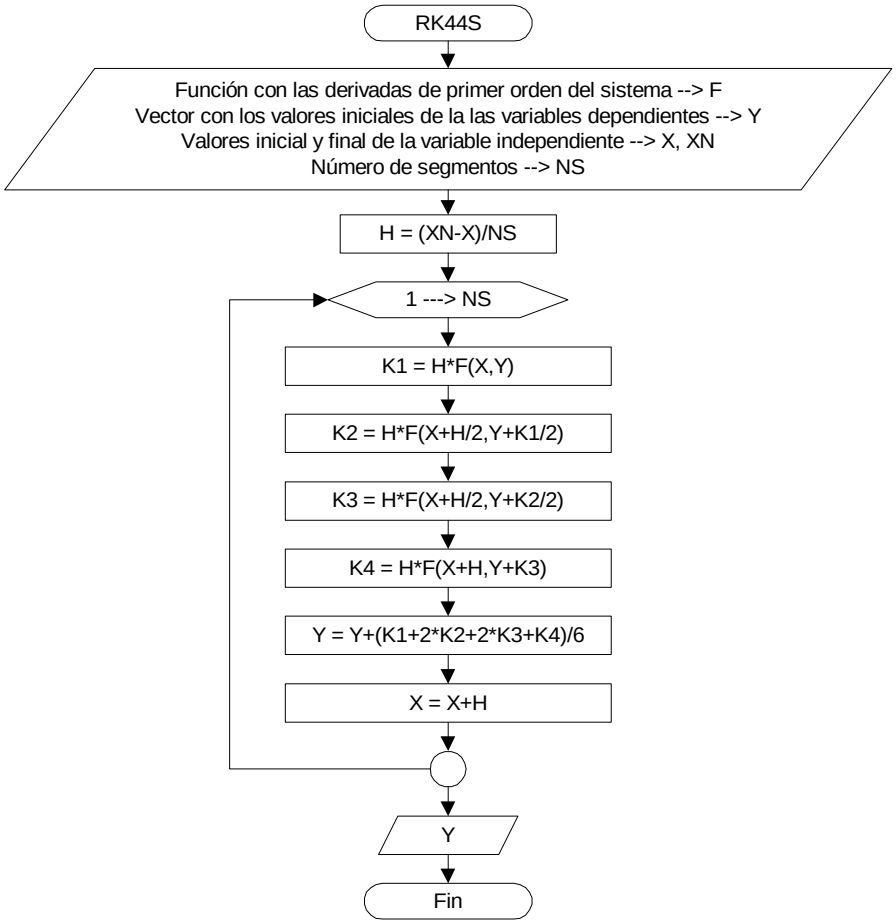


Figura 11.7. Algoritmo para resolver sistemas de ecuaciones diferenciales con la ecuación de Euler en su forma clásica.

En lo sucesivo se asumirá que el programa ha sido guardado con el nombre ***"RK44S"***. La función que se debe mandar a este programa no recibe las variables dependientes en forma de un vector, de manera que debe recibir la variable independiente y tantas variables como ecuaciones existan en el sistema.

Para ilustrar el uso de este programa resolveremos el ejemplo manual. Para ello elaboramos el siguiente programa:

```
«
  « □ X Y1 Y2 Y3
    « Y2 2 Y3 * - X SQ - 2 X 1.5 ^ * + 1.2 X 0.2 ^ * + 10 -
      5 X 1.5 ^ * 2 X * + 5 Y3 * - 35 -
      2 Y1 * Y2 - X SQ + 2 X 1.2 ^ * - 1.5 X 0.5 ^ * + 10 -
    »
  » [ 3 -4 -7 ] 0 0.35 10
  RK44S "Y" □TAG
»
```

Haciendo correr el programa se obtiene:

```
Y: [ 3.28274477202 -3.87584660093 -6.79452427755 ]
```

Que son los mismos resultados obtenidos en la solución manual.

12.3.4. Preguntas y ejercicios

1. ¿Cuantas formas de las ecuaciones de Runge – Kutta existen?
2. ¿Todas las formas de Runge – Kutta predicen correctamente los valores de la variable dependiente?
3. Para resolver la ecuación de cuarto orden del acápite 12.3.2.1, empleando el programa ***RK44S*** ¿Cuál sería la función que se debería mandar al programa?
4. Resuelva los ejercicios 19 al 29 del acápite 12.2.4 empleando el método de Runge Kutta.

2. ERRORES EN LOS CÁLCULOS NUMÉRICOS

Los métodos numéricos nos permiten resolver problemas matemáticos, sin recurrir a las relaciones que serían necesarias en la solución analítica. Para resolver un problema, empleando un método numérico, generalmente es suficiente conocer el concepto matemático del problema a resolver.

Las soluciones que proporcionan los métodos numéricos tienen un carácter más general, así, una vez implementado un método para la resolución de ecuaciones algebraicas, se puede resolver con el mismo prácticamente cualquier ecuación sin importar lo compleja que esta sea, algo que no ocurre con los métodos analíticos, donde cada ecuación es un problema que se resuelve de manera diferente.

La principal desventaja de estos métodos y la razón por la cual no fueron de aplicación práctica hasta la invención de la computadora, es la de requerir un elevado número de cálculos. Tanto los principios como los métodos numéricos en si, existen ya hace varios siglos, sin embargo, no eran empleados con frecuencia debido a lo tedioso, que resulta para el ser humano, la realización de un elevado número de cálculos numéricos.

Los métodos numéricos comenzaron a tener utilidad práctica con la invención de las computadoras, las cuales en realidad fueron inventadas justamente con este propósito. Como una computadora no sólo puede realizar cálculos numéricos, sino que además puede recordar la secuencia de pasos que debe seguir, resultan ideales para la resolución de problemas con métodos numéricos. El conocer un método numérico y aplicarlo manualmente resulta tan práctico como tener un camino asfaltado entre dos ciudades y realizar el viaje a pie, cuando se cuenta con un automóvil.

Como ya se dijo, en esta materia resolveremos los problemas con la ayuda de la calculadora HP, en sus modelos 48 y 49, la misma que como ya se vió en el anterior capítulo, es en realidad una computadora que puede ser programada.

Cuando se resuelven problemas empleando métodos numéricos se presentan algunos errores, que son causados por una parte por la naturaleza del método en sí y por otra por la forma en que operan las computadoras. Estos errores deben ser tomados en cuenta a fin de minimizarlos y de ser posible evitarlos. A continuación estudiaremos algunos de dichos errores.

2.1. Errores de truncamiento

Los errores de truncamiento se deben al método numérico. Se presenta en aquellos métodos que requieren un infinito número de cálculos para encontrar la solución exacta. Como en la práctica no es posible realizar un número infinito de cálculos, no queda otra alternativa que limitar (truncar) los cálculos a un número finito, introduciéndose así un error, pues no se obtiene el resultado exacto.

Por ejemplo para calcular el exponente de un número real “x” se emplea la siguiente serie de Taylor:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} + ...\infty$$

Que como vemos involucra la suma de un número infinito de términos, por lo tanto en la práctica no queda otra alternativa que limitar la suma a un número finito de términos, introduciéndose así un error de truncamiento.

En estos casos surge la pregunta: ¿cuántas veces debemos repetir los cálculos para obtener un resultado lo suficientemente preciso en un tiempo razonable? En la práctica los cálculos se repiten hasta que los dos últimos resultados obtenidos son iguales en un determinado número de dígitos, esto ocurre cuando se cumple la siguiente relación:

$$\left|\frac{s_1}{s_2} - 1\right| < 1x10^{-n} \qquad (1)$$

Donde *"n"* es el número de dígitos de precisión, es decir el número de dígitos que son iguales en los dos últimos resultados, siendo *"s_1"* y *"s_2"* los dos últimos resultados calculados. En la mayoría de los cálculos ingenieriles es suficiente una precisión de 6 dígitos.

Por ejemplo si los dos últimos resultados son: $s_1=12.345678943$ y $s_2=12.345678832$, aplicando la ecuación (1) se obtiene

$$\left|\frac{12.345678943}{12.345678832} - 1\right| = 0.00000000899 = 8.99x10^{-9}$$

Que es menor a $1x10^{-8}$, por lo tanto estos valores son iguales en los 8 primeros dígitos (algo que puede comprobar examinando los números). Es importante hacer notar que esta expresión compara los números sin importar la potencia a la que estén elevados, así con: $s_1=12.345678943x10^{-20}$ y $s_2=12.345678832x10^{-20}$, se obtiene el mismo resultado que en el caso anterior pues siguen siendo iguales en los 8 primeros dígitos.

Emplearemos la ecuación 1, como condición para terminar los métodos iterativos, cuando el resultado tenga los dígitos de precisión deseados.

2.2. Errores de redondeo

Todas las computadoras trabajan con un número finito de dígitos, por lo tanto los resultados de las operaciones son redondeados a ese número de dígitos. Por ejemplo si una computadora trabajara con 4 dígitos, el resultado de: 3.453 * 2.457 (que es 8.484021) será redondeado a 8.484, con lo que se pierden (definitivamente) los tres últimos dígitos (021).

Si este resultado se emplea en otras operaciones, estos resultados son también redondeados, con lo que los errores de redondeo se van acumulando y el error se va creciendo.

La mayoría de los lenguajes de programación permiten trabajar con diferentes dígitos de precisión. Mientras mayor sea el número de dígitos, menor será el error de redondeo, pero como es imposible trabajar con un número infinito de dígitos, es imposible evitar realmente este error. Por otra parte mientras más dígitos se empleen en los cálculos mayor es el tiempo de computación requerido.

Veamos por ejemplo cual sería el resultado que se obtiene cuando se realizan las siguientes operaciones con una computadora hipotética que trabaja con 5 dígitos de exactitud: (1.2345512*6.5423493)/6.7898482-0.9321351. En primer lugar al introducir el número 1.2345512 este número es redondeado a 5 dígitos: 1.2346, igualmente al introducir el número 6.5423493 se redondea a 6.5423, entonces se multiplican los números redondeados y el resultado (8.07712358) es redondeado a 5 dígitos: 8.0771, entonces se divide este resultado entre 6.7898482 previamente redondeado a 5 dígitos: 6.7898 y el resultado (1.18959321335) es también redondeado: 1.1896. Finalmente se le resta el número 0.9321351 el cual es previamente redondeado: 0.93214, con lo que se obtiene el resultado final: 0.25746.

Si en lugar de trabajar con 5 dígitos se trabaja con 12, el resultado obtenido es: 0.25741508451, que redondeado a 5 dígitos es 0.25742 que como vemos difiere del resultado calculado en el último dígito.

2.3. Error en los datos originales

Estos son los errores que se encuentran en los datos originales. Si bien en general es muy difícil evitarlos, es posible en ocasiones detectarlos mediante un análisis. Por ejemplo si los datos originales son los siguientes:

X	1	3	8	12	16
y	0.56	1.68	2.76	20.32	4.02

Es evidente que el cuarto dato no es correcto pues tiene un valor de *"y"* demasiado alto con relación a los otros.

Cuando los datos son ecuaciones empíricas, se puede realizar pruebas de sensibilidad para determinar cuáles de las constantes pueden causar más error. Por ejemplo si la ecuación empírica es la siguiente:

$$y=3.246+x^{2.251}$$

Calculamos primero algunos resultados con la misma, por ejemplo los siguientes (redondeados al primer dígito):

x	3	10	20	50	70
y	15.1	181.5	851.7	6677.2	14236.9

Se modifica luego alguna de las constantes, por ejemplo la primera: *3.246* que por ejemplo puede ser redondeada al primer dígito: *3.3* y se vuelven a repetir los cálculos con la constante modificada: $y=3.3+x^{2.251}$:

x	3	10	20	50	70
y	15.2	181.5	851.7	6677.2	14236.9

Entonces se comparan estos resultados con los de la ecuación original. En este caso por ejemplo son prácticamente iguales, por lo que podemos concluir que esta ecuación no es sensible a la primera constante y que por lo tanto no causa mucho error. Repitiendo el anterior procedimiento con el segundo parámetro: $y=3.246+x^{2.3}$ se obtiene:

x	3	10	20	50	70
y	15.8	202.8	985.8	8087.3	17531.0

Y como se puede observar estos resultados difieren considerablemente con relación a los obtenidos con la ecuación original, se concluye entonces que esta ecuación es sensible a la segunda constante y que por lo tanto puede ser causa de errores. Entonces se debe revisar cuidadosamente la exactitud de esta constante para no introducir errores en los cálculos.

2.4. Errores humanos

Las computadoras muy rara vez cometen errores, pero los que las programan si. Es frecuente que en la elaboración de programas se cometan errores, es también frecuente que se cometan errores al emplear un programa elaborado (principalmente al introducir los datos).

La solución más efectiva para reducir los errores en la elaboración de programas es la práctica. Mientras más programas se elaboren menos errores se cometerán.

Por otra parte es importante probar los programas haciéndolos correr con datos para los cuales se conocen los resultados y en caso de error, hacer correr el programa paso a paso para detectar él o los errores cometidos.

Al igual que sucede con la elaboración de programas, la forma más eficiente de reducir los errores al emplear un programa es la práctica. Mientras más veces se emplee el programa menos errores se cometerán.

2.5. Propagación del error

La propagación del error se presenta en aquellos métodos que emplean los resultados calculados en una etapa para calcular los resultados de la siguiente y los de estos para los de la subsiguiente y así sucesivamente. De esta manera los errores cometidos en la primera etapa se propagan a la siguiente y los de estos a la subsiguiente y así sucesivamente propagando e incrementando el error en cada etapa.

La propagación del error se reducirse si los resultados calculados en cada etapa se corrigen antes de pasar a la siguiente.

2.6. Error absoluto y relativo

El error cometido al calcular un resultado empleando métodos numéricos se expresa usualmente como *error absoluto* o *error relativo*.

El error absoluto se calcula con:

$$Error\ absoluto = \left| valor\ verdadero - valor\ aproximado \right|$$

Donde valor verdadero es el valor más exacto disponible. El problema con el error absoluto es que devuelve errores más grandes cuanto más grandes son las magnitudes involucradas. Por ejemplo si el *valor verdadero* es *45897.25* y el *valor aproximado 45845.26*, el error absoluto será: *|45897.25-45845.26| = 51.99*, que por su magnitud parece ser un error importante, aunque en realidad los dos valores son aproximadamente iguales y por lo tanto el error no es muy grande. Por el contrario si el *valor verdadero* es *0.00002478* y el *valor aproximado 0.00000578*, el error absoluto será: *|0.00002478-0.00000578|=0.000019*, que por su magnitud parece un error insignificante, no obstante los dos valores son totalmente diferentes por lo que el error es realmente grande.

Debido a lo anterior se prefiere el error relativo:

$$Error\ \text{relativo} = \left| \frac{valor\ verdadero - valor\ aproximado}{valor\ verdadero} \right|$$

Este error es prácticamente independiente de la magnitud de los valores, así para los mismos datos empleados anteriormente se obtiene: *|(45897.25-45845.26)/45897.25| = 0.00113 (1.1%)* para el primer ejemplo y *|(0.00002478-0.00000578)/0.00002478|=0.7667 (76.7%)* para el segundo, valores que reflejan mucho mejor los errores cometidos. Con frecuencia el error relativo se expresa en forma de porcentaje.

2.7. Convergencia, divergencia y oscilación

La mayoría de los métodos numéricos son métodos iterativos (repetitivos), es decir son procesos que se repiten hasta que se obtiene el resultado buscado. Siempre que se resuelve un problema iterativo se pueden presentar tres casos:

a) *Divergencia*: Se dice que las iteraciones son divergentes cuando los valores calculados en iteraciones sucesivas se alejan cada vez más de la solución.

b) *Convergencia*: Se dice que las iteraciones son convergentes cuando los valores calculados en iteraciones sucesivas se acercan cada vez más a la solución.

c) *Oscilación*: Se dice que las iteraciones son *oscilatorias* cuando los valores calculados en iteraciones sucesivas oscilan entre dos valores.

Además, las iteraciones pueden ser oscilatorias y divergentes u oscilatorias y convergentes a la vez.

En un método iterativo es deseable la *convergencia*, siendo indeseables la *divergencia* y la *oscilación*. Si un método converge en la mayoría de los casos, se dice que es *estable*, por el contrario si *diverge* u *oscila* en la mayoría de los casos es *inestable*. Por supuesto los métodos más utilizados en la práctica son los métodos estables.

2.8. Preguntas y ejercicios

1. ¿Son más generales las soluciones analíticas o numéricas?
2. ¿Por qué los métodos numéricos no eran empleados con frecuencia antes de la invención de las computadoras?
3. ¿A qué se deben los errores de truncamiento?
4. ¿Es posible evitar los errores de truncamiento?
5. ¿Qué relación se emplea para determinar si dos números son iguales en un determinado número de dígitos?
6. Si se desea comparar dos números: *x1* y *x2,* de manera que sean iguales en los 8 primeros dígitos ¿Cuál es la condición a emplear?
7. ¿A qué se deben los errores de redondeo?
8. ¿Es posible evitar los errores de redondeo?
9. Si trabaja con una computadora que tiene 6 dígitos de precisión ¿Cuál es el resultado que se obtiene de las siguientes operaciones: (6.79386/3.20839-1.29839568)* 4.8512556E-9?
10. ¿Qué se puede hacer para reducir el efecto de los errores en los datos originales?
11. Analice los siguientes datos y diga si es probable que exista algún error en los mismos.

X	10	30	60	90	120
Y	8.9	20.2	0.25	50.2	61.3

12. En la ecuación empírica: $y = x^{4.26}+x^{0.23}+6.34$, donde *"x"* varían entre 20 y 80. ¿Para qué constantes es sensible esta ecuación?
13. ¿Cuál es la forma más efectiva de reducir los errores humanos?
14. ¿En qué consiste la propagación del error?
15. ¿Cuál es la relación que permite calcular el error absoluto?
16. ¿Cuál es la relación que permite calcular el error relativo?
17. Entre el error absoluto y relativo, ¿Cuál de los dos refleja mejor el verdadero error cometido?

18. Calcule los errores absoluto y relativo si el valor exacto es *109.9392* y el valor aproximado *108.3829*.
19. ¿Cuándo se dice que las iteraciones son divergentes?
20. ¿Cuándo se dice que las iteraciones son convergentes?
21. ¿Cuándo se dice que las iteraciones son oscilatorias?
22. ¿Cuándo se dice que las iteraciones son oscilatorias y convergentes?

3. ECUACIONES ALGEBRAICAS CON UNA INCÓGNITA

En este capítulo estudiaremos algunos métodos que permiten resolver ecuaciones algebraicas con una incógnita, es decir funciones que dependen de una sola variable. Los métodos que estudiaremos pueden ser empleados también para resolver un sistema ecuaciones si las mismas dependen de una sola variable.

3.1. ROOT

La calculadora HP cuenta con el programa *ROOT* con el cual se puede resolver ecuaciones algebraicas con una incógnita. Para emplear este programa se deben colocar en la pila (y en ese orden) la ecuación algebraica a resolver (en forma de programa), la variable de la cual depende la ecuación (entre apóstrofes) y un valor inicial asumido.

La función a resolver debe estar en la forma *f(x)=0*, es decir la función debe estar igualada a cero. Por ejemplo, para resolver la siguiente ecuación:

$$x = \left(52 + 3\sqrt{x} - 8x^{0.8}\right)^{0.36} \qquad (3.1)$$

Primero la igualamos a cero:

$$f(x) = \left(52 + 3\sqrt{x} - 8x^{0.8}\right)^{0.36} - x = 0$$

Y escribimos lo siguiente en la pila:

```
« 52 3 X √ * + 8 X .8 ^ * - .36 ^ X - » 'X' 1.1 ROOT
```

Con lo que se obtiene la respuesta: *3.60967088614*.

Como se puede ver este método requiere una variable global: *'X'*, la cual, si no existía, es creada por *PROOT*, quedando el resultado guardado en la misma. Esto es algo que debe tomarse muy en cuenta pues si ya existe una variable con el mismo nombre, su contenido es reemplazado con el resultado calculado por ROOT.

Recuerde que los datos pueden ser separados con espacios (como se muestra en el ejemplo) o cada dato puede ser colocado en una línea diferente.

En la mayoría de los programas que estudiaremos controlaremos también el tiempo empleado en encontrar la solución, para ellos ejecutaremos los programas con el comando *TEVAL* (en lugar de *EVAL)*, el cual no existe en la HP48, por lo que para este modelo deberá escribir un programa adicional.

Los métodos de este capítulo serán programados en una librería, la librería número 525, a la cual denominaremos: *Alge1*. Para ello cree el proyecto *ALGE1* (dentro del directorio *RPN\ALGE1)*. En el proyecto seleccione la opción *The project is a library*, en el campo *Title* escriba *Alge1* y en el campo *RomId* escriba el número *20D* (que es el número 525 en hexadecimal). Active también el emulador *(Emulator -> Emulate)* y guarde la memoria del emulador *(File -> Save As)* con el nombre *MAT205* en el directorio *RPN*.

Como se dijo, la HP48 no cuenta con el comando *TEVAL*, por lo que para este modelo cree en el proyecto el archivo *Extras.S (Project -> New Source File*) y en él escriba el siguiente código:

```
xNAME TEVAL
:: ( f=2 : función a evaluar; t=1 : Tiempo inicial del sistem )
  CK1NOLASTWD
```

```
  CK&DISPATCH0
  EIGHT ( El dato tiene que ser un subprograma )
  ::
    TOD ( Tiempo antes de evaluar la función )
    ' NULLLAM TWO NDUPN DOBIND
    2GETLAM EVAL ( Se evalúa la función )
    TOD 1GETLAM %HMS- ( Tiempo final - Tiempo inicial )
    %HMS> % 3600 %* ( Convierte el resultado en segundos )
    %4 RNDXY  "s" >TAG ( Redondea al cuarto decimal y arma la etiqueta )
    ABND
  ;
;
```

Entonces guarde las modificaciones y compile el proyecto, con lo que la librería será cargada en el puerto cero y ahora podrá utilizar en la HP48 el comando *TEVAL* (no se olvide guardar la memoria del emulador *File -> Save)*.

Ahora volvamos a resolver la ecuación (3.1) pero controlando el tiempo que requiere el programa, para ello escriba lo siguiente en la pila:

```
« 52 3 X √ * + 8 X .8 ^ * - .36 ^ X - » 'X' 1.1 « ROOT » TEVAL
```

Donde como se puede ver, el comando ROOT ha sido colocado dentro de una secuencia para que no se ejecute automáticamente, entonces obtendrá el resultado *(3.60967088614)* y el tiempo en segundos empleado en encontrarlo (aproximadamente *1.234* segundos).

3.2. Método de sustitución directa

El método de *Sustitución Directa*, es el método más sencillo que existe para la resolución de ecuaciones algebraicas con una incógnita. Para aplicar este método, la ecuación (o sistema de ecuaciones) debe encontrarse en la forma:

$$x = g(x) \qquad (3.2)$$

Si no está en esta forma simplemente se despeja una de las variables independientes, normalmente se elige la expresión que resulte más sencilla de despejar.

Una vez que la ecuación se encuentra en la forma $x = g(x)$, se asume un valor para la solución (x_1) y se reemplaza en la función $g(x)$, con lo que se obtiene un nuevo valor de x (x_2), entonces se comparan el valor asumido (x_1) con el valor calculado (x_2) y si son aproximadamente iguales el proceso concluye, siendo la respuesta o solución x_2, caso contrario x_1 toma el valor de x_2 $(x_1 = x_2)$ y con este nuevo valor el proceso se repite.

Este método es tan sencillo que puede ser empleado simplemente programando la función a resolver. Por ejemplo para resolver la ecuación (3.1), que se encuentra ya en la forma adecuada *(x=g(x))* escribimos la función y la almacenamos en la variable *FX:*

```
« → X « 52 3 X √ * + 8 X .8 ^ * - .36 ^ » » 'GX' STO
```

Ahora asumimos un valor inicial, en nuestro caso 1.1, lo colocamos en la pila y evaluamos la función (pulsando la tecla correspondiente a la función) con lo que se obtiene:

3.98405538779

Que es el valor de la función *(g(x))* para "x" igual a 1.1. Puesto que el valor asumido (1.1) no es igual al calculado *(3.98405538779)*, se vuelve a evaluar la función empleando con el valor calculado y como el mismo está ya

en la pila, simplemente se pulsa la tecla correspondiente a la función *GX*, con lo que se obtiene:

3.55201213202

Que todavía no es igual al valor asumido *(3.98405538779)*, por lo que el proceso se repite (pulsando la tecla de la función *GX)*. De esa manera (pulsando repetidas veces la tecla de la función *GX)* se obtienen los siguientes valores:

3.61847959925
3.60832356549
3.60987692574
3.60963937657
3.60967570487
3.60967014922
3.60967099883
3.60967086891
3.60967088877
3.60967088574
3.6096708862
3.60967088613
3.60967088614
3.60967088614

Donde los dos últimos valores son iguales, por lo que la solución es: *3.60967088614*, que es el mismo resultado obtenido con *ROOT*.

Gráficamente el método de sustitución directa sigue el camino que se muestra en la *figura 3.1*.

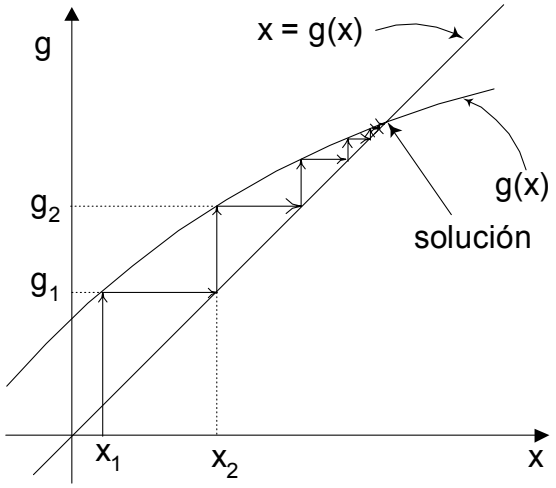


Figura 3.1. Método de Sustitución Directa

3.2.1. Algoritmo

El algoritmo para este método se presenta en el diagrama de actividades de la figura 3.2. Como se puede observar en dicho algoritmo se sigue el proceso descrito, sin embargo, como ningún método numérico puede asegurar la convergencia y este método en particular es inestable, se emplea una variable global para determinar el límite de iteraciones: *max*, que de ser alcanzado provoca la terminación del programa mediante la generación de un error: *"valor inicial erróneo".* El límite de iteraciones se fija por defecto en 50, pues en la mayoría de los problemas se logra convergencia en menos de 30 iteraciones, sin embargo como algunos problemas requieren 100 o más itera-

ciones se deja abierta la posibilidad de cambiar este valor. El proceso iterativo termina cuando los dos últimos valores son iguales en un determinado número de dígitos, para ello se emplea una variable global: *pre*, la cual se fija por defecto en $1x10^{-9}$, es decir que por defecto los dos últimos valores deben ser iguales en los primeros 9 dígitos, sin embargo, como en ocasiones se requiere menos precisión o mayor precisión, se deja abierta la posibilidad de cambiar este valor.

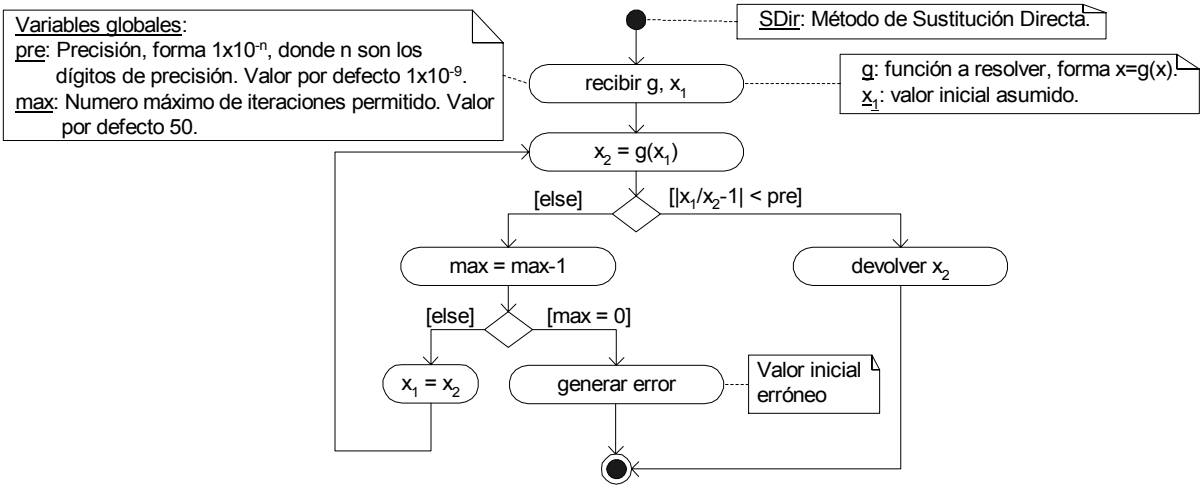


Figura 3.2. Diagrama de Actividades del método de Sustitución Directa

El código de este algoritmo, en *System RPN*, será escrito en el archivo *Formagx*, para ello, en el proyecto *Alge1* cree el archivo *Formagx (Project -> New Source File -> Formagx*) y en el mismo escriba el siguiente código:

```
xNAME Sdir ( Método de Sustitución directa )
:: ( Parámetros: función "g" forma x=g, valor inicial asumido "x1")
  CK2&Dispatch
  # 81
  :: ( g=5; x1=4; pre=3; max=2; x2=1 )
    ' ID pre @ ?SKIP % 1E-9  ( si no existe "pre" se emplea 1E-9 )
    ' ID max @ ?SKIP % 50    ( si no existe "max" se emplea 50 )
    %0
    ' NULLLAM FIVE NDUPN DOBIND
    ::
    BEGIN
      4GETLAM 5GETLAM EVAL 1PUTLAM  ( x2=g[x1] )
      "x:" 1GETLAM a%>$ &$ DispCoord1 ( mostrar: x2 )
      4GETLAM 1GETLAM %/ %1- %ABS 3GETLAM %< ( |x1/x2-1|<pre )
      IT :: 1GETLAM 2RDROP ; ( devolver x2 )
      2GETLAM %1- 2PUTLAM ( max=max-1 )
      2GETLAM %0= ( max==0 )
      IT :: ABND #A01 DO#EXIT ; ( valor inicial erróneo )
      1GETLAM 4PUTLAM ( x1=x2)
    AGAIN
    ;
    ABND
  ;
;
```

Que como vemos está escrito empleando variables locales sin nombre, pues como ya se dijo, *Debug4x* no compila bien cuando un proyecto tiene más de un programa y en esta librería escribiremos todos los programas de este capítulo.

En este programa se intenta recuperar el valor de las variables globales *pre* (' ID pre) y *max* (' ID max) con la función *@*. Esta función devuelve el valor de la variable y el valor lógico *TRUE* si la variable existe, pero si la variable no existe devuelve el valor lógico *FALSO*. Luego se emplea el comando *?SKIP*, el cual salta la siguiente instrucción si en la pila existe el valor lógico *TRUE* y la ejecuta en caso contrario. Por lo tanto, si la variable global existe su valor es colocado en la pila, caso contrario se coloca el valor por defecto.

En este programa se muestran los valores intermedios. Para ello se emplea la función *DispCoord1,* que muestra en la parte inferior de la pantalla el texto que se encuentra en la pila. El texto se arma sumando a "x:" el valor de la variable *1GETLAM* con el operador *&$*. Previamente el valor de la variable es convertido en texto con *a%>$*. El mostrar en pantalla los valores intermedios hace que el programa se ejecute más lentamente, por lo que si se quiere mayor velocidad debe suprimirse esta línea del programa.

Una vez escrito el programa, guarde las modificaciones *(File -> Save All)* y compile el proyecto. Entonces en el emulador se cargará la librería y podrá probar el programa. Para probar el programa resolvamos una vez más la ecuación (3.1), para ello escribimos el siguiente programa:

```
«
  « → X
    « 52 3 X √ * + 8 X .8 ^ * - .36 ^  »
  » 1.1 Sdir
»
```

Y evaluando el programa con *TEVAL*

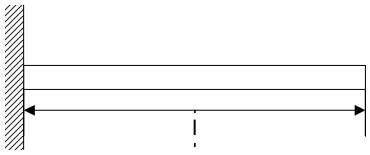
```
3.60967088574
s: 1.1917
```

Que es el resultado calculado con 9 dígitos de precisión y el tiempo empleado en calcularlo. Por supuesto es posible probar el programa escribiendo directamente los datos en la pila. Si no obtiene este resultado haga correr el programa paso a paso para ubicar el error. Una vez probado el programa no olvide guardar la memoria del emulador *(File -> Save).*

3.2.2. Ejemplos

1. La ecuación de frecuencia de una viga sujeta en uno de sus extremos es:

$$cos(kl)cosh(kl) = -1$$



Donde:

$k^2 = p/a$

$a^2 = EIg/(Ay)$

$l = 120\ pulg.$ (longitud de la viga).

$I = 170.6\ pulg^4.$ (momento de inercia del área de la viga)

$E = 3E6\ lb/pulg^2.$ (módulo elástico del material de la viga)

$y = 0.066\ lb/pulg^3.$ (densidad del material de la viga)

A = 32 pulg². (área de la sección transversal de la viga)

g = 386 pulg/seg². (aceleración de la gravedad).

p = frecuencia circular natural de la viga, radianes/seg.

Calcule la frecuencia natural de la viga *(p)* con el método de Sustitución Directa y empleando un valor inicial *(para k)* igual a 0.01.

Solución

Colocamos primero la ecuación en la forma *x=g(x)*, despejando el valor de *k* que se encuentra en el coseno:

$$k = g(k) = \frac{\arccos\left(\dfrac{-1}{\cosh(k * l)}\right)}{l}$$

Entonces escribimos el programa:

```
« 120 170.6 3E6 .066 32 386 0
  → l I E y A g a
  « E I * g * A y * / √ 'a' STO
    « → k
      « 1 NEG k l * COSH / ACOS l / »
    » .01 Sdir
    SQ a *
    "p" →TAG
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
P: 74.6766962649
s: 1.7705
```

Que es el valor buscado y el tiempo empleado en calcularlo.

2. Resuelva el siguiente sistema de ecuaciones no lineales. Emplee como valor inicial 2.1.

$$3x^{2.1} - 5y = 7.0$$
$$y^{1.2} + 4z = 14.3$$
$$x + y^2 + z^2 = 14.0$$

Solución

Primero colocamos una de las ecuaciones del sistema en la forma *x=g(x)*, y colocamos las otras variables en función de la variable independiente:

$$x = g(x) = 14.0 - y^2 - z^2$$
$$y = \frac{3x^{2.1} - 7.0}{5}$$
$$z = \frac{14.3 - y^{1.2}}{4}$$

Aunque la tercera ecuación depende explícitamente de *"y"*, depende implícitamente de *"x"* pues el valor de *"y"* se calcula con el valor de *"x"*. Ahora elaboramos el programa donde resolvemos la función:

```
« 0 0 → y z
  «
    « → x
```

```
      « 3 x 2.1 ^ * 7 - 5 / 'y' STO
        14.3 y 1.2 ^ - 4 / 'z' STO
        14 y SQ - z SQ -
      »
    » 2.1 Sdir
    "x" →TAG
    y "y" →TAG
    z "z" →TAG
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x: (14.,-9.99999999999E499)
y: (-2.E499,-2.E499)
z: (2.5E499,2.5E499)
s: 2.4565
```

Que son soluciones imaginarias e incorrectas. Como ya se dijo, este método es inestable, razón por la cual la convergencia no está asegurada y como en este caso se pueden obtener resultados erróneos.

3.2.3. Ejercicios

1. Resuelva la siguiente función colocándola en la forma *x=g(x)*, programándola y pulsando la tecla de la función hasta que los dos últimos valores sean iguales.

$$f(x) = x^3 + 2x^2 + 3x + 4 = 0$$

2. Resuelva la siguiente función colocándola en la forma *x=g(x)*, programándola y pulsando la tecla de la función hasta que los dos últimos valores sean iguales.

$$f(x) = 2x^2 + 1 - e^x = 0$$

3. Resuelva la ecuación del ejercicio 1 empleando el programa *SDir*, controlando el tiempo empleado en la resolución.

4. Resuelva el ejercicio 2 empleando el programa *SDir*, controlando el tiempo empleado en la resolución.

5. Resuelva el siguiente sistema de ecuaciones no lineales empleando *SDir*.

$$\begin{aligned} 2x^2 + 5xy - 4x &= 115 \\ e^{\frac{x+y}{5}} + x^2 y^2 - 70y &= 15 \end{aligned}$$

6. Basado en el trabajo de Frank y Kamenetski en 1955, las temperaturas en el interior de un material con fuentes de calor internas pueden ser determinada si se resuelve la siguiente ecuación:

$$e^{-(1/2)t\cosh^{-1}(e^{(1/2)t})} = \sqrt{\frac{1}{2}L_{cr}}$$

Para un espesor $L_{cr} = 0.088$, calcule la temperatura en el interior del material empleando el método *SDir*.

3.3. Método de Wegstein

El método de Wegstein, al igual que el método de sustitución directa, requiere que la ecuación algebraica sea colocada en la forma *x=g(x)*.

Este método calcula los dos primeros puntos siguiendo el método de sustitución directa, luego, empleando estos dos puntos, calcula un nuevo valor de prueba (x_3) mediante una extrapolación lineal, tal como se muestra en la la figura 3.3. De esta manera se consigue acelerar la velocidad de convergencia al mismo tiempo que se logra una mayor estabilidad.

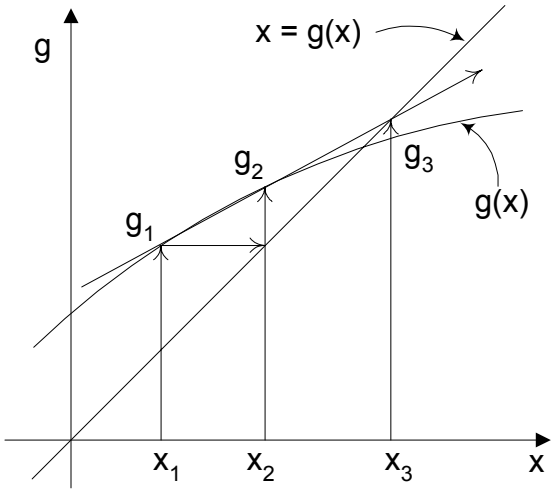


Figura 3.3. Método de Wegstein

Como se puede ver en la figura, el nuevo valor se encuentra en la intersección entre la línea recta que pasa por los puntos (x_1,g_1), (x_2,g_2) y la recta x=g. La ecuación de la línea recta es:

$$g = a + bx \tag{3.3}$$

Por lo tanto la pendiente de la recta que pasa a través de los dos puntos *(b)* es:

$$\left.\begin{aligned} g_1 &= a + bx_1 \\ g_2 &= a + bx_2 \end{aligned}\right\} \quad b = \frac{g_1 - g_2}{x_1 - x_2} \tag{3.4}$$

Entonces el valor de la ordenada en el origen *(a)* es:

$$g_1 = a + \frac{g_1 - g_2}{x_1 - x_2}x_1 \Rightarrow a = g_1 - \frac{g_1 - g_2}{x_1 - x_2}x_1 = \frac{g_1x_1 - g_1x_2 - g_1x_1 + g_2x_1}{x_1 - x_2} \Rightarrow a = \frac{g_2x_1 - g_1x_2}{x_1 - x_2} \tag{3.5}$$

Por lo tanto la ecuación de la línea recta que pasa a través de los dos puntos es:

$$g = \frac{g_2x_1 - g_1x_2}{x_1 - x_2} + \frac{g_1 - g_2}{x_1 - x_2}x \tag{3.6}$$

En la intersección $x=x_3$ y $g=x_3$, por lo tanto:

$$x_3 = \frac{g_2x_1 - g_1x_2}{x_1 - x_2} + \frac{g_1 - g_2}{x_1 - x_2}x_3$$

$$x_3\left(1 - \frac{g_1 - g_2}{x_1 - x_2}\right) = \frac{g_2x_1 - g_1x_2}{x_1 - x_2}$$

$$x_3\left(\frac{x_1-x_2-g_1+g_2}{x_1-x_2}\right)=\frac{g_2x_1-g_1x_2}{x_1-x_2}$$

$$x_3=\frac{g_2x_1-g_1x_2}{g_2-g_1+x_1-x_2} \qquad (3.7)$$

Que es la ecuación del método de *Wegstein*.

Por ejemplo, para resolver la función:

$$f(x)=2x^2+1-e^x=0 \qquad (3.8)$$

La colocamos primero en la forma *x=g(x)*:

$$x=g(x)=\sqrt{\frac{e^x-1}{2}}$$

Programamos entonces esta función y la guardamos en la variable *GX*:

```
« → X
  « X EXP 1 - 2 / √ »
» 'GX' STO
```

Ahora asumimos un valor inicial: $x_1=1$ y calculamos los dos primeros puntos siguiendo el método de *Sustitución Directa*, es decir con el valor asumido calculamos el valor de la función (pulsando la tecla correspondiente a *GX*):

1 [GX] => .926898545813

Como el valor calculado (g_1) no es igual al asumido, se vuelve a evaluar la función empleando este valor como valor de prueba:

[GX] => .873687784815

Una vez más el valor calculado no es igual al asumido, por lo que se debe calcular un nuevo valor. Ahora que tenemos dos puntos: $x_1=1$, $g_1=.926898545813$, $x_2=.926898545813$, $g_2=.873687784815$, calculamos el nuevo valor con la ecuación de *Wegstein (3.7)*:

$$x_3=\frac{0.873687784815*1-0.926898545813*0.926898545813}{0.873687784815-0.926898545813+1-0.926898545813}=0.731340554438$$

Y con este valor se vuelve a evaluar la función:

0.731340554438 [GX] => .734119960078

Como este valor (g_3) no es igual a x_3 se debe calcular un nuevo valor de prueba. Para ello realizamos un intercambio de variables: $x_1=x_2$, $g_1=g_2$, $x_2=x_3$, $g_2=g_3$ y con la ecuación (3.7) calculamos el nuevo valor:

$$x_3=\frac{0.734119960078*0.873687784815-0.873687784815*0.731340554438}{0.734119960078-0.873687784815+0.873687784815-0.731340554438}=0.74104824094$$

Que todavía no es igual al valor supuesto, por lo que proceso debe ser repetido. En tres iteraciones más se obtiene:

x_3	g_3
0.74104824094	0.740990456022
0.740850525205	0.740850497331
0.740850432027	0.740850431369

Los dos últimos valores son iguales en los 8 primeros dígitos, por lo que el proceso puede concluir, siendo el resultado 0.740850431369, con 8 dígitos de precisión. Con el método de sustitución directa, se requiere un total de

62 iteraciones para conseguir el mismo resultado, lo que demuestra el ahorro de tiempo que se logra con el método de *Wesgtein*.

3.3.1. Algoritmo

El algoritmo propuesto para el método de *Wegstein* se presenta en la figura 3.4. En el mismo, antes de aplicar el método propiamente, se comprueba si se obtiene el resultado al calcular los dos primeros puntos y de ser así el resultado es devuelto directamente. Al igual que en el método de sustitución directa se emplean variables globales para controlar el número de iteraciones *(max)* y la precisión del resultado *(pre)*.

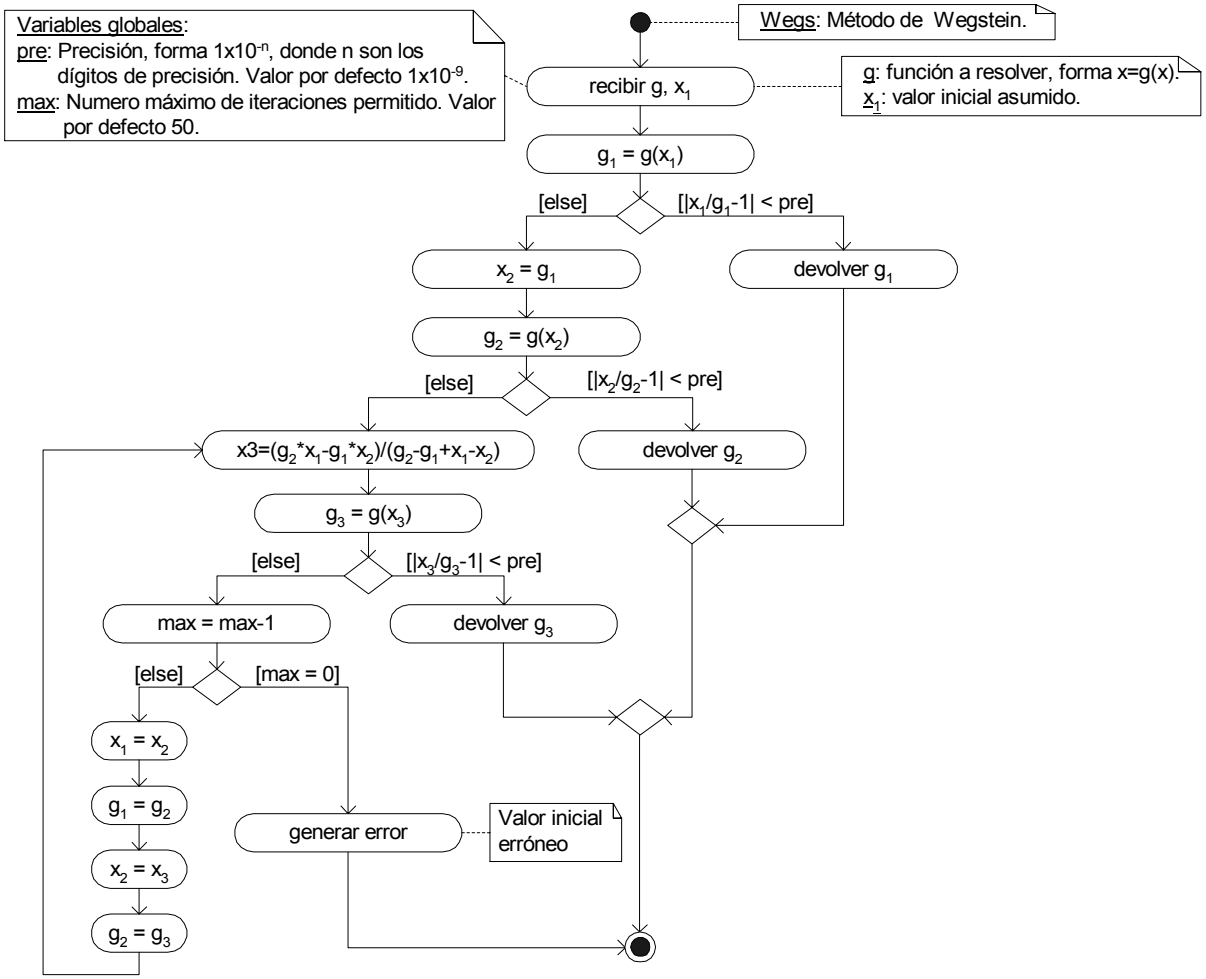


Figura 3.4. Diagrama de actividades para el método de Wegstein

El código de este algoritmo, escrito en *System RPN* y que debe ser añadido al archivo *Formagx*, es el siguiente:

```
xNAME Wegs ( Método de Wegstein )
:: ( Parámetros: función "g" forma x=g, valor inicial asumido "x1" )
  CK2&Dispatch
  # 81
  :: ( g=9; x1=8; pre=7; max=6; x2=5; x3=4; g1=3; g2=2; g3=1 )
    ' ID pre @ ?SKIP % 1E-9 ( si no existe "pre" se emplea 1E-9 )
    ' ID max @ ?SKIP % 50 ( si no existe "max" se emplea 50 )
    %0 %0 %0 %0 %0
    ' NULLLAM NINE NDUPN DOBIND
    ::
```

```
      8GETLAM 9GETLAM EVAL 3PUTLAM ( g1=g[1] )
      8GETLAM 3GETLAM %/ %1- %ABS 7GETLAM %< ( |x1/g1-1|<pre )
      IT :: 3GETLAM RDROP ; ( devolver g1 )
      3GETLAM 5PUTLAM ( x2=g1)
      5GETLAM 9GETLAM EVAL 2PUTLAM ( g2=g[2] )
      5GETLAM 2GETLAM %/ %1- %ABS 7GETLAM %< ( |x2/g2-1|<pre )
      IT :: 2GETLAM RDROP ; ( devolver g2 )
      BEGIN
        2GETLAM 8GETLAM %* 3GETLAM 5GETLAM %* %- 2GETLAM 3GETLAM %-
        8GETLAM %+ 5GETLAM %- %/ 4PUTLAM ( x3=[g2x1-g1x2]/[g2-g1+x1-x2] )
        "x:" 4GETLAM a%>$ &$ DispCoord1 ( mostrar x3 )
        4GETLAM 9GETLAM EVAL 1PUTLAM ( g3=g[x3] )
        4GETLAM 1GETLAM %/ %1- %ABS 7GETLAM %< ( |x3/g3-1|<pre )
        IT :: 1GETLAM 2RDROP ; ( devolver g3 )
        6GETLAM %1- 6PUTLAM ( max=max-1 )
        6GETLAM %0= ( max==0 )
        IT :: ABND #A01 DO#EXIT ; ( valor inicial erróneo )
        5GETLAM 8PUTLAM ( x1=x2)
        2GETLAM 3PUTLAM ( g1=g2)
        4GETLAM 5PUTLAM ( x2=x3)
        1GETLAM 2PUTLAM ( g2=g3)
      AGAIN
    ;
    ABND
  ;
;
```

Para probar este programa resolveremos la ecuación (3.8), mediante el siguiente programa:

```
«
  « → X
    « X EXP 1 - 2 / √ »
  » 1 Wegs
»
```

Haciendo correr este programa se obtiene:

.740850431369

s: .4022

Que es la solución y el tiempo empleado en encontrarla.

3.3.2. Ejemplos

1. Para la trampa que se muestra en la figura de la siguiente página, la relación que existe entre el desplazamiento angular de la mandíbula y el tiempo necesario para lograr dicho desplazamiento está dada por la siguiente ecuación:

$$\theta = \frac{T_0}{k}\left(1-\cos\left(\sqrt{\frac{k}{I_0}}t\right)\right)$$

Donde:

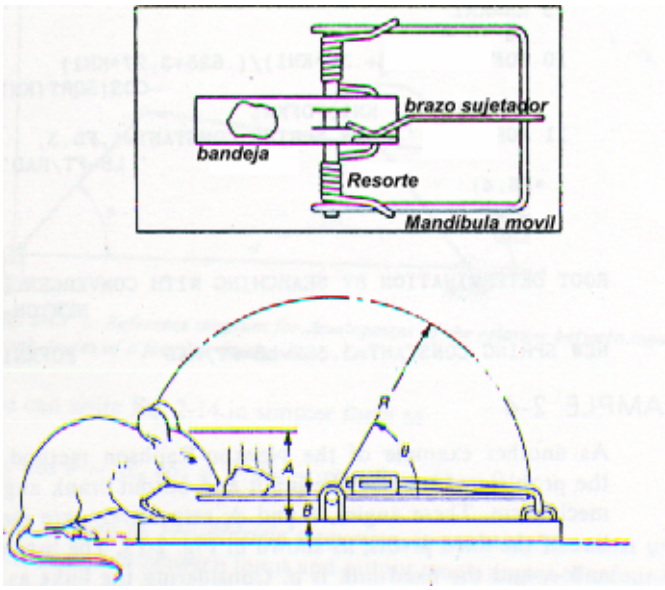
θ = Desplazamiento angular del dispositivo estrangulador.

T_o = Torque ejercido por el resorte sobre la mandíbula $\theta = 0$, (lb.pie).

k = Constante de torsión del resorte, (lb.pie/radianes).

I_o = Momento de inercia de la mandíbula alrededor de del eje de rotación, (lb.pie.seg^2).

t = Tiempo, seg.



Para una trampa con las dimensiones $A=1.125$ pulgadas, $B=0.5$ pulgadas y $R=3.75$ pulgadas se puede determinar que θ_c (desplazamiento angular de la mandíbula hasta tocar a la rata) es igual a 2.97 radianes y θ_k (desplazamiento angular de la mandíbula en su posición cerrada) es igual a 3.97 radianes.

El torque T_o está relacionado con la constante del resorte por la siguiente ecuación:

$$T_o = T_c + \theta_k k$$

Donde T_c es el torque cuando la mandíbula está cerrada.

El fabricante ha recibido quejas porque la trampa cierra muy lentamente (0.0382 segundos para golpear a la rata) permitiendo que muchas ratas escapen después de accionar la trampa. Por consiguiente es necesario instalar un nuevo resorte que golpee a la rata dos veces más rápido que la trampa actual, pero que todavía mantenga su torque en la posición cerrada $(T_c=0.625$ *lb.pie)* puesto que las trampas actuales se arman fácilmente con ese torque. Es también deseable mantener el mismo momento de inercia (0.0006 lb.pie.seg^2) para la mandíbula de manera que se disponga de energía adicional para matar a la rata. Calcule el valor que deberá tener la constante de torsión del nuevo resorte con una precisión de 6 dígitos.

Solución

En la nueva trampa se requiere que el dispositivo recorra los 2.97 radianes $(\theta=\theta_c)$ en la mitad del tiempo que emplea actualmente, es decir en: t = 0.0382/2 = 0.0191 s. Todas las otras constantes son proporcionadas en el planteamiento del problema.

Para el método de Wegstein, colocamos la ecuación en la forma $x=g(x)$, despejando k del denominador del segundo término:

$$k = g(k) = \frac{T_0}{\theta_c}\left(1-\cos\left(\sqrt{\frac{k}{I_0}}t\right)\right)$$

Guardamos la precisión en la variable *pre:*

1E-6 'pre' STO

Y resolvemos el problema con el siguiente programa:

```
« 3.27 2.97 .625 .0191 .0006 0
  → θk θc Tc t Io To
  «
   « → k
    « Tc θk k * + 'To' STO
      To θc / 1 k Io / √ t *
      COS - *
    »
   » 2.1 Wegs
   "k" →TAG
  »
»
```

Evaluando el programa con *TEVAL* se obtiene:

k: 3.36194259817

s: 1.1646

Que es el valor que deberá tener la constante de torsión del nuevo resorte y el tiempo empleado en calcularla.

2. Resuelva el siguiente sistema de ecuaciones no lineales:

$$\begin{aligned} x^{1/2} + z^2 &= 35 \\ y + e^{z/2} - \frac{8}{z} &= 30 \\ w + z^2 - 2z &= 31 \\ x + w + y &= 39 \end{aligned}$$

Solución:

Primero colocamos una de las ecuaciones del sistema en la forma *x=g(x)* y las otras variables las colocamos en función de *"x"*:

$$\begin{aligned} z &= \sqrt{35 - x^{1/2}} \\ y &= 30 + \frac{8}{z} - e^{z/2} \\ w &= 31 - z^2 + 2z \\ x &= g(x) = 39 - w - y \end{aligned}$$

Entonces resolvemos el problema con el siguiente programa:

```
« 0 0 0 → w y z
  «
    « → x
      « 35 x √ - √ 'z' STO
        30 8 z / + z 2 / EXP - 'y' STO
        31 z SQ - 2 z * + 'w' STO
        39 w - y -
      »
    » 1.1 Wegs
    "x" →TAG
    y "y" →TAG
    z "z" →TAG
    w "w" →TAG
  »
```

»

Haciendo correr el programa con *TEVAL* se obtiene:

x: 13.1891885978

y: 14.9776357239

z: 5.6007417143

w: 10.8331756783

s: .7808

Que son las soluciones del sistema y el tiempo empleado en obtenerlas.

3.3.3. Ejercicios

7. Resuelva la siguiente ecuación con 7 dígitos de precisión empleando el método de *Wegstein:*

$$x = \left(52 + 3\sqrt{x} - 8x^{0.8}\right)^{0.36}$$

8. Resuelva la siguiente ecuación con 10 dígitos de precisión empleando el método de *Wegstein:*

$$f(x) = x^3 + 2x^2 + 3x + 4 = 0$$

9. Resuelva la siguiente ecuación, con la precisión estándar, empleando el método de *Wegstein*:

$$f(z) = e^{z/2} - z^2 - 60 = 0$$

10. Resuelva el siguiente sistema de ecuaciones no lineales, con 6 dígitos de precisión, empleando el método de Wegstein:

$$\begin{aligned} 15y + 20x^2 - x^3 &= 1500 \\ 9z - 1.5x^2 + e^{\frac{x}{2}} &= 300 \\ y^2 - z^3 - 5x &= 166.81 \end{aligned}$$

11. La ecuación de Redlich-Kwong:

$$P = \frac{RT}{v-b} - \frac{A}{v(v+b)}$$

Puede ser empleada para calcular el volumen de gases a presiones superiores a la atmosférica. En una prueba experimental se han determinado los siguientes valores para el propano: *P=87.3, T=486.9, v=12.005, A=0.0837* y *R=1.98*. Con estos datos calcule la constante *"b"* de la ecuación de Redlich-Kwong para el propano con 8 dígitos de precisión.

12. Lee y Duffy (1976) relacionaron el factor de fricción para el flujo de una suspensión de partículas fibrosas con el número de Reynolds obteniendo la siguiente ecuación empírica:

$$\frac{1}{\sqrt{f}} = \left(\frac{1}{k}\right)\ln\left(\mathrm{Re}\sqrt{f}\right) + \left(14 - \frac{5.6}{k}\right)$$

En esta relación *f* es el factor de fricción, *Re* es el número de Reynolds y *k* es una constante determinada por la concentración de la suspensión. Para una suspensión con 0.08% de concentración, *k = 0.28*. ¿Cuál es el factor de fricción *f* (con 6 decimales de precisión) si *Re = 37.50*?

3.4. Método Incremental

Estudiaremos ahora el primero de los métodos que permiten resolver ecuaciones algebraicas cuando se encuentra en la forma *f(x) = 0*, es decir cuando la función está igualada a cero (tal como se requiere para el comando ROOT). En este caso la solución se obtiene cuando la función se hace cero (o casi cero), tal como se muestra en la figura 3.5.

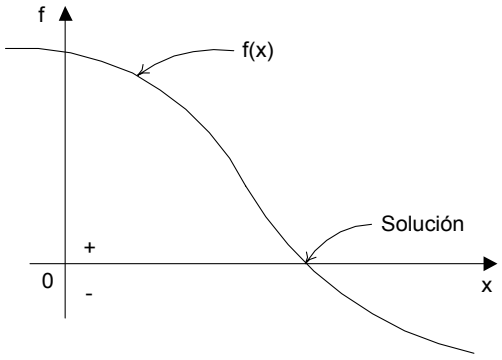


Figura 3.5. Resolución de ecuaciones algebraicas de la forma f(x)=0

El método incremental es, conceptualmente, el método más sencillo que existe para resolver ecuaciones que se encuentran en la forma *f(x)=0*. Se basa en el hecho de que a ambos lados de la solución la función cambia de signo (como se puede ver en la figura 3.5). Básicamente este método busca el lugar donde ocurre dicho cambio.

Para ello comenzando con un valor inicial asumido (x_1) y un incremento (Δx) se van calculando valores sucesivos de x ($x_{i+1}=x_i+\Delta x$) hasta que el valor de la función cambia de signo (como se muestra en la figura 3.6).

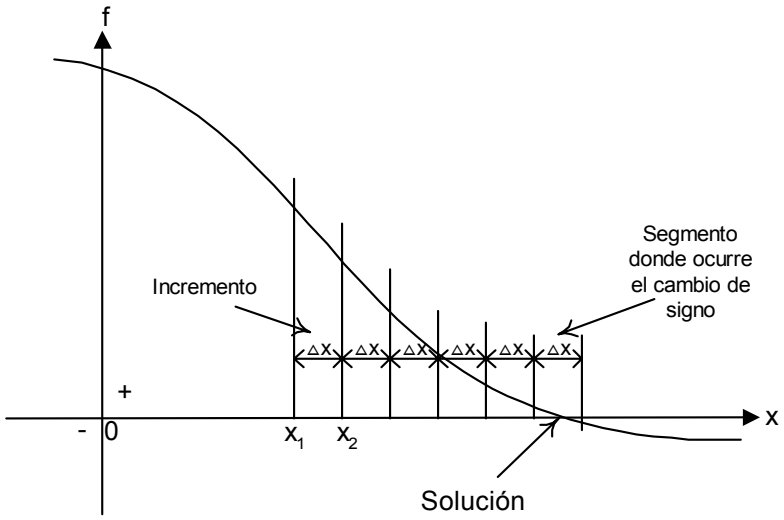


Figura 3.6. Representación gráfica del método Incremental

Una vez encontrado el segmento donde ocurre el cambio, se puede volver a repetir el método pero con un incremento más pequeño, usualmente la décima parte del incremento original, lo que permite determinar el lugar donde se encuentra la solución con mayor exactitud. Este proceso se puede repetir hasta que el segmento donde ocurre el cambio es tan pequeño que prácticamente es la solución buscada. No obstante ello requiere demasiadas repeticiones y es consecuentemente muy lento, razón por la cual normalmente el método

sólo se emplea para encontrar el segmento de solución y se recurre a otros métodos, que son más rápidos, para encontrar la solución.

Como el determinar el segmento de solución constituye la aplicación más importante del método Incremental, es conveniente contar con un programa que lleve a cabo dicha tarea.

Antes de elaborar el algoritmo encontremos el segmento de solución para la ecuación:

$$f(x) = \left(52 + 3\sqrt{x} - 8x^{0.8}\right)^{0.36} - x = 0 \qquad (3.9)$$

Que se encuentra ya en la forma *f(x)=0*. Para ello programe la función y guarde el programa en la variable *Fx:*

*« → X « 52 3 X √ * + 8 X .8 ^ * - .36 ^ X - » » 'Fx' STO*

Entonces comencemos la búsqueda con *x=0*, empleando un incremento igual a 1. Evaluando la función para *x=0 (0 [Fx])* se obtiene*:*

4.14725943376

Ahora incrementamos el valor de *x* en 1: *0+1=1* y evaluamos la función para este valor *(1 [Fx])*:

2.99903521835

Como no existe cambio de signo continuamos la búsqueda incrementando el valor de *x* en 1: 1+1=2 *(2 [Fx])*:

1.85064646856

Prosiguiendo de esa manera para x=2+1=3 se obtiene:

.70199038475

Y para x=3+1=4:

-.45046246722

Entonces como ha ocurrido un cambio de signo se sabe que la solución se encuentra entre *x=3* y *x=4*, de esa manera queda determinado el segmento donde se encuentra la solución.

3.4.1. Algoritmo: Ubicación del segmento

El algoritmo para este método se presenta en el diagrama de actividades de la siguiente página (*figura 3.7*). Al igual que en los otros métodos, el error permitido por defecto *(err)* es $1x10^{-10}$ y el máximo número de iteraciones *(max)* 50. En este método se requiere un límite de iteraciones por siguientes razones: a) Con el valor e incremento iniciales el método puede alejarse de la solución (divergencia), b) El incremento puede ser muy pequeño y requerir demasiadas iteraciones para alcanzar el segmento de solución y c) El incremento puede ser demasiado grande y saltar a través de dos soluciones (dos cambios de signo).

Puesto que el método se emplea en la práctica como primera etapa para encontrar la solución, este algoritmo devuelve la solución y el valor lógico *verdadero*, cuando la solución ha sido encontrada, por el contrario si sólo se encuentra el segmento de solución, devuelve el límite inferior del segmento y valor lógico *falso*. Cuando no se encuentra ni la solución ni el segmento de solución se genera un error.

El código de este algoritmo, escrito en *System RPN* y que debe ser añadido en un nuevo archivo: *Formafx.S (Project -> New Source File -> Formafx)* es el siguiente:

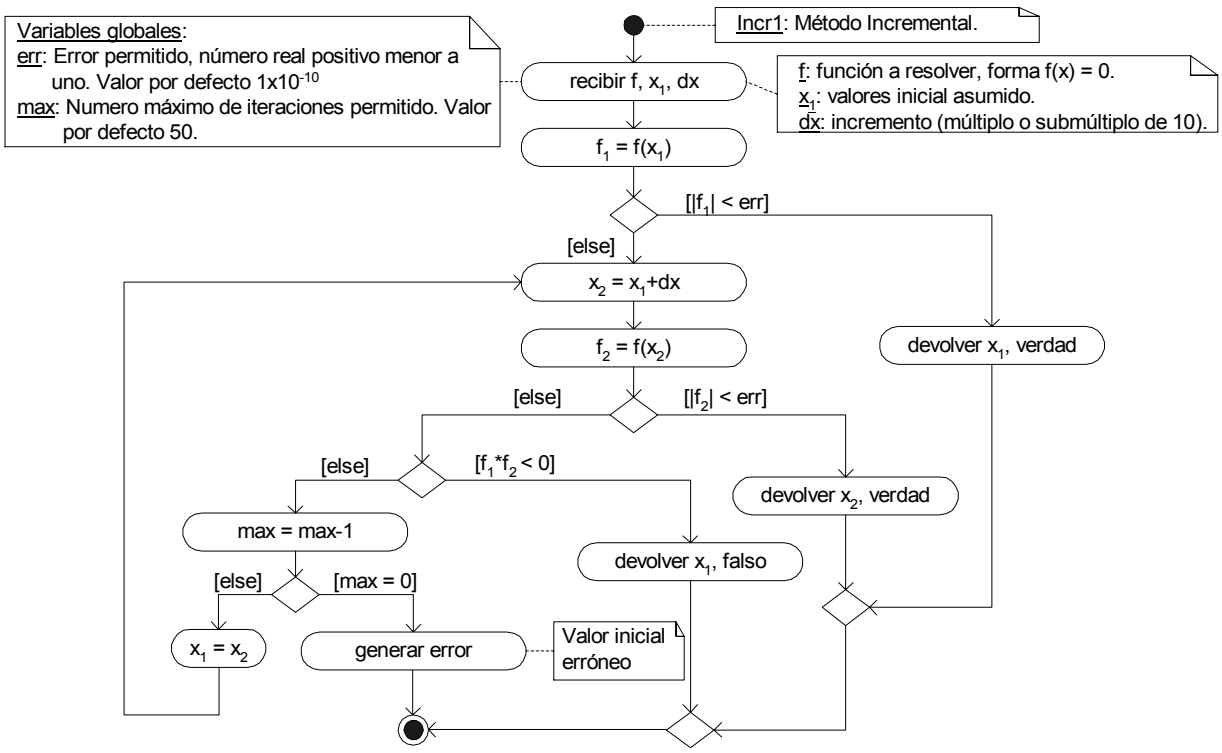


Figura 3.7. Método incremental: Determinación de segmento de solución.

```
xNAME Incr1 ( Método Incremental: ubicación del segmento )
:: ( Parámetros: función "f" forma f=0; valor inicial asumido "x1";
    incremento "dx" )
 CK3&Dispatch
 # 811
 :: ( f=8; x1=7; dx=6; err=5; max=4; x2=3; f1=2; f2=1 )
   ' ID err @ ?SKIP % 1E-9 ( si no existe "pre" se emplea 1E-9 )
   ' ID max @ ?SKIP % 50 ( si no existe "max" se emplea 50 )
   %0 %0 %0
   ' NULLLAM EIGHT NDUPN DOBIND
   ::
     7GETLAM 8GETLAM EVAL 2PUTLAM ( f1=f[x1] )
     2GETLAM %ABS 5GETLAM %< ( |f1|<err )
     IT :: 7GETLAM %1 RDROP ; ( devolver x1, 1=verdad )
     BEGIN
       7GETLAM 6GETLAM %+ 3PUTLAM ( x2=x1+dx )
       "x2:" 3GETLAM a%>$ &$ DispCoord1 ( mostrar x2 )
       3GETLAM 8GETLAM EVAL  1PUTLAM ( f2=f[x2] )
       1GETLAM %ABS 5GETLAM %< ( |f2|<err )
       IT :: 3GETLAM %1 2RDROP ; ( devolver x2, 1=verdad )
       2GETLAM 1GETLAM %* %0< ( |f1*f2|<0 )
       IT :: 7GETLAM %0 2RDROP ; ( devolver x1, 0=falso )
       4GETLAM %1- 4PUTLAM ( max=max-1)
       4GETLAM %0= ( max==0 )
       IT :: ABND #A01 DO#EXIT ; ( valor inicial erróneo )
       3GETLAM 7PUTLAM ( x1=x2 )
     AGAIN
   ;
   ABND
 ;
;
```

Para probar el programa resolveremos la ecuación (3.9) empleando el programa:

```
«
  « → X « 52 3 X √ * + 8 X .8 ^ * - .36 ^ X - » »
  0 1 « Incr1 » TEVAL
»
```

Evaluando este programa se obtiene:

```
3.
0.
s: .4139
```

Que son el valor inicial del segmento *(x_1=3)*, el valor lógico *falso* y el tiempo empleado en encontrar el segmento. Como el valor lógico es falso *(0)* significa que no se ha encontrado la solución, sino el lugar donde se encuentra la solución, el cual comienza en 3, siendo el otro límite ese valor más el incremento empleado, es decir 3+1=4, que por supuesto es el mismo segmento determinado manualmente.

3.4.2. Ejercicios

13. Programe la siguiente ecuación y comenzando con x=-10 y empleando un incremento igual a 1, encuentra el segmento de solución de la ecuación:

$$f(x) = x^3 + 2x^2 + 3x + 4 = 0$$

14. Resuelva el anterior ejercicio empleando el programa *Incr1*.

3.4.3. Búsqueda de la solución

Como ya se dijo anteriormente, una vez determinado el segmento donde se encuentra la solución, se puede volver a aplicar el método pero empleando un incremento igual a la décima parte del incremento original. Entonces, como se puede observar en la siguiente figura, se vuelve a determinar el segmento donde se encuentra la solución pero con una precisión 10 veces superior a la del segmento original. El proceso se puede repetir en el nuevo segmento (empleando un incremento igual a la décima parte del incremento anterior) y continuar así hasta que el resultado tenga la exactitud deseada.

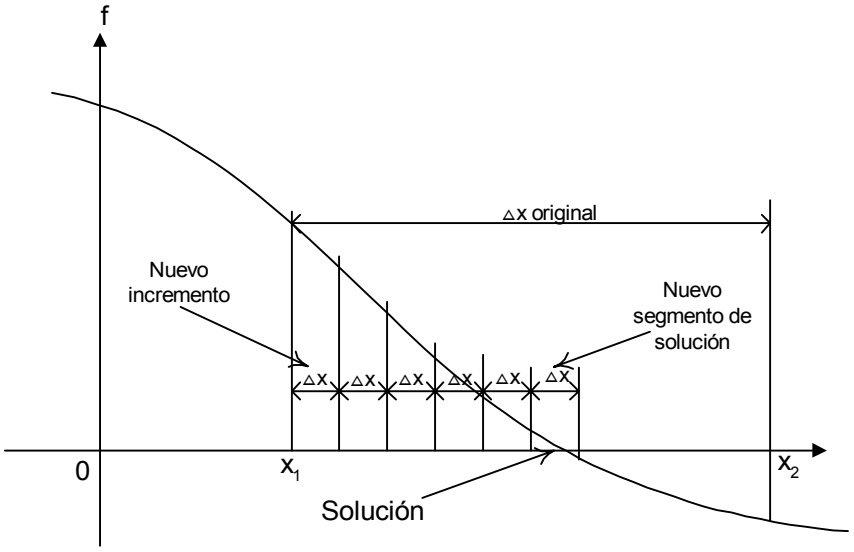


Figura 3.8. Cálculo de la solución con el método Incremental.

El proceso concluye cuando el segmento es lo suficientemente pequeño como para que cualquiera de sus límites pueda ser considerado una solución o cuando el valor de la función es casi cero. Como ya se señaló previamente, el principal inconveniente de este método es el excesivo número de veces que debe repetirse el proceso para lograr un resultado con una precisión aceptable.

Por ejemplo, podemos continuar buscando la solución de la ecuación (3.9). En la primera aplicación del método encontramos la solución se encontraba entre *x=3* y *x=4*. Ahora podemos continuar la búsqueda empleando como valor inicial *x=3* y un incremento igual a la décima parte del incremento original: *1/10 = 0.1*. Evaluando la función *Fx*, para valores sucesivos de *x*, hasta que ocurre un cambio de signo se obtiene:

x	f(x)
3.0	0.70199038475
3.1	0.58696228895
3.2	0.47189291511
3.3	0.35677957850
3.4	0.24161964076
3.5	0.12641050164
3.6	0.01114959139
3.7	-0.10416563623

Ahora sabemos que la solución se encuentra entre *3.6* y *3.7*. Volviendo a repetir el proceso, comenzando con *x=3.60* y empleando un incremento igual a la décima parte del incremento anterior: *0.1/10=0.01:*

x	f(x)
3.60	0.01114959139
3.61	-0.00037944525

Por lo tanto la solución está entre 3.60 y 3.61. Repitiendo el proceso con un incremento igual a la décima parte del valor anterior: *0.01/10=0.001:*

x	f(x)
3.600	0.01114959139
3.601	0.00999671221
3.602	0.00884382760
3.603	0.00769093755
3.604	0.00653804206
3.605	0.00538514113
3.606	0.00423223475
3.607	0.00307932293
3.608	0.00192640566
3.609	0.00077378293
3.610	-0.00037944525

Entonces la solución se encuentra entre *3.609* y *3.610*. Repitiendo el proceso con un incremento igual a la décima parte del incremento anterior: *0.001/10=0.0001*:

x	f(x)
3.6090	0.00077378293
3.6092	0.00054289774
3.6093	0.00042760505
3.6094	0.00031231232
3.6095	0.00019701952
3.6096	0.00008172668
3.6097	-0.00003356622

Ahora sabemos que la solución se encuentra entre *3.6096* y *3.6097*. Repitiendo el proceso con un incremento *0.0001/10=0.00001*:

x	f(x)
3.60960	0.00008172668
3.60961	0.00007019740
3.60962	0.00005866810
3.60963	0.00004713882
3.60964	0.00003560953
3.60965	0.00002408023
3.60966	0.00001255095
3.60967	0.00000102166
3.60968	-0.00001050764

Entonces la solución se encuentra entre 3.60967 y 3.60968 y el valor de la función es menor $1x10^{-5}$, es decir que cualquiera de estos dos valores puede ser considerado el resultado con una precisión de 5 dígitos. El proceso puede continuar hasta la precisión deseada, pero como se puede observar se requiere un gran número de repeticiones.

3.4.4. Algoritmo: Búsqueda de la solución

El algoritmo para encontrar la solución de una ecuación algebraica por el método Incremental se presenta en la siguiente figura:

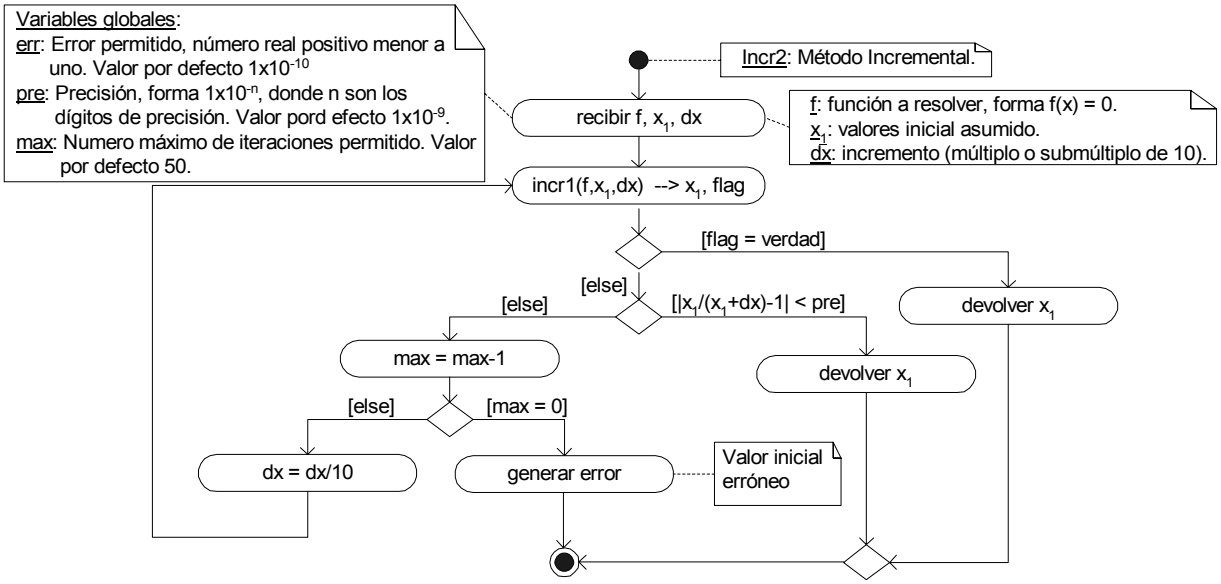


Figura 3.9. Método Incremental: Cálculo de la solución

Como se puede observar en el mismo esencialmente se hacen llamadas al módulo que determina el segmento de solución *(Incr1)*. El código de este algoritmo, que debe ser escrito en el archivo *Formafx.S*, es el siguiente:

```
xNAME Incr2 ( Método incremental: búsqueda de la solución )
:: ( Parámetros: función "f" forma f=0; valor inicial asumido "x1";
    incremento "dx")
 CK3&Dispatch
 # 811
 :: ( f=6; x1=5; dx=4; err=3; pre=2; max=1 )
   ' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
   ' ID pre @ ?SKIP % 1E-9  ( si no existe "pre" se emplea 1E-9 )
   ' ID max @ ?SKIP % 50    ( si no existe "max" se emplea 50 )
```

```
   ' NULLLAM SIX NDUPN DOBIND
   ::
     BEGIN
       6GETLAM 5GETLAM 4GETLAM xIncr1 ( Incr1[f,x1,dx]-->x1,flag )
       %1 %= IT :: 2RDROP ; ( devolver x1 )
       5PUTLAM ( nuevo valor de x1 )
       "x:" 5GETLAM a%>$ &$ DispCoord1 ( mostrar x1 )
       5GETLAM 5GETLAM 4GETLAM %+ %/ %1- %ABS
       2GETLAM %<   ( |x1/[x1+dx]-1|<pre )
       IT :: 5GETLAM 2RDROP ; ( devolver x1 )
       1GETLAM %1- 1PUTLAM ( max=max-1)
       1GETLAM %0= ( max==0 )
       IT :: ABND #A01 DO#EXIT ; ( valor inicial erróneo )
       4GETLAM %10 %/ 4PUTLAM ( dx=dx/10 )
     AGAIN
    ;
    ABND
 ;
;
```

Para probar el programa encontremos la solución de la ecuación (3.9)

```
«
  « → X « 52 3 X √ * + 8 X .8 ^ * - .36 ^ X - » »
  0 1 « Incr2 » TEVAL
»
```

Evaluando este programa se obtiene:

3.609670886

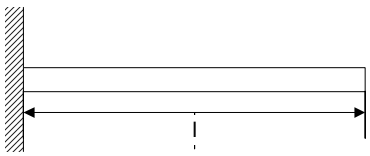
s: 7.2195

Que son la solución y el tiempo empleado en obtenerla. Como se puede observar este es el método más lento de todos los estudiados hasta ahora pues en este caso por ejemplo requiere alrededor de 7 segundos para encontrar la solución.

3.4.5. Ejemplos

3.4.5.1. Frecuencia natural de una viga

Como primer ejemplo volveremos a resolver el primer ejemplo del método de Sustitución Directa:



$$cos(kl)cosh(kl) = -1$$

Donde:

$k^2 = p/a$

$a^2 = EIg/(Ay)$

$l = 120\ pulg.$ (longitud de la viga).

$I = 170.6\ pulg^4.$ (momento de inercia del área de la viga)

$E = 3E6\ lb/pulg^2.$ (módulo elástico del material de la viga)

y = 0.066 lb/pulg³. (densidad del material de la viga)

A = 32 pulg². (área de la sección transversal de la viga)

g = 386 pulg/seg². (aceleración de la gravedad).

p = frecuencia circular natural de la viga, radianes/seg.

El problema consiste en calcula la frecuencia natura de la viga *(p)* con una precisión de 7 dígitos.

Primero colocamos la ecuación en la forma *f(x)=0:*

$$f(k) = cos(kl)cosh(kl) + 1 = 0$$

```
« 120 170.6 3E6 .066 32 386 0
  → l I E y A g a
  « 1E-7 'pre' STO
    E I * g * A y * / √ 'a' STO
    « → k
      « k l * COS k l * COSH * 1 + »
    »
    0 0.01 Incr2
    SQ a *
    "p" →TAG
    'pre' PURGE
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

p: 74.6766940035

s: 5.5608

Que es la frecuencia de la viga y el tiempo empleado en calcularla.

3.4.5.2. Resolución de un sistema de tres ecuaciones no lineales

Como segundo ejemplo resolveremos el sistema de tres ecuaciones resuelto con el método de *Sustitución Directa*, en este caso con 6 dígitos de precisión.

$$3x^{2.1} - 5y = 7.0$$
$$y^{1.2} + 4z = 14.3$$
$$x + y^2 + z^2 = 14.0$$

Primero colocamos el sistema de ecuaciones en la forma *f(x)=0*:

$$f(x) = x + y^2 + z^2 - 14 = 0$$
$$y = \frac{3x^{2.1} + 7.0}{5}$$
$$z = \frac{14.3 - y^{1.2}}{4}$$

Y escribimos el programa:

```
« 0 0 → y z
  « 1E-6 'pre' STO
    « → x
      « 3 x 2.1 ^ * 7 + 5 / 'y' STO
```

```
        14.3 y 1.2 ^ - 4 / 'z' STO
        x y SQ + z SQ + 14 -
      »
    »
    0 1 Incr2
    "x" →TAG
    y "y" →TAG
    z "z" →TAG
  'pre' PURGE
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

x: .9988616

y: 1.9985666399

z: 3.0011447366

s: 8.3345

Que son las tres soluciones y el tiempo empleado en obtenerlas.

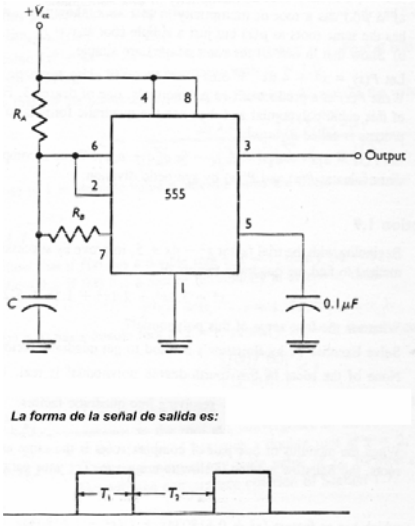
3.4.6. Ejercicios

15. Basado en el trabajo de Frank y Kamenetski en 1955, las temperaturas en el interior de un material con fuentes de calor internas pueden ser determinada si se resuelve la siguiente ecuación:

$$e^{-(1/2)t\cosh^{-1}(e^{(1/2)t})} = \sqrt{\frac{1}{2}L_{cr}}$$

Para un espesor L_{cr} = 0.088, calcule la temperatura en el interior del material con 5 dígitos de precisión.

16. En el circuito que se muestra en la figura:



Se cumple la siguiente relación:

$$T_2 = \frac{R_A R_B C}{R_A + R_B} * \ln\left(\left[\frac{R_A - 2R_B}{2R_A - R_B}\right]\right)$$

Para un circuito específico se tienen los siguientes valores: R_A = *8670*, $C = 0.01x10^{-6}$, $T_2 = 1.4x10^{-4}$. Calcule el valor de la resistencia R_B con 7 dígitos de exactitud.

3.5. Método de la Bisección (Boltzano)

Según vimos en el anterior acápite, una vez determinado el segmento donde se encuentra la solución, no es eficiente continuar empleando el método Incremental para encontrar el resultado final. En lugar de ello se recurre a otros métodos que pueden encontrar la solución en un menor número de iteraciones, uno de dichos métodos es el de la Bisección.

Este método, como su nombre sugiere, divide el segmento de búsqueda en 2 y calcula el valor de la función en la mitad del segmento, entonces si el valor de la función es cero (o casi cero) el proceso concluye, caso contrario se vuelve a repetir el procedimiento pero trabajando ahora con la mitad del segmento donde se encuentra la solución, tal como se puede ver en la figura (figura 3.10).

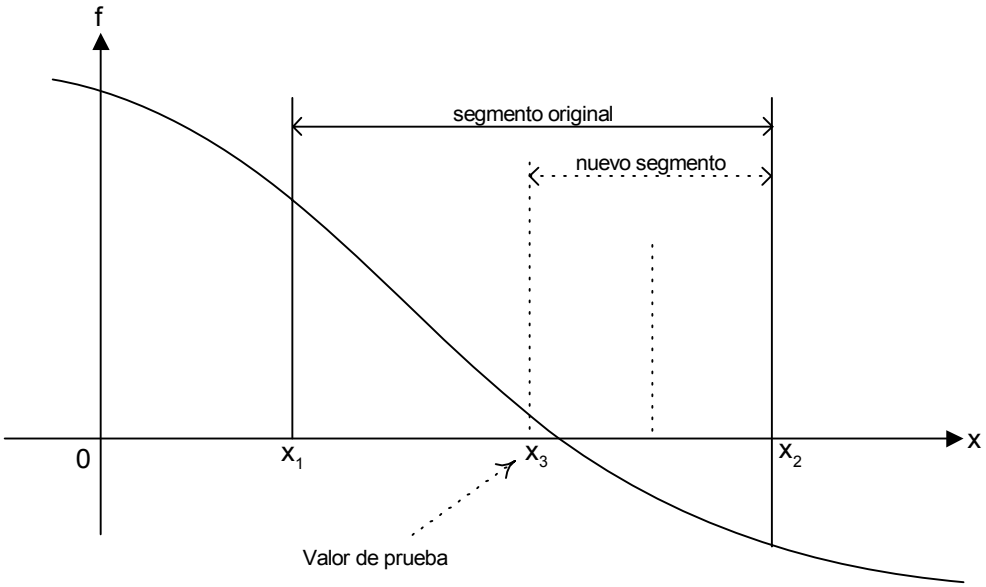


Figura 3.10. Método de la Bisección.

La parte donde se encuentra la solución se determina comparando el valor de la función al inicio del segmento con el valor de la función en el centro del segmento. Si existe cambio de signo (es decir si la multiplicación de estos valores es negativa), entonces la solución se encuentra en el segmento izquierdo, caso contrario se encuentra en el segmento derecho.

La ecuación que permite calcular el valor de la variable independiente (x) a la mitad del segmento es:

$$x_3 = \frac{x_2 - x_1}{2} + x_1 = \frac{x_2 - x_1 + 2x_1}{2} = \frac{x_1 + x_2}{2} \qquad (3.10)$$

Aplicando el método calculemos la solución de la ecuación:

$$f(x) = x^3 + 2x^2 + 3x + 4 = 0 \qquad (3.11)$$

Que se encuentra ya en la forma adecuada *(f(x)=0)*, por lo que podemos programarla directamente y almacenarla en la variable *Fx:*

*« → X « X 3 ^ 2 X SQ * + 3 X * + 4 + » » 'Fx' STO*

Primero ubicamos el segmento de solución con el método incremental:

'Fx' RCL -10 1 Incr1

Con lo que obtenemos:

-2

0

Lo que nos indica que la solución no ha sido encontrada *(0=Falso)* pero que el segmento de solución se encuentre entre *-2* y -2+1 = *-1*. Ahora que conocemos el segmento de solución podemos calcular el nuevo valor de prueba con la ecuación (3.10):

$$x_3 = \frac{x_1 + x_2}{2} = \frac{(-2)+(-1)}{2} = -1.5$$

Y evaluando la función para este nuevo valor se obtiene (-1.5 Fx):

0.625

Puesto que el valor de la función no es cero o cercano a cero repetimos el proceso, pero antes determinamos el lugar donde se encuentra la solución, multiplicando el valor de la función en el límite inicial *($f(x_1) = -2$)*, por el valor de la función en el punto medio *(0.625)*: -2*0.625 = -1.25. Puesto que el resultado es negativo, sabemos que la solución se encuentra en el segmento izquierdo, es decir entre *-2 y -1.5*. Por lo tanto el nuevo valor de prueba es:

$$x_3 = \frac{x_1 + x_2}{2} = \frac{(-2)+(-1.5)}{2} = -1.75$$

Evaluando la función con este valor *(-1.75 Fx)* se obtiene:

-.484375

Que tampoco es cero, por lo que el proceso debe ser repetido. Los valores obtenidos en iteraciones subsecuentes se muestran en la siguiente tabla:

x_1	x_2	x_3	$f(x_3)$
-1.75	-1.5	-1.625	0.115234375
-1.75	-1.625	-1.6875	-0.17260742187
-1.6875	-1.625	-1.65625	-0.02578735352
-1.65625	-1.625	-1.640625	0.04543685912
-1.65625	-1.640625	-1.6484375	0.01000452042
-1.65625	-1.6484375	-1.65234375	-0.00784629584
-1.65234375	-1.6484375	-1.650390625	0.00109037012
-1.65234375	-1.650390625	-1.6513671875	-0.00337514561
-1.6513671875	-1.650390625	-1.65087890625	-0.00114168378
-1.65087890625	-1.650390625	-1.65063476562	-0.00002548087
-1.65063476562	-1.650390625	-1.65051269531	0.00053248861
-1.65063476562	-1.6505126931	-1.65057373046	0.00025351488
-1.65063476562	-1.65057373046	-1.65060424804	0.00011401977
-1.65063476562	-1.65060424804	-1.65061950683	0.00004427014
-1.65063476562	-1.65061950683	-1.65062713622	0.00000939484

El último valor de la función puede considerarse ya cercano a cero, es menor a $1x10^{-5}$, por lo que el proceso puede concluir en este punto siendo el resultado aproximado: -1.65062713622. Como se ha visto en este ejemplo, el valor de la función en el punto inicial del segmento *(f_1)* sólo necesita ser

calculado una vez, pues sólo sirve para verificar si existe o no un cambio de signo.

Aunque este método reduce el número de iteraciones, todavía requiere muchas iteraciones por lo que no es el método más eficiente de los que estudiaremos para la forma *f(x)=0*.

3.5.1. Algoritmo

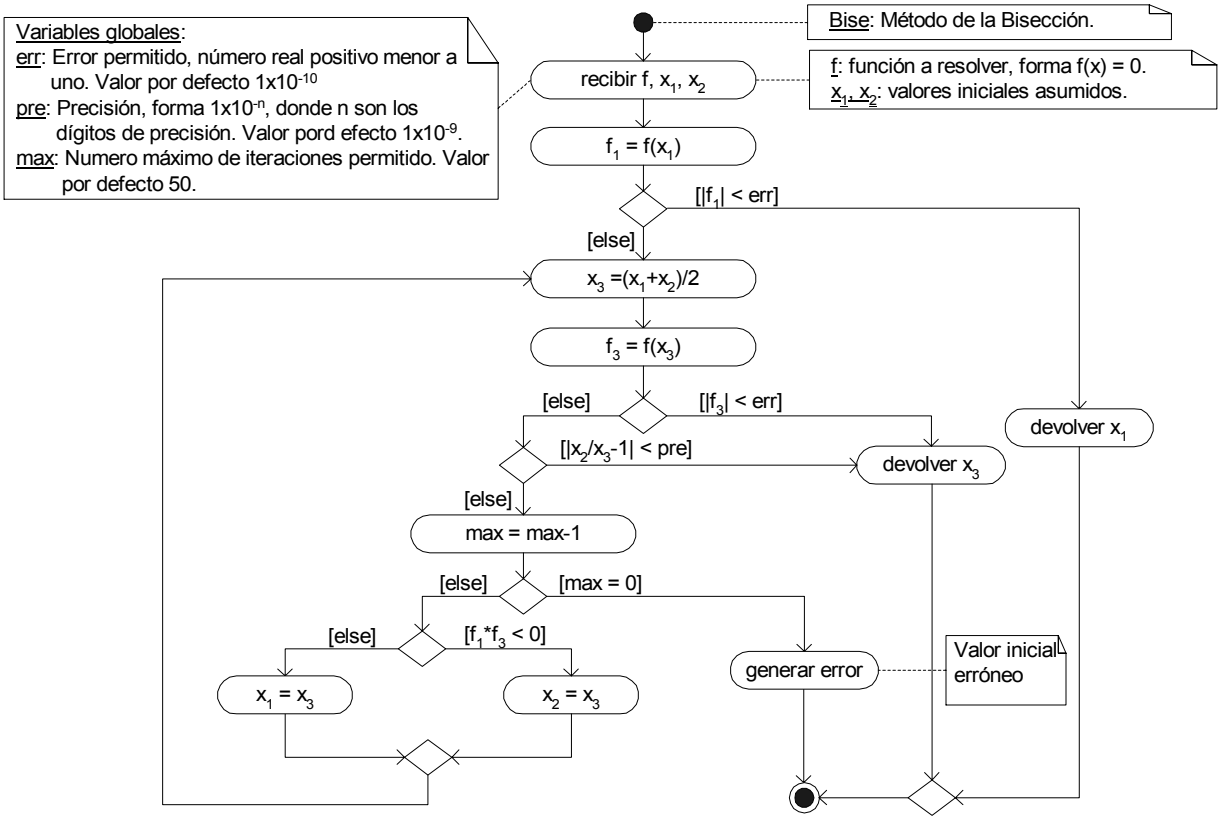


Figura 3.11. Método de la Bisección.

3.5.2. Código

Debe ser escrito en el archivo Formafx.S.

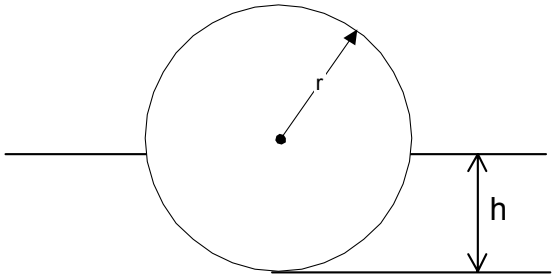
```
xNAME Bise ( Método de la Bisección )
:: ( Parámetros: función "f" forma f=0,
     límites del segmento: "x1", "x2", )
  CK3&Dispatch
  # 811
  :: ( f=9; x1=8; x2=7; err=6; pre=5; max=4; f1=3; f3=2; x3=1 )
   ' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
   ' ID pre @ ?SKIP % 1E-9  ( si no existe "pre" se emplea 1E-9 )
   ' ID max @ ?SKIP % 50    ( si no existe "max" se emplea 50 )
   %0 %0 %0
   ' NULLLAM NINE NDUPN DOBIND
   ::
     8GETLAM 9GETLAM EVAL 3PUTLAM ( f1=f[x1] )
     3GETLAM %ABS 6GETLAM %< ( |f1|<err )
     IT :: 8GETLAM RDROP ; ( devolver x1 )
     BEGIN
```

```
      8GETLAM 7GETLAM %+ %2 %/ 1PUTLAM ( x3=[x1+x2]/2 )
      "x:" 1GETLAM a%>$ &$ DispCoord1 ( mostrar x3 )
      1GETLAM 9GETLAM EVAL 2PUTLAM ( f3=f[x3] )
      2GETLAM %ABS 6GETLAM %< ( |f3|<err )
      IT :: 1GETLAM 2RDROP ; ( devolver x3 )
      7GETLAM 1GETLAM %/ %1- %ABS 5GETLAM %< ( |x2/x3-1|<pre )
      IT :: 1GETLAM 2RDROP ; ( devolver x3 )
      4GETLAM %1- 4PUTLAM ( max=max-1 )
      4GETLAM %0= ( max==0 )
      IT :: ABND #A01 DO#EXIT ; ( valor inicial erróneo )
      1GETLAM 3GETLAM 2GETLAM %* %0< ( x3, f1*f3<0 )
      ITE 7PUTLAM 8PUTLAM ( x2=x3 else x1=x3 )
    AGAIN
  ;
  ABND
 ;
;
```

3.5.3. Ejemplos

3.5.3.1. Profundidad de una esfera sumergida

Una esfera de densidad *d* igual a 0.6 *se* encuentra parcialmente sumergida en agua, como se muestra en la figura. El peso de la esfera de radio *r* es $4/3(\pi r^3 d)$ y el volumen del segmento sumergido es $\pi /3(3rh^2-h^3)$. Encuentre, como una fracción de su radio, la profundidad a la cual se sumergirá la esfera en el agua.



Volumen de la esfera:

$$\frac{4}{3}\pi r^3 d = \frac{\pi}{3}\left(3rh^2 - h^3\right)$$

Si $h= xr$, donde x es la fracción de r a la cual se sumerge la esfera.

$$\frac{4}{3}\pi r^3 d = \frac{\pi}{3}\left(3rr^2x^2 - r^3x^3\right) = \frac{\pi}{3}r^3\left(3x^2 - x^3\right)$$

$$x^3 - 3x^2 + 4d = 0$$

Que se encuentra en la forma f(x)=0, por lo que puede ser resuelta:

```
« .6 → d
  «
    « → x
      « x 3 ^ 3 x SQ * - 4 d * + »
    »
    DUP .1 1 Incr1
    IF 1 ≠ THEN
      DUP 1 + Bise
```

```
    END
    "x" →TAG
  »
»
```

Evaluando esta función con *TEVAL* se obtiene

x: 1.13413784596

s: 2.9335

3.5.3.2. Constante de torsión de una trampa para ratones

Como segundo ejemplo volveremos a resolver el primer problema resuelto con el método de *Wegstein*. En este caso la constante del resorte *(k)* será determinada con 8 dígitos de precisión.

$$\theta = \frac{T_0}{k}\left(1-\cos\left(\sqrt{\frac{k}{I_0}}t\right)\right)$$

$$T_o = T_c + \theta_k k$$

Donde:

θ = Desplazamiento angular del dispositivo estrangulador.

T_o = Torque ejercido por el resorte sobre la mandíbula $\theta = 0$, (lb.pie).

k = Constante de torsión del resorte, (lb.pie/radianes).

I_o = Momento de inercia de la mandíbula alrededor de del eje de rotación, (lb.pie.seg^2).

t = Tiempo, seg.

$A=1.125$ pulgadas, $B=0.5$ pulgadas, $R=3.75$, θ_c=2.97, θ_k=3.27 radianes, $T_c=0.625$ lb.pie, $t=0.0191$ s, Io=0.0006 lb.pie.s^2.

Primero colocamos la ecuación en la forma $f(x)=0$:

$$f(k) = \theta_c - \frac{T_0}{k}\left(1-\cos\left(\sqrt{\frac{k}{I_0}}t\right)\right) = 0$$

Y entonces elaboramos el programa que la resuelve:

```
« 3.27 2.97 .625 .0191 .0006 0
  → θk θc Tc t Io To
  « 1E-8 'pre' STO
    « → k
     « Tc θk k * + 'To' STO
       θc To k / 1 k Io / √ t * COS - * -
     »
    »
    DUP 0 1 Incr2
    IF 1 ≠ THEN
      DUP 1 + Bise
    END
    "k" →TAG
    'pre' PURGE
  »
»
```

Evaluando el programa con *TEVAL* se obtiene:

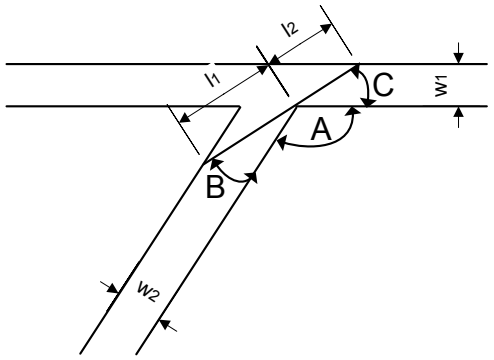
k: 3.3619498074

s: 3.5938

Que es el resultado buscado y el tiempo empleado en obtenerlo.

3.5.4. Ejercicios

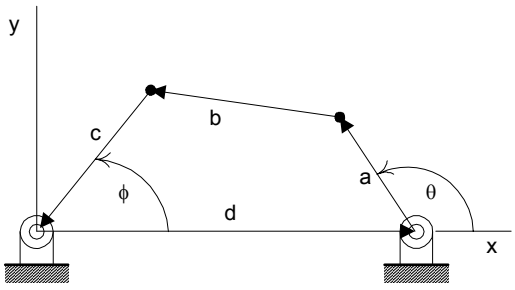
17. En una mina existen dos túneles que se unen en un ángulo de A, tal como se muestra en la figura. Para realizar trabajos en el interior de la mina se requiere introducir escaleras y mientras más largas sean estas mejor. Por consiguiente antes de adquirir las escaleras es necesario determinar la longitud de la escalera más larga que puede girar por la intersección. Para una mina en particular se tienen los siguientes valores: A = 123°, w1 = 7 pies y w2 = 9. Calcule la longitud de la escalera más larga que puede ser introducida en esta mina (en el cálculo desprecie el espesor de la escalera y asuma que la misma no está inclinada).



18. La siguiente ecuación:

$$R_1 cos(\theta) - R_2 cos(\phi) + R_3 - cos(\theta - \phi) = 0$$

Conocida como la ecuación de Freudenstein, es probablemente la expresión más empleada para determinar la relación entre los ángulos de entrada y salida de un mecanismo de cuatro barras como el representado esquemáticamente en la siguiente figura.



En esta ecuación:

$R_1 = d/c$;

$R_2 = d/a$;

$R_3 = (d^2 + a^2 - b^2 + c^2)/(2ca)$

Donde a, b, c y d son las longitudes de cada una de las barras. Como se puede observar, dado un valor θ, la ecuación de Freudenstein debe ser resuelta para encontrar el correspondiente valor de ϕ.

Un mecanismo de manubrio y palanca, en la cual el manubrio tiene un movimiento de 360 grados (θ), mientras que el movimiento de la palanca oscila (ϕ), tiene las siguientes dimensiones: a = 1 pulgada; b = 2 pulgadas; c = 2 pulgadas y d = 2 pulgadas. Calcule los ángulos de salida (ϕ) para los siguientes ángulos de entrada θ = 20°, 40°, 60°, 80°. 100°, 120° y 160° (los resultados deben tener una precisión de 4 dígitos).

3.6. Método de Regula-Falsi

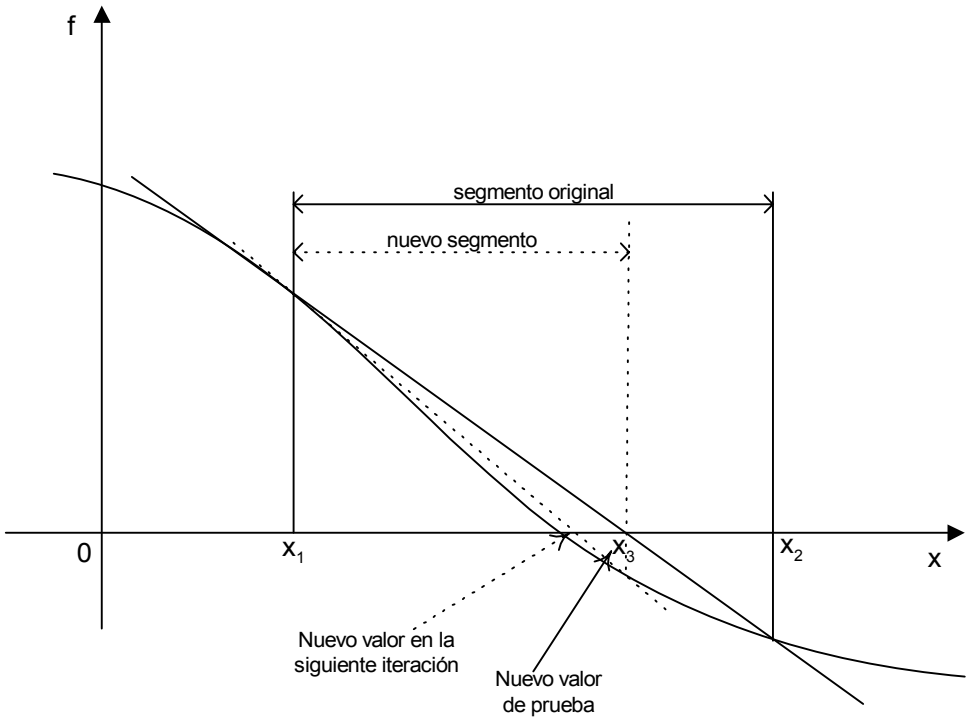


Figura 3.12. Método de Regula – Falsi.

3.6.1. Ecuación

$$x_3 = \frac{f_2 x_1 - f_1 x_2}{f_2 - f_1} \tag{3.12}$$

3.6.2. Algoritmo

El algoritmo se muestra en la figura 3.13.

3.6.3. Código

Debe ser escrito en el archivo *Formafx.S*:

```
xNAME Refa ( Método de Regula - Falsi )
:: ( Parámetros: función "f", límites del segmento: "x1", "x2", )
  CK3&Dispatch
  # 811
  :: ( f=12; x1=11; x2=10; err=9; pre=8; max=7; f1=6; f2=5; f3=4;
      fp=3; x3=2; xp=1 )
```

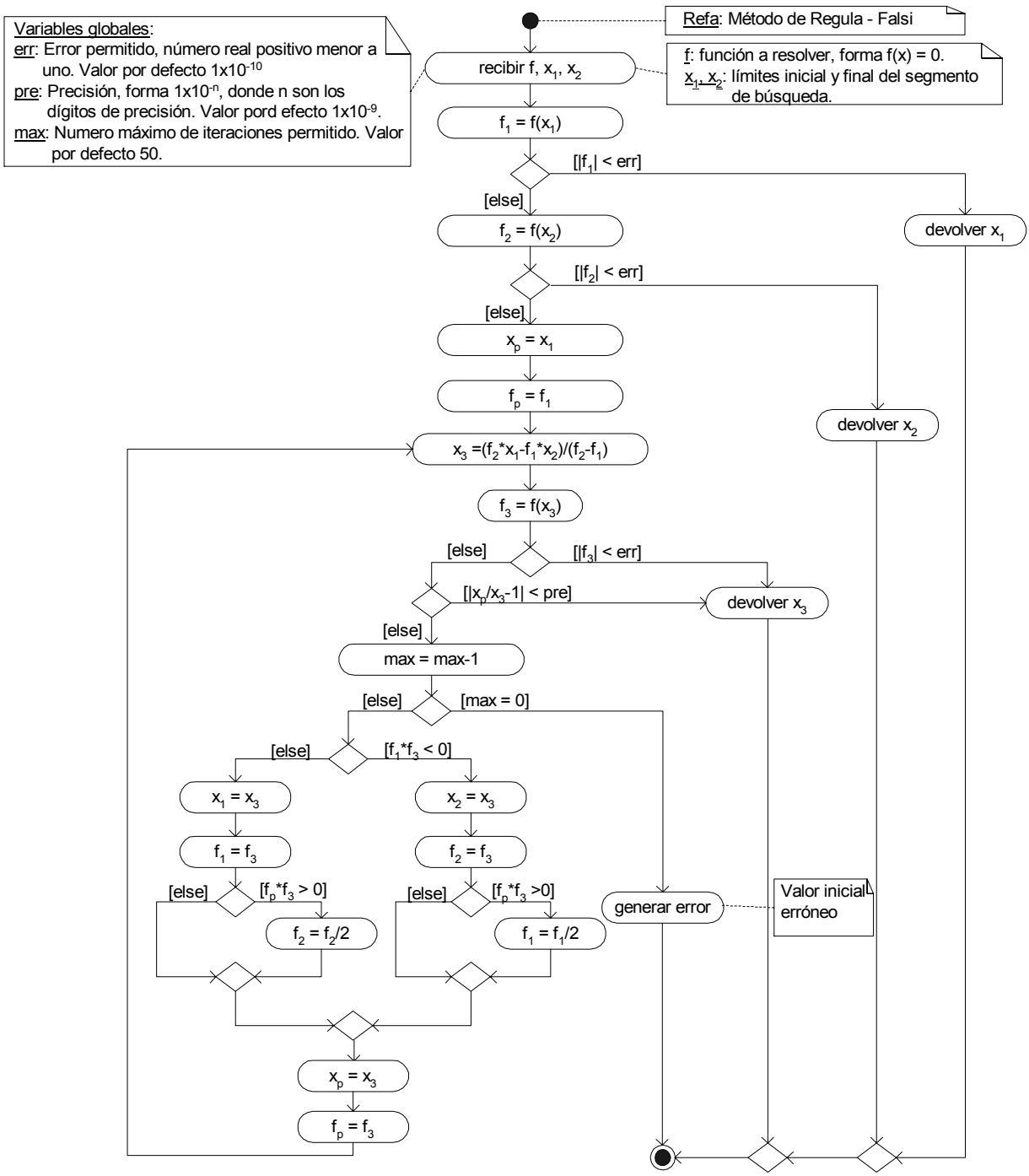


Figura 3.13. Método de Regula – Falsi.

```
' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
' ID pre @ ?SKIP % 1E-9  ( si no existe "pre" se emplea 1E-9 )
' ID max @ ?SKIP % 50    ( si no existe "max" se emplea 50 )
%0 %0 %0 %0 %0 %0
' NULLLAM TWELVE NDUPN DOBIND
::
  11GETLAM 12GETLAM EVAL 6PUTLAM ( f1=f[x1] )
  6GETLAM %ABS 9GETLAM %< ( |f1|<err )
  IT :: 11GETLAM RDROP ; ( devolver x1 )
  10GETLAM 12GETLAM EVAL 5PUTLAM ( f2=f[x2] )
```

```
      5GETLAM %ABS 9GETLAM %< ( |f2|<err )
      IT :: 10GETLAM RDROP ; ( devolver x2 )
      11GETLAM 1PUTLAM ( xp=x1)
      6GETLAM 3PUTLAM  ( fp=f1)
      BEGIN
        5GETLAM 11GETLAM %* 6GETLAM 10GETLAM %* %- 5GETLAM 6GETLAM %- %/
        2PUTLAM ( [x3=f2x1-f1x2]/[f2-f1] )
        "x:" 2GETLAM a%>$ &$ DispCoord1 ( mostrar x3 )
        2GETLAM 12GETLAM EVAL 4PUTLAM ( f3=f[x3] )
        4GETLAM %ABS 9GETLAM %< ( |f3|<err )
        IT :: 2GETLAM 2RDROP ; ( devolver x3 )
        1GETLAM 2GETLAM %/ %1- %ABS 8GETLAM %< ( |xp/x3-1|<pre )
        IT :: 2GETLAM 2RDROP ; ( devolver x3 )
        7GETLAM %1- 7PUTLAM ( max=max-1 )
        7GETLAM %0= ( max==0)
        IT :: ABND #A01 DO#EXIT ; ( valor inicial erróneo )
        3GETLAM 4GETLAM %* %0> 4GETLAM 2GETLAM ( f1*f3<0, f3, x3 )
        6GETLAM 4GETLAM %* %0< ( fp*f3>0 )
        ITE
        :: 10PUTLAM 5PUTLAM ( x2=x3, f2=f3 )
          IT :: 6GETLAM %2 %/ 6PUTLAM ; ( f1=f1/2 )
        ;
        :: 11PUTLAM 6PUTLAM ( x1=x3, f1=f3 )
          IT :: 5GETLAM %2 %/ 5PUTLAM ; ( f2=f2/2 )
        ;
        2GETLAM 1PUTLAM ( xp=x3 )
        4GETLAM 3PUTLAM ( fp=f3)
      AGAIN
    ;
    ABND
  ;
;
```

3.6.4. Ejemplos

3.6.4.1. Resolución de un sistema de cuatro ecuaciones no lineales

Como primer ejemplo resolveremos, con 5 dígitos de precisión, el sistema de ecuaciones resuelto en el segundo ejemplo del método de *Wegstein:*

$$x^{1/2} + z^2 = 35$$
$$y + e^{z/2} - \frac{8}{z} = 30$$
$$w + z^2 - 2z = 31$$
$$x + w + y = 39$$

Primero colocamos la ecuación en la forma *f(x)=0:*

$$f(x) = x + w + y - 39 = 0$$
$$z = \sqrt{35 - x^{1/2}}$$
$$y = 30 + \frac{8}{z} - e^{z/2}$$
$$w = 31 - z^2 + 2z$$

Y resolvemos el sistema con el programa:

```
« 0 0 0 → w y z
  « 1E-5 'pre' STO
```

```
    « → x
      « 35 x √ - √ 'z' STO
        30 8 z / + z 2 / EXP - 'y' STO
        31 z SQ - 2 z * + 'w' STO
        x w + y + 39 -
      »
    »
    DUP 1 1 Incr1
    IF 1 ≠ THEN
      DUP 1 + Refa
    END
    "x" →TAG
    y "y" →TAG
    z "z" →TAG
    w "w" →TAG
    'pre' PURGE
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

x: 13.1891885976

y: 14.977635724

z: 5.60074171429

w: 10.8331756784

s: 1.4613

Que son las soluciones del sistema y el tiempo empleado en obtenerlas.

3.6.4.2. Resolución de un sistema de ecuaciones no lineales

El siguiente sistema de ecuaciones no lineales es el modelo matemático de un proceso físico. Calcularemos los valores de L_1, x_1, V_1, y_1 resolviendo el sistema (que depende de una variable: x_1) con 9 dígitos de precisión:

$$f(x_1) = L_1 * x_1 + V_1 * y_1 - C_0 = 0$$
$$L_1 = \frac{L_c}{1 - x_1}$$
$$V_1 = M - L_1$$
$$y_1 = fe(x_1)$$
$$L_c = L_0 * (1 - x_0)$$
$$M = L_0 + V_0$$
$$C_0 = L_0 * x_0 + V_0 * y_0$$
$$L_0 = 300$$
$$x_0 = 0$$
$$V_0 = 100$$
$$y_0 = 0.2$$
$$fe = 1420 * x$$

Emplearemos como valor inicial el valor de x_0 y un incremento igual a 0.1. Como la ecuación se encuentra ya en la forma $f(x)=0$, puede ser programada directamente:

```
« 300 0 100 0.2 « → x « 1420 x * » »
```

```
  0 0 0 0 0 0
  → Lo xo Vo yo fe Co y1 M V1 Lc L1
  « Lo xo * Vo yo * + 'Co' STO
    Lo Vo + 'M' STO
    Lo 1 xo - * 'Lc' STO
    « → x1
      « Lc 1 x1 - / 'L1' STO
        M V1 - 'V1' STO
        x1 fe EVAL 'y1' STO
        L1 x1 * V1 y1 * + Co -
      »
    »
    DUP xo 0.1 Incr1
    IF 1 ≠ THEN
      DUP 0.1 + Refa
    END
    "x1" →TAG
    L1 "L1" →TAG
    y1 "y1" →TAG
    V1 "V1" →TAG
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

x1: 5.94641729792E-5

L1: 300.017840313

y1: 8.44391256305E-2

V1: 0.

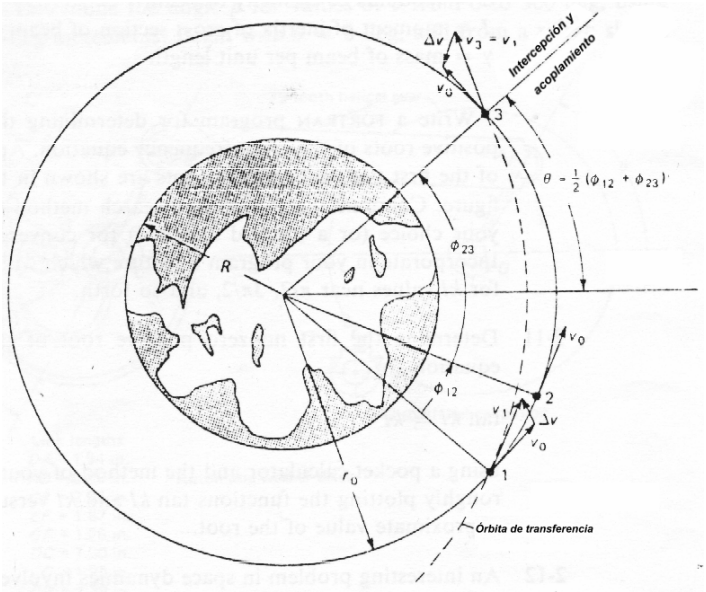
s: 4.8918

Que son las soluciones y el tiempo empleado en obtenerlas.

3.6.5. Ejercicios

19. Dos vehículos espaciales que giran alrededor de la tierra en la misma órbita circular (ver figura), tienen que acoplarse con fines de aprovisionamiento. Los dos vehículos tienen la misma velocidad y el vehículo dos se encuentra delante del vehículo uno, separado por una distancia angular ϕ_{12}. Para que el vehículo 1 alcance al vehículo 2, el vehículo 1 debe ser colocado en una órbita diferente (usualmente parabólica). El vehículo 2 es colocado en la nueva órbita mediante un impulso que además incrementa su velocidad. Cuando el vehículo 2 alcanza al vehículo 1, se le aplica otra fuerza impulsora que coloca el vehículo 2 en la órbita del vehículo uno y que disminuye su velocidad hasta igualar la del vehículo 1, a partir de este punto, los dos vehículos se mueven conjuntamente.

    Para determinar el incremento de velocidad que debe aplicarse al vehículo 2, es necesario calcular la excentricidad (e) de la órbita de transferencia. Esta excentricidad se determina basándose en el hecho de que los dos vehículos emplean el mismo tiempo para llegar hasta el punto de encuentro (punto 3). La ecuación resultante, deducida con la ecuación de movimiento de un cuerpo alrededor de un campo gravitacional central y la ecuación de la sección transversal de un cono, es la siguiente:



$$\frac{\pi \phi_{23}}{360} = \sqrt{\left(\frac{1+e\ \cos\ \theta}{e^2\ -\ 1}\right)^3} * \left[\frac{e\ \sqrt{e^2 - 1}}{1+e\ \cos\ \theta}\ \sin\ \theta - \ln\left(\frac{\sqrt{e\ +\ 1} + \sqrt{e\ -\ 1}\ \tan\left(\frac{\theta}{2}\right)}{\sqrt{e\ +\ 1} - \sqrt{e\ -\ 1}\ \tan\left(\frac{\theta}{2}\right)}\right)\right]$$

Donde ϕ_{23} es el desplazamiento angular (medido en grados) que debe realizar el vehículo 2 para llegar al punto de encuentro y θ es igual a $(\phi_{12}+\phi_{23})/2$. Calcule, con una precisión de 5 dígitos, la excentricidad (e) para los siguientes desplazamientos angulares: $\phi_{12} = 30°$ y $\phi_{23} = 20°$, $30°$, $40°$ y $50°$.

20. El siguiente sistema de ecuaciones no lineales, dependiente de una variable: T_2, es el modelo matemático de un proceso físico. Calcule los valores de T_2 y H_2 con 6 dígitos de precisión, empleando como valor inicial el 20% de T_1 y un incremento igual a 10.

$$f(T_2) = \frac{H_2 - H_1}{T_2 - T_1} + \frac{cs_1}{\lambda_2} = 0$$

$$H_2 = \frac{HP_2 * HS_2}{100}$$

$$HS_2 = \frac{18.02}{28.97} * \frac{PV(T_2)}{P - PV(T_2)}$$

$$PV(T_2) = e^{\left(18.3036 - \frac{3816.44}{T_2 + 227.02}\right)}$$

$$cs_1 = 1.005 + 1.88 * H_1$$

$$\lambda_2 = 2501.4 * \left(\frac{1 - \frac{T_2}{647.3}}{1 - 0.422}\right)^{0.375}$$

$$T_1 = 20$$

$$H_1 = 0.002$$

$$P = 540$$

$$HP_2 = 80$$

3.7. Método de la Secante

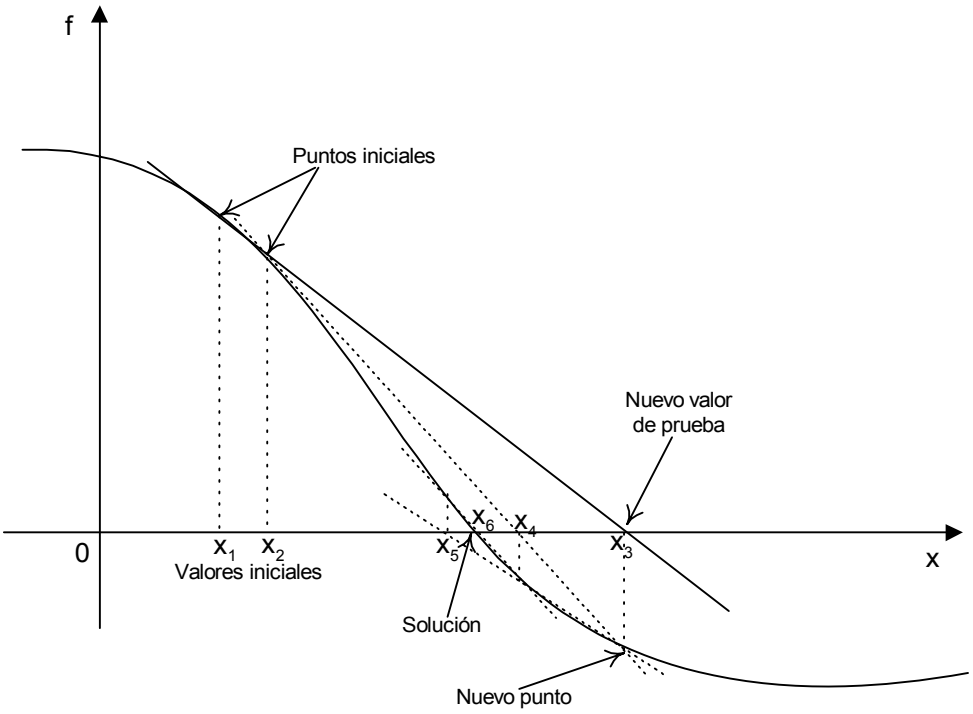


Figura 3.14. Método de la Secante

3.7.1. Ecuación

$$x_3 = \frac{f_2 x_1 - f_1 x_2}{f_2 - f_1} \tag{3.12}$$

3.7.2. Algoritmo

El algoritmo se presenta en la figura 3.15 de la siguiente página.

3.7.3. Código

El código, que debe ser escrito en el archivo *Formafx.S* es:

```
xNAME Seca ( Método de la Secante )
:: ( Parámetros: función "f", Valores asumidos: "x1", "x2", )
  CK3&Dispatch
  # 811
  :: ( f=10; x1=9; x2=8; err=7; pre=6; max=5; f1=4; f2=3; f3=2; x3=1 )
    ' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
    ' ID pre @ ?SKIP % 1E-9  ( si no existe "pre" se emplea 1E-9 )
    ' ID max @ ?SKIP % 50    ( si no existe "max" se emplea 50 )
    %0 %0 %0 %0
    ' NULLLAM TEN NDUPN DOBIND
    ::
      9GETLAM 10GETLAM EVAL 4PUTLAM ( f1=f[x1] )
      4GETLAM %ABS 7GETLAM %< ( |f1|<err )
```

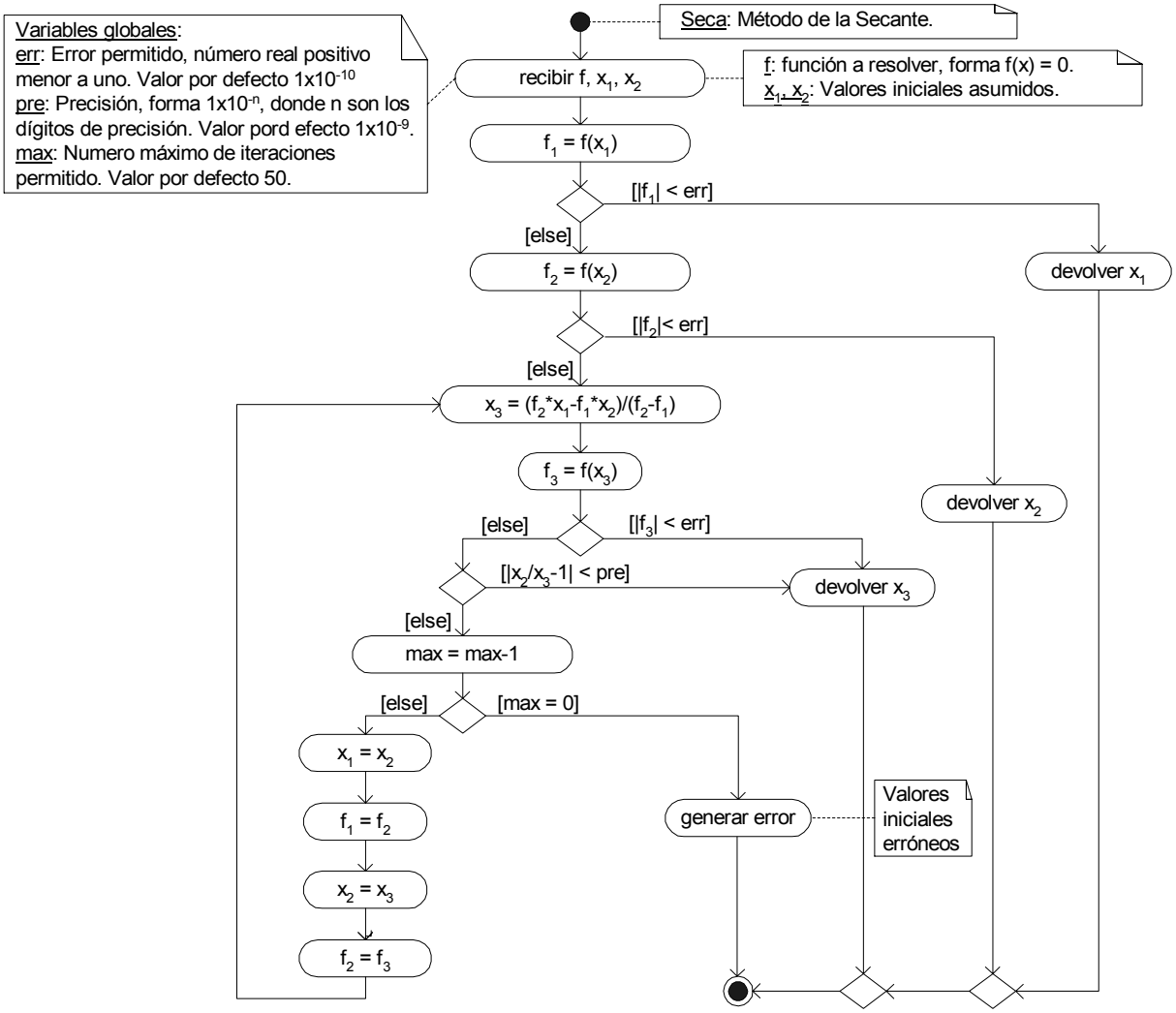


Figura 3.15. Algoritmo del método de la Secante.

```
IT :: 9GETLAM RDROP ; ( devolver x1 )
8GETLAM 10GETLAM EVAL 3PUTLAM ( f2=f[x2] )
3GETLAM %ABS 7GETLAM %< ( |f2|<err )
IT :: 8GETLAM ABND RDROP ; ( devolver x2 )
BEGIN
  3GETLAM 9GETLAM %* 4GETLAM 8GETLAM %* %- 3GETLAM 4GETLAM %- %/
  1PUTLAM ( x3=[f2x1-f1x2]/[f2-f1] )
  "x:" 1GETLAM a%>$ &$ DispCoord1 ( mostrar x3 )
  1GETLAM 10GETLAM EVAL 2PUTLAM ( f3=f[x3] )
  2GETLAM %ABS 7GETLAM %< ( |f3|<err )
  IT :: 1GETLAM 2RDROP ; ( devolver x3 )
  8GETLAM 1GETLAM %/ %1- %ABS 6GETLAM %< ( |x2/x3-1|<pre )
  IT :: 1GETLAM 2RDROP ; ( devolver x3 )
  5GETLAM %1- 5PUTLAM ( max=max-1 )
  5GETLAM %0= ( max==0 )
  IT :: ABND #A01 DO#EXIT ; ( valor inicial erróneo )
  8GETLAM 9PUTLAM ( x1=x2 )
  3GETLAM 4PUTLAM ( f1=f2 )
  1GETLAM 8PUTLAM ( x2=x3 )
  2GETLAM 3PUTLAM ( f2=f3 )
```

```
      AGAIN
    ;
    ABND
  ;
;
```

3.7.4. Ejemplo manual

Para comprender mejor el método y posibilitar la depuración del programa haciéndolo correr paso a paso, resolveremos manualmente la ecuación cúbica:

$$f(x) = x^3 + 2x^2 + 3x + 4 = 0 \qquad (3.11)$$

Simplemente para demostrar que en este método no se requiere conocer el segmento de búsqueda, tomaremos como valores iniciales: $x_1 = 1.1$ y $x_2 = 1.2$. Por supuesto si tomamos como valores iniciales los límites del segmento $x_1 = -2$ y $x_2 = -1$ (calculados con el método incremental) encontraríamos la solución en menos iteraciones.

Programamos primero la función:

```
« → X « X 3 ^ 2 X SQ * + 3 X * + 4 + » » 'fx' STO
```

Entonces con los valores iniciales asumidos encontramos primero los valores de la función: $f_1 = 1.1$ $fx = 11.051$; $f_2 = 1.2$ $fx = 12.208$.

Ahora con la ecuación 3.12, calculamos el nuevo valor de prueba:

$$x_3 = \frac{f_2 x_1 - f_1 x_2}{f_2 - f_1} = \frac{12.208*1.1 - 11.051*1.2}{12.208 - 11.051} = 0.144857389801$$

Y con el mismo calculamos el nuevo valor de la función: $f_3 = 0.144857389801$ $fx = 4.47957913487$. Dado que este valor no es cercano a cero repetimos el procedimiento realizando previamente un cambio de variables: $x1 = x2 = 1.2$; $x2 = x3 = 0.144857389801$; $f1 = f2 = 12.208$; $f2 = f3 = 4.47957913487$:

$$x_3 = \frac{f_2 x_1 - f_1 x_2}{f_2 - f_1} = \frac{4.47957913487*1.2 - 12.208*0.144857389801}{4.47957913487 - 12.208} = -0.466728716008$$

Entonces el nuevo valor de la función f_3 es: -0.466728716008 $fx = 2.93381506675$, que como vemos todavía no es cercano a cero, por lo que el proceso debe ser repetido.

Los resultados obtenidos en posteriores iteraciones se presentan en la siguiente tabla:

x_1	x_2	f_1	f_2	x_3	f_3
0.144857389801	-0.466728716008	4.47957913487	2.93381506675	-1.62750128722	0.10415599266
-0.466728716008	-1.62750128722	2.93381506675	0.10415599266	-1.67022778403	-0.09073071159
-1.62750128722	-1.67022778403	0.10415599266	-0.09073071159	-1.65033619895	0.00133907781
-1.67022778403	-1.65033619895	-0.09073071159	0.00133907781	-1.65062550535	0.00001684986
-1.65033619895	-1.65062550535	0.00133907781	0.00001684986	-1.65062919213	-0.00000000316
-1.65062550535	-1.65062919213	0.00001684986	-0.00000000316	-1.65062919144	-0.00000000001

Como vemos el último valor la función (f_3) es prácticamente cero, por lo tanto la solución es -1.65062919144.

3.7.5. Ejemplos

3.7.5.1. Resolución de una ecuación no lineal con una incógnita

Como primer ejemplo resolveremos la siguiente ecuación no lineal:

$$f(z) = ln(z) + e^{z+3} - z^{3.1} - 42 = 0$$

Como ya se encuentra en la forma *f(x)=0*, puede ser programada directamente:

```
«
  « → z
    « z LN z 3 + EXP + z 3.1 ^ - 42 -
    »
  »
  1.1 1.11 Seca
  "z" →TAG
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

z: .754179223075

s: .7913

Que es la solución y el tiempo (en segundos) empleado en encontrarla.

3.7.5.2. Resolución de un sistema de ecuaciones no lineales

Como segundo ejemplo calcularemos el valor de Y_{A1} resolviendo el siguiente sistema de ecuaciones no lineales:

$$\begin{aligned}
&L_N + V_1 = M \\
&L_N X_{AN} + V_1 Y_{A1} = C_{A0} \\
&L_N X_{BN} + V_1 Y_{B1} = C_{B0} \\
&Y_{B1} = 2Y_{A1} + Y_{A1}^2 + 3.44 Y_{A1}^3 \\
&X_{BN} = 2.2 + 5.1 X_{AN} - 7.2 X_{AN}^{2.2} \\
&X_{AN} = 0.12 \\
&M = 32.1 \\
&C_{A0} = 20.2 \\
&C_{B0} = 3.42
\end{aligned}$$

Primero colocamos este sistema de ecuaciones no lineales en la forma *f(x)=0:*

$$\begin{aligned}
&f(Y_{A1}) = L_N + V_1 - M = 0 \\
&Y_{B1} = 2Y_{A1} + Y_{A1}^2 + 3.44 Y_{A1}^3 \\
&X_{BN} = 2.2 + 5.1 X_{AN} - 7.2 X_{AN}^{2.2} \\
&\left.\begin{aligned} L_N X_{AN} + V_1 Y_{A1} &= C_{A0} \\ L_N X_{BN} + V_1 Y_{B1} &= C_{B0} \end{aligned}\right\} \quad L_N = \frac{C_{A0} Y_{B1} - C_{B0} Y_{A1}}{X_{AN} Y_{B1} - X_{BN} Y_{A1}};\ V_1 = \frac{C_{A0} - L_N X_{AN}}{Y_{A1}} \\
&X_{AN} = 0.12 \\
&M = 32.1 \\
&C_{A0} = 20.2 \\
&C_{B0} = 3.42
\end{aligned}$$

Y elaboramos el programa respectivo:

```
« 0.12 32.1 20.2 3.42 0 0 0 0 0
  → XAN M CA0 CB0 YA1 YB1 ABN LN V1
  «
    2.2 5.1 XAN * + 7.2 XAN 2.2 ^ * - 'XBN' STO
```

```
    « 'YA1' STO
      2 YA1 * YA1 SQ + 3.44 YA1 3 ^ * + 'YB1' STO
      CA0 YB1 * CB0 YA1 * - XAN YB1 * XBN YA1 * - /  'LN' STO
      CA0 LN XAN * - YA1 / 'V1' STO
      LN V1 + M -
    »
    XAN 0.5 * DUP 1.01 * Seca
    "YA1" →TAG
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene

Y_{A1}*: .411748600835*

s: 2.3972

Que es la solución buscada y el tiempo empleado en obtenerla.

3.7.5.3. Resolución de la ecuación de Benedict – Webb - Rubin

Como tercer ejemplo elaboraremos un programa para resolver la ecuación de estado de Benedict – Webb - Rubin:

$$P = \frac{RT}{V} + \frac{B_0 RT - A_0 - C_0 T^{-2}}{V^2} + \frac{bRT - a}{V^3} + \frac{a\alpha}{V^6} + \frac{c}{V^3 T^2}\left(1 + \frac{\gamma}{V^2}\right) e^{\frac{-\gamma}{V^2}}$$

$$R = 0.08206 \frac{atm.l}{K.gmol}$$

Donde las constantes dependen del compuesto cuyas propiedades se quieren calcular, así para el hexano las constantes son las siguientes: a=7.11671; A_0=1.44373E1; b=1.09131E-1; B_0=1.77813e-1; c=1.51276E6; C_0=3.31935E6; α=2.81086E-3; γ=6.66849E-2.

Calcularemos el volumen del hexano para una presión (P) de 30 atm y una temperatura de 600 K. Emplearemos como valor inicial el volumen calculado con la ecuación del gas ideal: *V=R*T/P*.

Primero colocamos la ecuación en la forma *f(x)=0*:

$$f(V) = \frac{RT}{V} + \frac{B_0 RT - A_0 - C_0 T^{-2}}{V^2} + \frac{bRT - a}{V^3} + \frac{a\alpha}{V^6} + \frac{c}{V^3 T^2}\left(1 + \frac{\gamma}{V^2}\right) e^{\frac{-\gamma}{V^2}} - P = 0$$

$$R = 0.08206 \frac{atm.l}{K.gmol}$$

Y elaboramos el programa respectivo:

```
« 0.08206 7.11671 1.44373E1 1.09131E-1 1.77813E-1
  1.51276E6 3.31935E6 2.81086E-3 6.66849E-2 30 600
  → R a A0 b B0 c C0 α γ P T
  «
    « → V
      «
        R T * V / B0 R * T * A0 - C0 T -2 ^ * - V SQ / +
        b R * T * a - V 3 ^ / + a α * V 6 ^ / +
        c V 3 ^ T SQ * / 1 γ V SQ / + * γ NEG V SQ / EXP * + P -
      »
    »
    R T * P / DUP 1.2 * Seca
    "V" →TAG
```

```
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

V: 1.30981987739

s: 2.5186

Que es el volumen buscado (en litros/gmol) y en tiempo empleado en calcularlo.

3.7.6. Ejercicios

21. Resuelva el ejercicio 9 empleando el método de la secante, con 7 dígitos de precisión.

22. La ecuación de estado de Beattie-Bridgeman es la siguiente:

$$V = \left(RT + \frac{\beta}{V} + \frac{\gamma}{V^2} + \frac{\delta}{V^3} \right) \frac{1}{P}$$

$$R = 0.08206 \frac{atm.l}{K.gmol}$$

Donde las constantes dependen del compuesto cuyas propiedades se quieren calcular, así para el caso del isobutano son: A_0=16.6037; B_0=0.2354; a= 0.11171; b=0.07697; c=300x104. Calcule el volumen del isobutano a una presión de 16 atm y una temperatura de 428 K. Emplee como valor inicial el volumen calculado con la ecuación del gas ideal: *V=R*T/P*.

23. Elabore un programa que empleando el método de la *Secante* calcule el valor de x_i resolviendo el siguiente sistema de ecuaciones no lineales con 6 dígitos de precisión (x=0.6; y=0.4).

$$\frac{k_x}{k_y} = -\frac{y - y_i}{x - x_i}$$

$$y_i = -.0405 + 1.03785714286 x_i$$

$$k_x = \frac{k'_x}{(1-x)_{im}};$$

$$k_y = \frac{k'_y}{(1-y)_{im}}$$

$$(1-x)_{im} = \frac{(1-x)-(1-x_i)}{\ln\left(\frac{1-x}{1-x_i}\right)};$$

$$(1-y)_{im} = \frac{(1-y_i)-(1-y)}{\ln\left(\frac{1-y_i}{1-y}\right)}$$

$$k'_x = 0.001967$$

$$k'_y = 0.001465$$

3.8. Método de Newton – Raphson. Derivada analítica

El método de Newton es probablemente el método más empleado en la práctica para resolver ecuaciones algebraicas con una incógnita. En este método se asume un valor inicial (x_1) y con el mismo se calcula la derivada de la fun-

ción, que como se ve en la figura 3.16, es la tangente a la curva. Entonces en la intersección de la tangente con la recta *f=0* se determina el nuevo valor de prueba *(x_2)* y con el mismo el valor de la función (f_2).

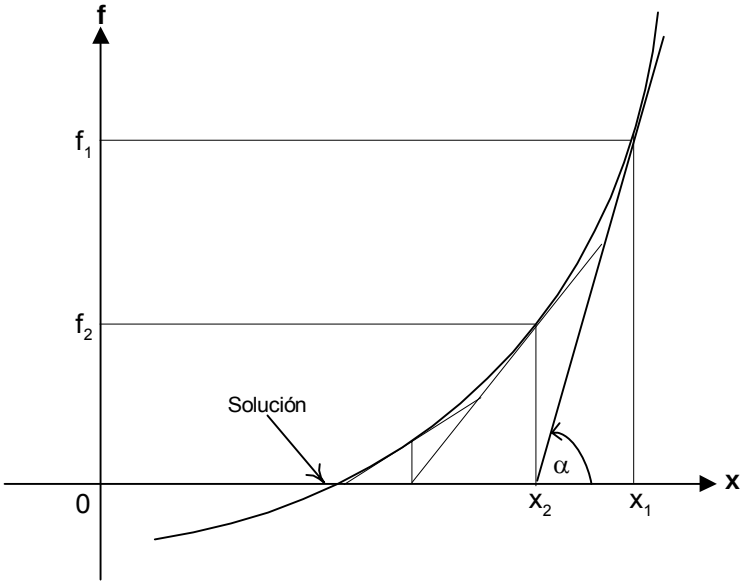


Figura 3.16. Método de Newton - Raphson

Si el valor de la función es casi cero el proceso concluye, caso contrario se repite (empleando el nuevo valor de prueba) hasta que la función es casi cero, o hasta que los dos últimos valores calculados (x_1 y x_2) son aproximadamente iguales.

La ecuación que permite calcular el nuevo valor de prueba *(x_2)* puede ser deducida del triángulo que forman la tangente con los segmentos *(x_1 - x_2)* y *(f_1 - 0)*:

$$\tan(\alpha) = f'(x_1) = \frac{f_1 - 0}{x_1 - x_2}$$

$$x_1 - x_2 = \frac{f_1}{f'(x_1)}$$

$$x_2 = x_1 - \frac{f_1}{f'(x_1)} \qquad (3.13)$$

Como se puede observar, la ecuación es sencilla, pero involucra el cálculo de la derivada de la función.

En la HP es posible calcular la derivada analítica de funciones explícitas (aunque no de funciones implícitas). Sin embargo, el cálculo de la derivada analítica es un proceso que suele consumir mucho tiempo (generalmente más tiempo que el requerido para resolver la ecuación en sí), por ello se emplea la derivada analítica sólo en aquellos casos de uso más frecuente o donde el cálculo de la derivada es muy sencillo.

Cuando la derivada es una expresión compleja o cuando no es posible derivar analíticamente la función (como sucede con la mayoría de las funciones implícitas) el cálculo de la derivada se realiza numéricamente (como veremos posteriormente).

Como ejemplo de la forma analítica, calculemos la raíz cuadrada de un número real "n" y sea x la solución buscada:

$$x = \sqrt{n}$$

Elevando ambos lados de la ecuación al cuadrado y colocando la ecuación en la forma *f(x)=0*:

$$f(x) = x^2 - n = 0$$

Entonces aplicamos la ecuación de *Newton* a esta función (ecuación 3.13):

$$x_2 = x_1 - \frac{x_1^{\,2} - n}{\frac{\partial}{\partial x_1}\left(x_1^{\,2} - n\right)}$$

Aunque en este caso el cálculo de la derivada es muy sencillo, la resolveremos empleando la calculadora, para ello introducimos el lado derecho de la anterior ecuación en el editor de ecuaciones [EQW] o la escribimos directamente en la pila:

'x1-(x1^2-n)/(∂x1(x1^2-n))'

Para resolver funciones analíticas deben estar libres las banderas 2, 3, 111 y 119: *{ -2 -3 -111 -119 } CF*. Entonces, pulsando la tecla EVAL obtenemos:

$$\frac{x_1^{\,2} + n}{2x_1}$$

Reordenando un poco esta ecuación y reemplazándola en la ecuación de *Newton* obtenemos:

$$x_2 = \frac{1}{2}\left(x_1 + \frac{n}{x_1}\right) \qquad (3.14)$$

Que es le ecuación de *Newton* para el cálculo de la raíz cuadrada.

El anterior procedimiento es útil cuando la función se resuelve con mucha frecuencia, no obstante, en la mayoría de los casos prácticos no se realizan las operaciones algebraicas que simplifican la ecuación resultante, sino que la función y la derivada se emplean directamente en la ecuación 3.13.

Para volver a trabajar con métodos numéricos, no debe olvidar volver a fijar las banderas 2, 3, 111 y 119: *{ -2 -3 -111 -119 } SF*.

3.8.1. Ejemplo manual

Para comprender mejor el método de Newton-Raphson y posibilitar la depuración haciendo correr el programa paso a paso, resolveremos una vez más la ecuación función:

$$f(x) = x^3 + 2x^2 + 3x + 4 = 0 \qquad (3.11)$$

Primero calculamos la derivada (*{ -2 -3 -105 -111 -119 } CF*):

*'∂x(x^3+2*x^2+3*x+4)' SIMPLIFY*

Con lo que obtenemos:

$$f'(x) = 3x^2 + 4x + 3$$

Entonces programamos la función y su derivada:

```
« → X « X 3 ^ 2 X SQ * + 3 X * + 4 + » » 'fx' STO
« → X « 3 X SQ * 4 X * + 3 +  » » 'dfx' STO
```

Asumimos un valor inicial $x_1 = 1.1$ y sustituimos este valor en la función y en su derivada:

$f(x_1)$ = 1.1 fx EVAL = 11.051

$f'(x_1)$ = 1.1 dfx EVAL = 11.03

Entonces con la ecuación del *Newton* (ecuación 3.13) obtenemos:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1.1 - \frac{11.051}{11.03} = 0.09809610154$$

Que es el nuevo valor de prueba. Como la función no es aproximadamente cero, ni son iguales los valores de x_1 y x_2, el proceso se repite, realizando antes un cambio de variables: $x_1=x_2$.

Los valores obtenidos en las otras iteraciones se muestran en la siguiente tabla:

x_1	$f(x_1)$	$f'(x_1)$	x_2
0.09809610154	4.31447795849	3.42125294157	-1.162985155
-1.162985155	1.64313596539	1.6413596539	-1.84601353732
-1.84601353732	-1.01329078306	5.83924379062	-1.6724823714
-1.6724823714	-0.10131569347	4.70166236232	-1.6509334626
-1.6509334626	-0.00139116173	4.57301004339	-1.65062925121
-1.65062925121	-0.00000027322	4.57121377001	-1.65062916144
-1.65062916144	-0.00000000001	4.57121341713	-1.65062919144

Puesto que los dos últimos valores son iguales en todos sus dígitos y además el valor de la función $(f(x_1))$ es casi cero, el proceso concluye, siendo la solución: *-1.65062919144*.

3.8.2. Algoritmo

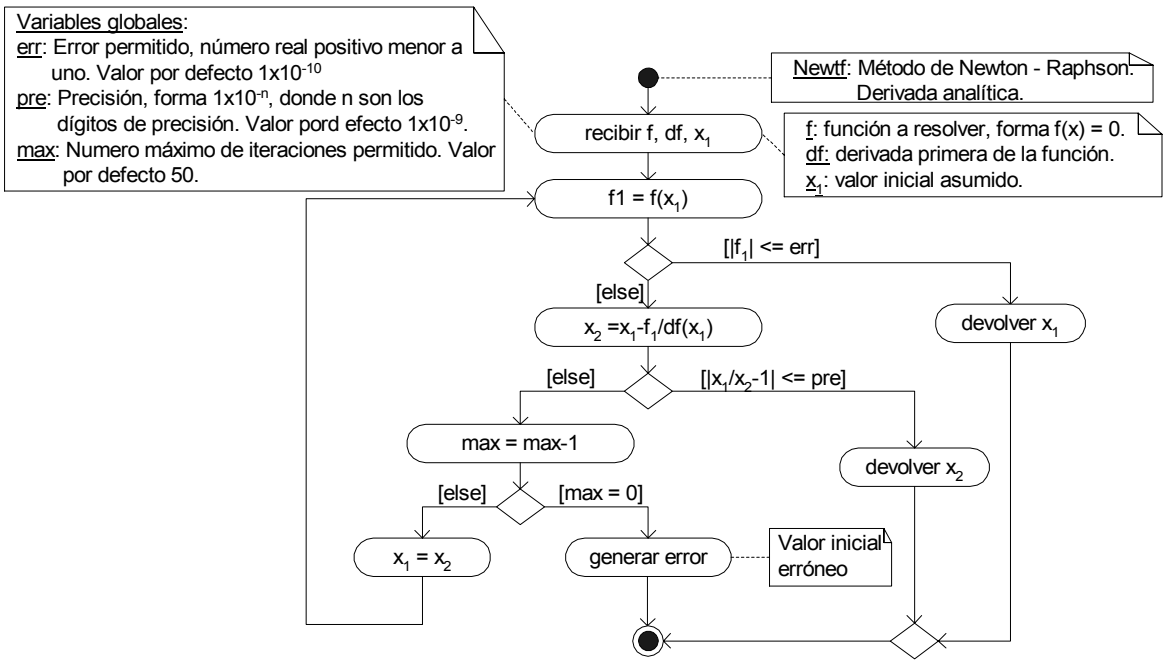


Figura 3.17. Algoritmo del método de Newton – Raphson. Derivada analítica.

3.8.3. Código

El código elaborado en *System RPN* siguiendo el algoritmo es el siguiente:

```
xNAME Newtf ( Método de Newton - Raphson. Derivada analítica conocida )
:: ( parámetros: f: función, forma fx=0, x1: valor asumido;
     df: derivada primera de la función )
  CK3&Dispatch
  # 881
  :: ( f=8; df=7; x1=6; err=5; pre=4; max=3; x2=2; f1=1 )
    ' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
    ' ID pre @ ?SKIP % 1E-9 ( si no existe "pre" se emplea 1E-9 )
    ' ID max @ ?SKIP % 50 ( si no existe "max" se emplea 50 )
    %0 %0
    ' NULLLAM EIGHT NDUPN DOBIND
    ::
      BEGIN
        6GETLAM 8GETLAM EVAL 1PUTLAM ( f1=f[x1])
        1GETLAM %ABS 5GETLAM %< ( |f1|<err )
        IT :: 6GETLAM 2RDROP ; ( devolver x1 )
        6GETLAM 1GETLAM 6GETLAM 7GETLAM EVAL %/ %- 2PUTLAM ( x2=x1-f1/df1 )
        "x:" 2GETLAM a%>$ &$ DispCoord1 ( mostrar x2 )
        6GETLAM 2GETLAM %/ %1- %ABS 4GETLAM %< ( |x1/x2-1|<pre )
        IT :: 2GETLAM 2RDROP ; ( devolver x2 )
        3GETLAM %1- 3PUTLAM  ( max=max-1 )
        3GETLAM %0= ( max==0 )
        IT :: ABND #A01 DO#EXIT ; ( valor incial erróneo )
        2GETLAM 6PUTLAM ( x1=x2 )
      AGAIN
    ;
    ABND
  ;
;
```

3.8.4. Ejemplos

3.8.4.1. Resolución de una ecuación no lineal con una incógnita

Como primer ejemplo resolveremos, con 7 dígitos de precisión, la ecuación:

$$y=\left(52+3\sqrt{y}-8y^{0.8}\right)^{0.36}$$

Primero colocamos la ecuación en la forma *f(x)=0:*

$$f(y)=y-\left(52+3\sqrt{y}-8y^{0.8}\right)^{0.36}=0$$

Y calculamos la derivada analítica de la función:

```
'δy(y-(52.+3.*√y-8.*y^.8)^.36)' SIMPLIFY SIMPLIFY
```

Con lo que se obtiene:

$$\frac{\left(2.304yy^{-0.2}-0.54\sqrt{y}\right)\left(52+3\sqrt{y}-8y^{0.8}\right)^{-0.64}+y}{y}$$

Ahora podemos elaborar el programa que resuelve la función:

```
« 1E-7 'pre' STO
```

```
  « → Y « Y 52 3 Y √ * + 8 Y 0.8^ * - 0.36 ^ - » »
  « → Y « 2.304 Y * Y -.2 ^ * 0.54 Y √ * -
          52 3 Y √ * + 8 Y 0.8 ^ * - -0.64 ^ * Y + Y / »
  »
  1.1  Newtf
  "Y" →TAG
  'pre' PURGE
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

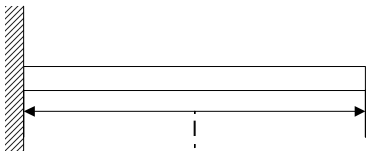
Y: 3.60967088614

s: .8718

Que es la solución buscada y el tiempo empleado en obtenerla (que es menor al tiempo requerido por *ROOT*).

3.8.4.2. Frecuencia natural de una viga

Como segundo ejemplo volveremos a resolver el ejemplo 3.4.5.1:



$$f(k) = cos(kl)cosh(kl) + 1 = 0$$

Primero calculamos la derivada analítica de esta función (*{ -2 -3 -111 -119 } CF):*

```
'∂k(COS(k*l)*COSH(k*l)+1)' SIMPLIFY
```

Con lo que obtenemos:

$$f'(k) = -\left(l\sin(kl)\right)\cosh(kl) + \cos(kl)l\sinh(kl)$$

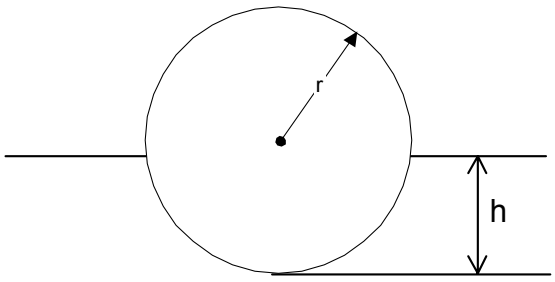
Entonces elaboramos el programa respectivo:

```
« 120 170.6 3E6 0.06 32 386 0
  → l I E y A g a
  « 1E-8 'pre' STO
    E I * g * A y * / √ 'a' STO
    « → k
      « k l * COS k l * COSH * 1 + »
    »
    « → k
      « l k l * SIN * NEG  k l * COSH * k l * COS l * k l * SINH * + »
    »
    0.01 Newtf
    SQ a * "p" →TAG
    'pre' PURGE
  »
»
```

3.8.4.3. Profundidad de una esfera sumergida

Como tercer ejemplo volveremos a resolverel ejemplo 3.5.3.1:

$$f(x) = x^3 - 3x^2 + 4d = 0$$
$$d = 0.6$$



Primero calculamos la derivada analítica de la función:

```
'δx(x^3-3*x^2+4*d)' SIMPLIFY SIMPLIFY
```

Con lo que se obtiene la derivada:

$$f'(x) = 3x^2 - 6x$$

Ahora podemos elaborar el programa respectivo:

```
« 0.6 → d
  «
    « → x « x 3 ^ 3 x SQ * - 4 d * + » »
    « → x « 3 x SQ * 6 x * - » »
    1.1 Newtf
    "x" →TAG
  »
»
```

Evaluando con *TEVAL* se obtiene:

x: 1.13413784571

s: .375

Que es la fracción del radio que se hunde y el tiempo empleado en obtenerlo.

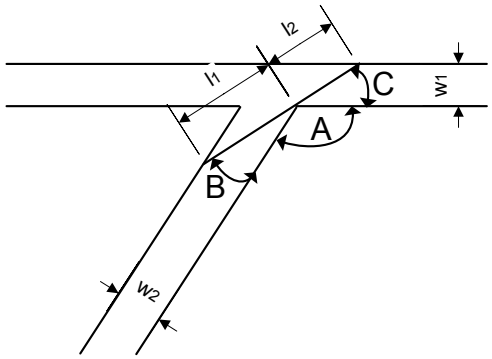
3.8.5. Ejercicios

24. Derive la ecuación de *Newton* para el cálculo de la raíz cúbica.

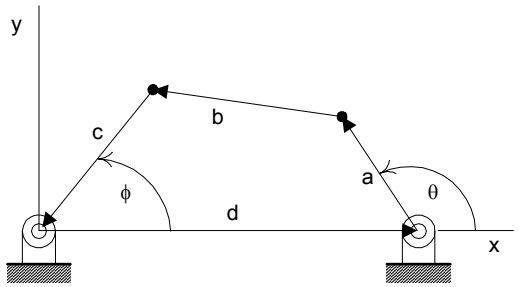
25. Derive la ecuación de *Newton* para encontrar una de las raíces (soluciones) de la ecuación cúbica: $ax^3+bx^2+cx+d=0$.

26. Elabore un programa que empleando el método de *Newton* con derivación analítica resuelva el ejercicio 9 con 8 dígito de precisión.

27. Elabore un programa que empleando el método de *Newton* con derivación analítica resuelva el ejercicio 17 con 6 dígitos de precisión.



28. Elabore un programa que resuelva el ejercicio 18 con el método de *Newton* con derivación analítica con 7 dígitos de precisión.



3.9. Método de Newton – Raphson. Derivada numérica

Como ya se señaló en el acápite anterior el cálculo de la derivada analítica suele requerir mucho tiempo y en ocasiones no es posible calcularla pues la función es implícita. Por estas razones el método de Newton – Raphson es de mayor utilidad cuando la derivada se calcula numéricamente.

3.9.1. Cálculo numérico de la derivada

El cálculo numérico de las derivadas se efectúa empleando fórmulas que son deducidas en base al concepto de la derivada:

$$\frac{dy}{dx} = \lim_{\Delta x \to 0}\left(\frac{\Delta y}{\Delta x}\right) \tag{3.15}$$

Donde numéricamente se consigue aproximarse al límite empleando valores de *"Δx"* pequeños. Básicamente se toman valores consecutivos de *"x"*, los que constituyen el incremento *"Δx"* y para estos valores se calculan los correspondientes valores de la función, los que constituyen los valores de *"Δy"*.

La deducción de las fórmulas para las derivadas de segundo orden, tercer orden y superiores siguen el mismo principio, pero en lugar de la función original se emplean las derivadas, así para obtener la derivada segunda se emplea la derivada primera, para la tercer la derivada segunda y así sucesivamente.

En función al número de puntos que se toman alrededor del punto a derivar se tienen diferentes tipos de errores: de primer orden si sólo se toma un punto a la derecha o a la izquierda; de segundo orden si se toman dos puntos a ambos lados del punto de derivación y de cuarto orden si se toman 4 puntos: 2 a la izquierda y 2 a la derecha del punto de integración.

Aun cuando para el método de Newton – Raphson sólo necesitamos las fórmulas para las derivadas de primer orden, presentamos a continuación fórmulas para calcular las derivadas hasta de cuarto orden, las mismas que podrán ser empleadas luego, cuando así se requieran.

3.9.1.1. Fórmulas de diferencia central. Error de orden h^2

$$y'_i = \frac{y_{i+1} - y_{i-1}}{2h}$$
$$y''_i = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} \tag{3.16a}$$

$$y''_i = \frac{y_{i+2} - 2y_{i+1} + 2y_{i-1} - y_{i-2}}{2h^3}$$
$$y'''_i = \frac{y_{i+2} - 4y_{i+1} + 6y_i - 4y_{i-1} + y_{i-2}}{h^4} \qquad (3.16b)$$

3.9.1.2. Fórmulas de diferencia central. Error de orden h^4

$$y'_i = \frac{-y_{i+2} + 8y_{i+1} + 8y_{i-1} + y_{i-2}}{12h}$$
$$y''_i = \frac{-y_{i+2} + 16y_{i+1} + 30y_i + 16y_{i-1} - y_{i-2}}{12h^2}$$
$$y'''_i = \frac{-y_{i+3} + 8y_{i+2} - 13y_{i+1} + 13y_{i-1} - 8y_{i-2} + y_{i-3}}{8h^3} \qquad (3\text{-}17)$$
$$y''''_i = \frac{-y_{i+3} + 12y_{i+2} - 39y_{i+1} + 56y_i - 39y_{i-1} - 12y_{i-2} - y_{i-3}}{6h^4}$$

3.9.1.3. Fórmulas de diferencia hacia adelante. Error de orden h

$$y'_i = \frac{y_{i+1} - y_i}{h}$$
$$y''_i = \frac{y_{i+2} - 2y_{i+1} + y_i}{h^2}$$
$$y'''_i = \frac{y_{i+3} - 3y_{i+2} + 3y_{i+1} - y_i}{h^3} \qquad (3.18)$$
$$y''''_i = \frac{y_{i+4} - 4y_{i+3} + 6y_{i+2} - 4y_{i+1} + y_i}{h^4}$$

3.9.1.4. Fórmulas de diferencia hacia adelante. Error de orden h^2

$$y'_i = \frac{-y_{i+2} + 4y_{i+1} - 3y_i}{2h}$$
$$y''_i = \frac{-y_{i+3} + 4y_{i+2} - 5y_{i+1} + 2y_i}{h^2}$$
$$y'''_i = \frac{-3y_{i+4} + 14y_{i+3} - 24y_{i+2} + 18y_{i+1} - 5y_i}{2h^3} \qquad (3.19)$$
$$y''''_i = \frac{-2y_{i+5} + 11y_{i+4} - 24y_{i+3} + 26y_{i+2} - 14y_{i+1} + 3y_i}{h^4}$$

3.9.1.5. Fórmulas de diferencia hacia atrás. Error de orden h

$$y'_i = \frac{y_i - y_{i-1}}{h}$$
$$y''_i = \frac{y_i - 2y_{i-1} + y_{i-2}}{h^2}$$
$$y'''_i = \frac{y_i - 3y_{i-1} + 3y_{i-2} - y_{i-3}}{h^3} \qquad (3.20)$$
$$y''''_i = \frac{y_i - 4y_{i-1} + 6y_{i-2} - 4y_{i-3} + y_{i-4}}{h^4}$$

3.9.1.6. Fórmulas de diferencia hacia atrás. Error de orden h^2

$$
\begin{aligned}
y'_i &= \frac{3y_i - 4y_{i-1} + y_{i-2}}{2h} \\
y''_i &= \frac{2y_i - 5y_{i-1} + 4y_{i-2} - y_{i-3}}{h^2} \\
y'''_i &= \frac{5y_i - 18y_{i-1} + 24y_{i-2} - 14y_{i-3} + 3y_{i-4}}{2h^3} \\
y''''_i &= \frac{3y_i - 14y_{i-1} + 26y_{i-2} - 24y_{i-3} + 11y_{i-4} - 2y_{i-5}}{h^4}
\end{aligned}
\qquad (3.21)
$$

En estas fórmulas "y_i" es el valor de la función en el punto "x_i" ($y_i=f(x_i)$) y "h" es el incremento de la variable independiente ($h=\Delta x$).

En teoría, mientras mayor es el orden de error más exacto es el resultado, no obstante, en la práctica las fórmulas de orden elevado implican un mayor número de operaciones, lo que implica un mayor error de redondeo. Por esta razón en la práctica se suelen emplear las fórmulas cuyo orden son de primer o segundo orden.

Igualmente en teoría, mientras menor sea el valor de "h" más exacto será el resultado obtenido. Una vez más esto no es cierto en la práctica debido a los errores de redondeo.

Tomando en cuenta las anteriores consideraciones y aprovechando el que las funciones a derivar son analítica (y que en consecuencia es posible calcular los valores de "y_i" a ambos lados del punto), emplearemos las fórmulas de diferencia central con un error de segundo orden.

Como incremento "h" emplearemos el valor de la variable independiente multiplicado por $1x10^{-4}$, es decir: $h = x*10^{-4}$, valor que se ha comprobado es lo suficientemente pequeño como para permitir un cálculo preciso de la derivada y que minimiza los errores de redondeo porque no es un valor demasiado pequeño.

El algoritmo para calcular la derivada primera con la fórmula de diferencia central (ecuación 3.16) es el siguiente:

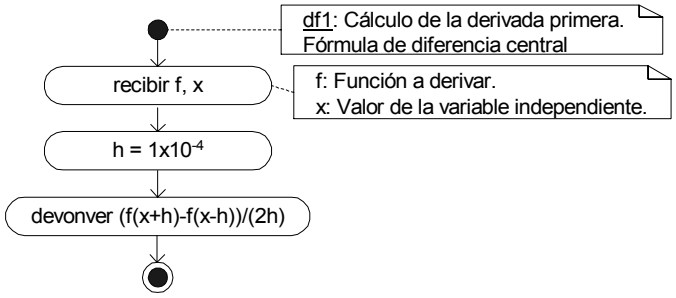


Figura 3.18. Algoritmo para el cálculo de la derivada primera.

Y el código respectivo en *System RPN* es:

```
xNAME df1 ( Cálculo de la derivada primera )
:: ( Datos en la pila: f: función a derivar; x: valor de la variable)
  CK2&Dispatch
  # 81
  :: ( f=3; x=2; h=1 )
    % 0.0001
    ' NULLLAM THREE NDUPN DOBIND
    2GETLAM 1GETLAM %+ 3GETLAM EVAL
```

```
    2GETLAM 1GETLAM %- 3GETLAM EVAL %-
    %2 1GETLAM %* %/
    ABND
  ;
;
```

Así por ejemplo la derivada de la ecuación cúbica:

$$f(x) = x^3 + 2x^2 + 3x + 4 = 0$$

Para x=3.2 se calcula con:

```
« → X  « X 3. ^ 2. X SQ * + 3. X * + 4. +  » » 3.2 df1
```

Con lo que se obtiene:

```
46.52
```

Que es el valor exacto de la derivada.

3.9.2. Algoritmo del método de Newton - Raphson con derivación numérica

El algoritmo para el método de Newton – Raphson con derivación numérica es el siguiente:

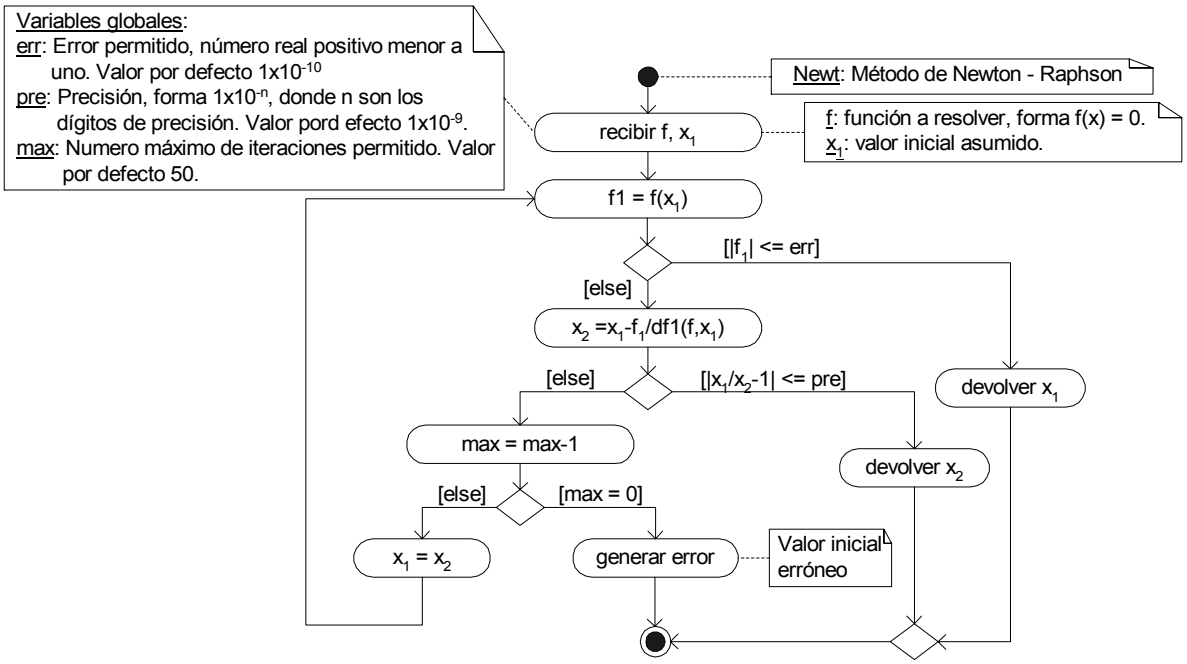


Figura 3.19. Algoritmo del método de Newton-Raphson. Derivada numérica.

3.9.3. Código

El código elaborado en *System RPN* siguiendo el algoritmo es:

```
xNAME Newt ( Método de Newton - Raphson )
:: ( parámetros f: función, forma fx=0, x1: valor asumido )
  CK2&Dispatch
  # 81
  :: ( f=7; x1=6; err=5; pre=4; max=3; x2=2; f1=1 )
    ' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
    ' ID pre @ ?SKIP % 1E-9  ( si no existe "pre" se emplea 1E-9 )
    ' ID max @ ?SKIP % 50    ( si no existe "max" se emplea 50 )
```

```
    %0 %0
    ' NULLLAM SEVEN NDUPN DOBIND
    ::
      BEGIN
        6GETLAM 7GETLAM EVAL 1PUTLAM ( f1=f[x1] )
        1GETLAM %ABS 5GETLAM %< ( |f1|<err )
        IT :: 6GETLAM 2RDROP ; ( devolver x1 )
        6GETLAM 1GETLAM 7GETLAM 6GETLAM xdf1 %/ %- 2PUTLAM ( x2=x1-f1/df )
        "x2:" 2GETLAM a%>$ &$ DispCoord1 ( mostrar x2 )
        6GETLAM 2GETLAM %/ %1- %ABS 4GETLAM %< ( |x1/x2-1|<pre )
        IT :: 2GETLAM 2RDROP ; ( devolver x2 )
        3GETLAM %1- 3PUTLAM ( max=max-1)
        3GETLAM %0= ( max==0)
        IT :: ABND #A01 DO#EXIT ; ( valor inicial erróneo )
        2GETLAM 6PUTLAM ( x1=x2)
      AGAIN
    ;
    ABND
  ;
;
```

3.9.4. Ejemplos

3.9.4.1. Resolución de una ecuación no lineal con una incógnita

Como primer ejemplo volveremos a resolver el ejemplo 3.8.4.1:

$$f(y) = y - \left(52 + 3\sqrt{y} - 8y^{0.8}\right)^{0.36} = 0$$

El programa que resuelve el problema es:

```
« 1E-7 'pre' STO
  « → Y « Y 52 3 Y √ * + 8 Y 0.8^ * - 0.36 ^ - » »
  1.1  Newt
  "Y" →TAG
  'pre' PURGE
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
Y: 3.60967088614
s: .9648
```

Que es el mismo resultado obtenido con la derivada analítica aunque el tiempo requerido es ligeramente superior.

3.9.4.2. Resolución de la ecuación de Benedict - Webb - Rubin

Como segundo ejemplo volveremos a resolver el problema 3.7.5.3:

$$f(V) = \frac{RT}{V} + \frac{B_0 RT - A_0 - C_0 T^{-2}}{V^2} + \frac{bRT - a}{V^3} + \frac{a\alpha}{V^6} + \frac{c}{V^3 T^2}\left(1 + \frac{\gamma}{V^2}\right)e^{\frac{-\gamma}{V^2}} - P = 0$$

$$R = 0.08206 \frac{atm.l}{K.gmol}$$

El programa que resuelve el problema es:

```
« 0.08206 7.11671 1.44373E1 1.09131E-1 1.77813E-1
  1.51276E6 3.31935E6 2.81086E-3 6.66849E-2 30 600
```

```
  → R a A0 b B0 c C0 α γ P T
  «
    « → V
      «
        R T * V / B0 R * T * A0 - C0 T -2 ^ * - V SQ / +
        b R * T * a - V 3 ^ / + a α * V 6 ^ / +
        c V 3 ^ T SQ * / 1 γ V SQ / + * γ NEG V SQ / EXP * + P -
      »
    »
    R T * P / DUP 1.2 * Newt
    "V" →TAG
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

V: 1.30981987739

s: 2.52

Que es el mismo resultado obtenido anteriormente y básicamente en el mismo tiempo, observe también que el programa es también el mismo excepto por el nombre del método.

3.9.4.3. Resolución de un sistema de ecuaciones no lineales

Como tercer ejemplo volveremos a resolver el ejemplo 3.7.5.2:

$$f(Y_{A1}) = L_N + V_1 - M = 0$$
$$Y_{B1} = 2Y_{A1} + Y_{A1}^2 + 3.44Y_{A1}^3$$
$$X_{BN} = 2.2 + 5.1X_{AN} - 7.2X_{AN}^{2.2}$$
$$\left.\begin{aligned} L_N X_{AN} + V_1 Y_{A1} &= C_{A0} \\ L_N X_{BN} + V_1 Y_{B1} &= C_{B0} \end{aligned}\right\} \; L_N = \frac{C_{A0}Y_{B1} - C_{B0}Y_{A1}}{X_{AN}Y_{B1} - X_{BN}Y_{A1}};\; V_1 = \frac{C_{A0} - L_N X_{AN}}{Y_{A1}}$$
$$X_{AN} = 0.12$$
$$M = 32.1$$
$$C_{A0} = 20.2$$
$$C_{B0} = 3.42$$

El programa que resuelve el problema es:

```
« 0.12 32.1 20.2 3.42 0 0 0 0 0
  → XAN M CA0 CB0 YA1 YB1 ABN LN V1
  «
    2.2 5.1 XAN * + 7.2 XAN 2.2 ^ * - 'XBN' STO
    « 'YA1' STO
      2 YA1 * YA1 SQ + 3.44 YA1 3 ^ * + 'YB1' STO
      CA0 YB1 * CB0 YA1 * - XAN YB1 * XBN YA1 * - /  'LN' STO
      CA0 LN XAN * - YA1 / 'V1' STO
      LN V1 + M -
    »
    XAN 0.5 * Newt
    "YA1" →TAG
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene

YA1: .411748600835

s: 3.7303

Que es la misma solución obtenida con el método del Secante aunque el tiempo requerido es un tanto mayor.

3.9.5. Ejercicios

29. Resuelva el ejercicio 22 con el método de *Newton* con diferenciación numérica.

30. Resuelva el ejercicio 23 con el método de *Newton* con diferenciación numérica.

31. Calcule, con 6 dígitos de precisión, el volumen del metano a 30 atm y 350 K, empleando la ecuación de estado de Soave – Redlich – Kwong:

$$P = \frac{RT}{V-b} - \frac{\Omega_a}{\Omega_b}\frac{RTbF}{V(V+b)}$$

$$a = \frac{\Omega_a R T_c}{P_c}$$

$$\Omega_a = \left[9\left(2^{1/3}-1\right)\right]^{-1}$$

$$\Omega_b = \frac{2^{1/3}-1}{3}$$

$$F = \frac{1}{T_r}\left[1+\left(0.480+1.574\omega-0.176\omega^2\right)\left(1-T_r^{0.5}\right)\right]^2$$

$$T_r = \frac{T}{T_c}$$

$$R = 0.08206\frac{l.atm}{K.gmol}$$

Las constantes para el metano son: Tc=190.6 K; Pc=45.4 atm; w=0.008.

4. ECUACIONES POLINOMIALES

Con los métodos estudiados en el anterior capítulo se puede encontrar una de las soluciones de la ecuación polinomial:

$$P_n(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + a_4 x^{n-3} + ... + a_n x + a_{n+1} = 0 \quad (1)$$

Sin embargo, para resolver ecuaciones polinomiales se cuenta con métodos específicos que nos permiten encontrar no sólo todas las soluciones reales, sino también las soluciones imaginarias. En ocasiones es necesario calcular estas soluciones, pues con las mismas se pueden resolver problemas reales.

4.1. PROOT

La calculadora *HP* cuenta con el programa *PROOT*, con el cual se pueden encontrar las soluciones reales e imaginarias de ecuación polinomiales de enésimo grado.

Para resolver una ecuación polinomial con *PROOT* simplemente se coloca en la pila un vector con los coeficientes y se ejecuta *PROOT* (escribiendo *PROOT* o siguiendo el camino: *ARITH => POLY => PROOT*).

Por ejemplo para resolver la ecuación:

$$P_5(x) = 5x^5 - 3x^4 - 10x^3 + 10x^2 + 44x - 48 = 0$$

Escribimos:

[5 -3 -10 10 44 -48] PROOT

Con lo que se obtiene:

[(1.04205581855,0.) (-1.49509786786,-.987870288052) (-1.49509786786, .987870288052) (1.27406995858,1.11607542411) (1.27406995858,-1.11607542411)]

Que son las cinco soluciones de la ecuación:

x_1 = *1.04205581855*

x_2 = *-1.49509786786 - 0.9878702880521*

x_3 = *-1.49509786786 + 0.9878702880521*

x_4 = *1.27406995858 + 1.11607542411i*

x_5 = *1.27406995858 - 1.11607542411i*

4.1.1. Ejemplos

4.1.1.1. Resolución de una ecuación polinomial de sexto grado

Como primer ejemplo encontraremos las soluciones de la siguiente ecuación polinomial:

$$P_6(x) = 693x^6 - 945x^4 + 315x^2 - 15 = 0$$

El programa que resuelve el problema es el siguiente:

```
« [ 693 0 -945 0 315 0 -15 ]
  PROOT
  AXL
  { x1 x2 x3 x4 x5 x6 }
  →TAG LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: -.238619186083
x2: .238619186083
x3: .661209386466
x4: -.661209386466
x5: .932469514203
x6: -.932469514203
s: 2.9386
```

Que son las 6 soluciones y el tiempo empleado en obtenerlas.

En la HP48 no existe el comando AXL, por lo que para reemplazar el mismo deberá escribir el siguiente programa. Lamentablemente en *DEBUG4x,* debido a un error de la aplicación, no se pueden escribir direcciones de memoria (con *PTR* y *ROMPTR*), por lo que este programa debe ser programado y depurado directamente en el emulador (o en la calculadora) empleando *JAZZ.*

```
::
  CK1&Dispatch
  FIVE
  ::
    PTR 1C973
    OVER TYPELIST?
    ITE
    ::
      COERCE 1LAMBIND
      1GETLAM #1+ ONE DO
        PTR 1C973 PTR 1D02C 1GETLAM UNROLL
      LOOP
      1GETLAM ROMPTR C3 15
      ABND
    ;
     PTR 1D02C
  ;
  FOUR
  ::
    PTR 1D0AB PTR 1C973
    %2 %= ITE
    ::
      COERCE2 2DUP #* ( n=columans=3; m=filas=2; ne=N° elementos=1 )
      ' NULLLAM THREE NDUPN DOBIND
      3GETLAM #1+ ONE DO
        2GETLAM {}N 1GETLAM 2GETLAM #- DUP 1PUTLAM INDEX@ #+ UNROLL
      LOOP
      3GETLAM {}N
      ABND
    ;
    ::
      COERCE {}N
    ;
  ;
;
```

Una vez codificado el programa, guarde el mismo con el nombre *AXL* y si tiene experiencia con *JAZZ* cree una librería con este programa de manera que pueda ser empleado en cualquier directorio.

4.1.1.2. Resolución de una ecuación polinomial de octavo grado

Como segundo ejemplo resolveremos la ecuación polinomial:

$$P_8(x) = x^8 + x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$$

El programa que resuelve el problema es el siguiente:

```
« [ 1 1 -9 13 21 -125 77 111 -90 ]
  PROOT
  AXL
  { x1 x2 x3 x4 x5 x6 x7 x8 }
  →TAG LIST→ DROP
»
```

Evaluando el programa con *TEVAL* se obtiene:

```
x1: (.999999866892,0.)
x2: (1.00000013311,0.)
x3: (-1.,0.)
x4: (2.,0.)
x5: (-2.99999917604,-6.04713663093E-7)
x6: (-2.99999917604,6.04713663093E-7)
x7: (.999999176041,1.99999835208)
x8: (.999999176041,-1.99999835208)
s: 7.9816
```

Que son las soluciones (4 reales y 4 imaginarias) y el tiempo empleado en obtenerlas.

4.1.2. Ejercicios

1. Elabore un programa que resuelva la siguiente ecuación polinomial con PROOT:

$$P_5(x) = x^5 - 5x^4 - 7x^3 + 12x^2 + 34x - 48 = 0$$

2. Elabore un programa que resuelva la siguiente ecuación polinomial con PROOT:

$$P_8(x) = x^8 - 7x^7 + 11x^6 + 41x^5 - 183x^4 + 231x^3 - 21x^2 - 265x - 150 = 0$$

4.2. División sintética

En la mayoría de los métodos que resuelven ecuaciones polinomiales es necesario evaluar el polinomio $(P_n(x))$ y en ocasiones también su derivada $(P'_n(x))$.

4.2.1. División sintética de un polinomio ente $(x-x_1)$

La forma más eficiente de evaluar un polinomio y su derivada es mediante división sintética. Para ello se divide el polinomio entre $(x-x_1)$, siendo "x_1" el valor para el cual se quiere evaluar la función, la derivada de la función se obtiene llevando a cabo una segunda división sintética con los coeficientes resultantes de la primera.

Así por ejemplo para evaluar el polinomio y su derivada:

$$P_3(x) = x^3 + x^2 - 3x - 3 = 0$$

En el punto $x_1=2$, realizamos la división sintética entre $(x-2)$:

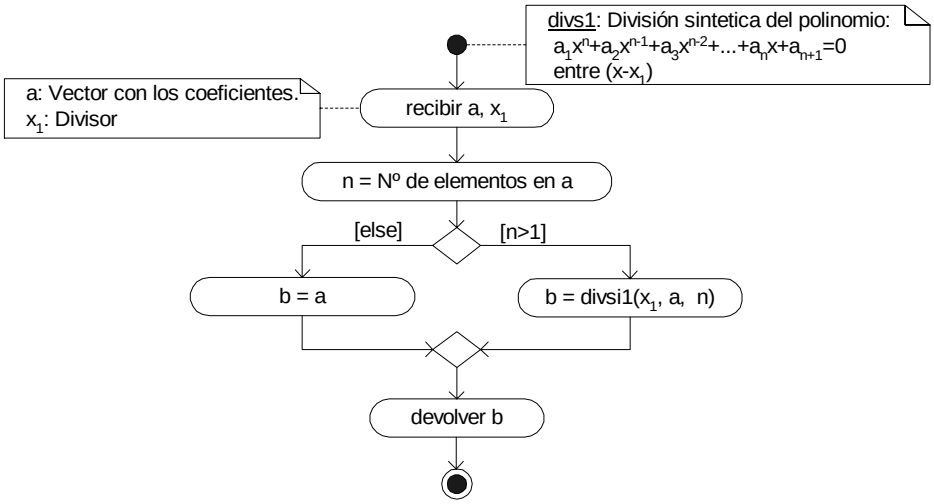
$x_1 = 2$ | 1 1 -3 -3
2 6 6
1 3 3 [3] ← valor de la función: f(2)
2 10
1 5 [13] ← derivada de la función: f'(2)

Como puede observar en este ejemplo, si *"a"* son los coeficientes del polinomio original, los coeficientes resultantes de la división sintética: *"b"* se calculan con:

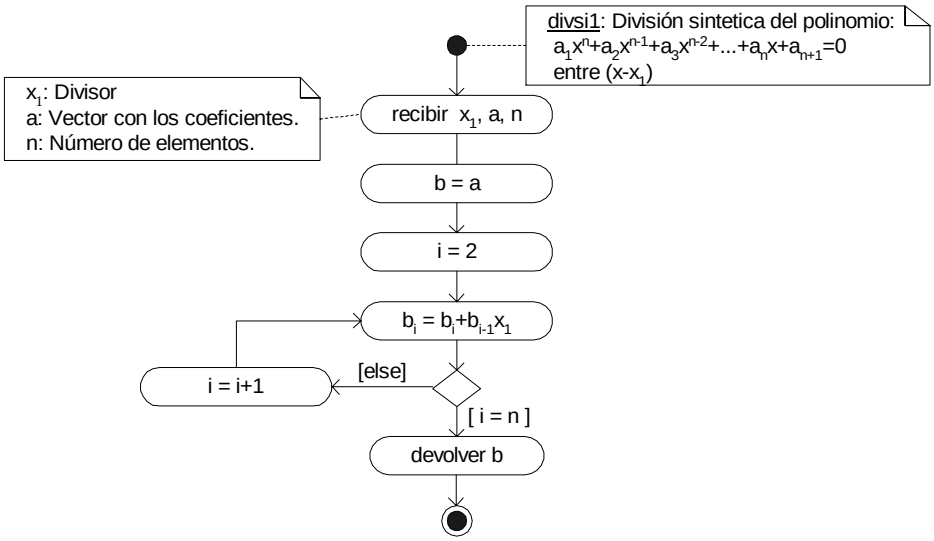
$$b_i = a_i + b_{i-1}x_1 \quad \begin{cases} b_1 = a_1 \\ i = 2 \rightarrow n+1 \\ resíduo = b_{n+1} \end{cases} \qquad (2)$$

4.2.1.1. Algoritmo

El algoritmo para la división sintética entre $(x-x_1)$ es el siguiente:

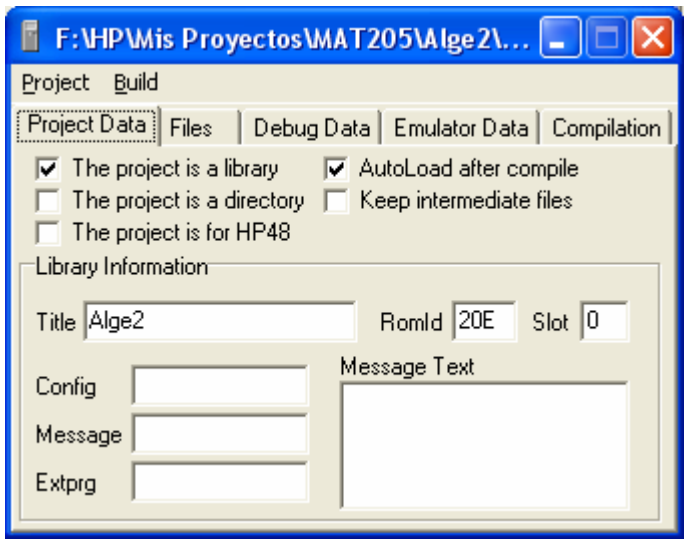


Donde el módulo ***divsi1*** es:



4.2.1.2. Código

Para escribir el código cree un nuevo proyecto con el nombre ***Alge2:***



Añada el archivo ***Alge2.S*** y en el mismo escriba el siguiente código:

```
NULLNAME divsi1 ( Módulo interno para la división sintética entre x-x1)
:: ( Datos en la pila: x1=divisor; a=coeficientes; n=N° de elementos)
  #1+ TWO DO
    INDEX@ PULLREALEL SWAP
    INDEX@ #1- PULLREALEL
    4PICK %* ROT %+
    INDEX@ PUTREALEL ( bi=bi+b[i-1]x1)
  LOOP
  SWAP DROP
;

xNAME divs1 ( División sintética entre (x-x1)
:: ( Datos en la pila: a=coeficientes; x1= divisor )
  CK2&Dispatch
  # 41
  ::
    SWAP TOTEMPOB
    DUP MDIMSDROP DUP #1 #>
    ITE divsi1 :: DROP SWAP DROP  ;
  ;
;
```

Para probar el programa lleve a cabo la división sintética realizada manualmente:

[1 1 -3 -3] 2 divs1

Con lo que se obtiene:

[1. 3. 3. 3.]

Que es el resultado de la primera división sintética. El resultado de la segunda división sintética se obtiene de la siguiente manera:

[1. 3. 3. 3.] 4 COL- DROP 2 divs1

Resultando:

[1. 5. 13.]

4.2.1.3. Ejercicios

3. Divida la siguiente ecuación polinomial entre (x-4).

$$P_7(x) = 7x^7 + 3x^5 - 8x^4 + 3x - 9 = 0$$

4. Divida la siguiente ecuación polinomial entre (x_1=5).

$$P_{10}(x) = 12x^{10} - 8x^9 + 6x^7 - 3x^3 + 5x^2 - 3x - 10 = 0$$

5. Divida la siguiente ecuación polinomial entre (x+4).

$$P_6(x) = x^6 - 9x^5 + 12x^4 - 3x^3 - 6x^2 + x - 7 = 0$$

6. Divida la siguiente ecuación polinomial entre (x_1=-7).

$$P_9(x) = 5x^9 - 8x^7 + 4x^6 - 2x^5 + x^4 - 6x^3 + 3x^2 - 2x - 80 = 0$$

7. Divida la siguiente ecuación polinomial entre (x-10).

$$P_{20}(x) = 2x^{20} + 3x^{17} - 4x^{15} + 12x^{12} - 9x^8 + 6x^6 - 2x^4 - 4x^3 + 3x^2 - x + 9 = 0$$

4.2.2. División sintética entre x^2-rx-s

En algunos métodos como *Bairstow*, se emplea la división sintética entre la ecuación de segundo grado x^2-rx-s. Al igual que en el anterior caso, una segunda aplicación nos permite obtener las derivadas parciales.

Por ejemplo para dividir el polinomio:

$$P_4(x) = x^4 - 1.1x^3 + 2.3x^2 + 0.5x + 3.3 = 0$$

Entre la ecuación de segundo grado: x^2+x+1, y luego una vez más el resultado procedemos de la siguiente manera:

	1	-1.1	2.3	0.5	3.3
$r=-1$		-1.0	2.1	-3.4	0.8
$s=-1$			-1.0	2.1	-3.4
	1	-2.1	3.4	-0.8	0.7
		-1.0	3.1	-5.5	
			-1.0	3.1	
	1	-3.1	5.5	-3.2	

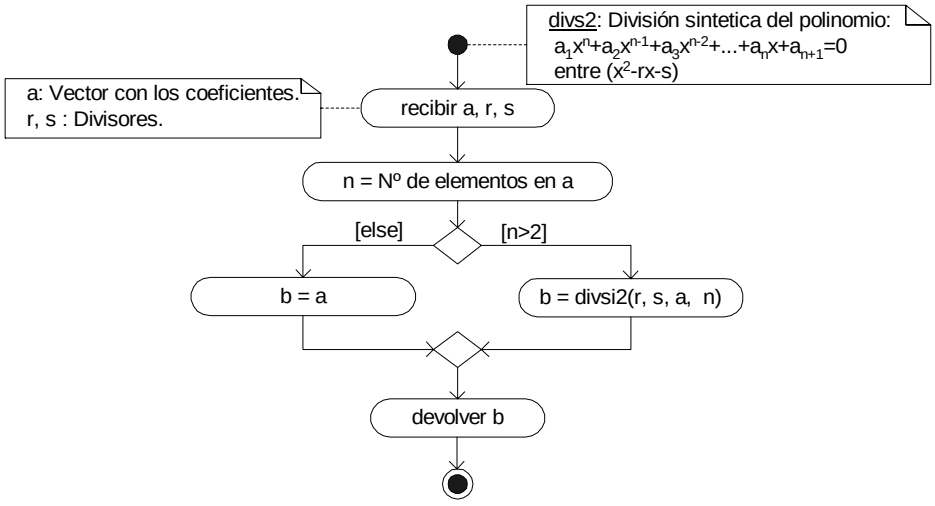
Si denominamos *"b"* a los coeficientes resultantes de la primera división, el residuo está dado por: $b_n x + b_{n+1}$. Y denominado *"c"* a los coeficientes resultantes de la segunda división, entonces $c_{i-1} = \partial bi/\partial r$ y $c_{i-2} = \partial bi/\partial s$.

La ecuación que nos permite calcular los coeficientes de la división sintética es la siguiente:

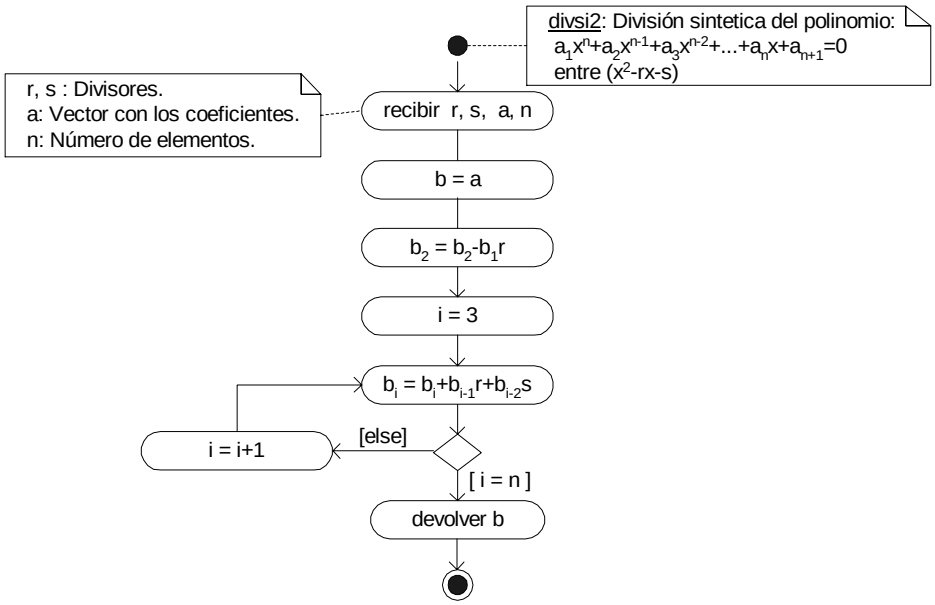
$$b_i = a_i + b_{i-1}r + b_{i-2}s \quad \begin{cases} i = 3 \rightarrow \text{n+1} \\ \text{b}_1 = a_1 \\ \text{b}_2 = a_2 + \text{c}_1 r \end{cases} \qquad resíduo = b_n x + b_{n+1} \tag{3}$$

4.2.3. Algoritmo

El algoritmo para resolver la ecuación (3) se presenta en la siguiente página.



Y el algoritmo del módulo interno ***divsi2*** se muestra en la siguiente página.



4.2.4. Código

El código elaborado siguiendo los algoritmos y que debe ser añadido al archivo ***Alge2.s***, es el siguiente:

```
NULLNAME divsi2 ( Módulo interno para la división sintética entre
                  x^2-rx-x)
:: ( Datos en la pila: r,s= divisores; a=coeficientes; n=N° de elemntos)
  4UNROLL TWO PULLREALEL
  SWAP ONE PULLREALEL
  5PICK %* ROT %+
  TWO PUTREALEL 4ROLL ( b2=b2+b1*r)
  #1+ THREE DO
    INDEX@ PULLREALEL
    SWAP INDEX@ #1- PULLREALEL
    5PICK %* ROT %+
    SWAP INDEX@ #2- PULLREALEL
    4PICK %* ROT %+
```

```
    INDEX@ PUTREALEL  ( bi = bi+b[i-1]r+b[i-2]s)
  LOOP
  UNROT 2DROP
;

xNAME divs2 ( División sintética entre x^2-rx-s)
::  ( Datos en la pila: a=coeficientes; r,s= divisores)
  CK3&Dispatch
  # 411
  ::
    ROT TOTEMPOB DUP
    MDIMSDROP DUP #2 #>
    ITE divsi2 :: DROP UNROT 2DROP ;
  ;
;
```

Por ejemplo la primera división del ejemplo manual se resuelve de la siguiente manera:

[1 -1.1 2.3 0.5 3.3] -1 -1 divs2

Con lo que se obtiene:

[1. -2.1 3.4 -.8 .7]

Con este resultado se puede realizar la segunda división sintética:

[1. -2.1 3.4 -.8 .7] 5 COL- DROP -1 -1 divs2

Resultando:

[1. -3.1 5.5 -3.2]

4.2.4.1. Ejercicios

8. Divida la siguiente ecuación polinomial entre (x^2-3-2).

$$P_7(x) = 7x^7 + 3x^5 - 8x^4 + 3x - 9 = 0$$

9. Divida la siguiente ecuación polinomial entre ($x^2+10x+5$).

$$P_{10}(x) = 12x^{10} - 8x^9 + 6x^7 - 3x^3 + 5x^2 - 3x - 10 = 0$$

10. Divida la siguiente ecuación polinomial entre (x^2-8x+7).

$$P_6(x) = x^6 - 9x^5 + 12x^4 - 3x^3 - 6x^2 + x - 7 = 0$$

11. Divida la siguiente ecuación polinomial entre ($x^2-3x+20$).

$$P_9(x) = 5x^9 - 8x^7 + 4x^6 - 2x^5 + x^4 - 6x^3 + 3x^2 - 2x - 80 = 0$$

12. Divida la siguiente ecuación polinomial entre ($x^2+2x+10$).

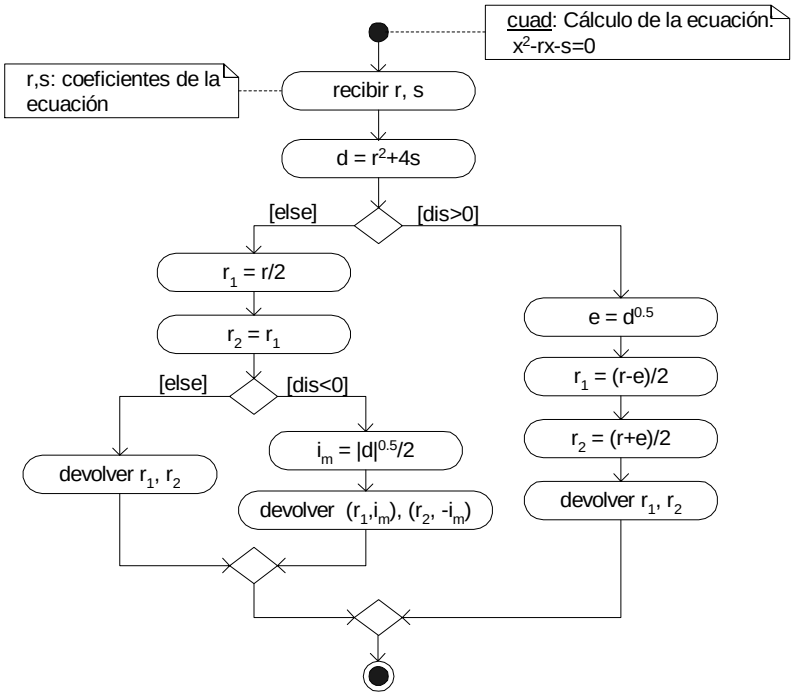
$$P_{20}(x) = 2x^{20} + 3x^{17} - 4x^{15} + 12x^{12} - 9x^8 + 6x^6 - 2x^4 - 4x^3 + 3x^2 - x + 9 = 0$$

4.3. Raíces de la ecuación cuadrática $x^2-rx-s=0$

Con frecuencia al encontrar las raíces de un polinomio, es necesario resolver la ecuación cuadrática *$x^2-rx-s=0$*.

4.3.1. Algoritmo

El algoritmo que resuelve la ecuación cuadrática *$x^2-rx-s=0$* se presenta en la siguiente página.



4.3.2. Código

El código elaborado en base al algoritmo es el siguiente:

```
xNAME cuad ( raíces de la ecuación x^2-rx-s=0 )
:: ( datos en la pila: r,s: coeficientes de la ecuación )
  CK2&Dispatch
  # 11
  ::
    OVER %2 %^ %4 ROT %* %+ ( d )
    DUP %0>
    ITE ( d>0 )
    ::
      %SQRT DUP
      3PICK SWAP %- %2 %/ ( r1 )
      UNROT %+ %2 %/ ( r2 )
    ;
    :: ( d<=0 )
      SWAP %2 %/ DUP ( r1, r2 )
      ROT DUP %0<
      ITE ( d < 0 )
      ::
        %ABS %SQRT %2 %/  DUP ( im )
        UNROT %>C% ( r1 )
        UNROT %CHS %>C% ( r2 )
      ;
      DROP
    ;
  ;
;
```

Por ejemplo para resolver la ecuación $x^2+2x+3=0$ escribimos el siguiente programa:

```
« -2 -3 cuad
```

```
  2 →LIST { x1 x2 } →TAG
  LIST→ DROP
»
```

Evaluando el programa con ***TEVAL*** se obtiene

```
x1: (-1.,1.41421356238)
x2: (-1.,-1.41421356238)
s: .1567
```

Que son las dos soluciones (imaginarias) y el tiempo empleado en obtenerlas.

Como otro ejemplo resolvamos la ecuación x^2-6x+5:

```
« 6 -5 cuad
  2 →LIST { x1 x2 } →TAG
  LIST→ DROP
»
```

Y evaluando con ***TEVAL*** se obtiene:

```
x1: 1.
x2: 5.
s: .1622
```

Como otro ejemplo resolvamos la ecuación $x^2-14+49=0$:

```
« 14 -49 cuad
  2 →LIST { x1 x2 } →TAG
  LIST→ DROP
»
```

Y evaluando con ***TEVAL*** se obtiene:

```
x1: 7.
x2: 7.
s: .1548
```

4.3.3. Ejercicios

13. Resuelva la ecuación $x^2+7x+12=0$.
14. Resuelva la ecuación $x^2+3x+20=0$.
15. Resuelva la ecuación $x^2-24x+144=0$.
16. Resuelva la ecuación $x^2-2x+14=0$.
17. Resuelva la ecuación $x^2+2x+10=0$.
18. Resuelva la ecuación $x^2+7x+20=0$.

4.4. Método QD

En este método los valores iniciales se calculan con las ecuaciones:

$$p_1 = -\frac{a_2}{a_1} \qquad (4)$$
$$p_i = 0 \quad \{i = 2 \rightarrow n$$

$$d_i = \frac{a_{i+2}}{a_{i+1}} \quad \{i = 1 \rightarrow n-1 \qquad (5)$$

En estas ecuaciones *"n"* es el grado del polinomio.

Una vez calculados los valores iniciales, los sucesivos valores se calculan con las ecuaciones:

$$\begin{aligned} q_1 &= d_1 + p_1 \\ q_i &= d_i - d_{i-1} + p_i \quad \{i = 2 \rightarrow n-1 \\ q_n &= p_n - d_{n-1} \end{aligned} \tag{6}$$

$$e_i = \frac{q_{i+1}}{q_i} \mathrm{d_i} \quad \{i = 1 \rightarrow n-1 \tag{7}$$

Si los valores de e_i son cero o casi cero el proceso concluye, caso contrario se realiza un cambio de variables: $p=q;\ d=e$ y se vuelven a evaluar las ecuaciones (6) y (7), repitiéndose el proceso hasta que los valores de e_i se hacen cero o casi cero.

Cuando el proceso concluye, las soluciones del polinomio son los valores de q_i.

Si existen raíces imaginarias, algunos de los valores de e_i no se aproximan a cero, sino que fluctúan en signo, en esos casos las soluciones imaginarias se encuentran resolviendo la ecuación cuadrática $x^2-rx-s=0$, donde los valores de *"r"* y *"s"* se calculan con las siguientes expresiones:

$$\begin{aligned} r &= q_i + q_{i+1} \\ w &= -p_i q_{i+1} \end{aligned} \tag{8}$$

Donde *"i"* es el índice del elemento e_i que fluctúa en signo.

La principal ventaja del método QD es la de no requerir valores iniciales, la principal desventaja es la lentitud con la que converge hacia el resultado. Por ello el método QD generalmente es empleado para encontrar valores iniciales adecuados y luego se continúa la búsqueda con otros métodos que convergen más rápidamente, como los métodos de *Newton* y *Bairstow*.

Como se puede deducir de las ecuaciones (4) y (5), si alguno de los coeficientes del polinomio es cero, el método "QD" no puede ser aplicado pues se produciría una división entre cero, en esos casos se debe efectuar un cambio de variable, siendo conveniente cambiar la variable "x" por "x-1". Para ello el polinomio se divide entre "x-1" y el polinomio resultante nuevamente entre "x-1" y así sucesivamente, siendo los residuos de cada división los coeficientes del nuevo polinomio. Finalmente una vez calculados los resultados simplemente se le suma 1 para obtener los resultados correctos.

Por ejemplo para resolver la siguiente ecuación polinomial:

$$P_4(x) = x^4 - 3x^2 + x - 5 = 0$$

Realizamos las siguientes divisiones sintéticas:

```
   [ 1 0 -3 1 -5]
1 divs1 5 COL- SWAP
1 divs1 4 COL- SWAP
1 divs1 3 COL- SWAP
1 divs1 2 COL- SWAP
        V→
```

Obtenemos:

```
-6
-1
 3
 4
```

1

Que son los coeficientes de la ecuación:

$$P_4(y) = y^4 + 4y^3 + 3y^2 - y - 6 = 0$$

Donde *"y"* es *"x-1"*.

Otra situación donde es necesario recurrir a un cambio de variable ocurre cuando los coeficientes están igualmente espaciados. En esos casos algunos de los valores de "q" se hacen cero o casi cero, dando lugar a resultados erróneos.

Como ejemplo apliquemos el método "QD" al polinomio:

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0$$

Los valores iniciales calculados con las ecuaciones (4) y (5) son:

p_1	p_2	p_3	P_4
2	0	0	0
d_1	d_2	d_3	
-0.625	-0.2	-0.03125	

Y los valores de las siguientes 6 iteraciones (calculados con las ecuaciones 6 y 7) son:

q_1	q_2	q_3	q_4
1.375	0.425	0.16875	0.03125
1.18181818182	0.538770053476	0.242374727669	0.037037037037
1.09375	0.591113523573	0.277215128112	3.79213483146E-2
1.04615384615	0.621955821956	0.293848015156	3.80423167343E-2
1.01785714285	0.6423370319	0.301747847627	3.80579776158E-2
0.999999999993	0.656475748194	0.30546429896	3.80599528461E-2
e_1	e_2	e_3	
-.193181818182	-7.94117647058E-2	-5.78703703703E-3	
-8.80681818182E-2	-3.57247117209E-2	-8.84311277568E-4	
-4.75961538462E-2	-1.67538554635E-2	-1.20968419738E-4	
-2.82967032968E-2	-7.91549335238E-3	-1.56608814801E-5	
-1.78571428573E-2	-3.71842656327E-3	-1.97523025102E-6	
-1.17227812179E-2	-1.73021862043E-3	-2.4610787732E-7	

Como se puede observar todos los valores de *"e"* convergen hacia cero, por lo tanto en este caso todas las soluciones son reales y los valores aproximados de las soluciones son los valores de *"q"*, es decir x_1*=0.999999999993;* x_2*=0.656475748194;* x_3*=0.30546429896 y* x_4*=3.80599528461E-2.*

Como ejemplo de un polinomio que tiene soluciones imaginarias apliquemos el método "QD" a la siguiente ecuación:

$$P_4(x) = x^4 - 6x^3 + 12x^2 - 19x + 12 = 0$$

Los valores iniciales son:

p_1	p_2	p_3	P_4
6.	0.	0.	0.
d_1	d_2	d_3	
-2.	-1.58333333333	-.631578947368	

Y los valores de las siguientes 7 iteraciones son:

q_1	q_2	q_3	q_4
4.	.41666666667	.951754385962	.631578947368
3.79166666666	-2.99166666661	4.14930875571	1.05069124424

3.95604395604	1.86011203285	-.97297518036	1.15681919147
4.03333333333	-.84099526066	1.77702359698	1.03063833035
4.01721763085	4.71924960568	-3.69392317794	.95745594141
3.99828561632	.39859610746	.62669363684	.976424639383
e_1	**e_2**	**e_3**	
-.208333333336	-3.61666666662	-.419112296872	
.164377289376	5.01615598884	-.106127947232	
7.72893772903E-2	-2.62381791622	.126180861123	
-1.61157024794E-2	5.54412916386	7.31823889401E-2	
-1.89320145335E-2	-4.33958551275	-1.89686979728E-2	
-1.88736574211E-3	-6.82292319585	-.029554319669	

Como se puede observar, en este caso los valores de *"e_2"* no se aproximan a cero, por lo tanto las soluciones x_2 y x_3 son imaginarias y se calculan con la ecuación cuadrática $x^2-rx-s=0$, donde los valores aproximados de *"r"* y *"s"* son:

$$r = q_2+q_3 = 0.39859610746+0.62669363684 = 1.0252897443$$

$$s = -p2*q3 = -4.71924960568*0.62669363684 = -2.95752369854$$

Entonces con el programa *"cuad"* se pueden calcular las dos soluciones imaginarias:

```
1.0252897443 -2.95752369854 cuad
```

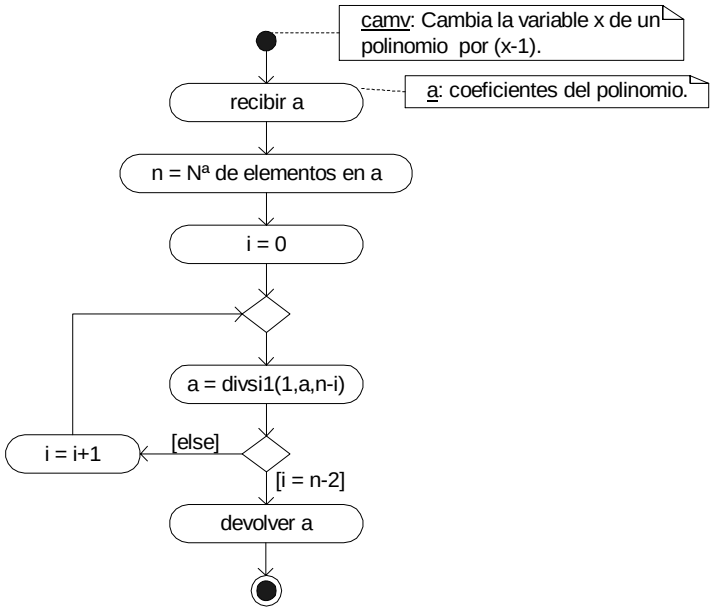
Con lo que se obtiene:

```
(.51264487215,1.64155990862) (.51264487215,-1.64155990862)
```

Por lo tanto las cuatro soluciones aproximadas son: $x_1 = 3.99828561632$, $x_2 = 0.51264487215 + 1.64155990862i$; $x_3 = 0.51264487215 - 1.64155990862i$; $x_4 = 0.976424639383$.

4.4.1. Cambio de variable

Como se mencionó anteriormente existen al menos dos ocasiones donde es necesario llevar a cabo un cambio de variable, razón por la cual es conveniente contar con un programa para este fin. El algoritmo de dicho programa es el siguiente:



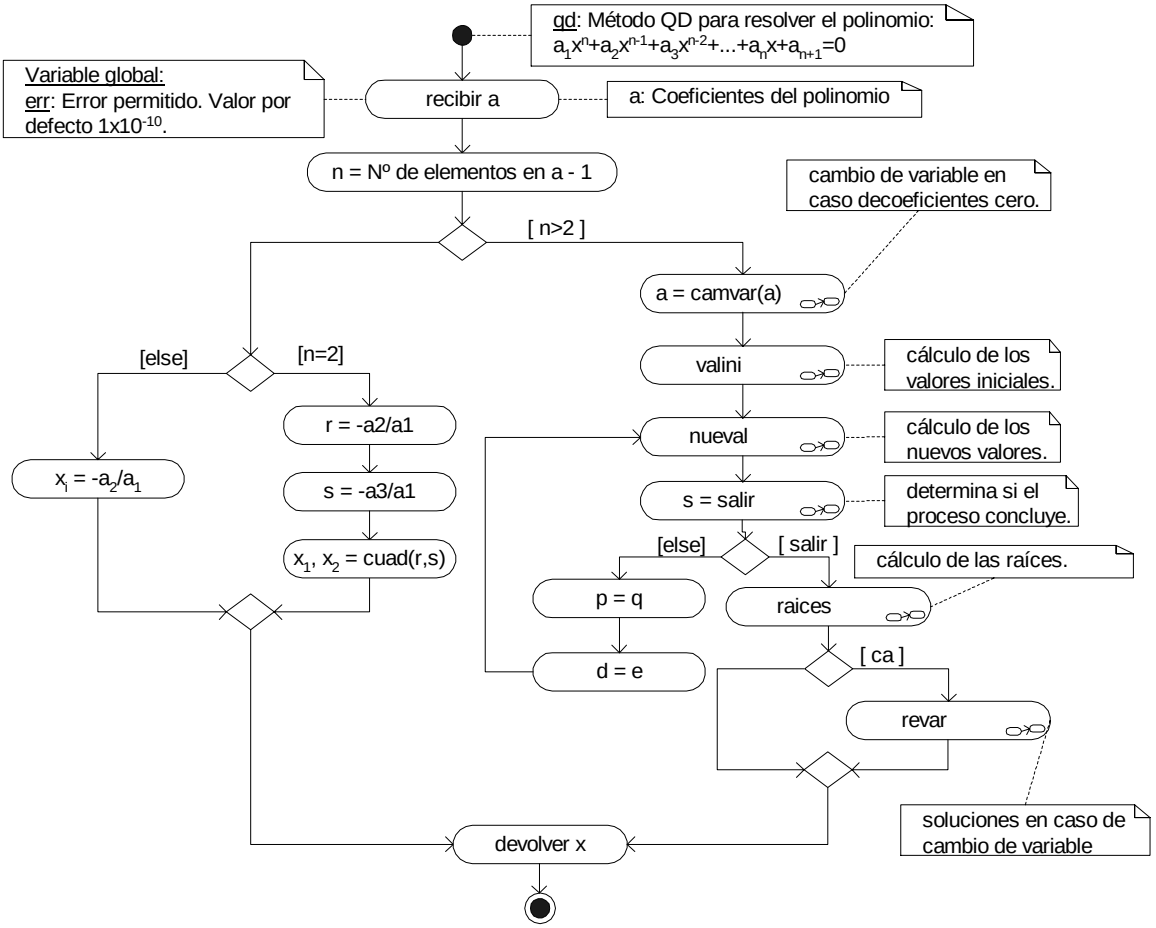
Y el código respectivo el siguiente:

```
xNAME camv ( cambio la variable x por x-1 en un polinomio )
:: ( dato en la pila: vector con los coeficientes )
  CK1&Dispatch
  FOUR
  ::
    DUP MDIMSDROP DUP UNROT #1-
    ZERO DO
      %1 SWAP 3PICK INDEX@ #- divsi1
    LOOP
    SWAP DROP
  ;
;
```

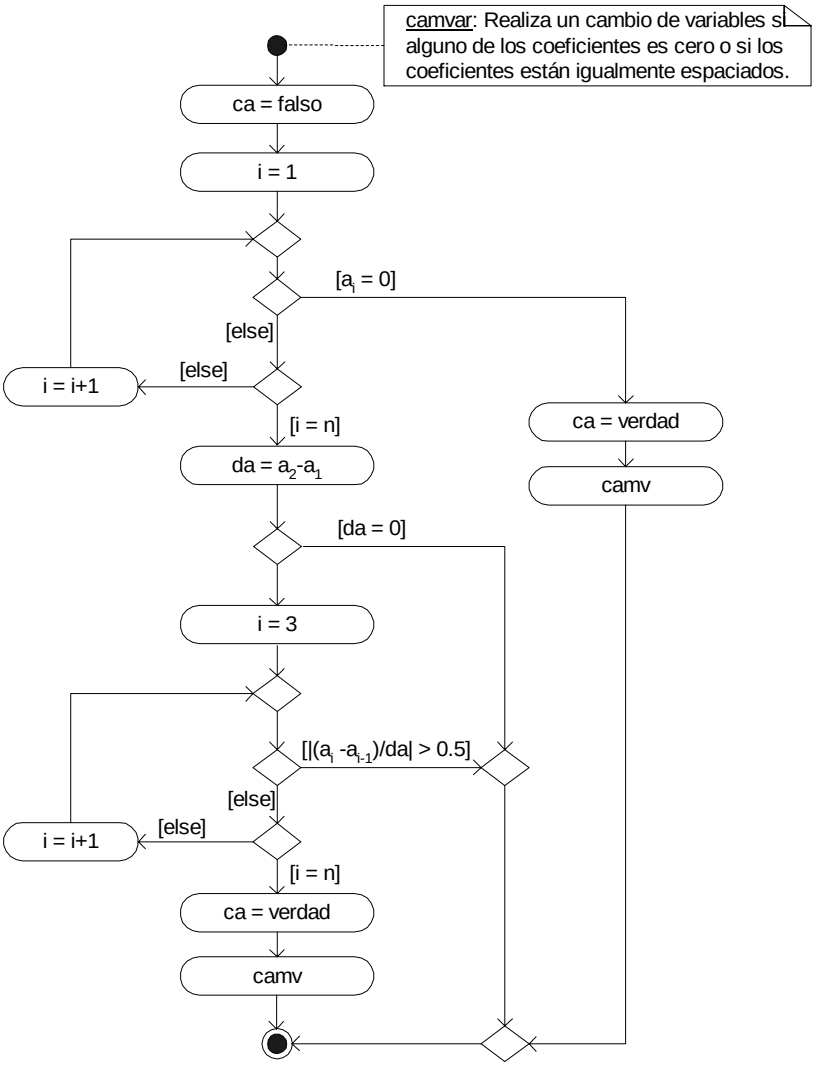
Así para la ecuación: $P_4(x) = x^4 - 3x^2 + x - 5 = 0$, el cambio de variables se lleva a cabo con: ***[1 0 -3 1 -5] camv***, con lo que se obtiene: ***[1. 4. 3. -1. -6.]***, que es el mismo resultado obtenido manualmente.

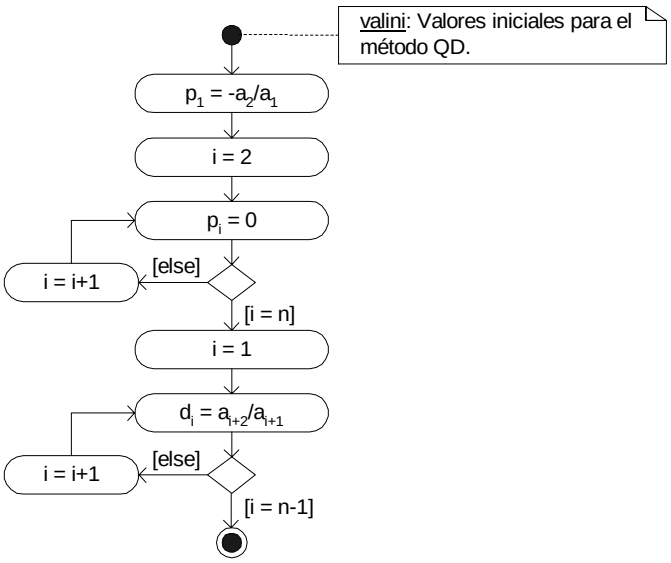
4.4.2. Algoritmo

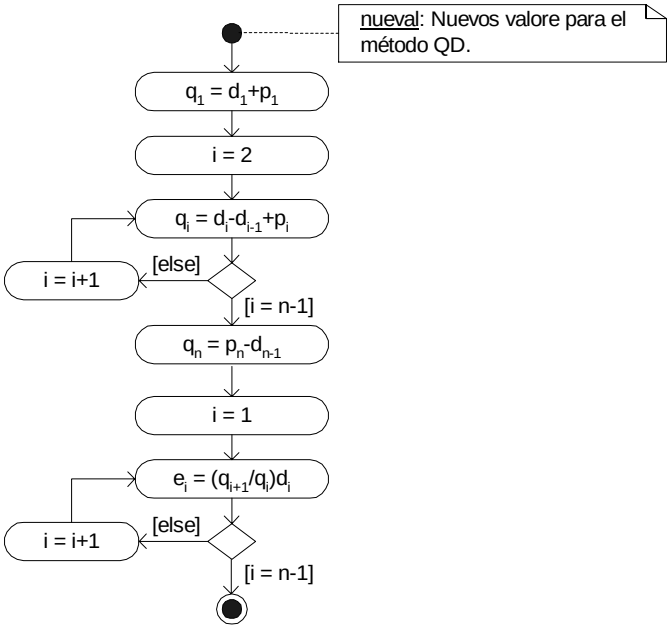
El algoritmo del método "QD" se presenta en el siguiente diagrama de actividades:

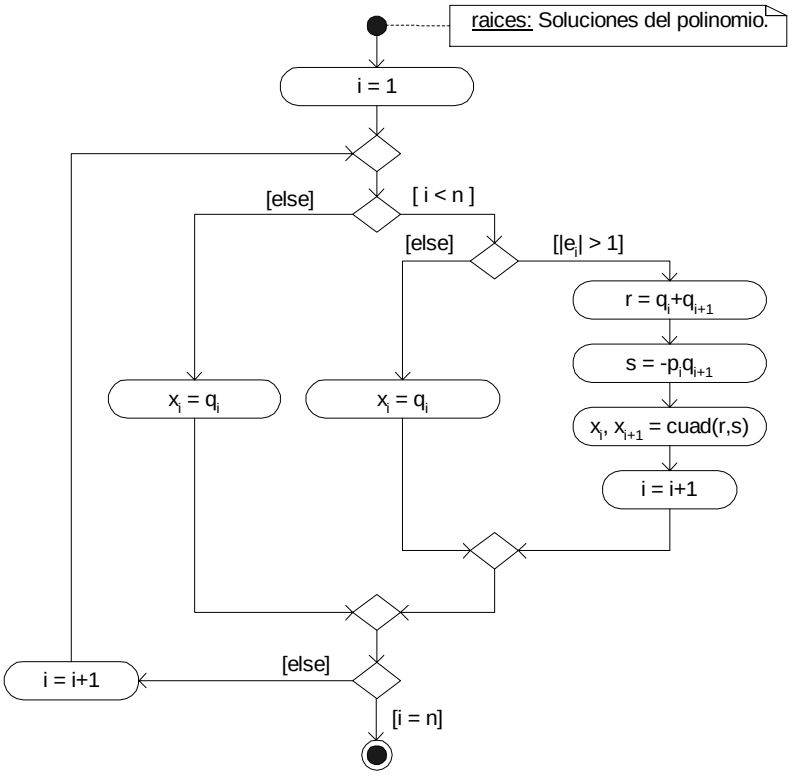


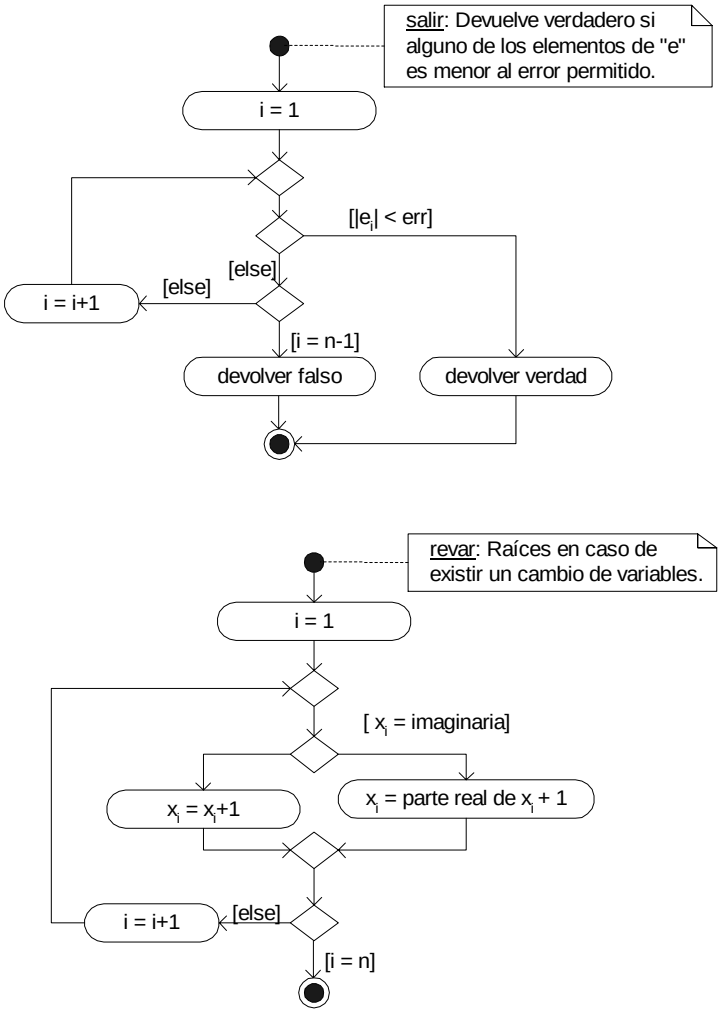
Los algoritmos de los submódulos que forman parte de este algoritmo se presentan en las siguientes páginas.


camvar: Realiza un cambio de variables si alguno de los coeficientes es cero o si los coeficientes están igualmente espaciados.
ca = falso
i = 1
[a_i = 0]
[else]
[else]
i = i+1
[i = n]
ca = verdad
da = a_2-a_1
camv
[da = 0]
i = 3
[|(a_i -a_i-1)/da| > 0.5]
[else]
[else]
i = i+1
[i = n]
ca = verdad
camv


valini: Valores iniciales para el método QD.
p_1 = -a_2/a_1
i = 2
p_i = 0
[else]
i = i+1
[i = n]
i = 1
d_i = a_i+2/a_i+1
[else]
i = i+1
[i = n-1]


nueval: Nuevos valore para el método QD.
$q_1 = d_1+p_1$
i = 2
$q_i = d_i-d_{i-1}+p_i$
[else]
i = i+1
[i = n-1]
$q_n = p_n-d_{n-1}$
i = 1
$e_i = (q_{i+1}/q_i)d_i$
[else]
i = i+1
[i = n-1]


raices: Soluciones del polinomio.
i = 1
[else]
[i < n]
[else]
[$|e_i|$ > 1]
$r = q_i+q_{i+1}$
$s = -p_iq_{i+1}$
x_i, x_{i+1} = cuad(r,s)
i = i+1
$x_i = q_i$
$x_i = q_i$
[else]
i = i+1
[i = n]



4.4.3. Código

El código elaborado en base a los diagramas de actividades del acápite anterior es el siguiente:

```
xNAME qd ( método Q-D para el cálculo de las raíces de un polinomio )
::
  CK1&Dispatch
  FOUR
  ::
    DUP MDIMSDROP #1- ( n )
    DUP #2 #>
    ITE
    ::  ( 8=a; 7=n; 6=p; 5=q; 4=d; 3=e; 2=ca; 1=err )
      %0 OVER NDUPN UNCOERCE ONE{}N FLASHPTR XEQ>ARRY ( p )
      DUP TOTEMPOB ( q )
      %0 4PICK #1- NDUPN UNCOERCE ONE{}N FLASHPTR XEQ>ARRY ( d )
      DUP TOTEMPOB ( e )
      FALSE
      ' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
      ' NULLLAM EIGHT NDUPN DOBIND
      8GETLAM camvar
      valini
      ::
```

```
        BEGIN
          nueval
          salir
          IT
          ::
            raices
            2GETLAM IT revar
            7GETLAM
            2RDROP
          ;
          5GETLAM TOTEMPOB 6PUTLAM
          3GETLAM TOTEMPOB 4PUTLAM
        AGAIN
      ;
      ABND
    ;
    ::
      SWAP OVER #2=
      ITE
      ::
        TWO PULLREALEL SWAP ONE PULLREALEL SWAP UNROT %/ %CHS
        SWAP THREE PULLREALEL SWAP ONE PULLREALEL SWAP DROP %/ %CHS
        xcuad ROT
      ;
      ::
        TWO PULLREALEL SWAP ONE PULLREALEL SWAP DROP %/ %CHS SWAP
      ;
    ;
    UNCOERCE ONE{}N FLASHPTR XEQ>ARRY
  ;
;

NULLNAME camvar ( cambio de variable x por "x-1":  submódulo de qd )
:: ( dato en la pila: vector con los coeficientes )
  7GETLAM #1+ ONE DO ( si uno de los coeficientes es cero se realiza )
    INDEX@ PULLREALEL %0= ( el cambio de variable )
    IT
    ::
      TRUE 2PUTLAM
      xcamv
      7GETLAM INDEXSTO
    ;
  LOOP
  2GETLAM  NOT
  IT
  ::
    TRUE SWAP TWO PULLREALEL SWAP ONE PULLREALEL SWAP UNROT %-
    DUP %0= IT :: ROT 2DROP RDROP ; ( si la diferencia es cero no se hace )
    SWAP                          ( el cambio de variable )
    7GETLAM #1+ THREE DO
      INDEX@ PULLREALEL SWAP
      INDEX@ #1- PULLREALEL SWAP UNROT %-
      3PICK %/ %1- %ABS % 0.5 %>
      IT
      ::
        ROT DROP FALSE UNROT
        7GETLAM INDEXSTO
      ;
```

```
    LOOP              ( si los coeficientes están igualmente espaciados)
    SWAP DROP SWAP IT ( se hace el cambio)
    ::
      TRUE 2PUTLAM
      xcamv
    ;
  ;
  DROP
;

NULLNAME valini ( valores iniciales: submódulo de qd )
::
  6GETLAM 8GETLAM TWO PULLREALEL %CHS SWAP ONE PULLREALEL SWAP 4UNROLL  %/
  ONE PUTREALEL DROP ( p1 )
  4GETLAM
  7GETLAM ONE DO
    SWAP INDEX@ #2+ PULLREALEL SWAP INDEX@ #1+ PULLREALEL SWAP 4UNROLL
     %/ INDEX@ PUTREALEL ( di )
  LOOP
  2DROP
;

NULLNAME nueval ( nuevos valores: submódulo de qd )
::
  5GETLAM 6GETLAM 4GETLAM
  ONE PULLREALEL SWAP 4UNROLL SWAP ONE PULLREALEL SWAP 5UNROLL %+
  ONE PUTREALEL UNROT ( q1=d1+p1)
  7GETLAM TWO DO
    INDEX@ PULLREALEL SWAP INDEX@ #1- PULLREALEL SWAP 5UNROLL %- SWAP
    INDEX@ PULLREALEL SWAP 5UNROLL %+ INDEX@ PUTREALEL
    UNROT ( qi=d[i]-d[i-1]+p[i])
  LOOP
  7GETLAM #1- PULLREALEL SWAP 4UNROLL SWAP 7GETLAM PULLREALEL SWAP DROP
  SWAP %- 7GETLAM PUTREALEL ( qn=pn-p[n-1] )
  3GETLAM UNROT ( e d q )
  7GETLAM ONE DO
    INDEX@ #1+ PULLREALEL SWAP INDEX@ PULLREALEL SWAP 5UNROLL %/
    SWAP INDEX@ PULLREALEL SWAP 5UNROLL %* INDEX@ PUTREALEL
    UNROT ( ei=[q[i+1]/qi]di )
  LOOP
  3DROP
;

NULLNAME salir ( devuelve verdadero si el método QD ha encontrado la
solución )
::
  FALSE 3GETLAM
  7GETLAM ONE DO
    INDEX@ PULLREALEL %ABS 1GETLAM %<
    IT
    ::
      SWAP DROP TRUE SWAP 7GETLAM INDEXSTO
    ;
  LOOP
  DROP
;

NULLNAME raices ( soluciones del polinomio: submódulo de QD)
```

```
::
  6GETLAM 5GETLAM 3GETLAM
  7GETLAM #1+ ONE DO
    INDEX@ 7GETLAM #<
    ITE
    ::
      INDEX@ PULLREALEL %ABS %1 %>
      ITE
      ::
        UNROT INDEX@ PULLREALEL SWAP INDEX@ #1+ PULLREALEL SWAP
        5UNROLL DUP ROT %+ ( r )
        ROT INDEX@ PULLREALEL 4ROLL %* %CHS SWAP 5UNROLL ( s )
        xcuad ( xi, xi+1 )
        FOUR INDEX@ #+ UNROLL FOUR INDEX@ #+ UNROLL
        INDEX@ #1+ INDEXSTO
      ;
      ::
        SWAP INDEX@ PULLREALEL THREE INDEX@ #+ UNROLL SWAP ( xi=qi)
      ;
    ;
    ::
      SWAP INDEX@ PULLREALEL THREE INDEX@ #+ UNROLL SWAP ( xi=qi)
    ;
  LOOP
  3DROP
;

NULLNAME revar ( suma 1 a las soluciones en caso de haberse efectuado )
::  ( un cambio de variables )
  7GETLAM #1+ ONE DO  ( Datos en la pila: soluciones del polinomio )
    DUP
    TYPECMP?
    ITE
    ::
      C%>% SWAP %1+ SWAP %>C%
    ;
    ::
      %1+
    ;
    7GETLAM UNROLL
  LOOP
;
```

Este programa concluye cuando uno de los valores de *"e"* cumple con el error permitido, por lo tanto los resultados obtenidos son sólo aproximados, esto porque como ya se dijo, el método ***QD*** se emplea generalmente para obtener valores iniciales y no las soluciones en si.

4.4.4. Ejemplos

4.4.4.1. Resolución de una ecuación de cuarto grado con soluciones reales

Como ejemplo empleemos el programa "qd" para encontrar las soluciones aproximadas de la ecuación:

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0$$

```
« 1E-10 'err' STO
  [ 128 -256 160 -32 1 ]
```

```
  qd
  AXL
  { x1 x2 x3 x4 } →TAG
  LIST→ DROP 'err' PURGE
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 3.80602336789E-2
x2: .308522093581
x3: .682258450989
x4: .971159221743
s: 1.2726
```

(Las soluciones son: x1: 3.80602337444E-2; x2: .308658283817; x3: .691341716183; x4: .961939766256)

4.4.4.2. Resolución de una ecuación de cuarto grado con soluciones imaginarias

Como segundo ejemplo resolveremos la ecuación:

$$P_4(x) = x^4 - 6x^3 + 12x^2 - 19x + 12 = 0$$

```
« 1E-5 'err' STO
  [ 1 -6 12 -19 12 ]
  qd
  AXL
  { x1 x2 x3 x4 } →TAG
  LIST→ DROP 'err' PURGE
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: (1.00004529467,0.)
x2: (.4999729587,1.65850981458)
x3: (.4999729587,-1.65850981458)
x4: (4.00000878792,0.)
s: 1.6926
```

(Las soluciones son: x1: (1.,0.); x2: (.5,1.65831239518); x3: (.5,-1.65831239518); x4: (4.,0.).)

4.4.4.3. Resolución de una ecuación de cuarto grado con coeficientes cero

Como tercer ejemplo resolveremos la ecuación:

$$P_4(x) = x^4 - 3x^2 + x - 5 = 0$$

```
« 1E-7 'err' STO
  [ 1 0 -3 1 -5 ]
  qd
  AXL
  { x1 x2 x3 x4 } →TAG
  LIST→ DROP 'err' PURGE
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: (1.95965307698,0.)
x2: (.08831630012,1.09458500854)
x3: (.08831630012,-1.09458500854)
x4: (-2.13628567722,0.)
```

s: 1.8688

(Las soluciones son: x1: (1.94992054774,0); x2: (9.31811135245E-2, 1.09161799092); x3: (9.31811135245E-2,-1.09161799092); x4: (-2.13628277479, 0.)).

4.4.4.4. Resolución de una ecuación de octavo grado

Como cuarto ejemplo encontraremos las soluciones aproximadas del polinomio:

$$P_8(x) = x^8 + x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$$

```
« 1E-9 'err' STO
  [ 1 1 -9 13 21 -125 77 111 -90 ]
  qd
  AXL
  { x1 x2 x3 x4 x5 x6 x7 x8 } →TAG
  LIST→ DROP 'err' PURGE
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: (.989473684187,0.)
x2: (-1.06947368388,0.)
x3: (1.07999999842,0.)
x4: (2.22318178769,0.)
x5: (.867344257225,1.92029016846)
x6: (.867344257225,-1.92029016846)
x7: (-2.86752427124,0.)
x8: (-3.09034602954,0.)
s: 6.215
```

(Las soluciones son: x1: (1.,0.); x2: (1.,0.); x3: (-1.,0.); x4: (2.,0.); :x5 (-3,0); x6: (-3,0); x7: (1,2); x8: (1,-2)).

4.4.5. Ejercicios

19. Encuentre las soluciones aproximadas del polinomio de Chebyshev:

$$P_6(x) = 32x^6 - 48x^4 + 18x^2 - 1 = 0$$

20. Encuentre las soluciones aproximadas del polinomio de Legendre:

$$P_6(x) = 693x^6 - 945x^4 + 315x^2 - 15 = 0$$

21. Encuentre las soluciones aproximadas del polinomio de Laguerre:

$$P_4(x) = x^4 - 16x^3 + 72x^2 - 96x + 24 = 0$$

22. Encuentre las soluciones aproximadas de la ecuación cúbica:

$$P_3(x) = x^3 + 2x^2 + 3x + 4 = 0$$

23. Encuentre las soluciones aproximadas de la ecuación:

$$P_4(x) = x^4 + 4x^3 + 21x^2 + 4x + 20 = 0$$

24. Encuentre las soluciones aproximadas de la ecuación:

$$P_6(x) = x^6 + x^5 - 8x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

4.5. Resolución de la ecuación cúbica $x^3+a_1x^2+a_2x+a_3=0$

La ecuación polinomial que con mayor frecuencia se presenta en la práctica es la ecuación cúbica, razón por la cual es conveniente contar con un programa que permita su resolución directa.

Una ecuación cúbica puede ser resuelta directamente con el método de *Cardano,* para ello se deben calcular los siguientes parámetros:

$$\begin{aligned} q &= \frac{3a_2 - a_1^2}{9} \\ r &= \frac{9a_1a_2 - 27a_3 - 2a_1^3}{54} \\ s &= \sqrt[3]{r + \sqrt{d}} \\ t &= \sqrt[3]{r - \sqrt{d}} \\ d &= q^3 + r^2 \end{aligned} \qquad (9)$$

Donde *"d"* es el discriminante.

Si el discriminante es cero todas las raíces son reales y al menos dos de ellas son iguales:

$$\begin{aligned} x_1 &= 2s - \frac{a_1}{3} \\ x_2 &= x_3 = -s - \frac{a_1}{3} \end{aligned} \qquad (10)$$

Si el discriminante es mayor a cero, una de las raíces es real y las otras dos son complejas conjugadas:

$$\begin{aligned} x_1 &= s + t - \frac{a_1}{3} \\ x_2 &= -\frac{s+t}{2} - \frac{a_1}{3} + \frac{\sqrt{3}(s-t)}{2}i \\ x_3 &= -\frac{s+t}{2} - \frac{a_1}{3} - \frac{\sqrt{3}(s-t)}{2}i \end{aligned} \qquad (11)$$

Si el discriminante es menor a cero todas las raíces son reales y distintas:

$$\begin{aligned} \theta &= tan^{-1}\left(\frac{\sqrt{-d}}{r}\right) \\ x_1 &= 2\sqrt{-q}\cos\left(\frac{\theta}{3}\right) - \frac{a_1}{3} \\ x_2 &= 2\sqrt{-q}\cos\left(\frac{\theta}{3} + \frac{2\pi}{3}\right) - \frac{a_1}{3} \\ x_3 &= 2\sqrt{-q}\cos\left(\frac{\theta}{3} + \frac{4\pi}{3}\right) - \frac{a_1}{3} \end{aligned} \qquad (12)$$

Si bien en teoría los métodos directos siempre devuelven los resultados correctos, esto no es del todo cierto en la práctica debido a los errores de redondeo. Así cuando se trabaja con coeficientes relativamente grandes el discriminante puede resultar diferente de cero, cuando debía ser cero. Por

esta razón, es más seguro analizar el resultado de la siguiente expresión y no directamente el valor del discriminante:

$$e = \frac{q^3 + r^2}{r^2} = \frac{q^3}{r^2} + 1 \qquad (13)$$

Cuando el discriminante es ser cero, esta expresión devuelve siempre resultados cercanos a cero. Para minimizar aún más los errores de redondeo es conveniente trabajar con reales de doble precisión.

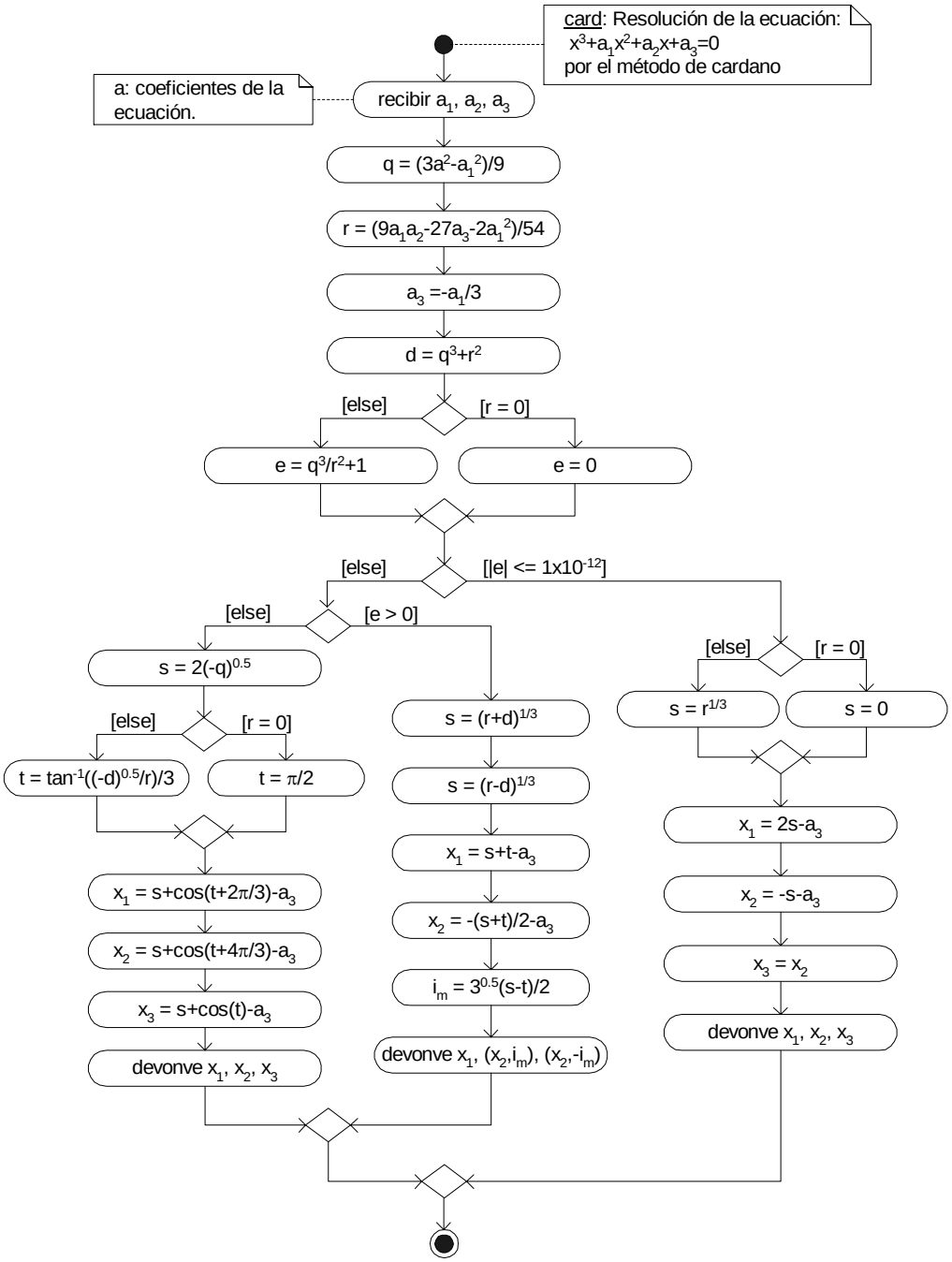
4.5.1. Algoritmo

El algoritmo del método de *Cardano* se presenta en el diagrama de actividades de la siguiente página.

4.5.2. Código

El código elaborado en base al algoritmo es el siguiente:

```
xNAME card ( resolución de la ecuación x^3+a1*x^2+a2*x+a3=0 )
:: ( dato en la pila: a = vector con los coeficientes )
  CK3&Dispatch
  # 111
  :: ( a1=9; a2=8; a3=7; q=6; r=5; d=4; e=3 ; s=2; t=1 )
    SETRAD
    %>%% UNROT %>%% UNROT %>%% UNROT
    %%0 SIX NDUPN DROP
    ' NULLLAM NINE NDUPN DOBIND
    %%3 8GETLAM %%* 9GETLAM DUP %%* %%- %% 9 %%/ 6PUTLAM ( q )
    %% 9 9GETLAM %%* 8GETLAM %%* %% 27 7GETLAM %%* %%- %%2 9GETLAM
    DUPDUP %%* %%* %%* %%- %% 54 %%/ 5PUTLAM ( r )
    9GETLAM %%3 %%/ 7PUTLAM ( a1/3 )
    6GETLAM DUPDUP %%* %%* 5GETLAM DUP %%* 2DUP %%+ 4PUTLAM ( d )
    5GETLAM %%0= ITE DROP :: %%/ %%1 %%+ ; 3PUTLAM
    3GETLAM %%ABS %% 1E-12 %%<=
    ITE
    :: ( raices reales, dos iguales )
      5GETLAM %%0=
      ITE
      ::
        %%0 2PUTLAM
      ;
      ::
        5GETLAM DUP %%ABS DUP %%3 %%1/ %%^ UNROT %%/ %%* 2PUTLAM
      ;
      2GETLAM 2GETLAM %%+ 7GETLAM %%- %%>% ( x1 )
      2GETLAM %%CHS 7GETLAM %%- %%>% DUP ( x2, x3 )
    ;
    ::
      3GETLAM %%0>
      ITE
      :: ( una raíz real dos imaginarias )
        4GETLAM %%SQRT 4PUTLAM
        5GETLAM 4GETLAM %%+ DUP %%ABS DUP %%3 %%1/ %%^ UNROT
        %%/ %%* 2PUTLAM
        5GETLAM 4GETLAM %%- DUP %%ABS DUP %%3 %%1/ %%^ UNROT
        %%/ %%* 1PUTLAM
        2GETLAM 1GETLAM %%+ 7GETLAM %%- %%>% ( x1 )
        2GETLAM 1GETLAM %%+ %%2 %%/ %%CHS 7GETLAM %%- %%>%
```




```
  %% 8.66025403784435E-1 2GETLAM 1GETLAM %%- %%* %%>%
  2DUP %>C% ( x2 )
  UNROT %CHS %>C% ( x3 )
;
:: ( raíces reales y diferentes )
  %%2 6GETLAM %%CHS %%SQRT %%* 2PUTLAM
  5GETLAM %%0=
  ITE
  ::
    %%PI %%2 %%/
  ;
  ::
```

```
          4GETLAM %%CHS %%SQRT 5GETLAM %%/ %%>% %ATAN %>%%
          %%3 %%/
        ;
        1PUTLAM ( theta )
        2GETLAM 1GETLAM %%2PI %%3 %%/ %%+ %%COS %%* 7GETLAM %%- %%>% ( x1 )
        2GETLAM 1GETLAM %%4 %%PI %%* %%3 %%/ %%+ %%COS %%* 7GETLAM
        %%- %%>% ( x2 )
        2GETLAM 1GETLAM %%COS %%* 7GETLAM %%- %%>% ( x3 )
      ;
    ;
    ABND
  ;
;
```

4.5.3. Ejemplos

4.5.3.1. Ecuación cúbica con soluciones imaginarias

Como primer ejemplo encontremos las soluciones de la ecuación cúbica:

$$P_3(x) = x^3 + 2x^2 + 3x + 4 = 0$$

```
« 2 3 4
  card
  3 →LIST
  { x1 x2 x3 } →TAG
  LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: -1.65062919144
x2: (-.17468540428,1.54686888723)
x3: (-.17468540428,-1.54686888723)
s: .2966
```

Que son las tres soluciones (una real y dos imaginarias) y el tiempo empleado en obtenerlas.

4.5.3.2. Ecuación cúbica con soluciones reales iguales

Como segundo ejemplo encontremos las soluciones de la ecuación cúbica:

$$P_3(x) = x^3 - 17x^2 + 91x - 147 = 0$$

```
« -17 91 -147
  card
  3 →LIST
  { x1 x2 x3 } →TAG
  LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 3.
x2: 7.
x3: 7.
s: .2568
```

Que son las tres soluciones reales (dos de ellas iguales) y el tiempo empleado en obtenerlas.

4.5.3.3. Ecuación cúbica con soluciones reales diferentes

Como tercer ejemplo encontremos las soluciones de la ecuación cúbica:

$$P_3(x) = x^3 - 60x^2 + 1100x - 6000 = 0$$

```
« -60 1100 -6000
  card
  3 →LIST
  { x1 x2 x3 } →TAG
  LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 10.
x2: 20.
x3: 30
s: .3081
```

Que son las tres soluciones reales y el tiempo empleado en obtenerlas.

4.5.4. Ejercicios

25. Encuentre las soluciones de la siguiente ecuación:

$$P_3(x) = x^3 - 21x^2 + 143x - 315 = 0$$

26. Encuentre las soluciones de la siguiente ecuación:

$$P_3(x) = 5x^3 - 250x^2 + 4000x - 20000 = 0$$

27. Encuentre las soluciones de la siguiente ecuación:

$$P_3(x) = x^3 + 8x^2 + 12x + 15 = 0$$

28. Encuentre las soluciones de la siguiente ecuación:

$$P_3(x) = 4x^3 - 96x^2 + 764x - 2016 = 0$$

29. Encuentre las soluciones del polinomio de *Laguerre*:

$$P_3(x) = x^3 - 9x^2 + 18x - 6 = 0$$

30. Encuentre las soluciones del polinomio de Legendre:

$$P_6(x) = 693x^6 - 945x^4 + 315x^2 - 15 = 0$$

4.6. Método de Newton – Raphson

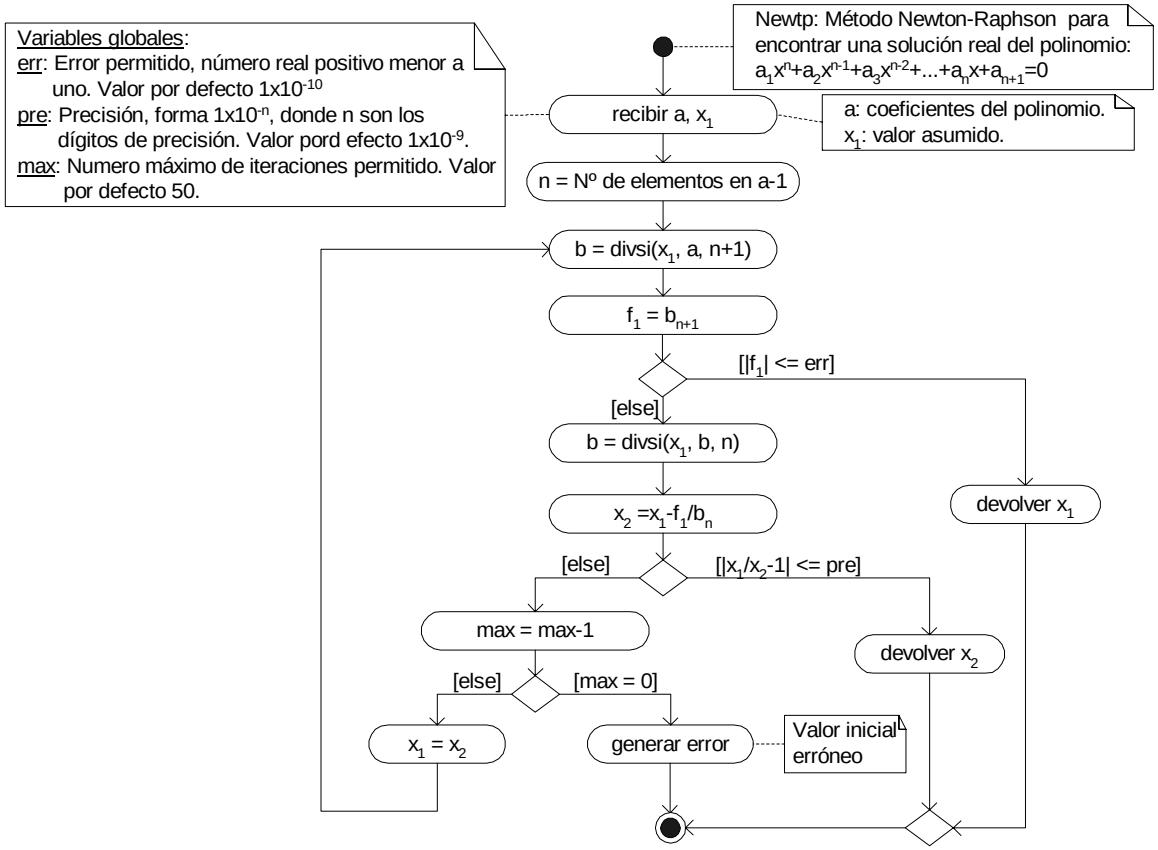
Como ya vimos en el capítulo anterior, el método de *Newton – Raphson* (al igual que los otros métodos estudiados en dicho capítulo) permite encontrar una de las soluciones reales de una ecuación polinomial.

La raíz encontrada depende del valor inicial asumido. Si se dan valores iniciales cercanos a las soluciones, el método de *Newton – Raphson*, puede encontrar todas las soluciones reales del polinomio. Valores iniciales adecuados pueden ser obtenidos con el método *QD*.

El método de *Newton – Raphson* no cambia cuando se trata de polinomios, pero ahora en lugar de recibir una función se reciben los coeficientes del polinomio y su derivada, se calcula mediante división sintética, que como vimos es más eficientes para evaluar polinomios.

4.6.1. Algoritmo

El algoritmo del método de *Newton – Raphson,* para encontrar una solución real de un polinomio se presenta en el siguiente diagrama de actividades.



4.6.2. Código

El código elaborado en base al algoritmo es el siguiente:

```
xNAME Newtp ( Método de Newton - Raphson para ecuaciones polinomiales)
:: ( datos en la pila: a=coeficientes del polinomio; x1=valor 8GETLAMsumido)
  CK2&Dispatch
  # 41
  :: ( a=8; x1=7; n=6 ; err=5; pre=4; max=3; x2=2; f1=1 )
    OVER MDIMSDROP #1-
    ' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
    ' ID pre @ ?SKIP % 1E-9  ( si no existe "pre" se emplea 1E-9 )
    ' ID max @ ?SKIP % 50    ( si no existe "max" se emplea 50 )
    %0 %0
    ' NULLLAM EIGHT NDUPN DOBIND
    ::
      BEGIN
        7GETLAM 8GETLAM TOTEMPOB 6GETLAM #1+ divsi1 ( b= divisi[x1,a,n+1] )
        6GETLAM #1+ PULLREALEL 1PUTLAM ( f1= b[n+1] )
        1GETLAM %ABS 5GETLAM %< ( |f1|<err )
        IT :: DROP 7GETLAM 2RDROP ; ( devolver x1 )
        7GETLAM SWAP 6GETLAM divsi1 ( b=divsi[x1,b,n] )
        7GETLAM 1GETLAM ROT 6GETLAM PULLREALEL SWAP DROP %/
        %- 2PUTLAM ( x2=x1-f1/df )
        7GETLAM 2GETLAM %/ %1- %ABS 4GETLAM %< ( |x1/x2-1|<pre )
```

```
        IT :: 2GETLAM 2RDROP ; ( devolver x2 )
        3GETLAM %1- 3PUTLAM ( max=max-1)
        3GETLAM %0= ( max==0)
        IT :: ABND #A01 DO#EXIT ; ( valor inicial erróneo )
        2GETLAM 7PUTLAM ( x1=x2)
      AGAIN
    ;
    ABND
  ;
;
```

4.6.3. Ejemplos

4.6.3.1. Soluciones reales de una ecuación de cuarto grado

Como primer ejemplo encontraremos las soluciones reales de la ecuación:

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0$$

Empleando como valores iniciales los valores calculados con método QD:

```
x1: 3.80602336789E-2
x2: .308522093581
x3: .682258450989
x4: .971159221743
```

El programa que resuelve el problema es el siguiente:

```
« 3.80602336789E-2 0.308522093581 0.68258450989 0.97115921743
  [ 128 -256 160 -32 1 ]
  → x1 x2 x3 x4 a
  « a x1 Newtp
    a x2 Newtp
    a x3 Newtp
    a x4 Newtp
    4 →LIST
    { x1 x2 x3 x4 } →TAG
    LIST→ DROP
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 3.80602337444E-2
x2: .308658283815
x3: .691341716178
x4: .96193976625
s: 1.2297
```

Que son las soluciones prácticamente exactas y el tiempo empleado en obtenerlas.

4.6.3.2. Soluciones reales de una ecuación de octavo grado

Como segundo ejemplo encontraremos las soluciones reales de la siguiente ecuación, empleando como valores iniciales los valores aproximados obtenidos con el método *QD*.

$$P_8(x) = x^8 - 60x^7 + 1554x^6 - 22680x^5 + 203889x^4 - 1155420x^3 + 4028156x^2 - 7893840x - 6652800 = 0$$

El programa que resuelve el problema es:

```
« [ 1. -60. 1554. -22680. 203889. -1155420. 4028156. -7893840. 6652800. ]
  0 0
  → a n x
  « 0.01 'err' STO
    a qd 'x' STO
    a SIZE 1 GET 1 - 'n' STO
    1 n FOR i
      a x i GET Newtp
    NEXT
    n →LIST
    { x1 x2 x3 x4 x5 x6 x7 x8 } →TAG
    LIST→ DROP
    'err' PURGE
  »
»
```

Haciendo correr el programa con ***TEVAL*** se obtiene:

```
x1: 4.00000001064
x2: 4.99999943608
x3: 5.99999963434
x4: 6.99999623428
x5: 7.99999622521
x6: 9.00002256003
x7: 10.000012268
x8: 11.0000036412
s: 9.3712
```

Que son las soluciones y el tiempo empleado en obtenerlas.

4.6.3.3. Soluciones reales de una ecuación de quinto grado

Como tercer ejemplo encontraremos las soluciones reales de la ecuación:

$$P_5(x) = x^5 - 11x^4 + 43x^3 - 79x^2 + 76x - 30 = 0$$

En este caso existen soluciones reales e imaginarias, por lo que es necesario analizar el tipo de resultado devuelto por ***"qd"*** antes de llamar al método de Newton. El programa que resuelve el problema es el siguiente:

```
« [ 1. -11. 43. -79. 76. -30. ]
  0 0 0
  □ a x n m
  « a qd 'x' STO
    a SIZE 1 GET 1 - 'n' STO
    1 n FOR i
      x i GET TYPE
      IF 1 == THEN
        x i GET IM
        IF 0 == THEN
          a x i GET RE Newtp 1 'm' STO+
        END
      ELSE
        a x i GET Newtp 1 'm' STO+
      END
    NEXT
    m □LIST
    1 m FOR i
      "x" i +
    NEXT
```

```
    m □LIST □TAG
    LIST□ DROP
  »
»
```

Haciendo correr el programa con ***TEVAL*** se obtiene:

```
x1.: .999999999993
x2.: 3.00000000006
x3.: 5.
s: 4.5264
```

Que son las tres soluciones reales y el tiempo empleado en obtenerlas (las otras dos soluciones son imaginarias).

4.6.4. Ejercicios

31. Empleando como valores iniciales los resultados obtenidos con QD, encuentre las soluciones reales del polinomio de Chebyshev:

$$P_6(x) = 32x^6 - 48x^4 + 18x^2 - 1 = 0$$

32. Empleando como valores iniciales los resultados obtenidos con QD, encuentre las soluciones reales del polinomio de Legendre:

$$P_6(x) = 693x^6 - 945x^4 + 315x^2 - 15 = 0$$

33. Empleando como valores iniciales los resultados obtenidos con QD, encuentre las soluciones reales del polinomio de Laguerre:

$$P_4(x) = x^4 - 16x^3 + 72x^2 - 96x + 24 = 0$$

34. Empleando como valores iniciales los resultados obtenidos con QD, encuentre las soluciones reales del polinomio:

$$P_6(x) = x^6 + x^5 - 8x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

4.7. Método de Bairstow

En este método las soluciones se encuentran dividiendo la ecuación polinomial entre una ecuación de segundo grado. El método busca los coeficientes de la ecuación de cuadrática que hacen el residuo de la división cero, cuando esto sucede dos de las soluciones se calculan resolviendo la ecuación cuadrática.

El procedimiento puede ser repetido con el polinomio restante (cuyo grado es "n-2") encontrándose así otras dos soluciones. Se puede continuar de esta manera hasta encontrar todas las soluciones del polinomio original.

El principal inconveniente de este método, al igual que el método de ***Newton - Raphson*** es que requiere valores iniciales para comenzar el proceso y si dichos valores no son adecuados el proceso puede requerir demasiadas iteraciones o incluso puede no converger.

Al dividir el polinomio general (ecuación 1), entre la ecuación de segundo grado: x^2-r*x-s, resulta:

$$\frac{P_n(x)}{x^2 - r - s} = Q_{n-2}(x)(x^2 - rx - s) + resíduo = 0 \qquad (14)$$

El método de ***Bairstow*** busca los valores de ***"r"*** y ***"s"*** que hacen el residuo cero (o casi cero). Cuando esto sucede la ecuación (14) queda igualada a cero y entonces se cumple que:

$$\begin{aligned} Q_{n-2}(x) &= 0 \\ x^2 - rx - s &= 0 \end{aligned} \tag{15}$$

Entonces dos de las raíces (soluciones) pueden ser calculadas resolviendo la ecuación de segundo grado. Posteriormente se pueden encontrar otras dos raíces repitiendo el procedimiento con el polinomio Q_{n-2} y continuar de esa manera hasta determinar todas las raíces del polinomio original.

La división del polinomio $P_n(x)$ entre la ecuación de segundo grado está dada por la ecuación 3 (división sintética) y el objetivo del método de *Bairstow* es hacer el residuo $(b_n x + b_{n+1})$ cero. Para ello, puesto que b_n y b_{n+1} dependen de los coeficientes r y s, se puede expandir bn y $bn+1$ en series de *Taylor*:

$$b_n(r+\Delta r, s+\Delta s) = b_n(r,s) + \frac{\partial b_n(r,s)}{\partial r}\Delta r + \frac{\partial b_n(r,s)}{\partial s}\Delta s + \frac{1}{2!}\left(\frac{\partial b_n(r,s)}{\partial r}\Delta r + \frac{\partial b_n(r,s)}{\partial s}\Delta s\right)^2 + \ldots \infty = 0$$

$$b_{n+1}(r+\Delta r, s+\Delta s) = b_{n+1}(r,s) + \frac{\partial b_{n+1}(r,s)}{\partial r}\Delta r + \frac{\partial b_{n+1}(r,s)}{\partial s}\Delta s + \frac{1}{2!}\left(\frac{\partial b_{n+1}(r,s)}{\partial r}\Delta r + \frac{\partial b_{n+1}(r,s)}{\partial s}\Delta s\right)^2 + \ldots \infty = 0$$

Donde "Δr" y "Δs" son los valores que deben sumarse a "r" y "s" para que tanto "b_n" como "b_{n+1}" se igualen a cero. Por supuesto, no es posible en la práctica resolver este sistema (que es más complicado aún que el problema original), razón por la cual sólo se toman los términos de primer orden y se desprecian los términos de segundo orden y superiores, con lo que el sistema se simplifica a:

$$\begin{aligned} b_n + \frac{\partial b_n}{\partial r}\Delta r + \frac{\partial b_n}{\partial s}\Delta s &= 0 \\ b_{n+1} + \frac{\partial b_{n+1}}{\partial r}\Delta r + \frac{\partial b_{n+1}}{\partial s}\Delta s &= 0 \end{aligned} \tag{16}$$

Donde las funciones siguen siendo evaluadas en el punto (r,s). Si se conocen los valores de las derivadas, entonces se tiene un sistema de 2 ecuaciones lineales con 2 incógnitas.

Las derivadas parciales pueden ser calculadas realizando una segunda división sintética del polinomio restante Q_{n-2} entre la ecuación cuadrática $x^2 - rx - s$. Si denominamos "c" a los coeficientes resultantes de esta segunda división sintética, entonces las derivadas parciales son:

$$\begin{aligned} \frac{\partial b_i}{\partial r} &= c_{i-1} \\ \frac{\partial b_i}{\partial s} &= c_{i-2} \end{aligned} \tag{17}$$

Empleando estas relaciones la ecuación (16) queda en la forma:

$$\begin{aligned} c_{n-1} * \Delta r + c_{n-2} * \Delta s &= -b_n \\ c_n * \Delta r + c_{n-1} * \Delta s &= -b_{n+1} \end{aligned} \tag{18}$$

Que puede ser resuelta con la regla de Cramer:

$$\Delta r = \begin{vmatrix} -b_n & c_{n-2} \\ -b_{n+1} & c_{n-1} \end{vmatrix} \Big/ \begin{vmatrix} c_{n-1} & c_{n-2} \\ c_n & c_{n-1} \end{vmatrix} = \frac{-b_n * c_{n-1} + b_{n+1} * c_{n-2}}{c_{n-1}^2 - c_n * c_{n-2}} \qquad (19)$$

$$\Delta s = \frac{-b_n - c_{n-1} * \Delta r}{c_{n-2}}$$

Sin embargo, puesto que estos valores han sido calculados despreciando los términos de segundo orden y superiores, son solo valores aproximados y al ser sumados a *"r"* y *"s"* no hacen que *"b_n"* y *"b_{n+1}"* se igualan a cero, aunque si aproximan su valor a cero.

Por lo tanto el cálculo de *"r"* y *"s"* que hacen *"b_n"* y *"b_{n+1}"* cero es iterativo: comenzando con los valores iniciales asumidos de *"r"* y *"s"* se calcula mediante división sintética los coeficientes *"b"*, se verifica si *"b_n"* y *"b_{n+1}"* son cero (o casi cero), de ser así el proceso termina y se calculan dos de las raíces con los valores de *"r"* y *"s"*, caso contrario se calculan los coeficientes *"c"*, los incrementos*"Δr"* y *"Δs"*, con ellos nuevos valores de *"r"* y *"s" (r=r+Δr, s=s+Δs)* y se repite el proceso.

El proceso puede concluir también cuando los valores absolutos de *"Δr"* y *"Δs"* son casi cero, es decir cuando los valores de *"r"* y *"s"* prácticamente no cambian.

Para comprender método de Bairstow, resolvamos la siguiente ecuación de cuarto grado:

$$P_4(x) = x^4 - 1.1x^3 + 2.3x^2 + 0.5x + 3.3 = 0$$

Nuestros valores asumidos serán *r=-1* y *s=-1:*

```
[ 1 -1.1 2.3 0.5 3.3] -1 -1 divs2
```

Con lo que obtenemos:

```
[ 1. -2.1 3.4 -.8 .7 ]
```

Puesto que los valores de b_n y b_{n+1} (b_4 y b_5) no están todavía cercanos a cero, el proceso debe continuar con el cálculo de los valores de *"c"*:

```
[ 1. -2.1 3.4 -.8 ] -1 -1 divs2
```

Con lo que obtenemos:

```
[ 1. -3.1 5.5 -3.2 ]
```

Entonces calculamos los valores de *"Δr"* y *"Δs"*:

$$\Delta r = \frac{-b_4 * c_3 + b_5 * c_2}{(c_3)^2 - c_4 * c_2} = \frac{-(-0.8) * (5.5) + (0.7) * (-3.1)}{(5.5)^2 - (-3.2) * (-3.1)} = 0.109690113133$$

$$\Delta s = \frac{-b_4 - c_3 * \Delta r}{c_2} = \frac{-(-0.8) - (5.5) * (0.109690113133)}{-3.1} = -6.34530250865E-2$$

Entonces lo nuevos valores de *"r"* y *"s"* son:

```
r = r+Δr = -1+(0.109690113133) = -0.890309886867

s= s+Δs = -1+(-6.34530250865E-2) =  -1.06345302509
```

Ahora repetimos el proceso calculando nuevos valores de *b:*

```
[ 1 -1.1 2.3 0.5 3.3] -0.890309886867 -1.06345302509 divs2
```

Con lo que se obtiene:

[1. -1.99030988687 3.00853954512 -.06193143199 .15569768585]

Como se puede observar los valores de *"b_n"* y *"b_{n+1}"* se están aproximando a cero, pero todavía no son relativamente grandes, por lo que el proceso debe continuar con el cálculo de nuevos valores de *"c"*:

[1. -1.99030988687 3.00853954512 -.06193143199]
-0.890309886867 -1.06345302509 divs2

Con lo que se obtiene:

[1. -2.88061977374 4.5097307849 -1.01358552437]

Por lo tanto los nuevos valores de *"Δr"* y *"Δs"* son:

$$\Delta r = -9.71480945591E-3$$
$$\Delta s = -3.67082834836E-2$$

Entonces los nuevos valores de *"r"* y *"s"* son:

$$r = r+\Delta r = -0.890309886867-9.71480945591E-3 = -0.900024696323$$

$$s = s+\Delta s = -1.06345302509-3.67082834836E-2 = -1.10016130857$$

Repetimos el proceso una vez más. Calculamos los nuevos valores de *"b":*

[1 -1.1 2.3 0.5 3.3] -0.900024696323 -1.10016130857 divs2

Con lo que obtenemos:

[1. -2.00002469632 2.99991031137 .00035642009 -.00070604063]

Como se puede observar los valores de *"b_n=b_4"* y *"b_{n+1}=b_5"* pueden ser considerados prácticamente cero, sin embargo, para conseguir un resultado más exacto repetiremos el proceso una vez más.

Los nuevos valore de *"c"* se calculan con:

[1. -2.00002469632 2.99991031137 .00035642009]
-0.900024696323 -1.10016130857 divs2

Con lo que obtenemos:

[1. -2.90004939264 4.50986507673 -.86811139133]

Por lo tanto los nuevos valores de *"Δr"* y *"Δs"* son:

$$\Delta r = 2.46977359345E-5$$
$$\Delta s = 1.61308820447E-4$$

Siendo los nuevos valores de *"r"* y *"s"* son:

$$r = r+\Delta r = -0.900024696323+2.46977359345E-5 = -0.899999998587$$

$$s = s+\Delta s = -1.10016130857+1.61308820447E-4 = -1.09999999975$$

Entonces los nuevos valores de *"b"* se calculan con:

[1 -1.1 2.3 0.5 3.3] -0.899999998587 -1.09999999975 divs2

Con lo que resulta:

[1. -1.99999999859 2.99999999616 .00000000564 -.00000000011]

Ahora los valores de *"b_n=b_4"* y *"b_{n+1}=b_5"* son prácticamente cero por lo que el proceso puede concluir, siendo lo valores de *"r"* y *"s"* aproximadamente: ***r=-0.9*** y ***s=-1.1***, por lo tanto dos de las soluciones se calculan con:

-0.9 -1.1 cuad

Con lo que se obtiene:

(-.45,.947364766075)
(-.45,-.947364766075)

Que son dos de las cuatro soluciones del polinomio.

El polinomio restante ***(Q_{n-2})*** es de segundo grado y sus coeficientes son los coeficientes ***"b"***, es decir:

$$Q_2 = x^2 - 2x + 3 = 0$$

Que puede ser resuelto con:

2 -3 cuad

Con lo que se obtienen las otras dos soluciones de la ecuación:

(1.,1.41421356238)
(1.,-1.41421356238)

En consecuencia las cuatro soluciones (o raíces) del polinomio son:

x_4 = -0.45+0.947364766075i
x_3 = -0.45-0.947364766075i.
x_2 = 1+1.41421356237i
x_1 = 1-1.41421356237i

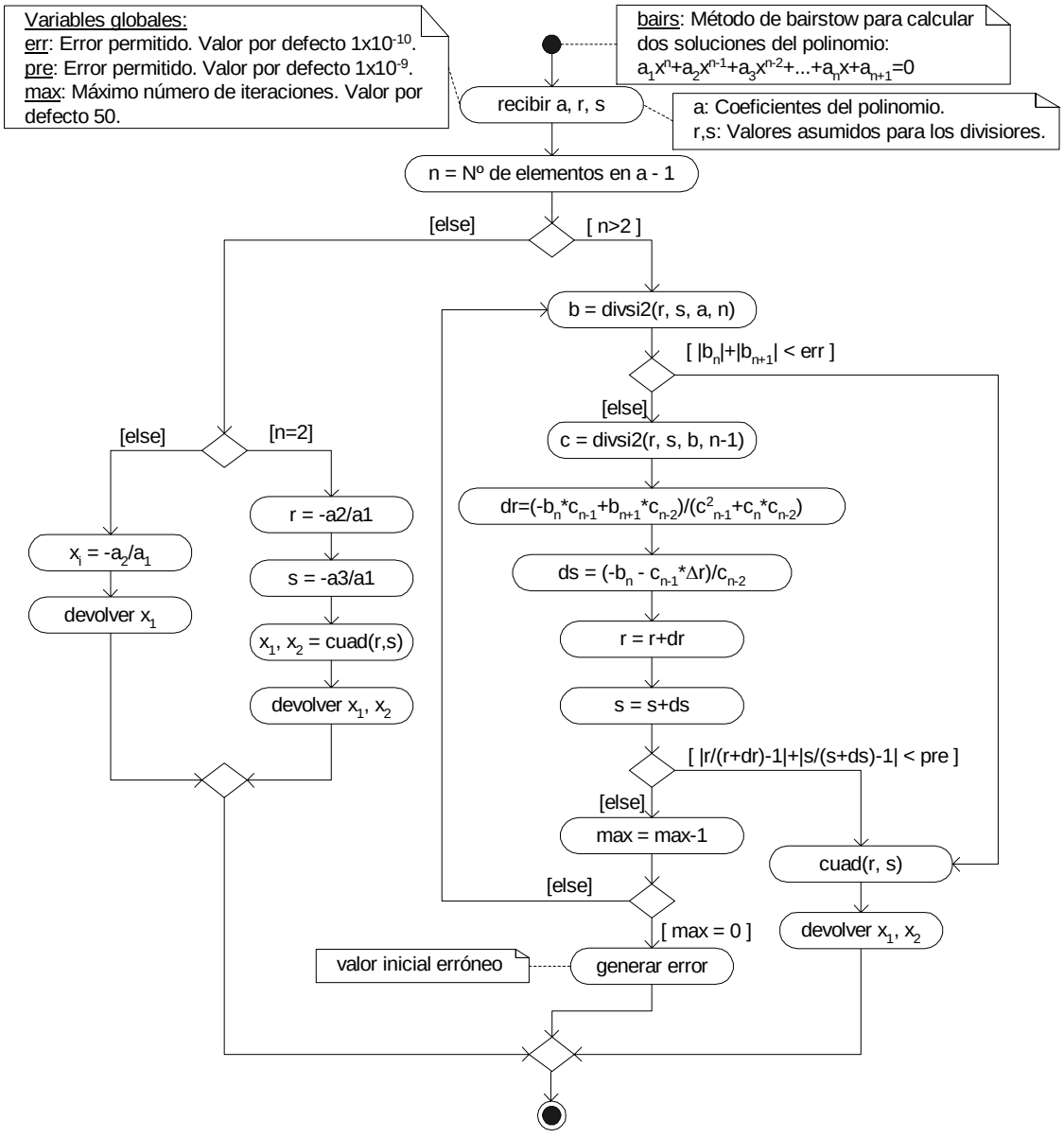
4.7.1. Algoritmo

El algoritmo del método de ***Bairstow*** se presenta en el diagrama de actividades de la siguiente página.

4.7.2. Código

El código elaborado en base al algoritmo es el siguiente:

```
xNAME bairs ( Cálculo de dos raíces de un polinomio por el método de
Bairstow)
::
  CK3&Dispatch ( Datos en la pila: a=coeficientes del polinomio; )
  # 411        ( r,s= coeficientes de la ecuación cuadrática x^2-rx-s)
  ::
    3PICK MDIMSDROP #1- ( n )
    DUP #2 #>
    ITE ( a=14; r=13; s=12; n=11; err=10; pre=9; max=8; bn=7; bn1=6; )
    ::  ( cn1=5; cn2=4; cn=3; dr=2; ds=1 )
      ' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
      ' ID pre @ ?SKIP % 1E-9  ( si no existe "pre" se emplea 1E-9 )
      ' ID max @ ?SKIP % 50    ( si no existe "max" se emplea 50 )
      %0 SEVEN NDUPN DROP
      '  NULLLAM FOURTEEN NDUPN DOBIND
      ::
      BEGIN
        13GETLAM 12GETLAM 14GETLAM TOTEMPOB 11GETLAM #1+ divsi2 ( b )
        11GETLAM PULLREALEL 7PUTLAM ( b[n] )
        11GETLAM #1+ PULLREALEL 6PUTLAM ( b[n+1] )
        7GETLAM %ABS 6GETLAM %ABS %+ 10GETLAM %< IT :: DROP 2RDROP ;
        13GETLAM 12GETLAM ROT 11GETLAM divsi2 ( c )
        11GETLAM #2- PULLREALEL 4PUTLAM ( c[n-2] )
        11GETLAM #1- PULLREALEL 5PUTLAM ( c[n-1] )
        11GETLAM PULLREALEL 3PUTLAM ( c[n] )
        6GETLAM 4GETLAM %* 7GETLAM 5GETLAM %* %- 5GETLAM DUP %*
```




```
    3GETLAM 4GETLAM %* %- %/ 2PUTLAM ( dr)
    7GETLAM %CHS 5GETLAM 2GETLAM %* %- 4GETLAM %/ 1PUTLAM ( ds )
    2GETLAM 13GETLAM 2GETLAM %+ %/ %ABS 1GETLAM 12GETLAM 2GETLAM
    %+ %/ %ABS %+
    9GETLAM %< IT :: DROP 2RDROP ;
    8GETLAM %1- 8PUTLAM
    8GETLAM %0= IT :: ABND # A01 DO#EXIT ;
    13GETLAM 2GETLAM %+ 13PUTLAM
    12GETLAM 1GETLAM %+ 12PUTLAM
    DROP
  AGAIN
  ;
  13GETLAM 12GETLAM xcuad
  ABND
;
::
```

```
      UNROT 2DROP #2=
      ITE
      ::
        TWO PULLREALEL SWAP ONE PULLREALEL SWAP UNROT %/ %CHS
        SWAP THREE PULLREALEL SWAP ONE PULLREALEL SWAP DROP %/ %CHS
        xcuad ( x1, x2 )
      ;
      ::
        TWO PULLREALEL SWAP ONE PULLREALEL SWAP DROP %/ %CHS ( x1 )
      ;
    ;
  ;
;
```

4.7.3. Ejemplos

4.7.3.1. Cálculo de dos raíces reales

Como primer ejemplo encontraremos dos raíces de la ecuación:

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0$$

Emplearemos los resultados calculados con el método QD para estimar los valores iniciales:

```
x1: 3.80602336789E-2
x2: .308522093581
```

La ecuación cuadrática correspondiente a estos dos resultados se obtiene con:

```
[ 3.80602336789E-2 0.308522093581 ] PCOEF
=> [ 1. -.34658232726 1.17424229768E-2 ]
```

Por lo tanto los valores iniciales son: r= 0.34658232726; s= -1.17424229768E-2.

El programa que resuelve el problema es el siguiente:

```
« [ 128 -256 160 -32 1 ] .34658232726 -1.17424229768E-2
  bairs
  "x2" →TAG SWAP
  "x1" →TAG
»
```

Haciendo correr el programa con ***TEVAL*** se obtiene:

```
x2: .30865828382
x1: .038060233744
s: .5659
```

Que son dos de las cuatro soluciones y el tiempo empleado en obtenerlas.

4.7.3.2. Soluciones reales de una ecuación de octavo grado

Como segundo ejemplo encontraremos las soluciones reales de la siguiente ecuación, empleando como valores iniciales los valores aproximados obtenidos con el método ***QD***.

$$P_8(x) = x^8 - 60x^7 + 1554x^6 - 22680x^5 + 203889x^4 - 1155420x^3 + 4028156x^2 - 7893840x - 6652800 = 0$$

El programa que resuelve el problema es el siguiente:

```
« [ 1. -60. 1554. -22680. 203889. -1155420. 4028156. -7893840. 6652800. ]
```

```
   0
  → a x
  « 0.01 'err' STO
    a qd 'x' STO
    1 8 FOR i
      a x i GET x i 1 + GET
      2 →ARRY PCOEF NEG
      V→ ROT DROP bairs
    2 STEP
    8 →LIST
    { x1 x2 x3 x4 x5 x6 x7 x8 } →TAG
    LIST→ DROP
    'err' PURGE
  »
»
```

Haciendo correr el programa con ***TEVAL*** se obtiene:

```
x1: 4.00000092622
x2: 4.99999501225
x3: 5.999998815
x4: 7.0000036025
x5: 8.000000515
x6: 9.0000024725
x7: 3.99999641887
x8: 11.0000004712
s: 11.165
```

Que son las soluciones y el tiempo empleado en obtenerlas. Como puede observar en el séptimo resultado el método nuevamente converge hacia la primera solución y no como era de esperar a la penúltima (10.0).

4.7.3.3. Soluciones de una ecuación de quinto grado

Como tercer ejemplo encontraremos las soluciones de la ecuación:

$$P_5(x) = x^5 - 11x^4 + 43x^3 - 79x^2 + 76x - 30 = 0$$

Emplearemos como valores iniciales los resultados calculados con ***"qd"***. Como es polinomio impar y el método de ***Bairstow*** calcula pares de resultados, una de las raíces será calculada con el método de ***Newton - Raphson***.

De una corrida previa empleando ***"qd"*** se sabe que la segunda y tercera raíces son imaginarias y el resto reales. El programa que resuelve el problema es el siguiente:

```
« [ 1. -11. 43. -79. 76. -30. ] DUP SIZE LIST→ 1 - 1 → a n x
  « a qd 'x' STO x 1 GET TYPE
    IF 1 == THEN
      1 n FOR i x i GET IM
        IF 0 == THEN a x i GET RE Newtp
        ELSE a x i GET x i 1 + GET 2 →ARRY PCOEF NEG
         V→ ROT DROP bairs 1 'i' STO+ END
      NEXT
    ELSE
      1 n FOR i a x i GET Newtp NEXT END
    5 →LIST { x1 x2 x3 x4 x5 } →TAG
    LIST→ DROP
  »
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: .999999999993
x2: (.99999999999,1.)
x3: (.99999999999,-1.)
x4: 3.00000000006
x5: 5.
s: 5.2086
```

Que son las cinco soluciones y el tiempo empleado en obtenerlas.

4.7.4. Ejercicios

35. Empleando como valores iniciales los resultados obtenidos con QD y el método de Bairstow encuentre las soluciones del polinomio de Chebyshev:

$$P_6(x) = 32x^6 - 48x^4 + 18x^2 - 1 = 0$$

36. Empleando como valores iniciales los resultados obtenidos con QD y el método de Bairstow encuentre las soluciones del polinomio de Legendre:

$$P_6(x) = 693x^6 - 945x^4 + 315x^2 - 15 = 0$$

37. Empleando como valores iniciales los resultados obtenidos con QD y el método de Bairstow encuentre las soluciones del polinomio de Laguerre:

$$P_4(x) = x^4 - 16x^3 + 72x^2 - 96x + 24 = 0$$

38. Empleando como valores iniciales los resultados obtenidos con QD y el método de Bairstow encuentre las soluciones del polinomio:

$$P_6(x) = x^6 + x^5 - 8x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

39. Empleando como valores iniciales los resultados obtenidos con QD y el método de Bairstow encuentre las soluciones del polinomio:

$$P_8(x) = x^8 - 7x^7 + 11x^6 + 41x^5 - 183x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

40. Empleando como valores iniciales los resultados obtenidos con QD y el método de Bairstow encuentre las soluciones del polinomio:

$$P_8(x) = x^8 + x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$$

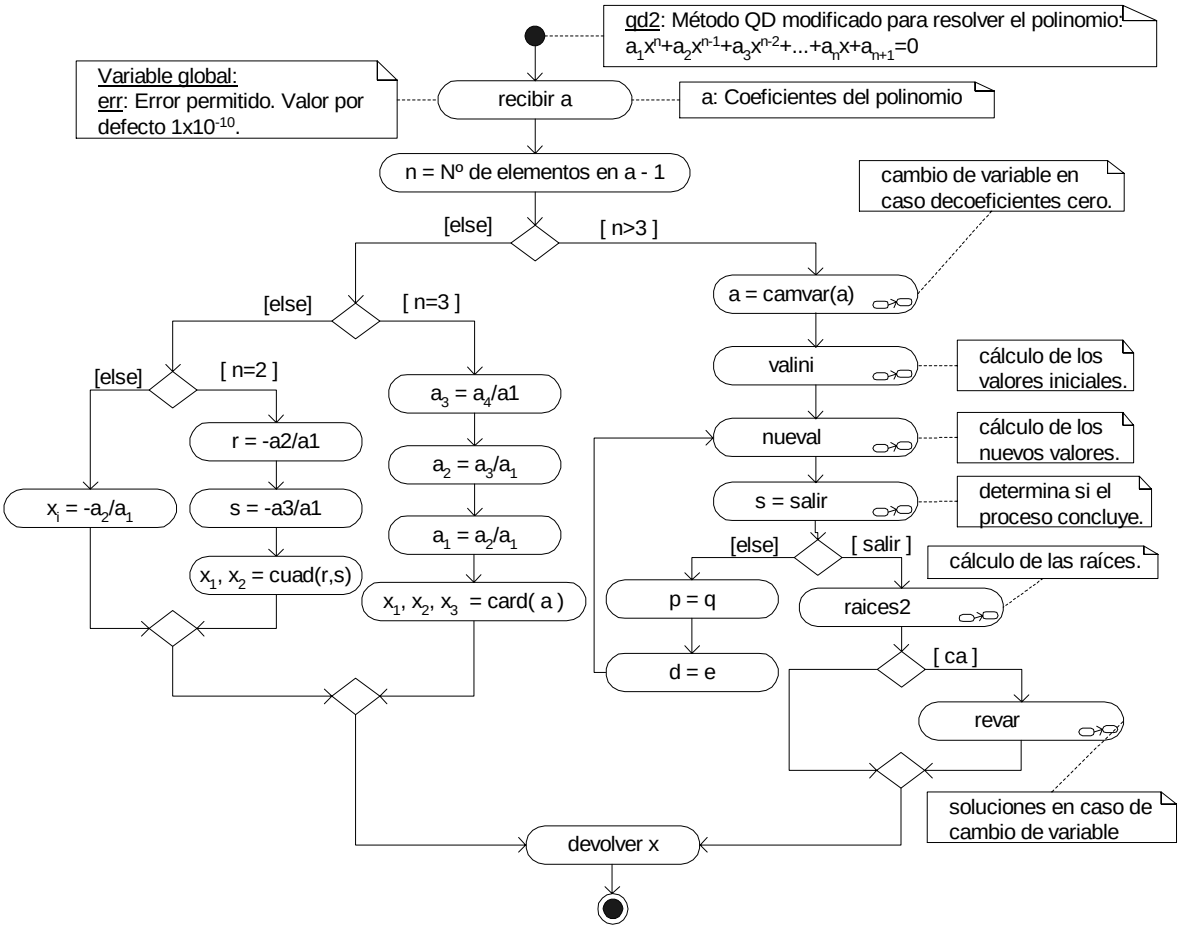
4.8. Método QD ampliado

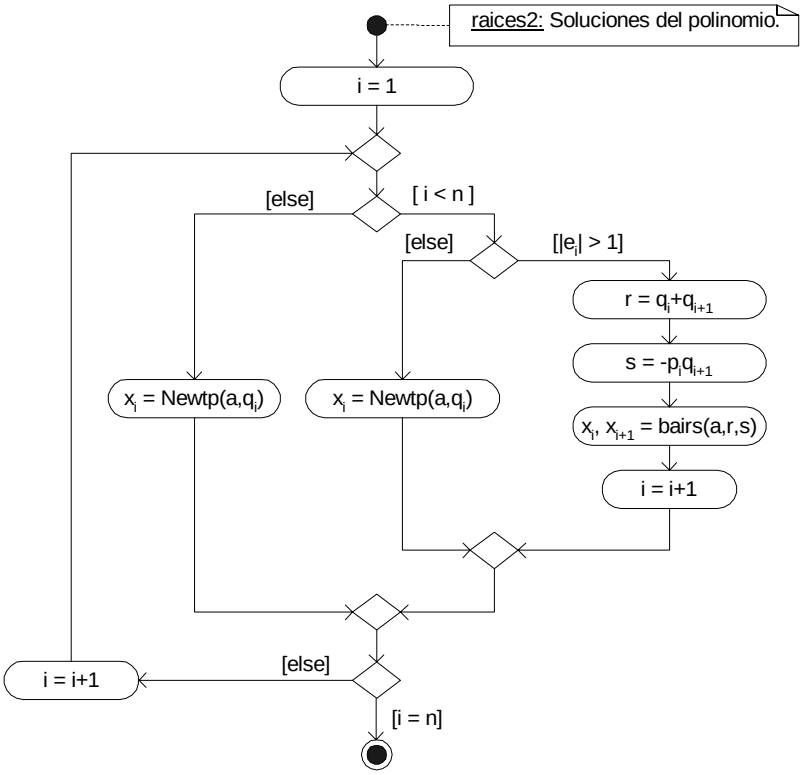
Ahora que contamos con algunos métodos para la resolución de ecuaciones polinomiales, podemos modificar el método *QD* de manera que encuentre no sólo soluciones aproximadas, sino soluciones prácticamente exactas.

Para ello y como antes, si el polinomio es de primer grado la solución se encuentra simplemente despejando la incógnita, si es de segundo se calcula con *"cuad"* y si es de tercero con *"card"*. Para ecuaciones de cuarto grado o superiores los valores iniciales se calculan con el método *QD* y una vez cumplido con el error permitido las soluciones reales se calculan con el método de *Newton - Raphson* y las imaginarias con el método de *Bairstow.*

4.8.1. Algoritmo

Para tomar en cuenta las anteriores consideraciones, sólo es necesario modificar la lógica del módulo principal *("qd")* y del submódulo donde se calculan las raíces *("raices")*. Los algoritmos modificados se presentan en los diagramas de actividades de la siguiente página.


qd2: Método QD modificado para resolver el polinomio:
$a_1x^n+a_2x^{n-1}+a_3x^{n-2}+...+a_nx+a_{n+1}=0$
Variable global:
err: Error permitido. Valor por defecto $1x10^{-10}$.
recibir a
a: Coeficientes del polinomio
n = Nº de elementos en a - 1
cambio de variable en caso decoeficientes cero.
[else]
[n>3]
a = camvar(a)
valini
cálculo de los valores iniciales.
nueval
cálculo de los nuevos valores.
s = salir
determina si el proceso concluye.
[else]
[salir]
cálculo de las raíces.
raices2
[ca]
revar
soluciones en caso de cambio de variable
p = q
d = e
[else]
[n=3]
$a_3 = a_4/a1$
$a_2 = a_3/a_1$
$a_1 = a_2/a_1$
x_1, x_2, x_3 = card(a)
[else]
[n=2]
r = -a2/a1
s = -a3/a1
x_1, x_2 = cuad(r,s)
$x_1 = -a_2/a_1$
devolver x


raices2: Soluciones del polinomio.
i = 1
[else]
[i < n]
[else]
[$|e_i|$ > 1]
$r = q_i+q_{i+1}$
$s = -p_iq_{i+1}$
x_i, x_{i+1} = bairs(a,r,s)
i = i+1
x_i = Newtp(a,q_i)
x_i = Newtp(a,q_i)
[else]
i = i+1
[i = n]

4.8.2. Código

El código de los dos módulos modificados es el siguiente:

```
xNAME qd2 ( método Q-D modificado )
::
  CK1&Dispatch
  FOUR
  ::
    DUP MDIMSDROP #1- ( n )
    DUP #3 #>
    ITE
    ::  ( 8=a; 7=n; 6=p; 5=q; 4=d; 3=e; 2=ca; 1=err )
      %0 OVER NDUPN UNCOERCE ONE{}N FLASHPTR XEQ>ARRY ( p )
      DUP TOTEMPOB ( q )
      %0 4PICK #1- NDUPN UNCOERCE ONE{}N FLASHPTR XEQ>ARRY ( d )
      DUP TOTEMPOB ( e )
      FALSE
      ' ID err @ ?SKIP % 1E-10 ( si no existe "err" se emplea 1E-10 )
      ' NULLLAM EIGHT NDUPN DOBIND
      8GETLAM camvar
      valini
      ::
        BEGIN
          nueval
          salir
          IT
          ::
            raices2
            2GETLAM IT revar
            7GETLAM
            2RDROP
          ;
          5GETLAM TOTEMPOB 6PUTLAM
          3GETLAM TOTEMPOB 4PUTLAM
        AGAIN
      ;
      ABND
    ;
    ::
      SWAP OVER #3=
      ITE
      ::
        FOUR PULLREALEL SWAP ONE PULLREALEL SWAP UNROT %/ ( a3)
        SWAP THREE PULLREALEL SWAP ONE PULLREALEL SWAP UNROT %/ ( a2 )
        UNROT TWO PULLREALEL SWAP ONE PULLREALEL SWAP DROP %/ ( a1 )
        UNROT xcard 4ROLL
      ;
      ::
        OVER #2=
        ITE
        ::
          TWO PULLREALEL SWAP ONE PULLREALEL SWAP UNROT %/ %CHS
          SWAP THREE PULLREALEL SWAP ONE PULLREALEL SWAP DROP %/ %CHS
          xcuad ROT
        ;
        ::
          TWO PULLREALEL SWAP ONE PULLREALEL SWAP DROP %/ %CHS SWAP
        ;
```

```
      ;
    ;
    UNCOERCE ONE{}N FLASHPTR XEQ>ARRY
  ;
;

NULLNAME raices2 ( soluciones del polinomio: submódulo de QD)
::
  6GETLAM 5GETLAM 3GETLAM ( p, q, e )
  7GETLAM #1+ ONE DO
    INDEX@ 7GETLAM #<
    ITE
    ::
      INDEX@ PULLREALEL %ABS %1 %>
      ITE
      ::
        UNROT INDEX@ PULLREALEL SWAP INDEX@ #1+ PULLREALEL SWAP
        5UNROLL DUP ROT %+ ( r=q[i]+q[i+1] )
        ROT INDEX@ PULLREALEL 4ROLL %* %CHS SWAP 5UNROLL ( s=-p[i]q[i+1] )
        8GETLAM UNROT xbairs ( xi, xi+1 )
        FOUR INDEX@ #+ UNROLL FOUR INDEX@ #+ UNROLL
        INDEX@ #1+ INDEXSTO
      ;
      ::
        SWAP INDEX@ PULLREALEL
        8GETLAM SWAP xNewtp THREE INDEX@ #+ UNROLL SWAP ( xi=qi)
      ;
    ;
    ::
      SWAP INDEX@ PULLREALEL
      8GETLAM SWAP xNewtp THREE INDEX@ #+ UNROLL SWAP ( xi=qi)
    ;
  LOOP
  3DROP
;
```

4.8.3. Ejemplos

Resolveremos las mismas ecuaciones que en el método ***QD*** sin modificar.

4.8.3.1. Resolución de una ecuación de cuarto grado con soluciones reales

Como primer ejemplo resolveremos la ecuación:

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0$$

```
« 1E-10 'err' STO
  [ 128 -256 160 -32 1 ]
  qd2
  AXL
  { x1 x2 x3 x4 } →TAG
  LIST→ DROP 'err' PURGE
»
```

Haciendo correr el programa con ***TEVAL*** se obtiene:

```
x1: 3.80602337444E-2
x2: .308658283815
x3: .691341716178
```

```
x4: .961939766246
s: 2.5645
```

Que son las soluciones y el tiempo empleado en obtenerlas.

4.8.3.2. Resolución de una ecuación de cuarto grado con soluciones imaginarias

Como segundo ejemplo resolveremos la ecuación:

$$P_4(x) = x^4 - 6x^3 + 12x^2 - 19x + 12 = 0$$

```
« 1E-7 'err' STO
  [ 1 -6 12 -19 12 ]
  qd2
  AXL
  { x1 x2 x3 x4 } →TAG
  LIST→ DROP 'err' PURGE
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: (.999999999998,0.)
x2: (.5,1.65831239518)
x3: (.5,-1.65831239518)
x4: (4.,0.)
s: 2.6378
```

Que son las soluciones y el tiempo empleado en obtenerlas.

4.8.3.3. Resolución de una ecuación de cuarto grado con coeficientes cero

Como tercer ejemplo resolveremos la ecuación:

$$P_4(x) = x^4 - 3x^2 + x - 5 = 0$$

```
« 1E-9 'err' STO
  [ 1 0 -3 1 -5 ]
  qd2
  AXL
  { x1 x2 x3 x4 } →TAG
  LIST→ DROP 'err' PURGE
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: (1.94992054774,0.)
x2: (.093181113525,1.09161799092)
x3: (.093181113525,-1.09161799092)
x4: (-2.13628277479,0.)
s: 3.6563
```

Que son las soluciones y el tiempo empleado en obtenerlas.

4.8.3.4. Resolución de una ecuación de octavo grado

Como cuarto ejemplo resolveremos la ecuación:

$$P_8(x) = x^8 + x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$$

```
« 1E-7 'pre' STO
  [ 1 1 -9 13 21 -125 77 111 -90 ]
  qd2
```

```
  AXL
  { x1 x2 x3 x4 x5 x6 x7 x8 } →TAG
  LIST→ DROP 'pre' PURGE
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: (.999999838753,0.)
x2: (-1.,0.)
x3: (1.00000065331,0.)
x4: (2.,0.)
x5: (1.,2.)
x6: (1.,-2.)
x7: (-2.99999802619,0.)
x8: (-3.00000121778,0.)
s: 22.9028
```

Que son las soluciones y el tiempo empleado en obtenerlas.

4.8.4. Ejercicios

Resuelva las siguientes ecuaciones empleando el método QD modificado.

41. Encuentre las soluciones del polinomio de Chebyshev:

$$P_6(x) = 32x^6 - 48x^4 + 18x^2 - 1 = 0$$

42. Encuentre las soluciones del polinomio de Legendre:

$$P_6(x) = 693x^6 - 945x^4 + 315x^2 - 15 = 0$$

43. Encuentre las soluciones del polinomio de Laguerre:

$$P_4(x) = x^4 - 16x^3 + 72x^2 - 96x + 24 = 0$$

44. Encuentre las soluciones de la ecuación cúbica:

$$P_3(x) = x^3 + 2x^2 + 3x + 4 = 0$$

45. Encuentre las soluciones de la ecuación:

$$P_4(x) = x^4 + 4x^3 + 21x^2 + 4x + 20 = 0$$

46. Encuentre las soluciones de la ecuación:

$$P_6(x) = x^6 + x^5 - 8x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

5.SISTEMAS DE ECUACIONES LINEALES

Con frecuencia en el campo de la ingeniería es necesario resolver sistemas de ecuaciones lineales que tienen entre 2 y miles de ecuaciones lineales. Los métodos no iterativos que estudiaremos en primer lugar nos permiten resolver sistemas de ecuaciones que cuentan con decenas o como máximo algunas centenas de ecuaciones lineales (no por las limitaciones de los métodos en si, sino por errores de redondeo).

Posteriormente estudiaremos dos métodos iterativos, con los cuales podremos resolver sistemas con cientos o miles de ecuaciones lineales.

5.1. Solución de ecuaciones lineales con programas existentes para la calculadora

La calculadora HP tiene incorporada una función que permite resolver sistemas de ecuaciones lineales, la función *RREF*, que recibe como dato la matriz del sistema de ecuaciones y devuelve como resultado una matriz con unos en la diagonal principal y las soluciones del sistema de ecuaciones en la última columna. Así por ejemplo para resolver el siguiente sistema de ecuaciones lineales:

$$\begin{array}{rcrcrcr} 2x_1 & + & 8x_2 & + & 2x_3 & = & 14 \\ x_1 & + & 6x_2 & - & x_3 & = & 13 \\ 2x_1 & - & x_2 & + & 2x_3 & = & 5 \end{array} \qquad (1)$$

Escribimos lo siguiente:

```
[[ 2 8 2 14 ]
 [ 1 6 -1 13 ]
 [ 2 -1 2 5 ]] RREF
```

Con lo cual obtenemos la matriz reducida:

```
[[ 1 1 0 5 ]
 [ 0 1 0 1 ]
 [ 0 1 1 -2 ]]
```

Por lo tanto las soluciones del sistema de ecuaciones son: $x_1 = 5$; $x_2 = 1$ y $x_3 = -2$.

Podemos obtener directamente las soluciones elaborando un programa:

```
« [[ 2 8 2 14 ]
   [ 1 6 -1 13 ]
   [ 2 -1 2 5 ]] RREF
   4 COL- SWAP DROP AXL
   {x1 x2 x3 } →TAG
   LIST→ DROP
»
```

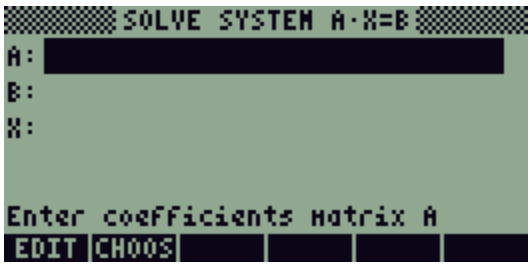
Y haciendo correr este programa con *TEVAL* se obtiene:

```
x1: 5.
x2: 1.
x3: -2.
s: .3126
```

Que son las soluciones y el tiempo empleado en obtenerlas.

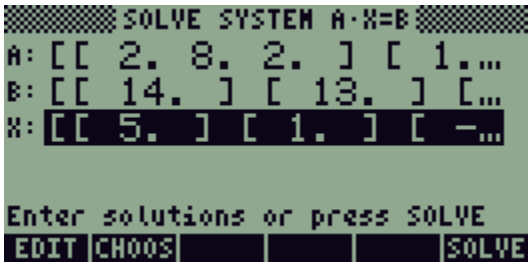
Se puede emplear también el solucionador numérico incorporado (*SOLVE* en la HP48 y *NUM.SLV* en la HP49). Cuando se ingresa aparece una caja de selec-

ción con varias opciones, de las cuales se debe seleccionar la cuarta: "*Solve lin sys…*" con lo que aparece la siguiente pantalla:



Para resolver un sistema de ecuaciones lineales, en el campo "*A*" se introduce la matriz de los coeficientes y en el campo "*B*" el vector de las constantes y estando en el campo "*X*" se pulsa la tecla correspondiente a la opcion *SOLVE* (resolver), con lo que el sistema es resuelto y los resultados aparecen en este campo.

Así para resolver el sistema de ecuaciones de la anterior página, se introduce en el campo "*A*" la matriz de los coeficientes: [[2 8 2] [1 6 -1] [2 -1 2]] y en el campo "*B*" el vector de las constantes: [[14] [13] [5]] y estando el curso en el campo *X* se presiona la tecla de *SOLVE (F)*. Entonces en el campo "*X*" aparece el vector con los resultados: [[5] [1] [-2]], tal como se muestra en la siguiente figura:



Pulsando una vez más *ENTER* en la pila queda lo siguiente:

```
Solutions:
[[ 5. ]
 [ 1. ]
 [ -2. ]]
```

Que son las 3 soluciones del sistema.

Podemos igualmente resolver sistemas de ecuaciones lineales si recordamos que las soluciones pueden ser encontradas multiplicando la inversa de la matriz de los coeficientes por el vector de las constantes:

$$X = A^{-1}B \qquad (2)$$

Donde X es el vector de las soluciones, A^{-1} es la inversa de la matriz de los coeficientes y B es la matriz de las constantes. Así el sistema de ecuaciones lineales (1) puede ser resuelto de la siguiente manera:

```
[[ 2 8 2] [1 6 -1] [2 -1 2]] INV [[14] [13] [5]] *
```

Con lo que obtenemos el vector:

```
[[5]
 [.999999999999]
 [-2]]
```

Que son las tres soluciones del sistema de ecuaciones: $x_1 = 5$; $x_2 \approx 1$ y $x_3 = -2$.

Para resolver unsistema de ecuaciones lineales se puede emplear también la librería *SolveSys*, que puede ser bajada gratuitamente del sitio: www.hpcalc.org.

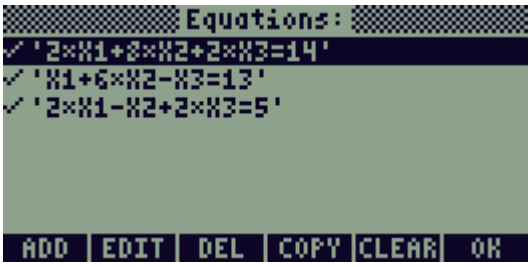
SolveSys ocupa menos de 4 kb y está disponible tanto para la HP48 como para la HP49. Esta librería permite resolver sistemas de ecuaciones lineales, sistemas de ecuaciones no lineales y realizar ajustes de curvas por el método de los mínimos cuadrados (razón por la cual emplearemos esta librería también en otros capítulos).

Al instalar esta librería en la HP49 se añade como la última opción del solucionador numérico *(NUM.SLV)*, en la HP48 se accede de la forma habitual (a través de la opción *LIBRARY*, ingresando a *SOLVE* y dentro a *SOLVESYS*).

Al ingresar a SOLVESYS, aparece un formulario para la introducción de ecuaciones como el que se muestra en la figura:

En este formulario es donde se presentan las ecuaciones del sistema a resolver. La opción *ADD* permite añadir una nueva ecuación al sistema, la opción *Edit* permite modificar una ecuación existente, la opción *DEL* borra la ecuación que está resaltada, la opción *COPY* duplica la ecuación que está resaltada, la opción *CLEAR* borra todas las ecuaciones del sistema y la opción *OK* pasa al siguiente formulario de *SolveSys*.

Así para resolver el sistema de ecuaciones lineales (1), añadimos (con *ADD*) las tres ecuaciones del sistema:



Entonces pasamos al siguiente formulario de *SolveSys* pulsando la opción *OK*:

En este formulario se pueden asignar valores iniciales a las variables del sistema, esto es importante cuando se resuelve un sistema de ecuaciones no lineales, pero para un sistema de ecuaciones lineales como el nuestro,

los valores iniciales no son importantes, pues realmente no son utilizados en la solución del sistema.

En este formulario, la opción *EDIT*, permite modificar el valor de la variable que está resaltada, la opción *RESET* asigna a las variables valores iniciales por defecto (sólo en la HP49), la opción *INFO* nos muestra en un recuadro, como el de la siguiente figura, información con relación al número de ecuaciones y variables (M/n), el error entre dos valores sucesivos de las variables *(Δx)*, el error en la igualdad de las ecuaciones *(eq)* y el error permitido cuando se realiza un ajuste por el método de los mínimos cuadrados *(lsq)*.

La opción *TOL* nos permite fijar los errores máximos permitidos para los valores antes mencionados. Cuando se elige esta opción aparece un recuadro como el de la siguiente figura:

Al pulsar *OK* aparece un recuadro que permite elegir el error máximo permitido para el tipo de error resaltado. Esta opción es útil sólo cuando se resuelven sistema de ecuaciones no lineales o se realiza ajuste de datos. En nuestro caso no se requiere modificar ninguno de estos valores.

La opción *→STK*, copia los resultados del sistema a la pila en forma de una lista.

La opción *SOLVE* resuelve el sistema de ecuaciones. Cuando *SolveSys* ha encontrado las soluciones aparece un mensaje como el que se muestra en la siguiente figura:

El cual nos informa que el error es cero y que en consecuencia los resultados son exactos.

Finalmente, al pulsar *OK*, los resultados aparecen en lugar de los valores iniciales asumidos, tal como se muestra en la figura de la siguiente página.

5.2. Método de eliminación de Gauss

El método de eliminación de Gauss transforma el sistema de ecuaciones lineales en un sistema triangular superior. Así el siguiente sistema de 4 ecuaciones lineales:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 &= c_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + a_{2,4}x_4 &= c_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + a_{3,4}x_4 &= c_3 \\ a_{4,1}x_1 + a_{4,2}x_2 + a_{4,3}x_3 + a_{4,4}x_4 &= c_4 \end{aligned} \qquad (3)$$

Se transforma en el siguiente sistema triangular superior:

$$\begin{aligned} x_1 + a'_{1,2}x_2 + a'_{1,3}x_3 + a'_{1,4}x_4 &= c'_1 \\ 0 + x_2 + a'_{2,3}x_3 + a'_{2,4}x_4 &= c'_2 \\ 0 + 0 + x_3 + a'_{3,4}x_4 &= c'_3 \\ 0 + 0 + 0 + x_4 &= c'_4 \end{aligned} \qquad (4)$$

Donde los apóstrofes se han empleado para diferenciar los coeficientes originales de los nuevos coeficientes.

Una vez reducido el sistema a la forma triangular superior, las soluciones se calculan por sustitución inversa.

Como se puede observar, el valor de la última variable es directamente el último elemento del vector de constantes, en nuestro caso "c'_4". Entonces como se conoce "x_4", el valor de "x_3" puede ser calculado con la penúltima ecuación:

$$x_3 = c'_3 - a'_{3,4}x_4$$

Ahora que se conocen los valores de "x_4" y "x_3" podemos calcular el valor de "x_2" con la antepenúltima ecuación:

$$x_2 = c'_2 - a'_{2,3}x_3 - a'_{2,4}x_4$$

Finalmente como conocemos los valores de "x_4", "x_3" y "x_2" podemos calcular el valor de "x_1" con la primera ecuación:

$$x_1 = c'_1 - a'_{1,2}x_2 - a'_{1,3}x_3 - a'_{1,4}x_4$$

Procediendo de esta manera (por sustitución inversa) se encuentran las soluciones, sin importar el número de ecuaciones existentes.

En general al implementar el método de eliminación de *Gauss* (y los otros métodos que estudiaremos posteriormente) sólo se trabaja con la matriz de los coeficientes aumentada con el vector de las constantes (matriz aumentada).

Así, en forma matricial, la aplicación del método de *Gauss* al sistema de ecuaciones lineales (1), da lugar a la siguiente cambio:

$$\begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{vmatrix} \xrightarrow{GAUSS} \begin{vmatrix} 1 & a'_{1,2} & a'_{1,3} & a'_{1,4} & a'_{1,5} \\ 0 & 1 & a'_{2,3} & a'_{2,4} & a'_{2,5} \\ 0 & 0 & 1 & a'_{3,4} & a'_{3,5} \\ 0 & 0 & 0 & 1 & a'_{4,5} \end{vmatrix} \quad (5)$$

Donde los elementos de la última columna (la columna 5) son las constantes del sistema de ecuaciones lineales.

Para convertir la matriz aumentada en una matriz diagonal superior, se deben efectuar reducciones de filas (donde se convierten en uno los elementos de la diagonal principal) y reducciones de columnas (donde se convierten en cero los elementos debajo de la diagonal principal). Estas operaciones se deben efectuar primero para la fila y columna 1, luego para la fila y columna 2, luego para la fila y columna 3 y así sucesivamente hasta llegar a la última fila del sistema.

Al programar el método, puesto que los valores originales de la matriz sólo se emplean una vez en los cálculos, no es necesario emplear dos matrices, pues los resultados pueden ser almacenados en la matriz original, ahorrando así memoria.

En las siguientes ecuaciones llamaremos *"p" (pivote)* al contador que determina la fila y columna que se está reduciendo, *"m"* al número de ecuaciones del sistema y *"n"* al número de columnas de la matriz aumentada.

La ecuación general para reducir la fila *"p"*, con lo que se convierte en en "1" el elemento $a_{p,p}$, es la siguiente:

$$a_{pj} = \frac{a_{pj}}{a_{pp}} \quad \{ j = p+1 \rightarrow n \quad (6)$$

Como se puede observar en esta ecuación no se calcula realmente el elemento $a_{p,p}$, pues se sabe que al aplicar la ecuación queda con el valor 1.

La ecuación general para reducir columna *"p"*, con lo que se convierten en cero los elementos que se encuentran debajo del elemento $a_{p,p}$, es la siguiente:

$$a_{ij} = a_{ij} - a_{ip} * a_{pj} \quad \begin{cases} i = p+1 \rightarrow m \\ j = p+1 \rightarrow n, \text{ para cada valor de i} \end{cases} \quad (7)$$

Al igual que en la reducción de filas, en esta ecuación no se calcula realmente el valor de los elementos que se encuentran debajo del elemento $a_{p,p}$, pues se sabe que al aplicar la ecuación quedan con el valor cero.

Una vez reducida la matriz a la forma triangular superior, aplicando las ecuaciones (6) y (7) desde *"p=1"* hasta *"p=n"*, las soluciones del sistema de ecuaciones se calculan por sustitución inversa. La ecuación general para calcular los resultados y almacenarlos en las columnas de las constantes es la siguiente:

$$a_{ij} = a_{ij} - \sum_{k=i+1}^{m} a_{ik} * a_{kj} \quad \begin{cases} i = m-1 \rightarrow 1 \\ j = m+1 \rightarrow n, \text{ para cada valor de i} \end{cases} \quad (8)$$

5.2.1. Ejemplo manual

Para comprender mejor el método y tener valores de referencia que nos permitan hacer correr el programa paso a paso en busca de errores, resolveremos el siguiente sistema de ecuaciones lineales (1).

La matriz aumentada del sistema es:

$$\begin{vmatrix} 2 & 8 & 2 & 14 \\ 1 & 6 & -1 & 13 \\ 2 & -1 & 2 & 5 \end{vmatrix}$$

Comenzamos con *p=1.* Entonces reducimos el elemento $a_{1,1}$ a 1, aplicando la ecuación (6) (como ya se dijo el valor de $a_{1,1}$ realmente no se calcula en el proceso)

$a_{1,2} = a_{1,2}/a_{1,1} = 8/2 = 4$

$a_{1,3} = a_{1,3}/a_{1,1} = 2/2 = 1$

$a_{1,4} = a_{1,4}/a_{1,1} = 14/2 = 7$

Ahora reducimos a cero los elementos que se encuentra debajo del elemento $a_{1,1}$ aplicando la ecuación (7) (una vez más en el proceso no se calculan realmente los valores de los elementos reducidos: $a_{1,2}$ y $a_{1,3}$)

$a_{2,2} = a_{2,2}-a_{2,1}*a_{1,2} = 6-1*4 = 2$

$a_{2,3} = a_{2,3}-a_{2,1}*a_{1,3} = -1-1*1 = -2$

$a_{2,4} = a_{2,4}-a_{2,1}*a_{1,4} = 13-1*7 = 6$

$a_{3,2} = a_{3,2}-a_{3,1}*a_{1,2} = -1-2*4 = -9$

$a_{3,3} = a_{3,3}-a_{3,1}*a_{1,3} = 2-2*1 = 0$

$a_{3,4} = a_{3,4}-a_{3,1}*a_{1,4} = 5-2*7 = -9$

Entonces con la primera aplicación de las ecuaciones (6) y (7) para *"p=1"*, la matriz aumentada queda con los siguientes valores:

$$\begin{vmatrix} 1 & 4 & 1 & 7 \\ 0 & 2 & -2 & 6 \\ 0 & -9 & 0 & -9 \end{vmatrix}$$

Donde no se han calculado realmente el uno y los ceros.

Ahora volvemos a aplicar las ecuaciones (6) y (7) para *"p=2"*. Primero reducimos el elemento $a_{2,2}$ a 1 aplicando la ecuación (6):

$a_{2,3} = a_{2,3}/a_{2,2} = -2/2 = -1$

$a_{3,4} = a_{3,4}/a_{2,2} = 6/2 = 3$

Luego reducimos el elemento que se encuentra debajo del elemento $a_{2,2}$ (el elemento $a_{3,2}$) a cero, aplicando la ecuación (7):

$a_{3,3} = a_{3,3}-a_{3,2}*a_{2,3} = 0-(-9)*(-1) = -9$

$a_{3,4} = a_{3,4}-a_{3,2}*a_{2,4} = -9-(-9)*(3) = 18$

De esa manera, con la segunda aplicación de las ecuaciones (6) y (7) para *"p=2"*, la matriz aumentada queda con los siguientes valores:

$$\begin{vmatrix} 1 & 4 & 1 & 7 \\ 0 & 1 & -1 & 3 \\ 0 & 0 & -9 & 18 \end{vmatrix}$$

Finalmente volvemos a repetir el proceso para *"p=3"*. Reducimos el elemento $a_{3,3}$ a 1 aplicando la ecuación (6):

$a_{3,4} = a_{3,4}/a_{3,3} = 18/(-9) = -2$

Y como ya no existen elementos debajo del elemento $a_{3,3}$, no es necesario aplicar la ecuación (7).

De esa manera la matriz aumentada queda en la forma triangular superior:

$$\begin{vmatrix} 1 & 4 & 1 & 7 \\ 0 & 1 & -1 & 3 \\ 0 & 0 & 1 & -2 \end{vmatrix}$$

Ahora calculamos las soluciones del sistema aplicando la ecuación (8) (sustitución inversa):

$a_{2,4} = a_{2,4} - a_{2,3} * a_{3,4} = 3 - (-1) * (-2) = 1$

$a_{1,4} = a_{1,4} - (a_{1,2} * a_{2,4} + a_{1,3} * a_{3,4}) = 7 - (4 * 1 + 1 * (-2)) = 5$

Finalmente la matriz aumentada queda con los siguientes valores:

$$\begin{vmatrix} 1 & 4 & 1 & 5 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 1 & -2 \end{vmatrix}$$

Donde las soluciones se encuentran en la última columna de matriz aumentada: $x_1 = 5$; $x_2 = 1$ y $x_3 = -2$. Como se ha señalado en reiteradas ocasiones, en el proceso no se calculan los unos de la diagonal principal ni los ceros debajo la misma, pues realmente no son necesarios, por lo tanto la matriz aumentada no queda con los ceros y unos que se muestran en este ejemplo.

5.2.2. Pivotaje

Al aplicar el método de eliminación de *Gauss* en ocasiones puede ocurrir que el elemento de la diagonal principal (el elemento pivote) sea cero, en cuyo caso se produciría un error por división entre cero. Entonces se debe realizar un intercambio de filas (y en ocasiones columnas) para que el elemento pivote sea diferente de cero.

Por otra parte es conveniente que el elemento pivote tenga el mayor valor absoluto posible, pues de esa manera se reducen considerablemente los errores de redondeo. Para ello con frecuencia se deben realizar intercambios de filas y columnas a fin de colocar el elemento con el mayor valor absoluto en la posición pivote.

Al proceso de buscar (en las filas y columnas no reducidas) el elemento con el mayor valor absoluto e intercambiar filas y columnas de manera que dicho valor quede en la posición pivote se conoce como *pivotaje total*. Si la búsqueda del mayor valor absoluto se realiza sólo en la columna pivote, el proceso se conoce como *pivotaje parcial*.

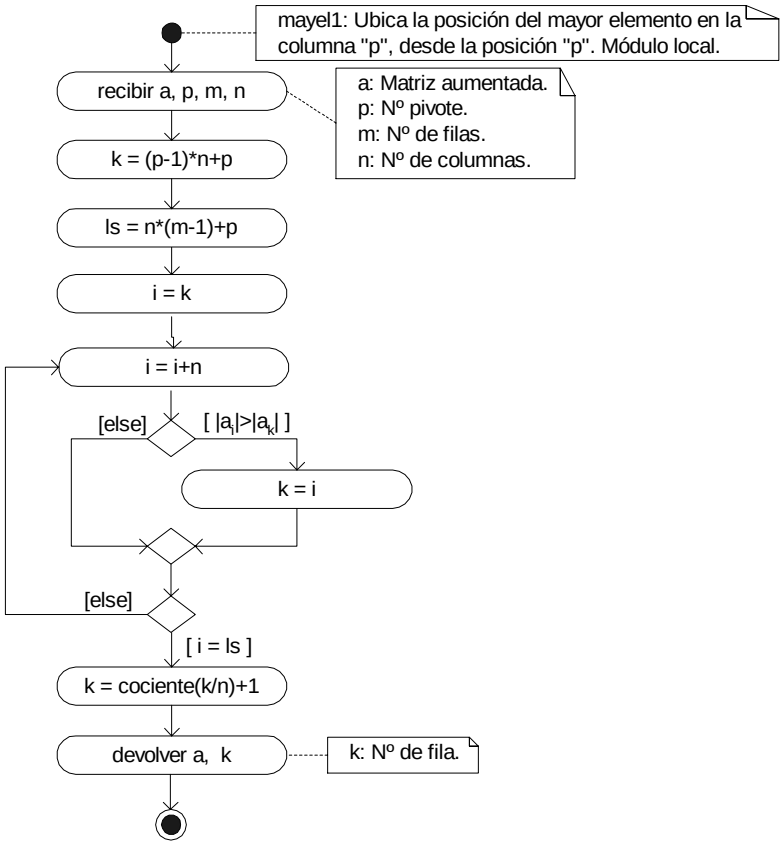
Al implementar el método de eliminación de *Gauss,* es necesario llevar a cabo por lo menos un pivotaje parcial, pues de otra manera se pueden producir divisiones entre cero y los errores de redondeo pueden ser muy grandes dando lugar a resultados erróneos.

5.2.2.1. Pivotaje parcial

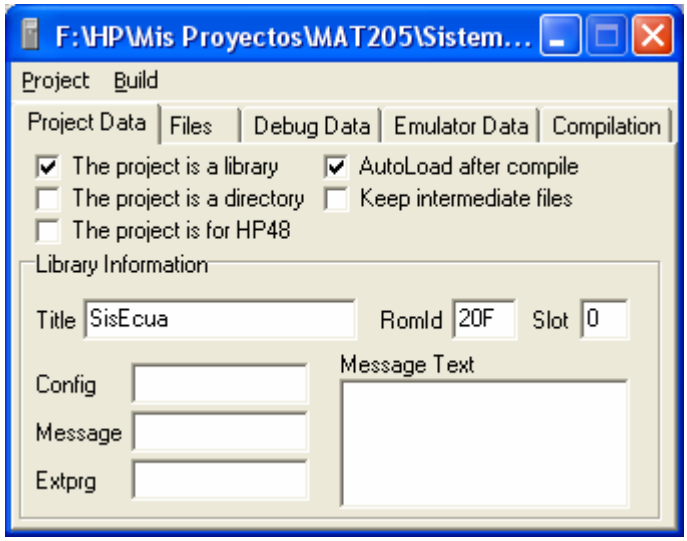
El algoritmo para realizar el pivotaje parcial es muy sencillo: en la columna pivote, se ubica la fila donde se encuentra el elemento con el mayor valor absoluto, buscando sólo desde la fila pivote hasta la última fila. Una vez ubicado el elemento, se intercambia la fila pivote con la fila donde se encuentra el mayor valor absoluto.

Para facilitar la resolución del problema y reducir la probabilidad de error, es conveniente realizar la búsqueda del mayor valor absoluto en un módulo y el intercambio de filas en otro.

El algoritmo que busca el mayor valor absoluto en una columna, a partir de la fila pivote es el siguiente:



Para escribir los programas de este capítulo deberá crear un nuevo proyecto con el nombre SisEcua, el número de librería 20F (527 en decimal) y añadir al mismo (File -> New Source File) el archivo SisEcua.s.



Como se especifica en el diagrama de actividades *"mayel1"* es un módulo local, por lo tanto sólo puede ser llamado desde otro módulo dentro del proyecto, pero no desde otros proyectos o desde el emulador. Los módulos locales se crean con la palabra reservada *NULLNAME*, así el código para *"mayel1"* es el siguiente:

```
NULLNAME mayel1 ( Búsqueda del mayor elemento en la columna pivote )
::                          ( datos: a= Matriz aumentada)
  BINT0 BINT0 BINT0               ( p=6= N° pivote )
  ' NULLLAM SIX NDUPN DOBIND      ( m=5= N° de filas )
  6GETLAM #1- 4GETLAM #*          ( n=4= N° de columnas )
  6GETLAM #+ 1PUTLAM              ( Locales: ls=3; i=2; k=1= [p-1]*n+p )
  4GETLAM 5GETLAM #1- #*
  6GETLAM #+ 3PUTLAM              ( ls = n*[m-1]+p )
  1GETLAM 2PUTLAM                 ( i = ls )
  BEGIN
    2GETLAM 4GETLAM #+ 2PUTLAM    ( i = i+n )
    1GETLAM PULLREALEL %ABS       ( |a[k]| )
    SWAP 2GETLAM PULLREALEL %ABS  ( |a[i]| )
    ROT
    %> IT :: 2GETLAM 1PUTLAM ;    ( |a[i]|>|a[k]|? => k=i )
    2GETLAM 3GETLAM #=            ( i=ls? )
  UNTIL
  1GETLAM 4GETLAM #/
  SWAP DROP #1+                   ( resíduo de [k/n]+1 )
  ABND
;
```

Para probar este tipo de módulos se debe escribir en el proyecto un módulo de prueba externo (xNAME) y desde el mismo llamar al módulo local, así por ejemplo para probar *"mayel1"* se puede escribir el siguiente módulo:

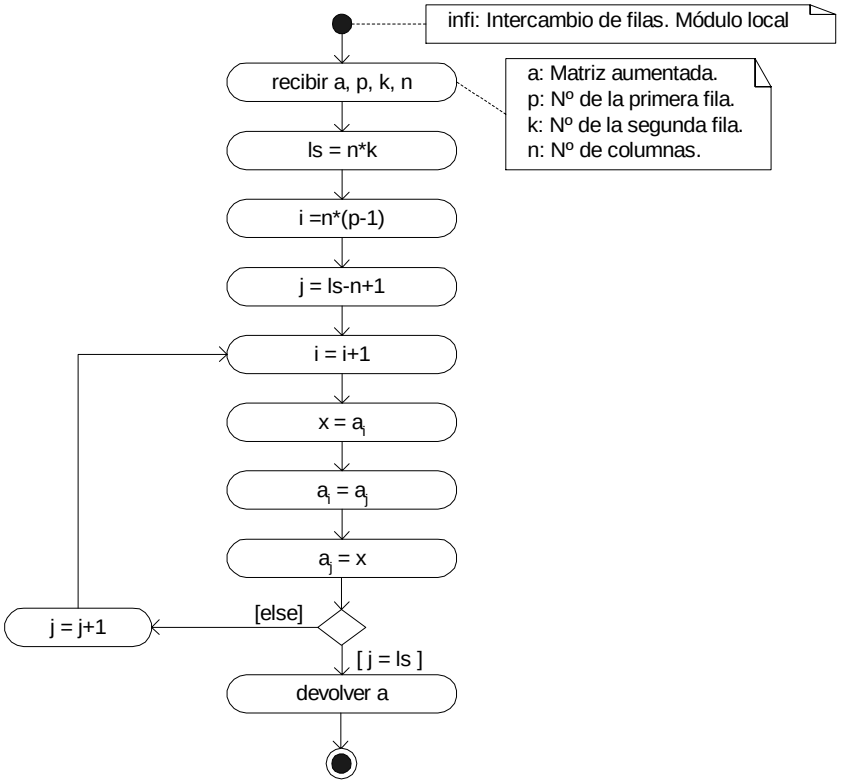
```
xNAME prue ( módulo de prueba )
::
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %24 %15 %16 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  TWO FOUR SIX mayel1
;
```

Donde se crea una matriz de 4 filas por 6 columnas y se manda a buscar el mayor elemento en la segundo columna (TWO). Al hacer correr el programa (desde el emulador) se obtiene la matriz y el número 3 (en binario), indicándonos que el mayor elemento en la segunda columna se encuentra en la tercera fila, lo cual es correcto. Para realizar otras pruebas debe modificar el módulo *"prue"*, compilar el proyecto y hacer correr el programa desde el emulador.

El algoritmo para intercambiar dos filas dadas se presenta en el diagrama de actividades de la siguiente página y el código elaborado en base al mismo es el siguiente:

```
NULLNAME infi ( intercambio de filas )
::                              ( Datos: a = Matriz aumentada)
  BINT0 BINT0 %0                      ( p=6= N° de la primera fila )
  ' NULLLAM SIX NDUPN DOBIND          ( k=5= N° de la segunda fila )
  4GETLAM 5GETLAM #* 3PUTLAM          ( n=4= N° de columnas )
  4GETLAM 6GETLAM #1- #* 2PUTLAM      ( Locales: ls=3; i=2; x=1 )
  3GETLAM #1+                         ( lim. sup.= ls )
  3GETLAM 4GETLAM #- #1+              ( lim. inf.= ls-n+1 )
  DO                                  ( ciclo j )
    2GETLAM #1+ 2PUTLAM               ( i=i+1 )
    2GETLAM PULLREALEL 1PUTLAM        ( x=a[i] )
    INDEX@ PULLREALEL 2GETLAM PUTREALEL ( a[i]=a[j] )
    1GETLAM INDEX@ PUTREALEL          ( a[j]=x )
```

```
  LOOP
  ABND
;
```



Como es un módulo local, puede ser probado con el módulo *"prue"*, así la siguiente modificación emplea el módulo *"infi"* para intercambiar las filas uno (ONE) y tres (THREE) de la matriz:

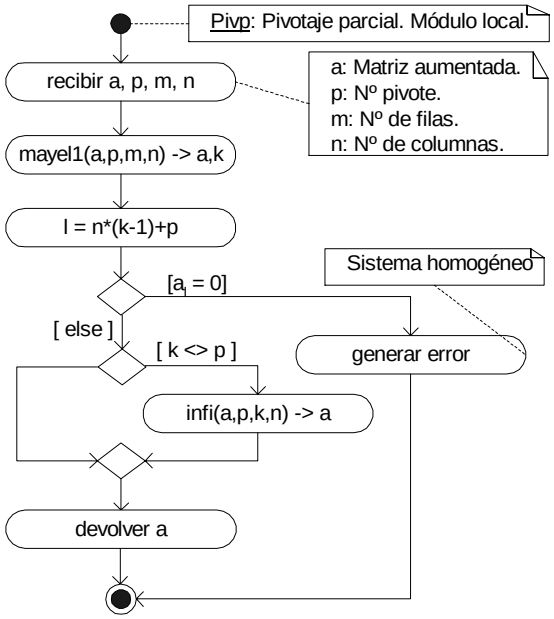
```
xNAME prue ( módulo de prueba )
::
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %24 %15 %16 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  ONE THREE SIX infi
;
```

Al hacer correr *"prue"* (desde el emulador) se obtiene la matriz de 4 por 6 con las filas uno y tres intercambiadas.

Con los módulos para ubicar el mayor elemento e intercambiar filas se puede elaborar el módulo para realizar el pivotaje parcial. El algoritmo de dicho módulo se presenta en el diagrama de actividades de la siguiente página y el código elaborado en base al mismo es el siguiente:

```
NULLNAME pivp ( Pivotaje parcial )
::                              ( Datos: a= Matriz aumentada)
  BINT0                                ( p=4= N° pivote )
  ' NULLLAM FOUR NDUPN DOBIND          ( m=3= N° de filas )
  4GETLAM 3GETLAM 2GETLAM              ( n=2= N° de columnas )
  mayel1 1PUTLAM                       ( Local: k=1 )
  2GETLAM 1GETLAM #1- #*
  4GETLAM #+                           ( l=n*[k-1]+p )
```

```
  PULLREALEL %0=                          ( a[l]=0? )
  IT :: # A03 DO#EXIT ;                   ( V => Sistema homogéneo )
  2GETLAM 4GETLAM #<>                     ( k<>p? )
  IT :: 4GETLAM 1GETLAM 2GETLAM infi ;    ( V => intercambiar filas )
  ABND
;
```



Observe que un vez ubicado el mayor valor absoluto se pregunta si el mismo es cero, pues si es así se trata de un sistema homogéneo y como tal no puede ser resuelto con los métodos que estudiamos en este capítulo.

Al igual que el módulo anterior, este módulo puede ser probado modificando *"prue"*, así por ejemplo la siguiente modificación prueba *"pivp"* cuando el número pivote es 2:

```
xNAME prue ( módulo de prueba )
::
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %24 %15 %16 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  TWO FOUR SIX pivp
;
```

Al hacer correr *"prue"* se obtiene la matriz con las filas 2 y 3 intercambiadas, lo que es correcto, pues con ese cambio, el mayor valor absoluto de la columna 2 queda en la posición pivote.

5.2.2.2. Pivotaje total

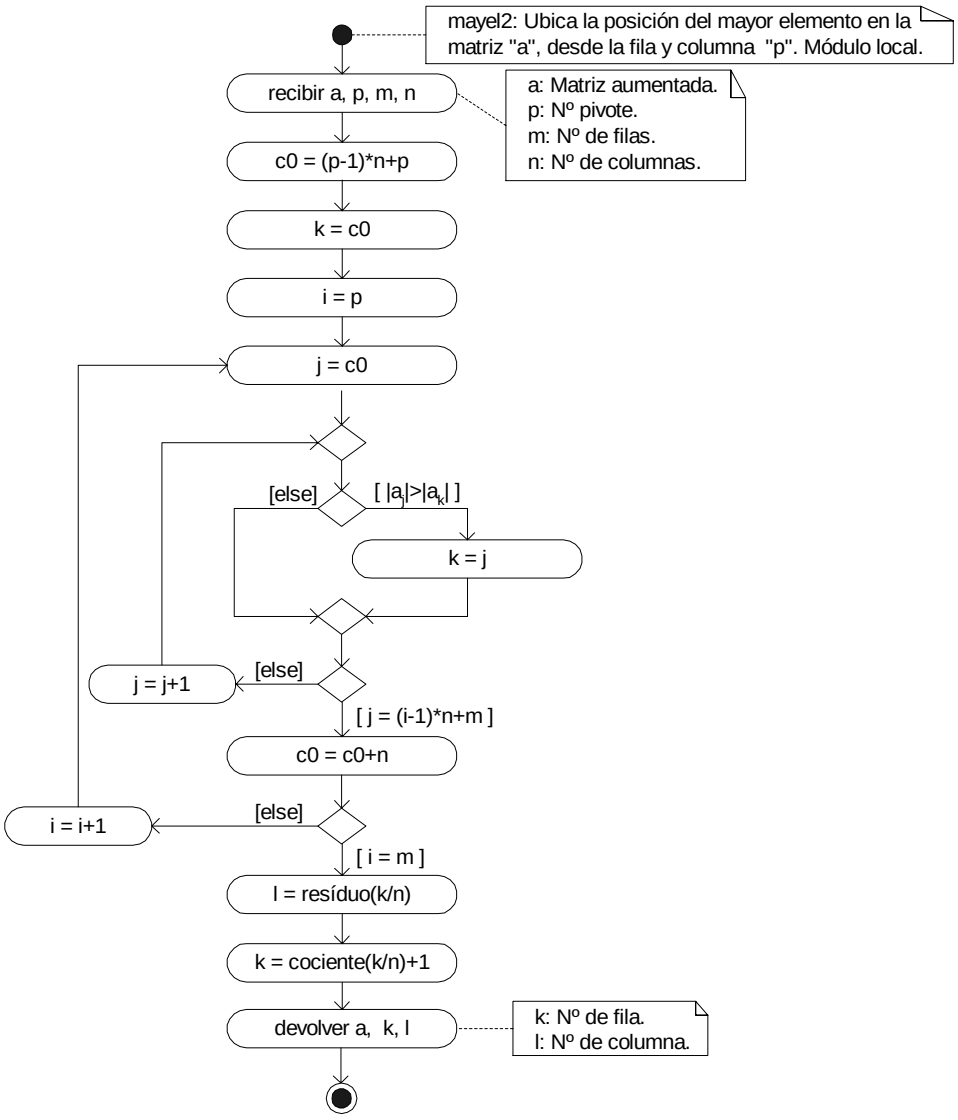
Para realizar el *pivotaje total* se debe ubicar la posición del elemento con el mayor valor absoluto buscando en las filas y columnas no reducidas. Una vez ubicada la posición del elemento con el mayor valor absoluto se efectúan intercambios de filas y columnas, según sea necesario, para llevar el mayor valor absoluto a la posición pivote.

En el pivotaje total, se debe hacer un seguimiento de las columnas intercambiadas, pues un intercambio de columna implica también un intercambio en

la posición de los resultados, así por ejemplo si se intercambia la columna 2 con la columna 4, entonces el resultado de la variable x_2 se encontrará en la fila 4 (y no en la fila 2), mientras que el resultado de la variable x_4 se encontrará en la fila 2 (y no en la fila 4).

Al igual que en el pivotaje parcial, se debe comprobar si el mayor valor absoluto es cero, pues de ser así el sistema es homogéneo y en consecuencia no puede ser resuelto con los métodos que estudiamos en este capítulo.

El algoritmo del módulo que ubica el mayor valor absoluto en las filas y columnas no reducidas es el siguiente:



El código de este módulo es el siguiente:

```
NULLNAME mayel2 ( Búsqueda del mayor elemento en la matriz no reducida )
::                              ( Datos: a= Matriz aumentada)
  BINT0 BINT0                           ( p=5= N° pivote )
  ' NULLLAM FIVE NDUPN DOBIND           ( m=4= N° de filas )
  5GETLAM #1- 3GETLAM #*                ( n=3= N° de columnas )
  5GETLAM #+ 1PUTLAM                    ( Locales: k=2; c0=1 )
  1GETLAM 2PUTLAM                       ( k = c0 )
  4GETLAM #1+                           ( lim. sup.= m )
```

```
  5GETLAM                          ( lim. inf.= p )
  DO                               ( ciclo i )
    INDEX@ #1- 3GETLAM #*
    4GETLAM #+ #1+                 ( lim. sup.= [i-1]*n+m )
    1GETLAM                        ( lim. inf.= c0 )
    DO                             ( ciclo j )
      2GETLAM PULLREALEL %ABS      ( |a[k]| )
      SWAP INDEX@ PULLREALEL %ABS  ( |a[j]| )
      ROT %> IT :: INDEX@ 2PUTLAM ; ( |a[j]|>|a[k]|? => k=j )
    LOOP
    1GETLAM 3GETLAM #+ 1PUTLAM     ( c0 = c0+n )
  LOOP
  2GETLAM 3GETLAM #/ #1+ SWAP      ( k l )
  ABND
;
```

Para probar este módulo podemos modificar el módulo *"prue"* de la siguiente manera:

```
xNAME prue ( módulo de prueba )
::
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %14 %15 %26 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  ONE FOUR SIX mayel2
;
```

Haciendo correr *"prue"* desde el emulador se obtiene la matriz y los números binarios 3 y 4, indicando, como es correcto, que el mayor elemento se encuentra en la fila 3 y la columna 4.

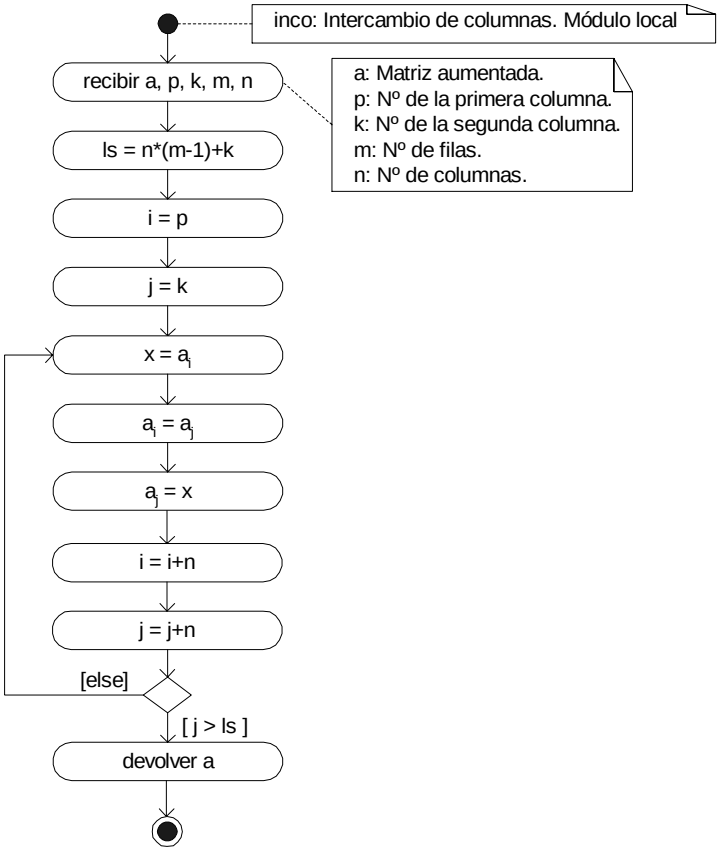
Puesto que en el pivotaje total puede ser necesario intercambiar columnas, se requiere de un módulo que lleve a cabo dicha labor. El algoritmo del módulo que intercambia dos columnas dadas se presenta en la siguiente página y el código elaborado en base al mismo es el siguiente:

```
NULLNAME inco ( intercambio de columnas )
::                               ( Datos: a= Matriz aumentada )
  BINT0 BINT0 BINT0 %0                    ( p=8= N° de la primera columna )
  ' NULLLAM EIGHT NDUPN DOBIND            ( k=7= N° de la segunda columna )
  5GETLAM 6GETLAM #1- #*                  ( m=6= N° de filas )
  7GETLAM #+ 4PUTLAM                      ( n=5= N° de columnas )
  8GETLAM 3PUTLAM                         ( Locales: ls=4; i=3; j=2; x=1 )
  7GETLAM 2PUTLAM                         ( j=k )
  BEGIN
    3GETLAM PULLREALEL 1PUTLAM            ( x=a[i] )
    2GETLAM PULLREALEL 3GETLAM PUTREALEL  ( a[i]=a[j] )
    1GETLAM 2GETLAM PUTREALEL             ( a[j]=x )
    3GETLAM 5GETLAM #+ 3PUTLAM            ( i=i+n )
    2GETLAM 5GETLAM #+ 2PUTLAM            ( j=j+n )
    2GETLAM 4GETLAM #>
  UNTIL
  ABND
;
```

Probamos este código modificando una vez más el módulo *"prue"*:

```
xNAME prue ( módulo de prueba )
::
```

```
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %14 %15 %26 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  TWO FOUR FOUR SIX inco
;
```

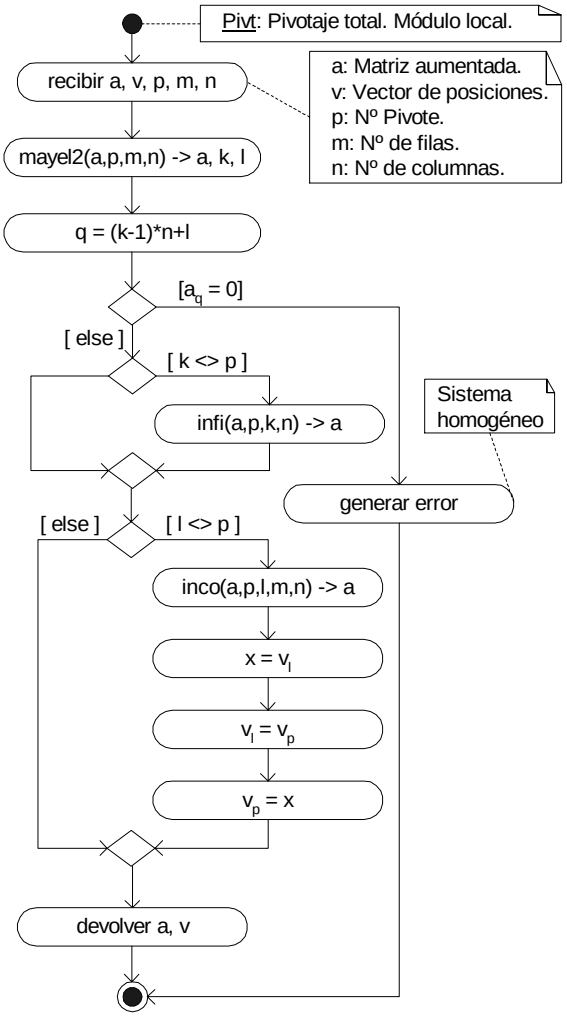


Haciendo correr *"prue"* desde el emulador se obtiene la matriz con las columnas 2 (TWO) y 4 (FOUR) intercambiadas.

Finalmente, con todos los módulos ya elaborados y probados, se puede realizar el algoritmo para el pivotaje total, el mismo que se presenta en el diagrama de actividades de la siguiente página. El código elaborado en base a dicho algoritmo es el siguiente:

```
NULLNAME pivt ( pivotaje total )
::                                  ( Datos: a= Matriz aumentada )
  BINT0 BINT0 %0                           ( v=7= Vector de posiciones )
  ' NULLLAM SEVEN NDUPN DOBIND             ( p=6= N° pivote )
  6GETLAM 5GETLAM 4GETLAM                  ( m=5= N° de filas )
  mayel2 2PUTLAM 3PUTLAM                   ( n=4= N° de columnas )
  3GETLAM #1- 4GETLAM #*                   ( Locales: k=3; l=2; x=1 )
  2GETLAM #+ PULLREALEL %0=                ( a[q]=0? )
  IT :: # A03 DO#EXIT ;                    ( V => Sistema homogéneo )
  3GETLAM 6GETLAM #<>                      ( k<>p? )
  IT :: 6GETLAM 3GETLAM 4GETLAM infi ;     ( V => infi[a,p,k,n] )
  2GETLAM 6GETLAM #<>                      ( l<>p? )
  ITE ::                                   ( V => )
    6GETLAM 2GETLAM 5GETLAM 4GETLAM inco   ( inco[a,p,l,m,n] )
    7GETLAM
```

```
    2GETLAM PULLREALEL 1PUTLAM           ( x=v[l] )
    6GETLAM PULLREALEL 2GETLAM PUTREALEL ( v[l]=v[p] )
    1GETLAM 6GETLAM PUTREALEL            ( v[p]=x )
  ;
  7GETLAM
  ABND
;
```



Para probar este algoritmo modificamos el módulo *"prue"*:

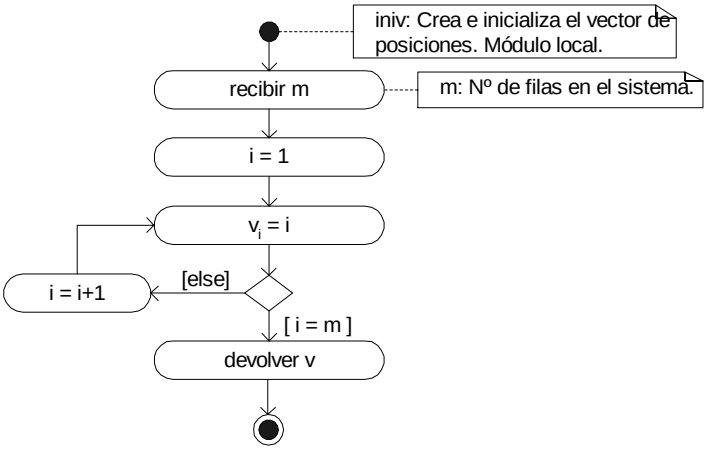
```
xNAME prue ( módulo de prueba )
::
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %14 %15 %26 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  %1 %2 %3 %4 { %4 } FLASHPTR XEQ>ARRY
  ONE FOUR SIX pivt
;
```

Haciendo correr *"prue"* se obtiene la matriz con el mayor valor absoluto en la posición pivote (en la posición 1) y el vector de posiciones [4 3 2 1] que nos muestra que ha ocurrido un intercambio entre las columnas 1 y 4.

5.2.3. Vector de posiciones

Como se ha visto en el anterior acápite, el pivotaje total requiere de un vector de posiciones para hacer un seguimiento de los intercambios de columnas efectuados. Es conveniente contar con un módulo que cree y devuelva un vector con números ordenados ascendentemente representando las posiciones originales de las soluciones.

El algoritmo del módulo que crea e inicializa el vector de posiciones es el siguiente:



Y el código elaborado en base a este algoritmo es:

```
NULLNAME iniv ( Creación e inicialización del vector de posiciones )
:: ( Datos: m=1= N° de filas )
  1LAMBIND
  1GETLAM #1+ ( lim. sup.= m )
  ONE ( lim. inf.= 1 )
  DO  ( ciclo i )
    INDEX@ UNCOERCE
  LOOP
  1GETLAM UNCOERCE ONE {}N
  FLASHPTR XEQ>ARRY ( vector de posiciones )
  ABND
;
```

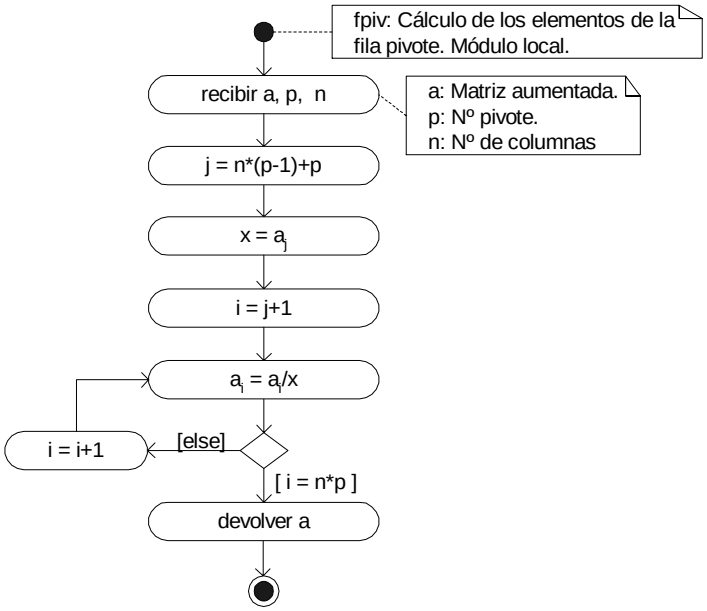
Para probar este módulo se puede modificar el módulo *"prue"* de manera que el vector de posiciones sea generado con *"iniv"* y no escrito directamente:

```
xNAME prue ( módulo de prueba )
::
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %14 %15 %26 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  FOUR iniv
  ONE FOUR SIX pivt
;
```

Como era de esperar, haciendo correr *"prue"* se obtiene el mismo resultado que con la anterior modificación, pues la única diferencia con relación a la misma es que el vector de posiciones es generado con *"iniv"* en lugar de ser escrito directamente.

5.2.4. Reducción de filas

Una vez que se tiene valor correcto en la posición pivote (llevando a cabo el pivotaje total o parcial), se procede a reducir la fila pivote aplicando la ecuación (6), lo que convierte en 1 el elemento pivote. El algoritmo del módulo que lleva a cabo esta labor es el siguiente:



Y el código elaborado en base al mismo es:

```
NULLNAME fpiv ( Cálculo de los elementos de la fila pivote)
::                      ( Datos: a = Matriz aumentada )
  BINT0 %0                      ( p=4= N° pivote )
  ' NULLLAM FOUR NDUPN DOBIND   ( n=3= N° de columnas )
  3GETLAM 4GETLAM #1-           ( Locales: j=2; x=1 )
  #* 4GETLAM #+ 2PUTLAM         ( j = n*[p-1]+p )
  2GETLAM PULLREALEL 1PUTLAM    ( x = a[j] )
  4GETLAM 3GETLAM #* #1+        ( límite superior = n*p )
  2GETLAM #1+                   ( límite inferior = j+1 )
  DO
    INDEX@ PULLREALEL 1GETLAM
    %/ INDEX@ PUTREALEL         ( a[i] = a[i]/x )
  LOOP
  ABND
;
```

Para probar este módulo modificamos el módulo *"prue"*:

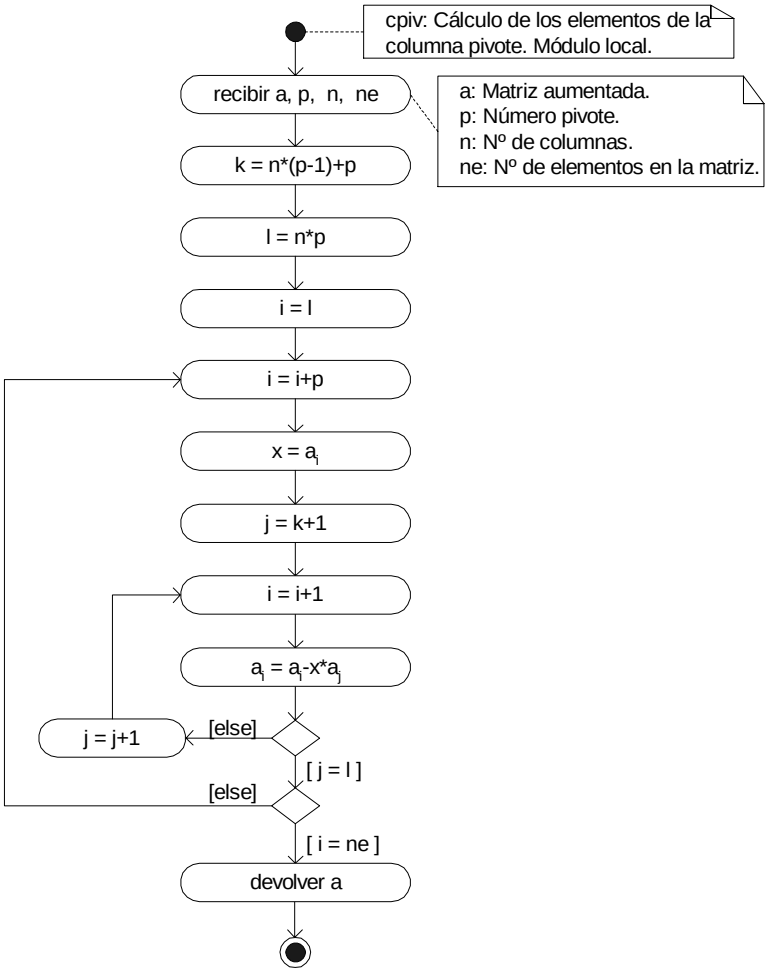
```
xNAME prue ( módulo de prueba )

::
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %14 %15 %26 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  TWO SIX fpiv
;
```

Haciendo correr *"prue"* se obtiene la matriz con los elementos de la segunda fila, que se encuentran después del pivote *(TWO)*, divididos entre 8.

5.2.5. Reducción de columnas

Después de reducir la fila pivote, se tiene que reducir los elementos de la columna pivote a cero aplicando la ecuación (7). El algoritmo del módulo que lleva a cabo esta labor es el siguiente:



El código elaborado en base a este algoritmo es:

```
NULLNAME cpiv ( cálculo de los elementos de la columna pivote )
::                          ( Datos: a = Matriz aumentada )
  BINT0 BINT0 BINT0 %0              ( p=7= N° pivote )
  ' NULLLAM SEVEN NDUPN DOBIND      ( n=6= N° de filas )
  6GETLAM 7GETLAM #1- #*            ( ne=5= N° de elementos )
  7GETLAM #+ 3PUTLAM                ( Locales: i=4 ; k=3; l=2; x=1 )
  6GETLAM 7GETLAM #* 2PUTLAM        ( l = n*p )
  2GETLAM 4PUTLAM                   ( i = l )
  BEGIN
    4GETLAM 7GETLAM #+ 4PUTLAM      ( i = i+p )
    4GETLAM PULLREALEL 1PUTLAM      ( x = a[i] )
    2GETLAM #1+                     ( límite superior = l )
    3GETLAM #1+                     ( límite inferior = k+1 )
    DO
      4GETLAM #1+ 4PUTLAM           ( i = i+1 )
      4GETLAM PULLREALEL            ( a[i] )
      1GETLAM ROT                   ( x )
      INDEX@ PULLREALEL SWAP        ( a[j] )
```

```
      4UNROLL %* %-
      4GETLAM PUTREALEL            ( a[i] = a[i]-x*a[j] )
    LOOP
    4GETLAM 5GETLAM #=             ( i = ne )
  UNTIL
  ABND
;
```

Para probar este código modificamos el módulo *"prue"*:

```
xNAME prue ( módulo de prueba )
::
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %14 %15 %26 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  TWO SIX TWENTYFOUR cpiv
;
```

Haciendo correr *"prue"* se obtiene la matriz con todos los elementos debajo de la fila pivote (la fila dos) y de la columna pivote (la columna 2) reducidos. Como ya se explicó previamente tanto en la reducción de filas como en la reducción de columnas no se calculan ni los ceros ni los unos, pues dichos valores no se emplean en los cálculos posteriores.

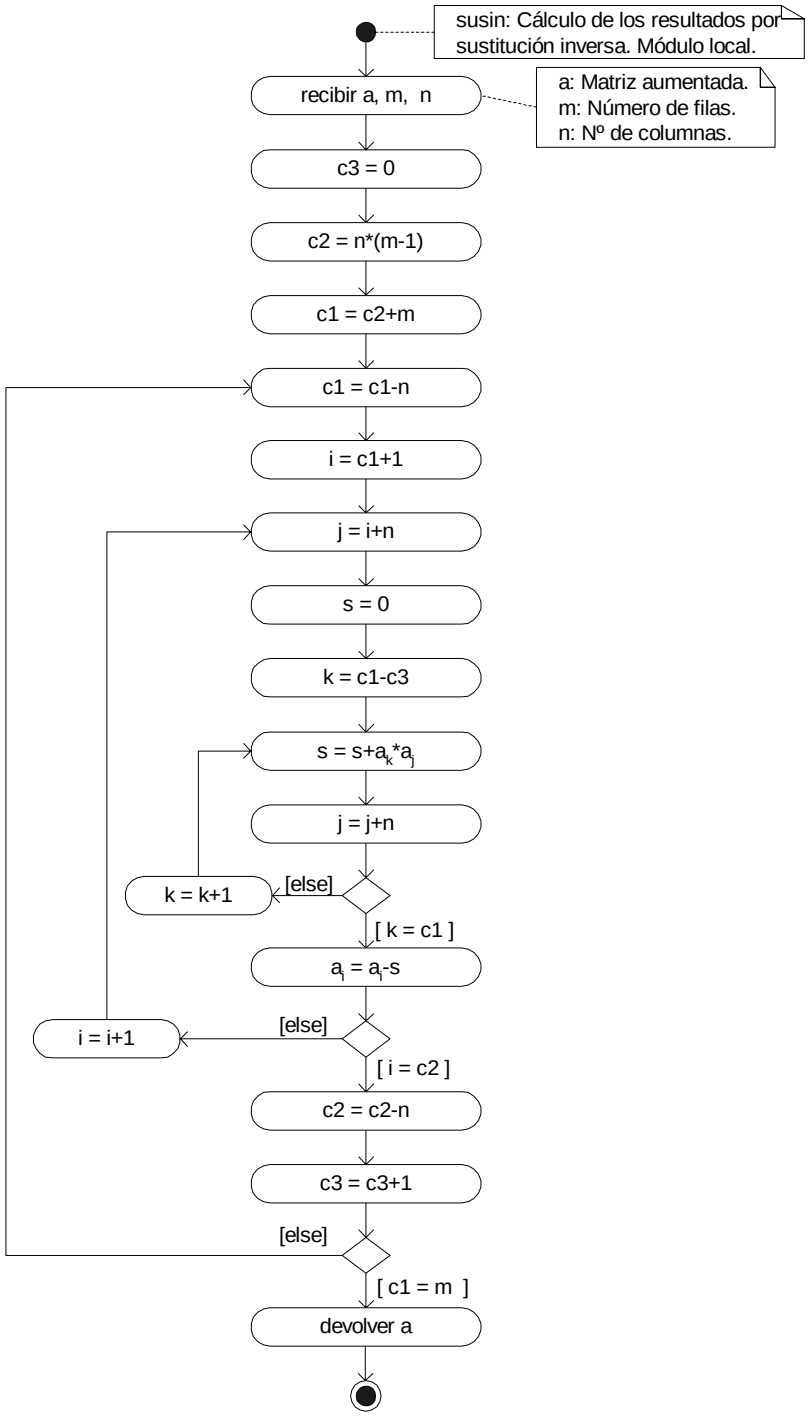
5.2.6. Sustitución inversa

Una vez reducidas las filas y columnas se calculan las soluciones por sustitución inversa (ecuación 8).

El algoritmo del módulo que lleva a cabo esa labor se muestra en el diagrama de actividades de la siguiente página. El código elaborado en base a dicho algoritmo es el siguiente:

```
NULLNAME susin ( cálculo de resultados por sustitución inversa )
::                          ( Datos:  a= Matriz aumentada)
  BINT0 BINT0 BINT0 BINT0 %0       ( m=7= N° de filas )
  ' NULLLAM SEVEN NDUPN DOBIND     ( n=6= N° de columnas )
  6GETLAM 7GETLAM #1- #* 3PUTLAM   ( Locales: j=5; c1=4; c2=3; c3=2; s=1 )
  3GETLAM 7GETLAM #+ 4PUTLAM       ( c1 = c2+m )
  BEGIN
    4GETLAM 6GETLAM #- 4PUTLAM     ( c1 = c1-n )
    3GETLAM #1+                    ( lim. sup. de i = c2 )
    4GETLAM #1+                    ( lim. inf. de i = c1+1 )
    DO                             ( ciclo i )
      INDEX@ 6GETLAM #+ 5PUTLAM    ( j = i+n )
      %0 1PUTLAM                   ( s = 0 )
      4GETLAM #1+                  ( lim. sup. de i = c1 )
      4GETLAM 2GETLAM #-           ( lim. inf de i = c1-c3 )
      DO                           ( ciclo k )
        INDEX@ PULLREALEL SWAP
        5GETLAM PULLREALEL SWAP
        UNROT %*
        1GETLAM %+ 1PUTLAM         ( s = s+a[k]*a[j] )
        5GETLAM 6GETLAM #+ 5PUTLAM ( j = j+n )
      LOOP
      INDEX@ PULLREALEL 1GETLAM %-
      INDEX@ PUTREALEL             ( a[i] = a[i]-s )
    LOOP
```

```
    3GETLAM 6GETLAM #- 3PUTLAM      ( c2 = c2-n )
    2GETLAM #1+ 2PUTLAM             ( c3 = c3+1 )
    4GETLAM 7GETLAM #=              ( c1=m? )
  UNTIL
  ABND
;
```



Para probar este módulo modificamos una vez más el módulo *"prue"*:

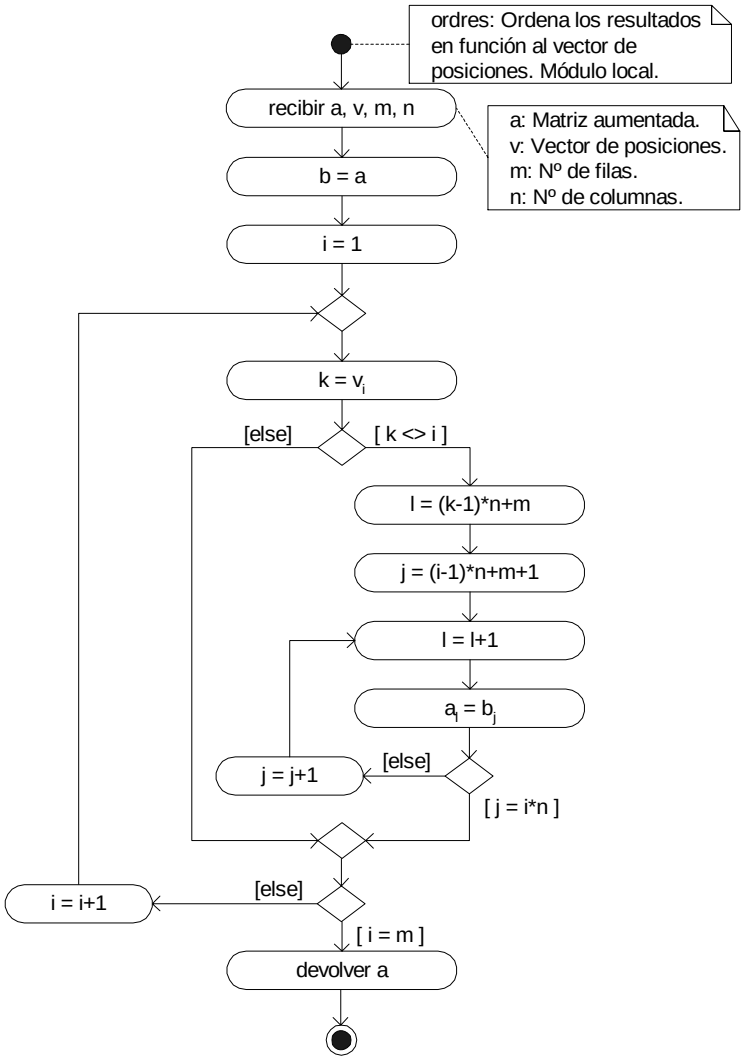
```
xNAME prue ( módulo de prueba )
::
  %1 %2 %3 %4 %5 %6
```

```
  %7 %8 %9 %10 %11 %12
  %13 %14 %15 %26 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  FOUR SIX susin
;
```

Haciendo correr *"prue"* se obtiene la matriz con las dos últimas columnas (las columnas correspondientes a los resultados) calculadas por sustitución inversa.

5.2.7. Ordenación de resultados

Como ya se dijo, cuando se emplea *pivotaje total* los resultados pueden quedar desordenados debido a los intercambios de columnas efectuados. Es necesario entonces ordenar los resultados de acuerdo a la información almacenada en el vector de posiciones. El algoritmo del módulo que lleva a cabo esta tarea es el siguiente:



El código elaborado en base a este algoritmo es:

```
NULLNAME ordres ( Ordena los resultados con el vector de posiciones)
::                                  ( Datos: a= Matriz aumentada )
  4PICK TOTEMPOB BINT0 BINT0                ( v=6= Vector de posiciones )
```

```
  ' NULLLAM SIX NDUPN DOBIND                ( m=5= N° de filas )
  5GETLAM #1+                               ( n=4= N° de columnas )
  ONE                                       ( Locales: b=3; l=2; k=1 )
  DO                                        ( ciclo i )
    6GETLAM INDEX@ PULLREALEL
    COERCE 1PUTLAM DROP                     ( k=v[i] )
    1GETLAM INDEX@ #<>                      ( k<>i? )
    IT ::                                   ( V => )
      1GETLAM #1- 4GETLAM #* 5GETLAM #+
      2PUTLAM                               ( l=[k-1]*n+m )
      INDEX@ 4GETLAM #* #1+                 ( lim. sup.= i*n )
      INDEX@ #1- 4GETLAM #* 5GETLAM #+ #1+  ( lim. inf.= [i-1]*n+m+1 )
      DO                                    ( ciclo j )
        2GETLAM #1+ 2PUTLAM                 ( l=l+1 )
        3GETLAM INDEX@ PULLREALEL SWAP DROP ( b[j] )
        2GETLAM PUTREALEL                   ( a[l]=b[j] )
      LOOP
    ;
  LOOP
  ABND
;
```

Para probar este módulo modificamos una vez más *"prue"*:

```
xNAME prue ( módulo de prueba )
::
  %1 %2 %3 %4 %5 %6
  %7 %8 %9 %10 %11 %12
  %13 %14 %15 %26 %17 %18
  %19 %20 %21 %22 %23 %24
  { %4 %6 } FLASHPTR XEQ>ARRY
  %2 %3 %4 %1 { %4 } FLASHPTR XEQ>ARRY
  FOUR SIX ordres
;
```

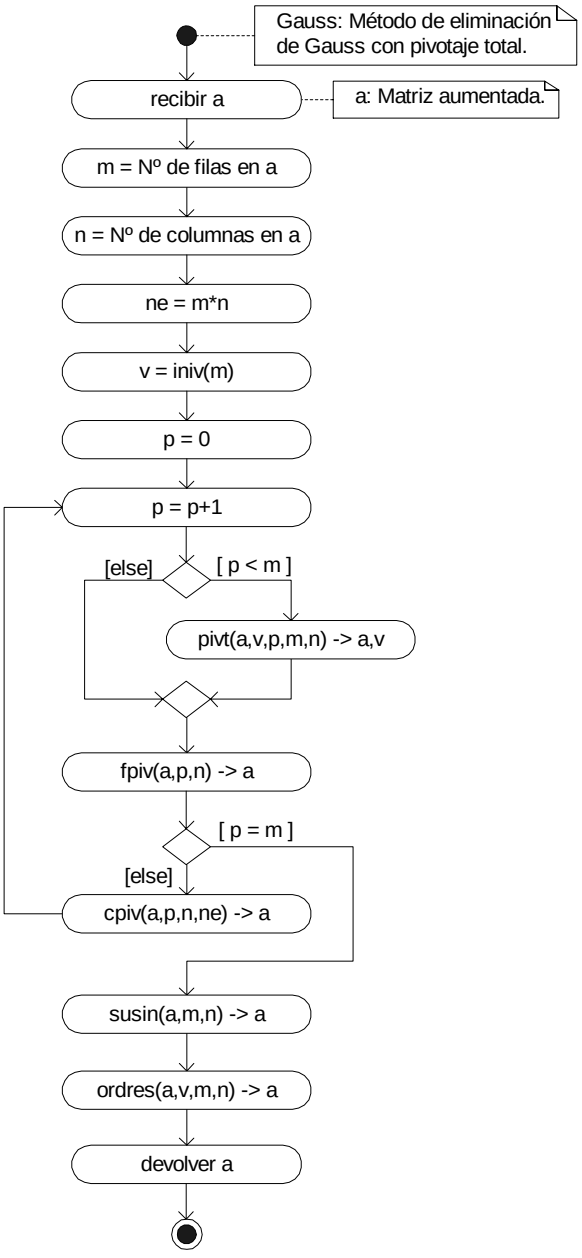
Donde se ha escrito el vector de posiciones con los elementos [2 3 4 1], indicando que en la fila 1 se encuentran las soluciones 2, en la fila 2 las soluciones 3, en la fila 3 las soluciones 4 y en la fila 4 las soluciones 1. Haciendo correr *"prue"* se obtiene la matriz con las dos últimas columnas (las columnas de los resultados) intercambiadas de acuerdo a la información del vector de posiciones.

5.2.8. Eliminación de Gauss con pivotaje total

Finalmente, con todos los módulos elaborados y probados, estamos en condiciones de implementar el método de eliminación de *Gauss* empleando pivotaje total. El algoritmo del método se presenta en el diagrama de actividades de la siguiente página y el código elaborado en base al mismo es el siguiente:

```
xNAME Gauss ( Método de eliminación de Gauss con pivotaje total )
::                              ( Dato: a= Matriz aumentada )
  CK1&Dispatch                        ( Locales: )
  #4                                  ( m=5= N° de filas )
  ::                                  ( n=4= N° de columnas )
    DUP MDIMSDROP 2DUP #*             ( ne=3= N° de elementos)
    3PICK iniv BINT0                  ( v=2= Vector de posiciones )
    ' NULLLAM FIVE NDUPN DOBIND       ( p=1= N° pivote )
    :: BEGIN
      1GETLAM #1+ 1PUTLAM             ( p=p+1 )
      1GETLAM 5GETLAM #<              ( p<m? )
```

```
      IT :: 2GETLAM 1GETLAM 5GETLAM ( V => )
            4GETLAM pivt 2PUTLAM ;  (  a,v=pivt[a,v,p,m,n] )
      1GETLAM 4GETLAM fpiv          ( a=fpiv[a,p,n] )
      1GETLAM 5GETLAM #=            ( p=m? )
      IT :: 2RDROP ;                ( V => Salir )
      1GETLAM 4GETLAM 3GETLAM cpiv  ( a=cpiv[a,p,n,ne] )
    AGAIN ;
    5GETLAM 4GETLAM susin           ( a=susin[a,m,n] )
    2GETLAM 5GETLAM 4GETLAM ordres  ( a=ordres[a,v,m,n] )
    ABND
  ;
;
```



Puesto que este es un módulo externo puede ser probado directamente. Para ello podemos resolver el sistema de ecuaciones lineales (1) con el siguiente programa:

```
« [ [ 2 8 2 14 ]
    [ 1 6 -1 13 ]
    [ 2 -1 2 5 ] ]
  Gauss
  4 COL- SWAP DROP AXL
  { x1 x2 x3 } →TAG
  LIST→ DROP
»
```

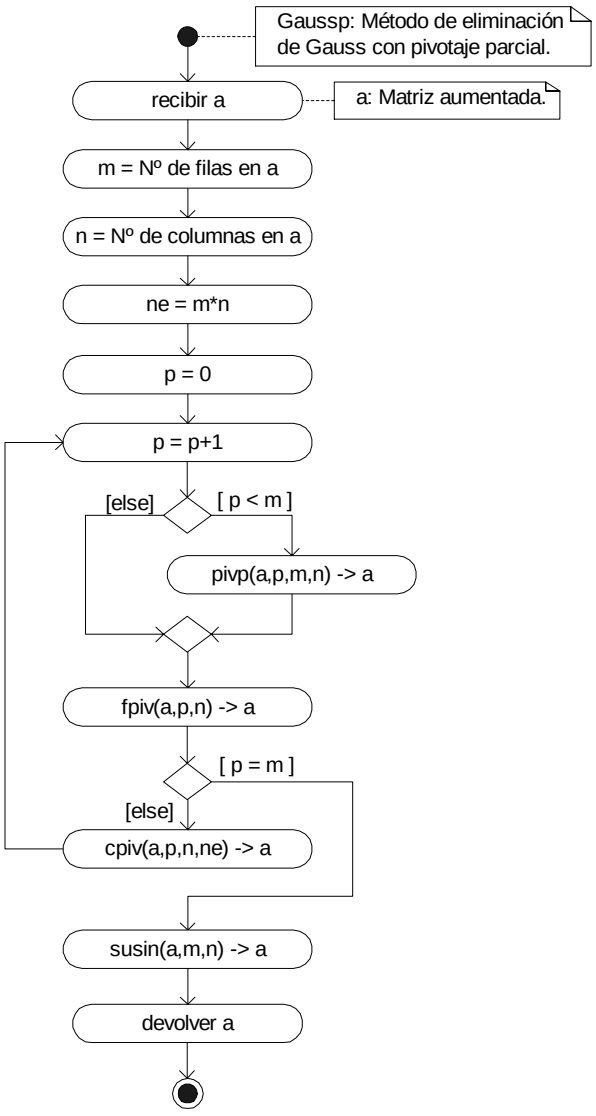
Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 5.
x2: 1.
x3: -2.
s: .7024
```

Que son las soluciones y el tiempo empleado en obtenerlas.

5.2.9. Eliminación de Gauss con pivotaje parcial

Igualmente estamos en condiciones de elaborar el módulo para el método de eliminación de *Gauss* con pivotaje parcial. El algoritmo del módulo es:



El código elaborado en base a este algoritmo es el siguiente:

```
xNAME Gaussp ( Método de eliminación de Gauss con pivotaje parcial )
::                              ( Dato: a= Matriz aumentada )
  CK1&Dispatch                        ( Locales: )
  #4                                  ( m=4= N° de filas )
  ::                                  ( n=3= N° de columnas )
    DUP MDIMSDROP 2DUP #* BINT0       ( ne=2= N° de elementos)
    ' NULLLAM FOUR NDUPN DOBIND       ( p=1= N° pivote )
    :: BEGIN
      1GETLAM #1+ 1PUTLAM             ( p=p+1 )
      1GETLAM 4GETLAM #<              ( p<m? )
      IT :: 1GETLAM 4GETLAM           ( V => )
            3GETLAM pivp ;            ( a=pivp[a,p,m,n] )
      1GETLAM 3GETLAM fpiv            ( a=fpiv[a,p,n] )
      1GETLAM 4GETLAM #=              ( p=m? )
      IT :: 2RDROP ;                  ( V => Salir )
      1GETLAM 3GETLAM 2GETLAM cpiv    ( a=cpiv[a,p,n,ne] )
    AGAIN ;
    4GETLAM 3GETLAM susin             ( a=susin[a,m,n] )
    ABND
  ;
;
```

Para probar este algoritmo volvemos a resolver el sistema (1):

```
« [ [ 2 8 2 14 ]
    [ 1 6 -1 13 ]
    [ 2 -1 2 5 ] ]
  Gaussp
  4 COL- SWAP DROP AXL
  { x1 x2 x3 } →TAG
  LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 5.
x2: 1.
x3: -2.
s: .5245
```

Que son las soluciones y el tiempo empleado en obtenerlas.

5.2.10. Ejemplos

5.2.10.1. Resolución de un sistema con 4 ecuaciones lineales y 4 incógnitas

Como primer ejemplo resolveremos el siguiente sistema de 4 ecuaciones lineales empleando el método de eliminación de *Gauss* con pivotaje total.

$$\begin{aligned} 2x_2 + x_4 &= 0 \\ 2x_1 + 2x_2 + 3x_3 + 2x_4 &= -2 \\ 4x_1 - 3x_2 + x_4 &= -7 \\ 6x_1 + x_2 - 6x_3 - 5x_4 &= 6 \end{aligned} \qquad (9)$$

Para resolver el sistema elaboramos el siguiente programa:

```
« [ [ 0 2 0 1 0 ]
    [ 2 2 3 2 -2 ]
```

```
    [ 4 -3 0 1 -7 ]
    [ 6 1 -6 -5 6 ] ]
  Gauss
  5 COL- SWAP DROP AXL
  { x1 x2 x3 x4 } →TAG
  LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: -.49999999999
x2: 1.
x3: .33333333335
x4: -2.00000000001
s: .9904
```

Que son las soluciones y el tiempo empleado en obtenerlas.

5.2.10.2. Resolución simultánea de 3 sistemas de ecuaciones lineales.

Como segundo ejemplo resolveremos, simultáneamente, los 3 siguientes sistemas de ecuaciones lineales que sólo difieren en las constantes:

$$\begin{aligned} x+2y+3z &= 21 \\ 5x-9y+2z &= 12 \\ 3x+y-4z &= 15 \end{aligned}$$

$$\begin{aligned} u+2v+3w &= 11 \\ 5u-9v+2w &= 2 \\ 3u+v-4w &= 4 \end{aligned} \qquad (10)$$

$$\begin{aligned} r+2s+3t &= 1 \\ 5r-9s+2t &= 2 \\ 3r+s-4t &= 3 \end{aligned}$$

Para resolver simultáneamente sistemas de ecuaciones lineales que sólo difieren en las constantes, se escribe la matriz de los coeficientes aumentada con las columnas de las constantes.

El programa que resuelve el problema es el siguiente:

```
« [ [ 1 2 3 21 11 1 ]
    [ 5 -9 2 12 2 2 ]
    [ 3 1 -4 15 4 3 ] ]
  Gaussp
  →COL DROP
  1 3 START 4 ROLL DROP NEXT
  3 ROLL AXL { x y z } →TAG LIST→ DROP
  5 ROLL AXL { u v w } →TAG LIST→ DROP
  7 ROLL AXL { r s t } →TAG LIST→ DROP
»
```

Haciendo correr el programa se obtiene:

```
x: 7.20329670328
y: 3.21428571428
z: 2.45604395604
u: 2.85714285715
v: 1.71428571429
```

```
w: 1.57142857143
r: .818681318682
s: .214285714286
t: -8.24175824176E-2
s: 1.0692
```

Que son las 9 soluciones y el tiempo empleado en obtenerlas.

5.2.10.3. Cálculo de la inversa de una matriz

Como tercer ejemplo encontraremos la inversa de la matriz de los coeficientes del sistema de ecuaciones lineales (19) empleando el método de eliminación de *Gauss* con pivotaje total:

Para encontrar la inversa de una matriz se manda al método de eliminación de *Gauss* (o a *RREF*) la matriz aumentada con la matriz unidad.

El programa que resuelve el problema es el siguiente:

```
« [ [ 1 2 3 ]
    [ 5 -9 2 ]
    [ 3 1 -4 ] ]
  3 IDN 4 COL+
  Gauss
  →COL DROP
  3 COL→ 4 ROLLD
  3 DROPN
»
```

Haciendo correr el programa con *EVAL* se obtiene:

```
[[ .186813186813 6.04395604394E-2 .170329670329 ]
 [ .142857142857 -7.14285714285E-2 7.14285714282E-2 ]
 [ .175824175824 2.74725274725E-2 -.104395604396 ]]
```

Que es la inversa de la matriz de los coeficientes.

5.2.10.4. Resolución simultánea de 3 sistemas de ecuaciones lineales empleando la matriz inversa

Como cuarto ejemplo volveremos a resolver el sistema de 3 ecuaciones lineales (10), pero empleando ahora la matriz inversa.

Para calcular las soluciones de 2 o más sistemas de ecuaciones lineales que sólo difieren en las constantes se multiplica la matriz inversa por la matriz de las constantes.

El programa que resuelve el problema es el siguiente:

```
« [ [ 1 2 3 ]
    [ 5 -9 2 ]
    [ 3 1 -4 ] ]
  3 IDN 4 COL+
  Gauss
  →COL DROP
  3 COL→ 4 ROLLD
  3 DROPN
  [ [ 21 11 1 ]
    [ 12 2 2 ]
    [ 15 4 3 ] ]
  *
  →COL DROP
```

```
  3 ROLL AXL { x y z } →TAG LIST→ DROP
  5 ROLL AXL { u v w } →TAG LIST→ DROP
  7 ROLL AXL { r s t } →TAG LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x: 7.20329670328
y: 3.21428571428
z: 2.45604395603
u: 2.85714285714
v: 1.71428571428
w: 1.57142857143
r: .818681318679
s: .214285714285
t: -.082417582419
s: 1.6293
```

Que son las 9 soluciones y el tiempo empleado en obtenerlas.

5.2.11. Ejercicios

1. Elabore un programa que resuelva el siguiente sistema de ecuaciones lineales empleando el método de Gauss con pivotaje parcial.

$$\begin{aligned} 10x_1 + x_2 + 2x_3 &= 44 \\ 2x_1 + 10x_2 + x_3 &= 51 \\ x_1 + 2x_2 + 10x_3 &= 61 \end{aligned}$$

2. Elabore un programa que resuelva el siguiente sistema de ecuaciones lineales empleando el método de Gauss con pivotaje total.

$$\begin{aligned} 3x_1 - x_2 + 2x_3 &= 12 \\ x_1 + 2x_2 + 3x_3 &= 11 \\ 2x_1 - 2x_2 - x_3 &= 2 \end{aligned}$$

3. Elabora un programa que resuelva el siguiente sistema de ecuaciones lineales empleando el programa *RREF*.

$$\begin{aligned} 2x_1 - 2x_2 + 5x_3 &= 13 \\ 2x_1 + 3x_2 + 4x_3 &= 20 \\ 3x_1 - x_2 + 3x_3 &= 10 \end{aligned}$$

4. Elabore un programa que resuelva el siguiente sistema de ecuaciones lineales calculado previamente la inversa de la matriz de los coeficientes con el método de eliminación de Gauss con pivotaje total.

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= -1 \\ 4x_1 + 5x_2 + 6x_3 + 7x_4 &= 0 \\ 6x_1 + 10x_2 + 15x_3 + 21x_4 &= 0 \\ 12x_1 + 30x_2 + 60x_3 + 105x_4 &= 0 \end{aligned}$$

5. Repita el ejercicio anterior empleando el método *RREF*.

6. Repita el ejercicio (4) empleando la función *INV*.

7. Elabore un programa que resuelva el siguiente sistema de ecuaciones lineales empleando el método de Gauss con pivotaje total.

$$\begin{aligned} 30x_1 + x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\ 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 24 \\ x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 17 \\ 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 37 \\ 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 23 \\ 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 24 \end{aligned}$$

8. Repita el ejercicio anterior empleando el método de *Gauss* con pivotaje parcial.

9. Repita el ejercicio (7) calculando la inversa con el método de eliminación de *Gauss* con pivotaje parcial.

10. Elabore un programa que resuelva simultáneamente los siguientes sistemas de 3 ecuaciones lineales empleando el método de eliminación de *Gauss* con pivotaje parcial.

$$\begin{aligned} 3x_1 + 2x_2 - x_3 + 2x_4 &= 0 \\ x_1 + 4x_2 + 2x_4 &= 0 \\ 2x_1 + x_2 + 2x_3 - x_4 &= 1 \\ x_1 + x_2 - x_3 + 3x_4 &= 0 \end{aligned}$$

$$\begin{aligned} 3x_1 + 2x_2 - x_3 + 2x_4 &= -2 \\ x_1 + 4x_2 + 2x_4 &= 2 \\ 2x_1 + x_2 + 2x_3 - x_4 &= 3 \\ x_1 + x_2 - x_3 + 3x_4 &= 4 \end{aligned}$$

$$\begin{aligned} 3x_1 + 2x_2 - x_3 + 2x_4 &= 2 \\ x_1 + 4x_2 + 2x_4 &= 2 \\ 2x_1 + x_2 + 2x_3 - x_4 &= 1 \\ x_1 + x_2 - x_3 + 3x_4 &= 0 \end{aligned}$$

11. Repita el ejercicio anterior empleando la inversa calculada con el método de eliminación de *Gauss* con pivotaje total.

12. Repita el ejercicio (10) empleando la inversa calculada con la función *INV*.

13. Elabore un programa que resuelva simultáneamente los siguientes 3 sistemas de ecuaciones lineales empleando el método de eliminación de *Gauss* con pivotaje total.

$$\begin{aligned} 30x_1 + 1x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\ x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 7 \\ 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 7 \\ 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 3 \\ 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 4 \\ 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 4 \end{aligned}$$

$$\begin{aligned} 30y_1 + 1y_2 + 2y_3 + 6y_4 + 5y_5 - 3y_6 &= 14 \\ y_1 + 2y_2 + 33y_3 + 5y_4 - y_5 + 2y_6 &= 1 \\ 3y_1 + 5y_2 + 7y_3 - 45y_4 + 4y_5 + y_6 &= 3 \\ 4y_1 + 3y_2 + 2y_3 + 3y_4 + 40y_5 - 4y_6 &= 2 \\ 5y_1 + 6y_2 - 3y_3 - 4y_4 + 3y_5 + 36y_6 &= 2 \\ 2y_1 + 25y_2 + y_3 - y_4 + 2y_5 - 5y_6 &= 7 \end{aligned}$$

$$\begin{aligned} 30z_1 + 1z_2 + 2z_3 + 6z_4 + 5z_5 - 3z_6 &= 4 \\ z_1 + 2z_2 + 33z_3 + 5z_4 - z_5 + 2z_6 &= 27 \\ 3z_1 + 5z_2 + 7z_3 - 45z_4 + 4z_5 + z_6 &= 27 \\ 4z_1 + 3z_2 + 2z_3 + 3z_4 + 40z_5 - 4z_6 &= 13 \\ 5z_1 + 6z_2 - 3z_3 - 4z_4 + 3z_5 + 36z_6 &= 14 \\ 2z_1 + 25z_2 + z_3 - z_4 + 2z_5 - 5z_6 &= 14 \end{aligned}$$

14. Repita el ejercicio anterior empleando la inversa calculada con el método de eliminación de *Gauss* con pivotaje parcial.

15. Repita el ejercicio (13) empleando la inversa calculada con la función *INV*.

5.3. Método de eliminación de Gauss - Jordán

El método de eliminación de Gauss – Jordán, es muy parecido al método de eliminación de Gauss, sólo que en este caso la matriz de los coeficientes es transformada en una matriz identidad (o unidad). Así si aplicamos el método de eliminación de Gauss-Jordán al sistema de ecuaciones (3):

$$\begin{aligned} a_{1,1}x_1 &+ a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 = c_1 \\ a_{2,1}x_1 &+ a_{2,2}x_2 + a_{2,3}x_3 + a_{2,4}x_4 = c_2 \\ a_{3,1}x_1 &+ a_{3,2}x_2 + a_{3,3}x_3 + a_{3,4}x_4 = c_3 \\ a_{4,1}x_1 &+ a_{4,2}x_2 + a_{4,3}x_3 + a_{4,4}x_4 = c_4 \end{aligned} \qquad (3)$$

Se transforma en:

$$\begin{aligned} x_1 &+ 0 + 0 + 0 = c_1' \\ 0 &+ x_2 + 0 + 0 = c_2' \\ 0 &+ 0 + x_3 + 0 = c_3' \\ 0 &+ 0 + 0 + x_4 = c_4' \end{aligned} \qquad (11)$$

Donde como se puede observar las soluciones del sistema se encuentran en el vector de las constantes. En forma matricial, la aplicación del método de Gauss-jordán a la matriz aumentada logra la siguiente transformación:

$$\begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{vmatrix} \xrightarrow{GAUSS-JORDAN} \begin{vmatrix} 1 & 0 & 0 & 0 & a_{1,5}' \\ 0 & 1 & 0 & 0 & a_{2,5}' \\ 0 & 0 & 1 & 0 & a_{3,5}' \\ 0 & 0 & 0 & 1 & a_{4,5}' \end{vmatrix} \qquad (12)$$

Donde las soluciones se encuentran en la columna (o columnas) aumentadas (las columnas de las constantes).

Las operaciones que se efectúan para llevar la matriz aumentada a la matriz identidad son prácticamente las mismas que en el método de Gauss, excepto que ahora la eliminación de las columnas se la realiza no sólo en la parte inferior de la diagonal, sino también en la parte superior de la misma. En consecuencia para la reducción de las columnas aplicamos la siguiente ecuación en lugar de la ecuación (7).

$$a_{ij} = a_{ij} - a_{ip} * a_{pj} \quad \begin{cases} i = 1 \rightarrow m, \quad i \neq p \\ j = p+1 \rightarrow n; \text{ para cada valor de i} \end{cases} \qquad (13)$$

Para la eliminación de las filas se aplica la misma ecuación que en el método de Gauss (ecuación 6) y en este caso no es necesario la sustitución inversa, pues los resultados se encuentran ya en la columna (o columnas) de las constantes.

Como en el método de Gauss, antes de efectuar la reducción de filas y columnas se debe realizar un pivotaje parcial o total para reducir los errores de redondeo y evitar una división entre cero.

5.3.1. Ejemplo manual

Para comprender mejor el método resolvamos el sistema de ecuaciones (1) con el método de Gauss-Jordán.

La matriz aumentada del sistema es:

$$\begin{vmatrix} 2 & 8 & 2 & 14 \\ 1 & 6 & -1 & 13 \\ 2 & -1 & 2 & 5 \end{vmatrix}$$

Comenzando con *p=1* y aplicando la ecuación (6) para reducir la fila 1 resulta:

$a_{12} = a_{12}/a_{11} = 8/2 = 4$

$a_{13} = a_{13}/a_{11} = 2/2 = 1$

$a_{14} = a_{14}/a_{11} = 14/2 = 7$

Con lo que la matriz queda con:

$$\begin{vmatrix} 1 & 4 & 1 & 7 \\ 1 & 6 & -1 & 13 \\ 2 & -1 & 2 & 5 \end{vmatrix}$$

Aplicando ahora la ecuación (13) para reducir la columna 1:

$a_{22} = a_{22}-a_{21}*a_{12} = 6-1*4 = 2$

$a_{23} = a_{23}-a_{21}*a_{13} = -1-1*1 = -2$

$a_{24} = a_{24}-a_{21}*a_{14} = 13-1*7 = 6$

$a_{32} = a_{32}-a_{31}*a_{12} = -1-2*4 = -9$

$a_{33} = a_{33}-a_{31}*a_{13} = 2-2*1 = 0$

$a_{34} = a_{34}-a_{31}*a_{14} = 5-2*7 = -9$

Con lo que la matriz queda con:

$$\begin{vmatrix} 1 & 4 & 1 & 7 \\ 0 & 2 & -2 & 6 \\ 0 & -9 & 0 & -9 \end{vmatrix}$$

Pasamos ahora a reducir la fila y columna 2 *(p=2)*. Aplicando la ecuación (1) obtenemos:

$a_{23} = a_{23}/a_{22} = -2/2 = -1$

$a_{34} = a_{34}/a_{22} = 6/2 = 3$

Con lo que la matriz queda con los valores:

$$\begin{vmatrix} 1 & 4 & 1 & 7 \\ 0 & 2 & -1 & 3 \\ 0 & -9 & 0 & -9 \end{vmatrix}$$

Aplicamos ahora la ecuación (13) (fila diferente a 2 $i \neq p$):

$a_{13} = a_{13}-a_{12}*a_{23} = 1-(4)*(-1) = 5$

$a_{14} = a_{14}-a_{12}*a_{24} = 7-(4)*(3) = -5$

$a_{33} = a_{33}-a_{32}*a_{23} = 0-(-9)*(-1) = -9$

$a_{34} = a_{34}-a_{32}*a_{24} = -9-(-9)*(3) = 18$

Con lo que la matriz aumentada queda en la forma:

$$\begin{vmatrix} 1 & 0 & 5 & -5 \\ 0 & 1 & -1 & 3 \\ 0 & 0 & -9 & 18 \end{vmatrix}$$

Finalmente procedemos a reducir la última fila *(p=3)*:

$a_{44} = a_{44}/a_{33} = 18/(-9) = -2$

Y la última columna:

$a_{14} = a_{14}-a_{13}*a_{34} = -5-(5)*(-2) = 5$

$a_{24} = a_{24}-a_{23}*a_{34} = 3-(-1)*(-2) = 1$

Con lo que la matriz queda con los valores:

$$\begin{vmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -2 \end{vmatrix}$$

Donde las soluciones del sistema se encuentra en la última columna: $x_1 = 5$; $x_2 = 1$ y $x_3 = -2$.

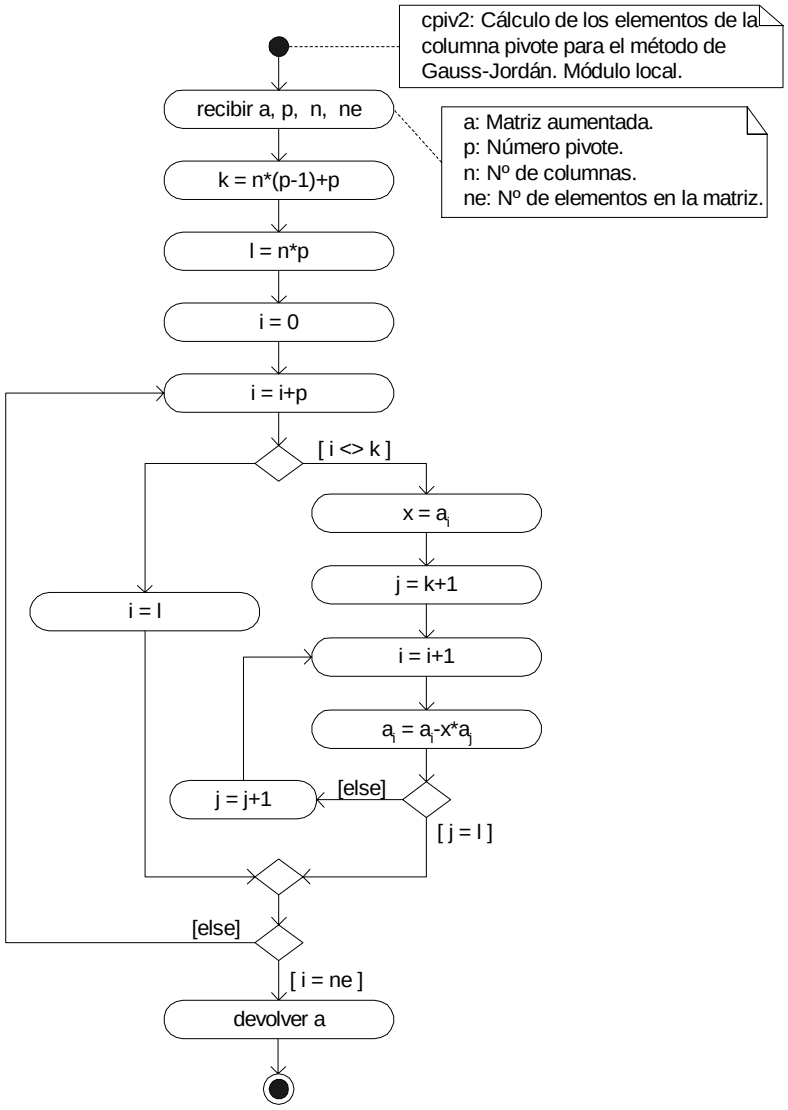
5.3.2. Reducción de columnas

Como ya se mencionó, prácticamente la única diferencia con relación al método de *Gauss*, es que en el método de *Gauss-Jordan* la eliminación de columnas se la realiza tanto por encima como por debajo de la posición pivote (aplicando la ecuación 13). El algoritmo del módulo que lleva a cabo esta tarea se presenta en el diagrama de actividades de la siguiente página, en el mismo, al igual que sucedía en el método de *Gauss* no se calculan los ceros pues no se utilizan en los cálculos posteriores.

El código elaborado en base a dicho algoritmo es el siguiente:

```
NULLNAME cpiv ( Elementos de la columna pivote, método de Gauss-Jordán)
::                          ( Datos: a = Matriz aumentada )
  BINT0 BINT0 BINT0 %0              ( p=7= N° pivote )
  ' NULLLAM SEVEN NDUPN DOBIND      ( n=6= N° de filas )
  6GETLAM 7GETLAM #1- #*            ( ne=5= N° de elementos )
  7GETLAM #+ 3PUTLAM                ( Locales: i=4 ; k=3; l=2; x=1 )
  6GETLAM 7GETLAM #* 2PUTLAM        ( l = n*p )
  BEGIN
    4GETLAM 7GETLAM #+ 4PUTLAM      ( i = i+p )
    4GETLAM 3GETLAM #<>             ( i<>k? )
    ITE
    ::
      4GETLAM PULLREALEL 1PUTLAM    ( x = a[i] )
      2GETLAM #1+                   ( límite superior = l )
      3GETLAM #1+                   ( límite inferior = k+1 )
      DO
        4GETLAM #1+ 4PUTLAM         ( i = i+1 )
        4GETLAM PULLREALEL          ( a[i] )
        1GETLAM ROT                 ( x )
        INDEX@ PULLREALEL SWAP      ( a[j] )
        4UNROLL %* %-
        4GETLAM PUTREALEL           ( a[i] = a[i]-x*a[j] )
      LOOP
    ;
    ::
      2GETLAM 4PUTLAM               ( i = l)
    ;
    4GETLAM 5GETLAM #=              ( i = ne )
  UNTIL
  ABND
;
```

Para probar este módulo elaboramos el siguiente módulo de prueba:



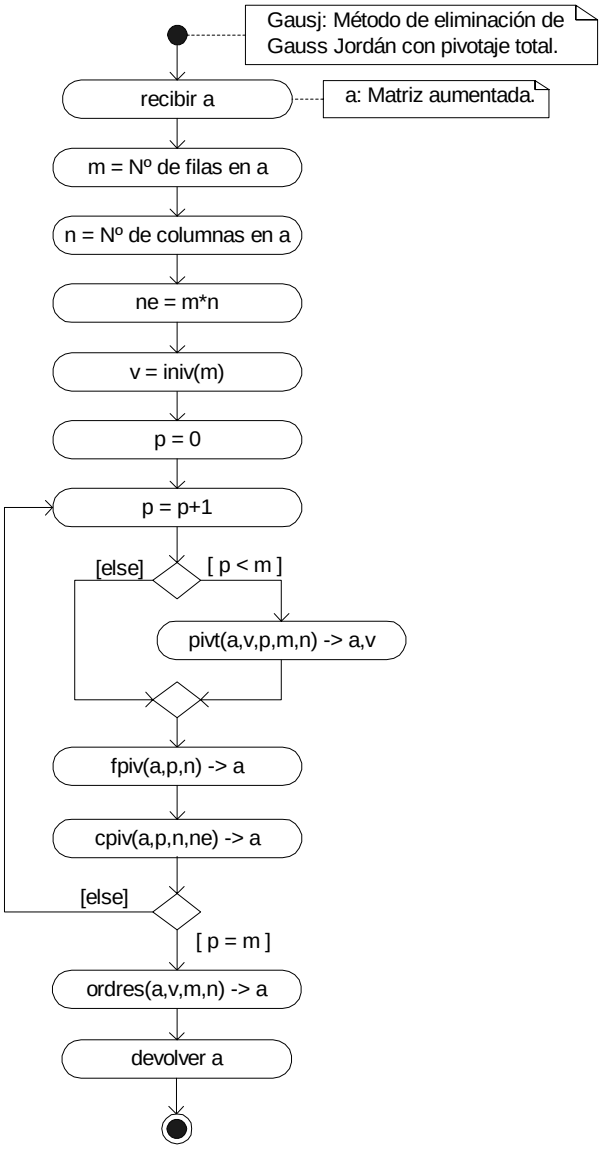

```
xNAME prue
::
  %1 %4 %1 %7
  %0 %2 %-1 %3
  %0 %-9 %0 %-9
  { %3 %4 } FLASHPTR XEQ>ARRY
  TWO FOUR TWELVE cpiv2
;
```

Haciendo correr este módulo (desde el emulador) se obtiene el mismo resultado que en el ejemplo manual cuando se reduce la segunda columna *(TWO)*, excepto que, como ya se dijo, no se calculan los ceros.

5.3.3. Eliminación de Gauss- Jordán con pivotaje total

El algoritmo para resolver un sistema de ecuaciones lineales con pivotaje total, mediante el método de *Gauss-Jordán*, se presenta en el diagrama de actividades de la siguiente página. El código elaborado en base al mismo es el siguiente:

```
xNAME Gausj ( Eliminación de Gauss-Jordán con pivotaje total )
::                              ( Dato: a= Matriz aumentada )
```




```
  CK1&Dispatch                        ( Locales: )
  #4                                  ( m=5= N° de filas )
  ::                                  ( n=4= N° de columnas )
    DUP MDIMSDROP 2DUP #*             ( ne=3= N° de elementos)
    3PICK iniv BINT0                  ( v=2= Vector de posiciones )
    ' NULLLAM FIVE NDUPN DOBIND       ( p=1= N° pivote )
    BEGIN
      1GETLAM #1+ 1PUTLAM             ( p=p+1 )
      1GETLAM 5GETLAM #<              ( p<m? )
      IT :: 2GETLAM 1GETLAM 5GETLAM   ( V => )
            4GETLAM pivt 2PUTLAM ;    (  a,v=pivt[a,v,p,m,n] )
      1GETLAM 4GETLAM fpiv            ( a=fpiv[a,p,n] )
      1GETLAM 4GETLAM 3GETLAM cpiv2   ( a=cpiv2[a,p,n,ne] )
      1GETLAM 5GETLAM #=              ( p=m? )
    UNTIL
    2GETLAM 5GETLAM 4GETLAM ordres    ( a=ordres[a,v,m,n] )
    ABND
  ;
;
```

Podemos probar este módulo resolviendo el sistema de ecuaciones lineales del ejemplo manual:

```
« [ [ 2 8 2 14 ]
    [ 1 6 -1 13 ]
    [ 2 -1 2 5 ] ]
  Gausj
  4 COL- SWAP DROP AXL
  { x1 x2 x3 } →TAG
  LIST→ DROP
»
```

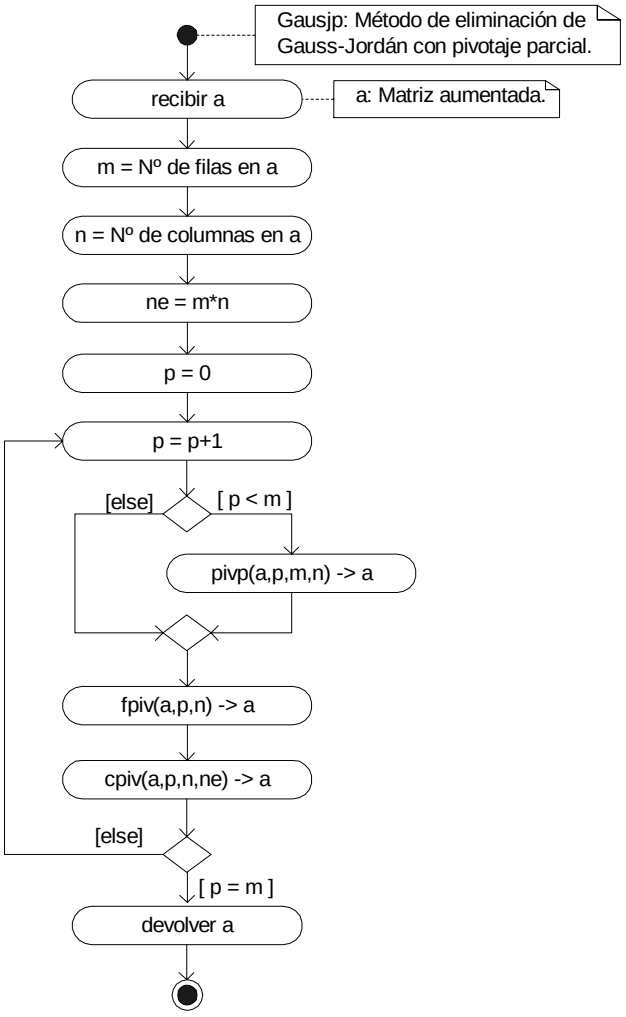
Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 5.
x2: 1.
x3: -2.
s: .7264
```

Que son las soluciones y el tiempo empleado en obtenerlos.

5.3.4. Eliminación de Gauss-Jordán con pivotaje parcial

El algoritmo del método de eliminación de *Gauss-Jordán* con pivotaje parcial se presenta en el siguiente diagrama de actividades:



El código elaborado en base al mismo es el siguiente:

```
xNAME Gausjp ( Método de eliminación de Gauss-Jordán con pivotaje total )
::                              ( Dato: a= Matriz aumentada )
  CK1&Dispatch                        ( Locales: )
  #4                                  ( m=4= N° de filas )
  ::                                  ( n=3= N° de columnas )
    DUP MDIMSDROP 2DUP #* BINT0       ( ne=2= N° de elementos)
    ' NULLLAM FOUR NDUPN DOBIND       ( p=1= N° pivote )
    BEGIN
      1GETLAM #1+ 1PUTLAM             ( p=p+1 )
      1GETLAM 4GETLAM #<              ( p<m? )
      IT :: 1GETLAM 4GETLAM           ( Verdad => )
            3GETLAM pivp ;            ( a=pivp[a,p,m,n] )
      1GETLAM 3GETLAM fpiv            ( a=fpiv[a,p,n] )
      1GETLAM 3GETLAM 2GETLAM cpiv2   ( a=cpiv2[a,p,n,ne] )
      1GETLAM 4GETLAM #=              ( p=m? )
    UNTIL
    ABND
  ;
;
```

Podemos probar este módulo resolviendo el sistema de ecuaciones lineales del ejemplo manual:

```
« [ [ 2 8 2 14 ]
    [ 1 6 -1 13 ]
    [ 2 -1 2 5 ] ]
  Gausjp
  4 COL- SWAP DROP AXL
  { x1 x2 x3 } →TAG
  LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 5.
x2: 1.
x3: -2.
s: .5381
```

Que son las soluciones y el tiempo empleado en obtenerlas.

5.3.5. Ejemplos

5.3.5.1. *Resolución de un sistema con 4 ecuaciones lineales y 4 incógnitas*

Como primer ejemplo volveremos a resolver el ejemplo 5.2.10.1 empleando el método de eliminación de Gauss-Jordán con pivotaje parcial.

$$\begin{aligned} 2x_2 + x_4 &= 0 \\ 2x_1 + 2x_2 + 3x_3 + 2x_4 &= -2 \\ 4x_1 - 3x_2 + x_4 &= -7 \\ 6x_1 + x_2 - 6x_3 - 5x_4 &= 6 \end{aligned} \qquad (9)$$

Para resolver el sistema elaboramos el siguiente programa:

```
« [ [ 0 2 0 1 0 ]
    [ 2 2 3 2 -2 ]
    [ 4 -3 0 1 -7 ]
```

```
    [ 6 1 -6 -5 6 ] ]
  Gausjp
  5 COL- SWAP DROP AXL
  { x1 x2 x3 x4 } →TAG
  LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: -.500000000001
x2: .999999999998
x3: .33333333333
x4: -2.
s: .8938
```

Que son las soluciones y el tiempo empleado en obtenerlas.

5.3.5.2. Resolución simultánea de 3 sistemas de ecuaciones lineales.

Como segundo ejemplo volveremos a resolver los 3 sistemas de 3 ecuaciones lineales del ejemplo 5.2.10.2 con el método de Gauss-Jordán con pivotaje total.

$$\begin{aligned} x+2y+3z &= 21 \\ 5x-9y+2z &= 12 \\ 3x+y-4z &= 15 \end{aligned}$$

$$\begin{aligned} u+2v+3w &= 11 \\ 5u-9v+2w &= 2 \\ 3u+v-4w &= 4 \end{aligned} \qquad (14)$$

$$\begin{aligned} r+2s+3t &= 1 \\ 5r-9s+2t &= 2 \\ 3r+s-4t &= 3 \end{aligned}$$

El programa que resuelve el problema es el siguiente:

```
« [ [ 1 2 3 21 11 1 ]
    [ 5 -9 2 12 2 2 ]
    [ 3 1 -4 15 4 3 ] ]
  Gausj
  →COL DROP
  1 3 START 4 ROLL DROP NEXT
  3 ROLL AXL { x y z } →TAG LIST→ DROP
  5 ROLL AXL { u v w } →TAG LIST→ DROP
  7 ROLL AXL { r s t } →TAG LIST→ DROP
»
```

Haciendo correr el programa se obtiene:

```
x: 7.2032967033
y: 3.2142857143
z: 2.45604395607
u: 2.85714285713
v: 1.71428571428
w: 1.57142857142
r: .818681318678
s: .214285714284
t: -.082417582418
```

```
s: 1.2826
```

Que son las 9 soluciones y el tiempo empleado en obtenerlas.

5.3.5.3. Cálculo de la inversa de una matriz

Como tercer ejemplo volveremos a resolveremos el ejemplo 5.2.10.3 empleando el método de *Gauss-Jordan* con pivotaje parcial.

El programa que resuelve el problema es el siguiente:

```
« [ [ 1 2 3 ]
    [ 5 -9 2 ]
    [ 3 1 -4 ] ]
  3 IDN 4 COL+
  Gausjp
  →COL DROP
  3 COL→ 4 ROLLD
  3 DROPN
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
[[ .186813186813 6.04395604395E-2 .170329670329 ]
 [ .142857142857 -7.14285714286E-2 7.14285714282E-2 ]
 [ .175824175824 2.74725274725E-2 -.104395604396 ]]
s: .7706
```

Que es la inversa de la matriz de los coeficientes y el tiempo empleado en obtenerla.

5.3.5.4. Resolución simultánea de 3 sistemas de ecuaciones lineales empleando la matriz inversa

Como cuarto ejemplo volveremos a resolver el ejercicio 5.2.10.4 con el método de *Gauss-Jordán* con pivotaje total.

El programa que resuelve el problema es el siguiente:

```
« [ [ 1 2 3 ]
    [ 5 -9 2 ]
    [ 3 1 -4 ] ]
  3 IDN 4 COL+
  Gausj
  →COL DROP
  3 COL→ 4 ROLLD
  3 DROPN
  [ [ 21 11 1 ]
    [ 12 2 2 ]
    [ 15 4 3 ] ]
  *
  →COL DROP
  3 ROLL AXL { x y z } →TAG LIST→ DROP
  5 ROLL AXL { u v w } →TAG LIST→ DROP
  7 ROLL AXL { r s t } →TAG LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x: 7.20329670328
y: 3.21428571428
z: 2.45604395603
u: 2.85714285714
```

```
v: 1.71428571428
w: 1.57142857143
r: .818681318679
s: .214285714286
t: -.082417582419
s: 1.6281
```

Que son las 9 soluciones y el tiempo empleado en obtenerlas.

5.3.6. Ejercicios

Resuelva los ejercicios 1, 2, 3, 4, 7, 8, 9, 10, 11, 13 y 14 empleando el método de eliminación de *Gauss-Jordán.*

5.4. Método de Cholesky

El metodo de Cholesky, al igual que en el método de eliminación de Gauss, transforma el sistema de ecuaciones lineales en un sistema triangular superior. Por consiguiente aplicando el método al sistema de ecuaciones (3):

$$\begin{array}{rcrcrcrcl} a_{1,1}x_1 & + & a_{1,2}x_2 & + & a_{1,3}x_3 & + & a_{1,4}x_4 & = & c_1 \\ a_{2,1}x_1 & + & a_{2,2}x_2 & + & a_{2,3}x_3 & + & a_{2,4}x_4 & = & c_2 \\ a_{3,1}x_1 & + & a_{3,2}x_2 & + & a_{3,3}x_3 & + & a_{3,4}x_4 & = & c_3 \\ a_{4,1}x_1 & + & a_{4,2}x_2 & + & a_{4,3}x_3 & + & a_{4,4}x_4 & = & c_4 \end{array} \qquad (3)$$

Se obtiene:

$$\begin{array}{rcrcrcrcl} x_1 & + & a'_{1,2}x_2 & + & a'_{1,3}x_3 & + & a'_{1,4}x_4 & = & c'_1 \\ 0 & + & x_2 & + & a'_{2,3}x_3 & + & a'_{2,4}x_4 & = & c'_2 \\ 0 & + & 0 & + & x_3 & + & a'_{3,4}x_4 & = & c'_3 \\ 0 & + & 0 & + & 0 & + & x_4 & = & c'_4 \end{array} \qquad (4)$$

Que es el mismo resultado que se obtiene con el método de *Gauss*.

En forma matricial, el método de *Cholesky* logra el siguiente cambio:

$$\begin{vmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{vmatrix} \xrightarrow{CHOLESKY} \begin{vmatrix} 1 & a'_{1,2} & a'_{1,3} & a'_{1,4} & a'_{1,5} \\ 0 & 1 & a'_{2,3} & a'_{2,4} & a'_{2,5} \\ 0 & 0 & 1 & a'_{3,4} & a'_{3,5} \\ 0 & 0 & 0 & 1 & a'_{4,5} \end{vmatrix} \qquad (15)$$

Que como se ve es exactamente lo mismo que hace el método de Gauss. En consecuencia en el método de Cholesky, los resultados también se calculan por sustitución inversa.

Sin embargo, la forma en que el sistema de ecuaciones es llevado a un sistema triangular superior es muy diferente en ambos métodos. En el método de Cholesky, el sistema triangular superior se calcula asumiendo que existen dos matrices, una triangular inferior *(L)* y una triangular superior *(U)* (que es la que matriz que nos interesa calcular). Estas matrices tienen la propiedad de que al multiplicarse reproducen la matriz aumentada original, es decir, para el anterior sistema se cumple que:

$$\begin{vmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{4,4} \end{vmatrix} * \begin{vmatrix} 1 & u_{12} & u_{13} & u_{14} & u_{15} \\ 0 & 1 & u_{23} & u_{24} & u_{25} \\ 0 & 0 & 1 & u_{34} & u_{35} \\ 0 & 0 & 0 & 1 & u_{4,5} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{4,4} & a_{4,5} \end{vmatrix} \qquad (16)$$

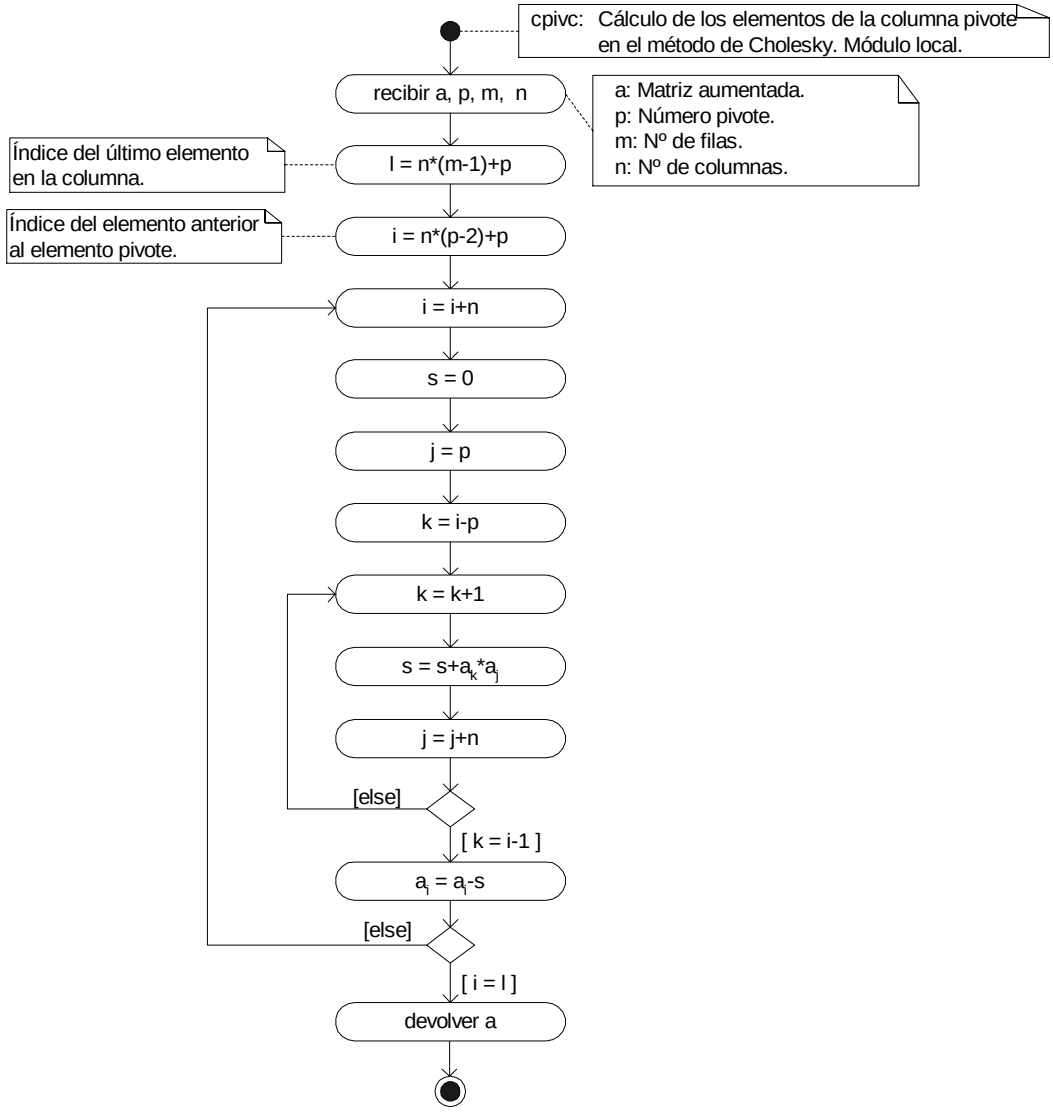
Las relaciones entre las matrices *U*, *L* y la matriz original *A*, pueden deducirse fácilmente llevando a cabo la multiplicación e igualando los elementos. Como no es necesario almacenar los ceros de la matriz *L*, ni los unos de la matriz *U* y dado que los elementos de la matriz *A*, sólo se emplean una vez, todos los valores pueden ser almacenados directamente en la matriz aumentada *A*, evitando así el emplear dos matrices adicionales:

$$a_{ip} = a_{ip} - \sum_{k=1}^{p-1} a_{ik} * a_{kp} \qquad \{i = p \rightarrow m \tag{17}$$

$$a_{pj} = \frac{a_{pj} - \sum_{k=1}^{p-1} a_{pk} * a_{kj}}{a_{pp}} \qquad \{j = p+1 \rightarrow n \tag{18}$$

5.4.1. Cálculo de los elementos de la columna pivote

Como se ha visto, en el método de *Cholesky* se calculan primero los elementos de la columna pivote aplicando la ecuación (19). El algoritmo del módulo que automatiza el cálculo de esta ecuación es:



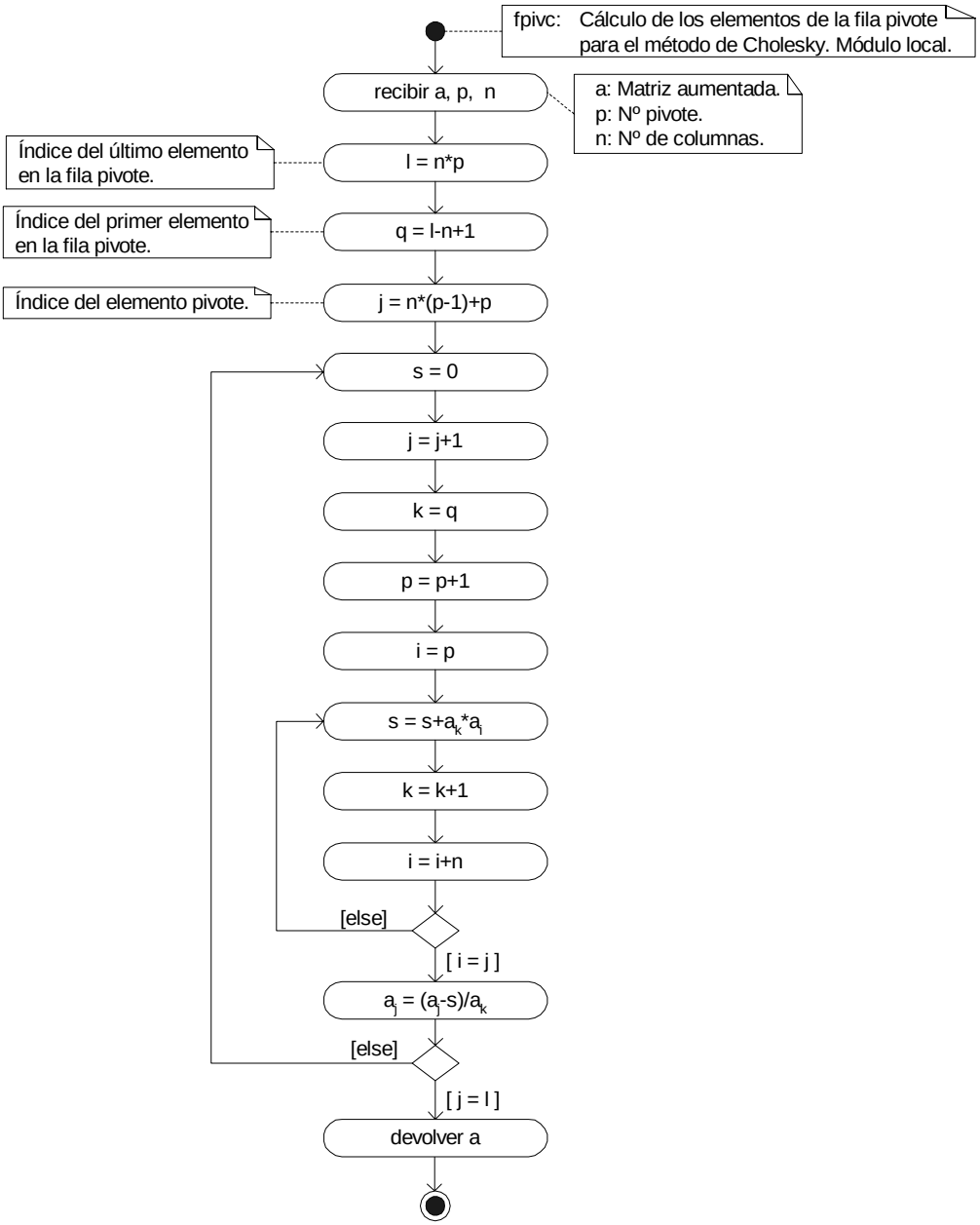
Y el código elaborado en base al mismo es:

```
NULLNAME cpivc ( Cálculo de la columna pivote, método de Cholesky )
::                          ( Datos: a = Matriz aumentada )
  BINT0 BINT0 BINT0 BINT0 %0       ( p=8= N° pivote )
  ' NULLLAM EIGHT NDUPN DOBIND     ( m=7= N° de filas )
  6GETLAM 7GETLAM #1- #*           ( n=6= N° de columnas )
  8GETLAM #+ 5PUTLAM               ( Locales: l=5; i=4; j=3; k=2; s=1 )
  6GETLAM 8GETLAM #2- #*
  8GETLAM #+ 4PUTLAM               ( i = n*[p -2]+p )
  BEGIN
    %0 1PUTLAM                     ( s = 0 )
    4GETLAM 6GETLAM #+ 4PUTLAM     ( i = i+n )
    8GETLAM 3PUTLAM                ( j = p )
    4GETLAM 8GETLAM #- 2PUTLAM     ( k = i-p )
    BEGIN
      2GETLAM #1+ 2PUTLAM          ( k = k+1 )
      2GETLAM PULLREALEL SWAP      ( a[k] )
      3GETLAM PULLREALEL ROT       ( a[j] )
      %* 1GETLAM %+ 1PUTLAM        ( s= s+a[k]*a[j] )
      3GETLAM 6GETLAM #+ 3PUTLAM   ( j=j+n )
      2GETLAM 4GETLAM #1- #=       ( k=i-1? )
    UNTIL
    4GETLAM PULLREALEL 1GETLAM %-
    4GETLAM PUTREALEL              ( a[i]=a[i]-s )
    4GETLAM 5GETLAM #=             ( i=l? )
  UNTIL
  ABND
;
```

5.4.2. Cálculo de los elementos de la fila pivote

El algoritmo del módulo que automatiza el cálculo de la ecuación (20) se presenta en la siguiente página y el código respectivo es:

```
NULLNAME fpivc ( Cálculo de la fila pivote, método de Cholesky )
::                          ( Datos: a = Matriz aumentada )
  BINT0 BINT0 BINT0 BINT0 BINT0 %0 ( p=8= N° pivote )
  ' NULLLAM EIGHT NDUPN DOBIND     ( n=7= N° de filas )
  7GETLAM 8GETLAM #* 5PUTLAM       ( Locales: q=6; l=5; i=4; )
  5GETLAM 7GETLAM #- #1+           (          j=3; k=2; s=1 )
  6PUTLAM                          ( q = l-p+1 )
  7GETLAM 8GETLAM #1- #*
  8GETLAM #+ 3PUTLAM               ( i = n*[p -2]+p )
  BEGIN
    %0 1PUTLAM                     ( s = 0 )
    3GETLAM #1+ 3PUTLAM            ( j = j+1 )
    6GETLAM 2PUTLAM                ( k = q )
    8GETLAM #1+ 8PUTLAM            ( p = p+1 )
    8GETLAM 4PUTLAM                ( i = j-n )
    BEGIN
      2GETLAM PULLREALEL SWAP      ( a[k] )
      4GETLAM PULLREALEL ROT       ( a[i] )
      %* 1GETLAM %+ 1PUTLAM        ( s= s+a[k]*a[i] )
      2GETLAM #1+ 2PUTLAM          ( k = k+1 )
      4GETLAM 7GETLAM #+ 4PUTLAM   ( i = i+n )
      4GETLAM 3GETLAM #=           ( i = j? )
    UNTIL
    3GETLAM PULLREALEL 1GETLAM %-
```

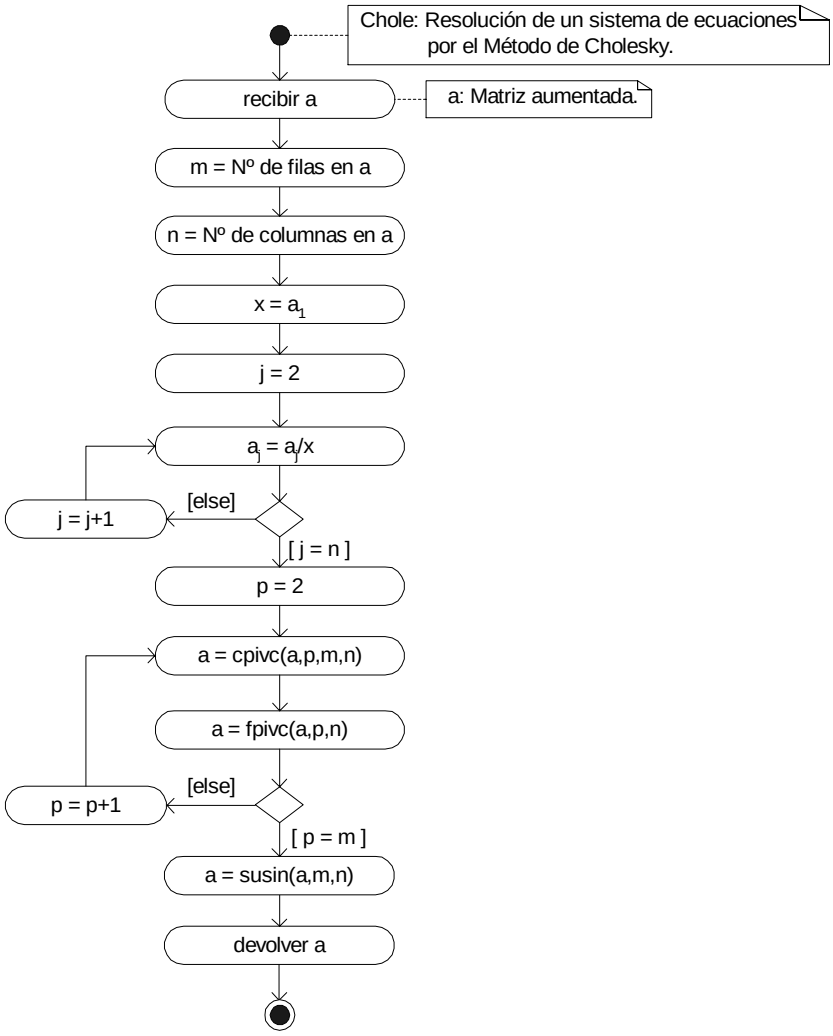



```
    SWAP 2GETLAM PULLREALEL SWAP
    3UNROLL %/ 3GETLAM PUTREALEL    ( a[j]=[a[j]-s]/a[q+p-1] )
    3GETLAM 5GETLAM #=              ( j=l? )
  UNTIL
  ABND
;
```

5.4.3. Módulo principal para el método de Cholesky

El algoritmo de este módulo se presenta en la siguiente página y el código respectivo es:

```
xNAME Chole ( Resolución de ecuaciones lineales, método de Cholesky )
::                              ( Dato: a= Matriz aumentada )
  CK1&Dispatch                        ( Locales: )
  #4                                  ( m=3= N° de filas )
```




```
  ::                                ( n=2= N° de columnas )
    DUP MDIMSDROP %0                ( x=1= Variable temporal)
    ' NULLLAM THREE NDUPN DOBIND
    ONE PULLREALEL 1PUTLAM          ( x=a[1])
    2GETLAM #1+ TWO
    DO                              ( desde 2 hasta n )
      INDEX@ PULLREALEL 1GETLAM
      %/ INDEX@ PUTREALEL           ( a[j]=a[j]/x )
    LOOP
    3GETLAM #1+ TWO
    DO                              ( desde 2 hasta m )
      INDEX@ 3GETLAM 2GETLAM cpivc  ( a=cpivc[a,p,m,n] )
      INDEX@ 2GETLAM fpivc          ( a=fpivc[a,p,n] )
    LOOP
    3GETLAM 2GETLAM susin           ( a=susin[a,m,n] )
    ABND
  ;
;
```

Podemos probar este módulo resolviendo el sistema de ecuaciones lineales del ejemplo manual:

```
« [ [ 2 8 2 14 ]
    [ 1 6 -1 13 ]
```

```
    [ 2 -1 2 5 ] ]
  Chole 4 COL- SWAP DROP AXL
  { x1 x2 x3 } →TAG LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 5.
x2: 1.
x3: -2.
s: .4302
```

Que son las soluciones y el tiempo empleado en obtenerlas. Los ejemplos resueltos con los otros métodos son válidos también para este método.

5.4.4. Ejercicios

Resuelva los ejercicios 1, 2, 3, 4, 7, 8, 9, 10, 11, 13 y 14 empleando el método de *Cholesky.*

5.5. Método de Jacobi

Cuando el número de ecuaciones en un sistema es elevado (con cientos o miles de ecuaciones) los métodos de eliminación estudiados hasta ahora no son de utilidad práctica. Esto es así porque en dichos métodos los resultados intermedios obtenidos se emplean para calcular otros resultados y estos para otros y así sucesivamente hasta a calcular los resultados finales. En este proceso los errores de redondeo se van acumulando e incrementando de manera que cuando el número de ecuaciones es muy grande los errores acumulados son también muy grandes, tanto que los resultados obtenidos suelen ser del todo erróneos.

Uno de los métodos iterativos más sencillos es el de ***Jacobi*** (que es equivalente al método de sustitución directa para resolver una ecuación no lineal). En este método se asumen valores iniciales para todas las incógnitas (generalmente estos valores son fijados en cero). Empleando los valores asumidos se calculan nuevos valores para las incógnitas empleando la primera ecuación del sistema para calcular el valor de la primera incógnita, la segunda ecuación para la segunda incógnita y así sucesivamente. Entonces, con los valores calculados, se verifica si se cumplen las igualdades del sistema (es decir si las ecuaciones se igualan a cero), de ser así el proceso concluye y se tienen ya las soluciones del sistema (los valores calculados), caso contrario los valores calculados se convierten en valores asumidos y se repite el proceso.

Para terminar el proceso iterativo, en lugar de verificar si las igualdades se cumplen, se pueden comparar también los valores calculados en dos iteraciones sucesivas, si dichos valores son casi iguales (es decir son iguales en un determinado número de dígitos) el proceso concluye, caso contrario el proceso se repite.

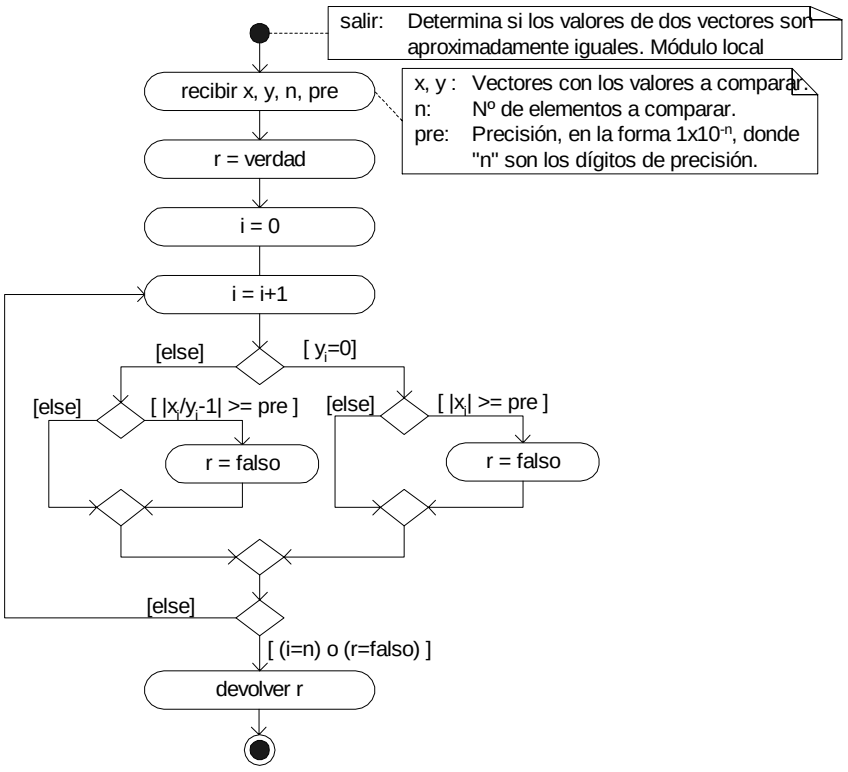
La ecuación general para el cálculo de los nuevos valores de "x", a los cuales denominaremos *"y"* es la siguiente:

$$y_i = \frac{a_{i,n} - \sum_{j=1}^{m} a_{ij} * x_j \quad \{j \neq i}{a_{ii}} \qquad \{\mathrm{i} = 1 \rightarrow m \qquad (19)$$

Donde los coeficientes son los coeficientes de la matriz aumentada.

5.5.1. Comprobación de la convergencia

Puesto que en este método se comparan "n" valores para determinar si se ha logrado o no la convergencia, es conveniente contar con un módulo que realice esta tarea devolviendo verdadero si los valores son aproximadamente iguales y falso en caso contrario. El algoritmo del módulo y el código respectivo son los siguientes:



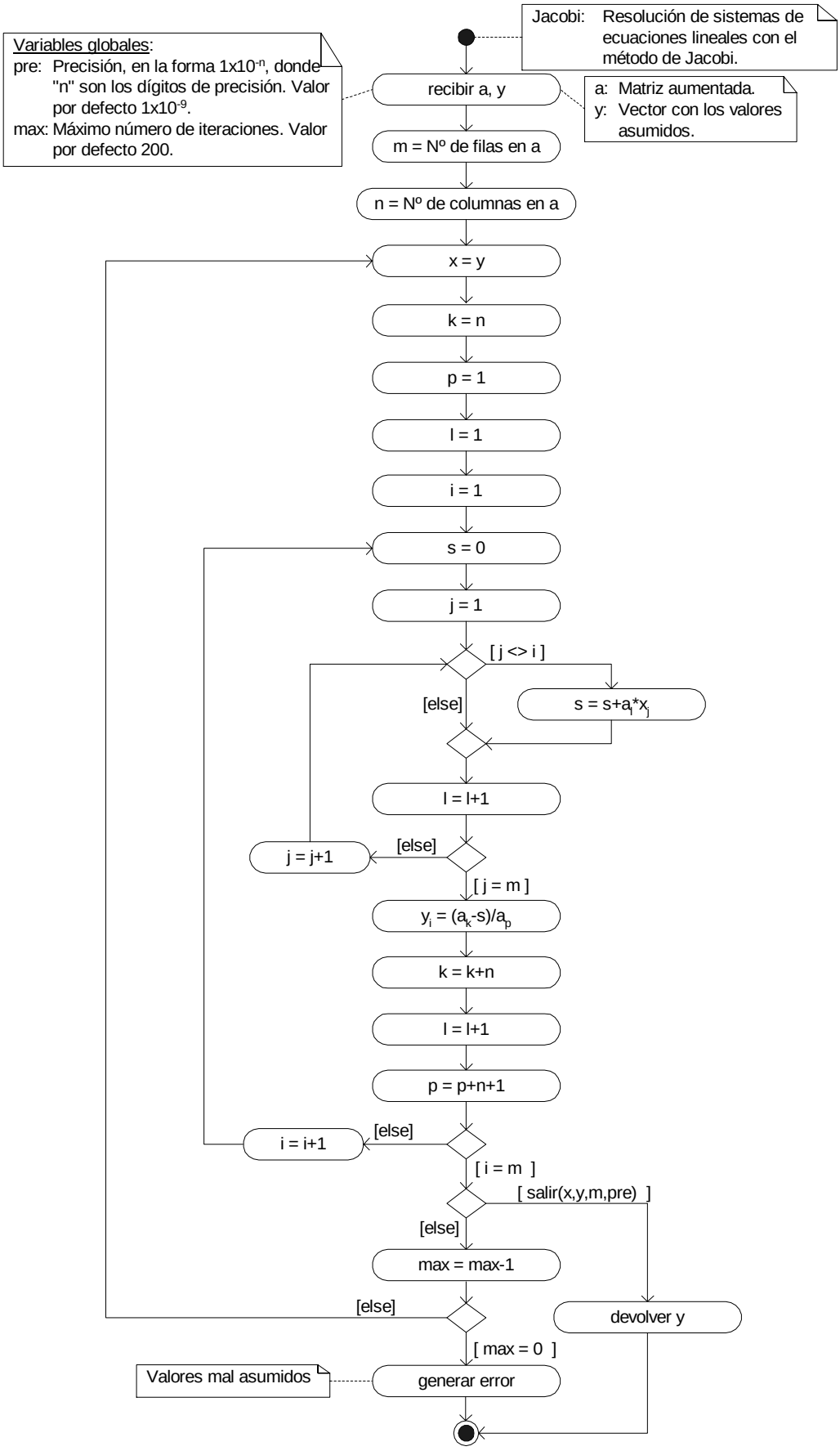

```
NULLNAME salir ( determina si dos vectores son aproximadamente iguales )
::                          ( Datos: x,y: vectores con los datos. )
  TRUE ZERO                       ( n=4= Nº de elementos a comparar)
  ' NULLLAM FOUR NDUPN DOBIND     ( pre=3= precisión )
  BEGIN                            ( Locales: r=2= resultado; i=1= contador)
    1GETLAM #1+ 1PUTLAM            ( i=i+1 )
    SWAP 1GETLAM PULLREALEL ROT    ( x[i] )
    1GETLAM PULLREALEL %0=         ( y[i]=0? )
    ITE
      ::
        UNROT %ABS 3GETLAM %>=     ( |x[i]|>=pre )
        IT :: FALSE 2PUTLAM ;      ( r=falso )
      ;
      ::
        1GETLAM PULLREALEL SWAP    ( y[i] )
        4UNROLL %/ %1- %ABS        ( |x[i]/y[i]-1| )
        3GETLAM %>=                ( ...>=pre? )
        IT
          :: FALSE 2PUTLAM ;       ( r=falso )
      ;
    1GETLAM 4GETLAM #=             ( [i=n]? )
    2GETLAM NOT OR                 ( ... o [r=false]? )
  UNTIL
  2DROP 2GETLAM                    ( devolver r )
```

```
  ABND
;
```

5.5.2. Módulo principal para el método de Jacobi

El algoritmo se presenta en el diagrama de actividades de la siguiente página. El código elaborado en base al mismo es:

```
xNAME Jacobi ( Resolución de ecuaciones lineales. Método de Jacobi )
::               ( Datos: a= Matriz aumentada )
  CK2&Dispatch           ( y=10= Vector con los valores asumidos )
  # 44                   ( Locales: m=9; n=8; k=7; p=6; l=5; )
                         (          x=4; s=3; pre=2; max=1 )
  ::
    OVER MDIMSDROP
    DUP ONE ONE
    6PICK TOTEMPOB %0
    ' ID pre @ ?SKIP % 1E-9      ( si no existe "pre", se emplea 1E-9 )
    ' ID max @ ?SKIP # 200       ( si no existe "max", se emplea 200 )
    ' NULLLAM TEN NDUPN DOBIND
    BEGIN
      4GETLAM 10GETLAM 4PUTLAM 10PUTLAM   ( x=y )
      8 GETLAM 7PUTLAM                    ( k=n )
      ONE 6PUTLAM                         ( p=1 )
      ONE 5PUTLAM                         ( l=1 )
      9GETLAM #1+ ONE
      DO                                  ( desde i=1 hasta m )
        %0 3PUTLAM                        ( s=0 )
        9GETLAM #1+ ONE
        DO                                ( desde j=1 hasta m )
          INDEX@ JINDEX@ #<>              ( i<>j )
          IT
            ::
              5GETLAM PULLREALEL          ( a[l] )
              4GETLAM INDEX@ PULLREALEL   ( x[j] )
              SWAP DROP
              %* 3GETLAM %+ 3PUTLAM       ( s=s+a[i]*x[j] )
            ;
          5GETLAM #1+ 5PUTLAM             ( l=l+1 )
        LOOP
        10GETLAM SWAP 7GETLAM PULLREALEL  ( a[k] )
        3GETLAM %- SWAP                   ( a[k]-s  )
        6GETLAM PULLREALEL SWAP           ( a[p] )
        4UNROLL %/                        ( [a[k]-s]/a[p] )
        INDEX@ PUTREALEL 10PUTLAM         ( y=[a[k]-s]/a[p] )
        7GETLAM 8GETLAM #+ 7PUTLAM        ( k=k+n )
        5GETLAM #1+ 5PUTLAM               ( l=l+1 )
        6GETLAM 8GETLAM #+ #1+ 6PUTLAM    ( p=p+n+1 )
      LOOP
      4GETLAM 10GETLAM 9GETLAM 2GETLAM
      salir                               ( salir[x,y,m,pre]? )
      ITE :: DROP 10GETLAM TRUE ; FALSE
      1GETLAM #1- 1PUTLAM                 ( max=max-1 )
      1GETLAM #0=                         ( max=0? )
      IT :: DROP ABND # A01 DO#EXIT ;     ( # A01 = Bad guesses )
    UNTIL
    ABND
  ;
;
```


Jacobi: Resolución de sistemas de ecuaciones lineales con el método de Jacobi.
Variables globales:
pre: Precisión, en la forma 1x10-n, donde "n" son los dígitos de precisión. Valor por defecto 1x10-9.
max: Máximo número de iteraciones. Valor por defecto 200.
recibir a, y
a: Matriz aumentada.
y: Vector con los valores asumidos.
m = Nº de filas en a
n = Nº de columnas en a
x = y
k = n
p = 1
l = 1
i = 1
s = 0
j = 1
[j <> i]
[else]
s = s+al*xj
l = l+1
[else]
j = j+1
[j = m]
yi = (ak-s)/ap
k = k+n
l = l+1
p = p+n+1
[else]
i = i+1
[i = m]
[salir(x,y,m,pre)]
[else]
max = max-1
[else]
devolver y
[max = 0]
Valores mal asumidos
generar error

Para probar este código resolveremos el sistema de ecuaciones (21), trabajando con la precisión por defecto (9 dígitos) y empleando ceros para los valores asumidos. El programa que resuelve este sistema es el siguiente:

```
« [ [ 10 1 2 44 ]
    [ 2 10 1 51 ]
    [ 1 2 10 61 ] ]
  [ 0 0 0 ] Jacobi 8 RND AXL { x1 x2 x3 } →TAG LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 3.
x2: 4.
x3: 5.
s: 2.3513
```

Que son las tres soluciones y el tiempo empleado en obtenerlas. El programa puede ser modificado para trabajar con una menor precisión, por ejemplo con 5 dígitos:

```
« 1E-5 'pre' STO
  [ [ 10 1 2 44 ]
    [ 2 10 1 51 ]
    [ 1 2 10 61 ] ]
  [ 0 0 0 ] Jacobi 4 RND
  AXL { x1 x2 x3 } →TAG LIST→ DROP 'pre' PURGE
»
```

Haciendo correr el programa se obtiene:

```
x1: 3.
x2: 4.
x3: 5.
s: 1.6383
```

Que redondeados al cuarto decimal son los mismos resultados que en el caso anterior, pero por supuesto el tiempo requerido es menor.

5.5.3. Ejercicios

16. Resuelva el siguiente sistema de ecuaciones lineales por el método de *Jacobi* con 6 dígitos de precisión.

$$\begin{aligned} 5x_1 + 22x_2 + x_3 &= 66 \\ 15x_1 + 3x_2 + 2x_3 &= 52 \\ 2x_1 + 4x_2 + 31x_3 &= 82 \end{aligned}$$

17. Resuelva el siguiente sistema de ecuaciones lineales por el método de *Jacobi* con 4 dígitos de precisión (recuerde ordenar el sistema de manera que quede con la diagonal dominante)

$$\begin{aligned} 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 24 \\ 30x_1 + x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\ 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 23 \\ x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 17 \\ 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 24 \\ 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 37 \end{aligned}$$

5.6. Método de Gauss - Seidel

El método de Gauss – Seidel es esencialmente el método de Jacobi, con la diferencia de que los nuevos valores se calculan empleando siempre los últimos valores calculados y no solamente los asumidos como ocurre en el método de *Jacobi*. Así la primera incógnita se calcula con los valores asumidos, pero la segunda se calcula con el valor calculado para la primera incógnita y los valores asumidos, la tercera se calcula con los valores calculados para la primera y segunda incógnita y los valores asumidos, procediendo así hasta calcular todas las incógnitas.

La ecuación general para calcular los nuevos valores en el método de *Gauss – Seidel* es:

$$y_i = \frac{a_{i,m} - \sum_{j=1}^{i-1} a_{ij} * y_j - \sum_{j=i+1}^{m} a_{ij} * x_j}{a_{ii}} \quad \{i = 1 \rightarrow m \qquad (20)$$

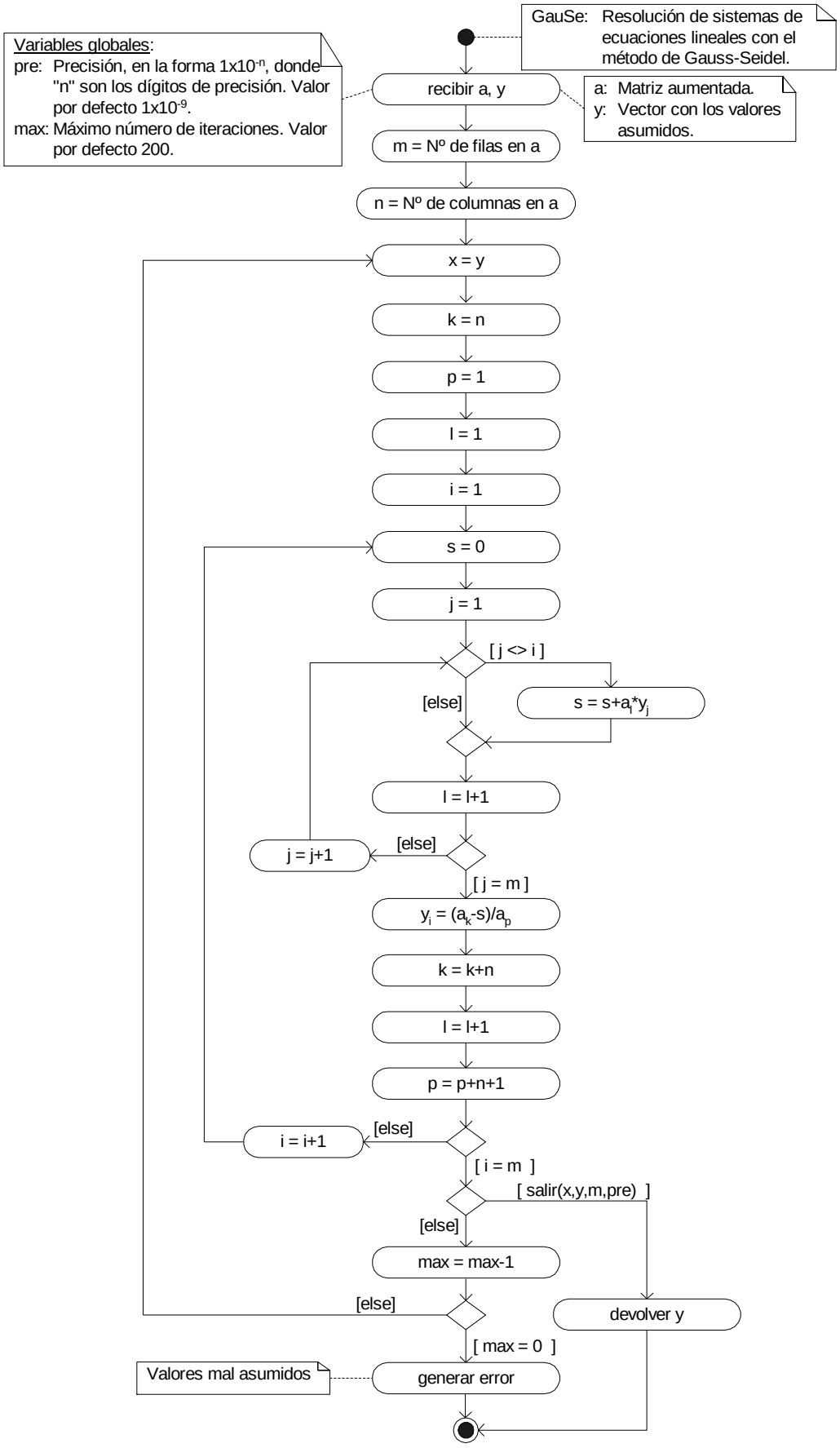
Donde *"a"* son los coeficientes de la matriz aumentada, *"x"* el vector con los valores asumidos, *"y"* el vector con los valores calculados y *"m"* es el número de ecuaciones en el sistema (filas de la matriz aumentada). Si inicialmente los valores asumidos se asignan al vector *"y"*, entonces los nuevos valores pueden ser calculados empleando únicamente el vector *"y"*, si se procede de esa manera la ecuación (23) toma la forma:

$$y_i = \frac{a_{i,n} - \sum_{j=1}^{m} a_{ij} * y_j \quad \{j \neq i}{a_{ii}} \quad \{i = 1 \rightarrow m \qquad (21)$$

5.6.1. Módulo principal del método de *Gauss-Seidel*

El algoritmo se presenta en la siguiente página y el código elaborado en base al mismo es:

```
xNAME GauSe ( Resolución de ecuaciones lineales. Método de Gauss-Seidel )
::                ( Datos: a= Matriz aumentada )
  CK2&Dispatch            ( y=10= Vector con los valores asumidos )
  # 44                    ( Locales: m=9; n=8; k=7; p=6; l=5; )
                          (          x=4; s=3; pre=2; max=1 )
  ::
    OVER MDIMSDROP
    DUP ONE ONE
    6PICK TOTEMPOB %0
    ' ID pre @ ?SKIP % 1E-9      ( si no existe "pre", se emplea 1E-9 )
    ' ID max @ ?SKIP # 200       ( si no existe "max", se emplea 200 )
    ' NULLLAM TEN NDUPN DOBIND
    BEGIN
      10GETLAM 4GETLAM REPLACE DROP       ( x=y )
      8GETLAM 7PUTLAM                     ( k=n )
      ONE 6PUTLAM                         ( p=1 )
      ONE 5PUTLAM                         ( l=1 )
      9GETLAM #1+ ONE
      DO                                  ( desde i=1 hasta m )
        %0 3PUTLAM                        ( s=0 )
        9GETLAM #1+ ONE
        DO                                ( desde j=1 hasta m )
          INDEX@ JINDEX@ #<>              ( i<>j )
          IT
```


GauSe: Resolución de sistemas de ecuaciones lineales con el método de Gauss-Seidel.
Variables globales:
pre: Precisión, en la forma 1x10-n, donde "n" son los dígitos de precisión. Valor por defecto 1x10-9.
max: Máximo número de iteraciones. Valor por defecto 200.
recibir a, y
a: Matriz aumentada.
y: Vector con los valores asumidos.
m = Nº de filas en a
n = Nº de columnas en a
x = y
k = n
p = 1
l = 1
i = 1
s = 0
j = 1
[j <> i]
[else]
s = s+a_l*y_j
l = l+1
[else]
j = j+1
[j = m]
y_i = (a_k-s)/a_p
k = k+n
l = l+1
p = p+n+1
[else]
i = i+1
[i = m]
[salir(x,y,m,pre)]
[else]
max = max-1
[else]
devolver y
[max = 0]
Valores mal asumidos
generar error

```
            ::
              5GETLAM PULLREALEL          ( a[l] )
              10GETLAM INDEX@ PULLREALEL  ( y[j] )
              SWAP DROP
              %* 3GETLAM %+ 3PUTLAM       ( s=s+a[i]*y[j] )
            ;
          5GETLAM #1+ 5PUTLAM             ( l=l+1 )
        LOOP
        10GETLAM SWAP 7GETLAM PULLREALEL  ( a[k] )
        3GETLAM %- SWAP                   ( a[k]-s  )
        6GETLAM PULLREALEL SWAP           ( a[p] )
        4UNROLL %/                        ( [a[k]-s]/a[p] )
        INDEX@ PUTREALEL 10PUTLAM         ( y=[a[k]-s]/a[p] )
        7GETLAM 8GETLAM #+ 7PUTLAM        ( k=k+n )
        5GETLAM #1+ 5PUTLAM               ( l=l+1 )
        6GETLAM 8GETLAM #+ #1+ 6PUTLAM    ( p=p+n+1 )
      LOOP
      4GETLAM 10GETLAM 9GETLAM 2GETLAM
      salir                               ( salir[x,y,m,pre]? )
      ITE :: DROP 10GETLAM TRUE ; FALSE
      1GETLAM #1- 1PUTLAM                 ( max=max-1 )
      1GETLAM #0=                         ( max=0? )
      IT :: DROP ABND # A01 DO#EXIT ;     ( # A01 = Bad guesses )
    UNTIL
    ABND
  ;
;
```

Para probar el código volveremos a resolver el sistema de ecuaciones (21) con 6 dígitos de precisión

```
« 1E-6 'pre' STO
  [ [ 10 1 2 44 ]
    [ 2 10 1 51 ]
    [ 1 2 10 61 ] ]
  [ 0 0 0 ]
  GauSe 5 RND AXL { x1 x2 x3 } →TAG LIST→ DROP
  'pre' PURGE
»
```

Haciendo correr el programa se obtiene:

```
x1: 3.
x2: 4.
x3: 5.
s: .9908
```

Que son las tres soluciones y el tiempo empleado en obtenerlas.

5.6.2. Ejercicios

Resuelva los ejercicios 16 y 17 empleando el método de *Gauss-Seidel*.

5.7. Sistemas tridiagonales

Con frecuencia en la resolución de problemas en el campo de la ingeniería se forman sistemas de ecuaciones lineales que sólo tienen valores en la diagonal principal y a ambos lados de la misma, tales sistemas se conocen con el nombre de *sistemas tridiagonales* y tienen la siguiente forma general:

$$\begin{array}{ccccccccccccccc}
a_{1,2}x_1 & + & a_{1,3}x_2 & + & 0x_3 & + & 0x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{1,n} \\
a_{2,1}x_1 & + & a_{2,2}x_2 & + & a_{2,3}x_3 & + & 0x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{2,n} \\
0x_1 & + & a_{3,2}x_2 & + & a_{3,3}x_3 & + & a_{3,4}x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{3,n} \\
0x_1 & + & 0x_2 & + & a_{4,3}x_3 & + & a_{4,4}x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{4,n} \\
0x_1 & + & 0x_2 & + & 0x_2 & + & a_{5,4}x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{5,n} \\
\ldots & & \ldots & & \ldots & & \ldots & & \ldots & & \ldots & & \ldots & & \ldots \\
0x_1 & + & 0x_2 & + & 0x_3 & + & 0x_4 & + & \ldots & + & a_{m,m-1}x_{m-1} & + & a_{m,m}x_m & = & a_{m,n}
\end{array} \qquad (22)$$

Donde, como en los anteriores sistemas *"m"* es el número de filas (o incógnitas) y *"n"* es el número de columnas. Dado que en estos sistemas la mayoría de los elementos son cero es un desperdicio de memoria y tiempo almacenarlos y realizar operaciones con ellos. Es más eficiente y económico almacenar y trabajar únicamente con los elementos diferentes de cero: los elementos de la diagonal principal, los adyacentes a la misma y la columna de las constantes:

$$\begin{array}{ccccccccccccccc}
a_{1,2}x_1 & + & a_{1,3}x_2 & + & 0x_3 & + & 0x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{1,4} \\
a_{2,1}x_1 & + & a_{2,2}x_2 & + & a_{2,3}x_3 & + & 0x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{2,4} \\
0x_1 & + & a_{3,1}x_2 & + & a_{3,2}x_3 & + & a_{3,3}x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{3,4} \\
0x_1 & + & 0x_2 & + & a_{4,1}x_3 & + & a_{4,2}x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{4,4} \\
0x_1 & + & 0x_2 & + & 0x_2 & + & a_{5,1}x_4 & + & \ldots & + & 0x_{m-1} & + & 0x_m & = & a_{5,4} \\
\ldots & & \ldots & & \ldots & & \ldots & & \ldots & & \ldots & & \ldots & & \ldots \\
0x_1 & + & 0x_2 & + & 0x_3 & + & 0x_4 & + & \ldots & + & a_{m,1}x_{m-1} & + & a_{m,2}x_m & = & a_{m,4}
\end{array} \qquad (23)$$

De esta manera se almacenan *m*4* elementos en lugar de *m*n*, lográndose así un gran ahorro de memoria y tiempo. El sistema involucra en realidad *m+2* variables, pero dos de ellas: x_0 y x_{m+1} son conocidas, razón por la cual no es necesario incluirlas en el sistema. Todos los métodos que hemos estudiado pueden ser adaptados para resolver sistemas tridiagonales. En este acápite emplearemos los métodos de *Gauss* y *Gauss-Seidel*, por ser los que mejor se adaptan a esta forma.

5.7.1. Método de Gauss para sistemas tridiagonales

Las ecuaciones generales para reducir a cero los elementos que se encuentran debajo de la diagonal principal son:

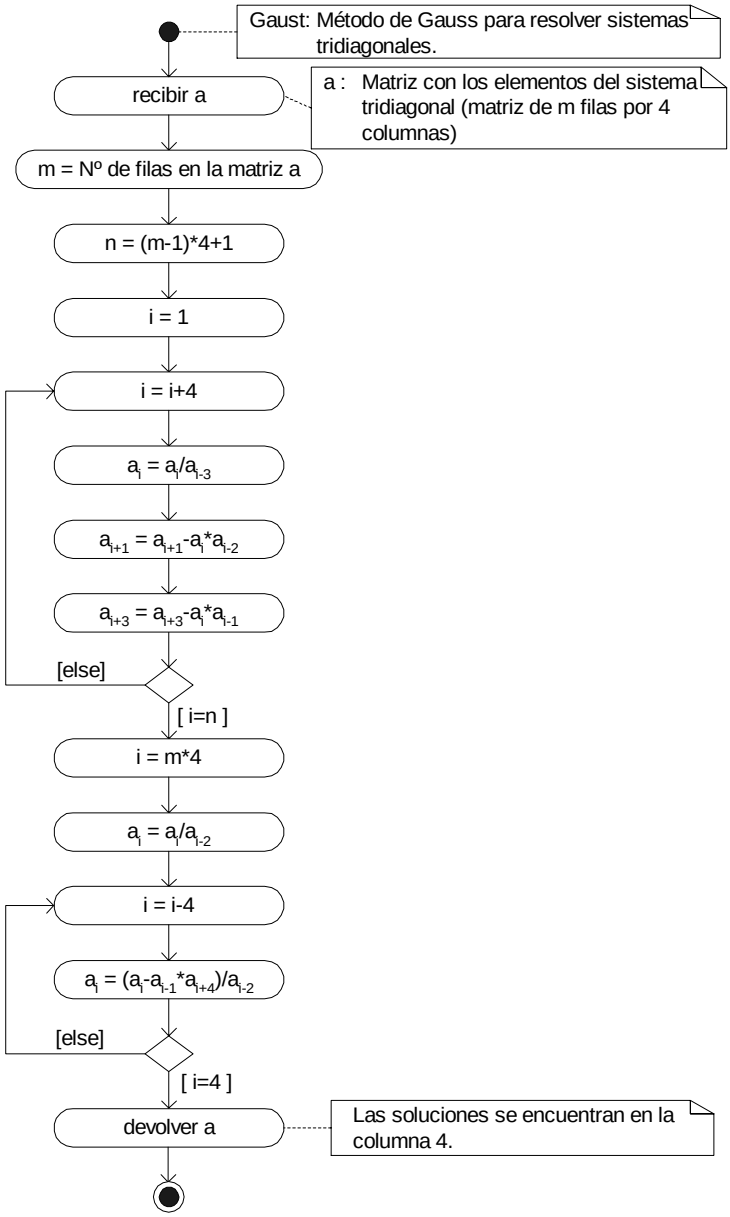
$$\left.\begin{array}{l}
a_{i,1} = \dfrac{a_{i,1}}{a_{i-1,2}} \\
a_{i,2} = a_{i,2} - a_{i,1} * a_{i-1,3} \\
a_{i,4} = a_{i,4} - a_{i,1} * a_{i-1,4}
\end{array}\right\} \quad i = 2 \rightarrow m \qquad (24)$$

Las ecuaciones generales para calcular los resultados por sustitución inversa son:

$$\begin{array}{l}
x_m = a_{m,4} = \dfrac{a_{m,4}}{a_{m,2}} \\
x_i = a_{i,4} = \dfrac{a_{i,4} - a_{i,3}x_{i+1}}{a_{i,2}} = \dfrac{a_{i,4} - a_{i,3}a_{i+1,4}}{a_{i,2}} \quad i = m-1 \rightarrow 1
\end{array} \qquad (25)$$

5.7.1.1. Algoritmo y código para resolver sistemas tridiagonales con el método de Gauss.

El algoritmo y el código para resolver sistemas tridiagonales con el método de *Gauss* son los siguientes:




```
xNAME Gaust ( Método de Gauss para resolver sistemas tridiagonales )
::          ( Dato en la pila a= Matriz con los elementos del sistema )
  CK1&Dispatch  ( Temporales: m=3; n=2; i=1 )
  FOUR
  ::
    DUP MDIMSDROP
    DROP DUP #1- FOUR #* #1+ ONE
    ' NULLLAM THREE NDUPN DOBIND
    BEGIN
      1GETLAM #4+ 1PUTLAM                ( i=i+4 )
      1GETLAM PULLREALEL SWAP            ( a[i] )
      1GETLAM #3- PULLREALEL SWAP 3UNROLL ( a[i-3] )
```

```
      %/ 1GETLAM PUTREALEL                ( a[i]=a[i]/a[i-3] )
      1GETLAM #1+ PULLREALEL SWAP         ( a[i+1] )
      1GETLAM PULLREALEL SWAP             ( a[i] )
      1GETLAM #2- PULLREALEL SWAP 4UNROLL ( a[i-2] )
      %* %- 1GETLAM #1+ PUTREALEL         ( a[i+1]=a[i+1]+a[i]*a[i-2] )
      1GETLAM #3+ PULLREALEL SWAP         ( a[i+3] )
      1GETLAM PULLREALEL SWAP             ( a[i] )
      1GETLAM #1- PULLREALEL SWAP 4UNROLL ( a[i-1] )
      %* %- 1GETLAM #3+ PUTREALEL         ( a[i+3]=a[i+3]+a[i]*a[i-1] )
      1GETLAM 2GETLAM #=                  ( i=n? )
    UNTIL
    3GETLAM FOUR #* 1PUTLAM               ( i=m*4 )
    1GETLAM PULLREALEL SWAP               ( a[i] )
    1GETLAM #2- PULLREALEL SWAP UNROT     ( a[i-2] )
    %/ 1GETLAM PUTREALEL                  ( a[i]=a[i]/a[i-2] )
    BEGIN
      1GETLAM #4- 1PUTLAM                 ( i=i-4 )
      1GETLAM PULLREALEL SWAP             ( a[i] )
      1GETLAM #1- PULLREALEL SWAP         ( a[i-1] )
      1GETLAM #4+ PULLREALEL SWAP 4UNROLL ( a[i+4] )
      %* %- SWAP
      1GETLAM #2- PULLREALEL SWAP 3UNROLL ( a[i-2] )
      %/ 1GETLAM PUTREALEL                ( a[i]=[a[i]-a[i-1]a[i+4]]/a[i-2])
      1GETLAM FOUR #=                     ( i=4? )
    UNTIL
    ABND
  ;
;
```

Probaremos este código resolviendo el siguiente sistema:

$$\begin{array}{rcrcrcrcrcl} 2x_1 & + & x_2 & + & 0x_3 & + & 0x_4 & + & 0x_5 & = & 6 \\ x_1 & + & 4x_2 & + & x_3 & + & 0x_4 & + & 0x_5 & = & 36 \\ 0x_1 & + & x_2 & + & 4x_3 & + & 1x_4 & + & 0x_5 & = & 72 \\ 0x_1 & + & 0x_2 & + & x_3 & + & 4x_4 & + & x_5 & = & 108 \\ 0x_1 & + & 0x_2 & + & 0x_2 & + & x_4 & + & 2x_5 & = & 66 \end{array} \qquad (26)$$

El programa elaborado para este fin es:

```
« [ [ 0 2 1 6 ]
    [ 1 4 1 36 ]
    [ 1 4 1 72 ]
    [ 1 4 1 108 ]
    [ 1 2 0 66 ] ]
  Gaust 4 COL- SWAP DROP
  AXL { x1 x2  x3 x4 x5 } →TAG LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 0.
x2: 6.
X3: 12.
x4: 18.
x5: 24.
s: .4596
```

Que son las 5 soluciones y el tiempo empleado en obtenerlas.

5.7.1.2. Ejercicios

18. Resuelva el siguiente sistema de ecuaciones lineales empleando el método de *Gauss* para sistemas tridiagonales.

$$\begin{array}{rcrcrcrcrcrcrcr}
0.6x_1 & + & 0.1x_2 & + & 0x_3 & + & 0x_4 & + & 0x_5 & + & 0x_6 & + & 0x_7 & = & -1.23 \\
0.1x_1 & + & 0.4x_2 & + & 0.1x_3 & + & 0x_4 & + & 0x_5 & + & 0x_6 & + & 0x_7 & = & 3.96 \\
0x_1 & + & 0.1x_2 & + & 0.4x_3 & + & 0.1x_4 & + & 0x_5 & + & 0x_6 & + & 0x_7 & = & 9.60 \\
0x_1 & + & 0x_2 & + & 0.1x_3 & + & 0.4x_4 & + & 0.1x_5 & + & 0x_6 & + & 0x_7 & = & 11.52 \\
0x_1 & + & 0x_2 & + & 0x_3 & + & 0.1x_4 & + & 0.4x_5 & + & 0.1x_6 & + & 0x_7 & = & -21.42 \\
0x_1 & + & 0x_2 & + & 0x_3 & + & 0x_4 & + & 0.1x_5 & + & 0.4x_6 & + & 0.1x_7 & = & -4.50 \\
0x_1 & + & 0x_2 & + & 0x_3 & + & 0x_4 & + & 0x_5 & + & 0.1x_6 & + & 0.6x_7 & = & 0.60
\end{array}$$

19. Resuelva el siguiente sistema de ecuaciones lineales empleando el método de Gauss para sistemas tridiagonales.

$$\begin{array}{rcrcrcrcrcrcrcrcr}
-2.0304x_1 & + & x_2 & + & 0x_3 & + & 0x_4 & + & 0x_5 & + & 0x_6 & + & 0x_7 & + & 0x_8 & + & 0x_9 & = & -1.952 \\
x_1 & - & 2.0288x_2 & + & x_3 & + & 0x_4 & + & 0x_5 & + & 0x_6 & + & 0x_7 & + & 0x_8 & + & 0x_9 & = & 0.056 \\
0x_1 & + & x_2 & - & 2.0272x_3 & + & x_4 & + & 0x_5 & + & 0x_6 & + & 0x_7 & + & 0x_8 & + & 0x_9 & = & 0.064 \\
0x_1 & + & 0x_2 & + & x_3 & - & 2.0256x_4 & + & x_5 & + & 0x_6 & + & 0x_7 & + & 0x_8 & + & 0x_9 & = & 0.072 \\
0x_1 & + & 0x_2 & + & 0x_2 & + & x_4 & - & 2.0240x_5 & + & x_6 & + & 0x_7 & + & 0x_8 & + & 0x_9 & = & 0.080 \\
0x_1 & + & 0x_2 & + & 0x_2 & + & 0x_4 & + & x_5 & - & 2.0224x_6 & + & x_7 & + & 0x_8 & + & 0x_9 & = & 0.088 \\
0x_1 & + & 0x_2 & + & 0x_2 & + & 0x_4 & + & 0x_5 & + & x_6 & - & 2.0208x_7 & + & x_8 & + & 0x_9 & = & 0.096 \\
0x_1 & + & 0x_2 & + & 0x_2 & + & 0x_4 & + & 0x_5 & + & 0x_6 & + & x_7 & - & 2.0192x_8 & + & x_9 & = & 0.104 \\
0x_1 & + & 0x_2 & + & 0x_2 & + & 0x_4 & + & 0x_5 & + & 0x_6 & + & 0x_7 & + & x_8 & - & 2.0176x_9 & = & 1.112
\end{array}$$

5.7.2. Método de Gauss - Seidel para sistemas tridiagonales

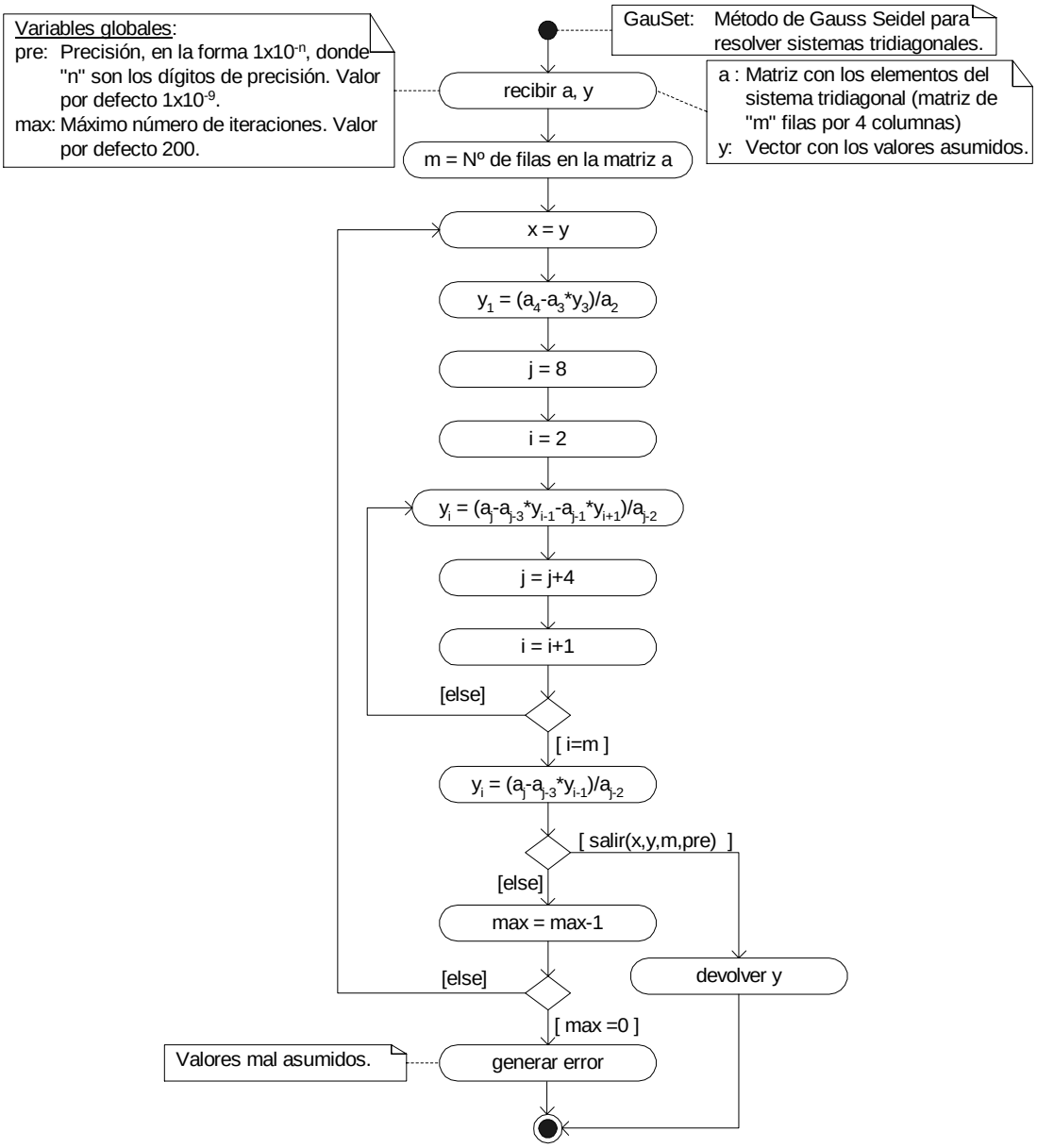
Como generalmente los sistemas tridiagonales tienen además una diagonal dominante, el método de *Gauss – Seidel* resulta particularmente adecuado para resolver estos sistemas porque, como se sabe, una diagonal dominante asegura la convergencia del método. Las ecuaciones generales para el método de *Gauss-Seidel:*

$$\left.\begin{array}{l}
y_1 = \dfrac{a_{1,4} - a_{1,3}y_2}{a_{1,2}} \\
y_i = \dfrac{a_{i,4} - a_{i,1}y_{i-1} - a_{i,3}y_{i+1}}{a_{i,2}} \\
y_m = \dfrac{a_{m,4} - a_{m,1}y_{m-1}}{a_{m,2}}
\end{array}\right\} \quad i = 2 \rightarrow m-1 \qquad (27)$$

5.7.2.1. Algoritmo y código para resolver sistemas tridiagonales con el método de Gauss - Seidel

El algoritmo se presenta en el diagrama de actividades de la siguiente página y el código elaborado en base al mismo es el siguiente:

```
xNAME GauSet ( Sistemas tridiagonales. Método de Gauss-Seidel )
::                ( Datos: a= Matriz aumentada )
  CK2&Dispatch            ( y=7= Vector con los valores asumidos )
  # 44                    ( Locales: x=6; m=5; i=4; j=3; pre=2; max=1 )
  ::
    DUP TOTEMPOB 3PICK MDIMSDROP
    DROP ONE ONE
    ' ID pre @ ?SKIP % 1E-9      ( si no existe "pre", se emplea 1E-9 )
    ' ID max @ ?SKIP # 200       ( si no existe "max", se emplea 200 )
    ' NULLLAM SEVEN NDUPN DOBIND
```




```
BEGIN
  7GETLAM 6GETLAM REPLACE DROP          ( x=y )
  7GETLAM SWAP FOUR PULLREALEL SWAP     ( y a[4] )
  THREE PULLREALEL SWAP 4UNROLL         ( a[3] )
  7GETLAM TWO PULLREALEL SWAP DROP      ( y[2] )
  %* %- ROT TWO PULLREALEL SWAP         ( * - a[2] )
  4UNROLL %/ ONE PUTREALEL DROP         ( y[1]= .../...)
  EIGHT 3PUTLAM                         ( j=8 )
  TWO 4PUTLAM                           ( i=2 )
  BEGIN
    7GETLAM SWAP 3GETLAM PULLREALEL     ( y a[j] )
    SWAP 3GETLAM #3- PULLREALEL         ( a[j-3] )
    SWAP 4UNROLL 7GETLAM 4GETLAM #1-
    PULLREALEL SWAP DROP %* %-          ( y[i-1] * - )
    ROT 3GETLAM #1- PULLREALEL SWAP
    4UNROLL                             ( a[j-1] )
    7GETLAM 4GETLAM #1+ PULLREALEL
```

```
        SWAP DROP %* %-                    ( y[i+1] * - )
        ROT 3GETLAM #2- PULLREALEL SWAP    ( a[j-2] )
        4UNROLL %/ 4GETLAM PUTREALEL DROP  ( y[i]=.../... )
        3GETLAM #4+ 3PUTLAM                ( j=j+4 )
        4GETLAM #1+ 4PUTLAM                ( i=i+1 )
        4GETLAM 5GETLAM #=                 ( i=m? )
      UNTIL
      7GETLAM SWAP 3GETLAM PULLREALEL      ( y a[j] )
      SWAP 3GETLAM #3- PULLREALEL          ( a[j-3] )
      SWAP 4UNROLL 7GETLAM 4GETLAM #1-
      PULLREALEL SWAP DROP %* %-           ( y[i-1] * - )
      ROT 3GETLAM #2- PULLREALEL SWAP      ( a[j-2] )
      4UNROLL %/ 4GETLAM PUTREALEL DROP    ( y[i]=.../... )
      6GETLAM 7GETLAM 5GETLAM 2GETLAM
      salir                                ( salir[x,y,m,pre]? )
      ITE :: DROP 7GETLAM TRUE ; FALSE
      1GETLAM #1- 1PUTLAM                  ( max=max-1 )
      1GETLAM #0=                          ( max=0? )
      IT :: DROP ABND # A01 DO#EXIT ;      ( # A01 = Bad guesses )
    UNTIL
    ABND
  ;
;
```

Probaremos este módulo volviendo a resolver el sistema (32).

$$\begin{array}{rcrcrcrcrcl} 2x_1 & + & x_2 & + & 0x_3 & + & 0x_4 & + & 0x_5 & = & 6 \\ x_1 & + & 4x_2 & + & x_3 & + & 0x_4 & + & 0x_5 & = & 36 \\ 0x_1 & + & x_2 & + & 4x_3 & + & 1x_4 & + & 0x_5 & = & 72 \\ 0x_1 & + & 0x_2 & + & x_3 & + & 4x_4 & + & x_5 & = & 108 \\ 0x_1 & + & 0x_2 & + & 0x_2 & + & x_4 & + & 2x_5 & = & 66 \end{array} \qquad (32)$$

El programa elaborado para este fin es:

```
« [[ 0 2 1 6 ]
   [ 1 4 1 36 ]
   [ 1 4 1 72 ]
   [ 1 4 1 108 ]
   [ 1 2 0 66 ]]
  [ 0 0 0 0 0 ]
  GauSet AXL { x1 x2  x3 x4 x5 } →TAG LIST→ DROP
»
```

Haciendo correr el programa con *TEVAL* se obtiene:

```
x1: 0.
x2: 6.
X3: 12.
x4: 18.
x5: 24.
s: .4596
```

Que son las 5 soluciones y el tiempo empleado en obtenerlas.

5.7.2.2. Ejercicios

Resuelva los ejercicios 18 y 19 con el método de *Gauss-Seidel* trabajando en el primer caso con 7 dígitos de precisión y en el segundo con 6.

6.SISTEMAS DE ECUACIONES NO LINEALES

Para la resolución de sistemas de ecuaciones no lineales, no se cuentan con métodos analíticos, por lo tanto dichos sistemas sólo pueden ser resueltos mediante métodos numéricos.

Como sucede con prácticamente todos los métodos numéricos, en estos métodos se requieren valores iniciales asumidos (valores de prueba) para comenzar el proceso. En general, a medida que incrementa el número de ecuaciones del sistema, incrementa también el tiempo requerido para lograr convergencia, es importante entonces que siempre que sea posible, se den valores iniciales cercanos a la solución.

En este capítulo estudiaremos un solo método numérico, el método de la *gradiente* y veremos también como resolver sistemas de ecuaciones no lineales empleando la liberaría *SolveSys* (la cual fue empleada en el capítulo anterior para resolver sistemas de ecuaciones lineales).

6.1. RESOLUCIÓN DE SISTEMAS DE ECUACIONES NO LINEALES CON SOLVESYS

El solucionador incorporado en la HP no tiene herramientas para resolver sistemas de ecuaciones no lineales, razón por la cual se debe emplear una librería o programa externo, siendo un buena opción la librería *SolveSys*.

Para resolver un sistema de ecuaciones no lineales con *SolveSys* se siguen los mismos pasos que en la resolución de sistemas lineales. Por ejemplo para resolver el sistema de ecuaciones no lineales:

$$x^2 + 2y^2 = 22$$

$$-2x^2 + xy - 3y = -11$$

Introducimos las siguientes ecuaciones en el editor de ecuaciones de *SolveSys*:

*'x^2+2*y^2= 22'*

*'-2*x^2+x*y-3*x*y= -11'*

Y pasamos a la siguiente ventana pulsando la opción *OK*, entonces *SolveSys* muestra los valores iniciales de las incógnitas *(normalmente 1)*, es en esta ventada donde podemos cambiar los valores iniciales. En nuestro caso nos quedaremos con los valores iniciales propuestos y procederemos a encontrar las soluciones presionando la opción *SOLVE*, con lo cual al cabo de un tiempo *SolveSys* muestra las soluciones:

x = 1.999999952156
y = 3.0000002949

Si se quiere mejorar la exactitud de estos resultados es posible volver a elegir la opción *SOLVE,* con lo cual *SolveSys* recalcula los resultados obteniéndose:

x = 2
y = 3

Que son las soluciones exactas del sistema. Cada vez que se elige la opción *SOLVE, SolveSys* emplea los últimos resultados como valores de prueba y al ser estos más cercanos a la solución los resultados obtenidos son cada vez más exactos.

Como se puede observar con *SolveSys*, los sistemas de ecuaciones se introducen tal como están planteados, no se requiere cambiar variables ni igualar las funciones a cero.

6.2. MÉTODO DE LA GRADIENTE

El método de la gradiente es esencialmente el método de Newton aplicado a sistemas de ecuaciones no lineales en lugar de una sola ecuación no lineal.

En este método se expanden las ecuaciones que conforman el sistema empleando series de Taylor, por ejemplo, si en el sistema tenemos 3 ecuaciones no lineales:

$$\begin{aligned} f_1(x_1, x_2, x_3) &= 0 \\ f_2(x_1, x_2, x_3) &= 0 \\ f_3(x_1, x_2, x_3) &= 0 \end{aligned} \tag{1}$$

Siendo x_1, x_2 y x_3, los valores asumidos para las tres variables del sistema. Expandiendo las tres ecuaciones en series de Taylor (todas las funciones se evalúan en x_1, x_2 y x_3) obtenemos:

$$\begin{aligned} f_1(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3) &= f_1 + \frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 + \frac{\partial f_1}{\partial x_3}\Delta x_3 + \ldots\infty = 0 \\ f_2(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3) &= f_2 + \frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 + \frac{\partial f_2}{\partial x_3}\Delta x_3 + \ldots\infty = 0 \\ f_3(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3) &= f_3 + \frac{\partial f_3}{\partial x_1}\Delta x_1 + \frac{\partial f_3}{\partial x_2}\Delta x_2 + \frac{\partial f_3}{\partial x_3}\Delta x_3 + \ldots\infty = 0 \end{aligned} \tag{2}$$

Donde Δx_1, Δx_2 y Δx_3 son los valores que deberían añadirse a los valores asumidos (x_1, x_2 y x_3) para que las funciones (f_1, f_2 y f_3) se hagan cero. En otras palabras si fuera posible calcular los valores de Δx_1, Δx_2 y Δx_3, las soluciones del sistema serían $y_1 = x_1+\Delta x_1$, $y_2 = x_2+\Delta x_2$ y $y_3 = x_3+\Delta x_3$. Por supuesto al tratarse de series infinitas, no es posible en la práctica calcular dichos valores, sin embargo, si es posible obtener una aproximación de los mismos tomando en cuenta sólo los términos de primero orden (que es lo que hace el método de Newton), es decir:

$$\begin{aligned} \frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 + \frac{\partial f_1}{\partial x_3}\Delta x_3 &= -f_1 \\ \frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 + \frac{\partial f_2}{\partial x_3}\Delta x_3 &= -f_2 \\ \frac{\partial f_3}{\partial x_1}\Delta x_1 + \frac{\partial f_3}{\partial x_2}\Delta x_2 + \frac{\partial f_3}{\partial x_3}\Delta x_3 &= -f_3 \end{aligned} \tag{3}$$

Que como vemos conforman un sistema de 3 ecuaciones lineales con 3 incógnitas (asumiendo por supuesto que sea posible calcular las derivadas parciales). Debemos recordar no obstante que estos valores, al resultar de los términos de primer orden, son sólo aproximados y que por lo tanto no nos permiten calcular realmente las soluciones aunque si nos aproximan a las mismas, por lo tanto ahora $y_1 = x_1+\Delta x_1$, $y_2 = x_2+\Delta x_2$ y $y_3 = x_3+\Delta x_3$, son valores más cercanos a la solución que los valores asumidos (x_1, x_2 y x_3), pero no constituyen la solución del sistema.

En consecuencia y como es la característica del método de Newton, la solución del sistema se convierte en un problema iterativo: comenzando con los

valores asumidos x_1 *a* x_n, se calculan nuevos valores y_1 *a* y_n resolviendo el sistema de ecuaciones lineales (previo cálculo de las derivadas parciales), entonces se compran los valores calculados con los asumidos y si son aproximadamente iguales el proceso concluye siendo las soluciones los valores desde y_1 *a* y_n, caso contrario los valores calculados (y_1 *a* y_n) se convierten en los nuevos valores asumidos (x_1 *a* x_n) y se repite el proceso.

En lugar de comparar los valores asumidos con los calculados, se pueden sustituir los valores calculados (y_1 *a* y_n) en las funciones y comprobar si las funciones son aproximadamente iguales a cero, de ser así el proceso concluye, siendo las soluciones los valores desde y_1 *a* y_n, caso contrario los valores calculados (y_1 *a* y_n) se convierten en los nuevos valores asumidos (x_1 *a* x_n) y se repite el proceso.

Si bien en algunos casos donde las funciones son sencillas, es posible calcular las derivadas analíticamente, en la mayoría de los casos es preferible calcularlas numéricamente. Para este fin se puede emplear, como se hizo en el método de Newton, la fórmula de diferencia central de segundo orden, adaptada para el cálculo de las derivadas parciales:

$$\frac{\partial f_i}{\partial x_j} = \frac{f_i(x_1, x_2, ..., x_{j+h}, ...x_n) - f_i(x_1, x_2, ..., x_{j-h}, ...x_n)}{2h}; \quad h = x_j * 10^{-4} \qquad (4)$$

Se ha comprobado que el valor propuesto para *h* (x_j*10^{-4}) es un valor adecuado en la mayoría de los casos, no obstante, la fórmula de diferencia central, sólo requiere que dicho valor sea un incremento pequeño de x_j, por lo tanto si en algún se considera que dicho valor no es adecuado, puede ser reemplazado por otro valor.

El sistema de ecuaciones lineales que se forma en el método, puede ser resuelto empleando alguno de los métodos estudiados en el capítulo anterior. En nuestro caso emplearemos el método de Cholesky.

6.2.1. Ejemplo manual

Para comprender mejor el método y contar con valores de prueba, resolveremos manualmente el sistema de ecuaciones resuelto con *SolveSys*:

$$x^2 + 2y^2 = 22$$
$$-2x^2 + xy - 3y = -11$$

Para estar de acuerdo con la simbología empleada en la explicación del método, colocamos primero estas ecuaciones en función de las variables x_1 y x_2 e igualamos las ecuaciones a cero:

$$f_1 = {x_1}^2 + 2{x_2}^2 - 22 = 0$$
$$f_2 = -2{x_1}^2 + x_1x_2 - 3x_2 + 11 = 0$$

Asumiendo los siguientes valores iniciales: $x_1=1.5$ y $x_2=2$, procedemos a calcular las funciones y las derivadas parciales que conforman el sistema de ecuaciones lineales (2 en el ejemplo):

$$h = x_1 * 10^{-4} = 1.5 * 10^{-4}$$
$$\frac{\partial f_1}{\partial x_1} = \frac{f_1(x_{1+h}, x_2) - f_1(x_{1-h}, x_2)}{2h} = \frac{f_1(1.50015, 2) - f_1(1.49985, 2)}{2 * 1.5 * 10^{-4}} = 3$$
$$\frac{\partial f_2}{\partial x_1} = \frac{f_2(x_{1+h}, x_2) - f(x_{1-h}, x_2)}{2h} = \frac{f_2(1.50015, 2) - f_2(1.49985, 2)}{2 * 1.5 * 10^{-4}} = -4$$

$$h = x_2 * 10^{-4} = 0.0002$$
$$\frac{\partial f_1}{\partial x_2} = \frac{f_1(x_1, x_{2+h}) - f_1(x_1, x_{2-h})}{2h} = \frac{f_1(1.5, 2.002) - f_1(1.5, 1.9998)}{2 * 0.0002} = 8$$
$$\frac{\partial f_2}{\partial x_2} = \frac{f_2(x_1, x_{2+h}) - f_2(x_1, x_{2-h})}{2h} = \frac{f_2(1.5, 2.002) - f_2(1.5, 1.9998)}{2 * 0.0002} = -1.5$$
$$-f_1 = -f_1(x_1, x_2) = -f_1(1.5, 2) = 11.75$$
$$-f_2 = -f_2(x_1, x_2) = -f_2(1.5, 2) = -3.5$$

Por lo tanto el sistema de ecuaciones lineales a resolver es:

$$\frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 = 3\Delta x_1 + 8\Delta x_2 = 11.75 = -f_1$$
$$\frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 = -4\Delta x_1 - 1.5\Delta x_2 = -3.5 = -f_2$$

De donde la matriz aumentada que se debe mandarse al método de Cholesky es:

$$\begin{vmatrix} 3 & 8 & 11.75 \\ -4 & -1.5 & -3.5 \end{vmatrix}$$

Mandando esta matriz al método de Cholesky obtenemos:

$$\begin{vmatrix} -1.5 & -3.435 & .377272727273 \\ 8 & .375 & 1.32727272727 \end{vmatrix}$$

Por lo tanto los nuevos valores del las incógnitas son:

$$y_1 = x_1 + \Delta x_1 = 1.5 + 0.377272727273 = 1.87727272727$$
$$y_2 = x_2 + \Delta x_2 = 2 + 1.32727272727 = 3.327272727$$

Que como vemos difieren de los valores originales (1.5 y 2), razón por la cual el proceso debe ser repetido empleando ahora como valores asumidos los valores calculados: $x_1=y_1=1.87727272727$, $x_2=y_2=3.327272727$.

Procediendo como antes obtenemos ahora el siguiente sistema de ecuaciones lineales:

$$\frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 = 3.75454556901\Delta x_1 + 13.309090847\Delta x_2 = -3.6656404958 = -f_1$$
$$\frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 = -4.18181806296\Delta x_1 - 1.12272732241\Delta x_2 = -0.2160743802 = -f_2$$

Y resolviendo el mismo con Cholesky obtenemos los siguientes valores para Δx_1 y Δx_2: Δx_1 = *0.135908838457* Δx_2= *-0.313764213575*. Por lo tanto los nuevos valores calculados son:

$$y_1 = x_1 + \Delta x_1 = 1.87727272727 + 0.135908838457 = 2.01318156573$$
$$y_2 = x_2 + \Delta x_2 = 3.327272727 - 0.313764213575 = 3.0135085137$$

Como vemos los valores asumidos y calculados aún son diferentes, razón por la cual es necesario repetir el proceso empleando ahora como valores asumidos los nuevos valores calculados: $x_1=y_1=2.01318156573$, $x_2=y_2=3.0135085137$.

Así obtenemos el siguiente sistema de ecuaciones lineales:

$$\frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 = 4.02636311497\Delta x_1 + 12.0540339723\Delta x_2 = -0.2153671409 = -f_1$$
$$\frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 = -5.03921761092\Delta x_1 - 0.986818350265\Delta x_2 = 0.0795857864 = -f_2$$

Resolviendo el mismo con Cholesky obtenemos: $\Delta x_1 = -0.013154952264$ $\Delta x_2 = -1.34727118489E\text{-}2$. Por lo tanto los nuevos valores calculados son:

$$y_1 = x_1 + \Delta x_1 = 2.01318156573 - 0.013154952264 = 2.00002661347$$
$$y_2 = x_2 + \Delta x_2 = 3.0135085137 - 1.3427118489E - 2 = 3.00003580185$$

Ahora los valores calculados y asumidos son iguales en los dos primeros dígitos. Repitiendo el proceso una vez más obtenemos el siguiente sistema de ecuaciones lineales:

$$\frac{\partial f_1}{\partial x_1}\Delta x_1 + \frac{\partial f_1}{\partial x_2}\Delta x_2 = 4.00005327235\Delta x_1 + 12.0001432909\Delta x_2 = -0.0005360794 = -f_1$$
$$\frac{\partial f_2}{\partial x_1}\Delta x_1 + \frac{\partial f_2}{\partial x_2}\Delta x_2 = -5.00007071538\Delta x_1 - 0.999973399701\Delta x_2 = 0.0001688697 = -f_2$$

Resolviendo el mismo con Cholesky obtenemos: $\Delta x_1 = -2.66134355722E\text{-}5$ $\Delta x_2 = -3.58015924923E\text{-}5$. Por lo tanto los nuevos valores calculados son:

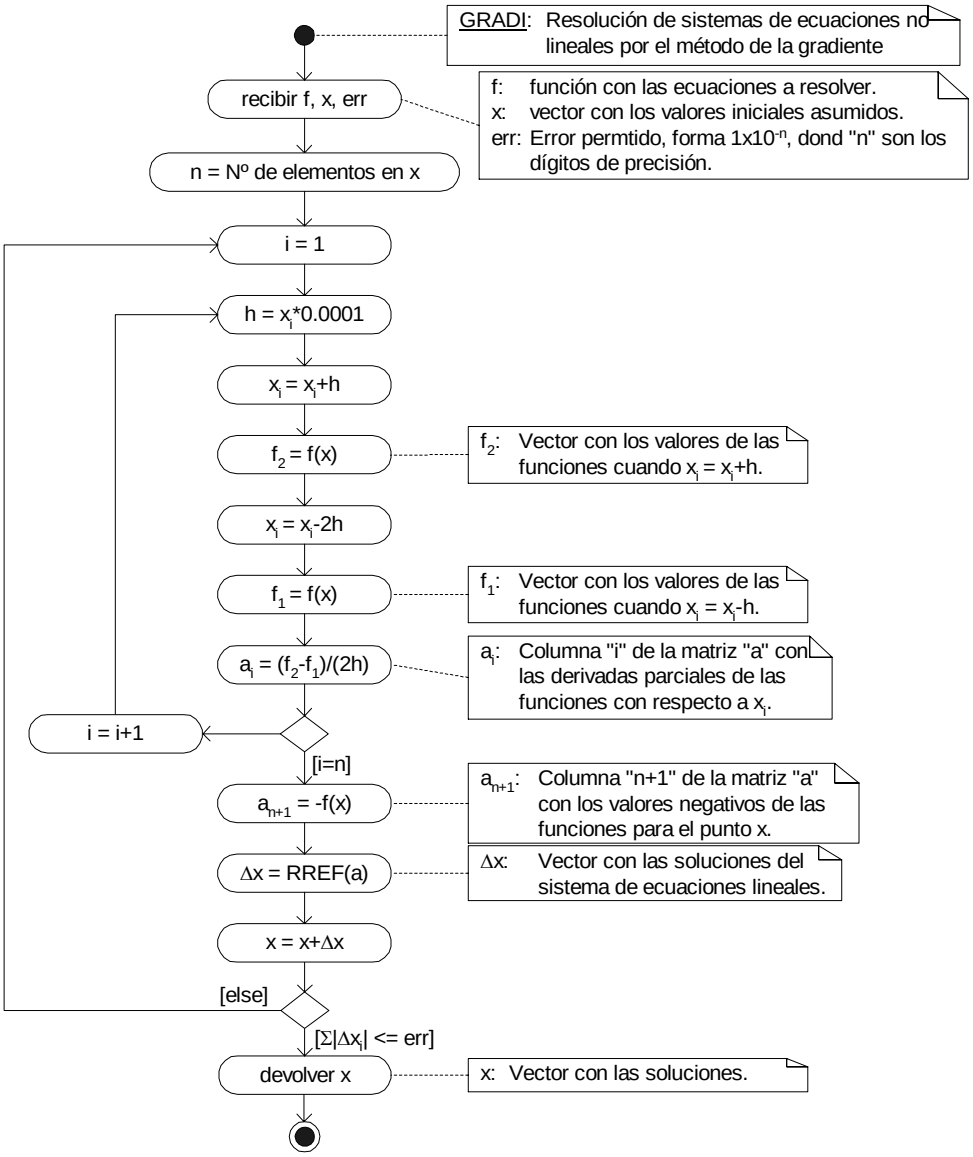
$$y_1 = x_1 + \Delta x_1 = 2.00002661347 - 2.66134355722E - 5 = 2.00000000003$$
$$y_2 = x_2 + \Delta x_2 = 3.00003580185 - 3.58015924923E - 5 = 3.00000000026$$

Ahora los valores calculados y asumidos son iguales en los primeros 5 dígitos, por lo que el proceso puede concluir en este punto. Si se repite una vez más el procedimiento se obtienen los resultados exactos: $x_1 = 2$ y $x_2 = 3$.

6.2.2. Algoritmo y código

El algoritmo del método de la Gradiente se presenta en el diagrama de actividades de la siguiente página y el código elaborado en base al mismo es el siguiente:

```
« 0 0 0 0   f x err n f1 f2 h
  « x SIZE 1 GET 'n' STO CLLCD
    DO
      1 n FOR i
        x i GET .0001 * 'h' STO
        x i x i GET h + PUT
        V  f EVAL n  ARRY 'f2' STO
        x i x i GET h – PUT
        V  f EVAL n  ARRY 'f1' STO
        f2 f1 - 2 h * /
      NEXT
      x V  f EVAL n  ARRY NEG
      n 1 + COL  RREF
      n 1 + COL- SWAP DROP
      DUP x + 'x' STO
      AXL ABS …LIST
      DUP "err"  TAG 6 DISP
      x "x"  TAG 7 DISP
      err ‰
```




```
    UNTIL
    END
    x
  »
»
```

El cual se asumirá que está guardado con el nombre "*GRADI*".

Como ejemplo resolveremos el sistema de ecuaciones no lineales resuelto manualmente empleando el programa *Gradi.* Para ello escribimos el siguiente programa:

```
«
   «   x y
     « x SQ 2 y SQ * + 22 –
       -2 x SQ * x y * + 3 y * - 11 +
     »
   » [1.5 2] 1E-6 GRADI AXL { x y }  TAG  LIST  DROP
»
```

Haciendo correr el programa (EVAL) se obtienen los resultados: *x: 2* y *y: 3*, que son las soluciones exactas del sistema de ecuaciones.

6.2.3. Ejercicios

Elabore programas para resolver los siguientes sistemas de ecuaciones no lineales por el método de la gradiente.

1.

$$4 - x^2 + y^2 = 0$$
$$1 - e^x - y = 8$$

2.

$$e^x - y = 0$$
$$xy - e^x = 0$$

3.

$$3xy - y + z^2 = 29$$
$$xyz - yz^2 = 8$$
$$2xy - y^2 - z^2 = 15$$

4.

$$5xy + 3x^2z - 2z = 319$$
$$6yz - x^{2.1} = 42.63$$
$$3x^2 + 2y^2 - z^3 = 80$$

5.

$$x^2 + y^2 + z^2 = 9$$
$$xyz = 1$$
$$z + y - z^2 = 80$$

6.

$$x^2 + y^2 - z^2 = 9$$
$$xyz = 1$$
$$z - y - z^2 = 80$$

7.

$$xyz - x^2 + y^2 = 1.34$$
$$xy - z^2 = 0.09$$
$$e^x - e^y + z = 0.41$$

8.

$$
\begin{aligned}
x_1 + x_4 &= 3 \\
2x_1 + x_2 + x_4 + x_7 + x_8 + x_9 + 2x_{10} &= 10 + r \\
x_2 + 2x_5 + x_6 + x_7 &= 8 \\
2x_3 + x_5 &= 4r \\
x_1 x_5 &= a_1 x_2 x_4 \\
x_6 x_2^{1/2} &= a_2 (x_2 x_4 x_{11})^{1/2} \\
x_7 x_4^{1/2} &= a_3 (x_1 x_4 x_{11})^{1/2} \\
x_8 x_4 &= a_4 x_2 x_{11} \\
x_9 x_4 &= a5x1(x_3 x_{11})^{1/2} \\
x_{10} x_4^{\,2} &= a_6 x_4^{\,2} x_{11} \\
x_{11} &= x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \\
a_1 &= 0.193; \quad a_2 = 0.002597; \quad a_3 = 0.003448; \\
a_4 &= 0.00001799; \quad a_5 = 0.002155; \quad a_6 = 0.00004836; \\
r &= 4.056734
\end{aligned}
$$

7. MÍNIMOS CUADRADOS

Frecuentemente, al diseñar o resolver un problema de ingeniería, se cuenta con una serie de datos puntuales en lugar de funciones o ecuaciones analíticas. Estos datos son empleados normalmente para realizar una serie de cálculos que pueden ir desde la estimación de valores intermedios hasta la integración y diferenciación numérica.

Como ya se señaló en el capítulo anterior una de las alternativas es el de obtener los resultados mediante interpolación, la otra alternativa, que estudiaremos en este capítulo es la de ajustar los datos a una ecuación analítica. Esto significa calcular los coeficientes y constantes de una ecuación analítica de manera que prediga lo más fielmente posible el comportamiento de los datos.

Una vez calculados los coeficientes y constantes de la ecuación analítica (es decir una vez efectuado el ajuste), la ecuación resultada puede ser empleada igual que cualquier ecuación analítica, se puede por ejemplo integrar, derivar, calcular valores intermedios, etc.

Normalmente se elige el ajuste (en lugar de la interpolación) cuando los datos con los que se cuenta no son muy exactos. Esta situación se presenta en la mayoría de los datos de origen experimental.

Para ajustar los datos estudiaremos el método de los mínimos cuadrados, desde el punto de vista del cálculo numérico (no estadístico). Específicamente el método que desarrollaremos nos permitirá ajustar datos experimentales a una ecuación analítica de la forma general:

$$y = c_1 f_1(x) + c_2 f_2(x) + ... + c_m f_m(x) \qquad (1)$$

Donde *"y"* es la variable dependiente, *"x"* es la variable independiente y *"c_1"* a *"c_m"* son los coeficientes cuyos valores deben ser calculados de manera que la ecuación refleje lo más fielmente posible el comportamiento de los datos.

Supongamos por ejemplo que se cuentan con los siguientes datos experimentales:

x	4.0	3.2	6.8	9.3	1.2	0.7	5.5	7.4	6.5	2.4
Y	6.2	5.3	9.1	12.0	3.3	1.5	7.9	10.6	8.7	6.3

Y queremos ajustar los mismos a la ecuación analítica:

$$y = a + bx^2 + cx^4 + d\ln(x)$$

Entonces debemos calcular los valores de los coeficientes: ***a, b, c*** y ***d*** de manera tal que al reemplazar un valor de *"x"* en la ecuación, obtengamos un resultado que sea lo más cercano posible al valor de tablas, así para esta tabla si reemplazamos 9.3 en la ecuación deberíamos obtener como resultado 12.0 o un valor cercano a 12. Mientras más cercano sea el valor calculado del valor tabulado, mejor es el ajuste realizado.

Esta ecuación tiene la forma general dada en la ecuación (1) (m=4) siendo los coeficientes: c_1=a, c_2=b, c_3=c, c_4=d y las funciones: $f_1=1$; $f_2=x^2$; $f_3=x^4$ y $f_4=ln(x)$.

Para calcular las constantes de la ecuación se reemplaza cada par de datos tabulados (x_i, y_i) en la ecuación 2 y dado que el ajuste puede no ser perfecto (y generalmente no lo es), esta igualdad no se cumple exactamente, razón por la cual existe una diferencia o residuo. Si llamamos *"r_i"*al residuo que se obtiene al reemplazar cada par de datos *"x_i - y_i"* en la ecuación 2, entonces podemos formar el siguiente sistema de ecuaciones:

$$\begin{aligned} &a+b(4.0)^{0.2}+c(4.0)^4+d\ln(4.0)-6.2=r_1\\ &a+b(3.2)^{0.2}+c(3.2)^4+d\ln(3.2)-5.3=r_2\\ &\ldots\\ &a+b(2.4)^{0.2}+c(2.4)^4+d\ln(2.4)-6.3=r_{10} \end{aligned} \qquad (2)$$

Si el ajuste fuera perfecto, los residuos serían cero, por consiguiente para lograr el mejor ajuste posible se deben minimizar estos residuos, haciendo que los mismos tiendan a cero y esto es justamente lo que se hace en el método de los mínimos cuadrados.

En general para cualquier ecuación de la forma (1) y para un número *"n"* de datos *"x_i - y_i"*, se forma el siguiente sistema de ecuaciones:

$$\begin{aligned} &c_1 f_1(x_1)+c_2 f_2(x_1)+\ldots+c_m f_m(x_1)-y_1=r_1\\ &c_1 f_1(x_2)+c_2 f_2(x_2)+\ldots+c_m f_m(x_2)-y_2=r_2\\ &\ldots\\ &c_n f_1(x_n)+c_2 f_2(x_n)+\ldots+c_m f_m(x_n)-y_n=r_n \end{aligned} \qquad (3)$$

Donde las incógnitas son los coeficientes *"c_1"* a *"c_m"* y el objetivo es minimizar la los residuos, es decir minimizar su sumatoria:

$$\sum_{i=1}^{n} r_i = \text{mínimo} \qquad (4)$$

Puesto que en esta sumatoria los restos *(r_i)* pueden tomar valores positivos o negativos y en consecuencia un valor positivo puede anular a otro negativo, resultando así un mínimo falso, se debe eliminar el signo, lo que se logra simplemente elevando los residuos al cuadrado. Por consiguiente el objetivo es minimizar el cuadrado de los residuos, es decir:

$$\sum_{i=1}^{n} r_i^2 = \text{mínimo} \qquad (5)$$

Esta es la ecuación del método de los mínimos cuadrados y de la cual (como se puede ver) deriva su nombre. El mínimo puede ser encontrado empleando diferentes métodos de optimización. En nuestro caso y como ya dijimos emplearemos los principios del cálculo diferencial, donde sabemos que el mínimo puede ser calculado si se iguala la derivada de la ecuación a cero.

Antes de proceder a encontrar el mínimo de la ecuación (5), es conveniente escribir la ecuación (3) en forma matricial:

$$\begin{bmatrix} f_1(x_1) & f_2(x_1) & \ldots & f_m(x_1)\\ f_1(x_2) & f_2(x_2) & \ldots & f_m(x_2)\\ . & . & . & .\\ f_1(x_n) & f_2(x_n) & \ldots & f_m(x_n) \end{bmatrix} * \begin{bmatrix} c_1\\ c_2\\ ..\\ c_m \end{bmatrix} - \begin{bmatrix} y_1\\ y_2\\ ..\\ y_n \end{bmatrix} = \begin{bmatrix} r_1\\ r_2\\ ..\\ r_n \end{bmatrix} \qquad (6)$$

O en forma simbólica:

$$F*C-Y=R \qquad (7)$$

Entonces derivamos la ecuación (5) e igualamos la derivada a cero:

$$\frac{\partial}{\partial c_k}\left(\sum_{i=1}^{n} r_i^2\right) = \sum_{i=1}^{n} \frac{\partial r_i^2}{\partial c_k} = \sum_{i=1}^{n} 2 r_i \frac{\partial r_i}{c_k} = 2\sum_{i=1}^{n} r_i \frac{\partial r_i}{\partial c_k} = 0 \qquad (8)$$

De donde obtenemos:

$$\sum_{i=1}^{n} r_i \frac{\partial r_i}{\partial c_k} = 0 \ \{\mathrm{k} = 1 \rightarrow \mathrm{m} \tag{9}$$

Que de manera desarrollada es:

$$\begin{aligned} & r_1 \frac{\partial r_1}{\partial c_1} + r_2 \frac{\partial r_2}{\partial c_1} + \ldots + r_n \frac{\partial r_n}{\partial c_1} = 0 \\ & r_1 \frac{\partial r_1}{\partial c_2} + r_2 \frac{\partial r_2}{\partial c_2} + \ldots + r_n \frac{\partial r_n}{\partial c_2} = 0 \\ & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ & r_1 \frac{\partial r_1}{\partial c_m} + r_2 \frac{\partial r_2}{\partial c_m} + \ldots + r_n \frac{\partial r_n}{\partial c_m} = 0 \end{aligned} \tag{10}$$

Observando la forma que tienen las funciones de la ecuación 4, vemos que las derivadas parciales son iguales a las funciones que contienen el coeficiente con relación al cual se está derivando.

Así por ejemplo en la derivada parcial de r_2 con respecto a c_1 $(\partial r_2/\partial c_1)$ sólo queda la función f_1 $(f_1(x_2))$; en la derivada parcial de r_1 con respecto a c_2, $(\partial r_1/\partial c_2)$ sólo queda la función f_2 $(f_2(x_1))$, y en la derivada parcial de r_n con respecto a c_2 $(\partial r_n/\partial c_n)$ sólo queda la función f_2 $(f_2(x_n))$. Por lo tanto para cualquier derivada se cumple la siguiente relación general:

$$\frac{\partial \mathrm{r_i}}{\partial c_k} = f_k(x_i) \ \begin{cases} \mathrm{i} = 1 \rightarrow \mathrm{n} \\ \mathrm{k} = 1 \rightarrow \mathrm{m} \end{cases} \tag{11}$$

Reemplazando la ecuación (11) en la ecuación (10) obtenemos:

$$\begin{aligned} & r_1 f_1(x_1) + r_2 f_1(x_2) + \ldots + r_n f_1(x_n) = 0 \\ & r_1 f_2(x_1) + r_2 f_2(x_2) + \ldots + r_n f_2(x_n) = 0 \\ & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ & r_1 f_m(x_1) + r_2 f_m(x_2) + \ldots + r_n f_m(x_n) = 0 \end{aligned} \tag{12}$$

Que expresado de manera matricial es:

$$\begin{bmatrix} f_1(x_1) & f_1(x_2) & \ldots & f_1(x_n) \\ f_2(x_1) & f_2(x_2) & \ldots & f_2(x_n) \\ . & . & . & . \\ f_m(x_1) & f_m(x_2) & \ldots & f_m(x_n) \end{bmatrix} * \begin{bmatrix} r_1 \\ r_2 \\ .. \\ r_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ .. \\ 0 \end{bmatrix} \tag{13}$$

La matriz de funciones de la ecuación (12) es la transpuesta de la matriz de funciones de la ecuación (6). Por consiguiente la ecuación (13) puede ser escrita como:

$$F^T * R = 0 \tag{14}$$

Reemplazando ahora la ecuación (7) en la ecuación (14) resulta:

$$F^T * (F * C - Y) = 0 \tag{15}$$

Realizando algunas operaciones y reordenando se tiene:

$$F^T * F * C = F^T * Y \tag{16}$$

La ecuación (16) es un sistema de *"m"* ecuaciones lineales con *"m"* incógnitas, donde los coeficientes del sistema son la matriz resultante de la

multiplicación F^T*F, el vector de las constantes resulta de la multiplicación: F^T*Y, siendo las incógnitas el vector de coeficientes C.

La matriz aumentada del sistema se obtiene añadiendo a la matriz de los coeficientes (F^T*F), el vector de las constantes (F^T*Y). Entonces las incógnitas (vector C) pueden ser calculadas mandando la matriz aumentada a uno de los métodos estudiados en el capítulo de sistemas de ecuaciones lineales o *RREF*.

7.1 Coeficientes de correlación

El coeficiente de correlación nos permite saber cuan bien se ajustan los datos a la ecuación propuesta y se calcula con la siguiente ecuación:

$$cor = \sqrt{\frac{\sum_{i=1}^{n}\left(y_{ci} - \bar{y}\right)^2}{\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2}} \qquad (17)$$

Donde "n" es el número de datos, "x_i" es el iésimo elemento del vector "x" (variable independiente); "y_i" es el iésimo elemento del vector "y" (variable dependiente), y_{ci} es el valor de la variable dependiente calculado al reemplazar "x_i" en la ecuación ajustada y $\bar{y}$ es el promedio de los datos del vector "y".

Si el coeficiente de correlación calculado con la ecuación (17) es 1, entonces el ajuste es perfecto, es decir la ecuación ajustada pasa por todos los puntos conocidos. ***Mientras más cercano sea a 1 el coeficiente de correlación, mejor es el ajuste***, por el contrario mientras más cercano se encuentre el valor de cero peor es el ajuste.

7.2 Ejemplo manual

Para comprender mejor el método ajustaremos manualmente los siguientes datos:

x	1.0	2.5	3.5	4.0
y	3.8	15.0	26.0	33.0

A la ecuación:

$$y = a + bx^2$$

Que escrita en la forma general de la ecuación (1) es:

$$y = c_1 f_1(x) + c_2 f_2(x)$$

Donde c_1= a; c_2= b; $f_1(x)$= 1 y $f_2(x)$= x^2.

Como el número de funciones es 2, entonces $m=2$ y puesto que se cuenta con cuatro datos $n=4$. Entonces la matriz de funciones (F) es:

$$F = \begin{bmatrix} f_1(x_1) & f_2(x_1) \\ f_1(x_2) & f_2(x_2) \\ f_1(x_3) & f_2(x_3) \\ f_1(x_3) & f_2(x_4) \end{bmatrix} = \begin{bmatrix} f_1(1.0) & f_2(1.0) \\ f_1(2.5) & f_2(2.5) \\ f_1(3.5) & f_2(3.5) \\ f_1(4.0) & f_2(4.0) \end{bmatrix} = \begin{bmatrix} 1 & 1.0^2 \\ 1 & 2.5^2 \\ 1 & 3.5^2 \\ 1 & 4.0^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 6.25 \\ 1 & 12.25 \\ 1 & 16 \end{bmatrix}$$

Siendo la matriz transpuesta:

$$F^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 6.25 & 12.25 & 16.0 \end{bmatrix}$$

Entonces la matriz de los coeficientes es:

$$F^T * F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 6.25 & 12.25 & 16.0 \end{bmatrix} * \begin{bmatrix} 1 & 1 \\ 1 & 6.25 \\ 1 & 12.25 \\ 1 & 16 \end{bmatrix} = \begin{bmatrix} 4 & 35.5 \\ 35.5 & 446.125 \end{bmatrix}$$

Y el vector de las constantes:

$$F^T * Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 6.25 & 12.25 & 16.0 \end{bmatrix} * \begin{bmatrix} 3.8 \\ 15.0 \\ 26.0 \\ 33.0 \end{bmatrix} = \begin{bmatrix} 77.8 \\ 944.05 \end{bmatrix}$$

En consecuencia el sistema de ecuaciones lineales formado para este ajuste es:

$$\begin{array}{ccccc} 4c_1 & + & 35.5c_2 & = & 77.8 \\ 35.5c_1 & + & 446.125c_2 & = & 944.05 \end{array}$$

Por lo tanto la matriz aumentada es:

$$\begin{bmatrix} 4 & 35.5 & 77.8 \\ 35.5 & 446.125 & 944.05 \end{bmatrix}$$

Introduciendo esta matriz en la pila:

```
[ [ 4 35.5 77.8 ] [ 35.5 446.125 944.05] ] RREF
```

Obtenemos la matriz:

$$\begin{bmatrix} 1 & 0 & 2.27896995708 \\ 0 & 1 & 1.9347639485 \end{bmatrix}$$

Que como sabemos contiene las soluciones en la última columna. Por lo tanto la ecuación ajustada es:

$$y = 2.27896995708 + 1.9347639485x^2$$

Calculemos ahora el coeficiente de correlación. Para ellos calculamos primero el promedio de los datos del vector *"y"* (variable dependiente):

$$\bar{y} = \frac{3.8 + 15 + 26 + 33}{4} = 19.45$$

Reemplazando ahora uno a uno los valores del vector *"x"* (variable independiente) en la ecuación ajustada obtenemos los valores de "y_{ci}":

x	1	2.5	3.5	4.0
y_c	4.21373390558	14.3712446352	25.9798283262	33.2351931331

Reemplazando ahora estos valores en la ecuación (17) y realizando las operaciones involucradas obtenemos:

$$cor = \sqrt{\frac{\sum_{i=1}^{n}\left(yc_i - \bar{y}\right)^2}{\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2}} = \sqrt{\frac{490.607768241}{491.23}} = 0.99936645876$$

Puesto que el valor del coeficiente de correlación es casi uno, se concluye que el ajuste casi perfecto. No obstante y aun cuando el coeficiente de correlación es casi uno los valores calculados con la ecuación no reproducen exactamente los valores conocidos (tal como se puede observar en la anterior tabla), algo que sucede normalmente cuando se ajustan datos a una ecuación analítica.

7.3 ALGORITMO

El algoritmo ha sido dividido en 3 módulos: a) un módulo para calcular los elementos de la matriz de funciones, b) un módulo para calcular el coeficiente de correlación y c) el módulo principal del método.

Cálculo de la matriz de funciones (F)

La matriz de las funciones se calcula reemplazando los valores conocidos de la variable independiente en cada una de las funciones, para ello se requieren dos contadores (en dos ciclos), uno para los datos conocidos, el cual va desde 1 hasta *"n"* y otro para las funciones que va desde 1 hasta *"m"*. El algoritmo propuesto se presenta en la figura 8.1.

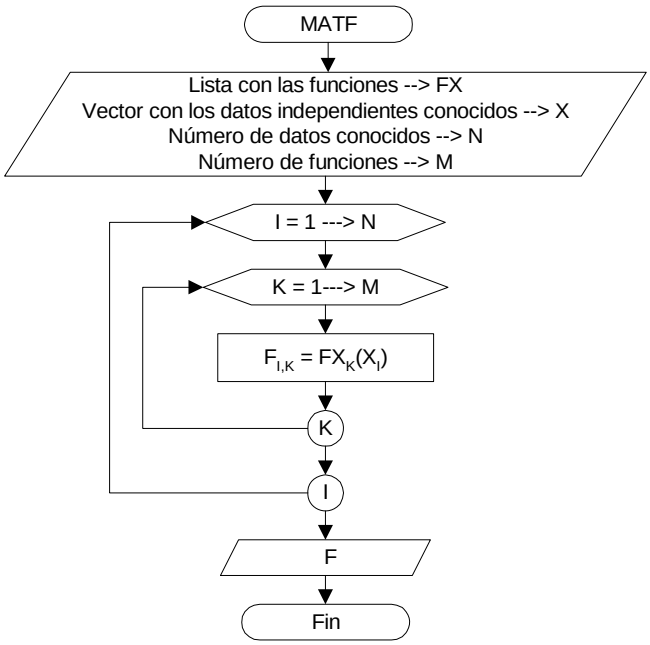


Figura 8.1. Cálculo de la matriz de funciones

El código elaborado en base al mismo es el siguiente:

```
« → FX X N M
 « N M 2 →LIST 0 CON
   1 N FOR I
     1 M FOR K
       I K 2 →LIST X I GET
       FX K GET EVAL PUT
     NEXT
   NEXT
 »
»
```

Como puede se puede observar en el código, la matriz resultante se crea y mantiene en la pila (no es almacenada en ninguna variable). Podemos hacer

correr el programa con los datos del ejemplo manual Para ello escribimos lo siguiente en la pila:

{« DROP 1 » « SQ »} [1 2.5 3.5 4] 4 2 MATF

Obteniéndose la matriz:

$$\begin{bmatrix} 1. & 1. \\ 1. & 6.25 \\ 1. & 12.25 \\ 1. & 16. \end{bmatrix}$$

Que es el mimo resultado obtenido en el ejemplo manual.

Cálculo del coeficiente de correlación

El algoritmo propuesto se presenta en la figura 8.2.

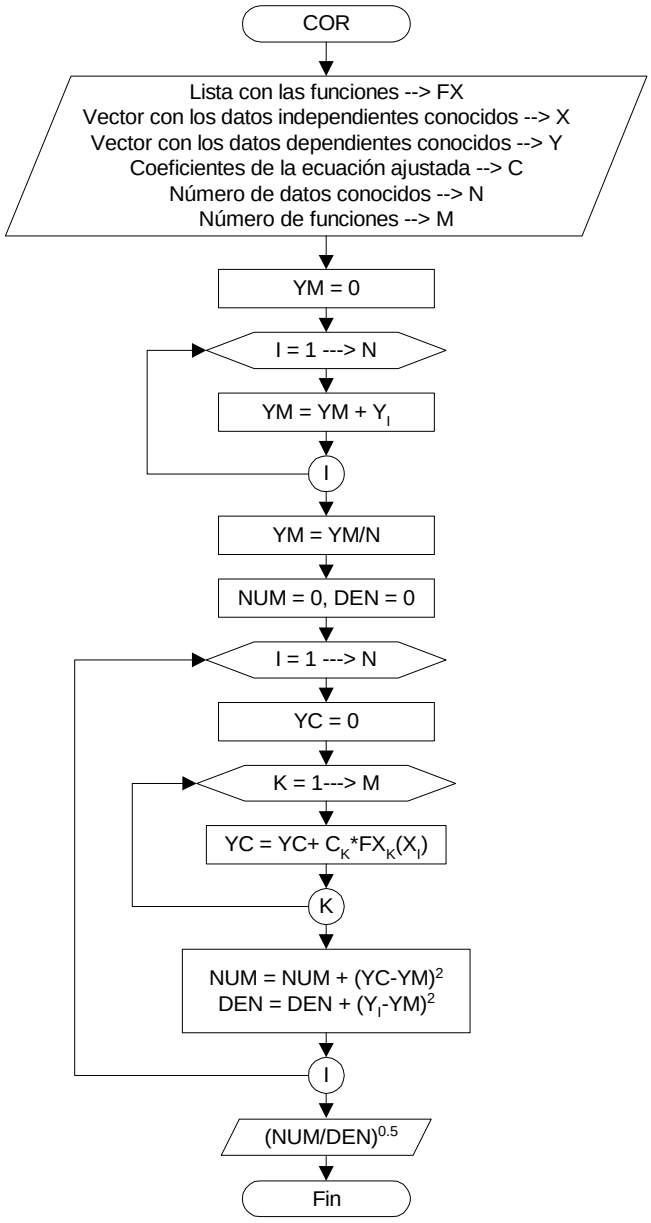


Figura 8.2. Cálculo del coeficiente de correlación

El código elaborado siguiendo el algoritmo es el siguiente:

```
« 0 0 0 0 → FX X Y C N M YM NUM
  DEN YC
 « 1 N FOR I
     Y I GET 'YM' STO+
   NEXT
   'YM' N STO/
   1 N FOR I
     0 'YC' STO
     1 M FOR K
       C K GET X I GET FX K GET
       EVAL * 'YC' STO+
     NEXT
     YC YM - SQ 'NUM' STO+
     Y I GET YM - SQ 'DEN' STO+
   NEXT
   NUM DEN / √
 »
»
```

Probamos escribiendo en la pila:

```
{« DROP 1 » « SQ »}[1 2.5 3.5 4][3.8 15 26 33][2.27896995708 1.9347639485] 4 2 COR
```

Resultando: 0.99936645876, que es el resultado obtenido en el ejemplo.

Módulo principal

El algoritmo del módulo principal para el método de los mínimos cuadrados se presenta en la figura 8.3

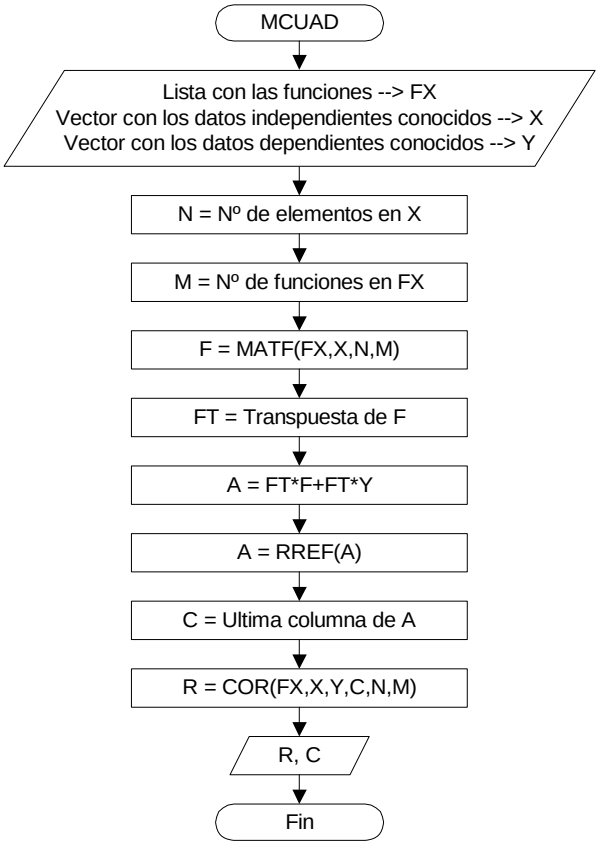


Figura 8.3. Método de los mínimos cuadrados

El código elaborado en base al mismo es el siguiente:

```
« 0 0 0 0 0 → FX X Y N M F FT C
 « X SIZE 1 GET 'N' STO
   FX SIZE 'M' STO
   FX X N M MATF 'F' STO
   F TRAN 'FT' STO
   FT F * FT Y * M 1 + COL+
   RREF M 1 + COL- 'C' STO DROP
   FX X Y C N M COR C
 »
»
```

Como puede observar en el código elaborado no se han almacenado en variables ni la matriz aumentada ***(A)*** ni el coeficiente de correlación ***R***.

Haciendo correr el programa con los datos del ejemplo manual:

{« DROP 1 » « SQ »} [1 2.5 3.5 4][3.8 15 26 33]

Se obtienen los resultados:

0.99936645876

[2.27896995708 1.9347639485]

Que son los mismos resultados del ejemplo.

7.3.2 Ejemplos

1. Ajuste los datos presentados al principio del capítulo a la ecuación 2.

R. En este caso se tienen cuatro coeficientes: $a = c_1$; $b = c_2$; $c = c_3$ y $d = c_4$, siendo las cuatro funciones: $f_1 = 1$; $f_2 = x^2$; $f_3 = x^4$ y $f_4 = ln(x)$. En consecuencia los datos que se deben mandar a ***MCUAD*** son:

{« DROP 1 » « SQ » « 4 ^ » « LN » }

[4 3.2 6.8 9.3 1.2 0.7 5.5 7.4 6.5 2.4]

[6.2 5.3 9.1 12.0 3.3 1.5 7.9 10.6 8.7 6.3]

Haciendo correr ***MCUAD*** se obtienen los resultados:

R = 0.984244860831

[2.68899888264 2.2900760871E-2 2.32865599926E-4 2.59101742141]

Puesto que el coeficiente de correlación es cercano a uno, se concluye que los datos se ajustan bien a la ecuación propuesta.

Los coeficientes de la ecuación ajustada son: $a = 2.68899888264$, $b = 2.2900760871E\text{-}2$, $c = 2.32865599926E\text{-}4$ y $d = 2.59101742141$.

2. Ajuste los datos de la tabla a la ecuación:

$$y = a*x^2 + b/x + c*e^{x/8}$$

x	2.3	4.5	6.7	8.4	9.2	10.1	11.2	12.6	14.5	16.3	17.2	18.1
y	11	41	90	141	169	205	251	318	421	545	592	655

En este caso la ecuación a ajustar tiene tres coeficientes: $a = c_1$, $b = c_2$ y $c = c_3$, siendo las tres funciones: $f_1 = x^2$, $f_2 = 1/x$ y $f_3 = e^{x/8}$.

En consecuencia los datos que se deben mandar a ***MCUAD*** son:

{« SQ » « INV » « 8 / EXP » }

[2.3 4.5 6.7 8.4 9.2 10.1 11.2 12.6 14.5 16.3 17.2 18.1]

[11 41 90 141 169 205 251 318 421 545 592 655]

Y haciendo correr ***MCUAD*** se obtienen:

1.00040090047

[2.08384199105 9.67526431059 -2.5593943924]

Una vez más, el coeficiente de correlación indica que el ajuste es bueno.

La ecuación ajustada es:

$y=2.08384199105*x^2+9.67526431059/x-2.5593943924*e^{x/8}$

7.3.3 Preguntas y Ejercicios

1. ¿Qué se hace cuando se ajuste de datos a una ecuación analítica?
2. ¿Es posible calcular valores intermedios con la ecuación ajustada?
3. ¿Cuál es la forma general de la ecuación a la cual se pueden ajustar los datos con el programa desarrollado en el presente capítulo?
4. ¿La ecuación: $z= x^3+c*e^x-d/x+e/x^3$ puede ser ajustada con el método desarrollado en el presente capítulo (escriba sus equivalencias)?
5. ¿Cuál es la ecuación que define el método de los mínimos cuadrados?
6. En el método desarrollado en el presenta capítulo ¿Cómo se encuentra el mínimo?
7. En el cálculo de los coeficientes de la ecuación ajustada ¿qué es necesario resolver?
8. ¿Para qué se calcula el coeficiente de correlación?
9. Ajuste los siguientes datos a la ecuación: $y = a*x^3 + b*x^{0.5} + c/x^{1/3}$.

x	3.8	1.3	9.2	3.4	1.0	5.1	5.7	9.0	8.0	6.4	6.6	8.1
y	15	8	124	13	8	27	35	116	84	46	50	87

10. Ajuste los siguientes datos a una ecuación de la forma: $y = a*ln(x) + b*sin(x) - c*cos(x)$.

x	1	3	4	5	8	11	14	15	20
y	1	3	2	2	5	4	6	6	7

11. Ajuste los siguientes datos a una ecuación de la forma: $z = a + b*ln(x) + c*ln(x))-d/x+e/x^2$.

x	2.0	1.4	4.0	3.5	7.4	6.2	1.1	3.0	5.5	7.3	2.9	3.2
z	1.71	1.36	4.51	3.44	10.4	6.93	1.03	3.72	4.68	3.94	2.86	3.26

8. MÉTODO DE OPTIMIZACIÓN SIMPLEX

El método *Simplex* es un método que puede ser empleado para resolver una amplia variedad de problemas donde se busque optimizar algo, es decir encontrar un máximo o un mínimo. Puede por ejemplo ser aplicado en el campo de la economía para optimizar las ganancias (es decir maximizar los beneficios) u optimizar los gastos (esto es minimizar los costos).

Nosotros aplicaremos el método Simplex en dos tipos de problemas: encontrar la solución de un sistema de ecuaciones no lineales y ajustar una serie de datos tabulados a una ecuación analítica.

Desde el punto de vista del cálculo diferencial, la determinación del punto mínimo requiere, como se vio en el método de los mínimos cuadrados, el cálculo de derivadas parciales, sin embargo, aun cuando en la mayoría de los casos estos métodos son eficientes, presentan ciertas limitaciones:

- Con frecuencia el cálculo de las derivadas parciales suele ser un proceso tedioso.
- No son métodos estables, es decir, no siempre se logra la convergencia hacia el resultado buscado.
- En algunos casos, en lugar de soluciones óptimas, se encuentran soluciones erróneas debido a que la función, en algunos puntos, puede tener una forma tal que el gradiente se hace igual a cero sin que se haya alcanzado realmente el mínimo.

Una de las alternativas al cálculo diferencial la constituyen los métodos de *búsqueda simple*. En estos métodos se evalúa la función a optimizar (denominada función de error) en varios puntos y en base a los resultados obtenidos se determinan nuevos puntos que se aproximen cada vez más al mínimo.

Existen diferentes métodos que pueden ser empleados para decidir los puntos que deben ser evaluados. Uno de los métodos más sencillos consiste en hacer variar un parámetro a la vez hasta que la función sea mínima con respecto a dicho parámetro. Es posible, igualmente, evaluar una serie de puntos y la sección que contenga un mínimo puede ser reevaluada, tomando en cuenta un mayor número de puntos.

Las principales ventajas de estos métodos son:

a) Estabilidad, pues la convergencia está prácticamente asegurada.

b) No es necesario el cálculo de derivadas parciales.

La principal desventaja es el elevado número de cálculos que es necesario efectuar.

El método S*implex* (que no debe ser confundido con el método de programación lineal Simplex) es uno de los *métodos de búsqueda simple* y es probablemente el método más ampliamente empleado en la resolución de problemas de optimización.

Consideremos que la función a optimizar (función a resolver) es la siguiente:

$$f(x_1, x_2, x_3, ..., x_n) = \alpha \qquad (1)$$

Esta función se resuelve cuando se encuentran valores de las variables x_1 a x_n tales que la igualdad se cumple. Una forma conveniente de conseguirlo es planteando la función como un problema de optimización (minimización):

$$g(x_1, x_2, x_3, ..., x_n) = f(x_1, x_2, x_3, ..., x_n) - \alpha = 0 \qquad (2)$$

Sin embargo esta forma tiene un problema pues la diferencia puede ser positiva o negativa y si es negativa entonces es menor a cero, con lo que se podría concluir erróneamente que se ha encontrado el mínimo.

La forma más sencilla de eliminar el problema del signo consiste en elevar ambos miembros de la ecuación (2) al cuadrado:

$$e(x_1, x_2, x_3, ..., x_n) = \left[g(x_1, x_2, x_3, ..., x_n) \right]^2 = \left[f(x_1, x_2, x_3, ..., x_n) - \alpha \right]^2 = 0 \qquad (3)$$

Cuando la ecuación toma esta forma se conoce como la función de error y es la forma que puede ser optimizada (resuelta) con el método Simplex. Note que cuando se resuelve esta función, se resuelve también la ecuación original, pues los valores de x_1 a x_n que se calculan satisfacen también la igualdad de la ecuación (1).

Es necesario recalcar que lo importante del proceso antes descrito es el llevar el problema original a una forma que siempre devuelve valores positivos.

En el método Simplex los nuevos puntos se encuentran proyectando, en dirección al mínimo, los puntos originales con líneas rectas trazadas a través de dos puntos. Por ello, antes de estudiar el método en si veremos en qué consiste la proyección de puntos n-dimensionales.

8.1 Proyección a través de 2 puntos n-dimensionales

Como se señaló previamente el proceso de proyección en el método Simplex es llevado a cabo con puntos n-dimensionales, sin embargo y debido a su sencillez estudiaremos primero el proceso de proyección en dos dimensiones (en el plano). Para ello nos basaremos la gráfica mostrada en la Figura 9.1.

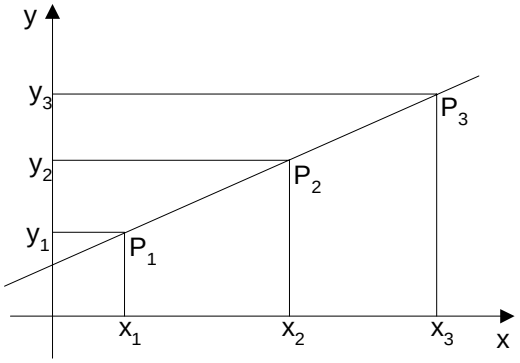


Figura 9.1. Proyección de un punto (P_1) a través de otro (P_2).

En este proceso se conocen los puntos P_1 y P_2 y se quiere calcular el punto P_3 (x_3,y_3). Ello es posible si se conocen las distancias que existen entre los puntos P_1, P_2 y P_1, P_3, es decir si se conoce la relación:

$$k = \frac{\overline{P_1 P_3}}{\overline{P_1 P_2}} \qquad (4)$$

Entonces conociendo k es posible calcular las dimensiones del punto P_3:

$$k = \frac{x_3 - x_1}{x_2 - x_1} \Rightarrow x_3 = x_1 + k(x_2 - x_1) = kx_2 + (1-k)x_1 \qquad (5)$$

$$k = \frac{y_3 - y_1}{y_2 - y_1} \Rightarrow y_3 = y_1 + k(y_2 - y_1) = ky_2 + (1-k)y_1 \qquad (6)$$

Estas relaciones que se cumplen para dos dimensiones *(x, y)*, se cumplen también con *"n"* dimensiones. En general si llamamos a las dimensiones x_1, x_2 hasta x_n, en lugar de *x, y, z, etc.*, para cualquier dimensión x_i se cumple que:

$$x_{3,i} = kx_{2,i} + (1-k)x_{1,i} \tag{7}$$

O en general para cualquier punto con *"n"* dimensiones:

$$P_3 = kP_2 + (1-k)P_1 \tag{8}$$

Donde cada punto (P_1, P_2 y P_3) es un vector con *"n"* elementos (los valores de las *"n"* dimensiones). En forma desarrollada la ecuación (8) es:

$$\begin{bmatrix} x_{3,1} \\ x_{3,2} \\ x_{3,3} \\ \dots \\ x_{3,n} \end{bmatrix} = k \begin{bmatrix} x_{2,1} \\ x_{2,2} \\ x_{23,} \\ \dots \\ x_{2,n} \end{bmatrix} + (1-k) \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ \dots \\ x_{1,n} \end{bmatrix} \tag{8.a}$$

En estas ecuaciones, el primer subíndice identifica el punto y el segundo la variable o dimensión en ese punto.

8.2 Plan inicial

Para comenzar la búsqueda del valor óptimo, el método Simplex requiere de un plan inicial. Si la función a optimizar consta de *n* variables (o dimensiones), se requieren *n+1* puntos para definir un *plan inicial simplex.*

Con los puntos del plan inicial se van generando otros planes con puntos que aproximan la función de error cada vez más a cero (el mínimo).

Consideremos por ejemplo la optimización de una función de error con 2 dimensiones (2 variables). Para optimizar una función con 2 dimensiones *(n=2)* el método *simplex* requiere un plan inicial con 3 puntos *(n+1)*.

Con 3 puntos se forma un triángulo como se puede ver en la Figura 9.2.

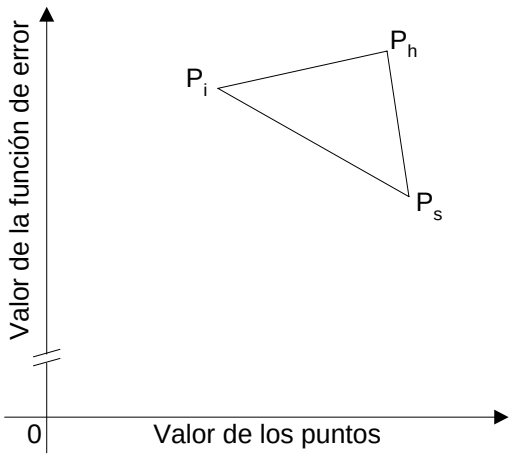


Figura 9.2. Plan inicial Simplex

Entre estos tres puntos del plan inicial, se identifica el punto para el cual la función de error tiene el mayor valor, es decir el punto que está más alejado del óptimo y se denomina al mismo P_h (*h* de highest = el más alto). Igualmente se identifica el punto para el cual la función de error tie-

ne el menor valor, es decir el punto que está más cercano al óptimo y se denomina al mismo P_s (*s* de *smallest = el más pequeño*).

8.3 Centroide

Puesto que el punto que se encuentra más alejado del óptimo es P_h, en el método Simplex se busca un nuevo plan (un conjunto de *n-1* puntos) donde este punto haya sido reemplazado por otro que se aproxime más al óptimo. Con este propósito se calcula el promedio de todos los puntos del plan Simplex, sin incluir en dicho cálculo el punto P_h y a este punto se denomina *centroide*. En consecuencia la expresión para el cálculo del centroide *(C)* es:

$$C = \frac{1}{n}\sum_{i=1}^{n+1} P_i \quad \{\text{Excepto para i} = \text{h} \tag{9}$$

8.4 Reflexión

Una vez calculado el *centroide*, se determina el punto que reemplazará al punto P_h, proyectando el mismo a través del centroide, tal como se muestra en la Figura 9.3.

Como este nuevo punto (P_r) se encuentra prácticamente a la misma distancia del centroide que el punto original, es la imagen *reflejada* del mismo. Esta es la razón por la cual es denomina *reflexión* a este proceso.

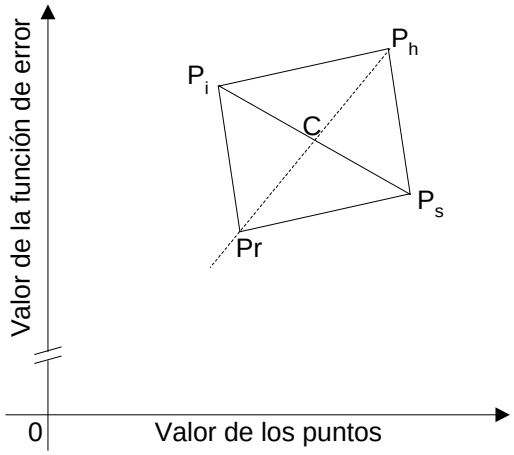


Figura 9.3. Cálculo de un nuevo punto por reflexión

Aplicando la ecuación de proyección (8) a este proceso, (donde $P_1 = P_h$, $P_2 = C$, $P_3 = P_r$ y $k = a$) se tiene:

$$P_r = aC + (1-a)P_h \tag{10}$$

Dado que la distancia entre P_h y P_r es el doble de la distancia entre P_h y C, el valor de *"a"* es igual a 2. En la práctica sin embargo no se toma un valor igual a 2, sino sólo aproximado (normalmente 1.95), pues un valor entero suele en ocasiones dar lugar a un comportamiento oscilatorio que provoca un ciclo infinito.

Si llamamos e_r al valor de la función de error en el punto P_r, e_h al valor de la función de error en el punto P_h y e_s al valor de la función de error en el punto P_s, entonces al efectuar la *reflexión* se pueden presentar los siguientes casos:

a) Si e_r es menor e_s, entonces la reflexión ha sido muy buena, pues se ha conseguido un punto para el cual la función de error es menor aún que el menor valor del plan. Entonces se proyecta el punto al doble de la distancia original proceso conocido como *expansión* (el cual estudiaremos un poco más adelante).

b) Si e_r es menor e_h pero no menor a e_s, entonces la reflexión es aceptable pues se ha conseguido un punto que se aproxima más al óptimo que el punto P_h. En este caso se reemplaza el punto P_h por el punto P_r.

c) Si e_r es mayor a e_h, entonces la reflexión ha sido mala pues se ha conseguido un punto que se aleja más aún del óptimo que el punto P_h. En este caso se lleva a cabo la proyección sólo hasta la mitad del centroide, proceso conocido como *contracción* (el cual estudiaremos también más adelante).

8.5 Expansión

Como se indicó en el acápite anterior, si la reflexión ha sido muy buena *($e_r < e_s$)*, se lleva a cabo la expansión, proceso que consiste en obtener un nuevo punto *(P_x)* reflejando el punto original *(P_h)* al doble de la distancia del punto reflejado *(P_r)* tal como se muestra en la Figura 9.4.

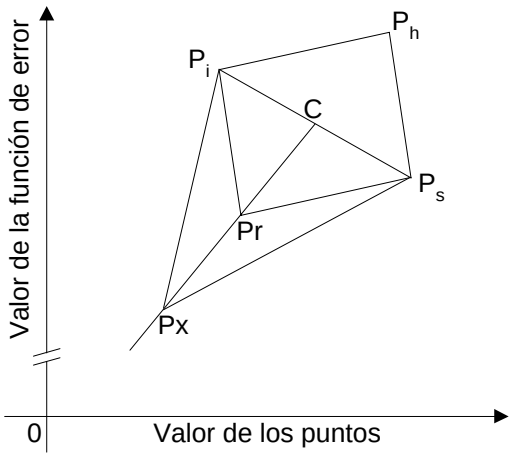


Figura 9.4. Proceso de expansión

Entonces aplicando la ecuación de proyección (ecuación 8) con $C = P_1$, $P_r = P_2$, $P_x = P_3$ y $b = k$ se tiene:

$$P_x = bP_r + (1-b)C \qquad (11)$$

Puesto que P_x está al doble de la distancia de P_r, con relación a C, el valor de b debería ser igual a 2, una vez más no se emplea un número entero, sino un valor aproximado (generalmente 1.95), pues el valor entero suele dar lugar comportamientos oscilatorios y como consecuencia ciclos infinitos.

Si llamamos e_x al valor de la función en el punto P_x y e_x es menor a e_r (el valor de la función en el punto reflejado), entonces el punto expandido *(P_x)* se aproxima más aún al mínimo que el punto reflejado *(P_r)*, por lo tanto se reemplaza el peor punto del plan Simplex *(P_h)*, con el punto expandido P_x. Si por el contrario e_x es mayor a e_r, entonces la expansión no ha sido buena, en ese caso se reemplaza el peor punto del plan Simplex *(P_h)* con el punto reflejado P_r (que ya sabemos es un mejor punto que P_h).

8.6 Contracción

Como ya se señaló en el acápite 8.4, cuando la reflexión proporciona un peor punto que el peor punto del plan Simplex *(P_h)*, entonces se procede con la *contracción*, proceso que consiste en realizar la proyección sólo hasta la mitad de la distancia que existe entre P_h y C (el centroide), tal como se muestra en la Figura 9.5.

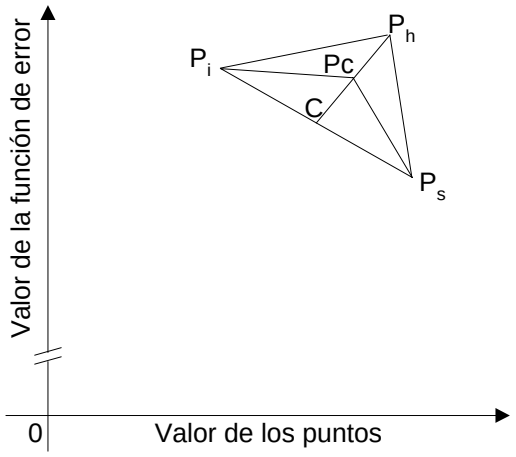


Figura 9.5. Proceso de contracción

Aplicando la ecuación de proyección (ecuación 8) con los puntos $P_h = P_1$, $C = P_2$, $P_c = P_3$ y $d = k$ resulta:

$$P_c = dC + (1-d)P_h \qquad (12)$$

Puesto que el punto proyectado P_c se encuentra a la mitad del punto C (con relación al punto P_h), la constante d es igual a 0.5. Una vez más para evitar comportamientos oscilatorios y ciclos infinitos se emplea un valor cercano a 0.5 (generalmente 0.45).

Si llamamos e_c al valor de la función de error en el punto P_c y si resulta que e_c es menor a e_h la contracción ha sido exitosa pues se consigue un punto que se aproxima más al óptimo que el punto P_h, en ese caso se reemplaza el punto P_h con el punto P_c.

Si por el contrario e_c es mayor a e_h la contracción es mala pues se consigue un punto que se aleja aún más del óptimo que el peor punto existente *(P_h)*, en este caso y dado que ya se ha proyectado el punto a ambos lados del centroide (con la reflexión y contracción), se proyectan todos los puntos del plan Simplex (con excepción del punto *P_s)*, acercando o alejando los puntos del mejor punto existente *(P_s)*, proceso que se conoce como *escalamiento*.

8.7 Escalamiento

Como ya se señaló en el acápite anterior, cuando la reflexión y la contracción no generan un punto que se aproxime más al óptimo (esto es al mínimo), se procede con el *escalamiento*.

El *escalamiento* consiste en proyectar los puntos del plan Simplex, sin incluir el punto P_s, en dirección del mejor punto existente (ver Figura 9.6). Esta proyección puede hacer que los puntos del plan se cerquen hacia el punto P_s o se alejen del mismo.

Si denominamos P^* a los puntos del nuevo plan y aplicamos la ecuación de proyección (ecuación 8) con $P_i = P_1$, $P_s = P_2$ y $P^*_i = P_3$, resulta:

$$P_i^* = kP_s + (1-k)P_i \qquad \begin{cases} i = 1 \rightarrow n+1 \\ \text{Excepto par i=s} \end{cases} \tag{13}$$

Si los nuevos puntos del plan se acercan a la mitad de la distancia que existía entre los antíguos puntos y el punto P_s, entonces el valor de k es igual a 0.5. Una vez más no se toma un valor igual a 0.5 sino aproximadamente igual a 0.5 (normalmente 0.45).

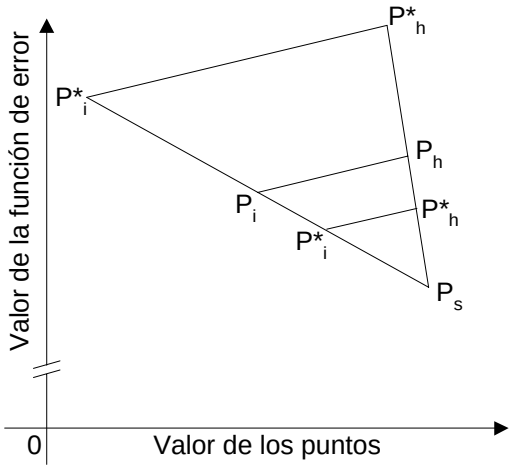


Figura 9.6. Proceso de escalamiento

Por el contrario si los nuevos puntos del plan se alejan al doble de la distancia que existía entre los antiguos puntos y el punto P_s, entonces el valor de k es -1. Una vez más se toma sólo un valor aproximado (normalmente -0.95).

Como el escalamiento cambia todos los puntos del plan (con excepción del punto P_s), es necesario recalcular los valores de la función de error para todos los nuevos puntos del plan.

En el método Simplex, las operaciones de *reflexión* y las consiguientes operaciones de *expansión*, *contracción* y/o *escalamiento* se repiten hasta que se encuentra un valor de la función de error aproximadamente igual a cero (con un margen de error predeterminado) o hasta que los valores de la función de error para los puntos P_h y P_s (e_h y e_s) son aproximadamente iguales.

8.8 Valores iniciales del plan Simplex

Como ya se señaló previamente un plan Simplex está conformado por $n+1$ puntos, siendo n el número de dimensiones o variables. Si por ejemplo se tiene un sistema con 5 dimensiones (o variables), se requieren 6 puntos para el plan inicial Simplex, ello implica asumir los valores de 30 variables (5 variables para cada punto), una labor que por cierto es muy tediosa y que puede desalentar la aplicación de este método.

Por esta razón existen diferentes métodos que generan los puntos del plan inicial Simplex en base a un solo punto asumido. Probablemente el método más sencillo (aunque no precisamente el mejor) sea el que se explica a continuación.

En base a los valores asumidos para un punto (P_1) se generan los otros puntos de la siguiente manera: el segundo punto (P_2) se genera multiplicando el primer elemento del punto asumido por 1.1, el tercer punto (P_3) se genera multiplicando el segundo elemento del punto asumido por 1.1, el cuarto punto (P_4) se genera multiplicando el tercer elemento del punto asumido por 1.1 y

así sucesivamente. Procediendo de esa manera se obtienen fácilmente los *n+1* puntos que conforman un plan inicial Simplex.

Por ejemplo, si el sistema tiene 3 dimensiones (o variables) y se asume el siguiente punto $P_1 = (15,15,150)$, los otros puntos del plan Simplex se obtienen de la siguiente manera:

$P_2 = (15*1.1, 15, 150) = (16.5, 15, 150)$

$P_3 = (15, 15*1.1, 150) = (15, 16.5, 150)$

$P_4 = (15, 15, 150*1.1) = (15, 15, 165)$.

8.9 Programa

El programa ha sido subdividido en subprogramas por dos razones: a) lograr un programa más claro y sencillo y b) evitar la repetición de código.

A continuación explicaremos la lógica de cada subprograma y la forma que se eligió para implementarla.

8.9.1 Plan Inicial Simplex

Como ya se explicó para comenzar el proceso de búsqueda Simplex, se requiere un plan inicial. Dicho plan se genera siguiendo el procedimiento descrito en el acápite 8.8.

Para el subprograma se requiere que el punto inicial asumido, se encuentre en la pila (en el nivel 1), que el número de puntos del plan (esto es el número de dimensiones más uno) esté almacenado en la variable *"n"*. Puesto que en *User RPN* no es posible crear un vector de vectores, el subprograma almacena los puntos del plan inicial en una lista *(P)*.

Los puntos se generan en un ciclo *FOR*, duplicando dos veces el punto asumido (con *DUP DUP*) extrayendo el elemento *"i"* (con *i GET*), multiplicándolo por 1.1 y colocando el resultado nuevamente en el vector (con *i SWAP PUT*). Finalmente el punto asumido se coloca como primer punto (con *n ROLLD*), se forma la lista de los puntos del plan (con *n →LIST*) y se guarda la lista resultante en la variable local *"P"*.

```
« 1 n 1 - FOR i
    DUP DUP i GET 1.1 *
    i SWAP PUT
    SWAP
  NEXT
  n ROLLD n →LIST 'P' STO
»
```

Este subprograma será almacenado en la variable local *"plan1"*.

8.9.2 Valores de la función de error para el plan Simplex

Los valores de las funciones de error deben ser calculados para cada uno de los puntos del plan Simplex.

El subprograma requiere la función de error *(fe)*, los puntos del plan *(P)* y el número de puntos del plan *(n)*.

Los valores de la función de error para cada uno de los puntos se calculan con un ciclo FOR que va desde 1 hasta *n*, donde en cada repetición del ciclo se calcula el valor de la función de error para un punto. Los valores calculados se almacenan finalmente como un vector en la variable local *"e"*:

```
« 1 n FOR i
    P i GET fe EVAL
  NEXT
  n →ARRY 'e' STO
»
```

Este subprograma será almacenado en la variable local *"vecte"*.

8.9.3 Índices del mejor y peor punto

Como ya se explicó anteriormente, en el método Simplex es necesario ubicar el punto con el menor valor de la función de error (el mejor punto: *s*) y el punto con el mayor valor de la función de error (el peor punto: *h*).

Para el subprograma se requiere conocer el vector con los valores de la función de error *(e)* y el número de puntos del plan *(n)*. El subprograma almacena los índices del peor punto en la variable *"h"* y el del mejor punto en la variable *"s"*.

Los índices se encuentran inicializando primero las variables *"h"* y *"s"* en 1 y recorriendo el vector *"e"* con un ciclo FOR que comienza desde el segundo elemento. En este ciclo se compara el valor de cada uno de los elementos del vector *"e"* con e_h y e_s. Si dicho elemento es mayor a e_h, entonces *"h"* toma el valor del índice *(i)*, igualmente si dicho elemento es menor a e_s, *"s"* toma el valor del índice *(i)*:

```
« 1 'h' STO 1 's' STO
  2 n FOR i
    e i GET DUP
    IF e h GET > THEN i 'h' STO END
    IF e s GET < THEN i 's' STO END
  NEXT
»
```

Este subprograma será almacenado en la variable local *"hys"*.

8.9.4 Centroide

El centroide, es simplemente el promedio de los puntos del plan Simplex (sin incluir el peor punto).

El subprograma que calcula el centroide requiere los puntos del plan simplex *(P)*, el número de puntos del plan *(n)* y el índice del peor punto *(h)*. Almacena el centroide calculado en la variable local *"C"*.

El promedio se calcula con dos ciclos FOR donde en el primero se extraen los puntos del plan con excepción del peor punto *(h)*. En el segundo se ciclo FOR se suman los puntos y se divide la suma entre el número de puntos menos 1:

```
« 1 n FOR i
    IF i h ≠ THEN P i GET END
  NEXT
  1 n 1 - FOR i + NEXT
  n 1 - / 'C' STO
»
```

Este subprograma será almacenado en la variable local *"centr"*.

8.9.5 Escalamiento

El escalamiento, que se realiza cuando la reflexión y contracción no consiguen un mejor plan Simplex, básicamente consiste en aplicar la ecuación (13) seguida de un recálculo de los valores de las funciones, con excepción del mejor punto y una nueva búsqueda del peor y mejor punto del nuevo plan.

El subprograma requiere los puntos del plan Simplex *(P)*, el número de puntos del plan *(n)*, el índice del mejor punto *(s)*, la constante de escalamiento *(k)*, la función de error *(fe)* y el subprograma *"hys"*.

Los nuevos puntos del plan y los valores de la función de error para estos puntos se calculan en un ciclo *FOR*. Los nuevos puntos se vuelven a almacenar en la lista *"P"* (*P i … PUT*) y los nuevos valores de la función de error en el vector *"e"* (e i … PUT) Para calcular los valores de la función de error, se hace una copia del nuevo punto calculado (con *3 PICK*). Finalmente los índices del peor y mejor punto se consiguen llamando al subprograma *"hys"*.

```
« 1 n FOR i
    IF i s ≠ THEN
      P i
      k P s GET * 1 k - P i GET * +
      e i 3 PICK fe EVAL PUT 'e' STO
      PUT 'P' STO
    END
  NEXT
  hys
»
```

Este programa se almacenará en la variable local *"escal"*.

8.9.6 Programa principal

El programa que lleva a cabo el proceso de búsqueda Simplex recibe como datos la función de error *(fe)*, el punto asumido *(Pa)* y el error permitido *(err)*. Devuelve como resultado el punto para el cual el valor de la función de error es mínimo.

Los valores de las constantes de reflexión *(a)*, de expansión *(b)*, de contracción *(d)* y de escalamiento *(k)* se fijan en 1.98, 1.98, 0.48 y 0.48 respectivamente.

Los puntos del plan inicial simplex se calculan llamando al subprograma *plan1* y los valores de la función de error llamando al subprograma *vecte*.

El proceso de búsqueda se lleva a cabo en un ciclo que se repite hasta que el valor de la función de error es menor o igual al error permitido *(err)* o hasta que los valores de la función de error para el peor punto *(e_h)* y el mejor punto *(e_s)* son iguales en el número de dígitos establecido en el error permitido *(err)*.

En el ciclo se determinan los índices del peor y mejor punto (con el subprograma *hys*), se calcula el centroide (con el subprograma *centr*) y se procede a efectuar la reflexión (ecuación 10).

Si el valor de la función de error *(er)* en el punto reflejado *(Pr)* es menor al menor valor existente *(e_s)*, se procede con la expansión (ecuación 11), entonces, si el valor de la función de error *(ex)* en el punto expandido *(Px)* es menor a *"er"* se emplea el punto expandido (Px) en lugar de P_h en el nuevo plan Simplex, caso contrario se emplea el punto reflejado *(Pr)* en lugar de P_h.

Si el valor de la función de error *(er)* en el punto reflejado es menor a e_h, pero no a e_s, entonces se emplea el punto reflejado en lugar de P_h en el nuevo plan Simplex.

Si el valor de la función de error en el punto reflejado (e_r) es mayor a e_h, entonces se procede con la contracción (ecuación 12). Si el valor de la función de error *(ec)* en el punto contraído *(Pc)* es menor a e_h, se emplea el punto contraído *(Pc)* en lugar de P_h, caso contrario se procede con el escalamiento (subprograma *escal*).

Dado que el método Simplex puede demorar bastante en encontrar el mínimo, en el programa se muestra en la línea 4 el mejor valor de la función de error (e_s) de cada iteración (CON "fe: " 3 DISP e s GET 4 DISP). Para que al mostrar estos valores no aparezcan basuras en la pantalla, al principio del programa se limpia la pantalla (con CLLCD). El listado del programa elaborado es el siguiente:

```
« 1.98 1.98 0.48 0.48
  1 17 START 0 NEXT
  → fe Pa err a b d k
    plan1 vecte hys centr escal
    n P C h s Pr e er Px ex Pc ec
  « CLLCD
    « 1 n 1 - FOR i
        DUP DUP i GET 1.1 *
        i SWAP PUT
        SWAP
      NEXT
      n ROLLD n →LIST 'P' STO
    » 'plan1' STO
    « 1 n FOR i
        P i GET fe EVAL
      NEXT
      n →ARRY 'e' STO
    » 'vecte' STO
    « 1 'h' STO 1 's' STO
      2 n FOR i
        e i GET DUP
        IF e h GET > THEN i 'h' STO END
        IF e s GET < THEN i 's' STO END
      NEXT
    » 'hys' STO
    « 1 n FOR i
        IF i h ≠ THEN P i GET END
      NEXT
      1 n 2 - FOR i + NEXT
      n 1 - / 'C' STO
    » 'centr' STO
    « 1 n FOR i
        IF i s ≠ THEN
          P i
          k P s GET * 1 k - P i GET * +
          e i 3 PICK fe EVAL PUT 'e' STO
          PUT 'P' STO
        END
      NEXT
      Hys EVAL
    » 'escal' STO
    Pa SIZE LIST→ + 'n' STO
    Pa plan1 EVAL vecte EVAL
```

```
    DO
      hys EVAL centr EVAL
      "fe: " 3 DISP
      e s GET 4 DISP
      a C * 1 a - P h GET * + DUP 'Pr' STO
      fe EVAL 'er' STO
      IF er e s GET < THEN
        b Pr * 1 b - C * + DUP 'Px' STO
        fe EVAL 'ex' STO
        P h e h
        IF ex er < THEN ex Px ELSE er Pr END
        4 ROLLD PUT 'e' STO PUT 'P' STO
      ELSE
        IF er e h GET < THEN
          P h Pr PUT 'P' STO e h er PUT 'e' STO
        ELSE
          d C * 1 d - P h GET * + DUP 'Pc' STO
          fe EVAL 'ec' STO
          IF ec e h GET < THEN
            P h Pc PUT 'P' STO e h ec PUT 'e' STO
          ELSE
            Escal EVAL
          END
        END
      END
    UNTIL
      e s GET err <
      e s GET e h GET / 1 - ABS err < OR
    END
    P s GET
  »
»
```

En lo sucesivo se asumirá que este programa ha sido almacenado en la variable global *"SIMPL".*

8.10 Ejemplos

Ejemplo 1

Como primer ejemplo resolveremos el siguiente sistema de ecuaciones no lineales empleando el método Simplex:

$$x^2 + 2y^2 = 22$$
$$-2x^2 + xy - 3y = -11$$

Lo primero que debemos hacer es colocar este sistema de ecuaciones en forma de una función de error, es decir una función igualada a cero que devuelva sólo valores positivos. Para ello igualamos las dos ecuaciones a cero, las elevamos al cuadrado y la función es la suma de dichos cuadrados:

$$fe = \left(x^2 + 2y^2 - 22\right)^2 + \left(-2x^2 + xy - 3y + 11\right)^2 = 0 + 0 = 0$$

En este caso asumiremos un punto inicial igual a (1.5, 2) (valores asumidos para "x" y "y") y el error permitido será 1E-4.

El programa elaborado para resolver este sistema con un error permitido de 1E-6 es:

```
« « V→ → x y
```

```
    « x SQ 2 y SQ * + 22 - SQ
      -2 x SQ * x y * + 3 y * - 11 + SQ +
    »
  »  [1.5 2] 0.000001 SIMPL
  AXL {x y } →TAG LIST→ DROP
»
```

Haciendo correr este programa se obtienen los siguientes resultados después de aproximadamente 1 minuto y 36 segundos (en una calculadora HP49G):

x: 1.99992407633

y: 3.00006405231

Que son las soluciones del sistema de ecuaciones no lineales (la solución exacta es x = 2; y=3).

Ejemplo 2

Como segundo ejemplo ajustaremos los siguientes datos a la ecuación: $y = a + bx^2 + cx^4 + d\ln(x)$ y emplearemos la ecuación ajustada para calcular valores de "y" para valores de "x" iguales a 1, 2, 3, 4 y 5.

X	4.0	3.2	6.8	9.3	1.2	0.7	5.5	7.4	6.5	2.4
Y	6.2	5.3	9.1	12.0	3.3	1.5	7.9	10.6	8.7	6.3

Como siempre primero colocamos nuestro problema en forma de una función de error (una función igualada a cero y que siempre devuelve valores positivos). Para ello empleamos la ecuación de definición de los mínimos cuadrados:

$$\sum_{i=1}^{n} r_i^2 = \text{mínimo}$$

Que adaptada a nuestro caso sería:

$$\sum_{i=1}^{n} \left(a + bx^2 + cx^4 + d\ln(x) - y\right)^2 = 0$$

Donde "n" es el número de datos existentes (en nuestro caso 10). Las variables (o dimensiones) son: *a, b, c y d*. En consecuencia debemos asumir un punto inicial con cuatro valores. Dado que el segundo y tercer término están son potencias de *"x"*, asumiremos los siguientes valores (1, 0.02, 0.0003, 4).

La función de error se programará empleando un ciclo FOR (que va desde 1 hasta 10), donde en cada repetición del ciclo calcularemos el valor de la función para cada par de datos "x" - "y":

```
« [4 3.2 6.8 9.3 1.2 0.7 5.5 7.4 6.5 2.4]
  [6.2 5.3 9.1 12.0 3.3 1.5 7.9 10.6 8.7 6.3]
  0 0 0 0
  → vx vy a b c d
  « « V→ 0 0 → a b c d x y
      « 0
        1 10 FOR i
          vx i GET 'x' STO vy i GET 'y' STO
          a b x SQ * + c x 4 ^ * + d x LN * + y - SQ +
        NEXT
      »
    »  [1 0.02 0.0003 4 ] 0.000001 SIMPL
    AXL {a b c d } DUP2 STO
```

```
    →TAG LIST→ DROP
    1 5 FOR x
      a b x SQ * + c x 4 ^ * + d x LN * +
    NEXT
    5  →ARRY "y" →TAG
  »
»
```

Haciendo correr el programa se obtienen los resultados:

a: 2.69000061793

b: 2.26596354419E-2

c: 2.34754445714E-4

d: 2.592985265

y: [2.71289500782 4.5817156565 5.76163792348 6.70729277444 7.57646182441]

Que son los coeficientes de la ecuación ajustada y los valores de "y" calculados con la ecuación ajustada.

Una ventaja importante del método de optimización Simplex, con relación al método estudiado en el capítulo de mínimos cuadrados, es que nos permite ajustar no sólo dos series de datos (los vectores "x" y "y" en el ejemplo), sino tres o más series de datos permitiendo realizar entonces un ajuste multivariable.

Las principales desventajas son la necesidad de asumir un punto inicial y principalmente el tiempo que demora en encontrar la solución debido al elevado número de evaluaciones funcionales que se realizan. Así este ejemplo encuentra el resultado en aproximadamente 8 minutos y 37 segundos (en una calculadora HP49G).

8.11 Preguntas y Ejercicios

1. ¿En qué consiste optimizar una determinada función?
2. ¿Cómo se encuentra el óptimo en los métodos de búsqueda simple?
3. ¿Cuáles son las principales ventajas de los métodos de búsqueda simple?
4. ¿Cuál es la principal desventaja de los métodos de búsqueda simple?
5. Para optimizar una función con el método Simplex ¿En qué forma debe ser colocada la misma?
6. ¿Para qué se emplea la proyección en el método Simplex?
7. ¿Qué se debe conocer para proyectar un punto a través de otro?
8. ¿Cuál es la ecuación general que nos permite proyectar un punto a través de otros dos?
9. ¿Cuántos puntos se requieren para formar un plan inicial Simplex?
10. En el plan inicial Simplex ¿Qué puntos se identifican?
11. ¿Qué es el centroide?
12. ¿En qué consiste la reflexión?
13. ¿En qué consiste la expansión?
14. ¿En qué consiste la contracción?
15. ¿En qué consiste el escalamiento?

16. ¿Cuándo se lleva a cabo el escalamiento?

17. ¿Cómo se genera el plan inicial Simplex?

18. Elabore un programa que ajuste los datos de la tabla a la ecuación: $y = a*x^2 + b/x + c*e^{x/8}$ y emplee la ecuación ajustada para calcular valores de "y" para valores de "x" iguales a 1, 2, 10, 12, 14, 16 y 18.

x	2.3	4.5	6.7	8.4	9.2	10.1	11.2	12.6	14.5	16.3	17.2	18.1
y	11	41	90	141	169	205	251	318	421	545	592	655

19. Elabore un programa que ajuste los datos de la tabla a la ecuación: $y = a*x^3 + b*x^{0.5} + c/x^{1/3}$ y resuelva la ecuación ajustada empleando el método incremental con 8 dígitos de exactitud.

x	3.8	1.3	9.2	3.4	1.0	5.1	5.7	9.0	8.0	6.4	6.6	8.1
y	15	8	124	13	8	27	35	116	84	46	50	87

20. Elabore un programa que ajuste los datos de la tabla a una ecuación de la forma: $y = a*ln(x) + b*sin(x) - c*cos(x)$ y resuelva la ecuación ajustada empleando el programa ROOT.

X	1	3	4	5	8	11	14	15	20
y	1	3	2	2	5	4	6	6	7

21. Elabore un programa que ajuste los datos de la tabla a una ecuación de la forma: z = (a+b*ln(x)+c*ln(y))/(d/x+e/y) y emplee la ecuación ajustada para calcular valores de "y" para valores de "x" iguales a 1, 2.2, 3.3, 4.4, 5.6, 7.7 y 8.8.

x	2.0	1.4	4.0	3.5	7.4	6.2	1.1	3.0	5.5	7.3	2.9	3.2
y	2.3	4.4	3.2	2.5	5.2	3.3	6.1	4.2	2.1	1.3	2.6	2.7
z	1.71	1.36	4.51	3.44	10.4	6.93	1.03	3.72	4.68	3.94	2.86	3.26

22. Elabore un programa que resuelva el siguiente sistema de ecuaciones no lineales empleando el método Simplex.

$$4 - x^2 + y^2 = 0$$
$$1 - e^x - y = 8$$

23. Elabore un programa que resuelva el siguiente sistema de ecuaciones no lineales empleando el método Simplex.

$$x^2 + y^2 + z^2 = 9$$
$$xyz = 1$$
$$z + y - z^2 = 80$$

24. Elabore un programa que resuelva el siguiente sistema de ecuaciones no lineales empleando el método Simplex.

$$
\begin{aligned}
x_1 + x_4 &= 3 \\
2x_1 + x_2 + x_4 + x_7 + x_8 + x_9 + 2x_{10} &= 10 + r \\
x_2 + 2x_5 + x_6 + x_7 &= 8 \\
2x_3 + x_5 &= 4r \\
x_1 x_5 &= a_1 x_2 x_4 \\
x_6 x_2^{1/2} &= a_2 (x_2 x_4 x_{11})^{1/2} \\
x_7 x_4^{1/2} &= a_3 (x_1 x_4 x_{11})^{1/2} \\
x_8 x_4 &= a_4 x_2 x_{11} \\
x_9 x_4 &= a5x1(x_3 x_{11})^{1/2} \\
x_{10} x_4^{2} &= a_6 x_4^{2} x_{11} \\
x_{11} &= x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \\
a_1 &= 0.193; \quad a_2 = 0.002597; \quad a_3 = 0.003448; \\
a_4 &= 0.00001799; \quad a_5 = 0.002155; \quad a_6 = 0.00004836; \\
r &= 4.056734
\end{aligned}
$$

9. INTERPOLACIÓN SEGMENTARIA

Cuando los datos experimentales con los que se cuenta tienen un comportamiento irregular, los métodos de interpolación polinomial, como Lagrange y Newton suelen predecir un comportamiento oscilatorio entre los puntos, razón por la cual devuelven resultados erróneos en la interpolación. En estos casos resulta más adecuado ajustar a una ecuación cada uno de los segmentos. Esto es justamente lo que hacen los métodos de interpolación segmentaria (o spline), en estos métodos se emplea una sola forma de ecuación (generalmente un polinomio) para predecir el comportamiento de los datos, pero se calculan nuevos coeficientes para cada uno de los segmentos.

La principal limitante de estos métodos, al igual que en los métodos de antes estudiados, es la confiabilidad de los datos originales.

Tal como sucedió con los métodos de Lagrange y Newton, el cálculo de los coeficientes y la interpolación se harán en programas separados. Esto permite reducir considerablemente el tiempo cuando se interpolan varios valores de una misma tabla, como esta es la situación más frecuente en la práctica, es de utilidad contar con los programas por separado.

En general, todos los programas de interpolación segmentaria ubican primero el segmento en el que se encuentra el valor a interpolar y entonces interpolan el valor empleando los coeficientes de dicho segmento.

De las muchas formas de ecuación que se pueden emplear en la interpolación segmentaria nosotros estudiaremos 2: Lineal y Cúbica.

9.1 INTERPOLACIÓN SEGMENTARIA LINEAL (SPLINE LINEAL)

Es básicamente el método de interpolación lineal, excepto que se calculan por separado los coeficientes de los n-1 segmentos. La ecuación para cada segmento es:

$$y = s_j(x) = a_j+b_j(x-x_j) \qquad \{j=1 \rightarrow n-1 \tag{1}$$

Donde n es el número de datos (x-y) con los que se cuenta. Puesto que en la interpolación segmentaria se tiene una ecuación para cada par de puntos y cada ecuación pasa exactamente a través de dos puntos, para el punto x_{j+1}, y_{j+1}, las ecuaciones que se encuentra a ambos lados de dicho punto ($s_j(x)$ y $s_{j+1}(x)$) deben devolver el mismo resultado, es decir:

$$s_j(x_{j+1}) = s_{j+1}(x_{j+1}) = y_{j+1}$$

$$a_j+b_j(x_{j+1}-x_j) = a_{j+1}+b_{j+1}(x_{j+1}-x_{j+1}) = y_{j+1}$$

$$a_j+b_j(x_{j+1}-x_j) = a_{j+1} = y_{j+1} \tag{2}$$

en consecuencia:

$$a_j = y_j \qquad \{j=1 \rightarrow n \tag{3}$$

Denominando hj a la diferencia:

$$h_j = x_{j+1}-x_j \tag{4}$$

la ecuación (2) puede ser re escrita como:

$$a_j+b_jh_j = a_{j+1} \qquad \{j= 1 \rightarrow n-1 \tag{5}$$

de donde podemos despejar b_j:

$$b_j = (a_{j+1}-a_j)/h_j \quad \{j=1 \rightarrow n-1 \tag{6}$$

Con la ecuación 6 y tomando en cuenta la ecuación 3 podemos calcular todos los coeficientes b_j que se requieren para la interpolación segmentaria lineal.

9.1.1 Diagramas de flujo

El algoritmo elaborado se presenta en los diagramas de flujo de las Figuras 1 y 2.

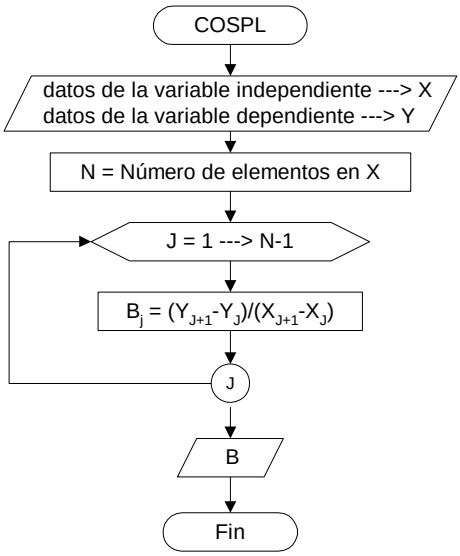


Figura 1. Cálculo de los coeficientes para el método de interpolación segmentaria lineal (spline lineal)

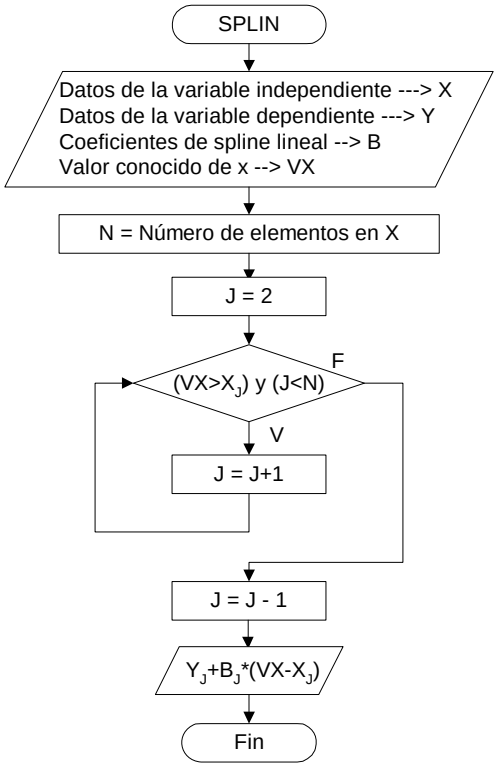


Figura 2. Interpolación segmentaria lineal (spline lineal)

Los programas elaborados en base a estos diagramas de flujo son los siguientes:

Para el cálculo de los coeficientes:

```
« 0. → X Y N
  « X SIZE 1. - DUP 1. GET 'N' STO
    0. CON
    1. N FOR J
      J
      Y J 1. + GET Y J GET -
      X J 1. + GET X J GET - /
      PUT
    NEXT
  »
»
```

En lo sucesivo se asume que este programa ha sido guardado con el nombre COSPL. Para la interpolación:

```
« 0 2   X Y B VX N J
  « X SIZE 1 GET 'N' STO
    WHILE VX X J GET > J N < AND
    REPEAT
      1 'J' STO+
    END
    'J' 1 STO-
    Y J GET B J GET VX X J GET - * +
  »
»
```

En lo sucesivo se asume que este programa ha sido guardado con el nombre SPLIN.

9.1.2 Ejemplos

1. Empleando los datos que se presentan en la siguiente tabla elabore un programa que interpole los valores de "y" para valores de "x" iguales a 1, 4, 6, 13, 17, 23, 25.5, 26.5, 28, 30, 42 y 45.

x	3	5	7	11	14	16	19	21	24	25	26	27	29	33	36	40	44
y	0	0	0	0	1	3	13	22	29	26	18	8	3	0	0	0	0

El programa elaborado para resolver este ejemplo es el siguiente:

```
« 0. 0. 0. → XE YE VX BE
  «
   [ 3. 5. 7. 11. 14. 16. 19. 21. 24. 25. 26. 27.
   29. 33. 36. 40. 44. ] 'XE' STO
   [ 0. 0. 0. 0. 1. 3. 13. 22. 29. 26. 18. 8. 3.
   0. 0. 0. 0. ] 'YE' STO
   [ 1. 4. 6. 13. 17. 23. 25.5 26.5 28. 30. 42.
   45. ] 'VX' STO
   XE YE COSPL 'BE' STO
   1. 12. FOR I XE YE BE VX I GET SPLIN NEXT
   VX SIZE 1 GET →ARRY "Y" →TAG
  »
»
```

Haciendo correr el programa se obtiene: Y: [0. 0. 0. .666666666666 6.33333333333 26.6666666667 22. 13. 5.5 2.25 0. 0.] que son los valores de "y" requeridos.

9.1.3 Ejercicios y preguntas para clases

1. ¿Cuándo resulta más conveniente emplear un método de interpolación segmentaria?

2. En el Spline lineal, ¿Cuál es la ecuación que se emplea para interpolar en cada uno de los segmentos?
3. En el algoritmo de interpolación segmentaria lineal ¿Cuál es el propósito del ciclo While?
4. Empleando los datos de la siguiente tabla calcule (interpole) los valores de "y" para "x" igual a 0.5, 2.5, 3.2, 4.3, 5.5, 6.5, 8.5, 10.5, 13.5, 15.5, 18.5, 20.5, 22.5 y 24.

x	1	2	3	3.5	4	4.5	5	6	7	8	9	10	12	13	14	15	16	17	18	19	20	21	22	23
y	0	0	0.5	1.5	3	5	9	11.8	12	12.5	12.5	12.5	12.5	12.4	12.0	11.5	10.5	9.5	8.3	7.0	5.5	4.0	3.0	2.5

5. Empleando Los datos de la tabla del ejercicio 1 elabore un programa que calcule los valores de "x" para valores de "y" iguales a 1, 2, 4, 6, 8, 10, 11 y 12.

9.2 INTERPOLACIÓN SEGMENTARIA CÚBICA (SPLINE CÚBICO)

En el spline cúbico la ecuación de cada segmento tiene la forma:

$$y = s_j(x) = a_j+b_j(x-x_j)+c_j(x-x_j)^2 +d_j(x-x_j)^3 \quad \{j=1 \rightarrow n-1 \qquad (1)$$

Al igual que en spline lineal, las ecuaciones a ambos lados de un punto deben devolver el mismo resultados, es decir:

$$s_j(x_{j+1}) = s_{j+1}(x_{j+1}) = y_{j+1}$$

$$a_j+b_j(x_{j+1}-x_j)+c_j(x_{j+1}-x_j)^2+d_j(x_{j+1}-x_j)^3 =$$

$$a_{j+1}+b_{j+1}(x_{j+1}-x_{j+1})+c_{j+1}(x_{j+1}-x_{j+1})^2+d_j(x_{j+1}-x_{j+1})^3 = y_{j+1}$$

$$a_j+b_jh_j+c_jh_j^2+d_jh_j^3 = a_{j+1} = y_{j+1} \quad \{j=1 \rightarrow n-1 \qquad (2)$$

(donde al igual que en spline lineal: $h_j=x_{j+1}-x_j$). Entonces:

$$a_j = y_j \quad \{j=1 \rightarrow n \qquad (3)$$

Para que las curvas a ambos lados de un punto tengan la misma pendiente en el punto y sean continuas, se igualan las derivadas primera y segunda de las ecuaciones a ambos lados del punto:

$$s'_j(x_{j+1}) = s'_{j+1}(x_{j+1})$$

$$b_j+2c_jh_j+3d_jh_j^2= b_{j+1}+2c_{j+1}(x_{j+1}-x_{j+1})+3d_{j+1}(x_{j+1}-x_{j+1})^2$$

$$b_j+2c_jh_j+3d_jh_j^2= b_{j+1} \quad \{j=1 \rightarrow n-1 \qquad (4)$$

y

$$s''_j(x_{j+1}) = s''_{j+1}(x_{j+1})$$

$$2c_j+6d_jh_j= 2c_{j+1}+6d_{j+1}(x_{j+1}-x_{j+1})$$

$$c_j+3d_jh_j= c_{j+1} \quad \{j=1 \rightarrow n-1 \qquad (5)$$

Despejando d_j de esta ecuación:

$$d_j = (c_{j+1}-c_j)/(3h_j) \quad \{j=1 \rightarrow n-1 \qquad (6)$$

y reemplazando esta ecuación en la ecuación 4:

$$b_j+2c_jh_j+3(c_{j+1}-c_j)/(3h_j)h_j^2= b_{j+1}$$

$$b_j+2c_jh_j+c_{j+1}h_j-c_jh_j = b_{j+1}$$

$$b_j+c_jh_j+c_{j+1}h_j = b_{j+1}$$

$$b_j+(c_j+c_{j+1})h_j = b_{j+1} \qquad (7)$$

Reemplazando la ecuación 6 en la ecuación 2:

$$a_j+b_jh_j+c_jh_j^2+(c_{j+1}-c_j)/(3h_j)h_j^3 = a_{j+1}$$

$$a_j+b_jh_j+(3c_jh_j^2+c_{j+1}h_j^2-c_jh_j^2)/3 = a_{j+1}$$

$$b_j = (a_{j+1}-a_j)/h_j-(2c_jh_j+c_{j+1}h_j)/3$$

$$b_j = (a_{j+1}-a_j)/h_j-(2c_j+c_{j+1})h_j/3 \quad \{j=1\rightarrow n-1 \qquad (8)$$

Reemplazando 8 en 7 y reajustando índices:

$$(a_{j+1}-a_j)/h_j-(2c_j+c_{j+1})h_j/3+(c_j+c_{j+1})h_j=(a_{j+2}-a_{j+1})/h_{j+1}-(2c_{j+1}+c_{j+2})h_{j+1}/3$$

$$(-2c_jh_j-c_{j+1}h_j+3c_jh_j+3c_{j+1}h_j)/3+(2c_{j+1}+c_{j+2})h_{j+1}/3 =$$

$$3((a_{j+2}-a_{j+1})/h_{j+1}-(a_{j+1}-a_j)/h_j)$$

$$(c_jh_j+2c_{j+1}h_j+2c_{j+1}h_{j+1}+c_{j+2}h_{j+1}=3((a_{j+2}-a_{j+1})/h_{j+1}-(a_{j+1}-a_j)/h_j)$$

$$(h_jc_j+2(h_j+h_{j+1})c_{j+1}+h_{j+1}c_{j+2}=3((a_{j+2}-a_{j+1})/h_{j+1}-(a_{j+1}-a_j)/h_j) \quad \{j=1\rightarrow n-2 \qquad (9)$$

La ecuación (9) constituye un sistema de n-2 ecuaciones con n incógnitas (los valores de "c"), por consiguiente se requieren de 2 valores conocidos de "c" para resolver el sistema. Como no se tiene ninguna información con relación al comportamiento de los datos antes del primer o después del último punto, podemos asumir que la función en el segmento 1 tiene la forma: $s_1(x) = a_1+b_1(x-x_1)+d_1(x-x_1)^3$ y en el segmento "n" (en realidad inexistente) tiene la misma forma: $s_n(x) = a_n+b_n(x-x_n)+d_n(x-x_n)^3$, o lo que es lo mismo asumimos que los valor de c_1 y c_n son cero.

Con esta asunción el sistema de ecuaciones (9) puede ser resuelto.

Para comprender mejor la solución de este sistema escribamos el mismo en forma matricial:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \dots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ p_2 \\ p_3 \\ p_4 \\ \dots \\ p_{n-1} \\ 0 \end{bmatrix} \qquad (9.a)$$

Donde la primera y última fila corresponde a los valores asumidos (c_1 y c_n). Los valores de "p" es la simbología elegida para representar el lado derecho de la ecuación (9), es decir:

$$p_j = 3((a_{j+1}-a_j)/h_j-(a_j-a_{j-1})/h_{j-1}) \quad \{j=1\rightarrow n-1 \qquad (10)$$

La matriz aumentada del sistema de ecuaciones 9 es la siguiente:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & \dots & 0 & 0 & 0 & p_2 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 & \dots & 0 & 0 & 0 & p_3 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 & \dots & 0 & 0 & 0 & p_4 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & p_{n-1} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \end{bmatrix} \qquad (11)$$

Para resolver el sistema de ecuaciones lineales, por un método de eliminación como el de Gauss, debemos llevar la matriz aumentada (11) a la forma:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & r_2 & 0 & 0 & \dots & 0 & 0 & 0 & s_2 \\ 0 & 0 & 1 & r_3 & 0 & \dots & 0 & 0 & 0 & s_3 \\ 0 & 0 & 0 & 1 & r_4 & \dots & 0 & 0 & 0 & s_4 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & r_{n-1} & s_{n-1} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \end{bmatrix} \qquad (12)$$

Donde r_j y s_j es la simbología elegida para representar los nuevos valores de los elementos de la matriz una vez llevada a la forma 11. Con la matriz aumentada en la forma 12, podemos calcular las incógnitas (los valores de c_2 a c_{n-1}) por sustitución hacia atrás.

Para llevar la matriz aumentada 11 la forma 12 se deben efectuar las siguientes operaciones: Los elementos de la primera columna se convierten en cero simplemente restando a la segunda fila la primera fila multiplicada por h_1, puesto que todos los otros elementos de la primera fila son cero, esta operación no modifica ninguno de los otros elementos de la segunda fila. Posteriormente para convertir en uno el segundo elemento de la segunda fila, dividimos esta fila entre $2(h_1+h_2)$, llamando q_2 ha esta expresión las operaciones que se realizan son:

$q_2 = 2(h_1+h_2)$

$r_2 = h_2/q_2$

$s_2 = p_2/h_2$

Con estas operaciones la segunda fila queda como en la ecuación 12.

En la tercera fila debemos eliminar el elemento h_2, esto se consigue restando a la tercera fila la segunda fila multiplicada por $-h_2$, con ello cambia los valores del tercer y último elemento de la tercera fila:

$q_3 = 2(h_2+h_3)-h_2r_2$

$p'_3 = p_3-h_2s_2$

Ahora convertimos en uno el tercer elemento de la tercera fila, para ello dividimos la fila entre q_3. Con esta operación cambian los valores del cuarto y último elemento de esta fila:

$r_3=h_3/q_3$

$s_3 = p'_3/q_3 = (p_3-h_2s_2)/q_3$

Con lo que la tercera fila queda como en la ecuación 12.

Procedemos de manera similar con la cuarta fila. Para eliminar el tercer elemento restamos a la cuarta fila la tercera fila multiplicada por $-h_3$, con ello cambian los valores del cuarto y último elementos de la siguiente manera:

$q_4 = 2(h_3+h_4)-h_4r_3$

$p'_4 = p_4-h_3s_3$

Ahora convertimos en uno el cuarto elemento de la cuarta fila dividiendo esta fila entre q_4, con ello cambian los valores del quinto y último elemento de esta fila:

$r_4 = h_4/q_4$

$s_4 = p'_4/q_4 = (p_4-h_3s_3)/q_4$

Con las anteriores ecuaciones se pueden deducir las relaciones generales para llevar la matriz aumentada de la forma 11 a la forma 12:

$$q_j = 2(h_{j-1}+h_j)-h_{j-1}r_{j-1} \quad \{j = 2 \rightarrow n-1 \tag{13}$$

$$r_j = h_j/q_j \quad \{j = 2 \rightarrow n-1 \tag{14}$$

$$s_j = (p_j-h_{j-1}s_{j-1})/q_j \quad \{j = 2 \rightarrow n-1 \tag{15}$$

Como se puede observar en la matriz aumentada 12, r_1 y s_1 son iguales a cero. Por otra parte "p" y "q" se emplean sólo como variables intermedias para calcular los valores de "r" y "s", por lo que no es necesario almacenarlos como vectores.

Estando la matriz aumentada en la forma 12, los valores de "c" pueden ser calculado por sustitución hacia atrás:

$$c_j = s_j-r_jc_{j+1} \quad \{j = n-1 \rightarrow 2 \tag{16}$$

Con los valores de "c" calculados se calculan los valores de "d" empleando la ecuación 6 y los valores de "b" empleando la ecuación 8.

9.2.1 Diagrama de flujo

El algoritmo para el Spline Cúbico se presentan en las figuras 3 y 4.

En base al diagrama de flujo de la figura 3 se ha elaborado el siguiente programa para el cálculo de los coeficientes:

```
« 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
    X Y N R S H B C D Q P I
  « X SIZE 0. CON { B C D H R S } STO
    X SIZE 1. - 1. GET 'N' STO
    H
    1. N FOR J
      J X J 1. + GET X J GET - PUT
    NEXT
    'H' STO
    2. N FOR J
      J 1. - 'I' STO
      2. H I GET H J GET + *
      H I GET R I GET * - 'Q' STO
      3. Y J 1. + GET Y J GET - H J GET /
      Y J GET Y I GET - H I GET / - * 'P' STO
      R J H J GET Q / PUT 'R' STO
      S J P H I GET S I GET * - Q / PUT 'S' STO
    NEXT
    N 2. FOR J
      C J
      S J GET R J GET C J 1. + GET * - PUT
      'C' STO
    -1. STEP
    1. N FOR J
      J 1. + 'I' STO
      D J C I GET C J GET - 3. / H J GET / PUT 'D' STO
      B J Y I GET Y J GET - H J GET /
      2. C J GET * C I GET + H J GET * 3. / -
      PUT 'B' STO
    NEXT
    B C D
  »
»
```

En lo sucesivo se asume que este programa ha sido guardado con el nombre *COSP3*.

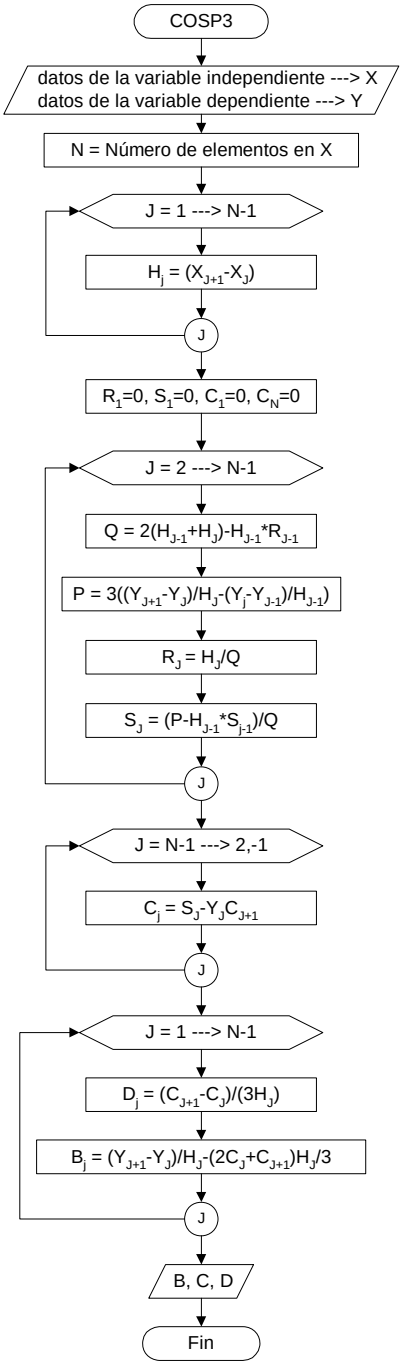


Figura 3. Cálculo de los coeficientes para el método de Interpolación Segmentaria Cúbico (Spline Cúbico)

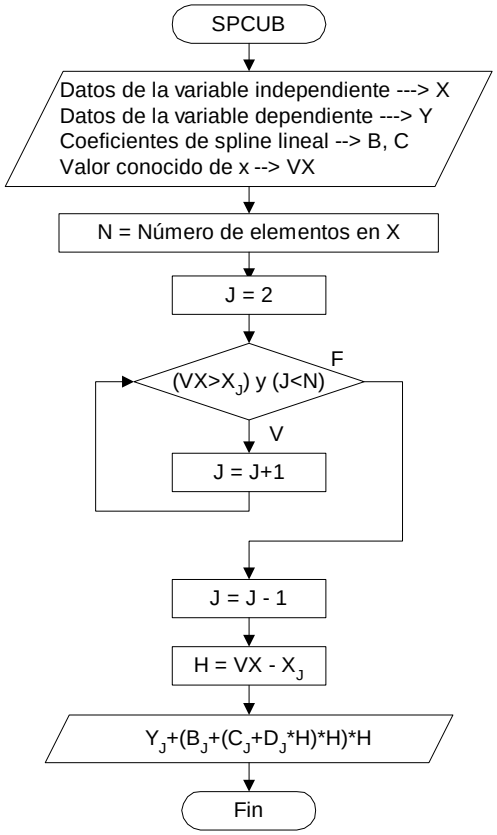


Figura 4. Interpolación Segementaria Cúbica (Spline Cúbico).

En base a la figura 4 se ha elaborado el siguiente programa para interpolar con el método Spline Cúbico.

```
« 0. 2. 0.   X Y B C D VX N J H
  « X SIZE 1. GET 'N' STO
    WHILE VX X J GET > J N < AND
    REPEAT
      1. 'J' STO+
    END
    'J' 1. STO-
    VX X J GET - 'H' STO
    Y J GET B J GET C J GET
    D J GET H * + H * + H * +
  »
»
```

En lo sucesivo se asume que este programa ha sido almacenado con el nombre SPCUB.

9.2.2 Ejemplos

3 Repita el ejemplo 1 empleando la interpolación segmentaria cúbica.

El programa elaborado en el ejercicio 1 debe ser modificado en la siguiente forma:

```
« 0 0 0. 0. 0. 0.   XE YE VX BE CE DE
  « [ 3. 5. 7. 11. 14. 16. 19. 21. 24. 25. 26. 27.
    29. 33. 36. 40. 44. ] 'XE' STO
    [ 0. 0. 0. 0. 1. 3. 13. 22. 29. 26. 18. 8. 3.
    0. 0. 0. 0. ] 'YE' STO
    [ 1. 4. 6. 13. 17. 23. 25.5 26.5 28. 30. 42.
    45. ] 'VX' STO
    XE YE COSP3 'DE' STO 'CE' STO 'BE' STO
    1. 12. FOR I
```

```
    XE YE BE CE DE VX I GET SPCUB
  NEXT
  VX SIZE 1 GET  ARRY "Y"  TAG
 »
»
```

Haciendo correr el programa se obtienen: y: [0. -2.88211755885E-3 8.64635267655E-3 .546933778822 5.4526609775 28.5916326188 22.590423809 12.7057998618 3.58993009265 2.51768329954 -3.15976857932E-2 .01974855362]

9.2.3 Ejercicios y preguntas para clases

1 En el método Spline cúbico, ¿cuál es la ecuación que se emplea para interpolar en cada uno de los segmentos?

2 En el método Spline cúbico ¿qué se asume para el primer y último segmento?

3 En el algoritmo de interpolación segmentaria cúbica, ¿cuál es el propósito del ciclo mientras?

4 Para calcular los coeficientes del Spline cúbico, ¿qué es necesario resolver y con qué método se resuelve?

5 Resuelva el ejercicio 1 empleando el método de interpolación segmentaría cúbico.

6 Resuelva el ejercicio 2 empleando el método de interpolación segmentaria cúbico.

9.3 Ejercicios para la casa

Empleando el método de Newton – Raphson y los métodos spline lineal y cúbico elabore dos programas que calculen los valores de x_i, y_i, k_x y k_y del siguiente sistema de ecuaciones:

$$f(x_i) = \frac{k_x}{k_y} + \frac{y - y_i}{x - x_i} = 0$$

$$k_y = \frac{k'_y}{(1-y)_{im}}; \quad (1-y)_{im} = \frac{(1-y_i)-(1-y)}{\ln\left(\frac{1-y_i}{1-y}\right)}$$

$$k_x = \frac{k'_x}{(1-x)_{im}}; \quad (1-x)_{im} = \frac{(1-x)-(1-x_i)}{\ln\left(\frac{1-x}{1-x_i}\right)}$$

$$k'_x = 1.967x10^{-3}; \quad k'_y = 1.465x10^{-3}; \quad y = 0.39; \quad x = 0.1$$

x_i	0	0.05	0.10	0.15	0.20	0.25	0.30	0.35
y_i	0	0.022	0.052	0.087	0.131	0.187	0.265	0.385