Métodos numéricos. Introducción, aplicaciones y propagación

Antonio Huerta Cerezuelo
Josep Sarrate-Ramos
Antonio Rodríguez-Ferran

Primera edición:septiembre de 1998

Con la colaboración del Servei de Publicacions de la UPC

Diseño de la cubierta: Antoni Gutiérrez

Edicions de la Universitat Politècnica de Catalunya, SL
Jordi Girona Salgado 31, 08034 Barcelona
Tel. 934 016 883 Fax. 934 015 885
Edicions Virtuals: www.edicionsupc.es
e-mail: edupc@sg.upc.es

Producción: CPET (Centre de Publicacions del Campus Nord)
La Cup. C. Gran Capità s/n, 08034 Barcelona

Depósito legal: B-31.600-98
ISBN: 84-8301-265-0

Índice

Prólogo

Este libro presenta una breve introducción a los métodos numéricos. Abarca desde la introducción a los ordenadores y la programación en lenguaje FORTRAN hasta las aplicaciones, haciendo una incursión en los métodos numéricos propiamente dichos.

De hecho, todos los temas del libro se tratan de forma básica. Sólo al abordar los métodos directos para sistemas lineales de ecuaciones se profundiza más, buscando dar una base sólida, puesto que es uno de los temas fundamentales en métodos numéricos para ingeniería.

Las erratas y errores son completamente atribuibles a los autores. Sin embargo, los aciertos, tanto en el enfoque como en el contenido, son de todos los profesores que participan y han participado en las asignaturas de métodos numéricos que impartimos. Seguramente, de entre todos ellos, el más señalado es Manuel Casteleiro, maestro de todos nosotros.

1 Introducción al uso de los ordenadores

Objetivos

- Describir las diferencias conceptuales entre los ordenadores analógicos y digitales.
- Presentar las características principales de los componentes básicos de un ordenador personal.

1.1 Introducción

Durante las últimas décadas, el ordenador se ha convertido en una de las herramientas más potentes y útiles de que dispone el ingeniero. Su utilización abarca desde la fase de diseño y validación experimental en un laboratorio, hasta la fase de construcción o producción industrial, pasando por la confección de planos y la redacción de los pliegos de condiciones en los que se utilizan diferentes equipos de CAD y ofimática. Paralelamente a este auge también ha aparecido la necesidad de recurrir a diferentes, y cada vez más sofisticados, métodos numéricos en varias de las anteriores fases.

A la vista de lo anterior y aunque el objetivo de este libro no sea el estudio detallado del funcionamiento interno de un ordenador, es muy interesante que un ingeniero posea unos conocimientos mínimos sobre dicho funcionamiento. Además, este conocimiento le facilitará la comprensión de los lenguajes de programación así como el análisis e interpretación tanto de los resultados obtenidos como de los posibles errores de programación.

1.2 Tipos de ordenadores

Desde el punto de vista conceptual, existen dos tipos de ordenadores: los ordenadores analógicos y los digitales. Los *ordenadores analógicos* se basan en una analogía entre las ecuaciones que rigen el problema que se desea simular y un fenómeno físico fácilmente reproducible

en el laboratorio. Se caracterizan por:

1. Ser difícilmente programables. Es decir, se diseñan específicamente para un tipo de problema.

2. La velocidad de cálculo depende del fenómeno físico que se utiliza para simular el problema que se desea resolver.

Con el propósito de ilustrar el funcionamiento de este tipo de ordenadores supóngase que se debe diseñar un determinado tipo de suspensión. En un estudio preliminar se puede aproximar el sistema de suspensión por un muelle perfectamente elástico (de constante elástica k) y un amortiguador viscoso (de viscosidad c) instalados en paralelo como muestra la figura 1.1.a. Así mismo se puede aproximar el cuerpo que reposa sobre dicho sistema por una masa puntual m.

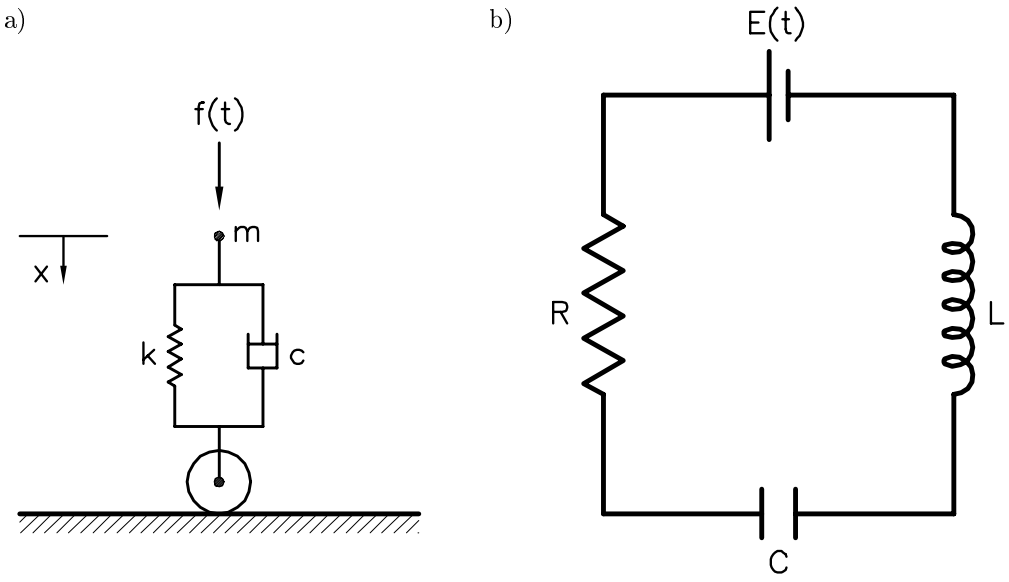


Fig. 1.1 a) esquema del tipo de amortiguador que se desea estudiar; b) ordenador analógico utilizado para su estudio

De acuerdo con la segunda ley de Newton, si sobre la masa puntual m actúa una fuerza $f(t)$, su movimiento se puede describir mediante la ecuación

$$f(t) - kx(t) - c\dot{x}(t) - m\ddot{x}(t) = 0 \tag{1.1}$$

donde $x(t)$, $\dot{x}(t)$ y $\ddot{x}(t)$ representan la posición, velocidad y aceleración de la masa puntual respectivamente.

Si se desea construir un ordenador analógico que permita simular el problema anterior, es

imprescindible hallar un fenómeno que pueda ser descrito mediante una ecuación similar a la 1.1. Para ello, se puede construir un circuito eléctrico por el que circule una corriente de intensidad I, formado por una fuente de alimentación de potencial $E(t)$, conectada en serie a una bobina de inductancia L, a un condensador de capacidad C y una resistencia R, como muestra la figura 1.1.b.

De acuerdo con la ley de Ohm y puesto que la intensidad es la derivada temporal de la carga eléctrica ($I = \dot{q}(t)$), la diferencia de potencial entre los bornes de la fuente de alimentación verifica

$$E(t) \; - \; \frac{1}{C} \; q(t) \; - \; R \; \dot{q}(t) \; - \; L \; \ddot{q}(t) \; = \; 0 \tag{1.2}$$

Como puede observarse, las ecuaciones que rigen ambos problemas son del mismo tipo, de forma que obteniendo los factores de escala pertinentes se puede predecir el comportamiento del sistema de suspensión a partir del circuito eléctrico. Es importante resaltar que este tipo de ordenador permite simular, casi en tiempo real, el anterior sistema de suspensión. Sin embargo, no permite calcular otras cosas, como por ejemplo las raíces de una ecuación de segundo grado. Por estas razones, en la actualidad la utilización de los ordenadores analógicos se limita, básicamente, a equipos de laboratorio destinados a la adquisición de datos.

Por el contrario, los *ordenadores digitales* basan su funcionamiento en las diferentes propiedades de los componentes electrónicos que los constituyen. Conceptualmente se identifican por su capacidad de realizar operaciones lógicas y aritméticas con dígitos. Se caracterizan por:

1. Ser fácilmente programables. En este sentido, se dice que son ordenadores de propósito general.

2. Presentar una gran potencia de cálculo.

3. La velocidad de cálculo depende del tipo de ordenador, pero, en general, suele ser inferior a la de los ordenadores analógicos.

1.3 Ordenadores digitales

Los ordenadores con los que habitualmente se trabaja (PCs, estaciones de trabajo, superordenadores, ...) son ordenadores digitales. Su funcionamiento se basa en un soporte físico o *hardware* constituido por todos los componentes materiales que lo forman (circuitos integrados, placas, pantallas, discos, ...), y un soporte lógico o *software* compuesto por un conjunto de programas que gestionan y/o se pueden ejecutar en el ordenador. Se denomina *sistema operativo* al conjunto de programas y utilidades necesarios para el funcionamiento del ordenador.

El *hardware* de un ordenador se compone básicamente de: 1) la unidad central de proceso o CPU (*Central Processing Unit*); existen ordenadores con más de una CPU; 2) la memoria central; 3) la unidad de control de entrada y salida con los periféricos; 4) la unidad de control de comunicación por red y 5) los periféricos (ver figura 1.2).

1. La unidad central de proceso (CPU) es el componente del ordenador encargado de ejecutar las instrucciones y los programas que residen, total o parcialmente, en la memoria. A nivel conceptual se compone de dos unidades. La primera se denomina *unidad de control* y se encarga de controlar la ejecución de los programas. La segunda es la *unidad aritmético–lógica*, que se encarga de realizar las operaciones ordenadas por la unidad de control sobre los datos que ésta le suministra: suma, resta, multiplicación, división, concatenación, comparación, etc.

2. La memoria es el componente del ordenador encargado de almacenar los datos y los programas que debe tratar la CPU. Se denomina memoria RAM (*Random Access Memory*) a la parte de la memoria del ordenador susceptible de ser modificada. En consecuencia, en ella residen los programas que desarrollan los usuarios y los datos que dichos programas precisan, así como una parte de los programas que gestionan el funcionamiento del ordenador. Se denomina memoria ROM (*Read Only Memory*) a la parte de la memoria que no es posible modificar y, en consecuencia, sólo puede ser leída. En ella reside la parte mínima del sistema operativo necesaria para que el ordenador se pueda poner en marcha. Por último se debe mencionar que la velocidad con que se puede acceder a los datos almacenados en este tipo de memorias es muy inferior (órdenes de magnitud) a la velocidad con que la CPU puede operar con ellos. A fin de paliar estas diferencias, entre la memoria del ordenador y su CPU se instala una memoria adicional llamada *memoria caché* (ver figura 1.2), que se caracteriza por una velocidad de acceso muy superior, por una capacidad de almacenamiento muy inferior, y en general, por un precio muy elevado.

3. La unidad de control de entrada y salida (E/S) con los periféricos es el componente del ordenador destinado a controlar y gestionar la comunicación con los diferentes periféricos conectados al mismo.

4. La unidad de control de comunicación por red es el componente del ordenador encargado del control y la gestión de los dispositivos destinados a la comunicación entre ordenadores mediante cable coaxial, fibra óptica o cualquier otro soporte similar.

5. Los periféricos son todos aquellos componentes del ordenador que facilitan su funcionamiento y la comunicación entre él y los usuarios. Por ejemplo:

    a) Unidades de discos fijos

    b) Unidades de discos extraíbles

    c) Unidades de cintas magnéticas

    d) Pantallas

    e) Teclados

f) Impresoras

g) *Plotters*

h) Equipos de lectura óptica (*scanners*, ...)

i) Digitalizadores

k) Equipos de comunicación mediante líneas telefónicas (*modems*)

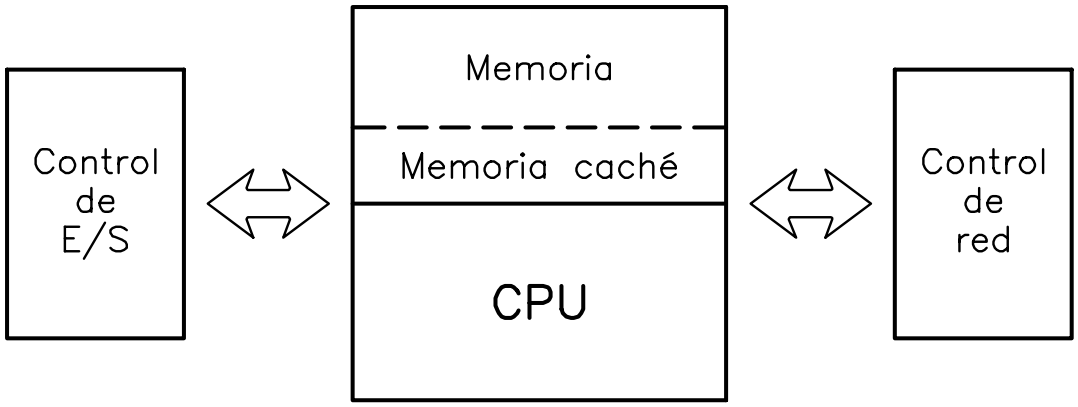


Fig. 1.2 Organización y estructura del hardware *de un ordenador*

En este curso de métodos numéricos se utilizará, básicamente, un tipo de ordenador digital denominado *ordenador personal* o PC (*Personal Computer*). Como su propio nombre indica, es un tipo de ordenador diseñado para que lo utilice un solo usuario y que éste sea el responsable de su gestión (en contraposición a los ordenadores diseñados para ser utilizados por varios usuarios al mismo tiempo y gestionados por una persona especialmente formada a tal efecto, denominados *ordenadores multiusuario*). Sin embargo, debido al gran nivel de expansión y a la ingente disponibilidad de *software* sobre este tipo de plataformas, ha sido preciso desarrollar nuevos procedimientos que permitan compartir recursos y gestionar conjuntos de PCs destinados a un mismo tipo de trabajo. En consecuencia, han aparecido en el mercado los productos de *hardware* y *software* necesarios para realizar dicha conexión. De esta forma han nacido las denominadas *redes de PCs* que no son más que un conjunto de ordenadores personales conectados, entre los cuales hay uno, denominado servidor (*server*), destinado a gestionar y servir recursos al resto de equipos.

1.4 *Software*

Desde un punto de vista muy genérico el *software* existente en un ordenador se puede clasificar en: 1) sistema operativo; 2) programas o utilidades genéricas y 3) programas y ficheros de los usuarios.

1. Como se ha comentado anteriormente, el sistema operativo está formado por un conjunto de programas encargados de gestionar el funcionamiento del ordenador. Sus tareas cubren un rango muy amplio de aplicaciones que van desde transmitir a la CPU determinados datos hasta visualizar por pantalla el contenido de un archivo.

2. Las utilidades genéricas son programas comercializados por el mismo fabricante del ordenador, o por otra marcas comerciales, que permiten realizar tareas muy diversas, como por ejemplo correo electrónico, compiladores, bases de datos, procesadores de texto, entre otras. Estas aplicaciones basan su funcionamiento en el sistema operativo.

3. Los programas y ficheros de los usuarios contienen el trabajo que realizan los diferentes usuarios del ordenador. Su funcionamiento y utilización se basa tanto en las utilidades genéricas como en el propio sistema operativo.

Tabla 1.1 Equivalencia entre las diferentes unidades de medida de la información

UNIDADES DE MEDIDA DE LA INFORMACIÓN	
Valor original	Valor equivalente
1 byte	8 bits
1 Kbyte	1024 bytes
1 Mbyte	1024 Kbytes
1 Gbyte	1024 Mbytes

Puesto que el espacio disponible para almacenar todos estos programas y datos es limitado, los usuarios de un ordenador deben poder saber cuánta información contiene cada programa (en otras palabras: cuánto ocupa). En un ordenador toda la información (programas, datos, etc.) se almacena en sistema binario, esto es, mediante secuencias de unos (1) y ceros (0). A la cantidad mínima de información, es decir, un (1) o un (0), se la denomina *bit*. Evidentemente, esta unidad es demasiado pequeña para medir la cantidad de información que normalmente se maneja en un ordenador. En consecuencia, se definen algunos múltiplos del bit (ver tabla 1.1).

Se denomina *byte* a una cadena de ocho bits, por ejemplo:

```
10101010
01101110
```

Así mismo, se define un kilobyte (Kbyte) como 1024 bytes ($1024 \times 8 = 8192$ bits). Del mismo modo se define un megabyte (Mbyte) como 1024 Kbytes y un gigabyte (Gbyte) como 1024 Mbytes. Mientras que las unidades anteriores son totalmente estándares y ampliamente utilizadas, en algunos ordenadores se define otra unidad denominada *bloque* que equivale a 512 bytes (1/2 Kbyte).

1.5 Bibliografía

BISHOP, P. *Conceptos de informática.* Anaya, 1989.

BORSE, G.J. *Programación FORTRAN77 con aplicaciones de cálculo numérico en ciencias e ingeniería.* Anaya, 1989.

GUILERA AGÜERA, LL. *Introducción a la informática.* Edunsa, 1988.

2 Introducción a los sistemas operativos

Escrito en colaboración con Miguel Ángel BRETONES

Objetivos

- Establecer la organización de los archivos según una estructura de directorios y subdirectorios.
- Describir las principales instrucciones del sistema operativo MS-DOS.
- Familiarizarse con el entorno MS Windows.
- Presentar las principales características de la hoja de cálculo MS Excel.

2.1 Introducción

Se denomina *sistema operativo* al conjunto de programas y utilidades necesarios para el funcionamiento del ordenador. Existen en la actualidad multitud de sistemas operativos, gran parte de ellos asociados casi unívocamente a un tipo de ordenador. Así, el sistema operativo de la inmensa mayoría de los ordenadores personales es el llamado MS-DOS (abreviatura de *MicroSoft Digital Operating System*).

El conocimiento del sistema operativo consiste, desde el punto de vista del usuario, en aprender a comunicarse con el ordenador de manera que éste ejecute órdenes. De esta manera, todo se reduce a conocer la manera de transmitirle instrucciones sin que sea necesario, por ejemplo, saber cómo está programado el sistema operativo.

El MS-DOS (de ahora en adelante DOS) nació a finales de los 80; actualmente el uso del entorno Windows se encuentra ampliamente generalizado. El Windows, en cualquiera de

sus sucesivas versiones, es un sistema operativo basado en la plataforma del DOS (es decir, aprovecha todas las facilidades que éste proporciona) pero con vocación de resultar más cómodo de manejo para el usuario. Desde este punto de vista, no puede ser considerado estrictamente distinto del DOS. En muchos casos, tan sólo cambia el *interfase* (el canal de comunicación o la manera de transmitir instrucciones) con la máquina.
A medida que el entorno Windows ha ido evolucionando, las diferencias por cuanto a facilidad y agilidad de uso se han ido acentuando, pero siempre conservando la mayor parte de ventajas (y carencias) del DOS.

Antes de conocer las instrucciones fundamentales de cualquiera de estos sistemas, conviene definir algunos conceptos básicos generales, que son de aplicación común a todo sistema operativo.

2.2 Estructura de directorios

Cualquier información, programa, hoja de datos o de resultados, etc., contenida en un ordenador debe estar almacenada en un *archivo* o fichero. Este término hace referencia a un concepto de *software*: la información está contenida en ficheros desde el punto de vista del *software* y no del *hardware*, desde el que se podría hablar de información almacenada en la memoria RAM, en el disco duro,...

El símil más frecuentemente empleado para describir esta idea consiste en imaginar la memoria del ordenador como un archivador. Cada una de las hojas de los diversos expedientes, carpetas o libros almacenados en él sería un fichero informático. Naturalmente, las hojas pueden contener información muy diversa, desde poesías a crucigramas, pasando por apuntes de clase, problemas, etc.

Ahora bien, resulta razonable suponer que los ficheros deberán organizarse siguiendo una estructura ordenada que facilite su gestión: es evidente la diferencia que existe entre un archivador cuyo contenido está correctamente clasificado y las mismas hojas almacenadas desordenadamente en una caja. Así, los ficheros se agrupan en *directorios* y *subdirectorios*, también llamados *carpetas* en el entorno Windows. Siguiendo con el ejemplo del archivador, los directorios representarían las carpetas donde se guardan las hojas de papel. El concepto de directorio es general e independiente del sistema operativo concreto que se esté tratando.

No existe una diferencia formal entre directorio y subdirectorio. Usualmente se denomina subdirectorio a aquel directorio contenido en otro directorio. Es perfectamente posible que unos directorios contengan a otros, de la misma manera que una carpeta puede contener, a su vez, otras carpetas junto con hojas sueltas. Análogamente, no puede ocurrir que un archivo contenga directorios.

Se puede establecer así una *estructura de árbol* en la que archivos y directorios se organizan en función de a qué directorio superior (aquel que los contiene) pertenezcan. El directorio que ocupa la cúspide del árbol es aquel que no está contenido por ningún otro y generalmente se

denomina *directorio principal*.

La estructura antes descrita permite una ordenación racional de la información. Por ejemplo, la figura 2.1 podría representar la estructura típica del archivador de un estudiante.

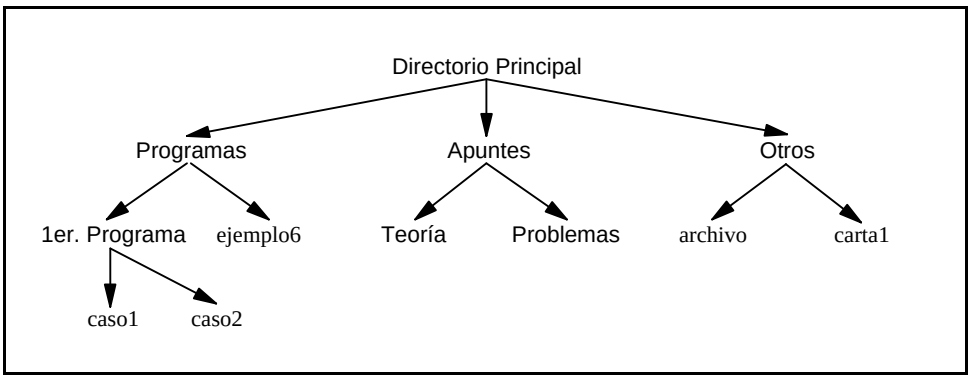


Fig. 2.1 Estructura de árbol de directorios

Como puede observarse, los diversos temas de interés están agrupados por conceptos o materias; lo mismo ocurrirá con los programas (ficheros) de ordenador. En Helvética figuran los nombres de los directorios o subdirectorios, mientras que los archivos aparecen con tipografía corriente.

2.3 Edición de un archivo

Hasta ahora se ha definido cuál debe ser la estructura interna de organización de los diversos archivos en un ordenador; en consecuencia se admite que, de alguna manera, éstos ya existen. Ahora bien, ¿cómo puede "generarse" un archivo? Resulta evidente que algunos de los ficheros que interesan a los usuarios, como por ejemplo los de resultados, los "escribirán" los programas que cada usuario diseña. No ocurrirá lo mismo con el propio programa, un archivo de datos, una carta, etc.

Para escribir (*editar*) archivos en general se utiliza una aplicación (conocida genéricamente como *editor*) que facilita esta tarea. Existen multitud de editores en el mercado, cuyas posibilidades y facilidad de manejo son bastante semejantes, al menos en el ámbito de los ordenadores personales. Además, muchos programas y aplicaciones informáticas incorporan su propio sistema de edición para la escritura de archivos de datos u otros.

En general, un archivo queda identificado por su *nombre*. Éste puede ser una cadena de números y letras (por ejemplo `carta1`). Además, resulta conveniente que ese nombre vaya acompañado de una *extensión*: una extensión no es más que una cadena adicional de letras que informa acerca del contenido del fichero. Así, existen un conjunto de extensiones estándares en función de que se trate de ficheros de texto (`txt`), de resultados (`res`), de datos (`dat`), etc. De esta forma, el nombre de un archivo podría ser `carta1.txt`. Algunas de estas extensiones son asignadas de manera automática (*por defecto*) por el propio sistema operativo, mientras que otras se podrán escoger libremente, respetando o no la convención antes establecida.

En MS-DOS, y en las versiones de Windows anteriores a Windows'95, existe una limitación acerca del número de *caracteres* (números o letras) que puede contener un nombre o una extensión, que no puede ser superior a 8 y 3 respectivamente. Conviene respetar, en la medida de lo posible, el mencionado criterio incluso en el caso de trabajar con entornos Windows, ya que de esta forman se evitarán, por ejemplo, posibles problemas de compatibilidad de nombres de archivos en entornos de trabajo en red.

Para editar el archivo `carta1.txt` desde el entorno de trabajo que proporciona el MS-DOS, se debería invocar la aplicación concreta de edición de la que se disponga desde el *prompt* (el símbolo que aparece en la pantalla del ordenador a la izquierda del cursor y a partir del cual se puede escribir) del PC. Para ello en la mayoría de casos basta con escribir su nombre seguido del nombre del archivo que se desea editar.

La manera de disponer de una sesión de DOS en un ordenador cuyo arranque por defecto se produzca en entorno Windows consiste, como posteriormente se verá, por ejemplo en activar el icono de acceso directo "Símbolo de MS-DOS" , elegir la opción "MS-DOS" dentro del menú desplegable de inicio o en reiniciar la computadora en MS-DOS.

2.4 Manipulación de ficheros

Dentro de cualquier ordenador, los archivos pueden ser manipulados de manera muy diversa: pueden ser copiados, borrados, cambiados de nombre, movidos de directorio, etc. En este apartado se pretenden mostrar las instrucciones elementales del sistema operativo DOS.

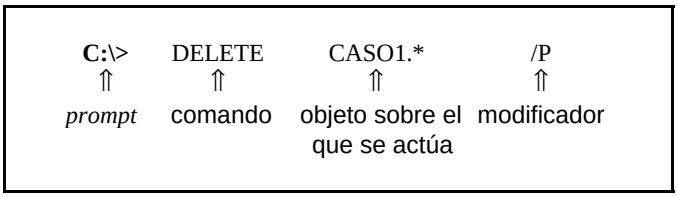
Todas las operaciones aquí descritas se podrán realizar de manera semejante a través del entorno Windows, si bien de un modo más "visual" y cómodo (véase apartado 2.5 para más detalles).

Debe tenerse en cuenta que, a diferencia del Windows, el *interfase* de comunicación en DOS es estrictamente alfanumérico: ello quiere decir que las sentencias necesariamente deberán ser cadenas de palabras que se introducirán en el ordenador usando el teclado. Posteriormente veremos que las posibilidades de empleo del ratón en sistema Windows amplían y simplifican la comunicación con el ordenador.

Como conceptos previos básicos, cabría destacar los siguientes:

2.4.1 Sintaxis de comandos

Cualquier conjunto de instrucciones en DOS (una *sentencia*) tiene la misma estructura. Por ejemplo:



El *comando* es el nombre propio que define la acción que se desea realizar; en este caso, borrar el archivo que anteriormente se ha editado. A su vez, dicho comando puede ir acompañado de *modificadores* (uno, muchos o ninguno) que alteran, aunque no de manera sustancial, la acción del comando. En este ejemplo, la variante /P (todos los calificadores en DOS comienzan por "/" seguidos de una letra) obliga a que el ordenador pregunte al usuario si realmente desea borrar el archivo antes de ejecutar la instrucción (¡el uso de este modificador resulta por tanto altamente recomendable!). Finalmente, el objeto sobre el que se actúa es, naturalmente, el que recibe la acción del comando.

Existen otras posibilidades, directamente importadas de otros sistemas operativos como el UNIX, para calificar comandos o encadenar sentencias; para aprender su funcionamiento, conviene consultar por ejemplo el significado de los siguientes símbolos: | , > , etc.

En las tablas 2.1 y 2.2 se presentan, agrupados por temas, los comandos fundamentales en DOS.

Tabla 2.1 Manejo de directorios

Uso	Comando	Modificadores habituales	Ejemplo
Cambiar el directorio de trabajo	CD		CD \PROGRAMAS\PROG1
Ver los archivos y directorios contenidos en el directorio de trabajo	DIR	P W S	DIR /W
Crear un directorio	MKDIR		MKDIR PRUEBAS
Borrar un directorio	RMDIR		RMDIR PROG2

Tabla 2.2 Manipulación de ficheros

Uso	Comando	Modificadores habituales	Ejemplo
Listar un archivo	TYPE		TYPE C:\PROG2\CASO1.FOR
Copiar un archivo	COPY	V	COPY CASO1.FOR ..*.*
Cambiar de nombre un archivo	RENAME		REN CASO1.FOR *.TXT
Mover un archivo	MOVE		MOVE *.* APUNTES
Borrar un archivo	DELETE	P	DELETE *.* /P

La primera tabla hace referencia a las operaciones más habituales en el manejo de directorios, como pueden ser su creación o borrado.
La segunda tabla contiene las sentencias relacionadas con la gestión de archivos y su relación con los directorios a los que pertenecen. A lo largo de este apartado se comprobará el significado concreto de algunos de los ejemplos que acompañan a los distintos comandos.

Muchos de ellos se pueden abreviar a la hora de ser introducidos en el ordenador. De esta manera, el comando `DELETE` puede ser abreviado empleando `DEL`, por ejemplo. Otro tanto ocurre con el comando `RENAME`, como puede también apreciarse en la tabla 2.2.

Uno de los comandos más empleados es el destinado a conocer los archivos y directorios contenidos en un determinado directorio.
La sentencia `DIR` proporciona dicha información, indicando los nombres y extensiones de los archivos. Los directorios aparecen diferenciados de los archivos por ir acompañados de la palabra clave $<$ `DIR` $>$.

2.4.2 Comodines

Como se puede observar en el ejemplo del subapartado precedente, el archivo (`caso1`) no queda especificado por un nombre y una extensión, sino que en lugar de ésta última aparece un asterisco (`*`). En DOS, al asterisco se le denomina *comodín*.
Un comodín es un carácter que actúa como sustituto de cualquier otro carácter (incluido el espacio en blanco) o grupo de caracteres. El mencionado concepto funciona de manera idéntica en entorno Windows.

De esta forma, la instrucción completa que servía de ejemplo en el subapartado 2.4.1 especifica que se borren, previa confirmación, todos los archivos `caso1` sea cual sea su extensión (`caso1.txt`, `caso1.dat`, `caso1.res`, etc).

2.4.3 Especificación de directorios

En DOS, un archivo queda definido por su nombre y su extensión (`caso1.for`); ahora bien, resulta perfectamente posible la existencia de dos archivos con igual nombre y extensión, situados en directorios diferentes. En ese caso, ¿cómo distinguirlos? Para responder a esta pregunta, en la figura 2.2 se presenta una posible estructura de directorios.

Suponiendo que el archivo `caso1.for` esté situado en el subdirectorio `PROG1`, el *nombre completo* de dicho archivo será `C:\PROGRAMAS\PROG1\caso1.for`. Obsérvese que, de esta manera, cualquier archivo queda caracterizado unívocamente, a pesar de que pueda compartir con otros nombre o extensión.

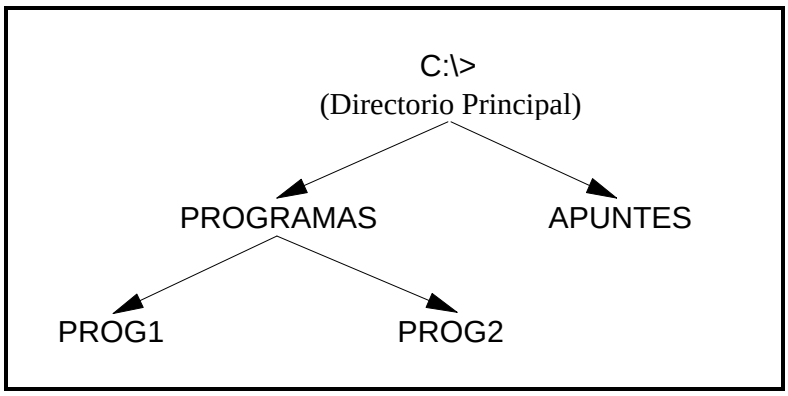


Fig. 2.2 Estructura de directorios

Al especificar un archivo tan sólo por su nombre y extensión (`caso1.for`) se asume que éste se encuentra en el *directorio de trabajo*. El usuario puede decidir en cuál de los directorios de los que eventualmente disponga quiere trabajar: eso significa que las sentencias que ejecute se realizarán en ese directorio. Así, por ejemplo, cuando en el apartado 2.3 se hacía referencia a la edición del archivo `carta1.txt`, éste quedaba grabado en el directorio de trabajo.

Al inicio de una sesión, el directorio de trabajo, también llamado *directorio por defecto*, es el directorio principal. Empleando la sentencia `CD` el usuario puede cambiar el directorio de trabajo. Así, en el ejemplo de la tabla 2.1 se puede ver cuál es la sentencia que hay que introducir para cambiar desde el directorio principal al que contiene `caso1.for`.

Trabajando desde cualquier directorio, el usuario puede especificar cualquier archivo en una sentencia utilizando bien su nombre y extensión o bien su nombre completo. Como se ha comentado anteriormente, para hacer referencia a un archivo contenido en el directorio de trabajo basta emplear su nombre y extensión. Por contra, si el archivo (por ejemplo `caso1.for`) está contenido en otro subdirectorio (`PROG2`) empleando como directorio de trabajo el principal hay que usar:

```
C:\> TYPE C:\PROGRAMAS\PROG2\CASO1.FOR
```

que es el ejemplo que figura en la tabla 2.2. Como puede verse, el uso del nombre completo de un archivo permite referirse a él con independencia del directorio por defecto que se esté usando en ese momento, si bien su abuso puede resultar farragoso a la hora de escribir las sentencias que se quieran ejecutar.

Existe una última posibilidad a la hora de especificar los nombres de los archivos presentes

en un ordenador, que representa un término intermedio entre los casos anteriores. En éste el nombre de un archivo no situado en el directorio de trabajo se especifica describiendo el camino que se debe recorrer, siguiendo el árbol de directorios, para acceder a él desde el directorio de trabajo.

Así por ejemplo, en la tabla 2.2 aparece la manera como se debería copiar el archivo `caso1.for` situado en el subdirectorio `PROG1` en el directorio `PROGRAMAS`. Observando el esquema de la figura 2.2, el archivo debe quedar copiado en el nivel superior del árbol respecto al que se encuentra inicialmente. La especificación formada por dos puntos consecutivos ".." significa precisamente *ascender un nivel en el árbol de directorios*. A partir de ahí, componiendo ascensos y descensos en los niveles de directorios, siempre separados por barras "\", se puede describir el nombre *relativo* de un archivo. Se debe tener en cuenta que, a diferencia de lo que ocurría anteriormente, el nombre relativo sí depende del directorio de trabajo.

Resulta fácil imaginar que existen muchas otras instrucciones y posibilidades en DOS; aquí tan sólo se han destacado las básicas. En cualquier caso, si desea saber más cosas, siempre queda el recurso al sistema de información que el propio sistema operativo pone a disposición del usuario; con él, se puede pedir ayuda acerca de las variantes y posibilidades de un comando cuyo nombre conozcamos. Esto se consigue a través del calificador /? . De esta forma, basta ejecutar

```
C:\> DIR  /?
```

Para obtener toda la información disponible sobre el comando `DIR`.

Windows'95 marca el declive del uso de las pantallas de MS-DOS, como vía para la manipulación de archivos o la ejecución de programas. Entre las causas de este fenómeno cabe destacar la masiva adaptación de los programas y aplicaciones informáticas al trabajo en entorno Windows, así como la mejora en las capacidades y versatilidad del propio sistema operativo.

2.5 Utilización del entorno Windows

El MS-Windows es probablemente el sistema operativo más popular. A diferencia de lo que ocurría con el sistema DOS, en el que está basado, toda la manipulación de ficheros puede realizarse de una manera visual, esto es, casi sin la intervención de sentencias alfanuméricas o el uso del propio teclado. Antes de presentar el entorno de Windows, es importante destacar un elemento de *hardware* fundamental en la gestión del sistema: el *ratón*. El ratón proporciona un cursor móvil a lo largo de la pantalla, que permite ejecutar instrucciones, seleccionar iconos y aplicaciones, arrastrar otros objetos, etc. Existen multitud de ratones, la mayoría de ellos con 2, 3 o 4 botones; en Windows estándar el más importante es el botón izquierdo, el cual, en función de su uso, tiene diversas aplicaciones. Por ejemplo:

1. Pulsar una vez (*simple-clic*) el botón izquierdo sirve para *activar* o *desactivar* ventanas o, en general, para *seleccionar* los diversos elementos del Windows.

2. Pulsar de manera rápida y repetida (*doble-clic*) el botón izquierdo tiene el efecto de *ejecutar* algún comando o de *activar* alguna aplicación representada por un icono. También se emplea para restituir ventanas u otras acciones relacionadas.

3. Finalmente, manteniendo el botón izquierdo pulsado sin soltarlo se consigue *arrastrar* comandos u objetos.
   Como posteriormente se comentará, ésta es una de las maniobras fundamentales del entorno Windows y su utilización resulta básica en aplicaciones como un procesador de textos o una hoja de cálculo, entre otras. A la vez, también sirve para mover elementos, alterar el tamaño de las ventanas, etc.

4. Una vez seleccionado un objeto empleando el procedimiento descrito en el punto 1, el botón derecho del ratón suele permitir ejecutar determinadas acciones sobre el objeto, que normalmente dependerán de su naturaleza.
   Ello se consigue gracias a la aparición, al pulsar el botón derecho, de un *menú desplegable* donde se contienen las posibles acciones a ejecutar.

5. Así mismo, el solo posicionamiento del puntero del ratón sobre determinados elementos puede producir efectos.
   Esta acción generalmente permitirá obtener información y eventualmente ayuda acerca del objeto al cual se "apunte". Para ello basta dejar unos segundos quieto el cursor, y aparecerá un globo de ayuda acerca del mencionado objeto; procediendo según el punto 1, se obtendrá la información.

El uso concreto de todos y cada uno de los movimientos del ratón depende mucho de la situación específica y de la habilidad del usuario; su manejo preciso y, en general, el de todo el sistema Windows, se convierte así en un proceso de aprendizaje, que contiene dosis importantes de intuición y experiencia.

2.5.1 Los elementos del entorno Windows

La figura 2.3 muestra el aspecto que presenta una pantalla típica de un ordenador personal funcionando en entorno Windows. Los globos de ayuda (que naturalmente no aparecen en la pantalla real) indican los nombres de los principales elementos que conforman el sistema de ventanas del Windows.

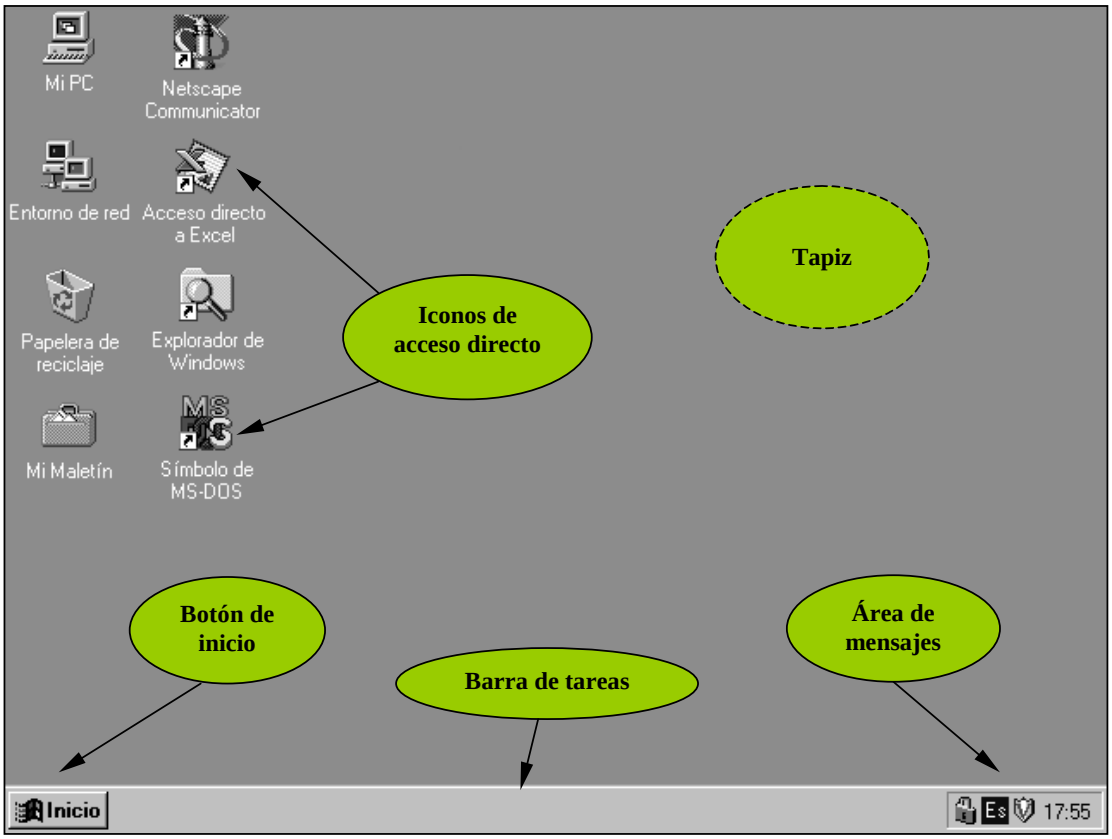


Fig. 2.3 Aspecto general del entorno Windows

En el lenguaje propio del Windows, lo que aparece en la figura 2.3 es el *escritorio* de nuestro ordenador (*desktop* en las versiones inglesas del programa). Sobre él se encuentran todos los elementos presentes y disponibles para el usuario en cada momento, las aplicaciones, los controles, las propias ventanas de trabajo, etc.

El escritorio se encuentra dividido en dos zonas: el *tapiz* y la *barra de tareas*. El tapiz ocupa la mayor parte de la pantalla y sobre él se "incrustarán" dos tipos de elementos fundamentales en Windows: los iconos y las ventanas de trabajo. La barra de tareas es la zona diferenciada del tapiz que normalmente se encuentra en el lado inferior de la pantalla. Como su propio nombre indica, sobre ella aparecerá información relativa, por ejemplo, a las aplicaciones que en aquel momento se estén ejecutando. Así, en el ejemplo de la figura 2.3, la barra de tareas aparece vacía.

De entre los *iconos* que aparecen en el tapiz, existen básicamente de dos tipos:

1. Por una parte, los propios del sistema, como "Mi PC", "Entorno de red", "Mi Maletín" o la "Papelera de reciclaje".
   Cada uno de ellos tiene una función específica pero en general están relacionados con la gestión y el manejo de archivos. Así, desde "Mi PC" se puede acceder al conjunto de carpetas que contiene el ordenador, de manera muy semejante a la propia de otros sistemas operativos como el OS de Macintosh. De la misma forma, cualquier archivo borrado pasa a ser depositado en la "Papelera de reciclaje".

2. Por otra parte, existen los *iconos de acceso directo*, que se distinguen de los primeros por tener dibujada una flecha en la esquina inferior izquierda.
   Se trata de iconos asignados unívocamente a las aplicaciones más frecuentemente empleadas por el usuario.
   El efecto de ejecutar sobre ellos un doble-clic con el ratón consiste en la activación de la aplicación deseada. En concreto, en la figura 2.3 se aprecian los iconos de acceso directo a dos aplicaciones, que son la hoja de cálculo Microsoft Excel, sobre la que se hablará más tarde, y el navegador de Internet Netscape Communicator.
   Finalmente, el icono de acceso al MS-DOS posibilita la entrada en el modo MS-DOS, que permite aplicar lo descrito en el apartado 2.4.

En la barra de tareas existen, a su vez, otros dos elementos integrados más. Por una parte, está el *área de mensajes*; en ella suelen aparecer una serie de iconos identificativos de diversos procesos presentes en el sistema.
Entre los más habituales destacan el reloj horario, el funcionamiento de los altavoces o, en general, de cualquier periférico como tarjetas de red, dispositivos de almacenamiento externo, la actividad de alguna aplicación antivirus, etc. Haciendo un doble-clic sobre cada uno de ellos se puede obtener información acerca de su estado de actividad.
En segundo lugar, aparece el *botón de inicio*. Se trata del objeto más importante del escritorio, puesto que bajo él se encuentra el *menú desplegable principal*. Si se ejecuta un simple-clic sobre el botón de inicio aparecerá el menú de la figura 2.4.

En él aparecen los grandes grupos de objetos presentes en el sistema. Así, por ejemplo, desde la opción "Ayuda" se podrá acceder al sistema de ayuda interactiva de Windows, desde la opción "Cerrar el sistema" se podrá apagar el equipo o reiniciarlo en modo MS-DOS, etc.

La opción "Programas" del menú desplegable principal contiene recogidas por *grupos* todas las aplicaciones y los programas presentes en el ordenador. Los grupos existentes representan los conjuntos de programas que contiene el sistema y que están asociados a una aplicación concreta.
Así, por ejemplo, en la figura 2.4 se aprecian, entre otros, el grupo asociado al paquete de programas "Microsoft Office" (Microsoft Excel, Microsoft Word y otros) o a los accesorios del sistema. Cada grupo está representado por un *icono* que, como su propio nombre indica, es un símbolo que representa al objeto en cuestión.

Fig. 2.4 Menú desplegable de inicio

Entre las diversas opciones del menú de programas también aparece la opción para abrir una ventana de MS-DOS. Así, el icono de acceso directo anteriormente aludido y que se hallaba en el tapiz no representa sino un "atajo" para ejecutar la mencionada aplicación, sin tener que desplegar los menús que aparecen en la figura 2.4. Seleccionando esta opción por cualquiera de los dos procedimientos se conseguirá idéntico resultado, eso es, la activación de una ventana en modo MS-DOS.

2.5.2 Las ventanas del Windows

Una de las aplicaciones fundamentales que proporciona el entorno Windows es el "Explorador de Windows". Con ella se puede gestionar todo lo referente al manejo de archivos y directorios especificado en el apartado 2.4, pero desde el punto de vista del Windows; así por ejemplo, se podrá cambiar el nombre de los archivos, su lugar de almacenamiento, borrar archivos,etc.

El sistema Windows basa todo su funcionamiento en la representación de un conjunto de símbolos y ventanas. Cada aplicación en ejecución lleva asociada una o más ventanas que quedan reflejadas en el tapiz.

El aspecto de la ventana del "Explorador de Windows", para el ejemplo descrito en la figura 2.2, podría ser el que aparece en la figura 2.5.

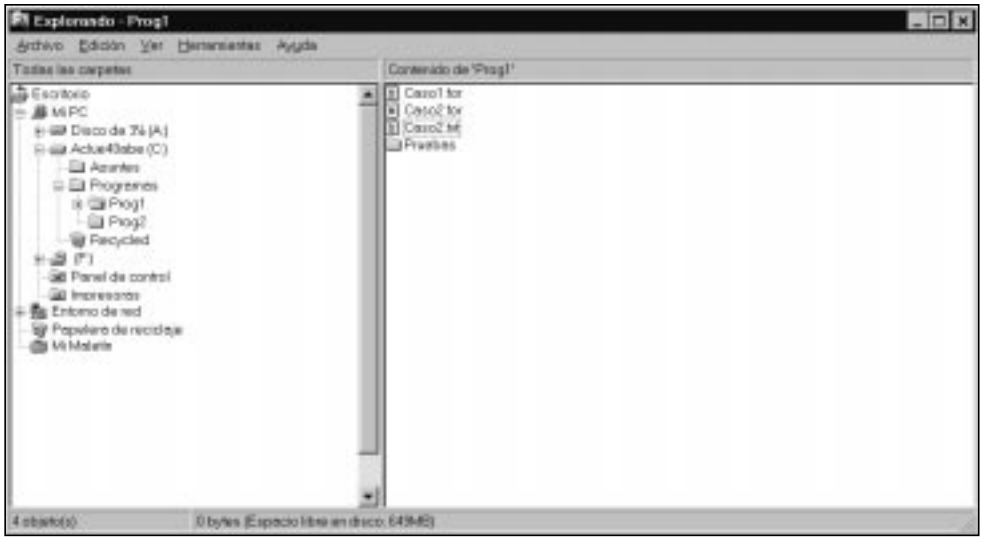


Fig. 2.5 Explorador de Windows

Esta ventana se *abre* seleccionando la opción del Explorador de Windows en el menú desplegable de programas, según se aprecia en la figura 2.4. De la misma forma, para ejecutar la mencionada acción también se habría podido emplear el icono de acceso directo presente en el tapiz del escritorio.

En el explorador de Windows, los archivos y directorios aparecen gráficamente representados. En la zona de la izquierda aparece el árbol de directorios correspondiente a la unidad de trabajo. Una vez se ha seleccionado un directorio (haciendo un simple-clic sobre el icono de la carpeta correspondiente), en la ventana de la derecha aparecen los archivos y subdirectorios que contiene. Ya sea a través de las sentencias ejecutables desde la barra de menús, o desde los menús desplegables que se pueden obtener con el botón derecho del ratón, se pueden realizar todas las operaciones habituales en la gestión de archivos y directorios (copiar, renombrar, eliminar, etc.). También se puede acceder a los diversos directorios, arrastrar archivos para moverlos, etc.

Todas las ventanas de Windows presentan una estructura muy parecida. En general, una ventana abierta consta, al menos, de los siguientes elementos:

1. Una *barra de título*, que contiene el nombre de la ventana; su color indica si dicha ventana está *activa* o no. En Windows, tan sólo puede haber una ventana activa en cada momento, si bien puede haber más de una ventana abierta. La diferencia entre un concepto y otro reside en que las instrucciones que el usuario introduce en el ordenador (a través del ratón o del teclado) se ejecutan siempre en la ventana activa. Para activar o desactivar ventanas basta con hacer un simple-clic sobre ellas. En general, los *procesos* que se ejecutan desde una ventana no se detienen por su desactivación. Se entiende por procesos aquellas acciones automáticas que no requieren de la intervención directa del usuario a través del teclado o del ratón.

2. La *barra de menú* contiene una serie de llamadas genéricas, tales como, por ejemplo, en este caso "Archivo", "Edición", "Ver", etc. Cuando se selecciona (simple-clic) una de éstas con el ratón, aparece un menú desplegable. En él se encuentran las opciones que pueden ejecutarse (seleccionándolas con el ratón) normalmente relacionadas con el tema que figura en la barra de menús. Así, por ejemplo, en el desplegable "Archivo" se encontrarán comandos relacionados con el manejo de los archivos, tales como crear nuevos archivos, etc. Para emplear estos menús desplegables es necesario situar el cursor del ratón sobre la opción deseada y hacer un simple-clic.

3. La *barra de movimiento* sirve para desplazar la parte de la ventana visible tanto en sentido vertical como horizontal, en el caso en que, dado el tamaño de la ventana, no se pueda ver todo su contenido. La dimensión de las ventanas puede ser modificado arrastrando con el ratón sus esquinas. En el caso del explorador de Windows, la ventana principal está a su vez dividida en dos ventanas secundarias, cada una de las cuales cuenta con sus propias barras de movimiento.

4. Finalmente, en el lado derecho de la barra de títulos aparecen unos *botones* cuadrados cuya función también está destinada al manejo de las ventanas. Estos botones son, de izquierda a derecha:

Botón principal de la aplicación: Normalmente representado por el icono de la propia aplicación. Si se selecciona con un simple-clic, aparece el menú desplegable de control de la ventana. Entre otras funciones, este menú permite abrir o cerrar la ventana, minimizarla, etc.

Botones de minimizar y maximizar: A fin de evitar que todas las aplicaciones abiertas durante una sesión de trabajo "tapen" la pantalla, existe la posibilidad de que algunas (o todas) sean reducidas (minimizadas). De esta forma, el botón de la izquierda transforma la aplicación en su icono en la barra de tareas, mientras que el de la derecha la "extiende" hasta ocupar toda la pantalla. Para devolver una aplicación minimizada a su estado normal basta hacer un simple-clic sobre el icono correspondiente en la barra de tareas.

Botones de minimizar y restaurar: En la situación en que la ventana haya sido maximizada empleando los botones anteriores, éstos son sustituidos por la pareja minimizar/restaurar. Con el de la izquierda se sigue pasando desde la ventana al icono, mientras que con el de la derecha se restituye el tamaño original que tenía la ventana antes de maximizarla.

Botón de ayuda: Puede aparecer en algunas ventanas especiales, como por ejemplo las relacionadas con los paneles de control o la configuración del sistema, para proporcionar ayuda específica sobre el contenido de las mismas.

Botón de cerrar: Se emplea en todos los casos para cerrar la ventana y, consiguientemente, la aplicación que ésta pueda representar.
A diferencia de otros botones, que pueden estar o no presentes en la ventana, siempre se encontrará el botón de cerrar en el extremo superior derecho de todas las ventanas.

2.6 Introducción al manejo de Excel

Una de las aplicaciones más empleadas, de entre todas las que pueden ejecutarse bajo Windows, es la *hoja de cálculo* Excel. Una hoja de cálculo es una potente herramienta con la que efectuar, con gran rapidez y de manera interactiva, multitud de cálculos aritméticos. Por ejemplo, con una hoja de cálculo un usuario puede desde representar en gráficos los resultados de sus programas hasta construir complejas macros, pasando por todo tipo de operaciones matemáticas.

El objetivo de este apartado no es describir exhaustivamente el funcionamiento de Excel (por lo demás, bastante semejante al de otras hojas de cálculo existentes en el mercado, como por ejemplo Lotus 1-2-3 o Quattro Pro) sino facilitar los conocimientos básicos necesarios para poder empezar a trabajar con ella.

Antes de comenzar, al igual que en el apartado anterior, en la figura 2.6 se presenta cuál es el aspecto de la ventana asociada a Excel (esto es, aquella que se abre cuando se hace doble-clic

sobre el icono de acceso directo de la figura 2.3).

La figura 2.6 contiene los elementos básicos descritos en toda ventana, como las barras de título, menú y movimiento o los botones. Además de estos elementos existen otros propios ya de la aplicación (en este caso la hoja de cálculo) como por ejemplo:

1. Los *botones de herramientas* situados bajo la barra de menú, que están asociados (es decir son sinónimos) de todos o algunos de los comandos de los menús desplegables de la barra de menú. Haciendo simple-clic sobre ellos se ejecuta la misma acción que seleccionando la orden del correspondiente menú, lo que agiliza el manejo de la hoja de cálculo.

2. La *barra de fórmulas*, inmediatamente por debajo de los botones de herramientas. Allí se irán reflejando los cálculos que el usuario vaya programando.

3. Las *celdas* de Excel. Se trata de cada uno de los rectángulos en que está dividida el *área de trabajo*, cada uno de los cuales se identifica con dos coordenadas: una letra creciente en sentido horizontal y un número en vertical. Las *reglas* que contienen los números y letras de las celdas aparecen en los bordes de la ventana.

4. Las *pestañas* de hoja, situadas sobre la barra inferior izquierda, que permiten seleccionar cada una de las *hojas* o diversas áreas de trabajo de las que consta una hoja de cálculo Excel.

5. Los *botones de desplazamiento de pestaña*, situados inmediatamente a la izquierda y que permiten cambiar de hoja.

6. La *barra de estado*, emplazada en extremo inferior, donde aparecen mensajes en función de la acción que se está llevando a cabo en cada momento.

Las celdas son los elementos fundamentales de la hoja de cálculo: a cada celda se podrá *asociar* un número o una fórmula (cuyo resultado, en general también será un número). De esta manera, a base de realizar cálculos aritméticos en las diversas celdas es como se resuelve un problema con una hoja de cálculo.
La gran potencia de estos sistemas radica en la facilidad para *vincular* unas operaciones aritméticas a otras, lo que permite realizar cálculos con una simplicidad extraordinaria: con todo, el mejor modo de comprender los fundamentos de Excel es conocerlos a través de un sencillo ejemplo como el siguiente:

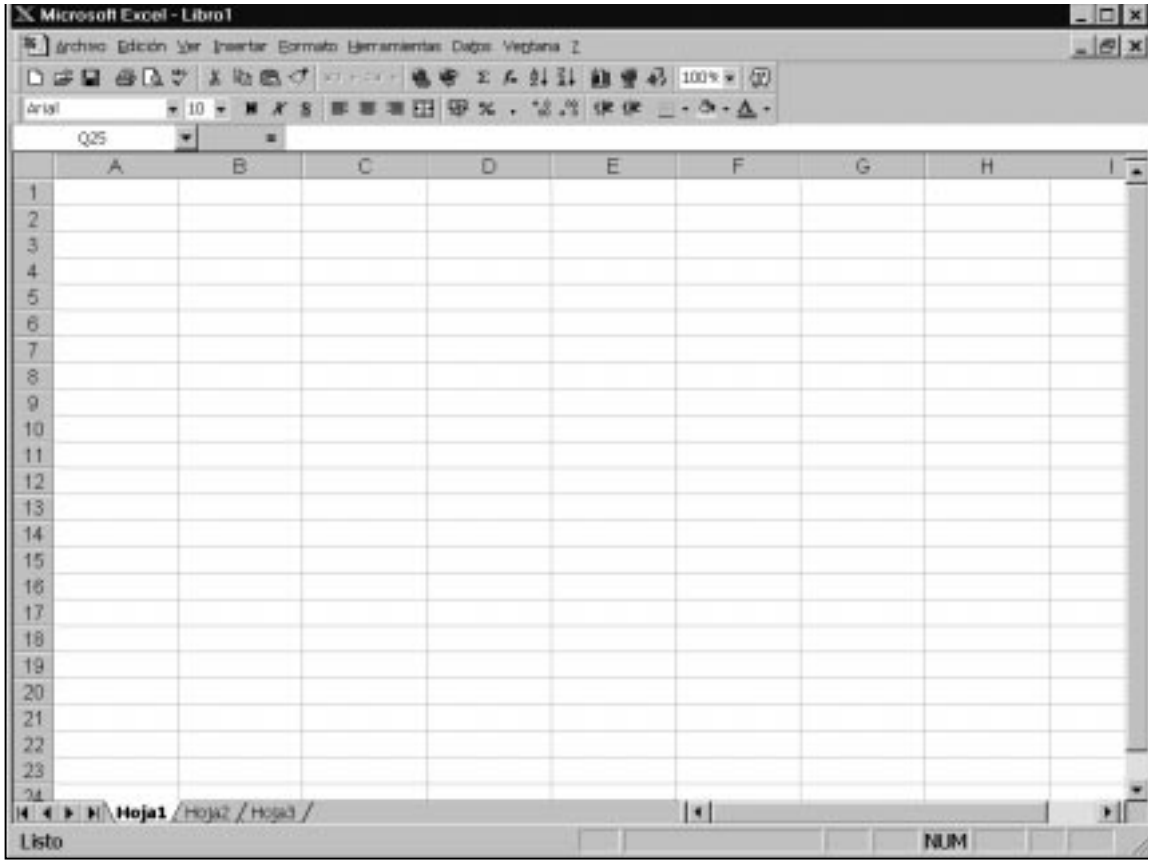
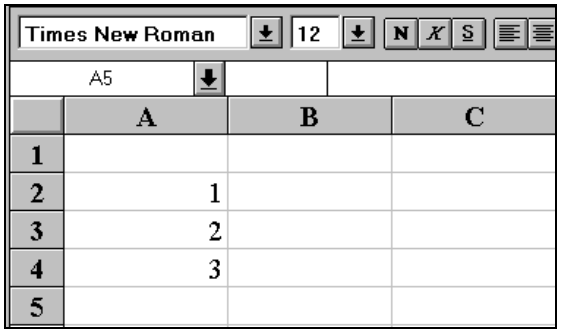


Fig. 2.6 Hoja de cálculo Excel

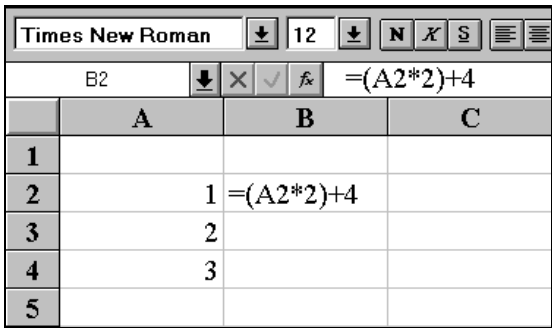
2.6.1 Paso 1: Introducción de constantes

Para asociar un escalar a una celda, basta seleccionar la casilla donde se desea colocarlo (haciendo simple-clic sobre ella con el ratón) e introducir el número. Por ejemplo, si se desea colocar los valores `1`, `2` y `3` en las celdas `A2`, `A3` y `A4`, se debe seleccionar con el ratón cada una de ellas e introducir respectivamente los valores anteriores. Una vez pulsado *Return*, el resultado obtenido es:



2.6.2 Paso 2: Introducción de fórmulas

A continuación se realizará una operación elemental con las tres casillas que ya contienen números; para ello, en la celda `B2` se define la operación consistente en tomar el número `1`, multiplicarlo por `2` y sumarle `4` unidades. La manera de proceder consiste en seleccionar la celda `B2` e introducir por teclado la fórmula. A medida que el usuario la escribe, ésta aparece reflejada tanto en la barra de fórmulas como en la propia celda:



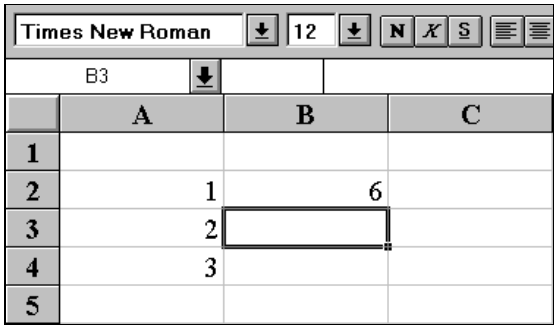
Como puede observarse, existen algunas diferencias con respecto al paso precedente:

1. En primer lugar, el primer carácter introducido es el signo de igualdad (`=`); ésta es la manera de decirle a Excel que efectúe el cálculo que a continuación se encuentra. Una vez pulsada la tecla *Return*, la celda `B2` dejará de contener la fórmula para mostrar el valor de la operación, según se ve en el dibujo posterior (en este caso `1 x 2 + 4 = 6`).

2. En segundo lugar, obsérvese que en la posición de la fórmula donde debería aparecer la cifra `1` figura la coordenada de la casilla en que éste se encuentra. Esto se consigue, durante la fase de escritura de la fórmula, haciendo simple-clic sobre la casilla cuyo valor se desea introducir en el momento en que ésta debe figurar en la fórmula.

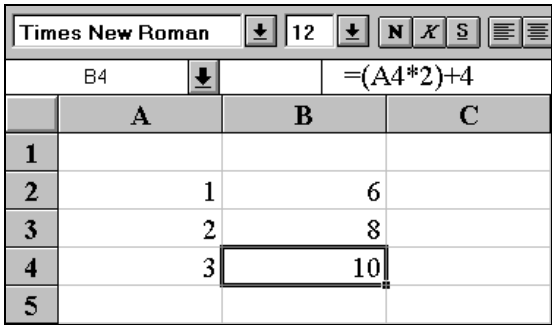
   Existe una diferencia fundamental entre introducir el `1` y la coordenada `A2` (que en este momento tiene como valor `1`); la manera aquí empleada establece una *relación dinámica* entre las casillas `A2` y `B2`, como se verá posteriormente en el paso 4.

Este sería hasta ahora el resultado del paso 2:



2.6.3 Paso 3: Arrastre de fórmulas

A continuación se selecciona la celda `B2` con simple-clic y se coloca el cursor en el ángulo inferior derecho de la celda, justo hasta que el cursor cambia de la forma habitual de flecha a la de una cruz (`+`). Seguidamente, manteniendo el botón pulsado, se arrastra el ratón desde la casilla `B2` hasta la `B4`. Al soltar el botón, se obtiene:



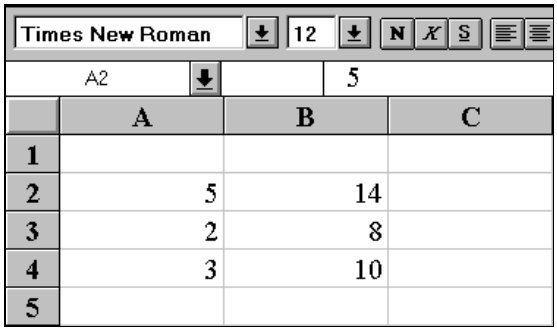
Lo que ha ocurrido es que la fórmula que contenía la celda `B2` ha sido arrastrada a las casillas `B3` y `B4`. Eso significa que si la fórmula original de `B2` era "`( A2 * 2 ) + 4`" la que está calculada en `B3` es "`( A3 * 2 ) + 4`". En la barra de fórmulas puede verse a qué corresponde la casilla `B4`: el hecho de que se haya sustituido la casilla `A2` por la `A3` o la `A4`, *siguiendo el sentido del arrastre* obedece a que cuando se arrastra una fórmula, los vínculos también son arrastrados en el mismo sentido. Esto puede ser evitado, en el caso en que se desee, anteponiendo el símbolo de dólar (`$`) en las especificaciones de las coordenadas de los vínculos.

Así por ejemplo, en el caso de introducir en el paso 2 la fórmula como "`( $A$2 * 2 ) + 4`" el resultado que quedaría arrastrado en la casillas `B3` y `B4` sería "`( $A$2 * 2 ) + 4`". La anteposición del carácter $ a una coordenada de fila y/o columna de una casilla tiene el efecto de bloquearla, impidiendo que se modifique dinámicamente cuando la fórmula es arrastrada. En una misma fórmula pueden coexistir casillas libres, bloqueadas por filas, por columnas, completamente bloqueadas, etc.

El efecto conseguido arrastrando celdas es el mismo que puede obtenerse con las opciones de *cortar* y *pegar* fórmulas que figuran tanto en el desplegable asociado a "Edición" como en los correspondientes botones de herramientas.

2.6.4 Paso 4: Modificación dinámica

Finalmente, continuando con el ejemplo, al sustituir el valor original de la casilla `A2`, por un `5`, se obtiene:



	A	B	C
1			
2	5	14	
3	2	8	
4	3	10	
5			

Como se puede observar, el valor de la casilla `B2` se ha *actualizado* de `6` a `14` es decir "`(5 x 2) + 4`". No ocurre lo mismo con las fórmulas arrastradas, dado que al depender únicamente de `A3` y `A4` no deben sufrir modificaciones. Este proceso se realiza automáticamente en todos los vínculos (de ahí el nombre de dinámicos) presentes en una hoja de cálculo. En este hecho y en el anterior (el concepto de arrastre) es donde radica gran parte de su potencia.

El ejemplo anterior es una muestra muy simplificada de las capacidades de Excel, muchas de las cuales sólo se van conociendo con el uso del programa.

Problema 2.1:
A partir del ejemplo descrito en el subapartado 2.6.3, modificarlo de manera que en la columna `C` de la hoja de cálculo aparezcan las ordenadas de la función $y = 3x^2 - 2x - 3$ siendo `A` la columna de las abscisas. ●

Problema 2.2:
Extender el cálculo de las funciones $y = 2x - 4$ e $y = 3x^2 - 2x - 3$ al intervalo en x [-2,2], obteniendo puntos cada 0.5 unidades de x. ●

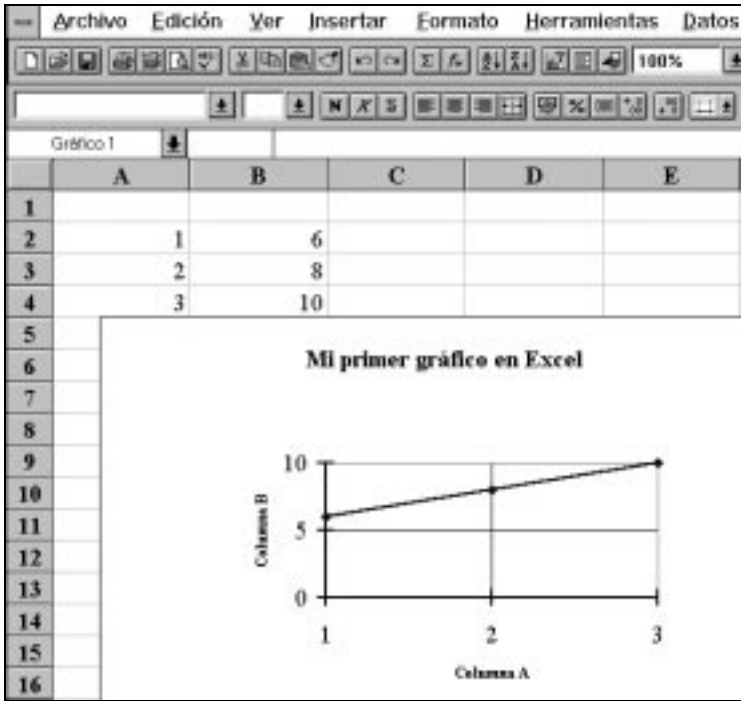
Problema 2.3:
Escribir una hoja de cálculo en la que aparezcan de manera ordenada las tablas de multiplicar de los números pares entre 2 y 10 para los números comprendidos entre 1 y 20. Para conseguirlo, se procurará emplear la mayor cantidad posible de vínculos dinámicos entre las diversas fórmulas necesarias, de manera que la cantidad de celdas en las que se deban introducir explícitamente fórmulas resulte mínima. ●

2.6.5 Representación gráfica

Un aspecto muy útil, por cuanto a la representación de resultados se refiere, consiste en la posibilidad que proporciona Excel de transformar conjuntos de datos en gráficos. Así por ejemplo, una vez generadas las coordenadas X Y de una gráfica, puede emplearse el *Asistente para gráficos* para construir el correspondiente dibujo, que quedará insertado en la zona de la hoja de cálculo que se escoja.

El mencionado asistente consiste en un conjunto de pantallas de ayuda paso a paso que orientan e informan al usuario acerca del proceso que se debe seguir para obtener el gráfico deseado. El Asistente para gráficos proporciona múltiples posibilidades a la hora de escoger la forma, tipo y formato del gráfico que se quiere representar, y sólo requiere de un poco de esfuerzo para familiarizarse con su uso.

Así, continuando con el ejemplo anterior, los valores calculados en el subapartado 2.6.3 pueden ser interpretados como 3 puntos de la recta $y = 2x + 4$. Al dibujar los datos anteriores se obtiene:



Problema 2.4:
Representar gráficamente las funciones del problema 2.2 y estimar sus puntos de intersección. ●

2.6.6 Importación de resultados

Finalmente, la última gran cualidad de Excel hace referencia a la posibilidad de adquirir conjuntos de datos generados por otros programas, por ejemplo los archivos de resultados de los programas de FORTRAN.

Así, casi cualquier conjunto de datos escrito en un fichero puede ser importado de manera automática por la hoja de cálculo, esto es, sin necesidad de introducirlos manualmente. A partir de ese momento, siguiendo lo comentado en los anteriores subapartados, pueden realizarse cálculos adicionales con esos resultados o simplemente obtener representaciones más vistosas de los mismos, por ejemplo con la ayuda de gráficos.

Para importar archivos de resultados debe emplearse el *Asistente para importación de texto* que se activa automáticamente en el momento de intentar abrir con Excel un archivo que no tenga formato de hoja de cálculo. Por último, debe tenerse en cuenta que, en general, Excel

identifica el punto decimal con el carácter "," de modo que el archivo de datos que se quiera importar debe respetar esta convención.

Problema 2.5:
El último problema de este capítulo ilustra una de las múltiples aplicaciones de las hojas de cálculo; concretamente, se calculará el valor de las amortizaciones de un crédito a un interés dado, así como su tipo anual equivalente (TAE).
Muchas de las fórmulas y procedimientos necesarios para resolver este problema representarán una novedad en el manejo de Excel y están dirigidas a contribuir a su aprendizaje.
Previamente, resulta necesario plantear el problema que se desea resolver. Para calcular las cuotas fijas de préstamos con interés constante, se define x_0 como el capital prestado en $t = t_0$ (instante inicial) a un interés fijo i expresado en tanto por uno.

Cálculo de cuotas

La liquidación del préstamo se realizará en N pagos por año (por ejemplo N son los periodos de liquidación anual, 12 si son liquidaciones mensuales); así, el interés asociado a cada periodo de liquidación será de i/N.
Al final del primer periodo, es decir en $t = t_1$, el capital adeudado (el necesario para cancelar el préstamo) será de $x_0\,(1 + i/N)$. Ahora bien, en vez de cancelar el préstamo se paga una cuota c (lógicamente inferior a la cantidad total adeudada). La deuda será ahora $x_1 = x_0\,(1 + i/N) - c$. Este proceso se repite sucesivamente hasta la total extinción del crédito. Expresando lo anterior en una tabla se tiene:

Vencimiento	Capital Prestado
t_0	x_0
t_1	$x_1 = x_0\left(1+\frac{i}{N}\right) - c$
t_2	$x_2 = x_0\left(1+\frac{i}{N}\right)^2 - c\left[1+\left(1+\frac{i}{N}\right)\right]$
t_3	$x_3 = x_0\left(1+\frac{i}{N}\right)^3 - c\left[1+\left(1+\frac{i}{N}\right)+\left(1+\frac{i}{N}\right)^2\right]$
$\ldots$	$\ldots$
t_n	$x_n = x_0\left(1+\frac{i}{N}\right)^n - c\ \frac{N}{i}\left[\left(1+\frac{i}{N}\right)^n - 1\right]$

Este proceso se detiene cuando x_n se anula (es decir, cuando ya no se tiene más deuda). A partir de la ecuación $x_n = 0$ resulta fácil determinar el valor de la cuota que se debe pagar c, en n periodos totales de liquidación:

$$c = x_0\,\frac{i}{N}\,\frac{\left(1+\frac{i}{N}\right)^n}{\left(1+\frac{i}{N}\right)^n - 1}$$

Así, el total pagado es de nc y por consiguiente el costo real del préstamo puede

evaluarse en $nc - x_0$.
La fórmula anterior es útil para determinar las cuotas mensuales que hay que pagar, c, o el costo del préstamo $nc - x_0$, a pesar de no contemplar las variaciones reales del valor del dinero.
A pesar de ello y puesto que c es un número real, la fórmula que se emplea en la práctica tomará el entero más próximo restos.

Tipo Anual Equivalente

Es usual tanto en el ámbito de préstamos como en el de intereses de cuentas bancarias que la información emitida por las entidades de ahorro se refiera al TAE. Por tanto, resulta necesario conocer cómo se relaciona el interés anual i (expresado en tanto por uno) con el tipo anual equivalente, TAE (también expresado como un tanto por uno).
Si hay N periodos de liquidación anual, el interés asociado a cada periodo de liquidación es de nuevo i/N. Sea x_0 la cantidad prestada o invertida en $t = t_0$ (instante inicial). La acumulación de los intereses para cada periodo Año$/N = \triangle t$ sería:

Vencimiento	Capital Prestado o Invertido
t_0	x_0
$t_1 = t_0 + \triangle t$	$x_0 \left(1 + \frac{i}{N}\right)$
$\ldots$	$\ldots$
$t_N = t_0 + N \triangle t = t_0 + \text{Año}$	$x_0 \left(1 + \frac{i}{N}\right)^N$

En consecuencia, si se define el TAE como el interés anual equivalente, entonces debe verificar:

$$1 + \text{TAE} = \left(1 + \frac{i}{N}\right)^N$$

A partir de esta fórmula es fácil deducir la relación entre i y TAE:

$$\text{TAE} = \left(1 + \frac{i}{N}\right)^N - 1$$

$$i = \left[(1 + \text{TAE})^{\frac{1}{N}} - 1\right] N$$

Se desea elaborar una hoja de cálculo que permita conocer de forma pormenorizada las cuotas, amortizaciones de capital y los intereses del periodo de liquidación dado el capital prestado, el número de periodos totales para devolverlo y el interés anual o el TAE del préstamo.
El resultado final deberá ser semejante a la hoja de cálculo que puede encontrarse a continuación. Seguidamente se indican algunas de las instrucciones clave necesarias para obtenerla.

	A	B	C	D	E
1	**AMORTIZACIONES DE UN PRÉSTAMO**				
2					
3	TAE:		12,13%		
4	Periodos anuales:	12			
5	Interés:	11,50%			
6	Capital:	1.500.000 Pts			
7	Periodos totales:	24			
8	Cuota:	70.260 Pts		Total a pagar:	1.686.240 Pts
9					
10	Vencimiento	Saldo préstamo	Amortización	Intereses	Cuota a pagar
11	5-ene-97	1.500.000 Pts	55.885 Pts	14.375 Pts	70.260 Pts
12	5-feb-97	1.444.115 Pts	56.421 Pts	13.839 Pts	70.260 Pts
13	5-mar-97	1.387.694 Pts	56.961 Pts	13.299 Pts	70.260 Pts
14	5-abr-97	1.330.733 Pts	57.507 Pts	12.753 Pts	70.260 Pts
15	5-may-97	1.273.226 Pts	58.058 Pts	12.202 Pts	70.260 Pts
16	5-jun-97	1.215.168 Pts	58.615 Pts	11.645 Pts	70.260 Pts
17	5-jul-97	1.156.553 Pts	59.176 Pts	11.084 Pts	70.260 Pts
18	5-ago-97	1.097.377 Pts	59.743 Pts	10.517 Pts	70.260 Pts
19	5-sep-97	1.037.634 Pts	60.316 Pts	9.944 Pts	70.260 Pts
20	5-oct-97	977.318 Pts	60.894 Pts	9.366 Pts	70.260 Pts
21	5-nov-97	916.424 Pts	61.478 Pts	8.782 Pts	70.260 Pts
22	5-dic-97	854.946 Pts	62.067 Pts	8.193 Pts	70.260 Pts
23	5-ene-98	792.879 Pts	62.662 Pts	7.598 Pts	70.260 Pts
24	5-feb-98	730.217 Pts	63.262 Pts	6.998 Pts	70.260 Pts
25	5-mar-98	666.955 Pts	63.868 Pts	6.392 Pts	70.260 Pts
26	5-abr-98	603.087 Pts	64.480 Pts	5.780 Pts	70.260 Pts
27	5-may-98	538.607 Pts	65.098 Pts	5.162 Pts	70.260 Pts
28	5-jun-98	473.509 Pts	65.722 Pts	4.538 Pts	70.260 Pts
29	5-jul-98	407.787 Pts	66.352 Pts	3.908 Pts	70.260 Pts
30	5-ago-98	341.435 Pts	66.988 Pts	3.272 Pts	70.260 Pts
31	5-sep-98	274.447 Pts	67.630 Pts	2.630 Pts	70.260 Pts
32	5-oct-98	206.817 Pts	68.278 Pts	1.982 Pts	70.260 Pts
33	5-nov-98	138.539 Pts	68.932 Pts	1.328 Pts	70.260 Pts
34	5-dic-98	69.607 Pts	69.593 Pts	667 Pts	70.260 Pts

Datos del problema: los datos del préstamo deben introducirse en las casillas `B4`, `B6`, `B7` y `B5`; este último valor será conocido a priori o bien se copiará de `C5`, en función de que se sepa como dato el interés anual o el TAE.

Cuota de amortización: en la casilla `B8` se introduce la fórmula definida anteriormente para la cuota fija a pagar; dicho valor debe ser entero.

Interés y TAE: ambos valores pueden darse como dato inicial del préstamo, a

pesar de ello, para los cálculos de la tabla sólo se emplea el interés anual y en particular el valor definido en `B5`.
Ambos (el interés anual y el TAE) se redondearán siempre con *dos* cifras decimales. La hoja de cálculo debe estar organizada de forma que si se introduce en valor del interés anual en `B5` (y por tanto no se introduce el TAE en `B3`) entonces aparece el valor del TAE en `C3` y queda vacío el valor de `C5`. Si, por el contrario, se conoce el valor del TAE (y no el interés anual) que se introduce en `B3`, entonces aparece calculado el valor del interés anual en `C5` (valor que debe ser copiado en `B5`) y queda vacío `C3`.
Como ejemplo, la instrucción que se debe poner en `C3` es
`=Si(B3=0;Redondear((1+B5/B4)^B4-1;4);" ")`.
En `C5` es necesario introducir una fórmula con estructura similar que evalúe el interés anual a partir del TAE. Puede resultar interesante consultar el Asistente de fórmulas para comprender el significado concreto de estas expresiones.

Total a pagar: es la suma de todas las cuotas que se deben pagar.

Fecha de vencimiento: en `A11` se introduce la fecha de inicio y en `A12` la fórmula:
`= Fecha(Año(A11);Mes(A11)+Entero(12/$B$4);Dia(A11)+ Residuo(12;$B$4)*30/$B$4)`.
Esta fórmula debe arrastrarse luego a lo largo de la columna.

Saldo préstamo: en `B11` se copia la cantidad introducida en `B6`; las demás cantidades se obtienen restando al saldo anterior la amortización realizada, por ejemplo `B12` es `=B11-C11`. A continuación se analiza cómo calcular `C11`.

Amortización: es siempre la cuota `$B$8` menos los intereses pagados en ese vencimiento, por ejemplo `C11` es `=$B$8-D11`.

Intereses: se evalúan siempre de la misma manera: el saldo del préstamo multiplicado por el interés asociado a cada periodo de liquidación. ¡Atención: es necesario trabajar con números enteros! De esta forma, en `D11`, por ejemplo, debe introducirse la siguiente fórmula: `=Redondear(B11*$B$5/$B$4;0)`.

Cuota a pagar: la cuota debe ser el valor que aparezca en `B8`; para verificar los cálculos conviene sumar la amortización y los intereses. Por ejemplo, `E11` es `=Suma(C11,D11)`.

Nota importante: ¿Está siempre bien evaluada la última cuota que se debe pagar?

Además de elaborar la hoja de cálculo anterior para el caso indicado, obtener la tabla asociada a un TAE del 12,1% manteniendo los periodos y el capital prestado. ●

2.7 Bibliografía

Presentando Microsoft Windows95. Manual del usuario. Microsoft Corporation, 1998.

Windows para trabajo en grupo & MS-DOS. Manual del usuario, Vols. I a III. Microsoft Corporation, 1994.

STINSON, C. *El libro del Windows 3.1.* Microsoft Press & Anaya, 1993.

MATTHEWS, MARTIN S. *Excel para Windows 95 a su alcance.* Osborne McGraw-Hill, 1996.

THE COBB GROUP (DODGE M.; KINATA C.; STINSON, C.) *Guía completa de Microsoft Excel 5 para Windows.* McGraw-Hill & MS Press, 1996.

Microsoft Excel version 5.0c Manual del usuario. Microsoft Corporation, 1994.

3 Introducción a la programación FORTRAN

Objetivos

- Describir las fases del desarrollo de un programa en FORTRAN.
- Presentar y analizar los elementos básicos de la sintaxis del lenguaje de programación FORTRAN.

3.1 Introducción

El lenguaje de programación FORTRAN fue diseñado por John Backus en 1954 y la primera versión data de 1955. Con posterioridad, han aparecido diferentes versiones que paulatinamente han incorporado mejoras y ampliaciones. Así por ejemplo, en 1958 apareció el FORTRAN II y en 1962 el FORTRAN IV. Una de las versiones más importantes y que ha perdurado durante más tiempo es el FORTRAN 77, que fue aprobado por el *American National Standards Institute* (ANSI) en 1977. Esta versión ha sido mundialmente aceptada y ha permanecido hasta la actualidad como el FORTRAN *universal*.

Aunque ya existe la versión 90 del FORTRAN, por las razones que se han expresado en el párrafo anterior, en este libro se presentará y se analizará la versión FORTRAN 77.

3.2 Fases del desarrollo de un programa en FORTRAN

Como se ha comentado en el capítulo 1, todos los ordenadores funcionan en el sistema binario. En consecuencia, todos los programas deben ser traducidos a dicho sistema, independientemente del lenguaje de programación en que hayan sido diseñados; en este sentido, el sistema binario se denomina también *lenguaje máquina*. En el proceso de traducción se distinguen varias fases (ver figura 3.1):

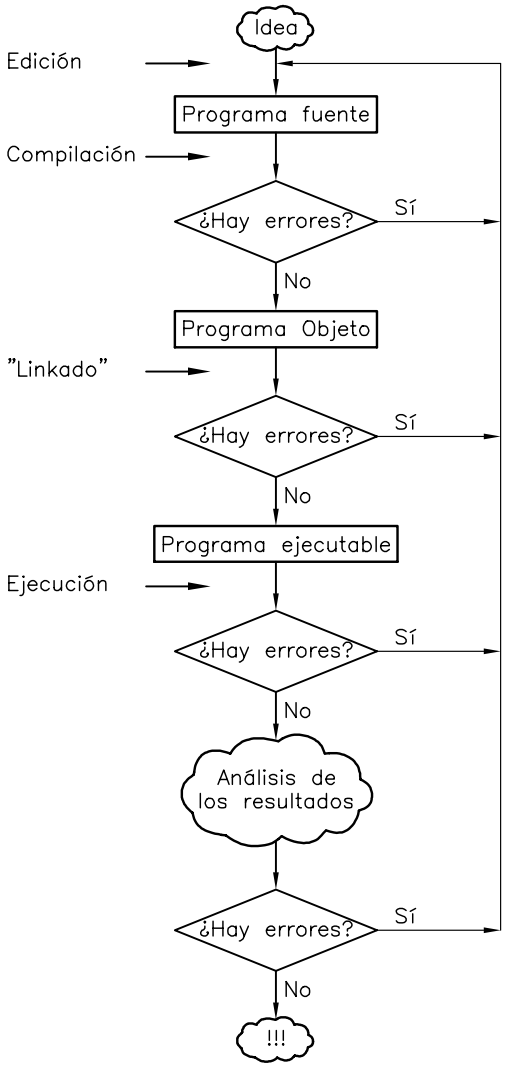


Fig. 3.1 Fases del desarrollo de un programa en FORTRAN

1. Es muy importante que, antes de empezar a escribir el programa en el ordenador, se haya pensado detalladamente cuáles son las tareas que el programa debe realizar y cómo deben programarse. El hecho de invertir cierto tiempo en el diseño del programa conlleva generalmente un ahorro tanto en el tiempo de programación como en el de ejecución.

2. La edición del programa se realiza mediante un editor de archivos (ver capítulo 2). Una vez éste se ha escrito de acuerdo con la sintaxis FORTRAN se obtiene el *programa fuente*. Es muy aconsejable que la extensión del programa fuente sea `FOR`.

3. Como ya se ha comentado, el programa fuente debe traducirse a lenguaje máquina. Esta traducción se realiza en la fase de compilación. El resultado de esta operación es un archivo intermedio que se denomina *programa objeto*. La mayoría de compiladores asignan al programa objeto la extensión `OBJ`.

4. Una vez se ha traducido el programa a lenguaje máquina, aún es preciso realizar ciertas operaciones antes de obtener el *programa ejecutable*. Éstas se realizan en la fase de ensamblado (que generalmente se denomina *linkado*). En esta fase se unen los diferentes módulos que componen el programa y se reserva el espacio de memoria para las variables, los vectores y las matrices que se hayan definido en el mismo. A los programas ejecutables se les asigna la extensión `EXE`.

5. El siguiente paso consiste en ejecutar el programa. Naturalmente, siempre se ejecuta el fichero ejecutable (`EXE`) y no el programa fuente (`FOR`).

6. Por último, los resultados obtenidos deben ser analizados tanto desde el punto de vista computacional como ingenieril.

Es importante resaltar que si en alguna de las anteriores fases se produce un error, su corrección debe realizarse sobre el programa fuente; por tanto, es preciso volver a realizar las fases de compilación, ensamblado y ejecución.

3.3 Organización general de un programa en FORTRAN

En este apartado se presentan los elementos básicos de la organización de un programa escrito en lenguaje FORTRAN.

3.3.1 Normas de escritura de un programa en FORTRAN

El archivo donde se van a escribir las instrucciones (sentencias) FORTRAN se puede imaginar como una hoja cuadriculada y por tanto, formada por filas y columnas.

Por lo que respecta al contenido de un programa por filas hay que resaltar dos aspectos:

1. Todos los compiladores FORTRAN ignoran las filas en blanco.

2. Sólo se puede escribir una sentencia en cada fila. Si el contenido de una sentencia excede la longitud de la fila, se puede utilizar más de una línea, como se detalla más adelante.

Si se analiza el contenido del fichero por columnas, se distinguen cinco zonas diferentes:

Columna 1 Si en la primera columna de una línea aparece una `C`, una `c`, o

un *, significa que dicha línea es un comentario. Por lo tanto, el compilador no la traducirá a lenguaje máquina. En general, los comentarios se utilizan para explicar el funcionamiento de cada bloque del programa.

Columnas 1-5 Estas columnas se reservan para etiquetar una sentencia. En FORTRAN las sentencias se etiquetan mediante números enteros positivos. Estos valores no afectan al orden en que se ejecutarán las instrucciones. Además, sólo deben etiquetarse aquellas líneas que lo precisen.

Columna 6 Si una sentencia ocupa más de una línea, debe indicarse al compilador que una línea es continuación de la anterior. Esto se realiza colocando cualquier carácter diferente de blanco o cero en la sexta columna.

Columnas 7-72 En estas columnas se escribe el contenido de las sentencias propiamente dichas.

Columnas 73-80 Estas columnas no tienen significado para el compilador FORTRAN y no se deben utilizar.

3.3.2 Elementos de un programa en FORTRAN

Desde un punto de vista conceptual, el lenguaje de programación FORTRAN consta de dos tipos de elementos: los comentarios y las sentencias.

Los comentarios, como se ha mencionado anteriormente, no afectan a la forma en que se procesa el programa. Sólo representan una ayudan al usuario para su correcta interpretación.

Las sentencias están formadas por todo el conjunto de instrucciones que forman el lenguaje FORTRAN. Éstas, a su vez, se pueden clasificar en *sentencias ejecutables* y *sentencias no ejecutables*.

1. Las sentencias ejecutables son aquellas cuyo significado se utiliza esencialmente en tiempo de ejecución (fase en la que se ejecuta el programa). Se incluyen dentro de este tipo la asignación de valores a variables, las sentencias de control, y las sentencias de entrada/salida entre otras.

2. Las sentencias no ejecutables son aquellas cuyo significado se utiliza en tiempo de compilación o ensamblado. Las tareas básicas que se realizan son la declaración de módulos y variables, la reserva de espacio de memoria para vectores y la señalización de final de módulo.

En cualquier módulo escrito en FORTRAN las sentencias deben aparecer en un orden predeterminado formando tres grandes grupos. Primero deben aparecer las sentencias no ejecutables, donde se declara el inicio de módulo así como las variables, los vectores y las matrices que se utilizarán en el mismo. En segundo lugar se deben escribir las sentencias ejecutables que forman el cuerpo del módulo. Por último aparece la sentencia no ejecutable de fin de módulo (sentencia `END`).

El programa 3.1 muestra algunos de los aspectos que se han discutido hasta ahora.

```
c
c        Este programa muestra algunos aspectos de la organizacion
c        general de un programa en FORTRAN
c
c_________________________________________ Sentencias no ejecutables
      real*4 a,b,c

c_________________________________________ Sentencias ejecutables
c___Primero hay una linea etiquetada
 111  a = 12.5

c___Despues una sentencia escrita en una linea
      b = -5.1

c___Seguidamente otra sentencia escrita en mas de una linea
      c = (a + b) *
     .    1234.5398754485429387029

c___Y por ultimo se detiene la ejecucion del programa
      stop
c_________________________________________ Sentencias no ejecutables
      end
```

Prog. 3.1 Organización general de un programa en FORTRAN

3.4 Constantes y variables en FORTRAN

Una constante es un valor que no puede cambiar durante la ejecución del programa. Por el contrario, una variable es un símbolo al que se asocia un valor que puede modificarse durante la ejecución del programa. El nombre de una variable debe estar formado por una cadena de caracteres alfanuméricos, el primero de los cuales debe ser una letra.

En FORTRAN existen diferentes tipos de constantes y variables, que se diferencian según el

tipo de información que contienen. Siempre es necesario declarar, al principio de cada módulo, el tipo de todas las variables; esto se lleva a cabo mediante sentencias no ejecutables. Sin embargo, como se verá en este mismo apartado, el FORTRAN ofrece algunas ayudas que conviene aprovechar.

3.4.1 Constantes y variables enteras

Las constantes enteras son números enteros escritos sin punto decimal. Las variables enteras son símbolos que sólo pueden representar números enteros. Los ordenadores guardan el valor asignado a estas variables como una cadena de longitud finita en sistema binario. Dependiendo de la longitud de esta cadena se distinguen dos tipos de variables enteras.

`INTEGER*2` Se almacena la variable entera en dos bytes (16 bits). El número mayor en valor absoluto que se puede almacenar es 32 767 (ver capítulo 4).

`INTEGER*4` Se almacena la variable entera en cuatro bytes (32 bits). El número mayor en valor absoluto que se puede almacenar es 2 147 483 647 (ver capítulo 4).

Una de las ayudas que ofrece el FORTRAN consiste en asignar por defecto un tipo entero a todas aquellas variables que empiezan por:

```
I, J, K, L, M, N
```

Los ordenadores modernos les asignan el tipo `INTEGER*4`, mientras que algunos de los ordenadores más antiguos les asignaban `INTEGER*2`. Es *muy importante* respetar este convenio, ya que de esta forma se evitará una pérdida de tiempo considerable.

El orden de prioridad de los operadores aritméticos que actúan sobre las variables enteras es el propio del álgebra, es decir:

1. `**` (potencia)
2. `*` y `/` (producto y división respectivamente)
3. `+` y `-` (suma y diferencia respectivamente)

Si, por algún motivo, se desea alterar dicha prioridad, se deben utilizar paréntesis (ver programa 3.2).

En este programa se resaltan tres aspectos. El primero es que no sería preciso declarar las variables `MVAR1`, `MVAR2`, `NPOT` y `JRES`, puesto que por defecto ya son enteras. El segundo es que *en ningún momento* se realizan operaciones que involucren variables de tipos diferentes. El tercero es la sentencia `WRITE (6,*) NFIN`. Como se presentará más adelante, esta sentencia significa escribir en la pantalla del ordenador y con formato libre (de acuerdo con un procedimiento predefinido por el compilador FORTRAN) el valor de la variable `NFIN`.

```
c
c        Este programa muestra algunos aspectos de la utilizacion
c        de variables enteras
c
c_____________________________________________________________________
c___Declaracion de las variables enteras
      integer*2 kk
      integer*4 mvar1,mvar2,npot,jres

c___Asignacion de la variable integer*2
      kk = 13

c___Asignacion de algunas variables integer*4
      mvar1 =  20
      mvar2 = -10
      npot  =   5

c___Calculo de una variable integer*4 declarada explicitamente
      jres = (mvar1 + mvar2) * 2

c___Calculo de una variable integer*4 declarada por defecto
      nfin = jres**npot

c___Escritura por pantalla del valor de una variable
      write (6,*) nfin

      stop
      end
```

Prog. 3.2 Programación con variables enteras

3.4.2 Constantes y variables reales

Las constantes reales son números reales escritos en notación de coma fija o bien en notación científica. Las variables reales son símbolos que sólo pueden representar números reales. Al igual que pasa con las variables enteras, los ordenadores guardan el valor asignado a las variables reales como una cadena de bits de longitud finita. Dependiendo de la longitud de esta cadena se distinguen tres tipos de variables reales.

`REAL*4` (*Simple precisión*). Se almacena la variable real en cuatro bytes. El número mayor en valor absoluto que se puede almacenar es del orden de $1.7 \cdot 10^{38}$ y el número menor en valor absoluto y diferente de cero que se puede almacenar es del orden de $0.29 \cdot 10^{-38}$ (capítulo 4).

`REAL*8` (*Doble precisión*). Se almacena la variable real de ocho bytes. Los números mayor y menor en valor absoluto que se puede almacenar son del orden de $0.9 \cdot 10^{308}$ y $0.56 \cdot 10^{-308}$, respectivamente (capítulo 4).

`REAL*16` (*Cuádruple precisión*). Se almacena la variable real en dieciséis bytes. Conviene tener en cuenta que no todos los compiladores aceptan este tipo de variables. Además, el número mayor y menor en valor absoluto que se puede almacenar depende también del tipo de ordenador. En general, con este tipo de variables se logran almacenar números mayores y más próximos a cero. Así mismo, también se logra más precisión.

Por defecto, el compilador FORTRAN considera variables del tipo `REAL*4` todas aquellas variables que *no* empiezan por:

```
I, J, K, L, M, N
```

En consecuencia, todas las variables del tipo `REAL*8` o `REAL*16` se tienen que declarar explícitamente. La prioridad con que actúan los operadores aritméticos sobre las variables reales es la misma con que actúan sobre las variables enteras. De nuevo, para alterar dicha prioridad deben utilizarse los paréntesis.

En el programa 3.3 se calcula el volumen de un cilindro y el de una esfera utilizando variables reales de tipo `REAL*4` y `REAL*8`; las variables `REAL*8` se han declarado explícitamente, mientras que las variables `REAL*4` han sido declaradas por defecto. Conviene notar que en ningún caso se opera con variables de diferente tipo; es decir, las operaciones se realizan siempre entre variables de simple precisión o entre variables de doble precisión. Es importante tener presente que se trata de un ejemplo *ilustrativo* en el sentido que, en la práctica, raramente se utilizan estos dos tipos de variables en el mismo programa; es decir, habitualmente, los cálculos asociados a un determinado problema se realizan, *o bien* en simple precisión, *o bien* en doble precisión. Por otra parte, se han introducido en este ejemplo dos sentencias nuevas. La primera de ellas es la sentencia `WRITE`; la instrucción `WRITE (6,*) ' RADIO DEL CILINDRO:'` significa escribir en la pantalla la cadena de caracteres `RADIO DEL CILINDRO:`. La segunda es la sentencia `READ`; la instrucción `READ (5,*) ALTURA` significa leer del teclado el valor de la variable `ALTURA`.

Problema 3.1:
Es muy importante saber qué sucede cuando la división entre dos números enteros da por resultado un número real no entero y éste se asigna a una variable entera. Por ejemplo, en el programa que se lista a continuación, se realizan cocientes entre variables enteras (declaradas por defecto) y los resultados, que son en todos los casos valores reales no enteros, se almacenan en las variables enteras `K1` y `K2`. ¿Cuál es el valor de las variables `K1` y `K2` que aparecerá por la pantalla? ¿Qué implican estos resultados?

```
      c
      c   Division de numeros enteros
      c
      c_______________________________
      c___Asignacion de variables
            ivar1 = 1
            ivar2 = 2
            ndeno = 3

      c___Calculo de los resultados
            k1 = ivar1 / ndeno
            k2 = ivar2 / ndeno

      c___Escritura por pantalla
            write (6,*) k1
            write (6,*) k2

            stop
            end
```

●

3.4.3 Constantes y variables complejas

Las constantes complejas se describen como un par ordenado de números reales entre paréntesis y separados por una coma: `(-67.54, 0.53E-1)`. El primero representa la parte real del número complejo mientras que el segundo representa su parte imaginaria. Las variables complejas se almacenan como dos números reales contiguos (naturalmente, no se almacenan ni los paréntesis ni la coma).

Dependiendo del tipo de números reales utilizados para definir el número complejo, se tienen dos tipos de variables complejas:

`COMPLEX*8`	Las partes real e imaginaria se representan mediante un `REAL*4`.
`COMPLEX*16`	Las partes real e imaginaria se representan mediante un `REAL*8`.

No existen variables complejas que permitan almacenar la parte real y la parte imaginaria mediante números reales de diferente tipo. La prioridad con que actúan los operadores aritméticos sobre las variables complejas es la misma con que actúan sobre los tipos de variables que se han visto hasta el momento. Por otro lado es importante notar que, para este tipo de variables, las operaciones aritméticas tienen significados (definiciones) *diferentes* de los que tenían para los anteriores tipos de variables.

En el programa 3.4 se manipulan variables complejas; como ejemplos de operaciones que se pueden realizar con ellas se han utilizado varias funciones propias del FORTRAN como son `CONJG()`, `REAL()` y `AIMAG()`, que permiten obtener el complejo conjugado, la parte real y la parte imaginaria de un número complejo respectivamente.

```
c
c        Este programa muestra algunos aspectos de la utilizacion
c        de variables reales
c
c-----------------------------------------------------------------------
c___Declaracion de las variables real*8
      real*8 radio2, vol2, pi2

c___Asignacion de las variables pi y pi2
      pi  = 3.141592653589793
      pi2 = 3.141592653589793d0

c___Introduccion desde teclado
c___de los datos para calculo del volumen de un cilindro
      write (6,*) 'radio del cilindro:'
      read (5,*) radio
      write (6,*) 'altura del cilindro:'
      read (5,*) altura

c___Calculo del volumen del cilindro en simple precision
      vol = pi * radio * radio * altura

c___Introduccion desde teclado
c___de los datos para calculo del volumen de una esfera
      write (6,*) 'radio de la esfera:'
      read (5,*) radio2

c___Calculo del volumen de la esfera en doble precision
      vol2 = (4.d0 * pi2 * radio2 * radio2 * radio2) / 3.d0

c___Escritura de resultados
      write (6,*) 'volumen del cilindro (simple precision)=', vol
      write (6,*) 'volumen de la esfera (doble precision)=', vol2

      stop
      end
```

Prog. 3.3 Programación con variables reales

3.4.4 Constantes y variables lógicas

A las constantes y variables lógicas sólo se les pueden asignar dos valores distintos, que deben aparecer siempre entre puntos:

`.TRUE.`	que significa verdadero
`.FALSE.`	que significa falso

```
c
c        Este programa muestra como utilizar variables complejas
c
c_____________________________________________________________________
c___Declaracion de las variables complejas
      complex*8 z1,z2,z3

c___Asignacion de una variable compleja
      z1 = ( 2.0 , 3.0 )

c___Calculo del conjugado de una variable compleja
      z2 = conjg(z1)

c___Calculo del producto de dos numeros complejos
      z3=z1*z2

c___Calculo de las partes reales e imaginarias
      x1 = real  (z1)
      y1 = aimag (z1)
      x2 = real  (z2)
      y2 = aimag (z2)
      x3 = real  (z3)
      y3 = aimag (z3)

c___Escritura por pantalla de los resultados
      write (6,*) ' z1 =', z1
      write (6,*) ' z2 =', z2
      write (6,*) ' z3 =', z3

      write (6,*) ' x1 y y1 =', x1 , y1
      write (6,*) ' x2 y y2 =', x2 , y2
      write (6,*) ' x3 y y3 =', x3 , y3

      stop
      end
```

Prog. 3.4 Programación con variables complejas

Las variables lógicas se deben declarar siempre mediante la sentencia `LOGICAL`. Como puede observarse en la tabla 3.1, los operadores lógicos se pueden clasificar en dos grupos:

- *operadores lógicos de relación,* que permiten comparar expresiones aritméticas dando como resultado una variable lógica.

- *operadores lógicos específicos,* que permiten operar con variables lógicas; también generan como resultado una variable lógica.

Como puede observarse, tanto las constantes lógicas como los operadores lógicos deben escribirse precedidos y seguidos de un punto (.). En caso de no observarse esta regla se producirá un error de compilación. En el programa 3.5 se muestran varios de los aspectos que se han comentado anteriormente.

Tabla 3.1 Operadores lógicos

OPERADORES LÓGICOS			
DE RELACIÓN		**ESPECÍFICOS**	
`.EQ.`	$=$	`.NOT.`	negación
`.NE.`	$\neq$	`.AND.`	y
`.LT.`	$<$	`.OR.`	o
`.LE.`	$\leq$	`.XOR.`	o exclusivo
`.GT.`	$>$		
`.GE.`	$\geq$		

3.4.5 Constantes y variables alfanuméricas

Las constantes alfanuméricas, como su propio nombre indica, están formadas por cadenas de caracteres (letras y/o números) y se introducen siempre delimitadas por comillas simples, por ejemplo: `'ESTO ES UNA CADENA DE DIGITOS'`. Conviene notar que los espacios en blanco también forman parte de la cadena.

Al principio de cada módulo es siempre necesario declarar todas las variables alfanuméricas. Además se debe especificar su longitud máxima (el número máximo de caracteres que pueden contener). Dicha declaración se realiza mediante la sentencia `CHARACTER*n`, donde `n` es un número entero que representa la longitud de la cadena.

Una de las operaciones más usuales de las que se realizan con variables alfanuméricas es la *concatenación*; es decir, la unión de dos o más variables alfanuméricas para formar una única cadena. Esta operación se representa mediante el símbolo `//`.

En el programa 3.6 se muestra cómo se opera con este tipo de variables. Puede observarse que la variable `CAD_TOTAL` se ha declarado con una longitud máxima igual a la suma de las longitudes con que se declaran las variables `CAD1`, `CAD2` y `CAD3`.

```
c
c        Este programa muestra como utilizar variables logicas
c
c_____________________________________________________________________
c___Declaracion de las variables logicas
      logical flag1,flag2,flag3

c___Asignacion de variables logicas
      flag1 = .true.
      flag2 = .false.

c___Operaciones con variables logicas
      flag3 = flag1 .and. flag2

c___Escritura por pantalla
      write (6,*) flag1
      write (6,*) flag2
      write (6,*) flag3

      stop
      end
```

Prog. 3.5 Programación con variables lógicas

3.4.6 Sentencia IMPLICIT

Con el propósito de simplificar la programación, en muchos casos resulta conveniente declarar del mismo tipo todas las variables que empiezan por una determinada letra. Esto puede realizarse mediante la sentencia:

`IMPLICIT` *tipo_de_variable lista_de_letras*

Así por ejemplo, la sentencia:

```
IMPLICIT COMPLEX*8 Z
```

obliga a que todas las variables que empiecen por la letra `Z` sean del tipo `COMPLEX*8`. Del mismo modo, la sentencia:

```
IMPLICIT REAL*8 (A-H,O-Z)
```

obliga a que, por defecto, todas las variables que empiecen por una letra comprendida entre la `A` y la `H` o entre la `O` y la `Z` sean del tipo `REAL*8`.

```
c
c        Este programa muestra como utilizar variables alfanumericas
c
c_____________________________________________________________________
c___Declaracion de las variables alfanumericas
      character*1 espacio
      character*5 cad1,cad2
      character*7 cad3
      character*19 cad_total

c___Asignacion de variables alfanumericas
      espacio = ' '
      cad1    = 'seat'
      cad2    = 'panda'
      cad3    = '16v GTI'

c___Operacion con variables alfanumericas
      cad_total = cad1//espacio//cad2//espacio//cad3

c___Escritura por pantalla de los resultados
      write (6,*) cad_total

      stop
      end
```

Prog. 3.6 Programación con variables alfanuméricas

3.5 Funciones en FORTRAN

En FORTRAN se pueden distinguir dos clases de funciones:

1. *Funciones intrínsecas.* Son las funciones que incorpora directamente el compilador. Aunque la mayoría de los nombres asociados a cada una de ellas es estándar, cada fabricante añade algunas funciones propias. Para cada compilador se debe consultar el manual correspondiente.

2. *Funciones de usuario.* Estas funciones las define el propio usuario (ver capítulo 5).

En los programas mostrados hasta el momento se han utilizado varias funciones intrínsecas (`REAL()`, `AIMAG()` o `CONJG()`, entre otras). Como se puede observar en los listados, las asignaciones del tipo $y = f(x)$ se realizan en FORTRAN de forma similar a como se realizan en matemáticas. Conviene notar, sin embargo, que en las asignaciones FORTRAN las variables x e y deben ser del mismo tipo.

3.6 Sentencias de entrada–salida en FORTRAN

En este apartado se exponen los aspectos esenciales de las sentencias asociadas a la lectura y escritura de datos de un archivo.

En FORTRAN, el proceso de lectura (o escritura) se puede dividir en tres fases:

1. Primero, se asigna al archivo un número denominado *unidad lógica*. A partir de este momento, cualquier referencia a dicho archivo se realiza a través de la unidad lógica. Como se comentará seguidamente, también se definen en esta fase algunas propiedades del fichero que se desea utilizar. Esta operación se denomina *abrir* el archivo y se realiza mediante la instrucción `OPEN`.

2. A continuación se realizan todas las operaciones de lectura y escritura. Las sentencias que se utilizan son `READ`, `WRITE` y `FORMAT`.

3. Finalmente, hay que romper la asignación fichero–unidad lógica. Esta operación se denomina *cerrar* el archivo y se realiza mediante la instrucción `CLOSE`.

La sentencia `OPEN` debe aparecer antes de leer o escribir en el archivo; así mismo, debe escribirse una instrucción `CLOSE` después de acceder por última vez al fichero. La sintaxis de estas dos instrucciones es la siguiente:

```
OPEN ( UNIT = unidad_lógica, FILE = nombre_de_fichero,
                    STATUS = { 'NEW'     } )
                             { 'OLD'     }
                             { 'UNKNOWN' }
CLOSE ( unidad_lógica )
```

donde:

- *unidad_lógica* es una constante o variable entera positiva que representa al fichero. Se puede utilizar cualquier valor entero excepto: 1) el `5`, que representa, por defecto, al teclado y 2) el `6`, que representa, también por defecto, a la pantalla. Naturalmente, la unidad `5` sólo se utiliza en instrucciones de lectura y la `6` sólo en instrucciones de escritura.

- *nombre_de_fichero* es una cadena alfanumérica y contiene el nombre del fichero que se desea manipular.

- `STATUS` representa el estado en que se encuentra el archivo: `NEW` indica que el archivo no existe y hay que crearlo; `OLD` indica que el archivo ya existe y no se debe crear y `UNKNOWN` indica que si el fichero existe sólo se debe utilizar mientras que si no existe hay que crearlo.

La sintaxis de las sentencias `READ` y `WRITE` es la siguiente:

`READ (` `UNIT =` *unidad_lógica,* `FORMAT =` *formato* `)` *lista_de_variables*
`WRITE (` `UNIT =` *unidad_lógica,* `FORMAT =` *formato* `)` *lista_de_variables*

donde:

- *unidad_lógica* es un número entero positivo que representa al fichero con el que se desea operar. Debe ser el mismo que aparece en la sentencia `OPEN` correspondiente.

- *formato* es la etiqueta de una línea (por tanto, un entero positivo) en la que se especifica cómo se desea leer o escribir el contenido de las variables que se detallan en *lista_de_variables*. Como se ha comentado anteriormente, si en lugar de una etiqueta aparece un `*`, la lista de variables se escribirá en formato libre.

- *lista_de_variables* es la lista de las variables que hay que leer o escribir; los nombres de las variables deben estar separados por comas.

La sintaxis de la sentencia `FORMAT` es la siguiente

etiqueta `FORMAT (` *lista_de_formato* `)`

donde:

- *etiqueta* es la etiqueta de la línea donde se encuentra la instrucción `FORMAT`.

- *lista_de_formato* es una lista formada por un conjunto de símbolos que indican cómo se desea leer o escribir la lista de variables que aparece en la correspondiente sentencia `READ` o `WRITE`; a cada variable que aparece dicha lista le corresponde un símbolo en la lista de formato.

Los símbolos que pueden aparecer en *lista_de_formato* son de dos tipos: *símbolos de composición* y *símbolos de variables*. Entre los primeros, los dos más usuales son:

`/` Significa saltar a la línea siguiente del fichero.

`nX` Significa dejar `n` espacios en blanco.

Entre los segundos, los más utilizados son:

`nIm` Significa leer o escribir `n` números enteros de `m` dígitos cada uno de ellos.

`nFm.d` Significa leer o escribir `n` números reales de simple precisión, expresados como parte entera y parte decimal, de `m` dígitos cada uno de ellos, de los cuales `d` se destinan a la parte fraccionaria.

`nEm.d` Significa leer o escribir `n` números reales de simple precisión, expresados en notación científica, de `m` dígitos cada uno de ellos, de los cuales `d` se destinan a la parte fraccionaria.

`nDm.d` Significa leer o escribir `n` números reales de doble precisión, expresados en notación científica, de `m` dígitos cada uno de ellos, de los cuales `d` se destinan a la parte fraccionaria.

`nAm` Significa leer o escribir `n` variables alfanuméricas de `m` dígitos cada una de ellas.

`nL` Significa leer o escribir `n` variables lógicas.

El programa 3.7 permite calcular las raíces de una ecuación de segundo grado con coeficientes reales. Este programa producirá un error de ejecución cuando el coeficiente cuadrático sea nulo o bien las raíces sean complejas (compruébese). Tal como se puede ver en las instrucciones `READ` y `WRITE` correspondientes, los datos se leen del teclado, mientras que los resultados (dos cadenas alfanuméricas y dos números reales en simple precisión) se escriben en el fichero `RESUL.RES`. Por otra parte, puede observarse también que el fichero de resultados se ha abierto con `STATUS='NEW'` puesto que no existía anteriormente.

Problema 3.2:
Qué sucede si el programa 3.7 se ejecuta dos veces consecutivas? ¿Por qué? ¿Cómo debería modificarse la sentencia `OPEN` para evitar este inconveniente?

●

3.7 Sentencias de control en FORTRAN

En este apartado se presentan tres sentencias destinadas a controlar el orden con que se ejecutan las instrucciones:

- La sentencia condicional `IF`
- La sentencia `GO TO`
- El bloque `DO - ENDDO`

3.7.1 La sentencia IF

En este subapartado se comentan las tres sintaxis más utilizadas de la sentencia `IF`. La primera de ellas, y también la más sencilla, es la siguiente:

`IF (` *condición* `)` *sentencia*

donde *condición* es una proposición lógica y *sentencia* es una instrucción que sólo se ejecutará si la condición lógica es cierta; si la condición es falsa, se saltará a la siguiente línea del programa.

```
c
c        Este programa calcula las raices de una ecuacion de segundo
c        grado. Todas las variables son del tipo REAL*4 y estan
c        declaradas por defecto
c
c-----------------------------------------------------------------------
c___Entrada de los coeficientes desde teclado
      write (6,*) ' Entra el coeficiente cuadratico:'
      read (5,*) a
      write (6,*) ' Entra el coeficiente lineal:'
      read (5,*) b
      write (6,*) ' Entra el coeficiente independiente:'
      read (5,*) c

c___Calculo de las raices
      dis = b*b - 4.*a*c

      x1 = (-1.0*b + sqrt(dis)) / (2.0 * a)
      x2 = (-1.0*b - sqrt(dis)) / (2.0 * a)

c___Escritura de resultados en un archivo
      open (unit=12, file='resul.res', status='new')

      write (12,100) ' Primera solucion = ', x1
      write (12,100) ' Segunda solucion = ', x2

 100  format (2x,a20,2x,e15.8)

      close (12)

      stop
      end
```

Prog. 3.7 Programación con instrucciones de entrada y salida

La segunda estructura de la sentencia `IF` está formada por el siguiente bloque:

```
IF ( condición ) THEN
        bloque
ENDIF
```

En este caso, si la proposición lógica *condición* es cierta, se ejecutarán todas las instrucciones contenidas en las líneas *bloque*; en caso contrario, se ejecutará la línea siguiente a la instrucción `ENDIF`.

La tercera estructura de la sentencia `IF` es:

```
IF ( condición ) THEN
        primer_bloque
ELSE
        segundo_bloque
ENDIF
```

En este caso, si la condición es verdadera se ejecuta el primer bloque de sentencias y a continuación la línea siguiente a la instrucción `ENDIF`; si la condición es falsa se ejecuta el segundo bloque de instrucciones.

El programa 3.8 es una ampliación del programa 3.7 donde se han introducido dos sentencias `IF` con dos sintaxis distintas. Este programa permite detectar si el coeficiente cuadrático es demasiado próximo a cero. Así mismo, se distinguen los casos en que la ecuación de segundo grado tiene raíces complejas.

3.7.2 La sentencia GO TO

Esta sentencia transfiere el control del programa a una línea determinada. Existen tres formas diferentes de utilizar la sentencia `GO TO`. Dos de ellas se mantienen por compatibilidad con las versiones anteriores del FORTRAN y están claramente en desuso. La sintaxis más utilizada es:

```
GO TO  etiqueta
```

donde *etiqueta* es la etiqueta de la línea que se ejecutará inmediatamente a continuación de la sentencia `GO TO`.

En el programa 3.9 se ha modificado el bloque inicial del programa 3.8 a fin de ilustrar la utilización de la sentencia `GO TO`. Cuando el coeficiente cuadrático es inferior, en valor absoluto, a una cierta tolerancia, se escribe un mensaje en la pantalla del ordenador y se pide que se introduzca un nuevo valor.

```
c
c        Este programa calcula las raices de una ecuacion de segundo
c        grado.
c
c_____________________________________________________________________
      complex*8 z1,z2

c___Entrada de los coeficientes desde teclado
      tol = 1.0e-5
      write (6,*) ' Entra el coeficiente cuadratico:'
      read (5,*) a
      if (abs(a).lt.tol) then
         write (6,*) ' El coeficiente cuadratico debe ser mayor'
         stop
      endif
      write (6,*) ' Entra el coeficiente lineal:'
      read (5,*) b
      write (6,*) ' Entra el coeficiente independiente:'
      read (5,*) c

      open (unit=12,file='resul.res',status='new')

c___Calculo y escritura de las raices
      dis = b*b - 4.*a*c
      if (dis.ge.0.) then
         x1 = (-1.0*b + sqrt(dis)) / (2.0 * a)
         x2 = (-1.0*b - sqrt(dis)) / (2.0 * a)
         write (12,100) ' Primera solucion = ', x1
         write (12,100) ' Segunda solucion = ', x2
      else
         z1 = (cmplx(-1.0*b) - csqrt (cmplx (dis)))/ (cmplx(2.0*a))
         z2 = (cmplx(-1.0*b) + csqrt (cmplx (dis)))/ (cmplx(2.0*a))
         write (12,200) ' Primera solucion = ', z1
         write (12,200) ' Segunda solucion = ', z2
      endif

 100  format (2x,a20,2x,e15.8)
 200  format (2x,a20,2x,'(',e15.8,','e15.8,')')

      close (12)
      stop
      end
```

Prog. 3.8 Utilización del bloque IF-ENDIF

```
c
c        Este programa calcula las raices de una ecuacion de segundo
c        grado.
c
c_____________________________________________________________________
c___Declaracion de variables
      complex*8 z1,z2

c___Entrada de los coeficientes desde teclado
      tol = 1.0e-5
  10  write (6,*) ' Entra el coeficiente cuadratico:'
      read (5,*) a
      if (abs(a).lt.tol) then
         write (6,*) ' El coeficiente cuadratico debe ser mayor'
         go to 10
      endif
      write (6,*) ' Entra el coeficiente lineal:'
      read (5,*) b
      write (6,*) ' Entra el coeficiente independiente:'
      read (5,*) c

                .
                .
                .
```

Prog. 3.9 Utilización de la sentencia `GO TO`

3.7.3 El bloque DO – ENDDO

El bloque `DO-ENDDO` se emplea para ejecutar cíclicamente un conjunto de instrucciones. Su sintaxis es:

```
DO  ivar = n1, n2, n3
        bloque
ENDDO
```

donde:

- *ivar* es la variable entera que controla el bucle.
- n_1 es el valor inicial de la variable *ivar*.

- n_2 es el valor final de la variable *ivar*.

- n_3 es el valor del incremento de la variable *ivar*. Si este valor no aparece, por defecto la variable n_3 vale 1.

Las sentencias contenidas entre las instrucciones `DO` y `ENDDO` se ejecutan variando el valor de *ivar* desde n_1 hasta n_2 e incrementándolo en n_3 a cada paso por el bucle.

Una sintaxis alternativa es:

```
      DO  etiqueta ivar = n1, n2, n3
                bloque
etiqueta  CONTINUE
```

donde *ivar*, n_1, n_2 y n_3 tienen el mismo significado que antes y *etiqueta* es una etiqueta que marca el principio y el final del bucle.

Un bucle `DO-ENDDO` siempre inicia su ejecución mediante la sentencia `DO`. Es decir, no es posible utilizar una instrucción `GO TO` dirigida a una sentencia situada en el interior de un bucle. Sin embargo, es posible salir prematuramente de un bucle, es decir, antes de que la variable *ivar* haya alcanzado su valor máximo. Pueden utilizarse estructuras `DO-ENDDO` anidadas siempre que el bucle externo incluya todas las instrucciones del bucle interno (ver figura 3.2).

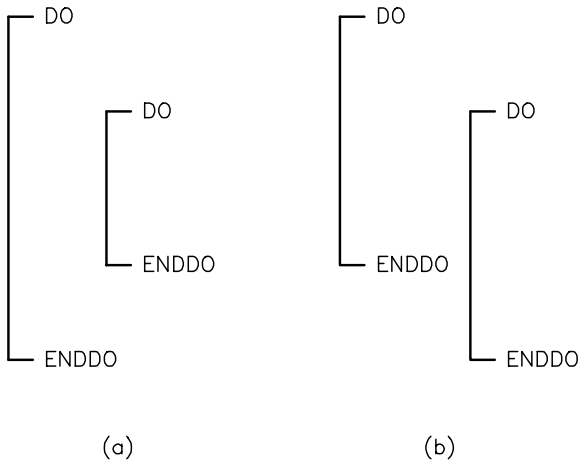


Fig. 3.2 a) estructura correcta, y b) estructura incorrecta

```
c
c        Este programa realiza una tabla de la funcion sin(x) entre
c        dos valores de x especificados por el usuario.
c
c_____________________________________________________________________
c___Declaracion de variables
      implicit real*8 (a-h,o-z)
      character*20 file_out

c___Entrada de los datos
      write (6,*) ' limite inferior:'
      read (5,*) xinf
      write (6,*) ' limite superior:'
      read (5,*) xsup
      write (6,*) ' numero de intervalos:'
      read (5,*) inter
      write (6,*) ' nombre del archivo:'
      read (5,'(a20)') file_out

c___Realizacion de la tabla
      open (unit=30,file=file_out,status='new')

      write (30,50)

      xpas = (xsup - xinf) / dfloat(inter)

      do i=1,inter+1
         x = xinf + dble(i-1)*xpas
         y = dsin (x)
         write (30,100) x,y
      enddo

  50  format (8x,' X ',10x,' SIN(X) '/)
 100  format (2(2x,d13.5))

      close (30)

      stop
      end
```

Prog. 3.10 Utilización del bloque DO-ENDDO

En el programa 3.10 se muestra cómo realizar una tabla de la función $f = \sin(x)$ entre dos valores escogidos por el usuario. En este caso no se especifica el valor de la variable n_3, por lo

que el incremento de la variable de control `I` vale 1. Otra novedad consiste en la utilización de una variable alfanumérica que contiene el nombre del archivo de resultados.

En la tabla 3.2 se muestra el fichero de resultados que se obtiene cuando el límite inferior vale 0, el límite superior vale 3.141592 y el número de intervalos es 10.

Tabla 3.2 Fichero de resultados

X	SIN(X)
.00000D+00	.00000D+00
.31416D+00	.30902D+00
.62832D+00	.58779D+00
.94248D+00	.80902D+00
.12566D+01	.95106D+00
.15708D+01	.10000D+01
.18850D+01	.95106D+00
.21991D+01	.80902D+00
.25133D+01	.58779D+00
.28274D+01	.30902D+00
.31416D+01	.65359D-06

Problema 3.3:
Escribir un programa FORTRAN que, dado el valor de un cierto radio r (entero y menor o igual que 100), calcule:

1. Todos los puntos de coordenadas enteras que estén sobre la circunferencia $x^2 + y^2 = r^2$
2. Todos los puntos de coordenadas enteras que estén dentro de la circunferencia $x^2 + y^2 = r^2$

Los resultados se deben mostrar por pantalla y guardar en un archivo. ●

Problema 3.4:
Se puede demostrar que al evaluar un polinomio $p_n(x)$ mediante la expresión:

$$p_n(x) = \sum_{i=0}^{n} a_i \, x^i$$

se deben realizar $n(n+1)/2$ multiplicaciones y n sumas. La *regla de Horner* evalúa el mismo polinomio de la siguiente forma:

$$p_n(x) = \Bigg(\Big(\big((a_n x + a_{n-1})x + a_{n-2} \big) x + a_{n-3} \Big) \ldots \Bigg) x + a_0$$

que sólo requiere n multiplicaciones y n sumas. Se pide escribir un programa en FORTRAN que construya, utilizando la regla de Horner, una tabla de valores del polinomio:

$$p_4(x) = 2x^4 - 20x^3 + 70x^2 - 100x + 48$$

para valores de x en el intervalo $[-4, -1]$, con saltos de x de valor $\Delta x = 0.5$.

●

3.8 Bibliografía

Borse, G. J. *Programación FORTRAN77 con aplicaciones de cálculo numérico en ciencias e ingeniería.* Anaya, 1989.

Ellis, T.M.R. *FORTRAN 77 Programming.* Addison–Wesley Publishing Company, 1990.

García Merayo, F. *Programación en FORTRAN 77.* Paraninfo, 1988.

Lignelet, P. *FORTRAN 77. Lenguaje FORTRAN V.* Masson S.A., 1987.

Mouro, D.M. *FORTRAN 77.* Edward Arnold Ed., 1982.

4 Número, algoritmo y errores

Objetivos

- Definir la expresión general de un número en una base de numeración.
- Comentar las bases de numeración más utilizadas.
- Detallar cómo se almacenan los números enteros y reales en un ordenador.
- Introducir los conceptos de *overflow* y *underflow*.
- Presentar los tipos de errores en cálculo numérico: errores de redondeo, errores de truncamiento y errores inherentes.
- Discutir la noción de cifras significativas correctas.
- Estudiar la propagación del error en las operaciones aritméticas elementales y en una secuencia de operaciones.

4.1 Introducción

En este capítulo se presentarán dos conceptos básicos que aparecerán posteriormente en la mayoría de las aplicaciones de los métodos numéricos.

El primero es la representación de un número en el ordenador. Para ello se estudiará cómo el ordenador almacena los números enteros y los números reales. La característica más importante es que siempre se utiliza un número *finito* de dígitos para su representación. Por tanto, resulta imposible almacenar números como $\pi = 3.14159265358979323846264 3\ldots$ exactamente mediante una cantidad finita de cifras. Esta restricción implica el concepto de *error de redondeo*,

es decir, el error que se comete al almacenar un número mediante una cadena finita de dígitos, cuando para almacenarlo exactamente se precisan muchos más (o infinitos).

El segundo es el concepto de algoritmo. Como se comenta más adelante, un algoritmo debe estar formado por un número *finito* de instrucciones. Sin embargo, existen procesos que requieren un número infinito de pasos. Por ejemplo, el cálculo exacto de la exponencial de un número mediante la serie de Taylor requiere infinitos sumandos

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$$

Es evidente que resulta imposible calcularlos todos. Por tanto, sólo se sumarán unos cuantos términos y, consecuentemente, sólo se obtendrá una aproximación a la exponencial. El error que se comete en los cálculos al truncar el proceso infinito se denomina *error de truncamiento*.

Por último, en este capítulo se analiza cómo estos errores influyen en los cálculos posteriores. Es decir, se estudia la propagación de los errores en los métodos numéricos.

4.2 Número

Antes de presentar cómo se almacenan los números en el ordenador es necesario repasar algunos conceptos básicos. El primero es la representación de un número en una base de numeración. Para representar cualquier cantidad en una cierta base de numeración n, se precisan n dígitos diferentes (en general: $0, 1, 2, \ldots, n-1$). En la tabla 4.1 se muestran las bases de numeración más utilizadas en el ámbito de los ordenadores.

La representación general de un número en una cierta base de numeración n, si se designa por $\mathtt{d}_i$ el dígito situado en la i–ésima posición, está definida por la siguiente expresión

$$\begin{aligned}&\left(\mathtt{d_p\, d_{p-1} \ldots d_2\, d_1\, d_0 \cdot d_{-1}\, d_{-2}\, d_{-3} \ldots d_{-(q-1)}\, d_{-q}}\right)_n = \\ &\mathtt{d_p}\, n^{\mathtt{p}} + \mathtt{d_{p-1}}\, n^{\mathtt{p-1}} + \ldots \mathtt{d_2}\, n^{\mathtt{2}} + \mathtt{d_1}\, n^{\mathtt{1}} + \mathtt{d_0}\, n^{\mathtt{0}} + \\ &\mathtt{d_{-1}}\, n^{\mathtt{-1}} + \mathtt{d_{-2}}\, n^{\mathtt{-2}} + \mathtt{d_{-3}}\, n^{\mathtt{-3}} \ldots \mathtt{d_{-(q-1)}}\, n^{\mathtt{-(q-1)}} + \mathtt{d_{-q}}\, n^{\mathtt{-q}}\end{aligned} \tag{4.1}$$

Así por ejemplo, los números $(\mathtt{745.863})_{10}$ y $(\mathtt{101.011})_2$ significan respectivamente

$$\left(\mathtt{745.863}\right)_{10} = \mathtt{7}\cdot 10^{2} + \mathtt{4}\cdot 10^{1} + \mathtt{5}\cdot 10^{0} + \mathtt{8}\cdot 10^{-1} + \mathtt{6}\cdot 10^{-2} + \mathtt{3}\cdot 10^{-3}$$

y

$$\left(\mathtt{101.011}\right)_{2} = \mathtt{1}\cdot 2^{2} + \mathtt{0}\cdot 2^{1} + \mathtt{1}\cdot 2^{0} + \mathtt{0}\cdot 2^{-1} + \mathtt{1}\cdot 2^{-2} + \mathtt{1}\cdot 2^{-3}$$

Tabla 4.1 Bases de numeración

BASES DE NUMERACIÓN			
DECIMAL base 10	**BINARIA** base 2	**OCTAL** base 8	**HEXADECIMAL** base 16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

La forma concreta de almacenar los números depende ligeramente del tipo de ordenador y del lenguaje de programación utilizado, pero siempre se almacenan en el sistema binario (base dos). Aunque la expresión 4.1 permite representar tanto números enteros (parte fraccionaria nula) como reales, ambos tipos de números de almacenan de forma distinta en el ordenador.

4.2.1 Almacenamiento de los números enteros

Los números enteros, representados en el sistema binario, se almacenan en `S` posiciones (bits), de las cuales se reserva una para indicar el signo del número (véase la figura 4.1):

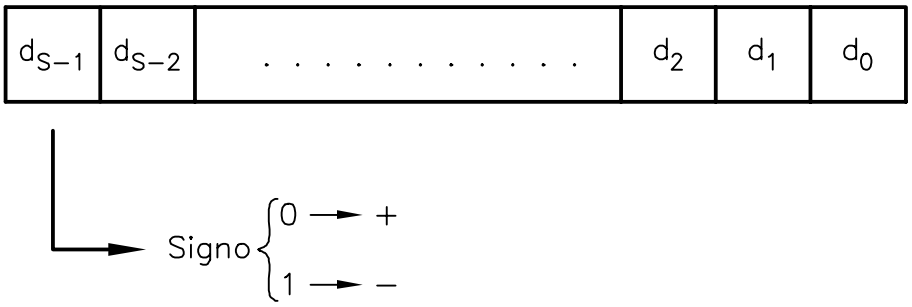


Fig. 4.1 Almacenamiento de un número entero

De acuerdo con la expresión 4.1, el valor del número entero es

$$\left(\pm \mathtt{d}_{\mathtt{S-2}} \ldots \mathtt{d}_2\, \mathtt{d}_1\, \mathtt{d}_0 \right)_2 \;=\; \pm\left(\mathtt{d}_{\mathtt{S-2}} 2^{\mathtt{S-2}} + \ldots + \mathtt{d}_2 2^2 + \mathtt{d}_1 2^1 + \mathtt{d}_0 2^0 \right)$$

En estas condiciones, el número entero máximo (en valor absoluto) que se puede almacenar se obtiene para $\mathtt{d}_{\mathtt{S-2}} = \cdots = \mathtt{d}_1 = \mathtt{d}_0 = 1$ (véase la figura 4.2)

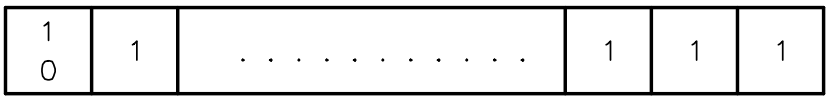


Fig. 4.2 Número entero máximo (en valor absoluto)

y su valor es

$$\left| N_{máx} \right| \;=\; 1 \cdot 2^{\mathtt{S-2}} + \ldots + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \;=\; \frac{1 - 2^{\mathtt{S-1}}}{1 - 2} \;=\; 2^{\mathtt{S-1}} - 1 \qquad (4.2)$$

Así pues, utilizando `S` posiciones pueden guardarse todos los enteros comprendidos entre $-(2^{\mathtt{S-1}} - 1)$ y $2^{\mathtt{S-1}} - 1$. En particular, el número entero no nulo más próximo a cero que puede almacenarse es lógicamente ± 1 (véase la figura 4.3):

$$\left| N_{mín} \right| \;=\; 1 \cdot 2^0 \;=\; 1 \qquad (4.3)$$

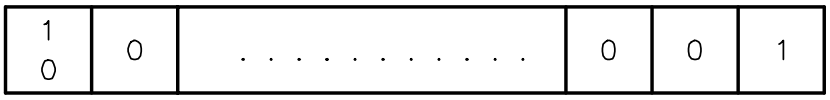


Fig. 4.3 Número entero no nulo mínimo (en valor absoluto)

A partir de la expresión 4.2 es fácil comprobar que para una variable `INTEGER*2` de FORTRAN (`S` $= 16$ posiciones para almacenar el número entero), el número máximo (en valor absoluto) es `32 767`. Así mismo, para las variables `INTEGER*4` (`S` $= 32$ posiciones) el número máximo (en valor absoluto) es `2 147 483 647`. Es importante remarcar que estas limitaciones en la capacidad de almacenar números está asociada a la utilización de una *aritmética finita*.

4.2.2 Almacenamiento de los números reales

El almacenamiento de los números reales podría realizarse mediante su representación general en una cierta base n (ver expresión 4.1). A este tipo de representación se le denomina *coma fija*. Sin embargo, los ordenadores almacenan los números reales en una representación denominada *coma flotante*. En ella, el número se representa mediante una *mantisa* (fraccionaria) y un *exponente* (entero). El número real es la mantisa multiplicada por la base de numeración elevada al exponente. Por ejemplo, el número $(\texttt{891.246})_{10}$ se representa como $(\texttt{0.891246})_{10} \cdot 10^{\texttt{3}}$ (mantisa `0.891246`, exponente `3`) y el número $(\texttt{0.00753})_{10}$ se escribe como $(\texttt{0.753})_{10} \cdot 10^{\texttt{-2}}$ (mantisa `0.753`, exponente -2). Como puede observarse, el punto decimal se desplaza (flota) hasta que la parte entera es nula y el primer dígito decimal es siempre diferente de cero. Esta misma representación puede utilizarse en cualquier base de numeración. Por ejemplo, si se utiliza el sistema binario el número $(\texttt{101.001})_2$ es igual a $(\texttt{0.101001})_2 \cdot 2^{(\texttt{11})_2}$.

Todos los ordenadores almacenan los números reales en base dos y en coma flotante. La diferencia entre un tipo de ordenador y otro reside, por una parte, en el número de dígitos (bits en el sistema binario) que se reserva para la mantisa y el exponente y por otra, en el orden en que éstos se almacenan. En cualquier caso, se puede suponer que el ordenador almacena los números reales en:

- Base dos.
- Coma flotante.
- Con `M` posiciones para la mantisa (una de ellas, para el signo de la mantisa).
- Con `E` posiciones para el exponente (una de ellas, para el signo del exponente).

Este tipo de almacenamiento se representa gráficamente en la figura 4.4. Puede observarse que se reservan dos posiciones para los signos (el de la mantisa y el del exponente).

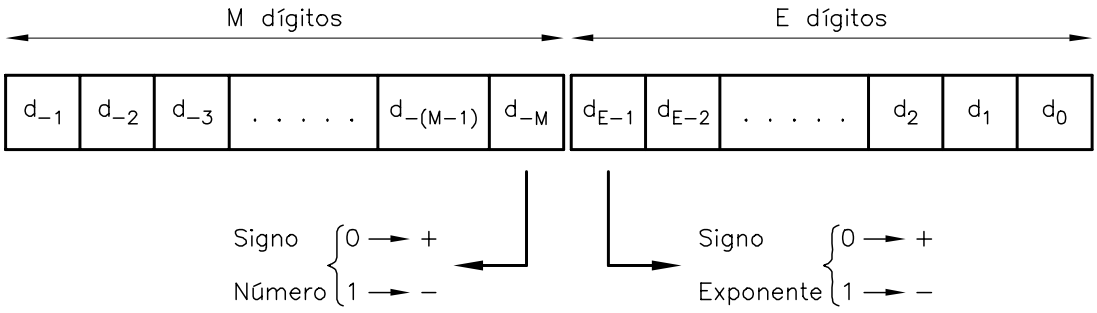


Fig. 4.4 Almacenamiento de un número real

Teniendo en cuenta la expresión 4.1 y la representación en coma flotante, el valor del número real es

$$\left(\pm .\mathtt{d_{-1}\,d_{-2}\,d_{-3}\,\ldots\,d_{-(M-1)}}\right)_2 \; 2^{\pm(\mathtt{d_{E-2}\,\ldots\,d_2\,d_1\,d_0})_2} \;=$$

$$\pm\left(\mathtt{d_{-1}}2^{-1} + \mathtt{d_{-2}}2^{-2} + \ldots + \mathtt{d_{-(M-1)}}2^{-(M-1)}\right) \; 2^{\pm(\mathtt{d_{E-2}}2^{E-2} + \ldots + \mathtt{d_1}2^1 + \mathtt{d_0}2^0)}$$

Mediante esta representación, el número real mayor (en valor absoluto) que se puede almacenar se obtiene a partir de la mantisa máxima y el exponente máximo (véase la figura 4.5):

$$\begin{aligned}
\left|N_{máx}\right| &= \left[1 \cdot 2^{-1} + 1 \cdot 2^{-2} + \ldots + 1 \cdot 2^{-(M-1)}\right] 2^{\left[1 \cdot 2^{E-2} + \ldots + 1 \cdot 2^1 + 1 \cdot 2^0\right]} \\
&= \left[\left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^2 + \ldots + \left(\frac{1}{2}\right)^{(M-1)}\right] 2^{\left[2^{(E-1)}-1\right]} \qquad (4.4) \\
&= \frac{1}{2}\left[\frac{1-(1/2)^{(M-1)}}{1-(1/2)}\right] 2^{\left[2^{(E-1)}-1\right]} = \left(1 - 2^{(1-M)}\right) 2^{\left[2^{(E-1)}-1\right]}
\end{aligned}$$

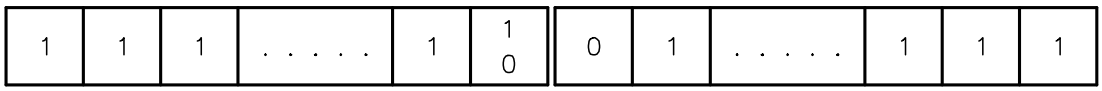


Fig. 4.5 Número real máximo (en valor absoluto)

Por otro lado, el número real no nulo más próximo a cero que puede almacenarse se obtiene para la mantisa mínima y el máximo exponente con signo negativo, como muestra la figura 4.6:

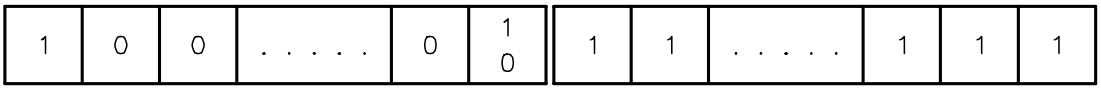


Fig. 4.6 Número real no nulo mínimo (en valor absoluto)

y su valor es

$$\begin{aligned} \left|N_{mín}\right| &= \left[1 \cdot 2^{\mathtt{-1}}\right] 2^{-\left[1\cdot 2^{\mathtt{E-2}} + \ldots + 1\cdot 2^{2} + 1\cdot 2^{1} + 1\cdot 2^{0}\right]} \\ &= 2^{-1} \cdot 2^{-\left[2^{(\mathtt{E}-1)}-1\right]} = 2^{-\left[2^{(\mathtt{E}-1)}\right]} \end{aligned} \tag{4.5}$$

Nótese que la mantisa mínima es 2^{-1} puesto que, en la definición de la representación de coma flotante, se exige que el primer dígito a la derecha de la coma sea distinto de cero.

Los valores de `M` y `E` en las variables reales del FORTRAN varían ligeramente según el modelo de ordenador. Para las variables `REAL*4`, puede tomarse, a título orientativo, $\mathtt{M} = 24$ posiciones para la mantisa y $\mathtt{E} = 8$ posiciones para el exponente. A partir de las expresiones 4.4 y 4.5 se obtiene que pueden almacenarse números reales comprendidos entre $0.29 \cdot 10^{-38}$ y $1.7 \cdot 10^{38}$. En cuanto a las variables `REAL*8`, los valores típicos son $\mathtt{M} = 53$ y $\mathtt{E} = 11$. Esto permite guardar números entre $0.56 \cdot 10^{-308}$ y $0.9 \cdot 10^{308}$.

Se ha presentado aquí la idea básica del almacenamiento de números reales en el ordenador. En la práctica, los distintos fabricantes de *hardware* han empleado ligeras modificaciones de esta idea, con el objetivo de conseguir el máximo de capacidad y de precisión para el espacio de memoria destinado a almacenar el número (Higham, 1996). El formato más habitual es el del IEEE (*Institute of Electrical and Electronic Engineers*), que se está convirtiendo en el estándar más extendido. En el formato IEEE los valores máximo y mínimo son algo distintos a los señalados, pero del mismo orden de magnitud: $10^{\pm 38}$ en simple precisión y $10^{\pm 308}$ en doble precisión.

4.2.3 *Overflow y underflow*

En el subapartado anterior se ha demostrado que la representación de un número real en el ordenador implica la existencia de un rango de números reales que pueden guardarse. Cuando en un programa se intenta almacenar un número mayor que el máximo (figura 4.5) se produce

un error que se denomina *overflow*. Del mismo modo, cuando se intenta almacenar un número menor que el mínimo (figura 4.6) se produce un error llamado *underflow*. Lo que sucede exactamente en ambos casos, así como el posible mensaje de error, depende del ordenador y del compilador utilizado. De todos modos, debe ser el programador quien tome las medidas oportunas a fin de evitar estos problemas.

Problema 4.1:
En el lenguaje de programación C existe un tipo de variable llamado *unsigned integer* que se caracteriza, básicamente, por ser una variable entera de 16 bits ó 32 bits (dependiendo del procesador de que se disponga) que sólo puede tomar valores positivos (no se reserva una posición para el signo). Se pide:

a) Determinar para cada caso de *unsigned integer* (16 bits ó 32 bits) cuál es el mayor número entero y el entero más próximo a cero (ambos en valor absoluto) que se puede almacenar.

b) ¿Qué tipo de variable permite almacenar un número mayor en valor absoluto: un *unsigned integer* de 16 bits, uno de 32 bits (lenguaje C) o un `INTEGER*4` en lenguaje FORTRAN?

c) ¿Cuál de los tres tipos de variables enteras mencionadas en el apartado *b* permite almacenar el número $k = 3125587976$? ¿Cuál de los tres anteriores tipos de variable enteras permite almacenar el número π? ●

4.3 Algoritmo

Un algoritmo es un sistema organizado para resolver un problema, formado por una serie finita de instrucciones que sólo se pueden interpretar unívocamente y que se realizan secuencialmente.

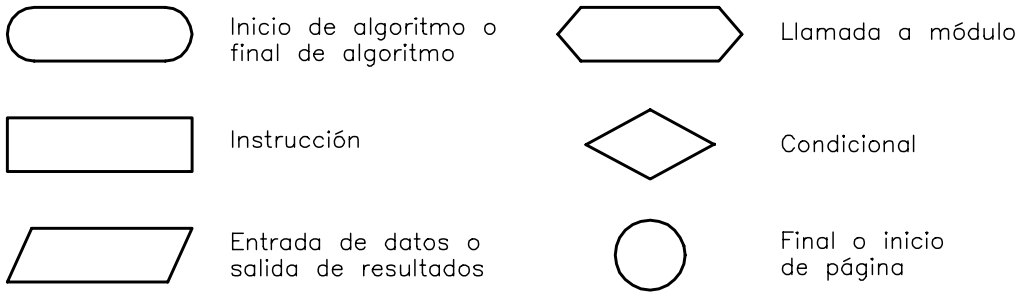


Fig. 4.7 Símbolos más utilizados en la representación de algoritmos mediante diagramas de flujo

En la actualidad existen, principalmente, dos formas de representar los algoritmos. La

primera es mediante *diagramas de flujo* y la segunda mediante *pseudocódigo*. En la representación de algoritmos mediante diagramas de flujo se utilizan una serie de símbolos con un significado predeterminado, que se unen mediante flechas que indican el orden en que se deben ejecutar las instrucciones. Aunque estos símbolos no están totalmente estandarizados, algunos de ellos si que están ampliamente aceptados. En la figura 4.7 se muestran los más utilizados.

En la figura 4.8 se presenta mediante un diagrama de flujo el algoritmo utilizado en el programa 3.8 para la resolución de la ecuación de segundo grado $ax^2 + bx + c = 0$.

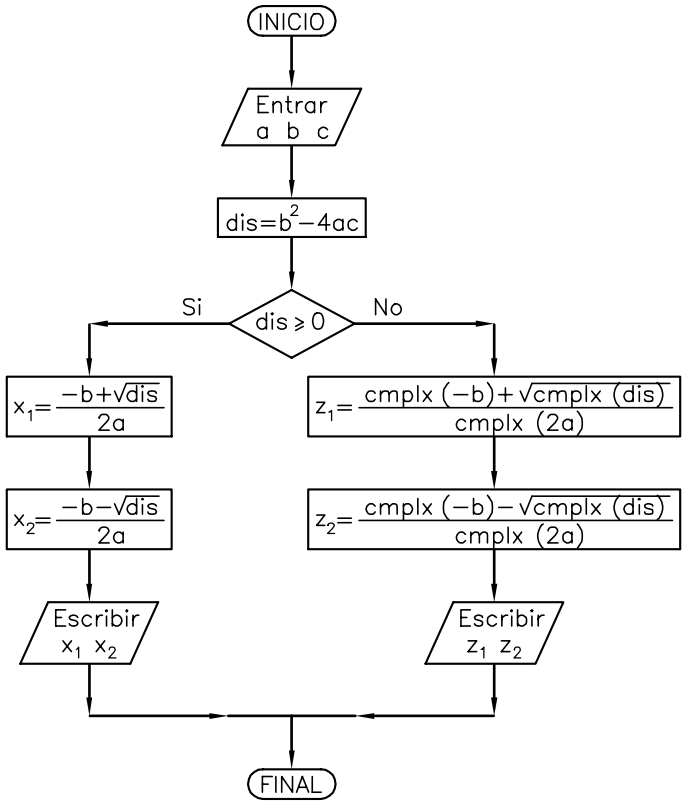


Fig. 4.8 Diagrama de flujo correspondiente a la resolución de una ecuación de segundo grado

En la representación mediante pseudocódigo, las instrucciones se especifican de forma similar a como se programan en un lenguaje de programación. Es importante resaltar que no existe ningún convenio acerca de cómo representar las instrucciones. En este sentido, la forma en que cada persona las detalla depende mucho del lenguaje de programación que acostumbra a utilizar. Con el propósito de ilustrar este método, a continuación se presenta el algoritmo de Euclides para determinar el máximo común divisor de dos números dados (`m` y `n`).

```
1.    Entrar m y n   (m > n)
2.    r = mod (m,n)
3.    Si r = 0   entonces   MCD = n; y FIN
4.    m ← n;   n ← r
5.    Ir al paso 2
```

donde `mod (m,n)` es una función que devuelve el resto de la división de `m` entre `n`.

Problema 4.2:
Escribir el algoritmo de resolución de la ecuación segundo grado utilizando una representación en pseudocódigo. Así mismo, escribir el algoritmo de Euclides mediante una representación en diagrama de flujo. ●

4.4 Errores

4.4.1 Error absoluto, error relativo y cifras significativas

Sea x el valor exacto de una cantidad y sea $\bar{x}$ su valor aproximado. Se define el *error absoluto* como

$$E_x = x - \bar{x} \tag{4.6}$$

El error absoluto mide la diferencia entre el valor exacto de una cantidad y su valor aproximado. De esta forma se puede afirmar que alguien ha medido la longitud de un campo de fútbol o la longitud de un bolígrafo con un error de un centímetro. Sin embargo, dicho error no tiene la misma importancia en ambos casos. Para cuantificar la importancia del error respecto del valor exacto de una cierta cantidad x se introduce el concepto de *error relativo*, que se define como

$$r_x = \frac{E_x}{x} = \frac{x - \bar{x}}{x} \tag{4.7}$$

Nótese que el error relativo no está definido para $x = 0$. La ecuación 4.7 muestra que el error relativo es una cantidad adimensional, que habitualmente se expresa en tanto por ciento (%).

Es importante resaltar que generalmente no se conoce el valor exacto de la cantidad x. En consecuencia, tampoco se puede conocer ni el error absoluto ni el error relativo cometido y hay que conformarse con calcular una cota del error.

Puesto que ahora se dispone de una definición cuantitativa de la importancia relativa del error, es posible plantearse cuál es la cota del error de redondeo cometido al almacenar un número. Como se ha comentado anteriormente, los números reales se almacenan en coma flotante. Por ejemplo, los números ± 23.487 se guardan como $\pm 0.23487 \cdot 10^2$. De forma genérica, puede escribirse

$$\pm m \cdot 10^e \tag{4.8}$$

donde $0 \leq m < 1$ representa la mantisa y e es un número entero que indica el exponente. Sea t el número de dígitos destinados a la representación de la mantisa (se supone que t no incluye la posición del signo). Por consiguiente, si una persona realiza unos cálculos trabajando en base diez, coma flotante y utilizando cinco dígitos para la mantisa ($t = 5$), puede representar los siguientes números: $0.23754 \cdot 10^2$, $0.10000 \cdot 10^5$, o $0.19876 \cdot 10^{-3}$.

Sin embargo, ¿qué le sucede cuando desea representar el número $a = 0.98567823$? Evidentemente no puede almacenarlo exactamente puesto que sólo dispone de cinco cifras para representar la mantisa. En consecuencia puede optar por una de las dos siguientes alternativas: $a_1 = 0.98567$ ó $a_2 = 0.98568$. La primera se denomina *redondeo por eliminación* mientras que la segunda se denomina *redondeo por aproximación*.

En general, se puede demostrar que si se representa un número en base n, coma flotante, reservando t dígitos para la mantisa (sin reservar una posición para el signo) y redondeando por eliminación, la cota del error relativo que se comete vale

$$|r_e| \ \leq \ n^{1-t}$$

Por el contrario, si el redondeo es por aproximación, la cota del error relativo es

$$|r_a| \ \leq \ \frac{1}{2}\, n^{1-t} \tag{4.9}$$

Como se puede observar, la cota del error relativo cuando se redondea por aproximación es la mitad que cuando se redondea por eliminación. Por este motivo todos los ordenadores almacenan los números reales redondeando por aproximación. Por ejemplo, si se realiza un cálculo representando los números en base diez, coma flotante, utilizando tres cifras para la mantisa y redondeando por aproximación, la cota del error relativo debido al redondeo vale

$$|r| \leq \frac{1}{2}\, 10^{1-3} \ = \ 0.005$$

En el ordenador sucede exactamente lo mismo. Cuando se representa un número real mediante una variable `REAL*4` (24 bits para la mantisa reservando uno para el signo, por tanto $t = 23$), la cota del error relativo debido al redondeo es

$$|r| \leq \frac{1}{2}\, 2^{1-23} = 2^{-23} = 1.19 \cdot 10^{-7}$$

Si se utilizan variables `REAL*8` (53 bits para la mantisa reservando uno para el signo, por tanto $t = 52$), entonces

$$|r| \leq \frac{1}{2}\, 2^{1-52} = 2^{-52} = 2.22 \cdot 10^{-16}$$

Puede verse que mediante variables `REAL*8` se obtiene una precisión mayor (menor error relativo) que con variables `REAL*4`.

El formato IEEE incorpora una modificación que mejora las precisiones recién indicadas. La idea es la siguiente (véase la figura 4.4): el primer dígito de la mantisa, $\mathtt{d\_1}$, siempre vale 1

(no puede ser 0 puesto que es el primer dígito a la derecha de la coma) y no hace falta guardarlo (Higham, 1996). Con ello se gana un dígito para la mantisa que permite obtener cotas del error relativo de redondeo de $2^{-24} = 0.60 \cdot 10^{-7}$ en simple precisión y de $2^{-53} = 1.11 \cdot 10^{-16}$ en doble precisión.

El error relativo está relacionado con la noción de *cifras significativas correctas*. Las cifras significativas de un número son la primera no nula y todas las siguientes. Así pues, 2.350 tiene cuatro cifras significativas mientras que 0.00023 tiene sólo dos.

Sea $\bar{x}$ una aproximación a x. Parece intuitivamente claro qué son las cifras significativas correctas de $\bar{x}$, pero no es fácil dar una definición precisa. Por ejemplo, una posible definición es la siguiente: una aproximación $\bar{x}$ a x tiene q cifras significativas correctas si al redondear $\bar{x}$ y x a q cifras significativas se obtiene el mismo resultado. Esta definición es aparentemente muy natural. Sin embargo, tómense los valores $x = 0.9949$ y $\bar{x} = 0.9951$. Según la definición, $\bar{x}$ tiene una cifra significativa correcta (al redondear, $\bar{x} \to 1$ y $x \to 1$) y también tres cifras significativas correctas ($\bar{x} \to 0.995$ y $x \to 0.995$). ¡En cambio, no tiene dos cifras significativas correctas! ($\bar{x} \to 1.0$ y $x \to 0.99$).

Para evitar estas anomalías, se adopta la siguiente definición: la aproximación $\bar{x}$ a x tiene q cifras significativas correctas si el error relativo verifica

$$|r_x| \leq \frac{1}{2} 10^{-q} \tag{4.10}$$

Con esta definición, un sencillo cálculo permite decir, sin ambigüedad, que $\bar{x} = 0.9951$ tiene tres cifras significativas correctas de $x = 0.9949$ (compruébese).

Problema 4.3:
En un proceso de cálculo hay que evaluar la serie

$$S_n = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^n}$$

Interesa determinar cuántos términos de la serie tiene sentido calcular considerando la precisión del tipo de variables que se utiliza. Como puede observarse, a excepción del primer término, cada sumando de la serie representa una potencia negativa del número 2. Por lo tanto, cada sumando puede interpretarse como un dígito en la representación en base dos y coma flotante de los números reales (figura 4.4). Para calcular dicho número de términos se propone el siguiente algoritmo:

```
1.    a ← 1; half ← 0.5; i ← 0
2.    b ← 2
3.    Repetir mientras que (b > 1)
4.            a ← a * half
5.            i ← i + 1
6.            b ← a + 1
```

```
7.    Fin de repetir
8.    Escribir i-1 y 2*a y FIN
```

Se pide:

a) Explicar razonadamente el funcionamiento del algoritmo. ¿Por qué se escriben las variables $i-1$ y $2*a$?

b) Realizar dos programas, uno en `REAL*4` y otro en `REAL*8`, en los que se implemente el anterior algoritmo. Comentar los resultados obtenidos.

●

4.4.2 Clasificación de los errores

En el contexto de los métodos numéricos, se considera que el error total que contiene un número puede ser debido a los siguientes tipos de errores:

1. *Error inherente.* En muchas ocasiones, los datos con que se inician los cálculos contienen un cierto error debido a que se han obtenido mediante la medida experimental de una determinada magnitud física. Así por ejemplo, el diámetro de la sección de una varilla de acero presentará un error según se haya medido con una cinta métrica o con un pie de rey. A este tipo de error se le denomina error inherente.

2. *Error de redondeo.* Como ya se ha comentado, un aspecto muy importante de la representación de los números reales en el ordenador es que éstos se almacenan siempre mediante una cadena finita de dígitos. Por tanto, en muchas ocasiones resulta imposible representar exactamente un número real (recordar el ejemplo del número π que se ha mencionado en la introducción de este capítulo). Sin embargo, hay que considerar esta propiedad en muchos otros casos en los que no parece tan evidente. Por ejemplo, resulta evidente que el número $(0.2)_{10}$ puede representarse exactamente mediante una cadena finita de dígitos en dicha base. Sin embargo, su expresión en base dos es

$$\begin{aligned}(0.2)_{10} &= \left(0.0011\ 0011\ 0011\ 0011\ 0011\ \ldots\right)_2 \\ &= \left(.11\ 0011\ 0011\ 0011\ 0011\ \ldots\right)_2 \cdot 2^{(-10)_2}\end{aligned}$$

   que obviamente no se puede almacenar mediante una cadena finita de dígitos. En consecuencia, cuando se almacena un número real se puede cometer un error. A este error se le denomina error de redondeo. Es importante recordar que la cota del error cometido depende de la base de numeración utilizada y del número de dígitos empleados para almacenar la mantisa, pero no depende de las posiciones reservadas para el exponente (ver expresión 4.5).

3. *Error de truncamiento.* En el apartado 4.3 se ha comentado que un algoritmo debe estar formado por un número finito de instrucciones. Sin embargo, existen muchos procesos que requieren la ejecución de un número infinito de instrucciones para hallar la solución exacta de un determinado problema. Puesto que es totalmente imposible realizar infinitas

instrucciones, el proceso debe truncarse. En consecuencia, no se halla la solución exacta que se pretendía encontrar, sino una aproximación a la misma. Al error producido por la finalización prematura de un proceso se le denomina error de truncamiento. Un ejemplo del error generado por este tipo de acciones es el desarrollo en serie de Taylor de una función $f(x)$

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + f''(x_0)\frac{(x - x_0)^2}{2!} + f'''(x_0)\frac{(x - x_0)^3}{3!} + \ldots + f^{(n)}(x_0)\frac{(x - x_0)^n}{n!} + R_n(x)$$

donde el residuo $R_n(x)$ (resto de Lagrange) representa la suma de todos los términos desde $n + 1$ hasta infinito y puede expresarse como

$$R_n(x) = f^{(n+1)}(\xi)\frac{(x - x_0)^{n+1}}{(n + 1)!}$$

donde ξ es cualquier valor entre x_0 y x. Puesto que, en general, se desconoce el valor de ξ, no puede evaluarse exactamente el valor de $R_n(x)$. Por este motivo, hay que conformarse con obtener una cota del error de truncamiento que se comete al truncar el desarrollo en serie en la derivada n-ésima.

Como en este ejemplo, en la mayoría de algoritmos tampoco se puede calcular exactamente el error de truncamiento cometido. En cualquier caso, siempre resulta muy interesante hallar una cota de su valor.

4.5 Propagación del error

En este apartado se cuantificará la propagación del error al efectuar operaciones. Se obtendrán expresiones que relacionan el error del resultado obtenido con el error de los datos. Sin embargo, el error de los datos es desconocido (si se conociera, a partir de éste y del valor aproximado siempre se podría calcular el valor exacto). Lo que generalmente se conoce, bien a partir de la precisión de los aparatos de medida o bien a partir de desigualdades como la expresión 4.9, es una cota del error de los datos. En consecuencia, el objetivo ahora es deducir una expresión para la cota del error en el resultado de una secuencia de operaciones. En este sentido, si la realización de un cálculo puede llevarse a cabo mediante dos expresiones, será aconsejable utilizar aquella que tenga asociada una cota del error menor.

4.5.1 Conceptos previos

Las consecuencias de la existencia de un error en los datos de un problema son más importantes de lo que aparentemente puede parecer. Desafortunadamente, estos errores se propagan y amplifican al realizar operaciones con dichos datos, hasta el punto de que puede suceder que

el resultado carezca de significado. Con el propósito de ilustrar esta situación, seguidamente se calcula la diferencia entre los números

$$a = 0.276435$$
$$b = 0.2756$$

Si los cálculos se realizan en base diez, coma flotante, redondeando por aproximación y trabajando con tres dígitos de mantisa, los valores aproximados a dichos números y el error relativo cometido es

$$\bar{a} = 0.276 \qquad |r_a| = 1.57 \cdot 10^{-3}$$
$$\bar{b} = 0.276 \qquad |r_b| = 1.45 \cdot 10^{-3}$$

Si ahora se calcula la diferencia entre los valores exactos y la diferencia entre los aproximados se obtiene

$$a - b = 0.000835$$
$$\bar{a} - \bar{b} = 0.0$$

Debe observarse que el error relativo de la diferencia aproximada es del 100%. Este ejemplo, extraordinariamente sencillo, pone de manifiesto cómo el error de redondeo de los datos se ha amplificado al realizar una única operación, hasta generar un resultado carente de significado.

El ejemplo anterior es un caso particular de unas propiedades generales que se analizarán en este apartado. Así mismo, también se propondrán algunas normas destinadas a reducir la propagación de los errores, como por ejemplo: evitar restar números muy parecidos o evitar dividir por números muy pequeños comparados con el numerador.

Por último, hay que tener en cuenta el efecto conjunto de los tres tipos de errores. Sus consecuencias se pueden ilustrar a partir del cálculo de la exponencial expuesto en la introducción de este capítulo

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \ldots$$

Si se representa el valor absoluto del error de truncamiento $|r_t|$ frente al número de términos de la serie considerados, se observa que tiende asintóticamente a cero al ir calculando más términos de dicha serie (ver figura 4.9). Por consiguiente, se puede concluir que cuantos más términos se calculen mejor. Sin embargo, si se considera la propagación del error de redondeo, es de esperar que el valor absoluto de dicho error $|r_r|$ aumente con el número de términos considerados, puesto que cada vez se realizan más operaciones (ver figura 4.9). Por lo tanto, si se calcula la suma de los valores absolutos de los dos errores $|r_s|$ (error total), se observa que existe un número de términos para el cual el error es mínimo. Así pues, se puede afirmar que en este tipo de procesos existe un paso más allá del cual empiezan a obtenerse peores resultados. Desafortunadamente, no existe, en general, un método para hallar el valor de dicho paso.

En la práctica, a partir de criterios físicos y numéricos se impone una cierta tolerancia (valor máximo del error que puede aceptarse). Cuando el error del proceso es menor que dicha tolerancia éste se detiene. Es interesante resaltar que el valor asignado a la tolerancia debe escogerse razonadamente. Por ejemplo, en la figura 4.9 se ilustra cómo al tomar una tolerancia excesivamente pequeña (tolerancia 2 _ _ _ _), el error total nunca es inferior a dicho valor y

en consecuencia, el proceso no se detendría nunca. Por el contrario, si el valor de la tolerancia es superior (tolerancia 1 _____), existe un cierto término en el cual se obtiene la precisión requerida.

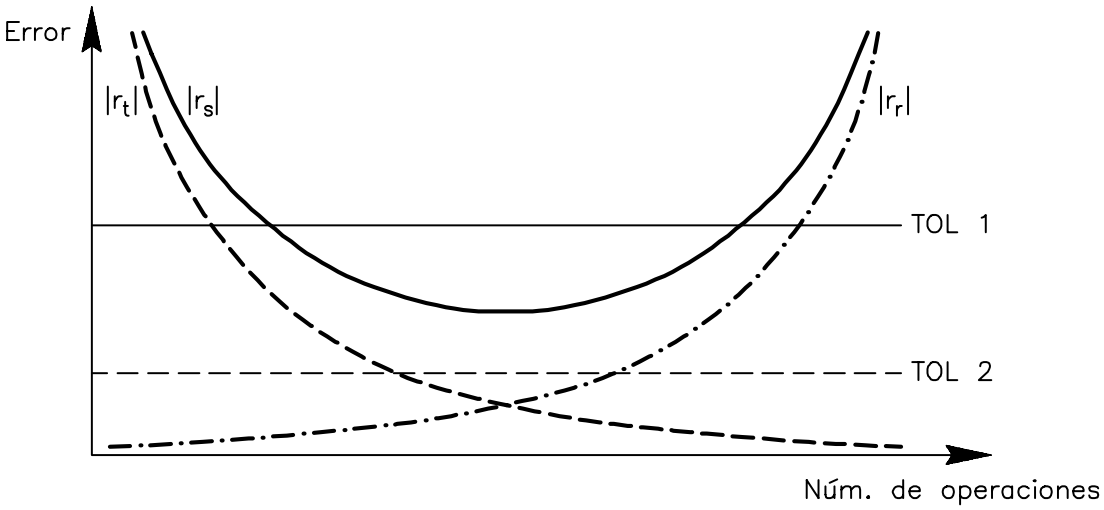


Fig. 4.9 Propagación del error en un algoritmo numérico

4.5.2 Propagación del error en la suma

En ésta y en las siguientes demostraciones se denotará por x e y los valores exactos de dos números y por $\bar{x}$ e $\bar{y}$ sus valores aproximados. Así mismo, los errores absolutos y relativos de estas cantidades se denotarán por E_x, E_y, r_x, r_y, respectivamente. Si se representa por $s = x + y$ al valor exacto de la suma y por $\bar{s} = \bar{x} + \bar{y}$ su valor aproximado, entonces el error absoluto de la suma es

$$E_s = s - \bar{s} = (x + y) - (\bar{x} + \bar{y}) = E_x + E_y$$

La expresión anterior indica que el error absoluto de la suma es la suma de valores absolutos de los sumandos. El error relativo vale

$$r_s = \frac{E_s}{s} = \frac{E_x + E_y}{x + y} = \frac{x}{x + y} r_x + \frac{y}{x + y} r_y \tag{4.11}$$

donde se puede observar que el error relativo de la suma es la suma de los errores relativos de los datos multiplicados por unos factores que dependen de dichos datos. Esta dependencia se muestra gráficamente en la figura 4.10.

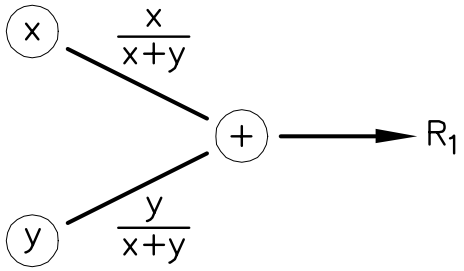


Fig. 4.10 Propagación del error relativo en una suma

4.5.3 Propagación del error en la resta

La deducción para la propagación del error mediante la resta es muy parecida a la anterior. Si se representa por $r = x - y$ al valor exacto de la resta y por $\bar{r} = \bar{x} - \bar{y}$ su valor aproximado, entonces el error absoluto es

$$E_r = r - \bar{r} = (x - y) - (\bar{x} - \bar{y}) = E_x - E_y$$

y el error relativo es

$$r_r = \frac{E_r}{r} = \frac{E_x - E_y}{x - y} = \frac{x}{x - y}\, r_x - \frac{y}{x - y}\, r_y \tag{4.12}$$

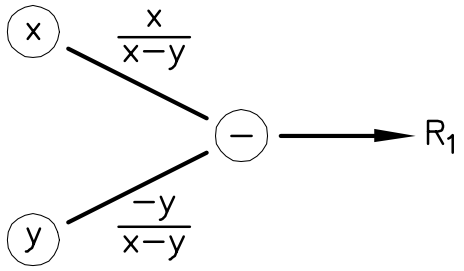


Fig. 4.11 Propagación del error relativo en una resta

En la figura 4.11 se representa gráficamente la propagación del error relativo de la resta. Ahora puede observarse el efecto de amplificación del error que los coeficientes de la expresión 4.12 pueden producir. En efecto, si se calcula la diferencia entre dos números muy parecidos, los términos $x/(x-y)$ y $-y/(x-y)$ serán extraordinariamente grandes y, en consecuencia, el error relativo r_r será muy superior a los errores r_x y r_y. Desafortunadamente, la única forma de evitar este comportamiento es no restando números muy parecidos. Obviamente, este mismo fenómeno se produce al sumar dos números x e y tales que $x \approx -y$ (ver expresión 4.11).

4.5.4 Propagación del error en el producto

Si se representa el producto de dos números exactos mediante $p = xy$ y el valor aproximado del producto por $\bar{p} = \bar{x}\bar{y}$, el error absoluto del producto se puede calcular como

$$\begin{aligned} E_p &= p - \bar{p} = (x\ y) - (\bar{x}\ \bar{y}) = (x\ y) - (x - E_x)\ (y - E_y) \\ &= x\,E_y + y\,E_x - E_x\,E_y \approx x\,E_y + y\,E_x \end{aligned}$$

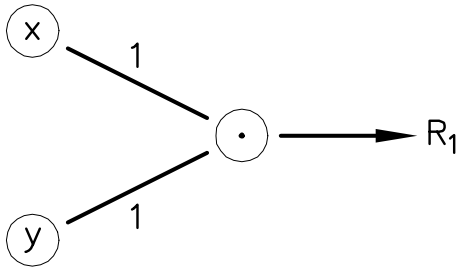


Fig. 4.12 Propagación del error relativo en un producto

A partir de este resultado, se obtiene que el error relativo del producto es

$$r_p = \frac{E_p}{p} = \frac{x\,E_y + y\,E_x - E_x\,E_y}{x\ \ y} = r_x + r_y + r_x\ r_y \approx r_x + r_y \qquad (4.13)$$

que indica que el error relativo del producto es suma de los errores relativos de los datos como se ilustra en la figura 4.12. En la expresión 4.13 se ha supuesto que los errores relativos son suficientemente pequeños como para despreciar el término cuadrático frente a los lineales.

4.5.5 Propagación del error en la división

Si ahora se representa el cociente de dos números exactos mediante $d = x\,/\,y$ y su valor aproximado mediante $\bar{d} = \bar{x}\,/\,\bar{y}$, el error absoluto del cociente vale

$$\begin{aligned} E_d &= d - \bar{d} = \frac{x}{y} - \frac{\bar{x}}{\bar{y}} = \frac{x}{y} - \frac{(x - E_x)}{(y - E_y)} \\ &= \frac{y\,E_x - x\,E_y}{y\,(y - E_y)} \approx \frac{y\,E_x - x\,E_y}{y^2} \end{aligned}$$

En este caso se debe resaltar que si el denominador es un número muy pequeño, el error absoluto del cociente puede ser muy superior al error absoluto de los datos. Por consiguiente y en la medida de lo posible, hay que evitar dividir por números pequeños comparados con el numerador.

De la expresión anterior se deduce que el error relativo del cociente es

$$r_d = \frac{E_d}{d} = \frac{\frac{y\,E_x - x\,E_y}{y\,(y-E_y)}}{\frac{x}{y}} = \frac{y\,E_x - x\,E_y}{x\,(y - E_y)} = \frac{r_x - r_y}{1 - r_y} \approx r_x - r_y \tag{4.14}$$

De acuerdo con la expresión anterior, en la figura 4.13 se ilustra como el error relativo del cociente es la resta de los errores relativos de los datos.

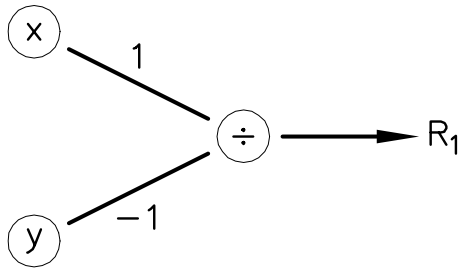


Fig. 4.13 Propagación del error relativo en una división

4.5.6 Propagación del error en una función

Sea $z = f(x)$ la imagen mediante la función f del valor exacto de un número x y sea $\bar{z} = f(\bar{x})$ la imagen de su valor aproximado. Entonces, el error absoluto de la imagen es

$$\begin{aligned} E_z &= f(x) - f(\bar{x}) = f(x) - f(x - E_x) \\ &= f(x) - \left[f(x) - f'(x)\,E_x + f''(x)\,\frac{{E_x}^2}{2!} - f'''(x)\,\frac{{E_x}^3}{3!} + \cdots \right] \\ &= f'(x)\,E_x - f''(x)\,\frac{{E_x}^2}{2!} + f'''(x)\,\frac{{E_x}^3}{3!} - \cdots \approx f'(x)\,E_x \end{aligned}$$

En consecuencia, el error relativo que se comete al evaluar la función f está determinado por la expresión

$$r_z = \frac{E_z}{z} = \frac{f'(x)\,E_x - f''(x)\,\frac{{E_x}^2}{2!} + f'''(x)\,\frac{{E_x}^3}{3!} - \ldots}{f(x)} \approx x\,\frac{f'(x)}{f(x)}\,r_x \tag{4.15}$$

que gráficamente se muestra en la figura 4.14.

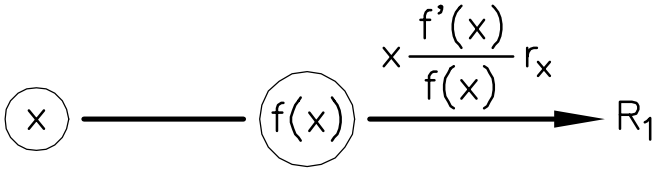


Fig. 4.14 Propagación del error relativo al evaluar una función

4.6 Análisis de perturbaciones

En el apartado anterior se han deducido las expresiones que gobiernan la propagación del error en las operaciones aritméticas elementales. El objetivo de este apartado es desarrollar un método que permita analizar qué sucede con la propagación del error cuando se realizan varias operaciones.

La característica básica de este tipo de estudios radica en el desconocimiento exacto del error que presentan los datos. Como se ha comentado anteriormente, lo único que se conoce es una cota del error. Por lo tanto, tan sólo se puede aspirar a conocer una cota del error final de las operaciones.

La cota del error que se comete en una secuencia de operaciones se puede calcular mediante los siguientes pasos:

1. Realizar un diagrama que represente el orden en que se realizan las operaciones (figura 4.15).

2. Numerar los resultados parciales que aparecen en dicho diagrama.

3. Especificar los coeficientes para el cálculo del error en cada una de las operaciones.

4. Determinar la expresión del error en cada uno de los resultados parciales que aparecen en el diagrama (el error correspondiente a la última operación será el error final de la secuencia de operaciones).

5. Calcular la cota del error final.

Con el propósito de ilustrar este método se propone el siguiente ejemplo. Se desea realizar el cálculo

$$z = a\,(b + c) \tag{4.16}$$

que alternativamente se puede realizar mediante la expresión

$$z = a\,b + a\,c \tag{4.17}$$

La cuestión es saber mediante cuál de las dos expresiones anteriores se obtiene una cota del error menor.

En la figura 4.15 se presentan los tres primeros pasos del estudio de propagación de error para cada una de las alternativas presentadas en las ecuaciones 4.16 y 4.17. El error relativo que se produce en la i-ésima operación de la primera secuencia de operaciones (expresión 4.16) se designa mediante R_i, mientras que el error relativo de la i-ésima operación de la segunda (expresión 4.17) se representa por $\hat{R}_i$.

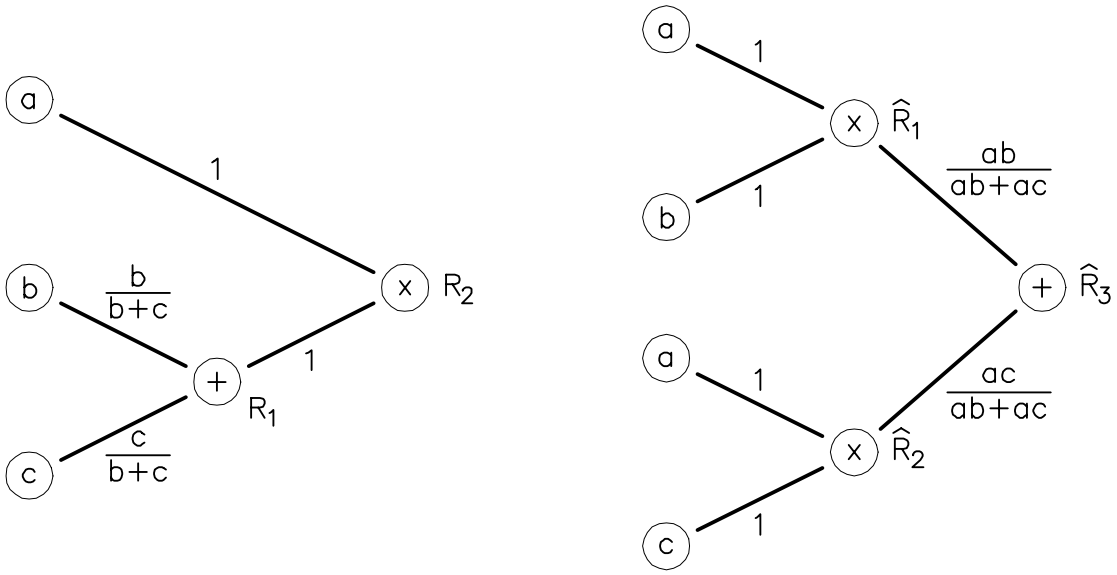


Fig. 4.15 Esquemas de representación de la propagación del error

En el cuarto paso del método hay que determinar el error en cada uno de los resultados parciales. El error que se comete en las diferentes operaciones de la primera alternativa es

$$\begin{aligned} R_1 &= \frac{b}{b+c}\, r_b + \frac{c}{b+c}\, r_c + r_1 \\ R_2 &= R_1 + r_a + r_2 = \frac{b}{b+c}\, r_b + \frac{c}{b+c}\, r_c + r_1 + r_a + r_2 \end{aligned} \tag{4.18}$$

donde r_1 y r_2 representan los errores de redondeo que se cometen al almacenar los resultados intermedios de las dos operaciones.

El error en las operaciones de la segunda alternativa es

$$\begin{aligned} \hat{R}_1 &= r_a + r_b + \hat{r}_1 \\ \hat{R}_2 &= r_a + r_c + \hat{r}_2 \\ \hat{R}_3 &= \frac{ab}{ab+ac}\, \hat{R}_1 + \frac{ac}{ab+ac}\, \hat{R}_2 + \hat{r}_3 \\ &= \frac{b}{b+c}\left(r_a + r_b + \hat{r}_1\right) + \frac{c}{b+c}\left(r_a + r_c + \hat{r}_2\right) + \hat{r}_3 \end{aligned} \tag{4.19}$$

Como en el caso anterior, $\hat{r}_1$, $\hat{r}_2$ y $\hat{r}_3$ representan los errores de redondeo cometidos al almacenar los resultados obtenidos en las tres operaciones.

Por último (quinto paso del método) hay que calcular la cota del error en la última expresión de cada alternativa. Para ello, se deben realizar las suposiciones pertinentes sobre cómo se realizan las operaciones y sobre la composición del error de los datos. Por ejemplo, en el estudio que se está realizando se puede suponer que:

1. Las operaciones se realizan manualmente (en base diez), coma flotante, utilizando cuatro dígitos para la parte fraccionaria y redondeando siempre por aproximación. En estas condiciones y de acuerdo con la expresión 4.9, la cota del error relativo de redondeo es $r = (1/2)\,10^{1-4} = 0.0005$.

2. Los datos no contienen error inherente. Es decir, el error que puedan presentar los datos sólo es debido al error de redondeo.

Bajo estas hipótesis, los errores de los datos (r_a, r_b, y r_c) y los errores cometidos al almacenar los resultados intermedios de las diversas operaciones (r_1, r_2, $\hat{r}_1$, $\hat{r}_2$ y $\hat{r}_3$) se deben sólo al error de redondeo. Por consiguiente verifican que

$$\begin{array}{lll} |r_a| \leq r & |r_1| \leq r & |\hat{r}_1| \leq r \\ |r_b| \leq r & |r_2| \leq r & |\hat{r}_2| \leq r \\ |r_c| \leq r & & |\hat{r}_3| \leq r \end{array}$$

Se debe resaltar que las desigualdades anteriores indican claramente que se desconoce el valor exacto del error relativo de redondeo. En este sentido, lo único que se puede afirmar es que el valor absoluto del error relativo de redondeo que se produce al almacenar los números es inferior a una cierta cantidad r. Por lo tanto, al realizar el análisis de perturbaciones hay que tomar el valor absoluto de las expresiones 4.18 y 4.19. En consecuencia, para la primera alternativa se obtiene

$$\begin{aligned} |R_2| &= \left| \frac{b}{b+c}\, r_b + \frac{c}{b+c}\, r_c + r_1 + r_a + r_2 \right| \\ &\leq \left| \frac{b}{b+c} \right| |r_b| + \left| \frac{c}{b+c} \right| |r_c| + |r_1| + |r_a| + |r_2| \\ &\leq \left| \frac{b}{b+c} \right| r + \left| \frac{c}{b+c} \right| r + r + r + r \\ &\leq \left[\left| \frac{b}{b+c} \right| + \left| \frac{c}{b+c} \right| + 3 \right] r \end{aligned} \tag{4.20}$$

mientras que para la segunda se obtiene

$$\begin{aligned}
|\hat{R}_3| &= \left| \frac{b}{b+c}\left(r_a + r_b + \hat{r}_1\right) + \frac{c}{b+c}\left(r_a + r_c + \hat{r}_2\right) + \hat{r}_3 \right| \\
&\leq \left| \frac{b}{b+c} \right| |\, r_a + r_b + \hat{r}_1 \,| + \left| \frac{c}{b+c} \right| |\, r_a + r_c + \hat{r}_2 \,| + |\hat{r}_3| \\
&\leq \left| \frac{b}{b+c} \right| \left[|\, r_a \,| + |\, r_b \,| + |\, \hat{r}_1 \,| \right] + \left| \frac{c}{b+c} \right| \left[|\, r_a \,| + |\, r_c \,| + |\, \hat{r}_2 \,| \right] + |\, \hat{r}_3 \,| \\
&\leq \left[\left[\left| \frac{b}{b+c} \right| + \left| \frac{c}{b+c} \right| \right] 3 + 1 \right] r
\end{aligned} \tag{4.21}$$

Las expresiones 4.20 y 4.21 muestran que la cota del error relativo de ambas operaciones depende linealmente de la cota del error relativo de redondeo. Además, muestran que la cota del error *no es la misma* para ambas alternativas. Este resultado parece contradecir la propiedad distributiva del producto respecto de la suma. Sin embargo se debe recordar que en esta deducción se ha considerado que los números se almacenan mediante una cadena finita de dígitos.

Por último, del resultado anterior se desprende la siguiente pregunta: ¿cuál de los dos métodos proporciona una cota del error menor? Para ello se plantea si es cierta la siguiente desigualdad

$$|R_2| \leq |\hat{R}_3|$$

Es decir

$$\left[\left| \frac{b}{b+c} \right| + \left| \frac{c}{b+c} \right| + 3 \right] r \leq \left[\left[\left| \frac{b}{b+c} \right| + \left| \frac{c}{b+c} \right| \right] 3 + 1 \right] r$$

$$\left| \frac{b}{b+c} \right| + \left| \frac{c}{b+c} \right| + 2 \leq \left[\left| \frac{b}{b+c} \right| + \left| \frac{c}{b+c} \right| \right] 3$$

$$1 \leq \left| \frac{b}{b+c} \right| + \left| \frac{c}{b+c} \right|$$

que evidentemente es cierta. Por lo tanto, desde el punto de vista numérico, es preferible utilizar la primera alternativa puesto que conlleva una cota del error del resultado menor.

Problema 4.4:
En el proceso de diseño de unas piezas metálicas, se debe calcular el perímetro P de una elipse de semiejes a y b con un error relativo inferior a $5 \cdot 10^{-2}$. Para ello se ha decidido utilizar la siguiente expresión

$$P = 2\pi \sqrt{\frac{a^2 + b^2}{2}}$$

Se pide:

a) Efectuar un estudio completo de propagación de errores, incluidos los errores inherentes y los errores de redondeo, para el cálculo del perímetro P.

b) Obtener una expresión de la cota del error relativo del perímetro P.

c) Un operario asegura que siempre se podrá calcular correctamente dicho perímetro si se miden los semiejes con una cinta métrica de precisión igual a 2.5% y se realizan las operaciones en base diez, coma flotante, utilizando tres dígitos para la mantisa (sin incluir el signo) y redondeando por aproximación. ¿Es cierta esta afirmación? En caso negativo, determinar cuál debería ser la precisión exigible a la cinta métrica. ●

Problema 4.5:
Para iniciar la fabricación en masa de rodamientos de alta calidad, un ingeniero debe medir, con la mayor precisión posible, el radio A de una pequeña esfera metálica que forma parte del prototipo. Para ello dispone de tres alternativas:

1. Medir el diámetro D con un pie de rey y obtener el radio A como $A = D/2$.
2. Medir la superficie S mediante técnicas indirectas y obtener el radio como $A = \sqrt{\frac{S}{4\pi}}$.
3. Medir el volumen V sumergiendo la esfera en un líquido y obtener el radio como $A = \sqrt[3]{\frac{3V}{4\pi}}$.

Se pide:

a) Efectuar un estudio completo de propagación de errores, incluidos los errores inherentes y los errores de redondeo, para cada una de las tres alternativas.

b) Obtener una cota del error relativo en el radio A, para cada una de las tres alternativas.

El ingeniero sabe que la cota del error relativo inherente de las medidas experimentales D, S y V es de 10^{-3}. Para efectuar los cálculos, utiliza un sencillo programa en FORTRAN, que trabaja con variables `REAL*4`. Ciertos condicionantes de diseño exigen la obtención del radio A con un error relativo máximo del 0.05%.

c) ¿Cuál de las tres alternativa(s) puede utilizar el ingeniero para obtener el radio A con la precisión requerida? ¿Cuál es la más indicada? ●

Problema 4.6:
Durante la construcción de un puente atirantado los ingenieros se plantean el siguiente problema: ¿con qué precisión hay que medir la posición de los anclajes de los tirantes, tanto en la pila como en el tablero, para tener un error en la longitud de los cables inferior a 25 cm? Se sabe que aproximadamente los cables miden 100 m, se puede suponer que se trabaja con infinitas cifras significativas correctas (sin errores de redondeo) y que el error inherente de las medidas necesarias es siempre el mismo.
En realidad la posición de los anclajes se conocía exactamente, siendo sus coordenadas números enteros, y uno de los ingenieros de obra encargó ya los tirantes. Dicho ingeniero tiene por costumbre realizar los cálculos en obra con dos dígitos de precisión. ¿Servirán los cables por él pedidos?
Nota: se supondrá que el peso propio de los tirantes es despreciable y que la estructura no se deforma. Así pues, el conjunto formado por el tablero, la pila y el cable define un triángulo rectángulo. ●

4.7 Bibliografía

HENRICI, P. *Elementos de análisis numérico.* Trillas, 1972.

HIGHAM, N.J. *Accuracy and Stability of Numerical Algorithms.* SIAM, 1996.

HILDEBRAND, F.B. *Introduction to Numerical Analysis.* McGraw-Hill, 1974.

5 Ceros de funciones

Objetivos

- Describir tres técnicas numéricas iterativas para hallar ceros de funciones ($f(x) = 0$): método de la bisección, método de Newton y método de la secante.
- Estudiar y comparar los tres métodos mediante algunos ejemplos numéricos.
- Explicar las funciones externas `FUNCTION` de FORTRAN.

5.1 Introducción

Muchos problemas pueden modelarse matemáticamente como una ecuación

$$f(x) = 0 \tag{5.1}$$

donde f es una cierta función de una variable x. Se trata pues de hallar los valores de x que satisfacen la ecuación 5.1. Estos valores se llaman *ceros* de la función f o *raíces* de la ecuación $f(x) = 0$, y se denotan por x^*. Gráficamente, los ceros de una función son los puntos de intersección de la gráfica $y = f(x)$ con el eje de las x.

Para algunos casos sencillos, la ecuación 5.1 puede resolverse analíticamente. Supóngase, por ejemplo, que f es un polinomio de segundo grado, $f(x) = ax^2 + bx + c$. Entonces, el número de ceros (reales) depende del valor del discriminante $\Delta = b^2 - 4ac$; para $\Delta > 0$, la función f tiene dos ceros $x = (-b \pm \sqrt{\Delta})/2a$.

En un problema más general, si f es una función cualquiera, la ecuación 5.1 *no puede resolverse analíticamente*. De hecho, ni siquiera se sabe a priori cuántos ceros tiene f: ¿varios, uno, ninguno? En estos casos, es necesario utilizar una *técnica numérica iterativa*: a partir de una aproximación inicial x^0 a un cero x^* de f, se construye iterativamente una sucesión de

aproximaciones $\{x^k\}$. El superíndice k es el contador de iteraciones: en la primera iteración, se calcula x^1; en la segunda, x^2, y así sucesivamente. El proceso iterativo se detiene cuando, para un cierto valor de k, el valor x^k es una aproximación *suficientemente* buena a x^*.

Puede verse pues que para obtener numéricamente un cero de f hay que responder las tres preguntas siguientes:

1. ¿Cómo se elige la aproximación inicial x^0?

2. ¿Cómo se construye la sucesión $\{x^k\}$ de aproximaciones?

3. ¿Cómo se decide si x^k es una aproximación suficientemente buena a x^*?

Estas preguntas se responden a lo largo del capítulo. De momento, y para terminar este apartado de introducción, se presentan dos ejemplos de la ecuación 5.1.

5.1.1 Cálculo de raíces cuadradas

Un ingeniero necesita calcular la raíz cuadrada x de un número s, $x = \sqrt{s}$, haciendo *únicamente* operaciones aritméticas elementales (suma, resta, producto y división). Ésta es la situación real en el diseño de algunos ordenadores, puesto que sólo estas cuatro operaciones están incorporadas a nivel de *hardware*, y las demás operaciones deben hacerse a partir de ellas.

Dado que no se puede calcular directamente la raíz cuadrada $\sqrt{s}$, se utiliza una estrategia alternativa. Elevando la expresión $x = \sqrt{s}$ al cuadrado y pasando s a la izquierda de la igualdad, puede escribirse

$$f(x) = x^2 - s = 0 \tag{5.2}$$

En la ecuación 5.2 queda claro que el cálculo de $\sqrt{s}$ equivale a obtener el cero de la función $f(x) = x^2 - s$. En otras palabras, se trata de hallar la intersección de la gráfica $y = f(x)$ con el eje de las x (véase la figura 5.1). Como se verá en los apartados siguientes, la ecuación 5.2 puede resolverse de manera iterativa empleando únicamente las cuatro operaciones aritméticas elementales.

5.1.2 Cómo jugar al billar en una mesa circular

La última moda entre los aficionados al billar es la mesa circular (véase la figura 5.2). Para los principiantes, el juego consiste simplemente en golpear la bola Q con la bola P después de un impacto I en la banda. Los parámetros del problema pueden verse en la figura 5.2: la mesa tiene radio R, la posición de las bolas P y Q queda determinada por las coordenadas cartesianas (x_P, y_P) y (x_Q, y_Q), y el punto de impacto I viene definido por el ángulo θ.

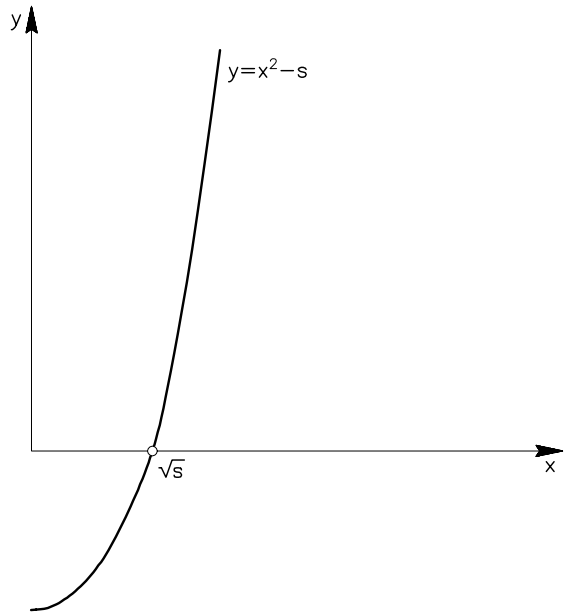


Fig. 5.1 Gráfica de la función $f(x) = x^2 - s$

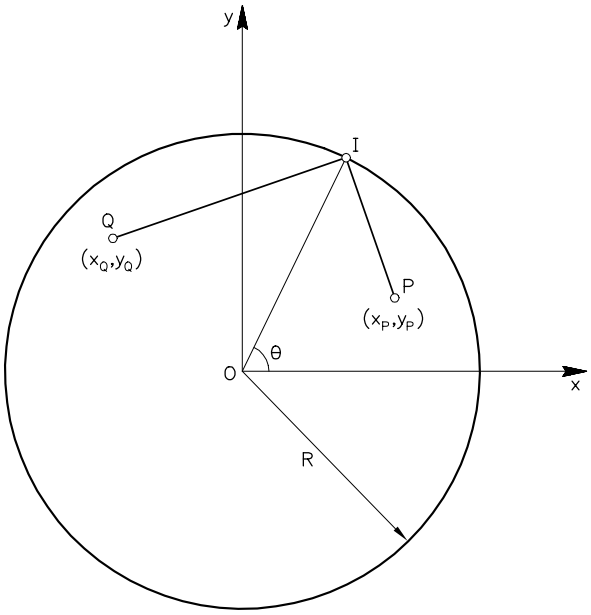


Fig. 5.2 Jugando al billar en una mesa circular

Mediante consideraciones geométricas sencillas (que se dejan como ejercicio al lector interesado) puede verse que los valores de θ que proporcionan los puntos de impacto I son las raíces de la ecuación

$$f(\theta) = \frac{x_P \sin\theta - y_P \cos\theta}{\sqrt{(R\cos\theta - x_P)^2 + (R\sin\theta - y_P)^2}} + \frac{x_Q \sin\theta - y_Q \cos\theta}{\sqrt{(R\cos\theta - x_Q)^2 + (R\sin\theta - y_Q)^2}} = 0 \quad (5.3)$$

Nótese que θ es la única incógnita de la ecuación 5.3. Los valores de R, x_P, y_P, x_Q e y_Q son datos del problema. Para resolver la ecuación 5.3 hay que utilizar una técnica numérica iterativa, que construya una sucesión $\{\theta^k\}$ de aproximaciones a un cero θ^* de la función f.

5.2 Método de la bisección

La primera técnica iterativa para hallar ceros de funciones que se presenta aquí es el *método de la bisección*. Se ilustrará mediante el cálculo de $\sqrt{2}$ a partir de operaciones aritméticas elementales. Se trata, pues, de tomar $s = 2$ en la ecuación 5.2. Tal como ya se ha comentado, $\sqrt{2}$ es el cero de la función $f(x) = x^2 - 2$ (véase la figura 5.3).

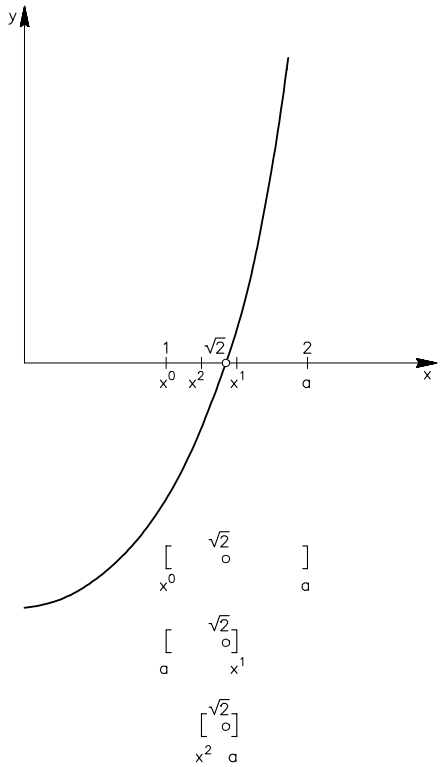


Fig. 5.3 Cálculo de $\sqrt{2}$ por el método de la bisección

El método de la bisección consiste en:

1. Inicializar el contador de iteraciones a cero ($k = 0$).

2. Elegir una aproximación inicial x^0 y otro valor a de manera que el intervalo que definen estos dos puntos (que será $[x^0, a]$ para $x^0 < a$ y $[a, x^0]$ para $x^0 > a$) contenga el cero buscado x^* y sólo ese cero. Para el problema que se está estudiando es muy sencillo ver que la función $f(x) = x^2 - 2$ tiene un único cero $x^* = \sqrt{2}$ en el intervalo $[1, 2]$. Se toma, por tanto, $x^0 = 1$ y $a = 2$. Para un problema general, se utiliza el siguiente control para elegir x^0 y a: si f es una función continua y el intervalo de extremos x^0 y a contiene un cero de f, entonces $f(x^0)$ y $f(a)$ tienen signos distintos (es decir, $f(x^0)f(a) < 0$). En la figura 5.3 puede verse cómo, efectivamente, $f(x^0) < 0$ y $f(a) > 0$.

3. Tomar el punto medio del intervalo, $x^{k+1} = (x^k + a)/2$, como siguiente aproximación a x^*. Nótese que x^{k+1} divide al intervalo de extremos x^k y a en dos nuevos intervalos con la mitad de longitud. Por este motivo, se habla de método de la *bisección*.

4. Decidir si x^{k+1} es una aproximación suficientemente buena a x^*. En caso afirmativo, detener el proceso iterativo y tomar $x^* \approx x^{k+1}$. En caso negativo, seguir iterando. Para tomar la decisión es necesario emplear algún *criterio de convergencia* (ver apartado 5.3).

5. Detectar cuál de los dos intervalos obtenidos en el paso 3 contiene x^*. Puede hacerse de manera sistemática, sin necesidad de dibujar la gráfica de la función, a partir del signo de $f(x^{k+1})$. Si $f(x^{k+1})$ y $f(x^k)$ tienen signos distintos, entonces x^* está en el intervalo de extremos x^k y x^{k+1}. Si, por el contrario, el cambio de signo de f se produce entre x^{k+1} y a, el cero x^* está entre x^{k+1} y a. Por último, puede ocurrir que $f(x^{k+1}) = 0$; en este caso, x^{k+1} es el cero x^* y se detiene el proceso iterativo de bisección.

6. Tomar el intervalo escogido en el paso 5 como nuevo intervalo de trabajo. Para obtener un algoritmo más compacto y facilitar su programación, interesa denotar por a uno de los extremos del intervalo (el otro extremo es x^{k+1}) durante todo el proceso. Para ello se adopta el siguiente criterio:

$$\text{si} \quad \begin{cases} f(x^{k+1})f(x^k) < 0 \quad ; \quad a \leftarrow x^k \\ f(x^{k+1})f(x^k) \geq 0 \quad ; \quad a \leftarrow a \end{cases}$$

7. Incrementar en 1 el contador de iteraciones ($k \leftarrow k + 1$) y volver al paso 3.

En resumen, en el método de la bisección se parte de un intervalo inicial que contiene el cero x^*, y se va subdividiendo este intervalo hasta "encerrar" a x^* en un intervalo tan pequeño como se desee.

Obsérvese que el algoritmo que se acaba de presentar aborda las tres cuestiones planteadas en el apartado 5.1: elección de la aproximación inicial x^0 (paso 2), construcción de la sucesión $\{x^k\}$ de aproximaciones a x^* (pasos 3, 5, 6 y 7) y finalización de las iteraciones (paso 4).

En el apartado 5.7 se muestra un programa FORTRAN (programa 5.1) que calcula raíces cuadradas por el método de la bisección. El programa trabaja en doble precisión (variables `REAL*8`). Al utilizar el programa para calcular $\sqrt{2}$ con $x^0 = 1$, $a = 2$, y tolerancias de convergencia (ver apartado siguiente) de $tol_x = TOL_f = 0.5 \cdot 10^{-8}$, se obtienen los resultados de la tabla 5.1.

Tabla 5.1 Cálculo de $\sqrt{2}$ por el método de la bisección a partir de $x^0 = 1$ y $a = 2$

Iteracion	Extremo a	Aproximacion x	f(x)	Error relativo en x
0	2.0000000	1.0000000	-1.000D+00	-3.333D-01
1	1.0000000	1.5000000	2.500D-01	2.000D-01
2	1.5000000	1.2500000	-4.375D-01	-9.091D-02
3	1.5000000	1.3750000	-1.094D-01	-4.348D-02
4	1.3750000	1.4375000	6.641D-02	2.222D-02
5	1.4375000	1.4062500	-2.246D-02	-1.099D-02
6	1.4062500	1.4218750	2.173D-02	5.525D-03
7	1.4218750	1.4140625	-4.272D-04	-2.755D-03
8	1.4140625	1.4179688	1.064D-02	1.379D-03
9	1.4140625	1.4160156	5.100D-03	6.901D-04
10	1.4140625	1.4150391	2.336D-03	3.452D-04
11	1.4140625	1.4145508	9.539D-04	1.726D-04
12	1.4140625	1.4143066	2.633D-04	8.632D-05
13	1.4143066	1.4141846	-8.200D-05	-4.316D-05
14	1.4141846	1.4142456	9.063D-05	2.158D-05
15	1.4141846	1.4142151	4.315D-06	1.079D-05
16	1.4142151	1.4141998	-3.884D-05	-5.395D-06
17	1.4142151	1.4142075	-1.726D-05	-2.697D-06
18	1.4142151	1.4142113	-6.475D-06	-1.349D-06
19	1.4142151	1.4142132	-1.080D-06	-6.743D-07
20	1.4142132	1.4142141	1.617D-06	3.372D-07
21	1.4142132	1.4142137	2.687D-07	1.686D-07
22	1.4142137	1.4142134	-4.056D-07	-8.429D-08
23	1.4142137	1.4142135	-6.846D-08	-4.215D-08
24	1.4142135	1.4142136	1.001D-07	2.107D-08
25	1.4142135	1.4142136	1.584D-08	1.054D-08
26	1.4142136	1.4142136	-2.631D-08	-5.268D-09
27	1.4142136	1.4142136	-5.237D-09	-2.634D-09
28	1.4142136	1.4142136	5.300D-09	1.317D-09
29	1.4142136	1.4142136	3.154D-11	6.585D-10

```
Convergencia en la iteracion 29
La raiz cuadrada de  2.0000000 es  1.4142136
```

Los valores de las tolerancias utilizados en el ejemplo de la tabla 5.1 son muy estrictos, y se utilizan en este capítulo para distinguir bien el comportamiento relativo de los distintos métodos.

Nótese que, con estas tolerancias, se obtienen ocho cifras significativas en los resultados. Puede verse en la tabla 5.1 cómo el intervalo inicial $[1,2]$ se va subdividiendo hasta llegar, después de 29 iteraciones, a $x^{29} = 1.4142136$, que se toma como aproximación a $\sqrt{2}$. Otra entrada del programa es la variable `MAXITER`, que representa el número de iteraciones que se desea realizar *como máximo*. En caso de alcanzarse este valor, el programa finalizará sin ningún mensaje de convergencia. Habitualmente, al detectarse este fenómeno se debe ejecutar de nuevo el programa inicializándose el método más cerca de la solución.

Considérese ahora al problema del billar circular. Tomando como punto de partida el programa 5.1, se escribe el programa 5.2 (apartado 5.7) que resuelve la ecuación 5.3 en lugar de la ecuación 5.2. Se toman los valores $R = 1$, $x_P = 0.6$, $y_P = 0$, $x_Q = -0.6$, $y_Q = 0$ (véase la figura 5.4). Un punto de impacto I viene dado entonces por $\theta = \pi/2$. Se puede capturar esta solución tomando $x^0 = 1.5 < \pi/2$, $a = 1.6 > \pi/2$. Si se mantienen las tolerancias $tol_x = TOL_f = 0.5 \cdot 10^{-8}$, el método de la bisección proporciona los resultados de la tabla 5.2. Efectivamente, en 23 iteraciones se obtiene $\theta^{23} = 1.5707963 \approx \pi/2$.

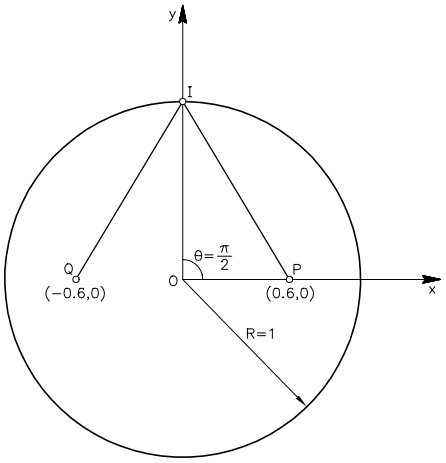


Fig. 5.4 Problema del billar para $R = 1$, $x_P = 0.6$, $y_P = 0$, $x_Q = -0.6$ *e* $y_Q = 0$

En cada iteración del método de la bisección es necesario evaluar $f(x^k)$ (paso 5). Para ello, en el programa FORTRAN se trabaja con una función externa (`FUNCTION`). Se ha visto en el capítulo 3 que el FORTRAN dispone de una biblioteca de funciones intrínsecas (trigonométricas, logarítmicas, exponenciales, etc.) ya incorporadas. Las `FUNCTION`s, por el contrario, son funciones de usuario que se pueden definir a voluntad para resolver un problema concreto. Por ejemplo, para el problema del billar circular se ha definido $f(\theta)$ (ver ecuación 5.3) como una `FUNCTION`. En el apartado 5.7 se explica como definir y trabajar con funciones externas `FUNCTION`.

Problema 5.1:

a) Determinar, por simple inspección visual, cuáles son los demás ceros de la función $f(\theta)$ (ecuación 5.3) para los datos de la figura 5.4 (Pista: hay un total

de cuatro ceros). ¿Son todos estos ceros soluciones válidas desde un punto de vista físico?

b) Verificar que los valores obtenidos en el apartado *a* son realmente ceros de $f(\theta)$ utilizando el método de la bisección. Justificar razonadamente los resultados obtenidos. ●

Tabla 5.2 Obtención de la solución $\theta = \pi/2$ por el método de la bisección

Iteracion	Extremo a	Aproximacion x	f(x)	Error relativo en x
0	1.6000000	1.5000000	3.211D-02	-3.226D-02
1	1.6000000	1.5500000	9.440D-03	-1.587D-02
2	1.5500000	1.5750000	-1.908D-03	8.000D-03
3	1.5750000	1.5625000	3.766D-03	-3.984D-03
4	1.5750000	1.5687500	9.290D-04	-1.988D-03
5	1.5687500	1.5718750	-4.897D-04	9.950D-04
6	1.5718750	1.5703125	2.196D-04	-4.973D-04
7	1.5703125	1.5710938	-1.350D-04	2.487D-04
8	1.5710938	1.5707031	4.231D-05	-1.243D-04
9	1.5707031	1.5708984	-4.635D-05	6.217D-05
10	1.5707031	1.5708008	-2.022D-06	3.109D-05
11	1.5708008	1.5707520	2.014D-05	-1.554D-05
12	1.5708008	1.5707764	9.061D-06	-7.771D-06
13	1.5708008	1.5707886	3.519D-06	-3.886D-06
14	1.5708008	1.5707947	7.486D-07	-1.943D-06
15	1.5707947	1.5707977	-6.368D-07	9.714D-07
16	1.5707977	1.5707962	5.592D-08	-4.857D-07
17	1.5707962	1.5707970	-2.904D-07	2.429D-07
18	1.5707962	1.5707966	-1.173D-07	1.214D-07
19	1.5707962	1.5707964	-3.067D-08	6.071D-08
20	1.5707964	1.5707963	1.263D-08	-3.036D-08
21	1.5707963	1.5707963	-9.020D-09	1.518D-08
22	1.5707963	1.5707963	1.803D-09	-7.589D-09
23	1.5707963	1.5707963	-3.608D-09	3.795D-09

```
Convergencia en la iteracion 23
Solucion para theta=  1.5707963
```

5.3 Criterios de convergencia

Se dice que la sucesión $\{x^k\}$ *converge* a x^* si

$$\lim_{k \to \infty} x^k = x^*$$

que puede ponerse también como

$$\lim_{k\to\infty} E^k = \lim_{k\to\infty} (x^k - x^*) = 0 \tag{5.4}$$

donde E^k es el *error absoluto* de la aproximación x^k a x^*. Es decir, la sucesión converge si el error absoluto tiende a cero cuando el contador de iteraciones k tiende a infinito.

Al dividir la ecuación 5.4 por x^* (suponiendo $x^* \neq 0$) se obtiene

$$\lim_{k\to\infty} r^k = \lim_{k\to\infty} \frac{x^k - x^*}{x^*} = 0 \tag{5.5}$$

donde r^k es el *error relativo* de la aproximación x^k.

Para aceptar una aproximación x^k como suficientemente buena se exige que su error relativo r^k sea, en valor absoluto, inferior a una tolerancia preestablecida tol_x:

$$|r^k| < tol_x \tag{5.6}$$

Sin embargo, a la vista de la expresión de r^k (ecuación 5.5) está claro que en la práctica *no* se puede calcular r^k, puesto que para ello sería necesario conocer el cero x^*. Dado que x^* es precisamente la incógnita del problema, se hace la siguiente aproximación:

$$r^k \approx \frac{x^k - x^{k+1}}{x^{k+1}} \tag{5.7}$$

En la ecuación 5.7 se ha sustituido x^* por la aproximación en la *siguiente* iteración, x^{k+1}. La idea es que, si la sucesión converge, entonces x^{k+1} es más próxima al cero x^* que x^k y puede utilizarse como valor aproximado de referencia. Combinando las ecuaciones 5.6 y 5.7 se obtiene el *criterio práctico de convergencia*

$$\left|\frac{x^k - x^{k+1}}{x^{k+1}}\right| < tol_x \tag{5.8}$$

Este criterio de convergencia fallaría si $x^* = 0$ (división por cero), porque el error relativo dejaría de estar definido. Se haría necesario entonces trabajar con errores absolutos, reescribiendo la ecuación 5.8 como

$$\left|x^k - x^{k+1}\right| < tol_x \left|x^{k+1}\right| + E \tag{5.9}$$

donde E es una cota del error absoluto $\left|x^k - x^{k+1}\right|$ para el caso $x^* = 0$. Típicamente E se escoge órdenes de magnitud menor que tol_x. De esta forma, para $x^* \neq 0$, el criterio de convergencia 5.9 coincide prácticamente con el criterio de convergencia 5.8.

Desde un punto de vista algorítmico, los criterios de convergencia 5.8 o 5.9 implican que para decidir si x^k es o no una aproximación suficientemente buena a x^* es necesario calcular la siguiente aproximación x^{k+1}.

Debido a la aproximación hecha en la ecuación 5.7, puede ocurrir en algunos casos que la condición de convergencia dada por la ecuación 5.8 se cumpla estando x^k lejos de x^*. Para evitar estos problemas, se complementa el criterio relativo en x (ecuación 5.8) con un criterio absoluto en f.

Para ello, basta darse cuenta de que, si $\{x^k\}$ converge a x^*, se verifica también

$$\lim_{k\to\infty} f(x^k) = 0$$

puesto que $f(x^*) = 0$. Esto significa que $f(x^k)$ es directamente el *error absoluto* en f. Para aceptar x^k como aproximación final a x^* se exige —además de la condición 5.8— que este error absoluto sea, en valor absoluto, inferior a una tolerancia TOL_f preestablecida:

$$|f(x^k)| < TOL_f$$

5.4 Método de Newton

Se ha comprobado en el apartado 5.2 que el método de la bisección es una técnica *robusta* para hallar ceros de funciones: basta que f sea una función continua, que el intervalo inicial definido por x^0 y a contenga un cero x^* y que f tenga signos distintos en los extremos del intervalo ($f(x^0)f(a) < 0$) para *garantizar* que el método va "encerrando" a x^* en un intervalo cada vez más pequeño. La longitud del intervalo final puede controlarse mediante las tolerancias de convergencia. Sin embargo, los dos ejemplos del apartado 5.2 también muestran que la bisección es una técnica *lenta*: para las tolerancias exigidas, han sido necesarias entre 20 y 30 iteraciones para alcanzar la convergencia.

Una técnica más rápida (aunque no tan robusta, como se verá) es el *método de Newton*. Se hará en primer lugar una deducción analítica del método y luego una deducción gráfica.

5.4.1 Deducción analítica del método de Newton

Supóngase que x^k es una aproximación a un cero x^* de una cierta función f. Puesto que $x^k \neq x^*$, resulta

$$f(x^k) \neq 0$$

Dado que x^k *no* es el cero x^* buscado, se intenta que la siguiente aproximación x^{k+1} *sí* lo sea. Para ello se define x^{k+1} como

$$x^{k+1} = x^k + \Delta x^{k+1} \tag{5.10}$$

donde Δx^{k+1} es la *corrección* que se hace a x^k para obtener x^{k+1}. El criterio para calcular esta corrección Δx^{k+1} es precisamente imponer que x^{k+1} sea un cero de f, es decir, $f(x^{k+1}) = 0$. Teniendo en cuenta la ecuación 5.10, esto se escribe como

$$f(x^k + \Delta x^{k+1}) = 0 \tag{5.11}$$

Está claro que para una función f arbitraria *no* es posible despejar Δx^{k+1} en la ecuación 5.11. Por este motivo, se hace un *desarrollo en serie de Taylor de primer orden* de f alrededor de x^k, y se obtiene

$$f(x^k + \Delta x^{k+1}) \approx f(x^k) + f'(x^k)\Delta x^{k+1} \tag{5.12}$$

Es importante resaltar que la ecuación 5.12 es una *aproximación* y no una igualdad, porque se han despreciado los términos del desarrollo de Taylor con derivadas de orden superior a uno.

Si ahora se sustituye $f(x^k + \Delta x^{k+1})$ en la ecuación 5.11 por la aproximación obtenida en 5.12, resulta

$$f(x^k) + f'(x^k)\Delta x^{k+1} = 0$$

de donde puede aislarse Δx^{k+1} como

$$\Delta x^{k+1} = -\frac{f(x^k)}{f'(x^k)} \tag{5.13}$$

siempre que $f'(x^k) \neq 0$. Finalmente, reemplazando esta expresión de Δx^{k+1} en la ecuación 5.10 se llega a la expresión del método de Newton:

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)} \tag{5.14}$$

La ecuación 5.14 proporciona una estrategia para construir la sucesión de aproximaciones $\{x^k\}$ a un cero x^*. Como ya se ha comentado, para completar el método es necesario elegir una aproximación inicial x^0 y un criterio de finalización de iteraciones. En cuanto a este último punto, se emplean los mismos criterios de convergencia (relativo en x y absoluto en f) que para el método de la bisección (ver apartado 5.3).

5.4.2 Deducción gráfica del método de Newton

El método de Newton puede deducirse también de manera gráfica, tal y como se muestra en la figura 5.5. La idea es la siguiente: dada una cierta x^k, se aproxima la función f por la *recta tangente* a la curva $y = f(x)$ en el punto $(x^k, f(x^k))$. La pendiente de esta recta es justamente la derivada de f en x^k. A continuación se toma la intersección de esta recta con el eje de las x como siguiente aproximación x^{k+1}. En la figura 5.5 se puede observar que $f(x^k)$, Δx^{k+1} y la pendiente $f'(x^k)$ están relacionados según

$$\frac{f(x^k)}{-\Delta x^{k+1}} = f'(x^k)$$

que es equivalente a la ecuación 5.13 obtenida en la deducción analítica del método.

Se ha escrito un programa en FORTRAN para calcular raíces cuadradas a partir de operaciones elementales mediante el método de Newton (programa 5.3, apartado 5.7). En este caso, es necesario definir dos funciones externas `FUNCTION`: una para la función f y otra para su derivada f'.

Si se emplea el programa 5.3 para calcular $\sqrt{2}$ con los mismos datos utilizados para el método de la bisección ($x^0 = 1$, $tol_x = TOL_f = 0.5 \cdot 10^{-8}$), el método de Newton arroja los resultados de la tabla 5.3.

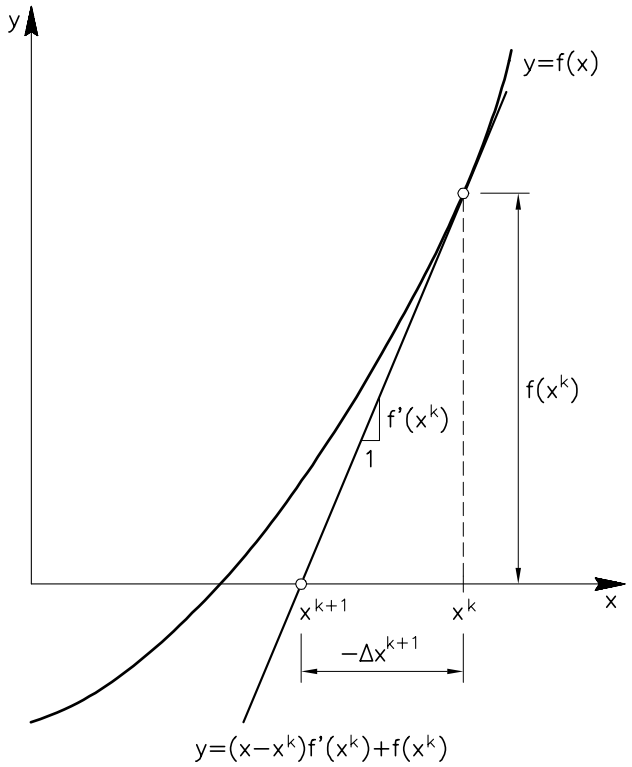


Fig. 5.5 Deducción gráfica del método de Newton

Tabla 5.3 Cálculo de $\sqrt{2}$ por el método de Newton a partir de $x^0 = 1$

```
 Iteracion   Aproximacion x       f(x)       Error relativo en x
 =========   ==============    ==========    ===================
     0          1.0000000      -1.000D+00        -3.333D-01
     1          1.5000000       2.500D-01         5.882D-02
     2          1.4166667       6.944D-03         1.733D-03
     3          1.4142157       6.007D-06         1.502D-06
     4          1.4142136       4.511D-12         1.128D-12

Convergencia en la iteracion  4
La raiz cuadrada de  2.0000000 es  1.4142136
```

Las tablas 5.1 y 5.3 ponen de manifiesto que la convergencia a $\sqrt{2}$ es mucho más rápida para el método de Newton que para el método de la bisección. Esto es debido a que el método de Newton se basa en una estrategia muy "inteligente": a medida que la aproximación x^k se va acercando al cero x^* de f, la recta tangente se va pareciendo parece cada vez más a la curva $y = f(x)$, hasta confundirse con ella (véase la figura 5.6).

Debido a su rapidez, el método de Newton es ampliamente utilizado en la práctica. Sin embargo, no es tan robusto como el método de la bisección, tal como se ilustra en el problema 5.2.

Problema 5.2:
Se desea calcular $\sqrt{2}$ tomando $x^0 = 0$ como aproximación inicial. Verificar que:
a) puede hacerse sin dificultades mediante el método de la bisección (con $a = 2$, por ejemplo).
b) el método de Newton falla. ¿Por qué? ●

Problema 5.3:
a) Escribir un programa en FORTRAN que resuelva el problema del billar circular mediante el método de Newton.
b) Utilizar el programa para hallar las soluciones con los datos de la figura 5.4. ●

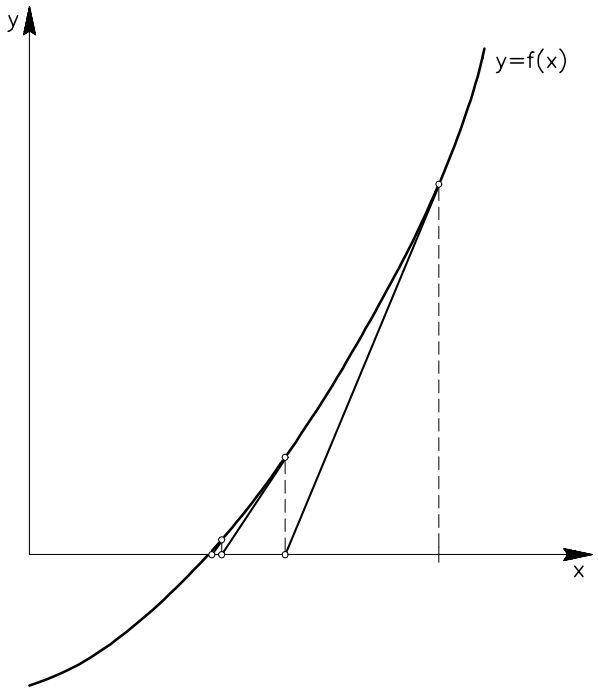


Fig. 5.6 ¡El método de Newton es rápido!

5.5 Método de la secante

Se ha visto en el apartado anterior que la rapidez del método de Newton es debida a la utilización de la recta tangente en cada punto. Ello obliga a efectuar *dos* evaluaciones funcionales en cada iteración (la función y su derivada), en lugar de trabajar únicamente con valores de la función. Esto hace que el *coste computacional* por iteración sea más elevado para el método de Newton que para el método de la bisección. En los ejemplos expuestos aquí el cálculo de f y f' no es excesivamente costoso, pero es bastante común en problemas reales que la evaluación de f y f' requiera resolver un problema complejo cada vez.

Por otro lado, puede ocurrir que para un determinado problema el cálculo de derivadas sea muy farragoso o incluso imposible (por ejemplo, si no se dispone de una expresión analítica de $f(x)$). En estos casos, se puede utilizar el *método de la secante*. La idea básica de este método se ilustra en la figura 5.7: la recta tangente a la curva $y = f(x)$ en el punto $(x^k, f(x^k))$ se aproxima mediante la *recta secante* que pasa por este punto y el punto $(x^{k-1}, f(x^{k-1}))$, obtenido en la iteración anterior. En otras palabras, la derivada $f'(x^k)$ no se calcula, sino que se aproxima por

$$f'(x^k) \approx \frac{f(x^k) - f(x^{k-1})}{x^k - x^{k-1}}$$

La intersección de esta recta secante con el eje de las x se toma como siguiente aproximación x^{k+1}. Este método necesita *dos* aproximaciones iniciales (x^0 y x^1) para poder trazar la primera recta secante.

El programa 5.4 (ver apartado 5.7) es un programa FORTRAN que halla raíces cuadradas mediante el método de la secante. Al calcular $\sqrt{2}$ con $x^0 = 1$, $x^1 = 1.5$ (es decir, las dos primeras aproximaciones del método de Newton, véase la tabla 5.3) y $tol_x = TOL_f = 0.5 \cdot 10^{-8}$, se obtienen los resultados de la tabla 5.4. Comparando las tablas 5.1, 5.3 y 5.4 puede verse que la convergencia del método de la secante es ligeramente más lenta que la del método de Newton, pero claramente más rápida que la del método de la bisección.

Tabla 5.4 Cálculo de $\sqrt{2}$ por el método de la secante a partir de $x^0 = 1$ y $x^1 = 1.5$

```
   Iteracion   Aproximacion x       f(x)       Error relativo en x
   =========   ==============    ==========    ===================
       0          1.0000000      -1.000D+00        -3.333D-01
       1          1.5000000       2.500D-01         7.143D-02
       2          1.4000000      -4.000D-02        -9.756D-03
       3          1.4137931      -1.189D-03        -2.988D-04
       4          1.4142157       6.007D-06         1.502D-06
       5          1.4142136      -8.931D-10        -2.233D-10

 Convergencia en la iteracion  5
 La raiz cuadrada de  2.0000000 es  1.4142136
```

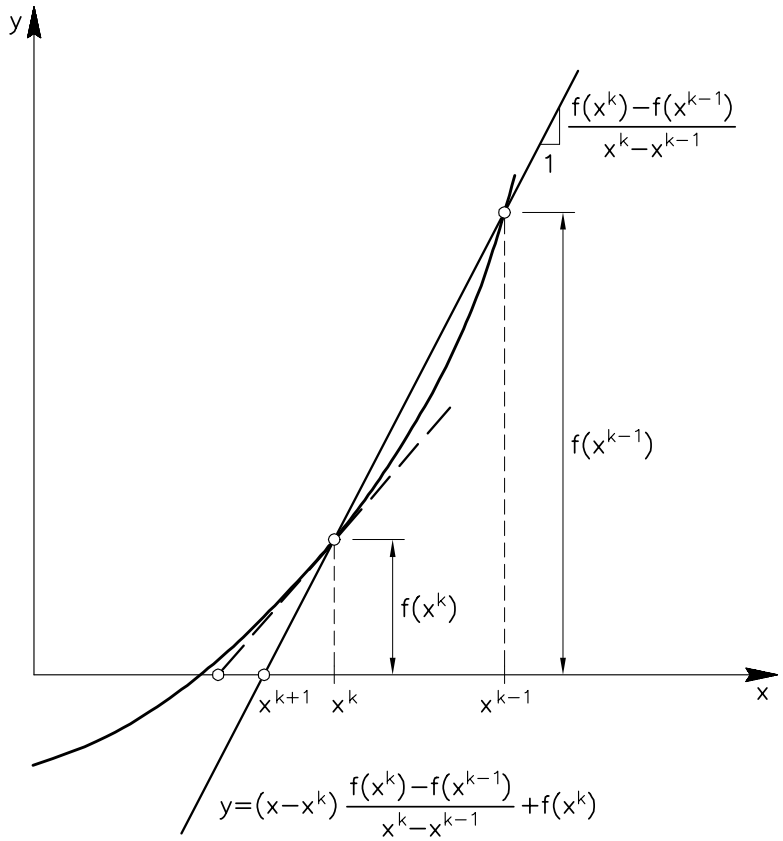


Fig. 5.7 Método de la secante

Problema 5.4:

a) Escribir un programa en FORTRAN que resuelva el problema del billar circular mediante el método de la secante.

b) Utilizar el programa para hallar las soluciones con los datos de la figura 5.4.

●

5.6 Gráficas de convergencia

Una manera habitual de presentar los resultados de convergencia de los distintos métodos es mediante gráficas que muestran cómo decrece el error relativo (en escala logarítmica) al aumentar el número de iteraciones.

En la figura 5.8 se muestran las curvas correspondientes a los tres métodos expuestos para el problema del cálculo de $\sqrt{2}$. La mayor rapidez del método de Newton queda reflejada en una curva que decrece mucho en pocas iteraciones. En el otro extremo está el método de la

bisección, con un decrecimiento mucho más lento del error.

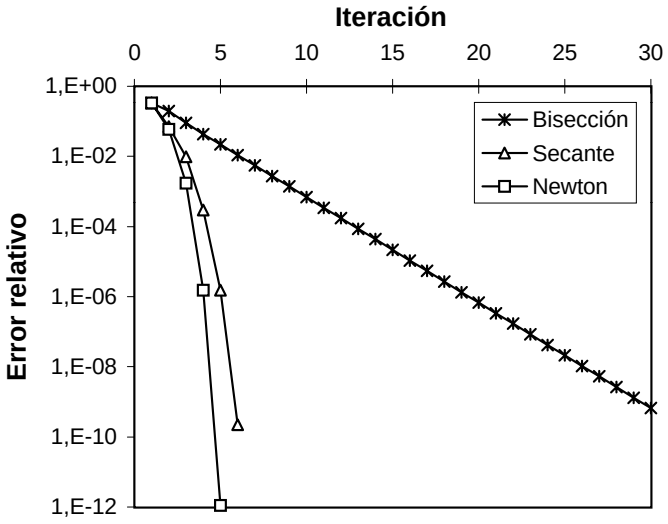


Fig. 5.8 Comparación de los tres métodos para el cálculo de $\sqrt{2}$

Problema 5.5: Representar en gráficas de convergencia los resultados obtenidos en los problemas 5.1, 5.3 y 5.4, correspondientes a la resolución del problema del billar circular por los métodos de bisección, Newton y secante. Comparar razonadamente las distintas curvas. ●

Problema 5.6: Escribir un programa FORTRAN para obtener el(los) cero(s) de la función $f(x) = x^2 - 2x + 1$ mediante los métodos de la bisección, de Newton y de la secante (el usuario debe poder elegir, al principio de la ejecución, qué método desea emplear). Comentar razonadamente los resultados obtenidos. ¿Presentan los métodos su comportamiento típico? ●

5.7 Aspectos computacionales: las funciones externas `FUNCTION` en FORTRAN

La sentencia `FUNCTION` permite al programador definir sus propias funciones. Estas funciones se llaman *externas* para distinguirlas de las funciones intrínsecas, ya incorporadas en el lenguaje FORTRAN.

Supóngase que en un programa es necesario calcular el área de muchos triángulos distintos A a partir de su base y su altura ($A = (bh)/2$). Para no tener que programar esta fórmula

repetidamente a lo largo del programa principal, se define como una función externa en un subprograma:

```
function area(base,altura)

area = (base*altura)/2.
return
end
```

Cada vez que sea necesario evaluar el área de un triángulo, desde el programa principal puede utilizarse la función externa haciendo:

```
a = area(b,h)
```

Tal como se puede observar en el ejemplo:

- La función externa se declara con la sentencia `FUNCTION` seguida del nombre de la función y de sus argumentos entre paréntesis.

- Puesto que se trata de un subprograma independiente del programa principal, es necesario terminar la función externa con las sentencias `RETURN` (para volver al programa principal) y `END` (para indicar el final del subprograma). Ambos programas (el principal y el subprograma) pueden estar en un mismo archivo o en archivos distintos.

- Los nombres de los argumentos de la función pueden ser distintos en la llamada desde el programa principal (`b, h`) y en el subprograma (`base, altura`). Lo único que importa es el orden que ocupan en la lista de argumentos, su nombre es indiferente.

- Debe existir concordancia de tipo entre la función y el resultado que devuelve. Es decir que, por ejemplo, utilizando declaraciones por defecto, si el resultado arrojado por la función es un `INTEGER*4`, el nombre de la función debe empezar por `I`, `J`, `K`, `L`, `M` o `N`, mientras que si es un `REAL*4` la inicial del nombre de la función debe estar comprendida entre `A-H` y `O-Z`. Si el resultado es de otro tipo (`INTEGER*2`, `REAL*8`, `REAL*16`, `COMPLEX`, `LOGICAL` o `CHARACTER`), debe constar explícitamente en la declaración de la función. Así, por ejemplo, para calcular el área de un triángulo en doble precisión se puede utilizar la función externa

  ```
  real*8 function area(base,altura)

  implicit real*8 (a-h,o-z)
  area = (base*altura)/2.d0
  return
  end
  ```

El programa 5.1, empleado en el apartado 5.2 para calcular raíces de ecuaciones del tipo $f(x) = 0$ por el método de la bisección, utiliza una `FUNCTION` para programar la función f. Como puede observarse, la función externa `FUNCTION` que evalúa la función f es del mismo tipo (`REAL*8`) que la variable `F` a la que está asignada, y en el interior es necesario volver a declarar

todas las variables (en este caso, mediante la sentencia `IMPLICIT`).

```
c
c        Este programa calcula raices cuadradas a partir de
c        operaciones aritmeticas elementales mediante el
c        metodo de la BISECCION
c
c-----------------------------------------------------------------
      implicit real*8 (a-h,o-z)
c___Numero s cuya raiz cuadrada se desea calcular
      write (6,10)
      read  (5,*) s
 10   format (1x,'Raiz cuadrada de:')

c___Aproximacion inicial x0
      write (6,20)
      read  (5,*) x
 20   format (1x,'Aproximacion inicial')

c___Extremo a del intervalo inicial
      write (6,30)
      read  (5,*) a
 30   format (1x,'Extremo a del intervalo inicial')

c___Tolerancia en x
      write (6,40)
      read  (5,*) tol_x
 40   format (1x,'Tolerancia en x')

c___Tolerancia en f
      write (6,50)
      read  (5,*) tol_f
 50   format (1x,'Tolerancia en f')

c___Numero maximo de iteraciones
      write (6,60)
      read  (5,*) maxiter
 60   format (1x,'Numero maximo de iteraciones')

c___Titulos de la salida de resultados
      write (6,510)
 510  format (/,1x, 'Iteracion', 3x, 'Extremo a', 4x,
```

```
     .          'Aproximacion x', 4x, '   f(x)   ', 4x,
     .          'Error relativo en x')
      write (6,520)
 520  format (1x, '=========', 3x, '==========',
     .        3x, '==============', 4x, '==========',
     .        4x, '====================')

c___Control para decidir si se puede iniciar la biseccion
c___Tienen f(a) y f(x0) signos opuestos?
      fa = f(a,s)
      fx = f(x,s)
      if( (fa*fx) .gt. 0.d0 ) then
        write (6,*) ' No se cumple la condicion f(a)*f(x0) < 0'
        stop
      endif

c___Inicio del proceso iterativo
c___del metodo de la biseccion

      do k=0, maxiter

         pmedio = (a+x)/2.d0
         fmedio = f(pmedio,s)

c___Calculo del error relativo en x
         rel_x = (x - pmedio)/pmedio

c___Impresion de resultados
         write (6,600) k,a,x,fx,rel_x
 600     format (3x, i3, 7x, f10.7, 5x, f10.7, 6x, 1pd10.3,
     .           8x, 1pd10.3)

c___Control de convergencia
         if ( (abs(fx).lt.tol_f) .and. (abs(rel_x).lt.tol_x) ) then
           write (6,700) k
 700       format (/,1x, 'Convergencia en la iteracion', i3)
           write (6,800) s,x
 800       format (1x,'La raiz cuadrada de ', f10.7, ' es ', f10.7)
           stop
         endif

c___Eleccion del nuevo intervalo segun el valor de f(pmedio)
         if ( (fmedio*fx) .lt. 0.d0 ) then
           a  = x
           x  = pmedio
           fx = fmedio
```

```
          else
            x  = pmedio
            fx = fmedio
          endif

       enddo

       end

c_____________________Function f(x,s)
       real*8 function f(x,s)
       implicit real*8 (a-h,o-z)
       f = x*x - s
       return
       end
```

Prog. 5.1 Cálculo de raíces cuadradas por el método de la bisección

Como se puede ver, la utilización de las sentencias `FUNCTION` permite realizar de manera natural una programación en módulos (*programación estructurada*); este aspecto se comentará en detalle en el capítulo 7. En este sentido, es interesante resaltar que si se desea utilizar el programa 5.1 para hallar el cero de otra función distinta, sólo es necesario modificar la `FUNCTION` correspondiente (y la entrada de datos necesarios para este caso). Por ejemplo, el programa 5.2 se ha obtenido modificando adecuadamente el programa 5.1 presentado anteriormente para hallar las raíces de la ecuación 5.3 (juego del billar en una mesa circular) mediante el método de la bisección.

```
c
c      Este programa resuelve el problema del billar circular
c      mediante el metodo de la BISECCION
c
c_____________________________________________________________________
       implicit real*8 (a-h,o-z)
c___Radio R de la mesa de billar y posicion de las dos bolas P y Q
       write (6,100)
       read  (5,*) r
       write (6,110)
       read  (5,*) xp,yp
       write (6,120)
       read  (5,*) xq,yq

  100  format (1x,'Radio de la mesa:')
  110  format (1x,'Coordenadas de la bola P:')
  120  format (1x,'Coordenadas de la bola Q:')
```

```
c___Aproximacion inicial x0
      write (6,200)
      read  (5,*) x
 200  format (1x,'Aproximacion inicial')

c___Extremo a del intervalo inicial
      write (6,210)
      read  (5,*) a
 210  format (1x,'Extremo a del intervalo inicial')

c___Tolerancia en x
      write (6,220)
      read  (5,*) tol_x
 220  format (1x,'Tolerancia en x')

c___Tolerancia en f
      write (6,230)
      read  (5,*) tol_f
 230  format (1x,'Tolerancia en f')

c___Numero maximo de iteraciones
      write (6,240)
      read  (5,*) maxiter
 240  format (1x,'Numero maximo de iteraciones')

c___Titulos de la salida de resultados
      write (6,510)
 510  format (/,1x, 'Iteracion', 3x, 'Extremo a', 4x,
     .          'Aproximacion x', 4x, '   f(x)   ', 4x,
     .          'Error relativo en x')
      write (6,520)
 520  format (1x, '=========', 3x, '==========',
     .        3x, '==============', 4x, '==========',
     .        4x, '==================')

c___Control para decidir si se puede iniciar la biseccion
c___Tienen f(a) y f(x0) signos opuestos?
      fa = f(a,r,xp,yp,xq,yq)
      fx = f(x,r,xp,yp,xq,yq)
      if( (fa*fx) .gt. 0.d0 ) then
        write (6,*) ' No se cumple la condicion f(a)*f(x0) < 0'
        stop
      endif

c___Inicio del proceso iterativo
c___del metodo de la biseccion
```

```
      do k=0, maxiter

         pmedio = (a+x)/2.d0
         fmedio = f(pmedio,r,xp,yp,xq,yq)

c___Calculo del error relativo en x
         rel_x = (x - pmedio)/pmedio

c___Impresion de resultados
         write (6,600) k,a,x,fx,rel_x
 600     format (3x, i3, 7x, f10.7, 5x, f10.7, 6x, 1pd10.3,
     .        8x, 1pd10.3)

c___Control de convergencia
         if ( (abs(fx).lt.tol_f) .and. (abs(rel_x).lt.tol_x) ) then
           write (6,700) k
 700       format (/,1x, 'Convergencia en la iteracion', i3)
           write (6,800) x
 800       format (1x,'Solucion para theta= ', f10.7)
           stop
         endif

c___Eleccion del nuevo intervalo segun el valor de f(pmedio)
         if ( (fmedio*fx) .lt. 0.d0 ) then
           a  = x
           x  = pmedio
           fx = fmedio
         else
           x  = pmedio
           fx = fmedio
         endif

      enddo

      end

c___________________Function f(theta)
      real*8 function f(theta,r,xp,yp,xq,yq)
      implicit real*8 (a-h,o-z)

      stheta = sin(theta)
      ctheta = cos(theta)
      distp  = ((r*ctheta-xp)*(r*ctheta-xp)+
     .          (r*stheta-yp)*(r*stheta-yp))**0.5
```

```
      distq  = ((r*ctheta-xq)*(r*ctheta-xq)+
     .          (r*stheta-yq)*(r*stheta-yq))**0.5
      f      = ((xp*stheta-yp*ctheta)/distp) +
     .          ((xq*stheta-yq*ctheta)/distq)

      return
      end
```

Prog. 5.2 Cálculo de las raíces de la ecuación 5.3 por el método de la bisección

En el programa 5.3 (método de Newton) se hace necesario utilizar dos `FUNCTION`s; la primera calcula los valores de $f(x)$ y la segunda, los de su derivada $f'(x)$. De nuevo, si se desea utilizar este programa para hallar los ceros de otra función distinta, sólo es preciso modificar estos dos módulos.

```
c
c        Este programa calcula raices cuadradas a partir de
c        operaciones aritmeticas elementales mediante el
c        metodo de NEWTON
c
c-----------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
c___Numero s cuya raiz cuadrada se desea calcular
      write (6,10)
      read  (5,*) s
 10   format (1x,'Raiz cuadrada de:')

c___Aproximacion inicial x0
      write (6,20)
      read  (5,*) x_actual
 20   format (1x,'Aproximacion inicial')

c___Tolerancia en x
      write (6,30)
      read  (5,*) tol_x
 30   format (1x,'Tolerancia en x')

c___Tolerancia en f
      write (6,40)
      read  (5,*) tol_f
 40   format (1x,'Tolerancia en f')
```

```
c___Numero maximo de iteraciones
      write (6,50)
      read  (5,*) maxiter
 50   format (1x,'Numero maximo de iteraciones')

c___Titulos de la salida de resultados
      write (6,510)
 510  format (/,1x, 'Iteracion', 3x, 'Aproximacion x',
     .          4x, '   f(x)   ', 4x, 'Error relativo en x')
      write (6,520)
 520  format (1x, '=========', 3x, '==============',
     .        4x, '==========', 4x, '==================')

c___Inicio del proceso iterativo
c___del metodo de Newton

      do k=0, maxiter

c___Nueva aproximacion
         fx =  funcion(x_actual,s)
         dx = derivada(x_actual)
         x_nuevo = x_actual - (fx/dx)

c___Calculo del error relativo en x
         rel_x = (x_actual - x_nuevo)/x_nuevo

c___Impresion de resultados
         write (6,600) k,x_actual,fx,rel_x
 600     format (3x, i3, 9x, f10.7, 6x, 1pd10.3, 8x, 1pd10.3)

c___Control de convergencia
         if ((abs(fx) .lt. tol_f) .and. (abs(rel_x) .lt. tol_x)) then
           write (6,700) k
 700       format (/,1x, 'Convergencia en la iteracion', i3)
           write (6,800) s,x_actual
 800       format (1x,'La raiz cuadrada de ', f10.7, ' es ', f10.7)
           stop
         endif

         x_actual = x_nuevo

      enddo

      end
```

```
c__________________Function funcion(x,s)
      real*8 function funcion(x,s)
      implicit real*8 (a-h,o-z)
      funcion = x*x - s
      return
      end

c__________________Function derivada(x)
      real*8 function derivada(x)
      implicit real*8 (a-h,o-z)
      derivada = 2.d0*x
      return
      end
```

Prog. 5.3 Cálculo de raíces cuadradas por el método de Newton

Como se ha podido observar, la utilización adecuada de `FUNCTION`s facilita y rentabiliza considerablemente la programación. Incluso es posible reutilizar casi totalmente el programa 5.3 para implementar el método de la secante (ver programa 5.4).

```
c
c      Este programa calcula raices cuadradas a partir de
c      operaciones aritmeticas elementales mediante el
c      metodo de la SECANTE
c
c__________________Entrada de datos
      implicit real*8 (a-h,o-z)
c___Numero s cuya raiz cuadrada se desea calcular
      write (6,10)
      read  (5,*) s
 10   format (1x,'Raiz cuadrada de:')

c___Aproximaciones iniciales
      write (6,20)
      read  (5,*) x_actual
 20   format (1x,'Aproximacion inicial x0')

      write (6,30)
      read  (5,*) x_nuevo
 30   format (1x,'Aproximacion inicial x1')
```

```
c___Tolerancia en x
      write (6,40)
      read  (5,*) tol_x
 40   format (1x,'Tolerancia en x')

c___Tolerancia en f
      write (6,50)
      read  (5,*) tol_f
 50   format (1x,'Tolerancia en f')

c___Numero maximo de iteraciones
      write (6,60)
      read  (5,*) maxiter
 60   format (1x,'Numero maximo de iteraciones')

c___Titulos de la salida de resultados
      write (6,510)
 510  format (/,1x, 'Iteracion', 3x, 'Aproximacion x',
     .          4x, '   f(x)   ', 4x, 'Error relativo en x')
      write (6,520)
 520  format (1x, '=========', 3x, '==============',
     .        4x, '==========', 4x, '==================')

c___Inicio del proceso iterativo
c___del metodo de la secante

      do k=0, maxiter

c___Calculo del error relativo en x
         rel_x = (x_actual - x_nuevo)/x_nuevo

c___Valor de f en x_actual
         f_actual  = f(x_actual,s)

c___Impresion de resultados
         write (6,600) k,x_actual,f_actual,rel_x
 600     format (3x, i3, 9x, f10.7, 6x, 1pd10.3, 8x, 1pd10.3)

c___Control de convergencia
         if ((abs(f_actual).lt.tol_f).and.(abs(rel_x).lt.tol_x)) then
           write (6,700) k
 700       format (/,1x, 'Convergencia en la iteracion', i3)
           write (6,800) s,x_actual
 800       format (1x,'La raiz cuadrada de ', f10.7, ' es ', f10.7)
           stop
         endif
```

```
c___Nueva aproximacion
         f_nuevo   = f(x_nuevo,s)
         pendiente = (f_nuevo-f_actual)/(x_nuevo-x_actual)
         x_actual  = x_nuevo
         x_nuevo = x_actual - (f_nuevo/pendiente)

      enddo

      end

c__________________Function f(x,s)
      real*8 function f(x,s)
      implicit real*8 (a-h,o-z)
      f = x*x - s
      return
      end
```

Prog. 5.4 Cálculo de raíces cuadradas por el método de la secante

5.8 Bibliografía

BORSE, G.J. *Programación FORTRAN77 con aplicaciones de cálculo numérico en ciencias e ingeniería.* Anaya, 1989.

BREUER, S.; ZWAS, G. *Numerical Mathematics. A Laboratory Approach.* Cambridge University Press, 1993.

CHAPRA, S.C.; CANALE, R.P. *Métodos numéricos para ingenieros con aplicaciones en computadores personales.* McGraw-Hill, 1988.

HOFFMAN, J.D. *Numerical Methods for Engineers and Scientists.* McGraw-Hill, 1992.

6 Una introducción a los métodos gaussianos para sistemas lineales de ecuaciones

Objetivos

- Presentar y clasificar los métodos de resolución numérica de sistemas lineales de ecuaciones.
- Estudiar detalladamente los métodos directos: de solución inmediata, de eliminación y de descomposición.
- Desarrollar el análisis matricial del método de Gauss.

6.1 Consideraciones generales

6.1.1 Introducción

El objetivo de este tema es introducir al lector en la resolución de sistemas lineales de ecuaciones por métodos gaussianos. Parece conveniente, en primer lugar, establecer la notación y algunas de las bases de álgebra lineal necesarias para alcanzar el objetivo planteado.

Siguiendo la notación introducida por Householder en 1964, en general se emplean

mayúsculas en negrita	$\boldsymbol{A}, \boldsymbol{L}, \boldsymbol{U}, \boldsymbol{\Delta}, \boldsymbol{\Lambda}$	para las matrices,
minúsculas con subíndices	$a_{ij}, l_{ij}, u_{ij}, \delta_{ij}, \lambda_{ij}$	para los coeficientes de matrices,
minúsculas en negrita	$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$	para los vectores,
letras griegas en minúscula	$\alpha, \beta, \gamma, \delta, \theta, \mu$	para los escalares.

El espacio vectorial de las matrices reales $m \times n$ se denota por $\mathbb{R}^{m \times n}$; un elemento cualquiera de ese espacio, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, es una matriz rectangular de m filas y n columnas que puede escribirse

como

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1,n-1} & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2,n-1} & a_{2n} \\ \vdots & \vdots & & \vdots & \vdots \\ a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} & a_{m-1,n} \\ a_{m1} & a_{m2} & \cdots & a_{m,n-1} & a_{mn} \end{pmatrix}$$

Si $m = n$ entonces $\boldsymbol{A}$ es cuadrada y se dice que tiene orden n. De la misma forma, $\mathbb{C}^{m\times n}$ es el espacio vectorial de las matrices de coeficientes complejos.

Los vectores, que pueden ser interpretados como un caso particular del anterior con $\mathbb{R}^{n\times 1}$ (equivalente a $\mathbb{R}^n$), siempre se asumen como *vectores columna*, es decir $\boldsymbol{x} \in \mathbb{R}^n$ es

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}$$

donde las componentes x_i son números reales. Puesto que por convención se toman los vectores como columna, objetos del tipo $(x_1\ x_2\ \cdots\ x_{n-1}\ x_n)$ o bien $(x_1, \ldots, x_n)$ son *vectores fila* y se denotan por $\boldsymbol{x}^T$ (^{T} indica matriz o vector traspuesto).

Además de las operaciones inherentes al espacio vectorial (suma interna y producto exterior por reales) conviene resaltar por su importancia el producto escalar de vectores. Si $\boldsymbol{x}$ e $\boldsymbol{y}$ son dos vectores de $\mathbb{R}^n$, entonces $\boldsymbol{x}^T\boldsymbol{y} = x_1y_1 + x_2y_2 + \cdots + x_ny_n \in \mathbb{R}$. Nótese, que de forma equivalente, si $\boldsymbol{A}$ y $\boldsymbol{B}$ son dos matrices de $\mathbb{R}^{m\times n}$, entonces, $\boldsymbol{A}^T\boldsymbol{B} \in \mathbb{R}^{n\times n}$. El producto escalar de vectores permite definir una métrica: la norma euclídea de $\boldsymbol{x}$, que es simplemente $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^T\boldsymbol{x}}$.

Del mismo modo que se ha definido $\boldsymbol{x}^T\boldsymbol{y}$, también se puede definir $\boldsymbol{x}\boldsymbol{y}^T$. Sin embargo, el significado de este último producto entre vectores es radicalmente distinto. Sean $\boldsymbol{x}$ e $\boldsymbol{y}$ dos vectores no nulos; la matriz $\boldsymbol{x}\boldsymbol{y}^T$ es de $\mathbb{R}^{n\times n}$, se escribe como

$$\boldsymbol{x}\boldsymbol{y}^T = \begin{pmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_{n-1} & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_{n-1} & x_2y_n \\ \vdots & \vdots & & \vdots & \vdots \\ x_{n-1}y_1 & x_{n-1}y_2 & \cdots & x_{n-1}y_{n-1} & x_{n-1}y_n \\ x_ny_1 & x_ny_2 & \cdots & x_ny_{n-1} & x_ny_n \end{pmatrix}$$

y todas sus columnas (y filas) definen vectores paralelos, es decir, elementos de un espacio vectorial de dimensión uno. Por consiguiente, esta matriz es de rango uno. En realidad, toda matriz de rango uno puede expresarse como el producto de dos vectores de la forma $\boldsymbol{x}\boldsymbol{y}^T$. Estas matrices son comunes en métodos numéricos y conviene saber trabajar con ellas. Por ejemplo, su almacenamiento no se hace guardando todos los coeficientes de la matriz, lo que implicaría almacenar n^2 números reales; estas matrices se almacenan guardando únicamente las componentes de los vectores $\boldsymbol{x}$ e $\boldsymbol{y}$, es decir $2n$ números reales. Para tener una idea del ahorro computacional que esto representa basta suponer que $n = 1000$: mientras almacenar $\boldsymbol{x}$ e $\boldsymbol{y}$ sólo necesita de 2000 variables reales, $\boldsymbol{x}\boldsymbol{y}^T$ requiere un millón (es decir, 500 veces más memoria).

6.1.2 Planteamiento general

A lo largo de este tema se plantea la resolución de sistemas lineales de ecuaciones

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{6.1}$$

donde $\boldsymbol{A}$ es una matriz de $n \times n$ coeficientes reales a_{ij} con $i = 1, \ldots, n$, $j = 1, \ldots, n$; $\boldsymbol{b}$ es el término independiente, también de coeficientes reales, $\boldsymbol{b}^T = (b_1, \ldots, b_n)$; y finalmente $\boldsymbol{x}^T = (x_1, \ldots, x_n)$ es el vector solución del sistema.

La existencia y unicidad de soluciones del sistema definido en 6.1 es por fortuna un tema ampliamente estudiado en el álgebra lineal. Precisamente, el álgebra lineal proporciona una serie de condiciones que permiten verificar si 6.1 tiene solución:

Si $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, entonces las siguientes afirmaciones son equivalentes:

1. Para cualquier $\boldsymbol{b} \in \mathbb{R}^n$, el sistema $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ tiene solución.
2. Si $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ tiene solución, ésta es única.
3. Para cualquier $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0} \implies \boldsymbol{x} = \boldsymbol{0}$.
4. Las columnas (filas) de $\boldsymbol{A}$ son linealmente independientes.
5. Existe $\boldsymbol{A}^{-1}$ matriz inversa de $\boldsymbol{A}$ tal que $\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}$ ($\boldsymbol{I}$ matriz identidad de orden n).
6. $\det(\boldsymbol{A}) = |\boldsymbol{A}| \neq 0$.

A pesar de la indudable importancia de todas estas condiciones, en el ámbito de la resolución numérica de sistemas de ecuaciones deben ser empleadas con sumo cuidado.

6.1.3 Resolución algebraica: método de Cramer

A continuación se plantea la resolución analítica de problemas muy pequeños siguiendo un posible enfoque algebraico clásico. El sistema de ecuaciones planteado en 6.1 tiene solución única si y sólo si $\det(\boldsymbol{A}) = |\boldsymbol{A}| \neq 0$. En este caso, existe la matriz inversa de $\boldsymbol{A}$, $\boldsymbol{A}^{-1}$, que permite escribir la solución del sistema de ecuaciones como

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{6.2}$$

La ecuación anterior no es sólo una expresión formal de la solución sino que describe un posible algoritmo que permitiría obtenerla:

1. `Calcular` $|\boldsymbol{A}|$.
2. `Si` $|\boldsymbol{A}| = 0$ `indicar que la matriz es singular y FIN.`
3. `Calcular la inversa` $\boldsymbol{C} = \boldsymbol{A}^{-1}$.
4. `Calcular la solución` $\boldsymbol{x} = \boldsymbol{C}\boldsymbol{b}$.
5. `Escribir la solución y FIN.`

A pesar de tener todo el fundamento analítico necesario, este algoritmo para obtener la solución de 6.1 es el peor método posible desde un punto de vista numérico. De hecho, salvo en contadas excepciones, este algoritmo está condenado al más absoluto fracaso. Para darse cuenta de ello, basta observar sólo dos de los problemas que plantea.

En primer lugar, el cálculo del determinante puede ser bastante tedioso puesto que el determinante puede variar bruscamente con pequeños escalados de la matriz. Obsérvese que si $\boldsymbol{A}$ es de orden n, entonces $\det(\gamma\boldsymbol{A}) = \gamma^n \det(\boldsymbol{A})$. Para ver las implicaciones que esta igualdad impone basta tomar el caso particular de $n = 100$ (número de ecuaciones pequeño hoy en día), entonces: $\det(0.1\ \boldsymbol{A}) = 10^{-100} \det(\boldsymbol{A})$. Es decir, dividiendo los coeficientes de $\boldsymbol{A}$ por diez, se reduce el determinante de $\boldsymbol{A}$ en un factor de 10^{-100}. Por consiguiente, es muy difícil determinar numéricamente si el determinante de una matriz es realmente nulo. El uso del determinante se centra básicamente en estudios teóricos.

En segundo lugar, el cálculo de la inversa de $\boldsymbol{A}$ (que presenta serios problemas asociados al almacenamiento de la matriz y a la precisión con la que obtengan los resultados), no se emplearía ni en el caso escalar ($n = 1$). Por ejemplo, para resolver $15x = 3$ no se evaluaría primero $c = 1/15$ para después calcular x como $x = 3c$. Lo más lógico sería dividir directamente 3 por 15, $x = 3/15$, lo que permitiría ahorrarse una operación y un error de almacenamiento. Esta situación puede extrapolarse al caso de orden n donde la diferencia en número de operaciones es muy considerable y además los errores de redondeo pueden dar lugar a inestabilidades numéricas.

A continuación se estudia el *método de Cramer*. Este método es una mejora del algoritmo anterior puesto que permite realizar los pasos 3 y 4 de una sola vez. A pesar de ello no es un método adecuado desde un punto de vista numérico.

La expresión general de la solución por el método de Cramer es:

$$x_i = \frac{\begin{vmatrix} a_{11} & \dots & a_{1,i-1} & b_1 & a_{1,i+1} & \dots & a_{1n} \\ a_{21} & \dots & a_{2,i-1} & b_2 & a_{2,i+1} & \dots & a_{2n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & \dots & a_{n,i-1} & b_n & a_{n,i+1} & \dots & a_{nn} \end{vmatrix}}{|\boldsymbol{A}|} \qquad i = 1, \dots, n \tag{6.3}$$

Es interesante evaluar el número de operaciones elementales (sumas, productos y divisiones) necesarias para obtener la solución del sistema. En primer lugar hay que evaluar $n + 1$ determinantes y luego realizar n divisiones. Para el cálculo de los determinantes, una de las posibles técnicas necesita de $n!$ n multiplicaciones y $n!-1$ sumas. Por consiguiente, el método de Cramer necesita de

$$\begin{cases} (n+1)\ (n!-1) & \text{sumas} \\ (n+1)\ n!\ n & \text{productos} \\ n & \text{divisiones} \end{cases}$$

Cada operación elemental puede tener un coste computacional distinto (por ejemplo, muchos ordenadores dividen empleando el método de Newton para ceros de funciones). A pesar de ello, aquí se les asignará el mismo coste computacional a todas las operaciones elementales

puesto que ello ya permite realizar las comparaciones pertinentes. El número de operaciones elementales con el método de Cramer es $T_C = (n+1)^2\, n! - 1$. La tabla 6.1 muestra los valores de T_C para diferentes tamaños del sistema de ecuaciones.

Tabla 6.1 Operaciones elementales del método de Cramer según el tamaño de la matriz (n)

n	T_C
5	4 319
10	4×10^8
100	10^{158}

Los números presentados en la tabla 6.1 adquieren un mayor relieve cuando se asocian al tiempo necesario para efectuarlos. Si se dispusiera de un superordenador capaz de realizar 100 millones de operaciones en coma flotante por segundo (100 Mflops), el sistema de $n = 100$ necesitaría aproximadamente $3 \cdot 10^{142}$ *años* para ser resuelto. Es evidente que el número de operaciones asociado a este método hace prohibitivo su uso, aún para sistemas pequeños. Si además se tiene en cuenta que el *ENIAC* (primer ordenador digital, fabricado en 1940) realizaba sólo 300 operaciones por segundo y tenía un tiempo medio entre averías de 12 horas, se comprenderá por qué la resolución de sistemas lineales de ecuaciones está en el origen del desarrollo de los métodos numéricos.

6.1.4 Resolución numérica: un enfoque global

La estrategia y metodología que se aplica a la resolución numérica de sistemas lineales de ecuaciones parte de una filosofía distinta a la expuesta anteriormente. La regularidad de la matriz $\boldsymbol{A}$ no se determina por un cálculo previo de su determinante. Sin embargo, en algunos problemas se puede estudiar la regularidad de $\boldsymbol{A}$ en función de su origen (por ejemplo cuando proviene de la discretización de ecuaciones diferenciales) o a partir de propiedades fácilmente computables como la dominancia diagonal. Por lo general se aplica alguno de los métodos de resolución que se verán seguidamente sin evaluar previamente el determinante; en muchas ocasiones el determinante y la inversa de la matriz son un subproducto de los cálculos efectuados.

En realidad los algoritmos eficaces para la resolución de sistemas lineales plantean procesos con un enfoque radicalmente distinto, sobretodo desde una perspectiva que contempla el hecho de que los cálculos se realizan en ordenadores digitales. Por consiguiente, es lógico que los algoritmos se evalúen en función de su eficacia y siguiendo criterios directamente relacionados con su implementación en ordenadores digitales. Existen tres criterios fundamentales para analizar los algoritmos:

1. *Número de operaciones necesarias*, íntimamente ligado al tiempo de CPU. Se tendrán en cuenta las operaciones elementales entre números en coma flotante (*flop*): `+`, `-`, `/` ó `*`, todas a un mismo coste computacional aunque no sea exactamente cierto. El número de operaciones es obviamente un excelente indicador del coste computacional pero no debe tomarse

en un sentido estricto. De hecho, multiplicar el tiempo necesario para una operación por el número de operaciones siempre infravalora el tiempo necesario del algoritmo. Además del tiempo invertido en efectuar las operaciones hay una sobrecarga, debido a la gestión de la memoria, al manejo de los índices enteros, a las instrucciones lógicas en los propios bucles, etc. A pesar de ello, y por fortuna, el número de operaciones es un buen indicador del tiempo de CPU porque esta sobrecarga es generalmente proporcional al número de operaciones, de forma que, aunque no se pueda predecir exactamente el tiempo de CPU, se puede saber cómo varía (linealmente, cuadráticamente, etc.) al modificar, por ejemplo, el orden n de la matriz.

2. *Necesidades de almacenamiento*, que inciden clara y directamente en las limitaciones de la memoria de los diversos ordenadores; los diferentes métodos de resolución requieren almacenar las matrices de distinta forma en el ordenador y esto varía considerablemente las necesidades de memoria.

3. *Rango de aplicabilidad*: no todos los métodos sirven para cualquier matriz no singular; además, en función del método y de las propiedades de la matriz, la precisión de los resultados puede verse afectada dramáticamente. Como se verá más adelante, pequeños errores de redondeo pueden producir errores en la *solución numérica* completamente desproporcionados. No se debe olvidar que debido al enorme número de operaciones necesarias para la resolución de un sistema de ecuaciones de tamaño medio–grande, el análisis estándar de propagación de errores de redondeo no es en absoluto trivial.

Conviene resaltar que cada uno de estos criterios puede ser determinante para rechazar un algoritmo. Por ejemplo, para un tipo de ordenador dado, métodos que impliquen exceder la memoria disponible son inutilizables por muy rápidos y precisos que resulten. Por lo tanto, el desarrollo de los algoritmos que se plantean a continuación debe tener presentes estos tres criterios simultáneamente.

Desde un punto de vista general las matrices más usuales en las ciencias aplicadas y en ingeniería pueden englobarse en dos grandes categorías:

1. Matrices llenas pero no muy grandes. Por *llenas* se entiende que poseen pocos elementos nulos y por no muy grandes que el número de ecuaciones es de unos pocos miles a lo sumo. Estas matrices aparecen en problemas estadísticos, matemáticos, físicos e ingenieriles.

2. Matrices vacías y muy grandes. En oposición al caso anterior, *vacías* indica que hay pocos elementos no nulos y además están situados con una cierta regularidad. En la mayoría de estos casos el número de ecuaciones supera los miles y puede llegar en ocasiones a los millones. Estas matrices son comunes en la resolución de ecuaciones diferenciales de problemas de ingeniería.

Parece lógico que los métodos para resolver sistemas lineales de ecuaciones se adecuen a las categorías de matrices anteriormente expuestas. En general los *métodos directos* se aplican al primer tipo de matrices, mientras que los *métodos iterativos* se emplean con el segundo

grupo. Es importante observar que no existen reglas absolutas y que todavía en la actualidad existe cierta controversia sobre los métodos óptimos a aplicar en cada caso. En particular, la distinción establecida entre matrices llenas y vacías depende en gran medida del ordenador disponible (fundamentalmente de la memoria). De hecho, los límites han ido evolucionando a lo largo de los años a medida que también evolucionaban los ordenadores (cada vez más rápidos y con más memoria, o al menos más barata). A pesar de ello, casi nadie recomendaría métodos iterativos para matrices llenas con pocas ecuaciones; en cambio, algunos autores trabajan con métodos directos altamente sofisticados y particularizados al entorno informático disponible para resolver sistemas con varios millones de ecuaciones.

Observación: En todo lo que sigue se supone que $\boldsymbol{A}$ y $\boldsymbol{b}$ son de coeficientes reales. Si los elementos de $\boldsymbol{A}$ o $\boldsymbol{b}$ son complejos, aparte de las generalizaciones de los métodos que aquí se estudian o de los algoritmos específicos para este caso, se puede replantear el problema como un sistema lineal, con matriz y término independiente reales, de $2n$ ecuaciones e incógnitas. Para ello se escriben la matriz y los vectores de la siguiente manera:

$$\begin{aligned} \boldsymbol{A} &= \boldsymbol{C} + i\boldsymbol{D} \\ \boldsymbol{b} &= \boldsymbol{c} + i\boldsymbol{d} \\ \boldsymbol{x} &= \boldsymbol{y} + i\boldsymbol{z} \end{aligned}$$

donde $\boldsymbol{C}$ y $\boldsymbol{D}$ son matrices reales $n \times n$, y $\boldsymbol{c}$ $\boldsymbol{d}$, $\boldsymbol{y}$ y $\boldsymbol{z}$ son de $\mathbb{R}^n$. El sistema lineal de ecuaciones original se escribe ahora como:

$$\begin{pmatrix} \boldsymbol{C} & -\boldsymbol{D} \\ \boldsymbol{D} & \boldsymbol{C} \end{pmatrix} \begin{pmatrix} \boldsymbol{y} \\ \boldsymbol{z} \end{pmatrix} = \begin{pmatrix} \boldsymbol{c} \\ \boldsymbol{d} \end{pmatrix}$$

que es el resultado deseado. □

6.2 Métodos directos

6.2.1 Introducción

Los *métodos directos* de resolución de sistemas lineales de ecuaciones son aquellos que permiten obtener la solución después de un número finito de operaciones aritméticas. Este número de operaciones es función del tamaño de la matriz.

Si los ordenadores pudieran almacenar y operar con todas las cifras de los números reales, es decir, si emplearan una aritmética exacta, con los métodos directos se obtendría la solución exacta del sistema en un número finito de pasos. Puesto que los ordenadores tienen una precisión finita, los errores de redondeo se propagan y la solución numérica obtenida siempre difiere de la solución exacta. La cota del error, para una matriz y término independiente dados, se asocia por lo general al número de operaciones de cada método. Se pretende, por lo tanto, obtener métodos con el mínimo número de operaciones posible.

- **Sistemas con solución inmediata**
  - ▷ Matriz diagonal, $\boldsymbol{A} = \boldsymbol{D}$
  - ▷ Matriz triangular superior, $\boldsymbol{A} = \boldsymbol{U}$
  - ▷ Matriz triangular inferior, $\boldsymbol{A} = \boldsymbol{L}$
- **Métodos de eliminación**
  - ▷ Método de Gauss
  - ▷ Método de Gauss-Jordan
- **Métodos de descomposición**
  - ▷ Método de Doolittle, $\boldsymbol{A} = \boldsymbol{LU}$
  - ▷ Método de Crout, $\boldsymbol{A} = \boldsymbol{LU}$
  - ▷ Método de Cholesky, $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$
  - ▷ Descomposición generalizada, $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$
  - ▷ Método de Thomas ($\boldsymbol{A}$ tridiagonal)
- **Métodos de ortogonalización,** $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$

Fig. 6.1 Clasificación de los métodos directos

Otra particularidad de los métodos directos es que siempre conducen, después de ciertas operaciones, a la resolución de uno o varios sistemas con solución inmediata. Es decir, sistemas donde la matriz es diagonal o triangular. Los métodos para sistemas de resolución inmediata son de hecho métodos directos. Además de éstos, los métodos directos se dividen en *métodos*

de eliminación y *métodos de descomposición*. En la figura 6.1 se presenta un esquema con la clasificación de los métodos directos más característicos.

6.2.2 Sistemas con solución inmediata

MATRIZ DIAGONAL

En este caso la matriz $\boldsymbol{A}$ se escribe como:

$$\boldsymbol{A} = \boldsymbol{D} = \begin{pmatrix} d_{11} & 0 & \cdots & \cdots & 0 \\ 0 & d_{22} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & d_{n-1,n-1} & 0 \\ 0 & \cdots & \cdots & 0 & d_{nn} \end{pmatrix} \tag{6.4}$$

y la solución se obtiene directamente

$$x_i = \frac{b_i}{d_{ii}} \qquad i = 1, \ldots, n \tag{6.5}$$

Existe solución si todos los términos de diagonal son no nulos. Además, si se desea evaluar el determinante de $\boldsymbol{A}$ sólo es necesario calcular el producto de todos los términos de la diagonal. Por último, el número de operaciones necesario es de n divisiones, es decir $T_D = n$ operaciones elementales.

MATRIZ TRIANGULAR SUPERIOR

$$\boldsymbol{A} = \boldsymbol{U} = \begin{pmatrix} u_{11} & u_{12} & \cdots & \cdots & u_{1n} \\ 0 & u_{22} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & u_{n-1,n-1} & u_{n-1,n} \\ 0 & \cdots & \cdots & 0 & u_{nn} \end{pmatrix} \tag{6.6}$$

En este caso la solución de la última ecuación es trivial $x_n = b_n \;/\; u_{nn}$. Una vez conocido x_n, la penúltima ecuación (la $n-1$) sólo tiene una incógnita que se deduce de forma sencilla. Conocidos ahora x_n y x_{n-1}, se pasa a la ecuación anterior (la $n-2$) y se resuelve para su única incógnita, x_{n-2}. Retrocediendo progresivamente se obtiene el algoritmo de *sustitución hacia atrás* que se escribe de la siguiente forma

$$\begin{aligned} x_n &= b_n \;/\; u_{nn} \\ x_i &= \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) \;/\; u_{ii} \qquad i = n-1, n-2, \ldots, 1 \end{aligned} \tag{6.7}$$

De nuevo la solución existe si todos los términos de la diagonal de $\boldsymbol{U}$ son no nulos. El determinante se evalúa multiplicando los términos de la diagonal. El número de operaciones es ahora:

$$\begin{cases} 1+2+\cdots+(n-1)=\dfrac{n(n-1)}{2} & \text{sumas} \\ 1+2+\cdots+(n-1)=\dfrac{n(n-1)}{2} & \text{productos} \\ n & \text{divisiones} \end{cases}$$

por consiguiente $T_{\triangle}=n^2$ operaciones elementales.

MATRIZ TRIANGULAR INFERIOR

$$\boldsymbol{A}=\boldsymbol{L}=\begin{pmatrix} l_{11} & 0 & \cdots & \cdots & 0 \\ l_{21} & l_{22} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & l_{n-1,n-1} & 0 \\ l_{n1} & \cdots & \cdots & l_{n,n-1} & l_{nn} \end{pmatrix} \tag{6.8}$$

Se aplica un algoritmo similar al anterior que se denomina de *sustitución hacia adelante*:

$$\begin{aligned} x_1 &= b_1 \,/\, l_{11} \\ x_i &= \left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j \right) \,/\, l_{ii} \qquad i=2,\ldots,n \end{aligned} \tag{6.9}$$

La existencia de solución, el determinante y el número de operaciones se evalúan exactamente como en el caso anterior y se llega a los mismos resultados.

6.2.3 Métodos de eliminación

MÉTODO DE GAUSS

En el método de eliminación de Gauss el problema original, $\boldsymbol{Ax}=\boldsymbol{b}$, se transforma mediante permutaciones adecuadas y combinaciones lineales de las ecuaciones en un sistema de la forma $\boldsymbol{Ux}=\boldsymbol{c}$ donde $\boldsymbol{U}$ es una matriz triangular superior. Este nuevo sistema equivalente al original es de resolución inmediata: sólo es necesario aplicar el algoritmo de sustitución hacia atrás presentado en el subapartado anterior.

Durante la transformación del sistema original al equivalente con matriz triangular, las operaciones (que *sólo* dependen de la matriz $\boldsymbol{A}$) se realizan sobre la matriz y *al mismo tiempo* sobre el término independiente. Esto constituye una de las grandes ventajas y a la vez inconvenientes de los métodos de eliminación. Si se dispone de una serie de términos independientes,

$\boldsymbol{b}_j$ $j = 1, \ldots, m$, conocidos de antemano, se efectúan sobre *todos* ellos, y al mismo tiempo, las operaciones necesarias para reducir el sistema y obtener una serie de $\boldsymbol{c}_j$ $j = 1, \ldots, m$. Por consiguiente, se almacenan y se manipulan todos los términos independientes a la vez. Posteriormente se resuelve un sistema con matriz triangular $\boldsymbol{U}$ para cada uno de los $\boldsymbol{c}_j$. Si, por el contrario, no se conocen todos los términos independientes al iniciar los cálculos, es necesario *recordar* todas las transformaciones necesarias para obtener $\boldsymbol{c}_1$ partiendo de $\boldsymbol{b}_1$; seguidamente se *repiten* todas estas operaciones sobre los demás términos independientes hasta obtener todos los $\boldsymbol{c}_j$ deseados. Hoy en día, en la mayoría de los problemas con matrices de tamaño pequeño o medio, esta propiedad de los métodos de eliminación es la determinante para su elección frente a los métodos de descomposición.

Otro punto importante que conviene valorar en el método de Gauss es su importante valor pedagógico. Muchos autores denominan de manera genérica *métodos gaussianos* al resto de los métodos de eliminación y de descomposición, puesto que la mayoría derivan del trabajo original de Gauss escrito en 1823 (que como otros métodos fundamentales del cálculo numérico fue desarrollado mucho antes de la aparición del primer ordenador). Su implementación en un ordenador sigue siendo la más simple, y con pocas modificaciones, como ya se verá, se obtiene el método más general que existe para la resolución de sistemas lineales de ecuaciones.

El algoritmo que se presenta a continuación parte de la ecuación 6.1 que se escribirá como:

$$\begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \cdots & a_{1n}^{(0)} \\ a_{21}^{(0)} & a_{22}^{(0)} & a_{23}^{(0)} & \cdots & a_{2n}^{(0)} \\ a_{31}^{(0)} & a_{32}^{(0)} & a_{33}^{(0)} & \cdots & a_{3n}^{(0)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(0)} & a_{n2}^{(0)} & a_{n3}^{(0)} & \cdots & a_{nn}^{(0)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ b_3^{(0)} \\ \vdots \\ b_n^{(0)} \end{pmatrix} \tag{6.10}$$

donde el superíndice $^{(0)}$ indica coeficiente de la matriz o del término independiente originales. Si $a_{11}^{(0)} \neq 0$, se sustrae de todas la ecuaciones, a partir de la segunda fila, la primera ecuación multiplicada por $m_{i1} = \frac{a_{i1}^{(0)}}{a_{11}^{(0)}}$ con $i = 2, \ldots, n$. Esto induce el primer sistema equivalente derivado del original donde la primera columna tiene todos los coeficientes nulos exceptuando el primer coeficiente, y el resto de los coeficientes de la matriz y del término independiente se han visto modificados a partir de la segunda fila.

$$\begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \cdots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(0)} \\ b_2^{(1)} \\ b_3^{(1)} \\ \vdots \\ b_n^{(1)} \end{pmatrix} \tag{6.11}$$

con

$$\begin{aligned} a_{ij}^{(1)} &= a_{ij}^{(0)} - m_{i1} a_{1j}^{(0)} = a_{ij}^{(0)} - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}} a_{1j}^{(0)} \\ b_i^{(1)} &= b_i^{(0)} - m_{i1} b_1^{(0)} = b_i^{(0)} - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}} b_1^{(0)} \end{aligned} \qquad \begin{cases} i = 2, \ldots, n \\ j = 2, \ldots, n \end{cases} \tag{6.12}$$

Ahora, si $a_{22}^{(1)}$ (que ya no coincide con el coeficiente que originalmente se encontraba en su posición, $a_{22}^{(0)}$) es distinto de cero, se sustrae de todas la ecuaciones siguientes la segunda ecuación multiplicada por $m_{i2} = \frac{a_{i2}^{(1)}}{a_{22}^{(1)}}$ con $i = 3, \ldots, n$. Después de realizadas estas operaciones sobre el sistema 6.11 se obtiene el segundo sistema equivalente al original

$$\begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \cdots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(0)} \\ b_2^{(1)} \\ b_3^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix} \tag{6.13}$$

donde la segunda columna a partir de la tercera fila sólo contiene términos nulos y los nuevos coeficientes de la matriz y término independiente se obtienen con las siguientes ecuaciones

$$\begin{aligned} a_{ij}^{(2)} &= a_{ij}^{(1)} - m_{i2} a_{2j}^{(1)} = a_{ij}^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} a_{2j}^{(1)} \\ b_i^{(2)} &= b_i^{(1)} - m_{i2} b_2^{(1)} = b_i^{(1)} - \frac{a_{i2}^{(1)}}{a_{22}^{(1)}} b_2^{(1)} \end{aligned} \qquad \begin{cases} i = 3, \ldots, n \\ j = 3, \ldots, n \end{cases} \tag{6.14}$$

Cada paso conduce a un nuevo sistema equivalente al original (ecuación 6.1) con la particularidad de que la k-ésima matriz es triangular superior si sólo se miran las primeras k ecuaciones y k incógnitas. En general, se escribe como

$$\begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1k}^{(0)} & a_{1,k+1}^{(0)} & \cdots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2k}^{(1)} & a_{2,k+1}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \ddots & \ddots & \vdots & \vdots & \cdots & \vdots \\ \vdots & & \ddots & a_{kk}^{(k-1)} & a_{k,k+1}^{(k-1)} & \cdots & a_{kn}^{(k-1)} \\ 0 & \cdots & \cdots & 0 & a_{k+1,k+1}^{(k)} & \cdots & a_{k+1,n}^{(k)} \\ \vdots & & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & a_{n,k+1}^{(k)} & \cdots & a_{nn}^{(k)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ x_{k+1} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(0)} \\ b_2^{(1)} \\ \vdots \\ b_k^{(k-1)} \\ b_{k+1}^{(k)} \\ \vdots \\ b_n^{(k)} \end{pmatrix} \tag{6.15}$$

que se ha obtenido a partir de las siguientes ecuaciones

$$\begin{aligned} a_{ij}^{(k)} &= a_{ij}^{(k-1)} - m_{ik} a_{kj}^{(k-1)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} \\ b_i^{(k)} &= b_i^{(k-1)} - m_{ik} b_k^{(k-1)} = b_i^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} b_k^{(k-1)} \end{aligned} \qquad \begin{cases} i = k+1, \ldots, n \\ j = k+1, \ldots, n \end{cases} \tag{6.16}$$

Obsérvese que al pasar del $(k-1)$-ésimo al k-ésimo sistema es necesario realizar las siguientes operaciones

$$\begin{cases} (n-k)(n-k+1) & \text{sumas} \\ (n-k)(n-k+1) & \text{productos} \\ n-k & \text{divisiones} \end{cases}$$

Finalmente, al deducir el $(n-1)$-ésimo sistema se obtiene una matriz triangular superior:

$$\begin{pmatrix} a_{11}^{(0)} & a_{12}^{(0)} & a_{13}^{(0)} & \cdots & a_{1n}^{(0)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & a_{n-1,n}^{(n-2)} \\ 0 & 0 & \cdots & \cdots & a_{nn}^{(n-1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(0)} \\ b_2^{(1)} \\ \vdots \\ b_{n-1}^{(n-2)} \\ b_n^{(n-1)} \end{pmatrix} \tag{6.17}$$

A cada uno de los términos que aparecen en la diagonal de la matriz anterior se le denomina *pivote*. Conviene resaltar que los pivotes no coinciden con los términos originales de la diagonal de $\boldsymbol{A}$; es decir, $a_{kk}^{(k-1)} \neq a_{kk}^{(0)}$ para $k = 1, \ldots, n$.

Para resumir todos los pasos realizados hasta obtener el sistema 6.17, es necesario suponer que todos los pivotes son no nulos. Es decir, $a_{ii}^{(i-1)} \neq 0 \quad i = 1, \ldots, n$. A continuación se presenta el algoritmo que permite obtener la matriz y el término independiente del sistema 6.17,

$$\begin{aligned} a_{ij}^{(k)} &= a_{ij}^{(k-1)} - m_{ik} a_{kj}^{(k-1)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} \\ b_i^{(k)} &= b_i^{(k-1)} - m_{ik} b_k^{(k-1)} = b_i^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} b_k^{(k-1)} \end{aligned} \qquad \begin{cases} k = 1, \ldots, n-1 \\ i = k+1, \ldots, n \\ j = k+1, \ldots, n \end{cases} \tag{6.18}$$

donde los términos de superíndice (0) son iguales a los originales del sistema de ecuaciones. El sistema triangular obtenido en 6.17 es de resolución inmediata (véase el subapartado 6.2.2). El número de operaciones necesarias para realizar esta primera fase de eliminación ha sido de

$$\begin{cases} \displaystyle\sum_{k=1}^{n-1} (n-k)(n-k+1) = \frac{n(n^2-1)}{3} & \text{sumas} \\ \displaystyle\sum_{k=1}^{n-1} (n-k)(n-k+1) = \frac{n(n^2-1)}{3} & \text{productos} \\ \displaystyle\sum_{k=1}^{n-1} n-k = \frac{n(n-1)}{2} & \text{divisiones} \end{cases}$$

Si se tienen en cuenta las operaciones correspondientes a la segunda fase de sustitución hacia atrás, el número total de operaciones elementales necesarias para el método de Gauss es $T_G = \frac{4n^3+9n^2-7n}{6}$. La tabla 6.2 muestra el número de operaciones elementales para distintos tamaños

del sistema de ecuaciones. Obviamente, se ha obtenido una importante reducción al disponer ahora de un método que crece con n^3, en vez de $n!\ n^2$ (Cramer).

Tabla 6.2 Operaciones elementales del método de Gauss sin pivotamiento según el tamaño de la matriz (n)

n	T_G
5	115
10	805
100	681 550
1000	6.68×10^8

Como ya se ha comentado, se ha supuesto a lo largo de toda esta deducción que los pivotes eran distintos de cero. Si durante el proceso de eliminación se obtiene un pivote nulo, por ejemplo el $a_{kk}^{(k-1)}$, se debe buscar en la parte inferior de la columna k-ésima un coeficiente no nulo, es decir de entre los $a_{ik}^{(k-1)} \quad i = k+1, \ldots, n$ se toma uno que sea distinto de cero. Se sustituye entonces la fila k (y su término independiente) por la fila i (y su término independiente) que se haya escogido. Si dicho coeficiente no nulo no existiera, *la matriz sería singular*. Más adelante se verá una justificación teórica de este proceder.

Esta permutación de filas no sólo tiene interés cuando el pivote es exactamente cero. Es obvio que valores pequeños del pivote pueden producir grandes errores de redondeo, ya que siempre se divide por el valor del pivote. Por consiguiente, para reducir los errores de redondeo conviene escoger el pivote máximo en valor absoluto. Para ello, hay dos técnicas posibles:

1. En el k-ésimo sistema (véanse las ecuaciones 6.15 y 6.16) se toma como pivote el coeficiente mayor en valor absoluto de la columna k situado por debajo de la fila k inclusive. Para ello es necesario permutar las filas k y la correspondiente al pivote escogido en la matriz y su término independiente. Esta técnica se denomina *método de Gauss con pivotamiento parcial*.

2. En el k-ésimo sistema, se toma como pivote el coeficiente mayor en valor absoluto de la submatriz de orden $n-k$ definida por los coeficientes que quedan por debajo de la fila k y a la derecha de la columna k. Para ello, se permuta la fila k (y el término independiente asociado) y la columna k con las correspondientes al coeficiente que cumple la condición citada. Al final del proceso deben ponerse en el orden inicial las componentes del vector solución, puesto que su posición ha sido modificada al realizar las permutaciones de columnas. Esta técnica es el *método de Gauss con pivotamiento total*.

Estas dos últimas estrategias producen métodos numéricamente estables. El método de Gauss sin pivotamiento no es necesariamente estable. El estudio detallado de la estabilidad y propagación de errores de redondeo del método de Gauss no es trivial (véase Wilkinson (1965)).

Desde el punto de vista de la implementación práctica de los métodos de Gauss con pivotamiento, conviene señalar que las permutaciones no se realizan físicamente en el ordenador, sino que se emplean vectores de redireccionamiento de memoria similares a los empleados en los esquemas de almacenamiento específicos para matrices con estructuras simples.

Observación: Si la matriz $\boldsymbol{A}$ es simétrica, las matrices llenas de orden $n-k$ sobre las que se aplica sucesivamente el algoritmo sólo permanecen simétricas si no se realiza ninguna permutación de filas o columnas (véase el problema 6.1). La misma observación es válida si la matriz $\boldsymbol{A}$ tiene una estructura que permite el almacenamiento en banda o en perfil (apartado 7.4). □

Problema 6.1:
Sea $\boldsymbol{A}$ una matriz regular *simétrica*. Se desea resolver el sistema lineal $\boldsymbol{Ax} = \boldsymbol{b}$ mediante el método de Gauss *sin pivotamiento*.

a) Adaptar el algoritmo de Gauss al caso de matrices simétricas, aprovechando la simetría para eliminar las operaciones innecesarias. Sugerencia: nótese que en cada paso del proceso de eliminación, cuando se anulan los términos de la columna k-ésima por debajo del pivote, $a_{kk}^{(k-1)}$, sólo se modifica la submatriz llena de orden $n-k$:

$$\begin{pmatrix} a_{k+1,k+1}^{(k)} & \cdots & a_{k+1,n}^{(k)} \\ \vdots & \ddots & \vdots \\ a_{n,k+1}^{(k)} & \cdots & a_{nn}^{(k)} \end{pmatrix}$$

Emplear la propiedad de que en $k=0$ la submatriz correspondiente (que es la matriz original $\boldsymbol{A}$) es simétrica.

b) Calcular el número de operaciones necesarias, y compararlo con el caso general (matrices no simétricas).

c) ¿Puede emplearse el algoritmo desarrollado en el apartado *a* si es necesario pivotar? ¿Por qué? ●

Método de Gauss-Jordan

A continuación se presenta una variante del método de Gauss que conviene considerar. En este método, además de sustraer la fila k multiplicada por $m_{ik} = a_{ik}^{(k-1)}/a_{kk}^{(k-1)}$ a las filas posteriores, también se sustrae a las anteriores. Es práctica común, en este caso, dividir la fila k por su pivote para que el término de la diagonal quede unitario. De esta forma, el k-ésimo sistema así obtenido se escribe como:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & a_{1,k+1}^{(k)} & \cdots & a_{1n}^{(k)} \\ 0 & 1 & 0 & \cdots & 0 & a_{2,k+1}^{(k)} & \cdots & a_{2n}^{(k)} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & & \vdots \\ \vdots & & \ddots & \ddots & 0 & a_{k-1,k+1}^{(k)} & \cdots & a_{k-1,n}^{(k)} \\ \vdots & & & \ddots & 1 & a_{k,k+1}^{(k)} & \cdots & a_{kn}^{(k)} \\ 0 & \cdots & \cdots & \cdots & 0 & a_{k+1,k+1}^{(k)} & \cdots & a_{k+1,n}^{(k)} \\ \vdots & & & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & 0 & a_{n,k+1}^{(k)} & \cdots & a_{nn}^{(k)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{k-1} \\ x_k \\ x_{k+1} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_{k-1}^{(k)} \\ b_k^{(k)} \\ b_{k+1}^{(k)} \\ \vdots \\ b_n^{(k)} \end{pmatrix} \tag{6.19}$$

Como se puede observar, al anular todos los coeficientes de la columa k, excepto el diagonal, se va transformando la matriz original en la identidad. Al final, la $(n-1)$-ésima matriz obtenida por operaciones simples de fila es la identidad, y por lo tanto, el $(n-1)$-ésimo término independiente, $(b_1^{(n-1)}, \ldots, b_n^{(n-1)})^T$ es la solución del sistema de ecuaciones original.

El algoritmo necesario para la transformación de la matriz es el siguiente

$$\begin{aligned} a_{kj}^{(k)} &= \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}} \\ a_{ij}^{(k)} &= a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k-1)} \\ b_k^{(k)} &= \frac{b_k^{(k-1)}}{a_{kk}^{(k-1)}} \\ b_i^{(k)} &= b_i^{(k-1)} - a_{ik}^{(k-1)} b_k^{(k-1)} \end{aligned} \qquad \begin{cases} k = 1, \ldots, n-1 \\ i = 1, \ldots, k-1, k+1, \ldots, n \\ j = k+1, \ldots, n \end{cases} \tag{6.20}$$

El número de operaciones que se deben realizar es

$$\begin{cases} \displaystyle\sum_{k=1}^{n-1} (n-1)(n-k+1) = \frac{(n-1)^2(n-2)}{2} & \text{sumas} \\ \displaystyle\sum_{k=1}^{n-1} (n-1)(n-k+1) = \frac{(n-1)^2(n-2)}{2} & \text{productos} \\ \displaystyle\sum_{k=1}^{n-1} (n-k+1) = \frac{(n-1)(n-2)}{2} & \text{divisiones} \end{cases}$$

Por consiguiente el número total de operaciones elementales del método de Gauss-Jordan, tal como se ha presentado aquí, es de $T_{GJ} = n^3 + \frac{1}{2}n^2 - \frac{5}{2}n + 1$

Análisis matricial del Método de Gauss: Gauss compacto

La presentación del método de Gauss se ha realizado de forma constructiva mostrando el desarrollo del algoritmo. A continuación se presenta el desarrollo matricial del método. De esta forma se sientan las bases de los métodos de descomposición y, además, se analizan las justificaciones teóricas del método de Gauss sin o con pivotamiento. Se puede suponer que de momento, la eliminación (reducción de la matriz original a una triangular superior) puede realizarse sin pivotamiento. Entonces se toman las matrices:

$$\boldsymbol{G}^{(k)} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & \ddots & & \vdots & \vdots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & & & \vdots \\ \vdots & & \ddots & 1 & 0 & \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 & 1 & 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots & -m_{k+1,k} & 1 & \ddots & & \vdots \\ \vdots & & & \vdots & -m_{k+2,k} & 0 & \ddots & \ddots & \vdots \\ \vdots & & & \vdots & \vdots & \vdots & \ddots & 1 & 0 \\ 0 & \cdots & \cdots & 0 & -m_{n,k} & 0 & \cdots & 0 & 1 \end{pmatrix} \qquad k = 1, \ldots, n-1 \qquad (6.21)$$

donde $m_{ik} = \dfrac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}$ para $i = k+1, \ldots, n$.

Denominando $\boldsymbol{A}^{(k)}$ y $\boldsymbol{b}^{(k)}$ a los k-ésimos matriz y vector de términos independientes obtenidos por el método de Gauss sin pivotamiento, se puede observar que las ecuaciones que realizan el paso entre $(k-1)$ y k (ecuaciones 6.16) pueden escribirse como

$$\begin{aligned} \boldsymbol{A}^{(k)} &= \boldsymbol{G}^{(k)}\ \boldsymbol{A}^{(k-1)} \\ \boldsymbol{b}^{(k)} &= \boldsymbol{G}^{(k)}\ \boldsymbol{b}^{(k-1)} \end{aligned}$$

Por lo tanto, el $(n-1)$-ésimo sistema de ecuaciones obtenido al final del proceso de eliminación es simplemente

$$\begin{aligned} \boldsymbol{A}^{(n-1)} &= \boldsymbol{G}^{(n-1)}\boldsymbol{G}^{(n-2)} \ldots \boldsymbol{G}^{(1)}\ \boldsymbol{A} \\ \boldsymbol{b}^{(n-1)} &= \boldsymbol{G}^{(n-1)}\boldsymbol{G}^{(n-2)} \ldots \boldsymbol{G}^{(1)}\ \boldsymbol{b} \end{aligned} \qquad (6.22)$$

Puesto que $\boldsymbol{A}^{(n-1)}$ es triangular superior, y que el producto de matrices triangulares inferiores con diagonal unitaria es una matriz triangular inferior con diagonal unitaria, la primera ecuación de 6.22 puede escribirse como

$$\begin{aligned} & \boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{U}\,, \\ \text{donde} \quad & \boldsymbol{U} = \boldsymbol{A}^{(n-1)} \\ & \boldsymbol{L} = \left[\boldsymbol{G}^{(n-1)}\boldsymbol{G}^{(n-2)} \ldots\ \boldsymbol{G}^{(1)}\right]^{-1} \\ & \quad = \left[\boldsymbol{G}^{(1)}\right]^{-1}\left[\boldsymbol{G}^{(2)}\right]^{-1} \ldots\ \left[\boldsymbol{G}^{(n-1)}\right]^{-1} \end{aligned} \qquad (6.23)$$

Puesto que las inversas de las $\boldsymbol{G}^{(k)}$ son triangulares inferiores con diagonal unitaria, puede comprobarse que

$$\boldsymbol{L} = \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ m_{21} & 1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ m_{n1} & \cdots & \cdots & m_{n,n-1} & 1 \end{pmatrix} \tag{6.24}$$

Por consiguiente, la ecuación 6.23 demuestra que se ha descompuesto la matriz original $\boldsymbol{A}$ en el producto de una matriz triangular inferior con diagonal unitaria por una triangular superior. A partir de esta misma ecuación se puede observar que el determinante de la matriz $\boldsymbol{A}$ se obtiene como un subproducto del método de eliminación: puesto que $\boldsymbol{L}$ es triangular con diagonal unitaria y $\boldsymbol{U} = \boldsymbol{A}^{(n-1)}$ es una triangular, el determinante de $\boldsymbol{A}$ es

$$\det(\boldsymbol{A}) = \det(\boldsymbol{A}^{(n-1)}) = \prod_{i=1}^{n} a_{ii}^{(i-1)} \tag{6.25}$$

De esta forma el cálculo del determinante de una matriz necesita de $\frac{2n^3}{3}$ operaciones aproximadamente que es muy inferior al orden $n!$ que se había obtenido previamente.

El método de Gauss planteado en su forma compacta, como se acaba de describir, se compone de dos fases diferenciadas: 1) descomposición de la matriz original en el producto de $\boldsymbol{L}$ por $\boldsymbol{U}$ sin que sea necesario efectuar operaciones sobre el término independiente, y 2) resolución de dos sistemas triangulares, el primero de los cuales se corresponde con la segunda ecuación de 6.22. Estas dos fases aparecen también en los métodos de descomposición (véase el subapartado 6.2.4), que se distinguen del método de Gauss compacto en la forma de calcular las matrices triangulares $\boldsymbol{L}$ y $\boldsymbol{U}$.

Como ya se había indicado, conviene aprovechar la forma compacta del método de Gauss para justificar teóricamente los algoritmos que se han presentado. Para ello se consideran los siguientes teoremas:

Teorema: Si $\boldsymbol{A}$ es inversible y factorizable bajo la forma $\boldsymbol{A} = \boldsymbol{LU}$ (donde $\boldsymbol{L}$ tiene la diagonal unitaria), entonces la descomposición es única.

Demostración: Supóngase que existen dos descomposiciones posibles de la matriz $\boldsymbol{A}$, es decir, $\boldsymbol{A} = \boldsymbol{L}_1\boldsymbol{U}_1 = \boldsymbol{L}_2\boldsymbol{U}_2$. Si $\boldsymbol{A}$ es inversible, entonces $\boldsymbol{L}_1$, $\boldsymbol{U}_1$, $\boldsymbol{L}_2$, y $\boldsymbol{U}_2$ también lo son (tómense por ejemplo determinantes en la ecuación anterior). Por consiguiente, $\boldsymbol{L}_1\boldsymbol{U}_1 = \boldsymbol{L}_2\boldsymbol{U}_2 \iff \boldsymbol{L}_2^{-1}\boldsymbol{L}_1 = \boldsymbol{U}_2\boldsymbol{U}_1^{-1}$ pero el primer miembro de la igualdad es una triangular inferior con diagonal unitaria mientras que el segundo es una triangular superior. Ambos miembros, por lo tanto, son iguales a la matriz identidad, de donde se deduce que $\boldsymbol{L}_1 = \boldsymbol{L}_2$ y $\boldsymbol{U}_1 = \boldsymbol{U}_2$. ■

Teorema: Si $\boldsymbol{A}$ es inversible, existe una matriz de permutación $\boldsymbol{P}$ tal que $\boldsymbol{P}\,\boldsymbol{A}$ es factorizable bajo la forma $\boldsymbol{LU}$ donde $\boldsymbol{L}$ es triangular inferior con diagonal unitaria y $\boldsymbol{U}$ es triangular superior.

Demostración: Sea $\boldsymbol{P}_{ij}$ una matriz que permute las filas i, j. Esta matriz se escribe como

$$\boldsymbol{P}_{ij} = \begin{array}{c} \\ \\ \\ \\ i \\ \\ \\ \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccccccccccc} & & & & i & & & & & j & & & \end{array} \\ \left(\begin{array}{cccccccccccc} 1 & 0 & \cdots & & 0 & & \cdots & & & 0 & & \cdots & 0 \\ 0 & \ddots & & & \vdots & & & & & \vdots & & & \vdots \\ \vdots & & & 1 & 0 & & & & & 0 & & & \\ 0 & \cdots & & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & & 0 \\ & & & & 0 & 1 & & & 0 & & & & \\ \vdots & & & & \vdots & & \ddots & & \vdots & & & & \vdots \\ & & & & 0 & & & 1 & 0 & & & & \\ 0 & \cdots & & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & & 0 \\ & & & & 0 & & & & 0 & 1 & & & \vdots \\ \vdots & & & & \vdots & & & & \vdots & & \ddots & & 0 \\ 0 & \cdots & & & 0 & & \cdots & & 0 & \cdots & & 0 & 1 \end{array}\right) \end{array} \tag{6.26}$$

Esta matriz sólo difiere de la identidad en las filas y columnas i y j. La k-ésima matriz, $\boldsymbol{A}^{(k)}$, obtenida con el algoritmo de Gauss con pivotamiento parcial, puede escribirse como $\boldsymbol{A}^{(k)} = \boldsymbol{G}^{(k)}\boldsymbol{P}_{k,\ell_k}\boldsymbol{A}^{(k-1)}$, donde se ha permutado la fila k con la ℓ_k ($k \leq \ell_k$). Obsérvese que si no es necesario pivotar $\boldsymbol{P}_{k,\ell_k} = \boldsymbol{I}$, es decir $k = \ell_k$. Esta k-ésima matriz también puede escribirse como

$$\boldsymbol{A}^{(k)} = \boldsymbol{G}^{(k)}\boldsymbol{P}_{k,\ell_k}\boldsymbol{G}^{(k-1)}\boldsymbol{P}_{k-1,\ell_{k-1}} \cdots \boldsymbol{G}^{(1)}\boldsymbol{P}_{1,\ell_1}\,\boldsymbol{A} \tag{6.27}$$

Para poder seguir con el proceso de eliminación, el pivote de esta matriz y todos los términos debajo de él deben ser no nulos. Si no es así, el determinante de $\boldsymbol{A}^{(k)}$ es nulo y por lo tanto también es nulo el determinante de la matriz original $\boldsymbol{A}$ (las matrices $\boldsymbol{G}$ y $\boldsymbol{P}$ tienen diagonales unitarias). Es decir, si $\boldsymbol{A}$ es regular, entonces $\boldsymbol{A}^{(k)}$ también lo es y siempre se puede encontrar la permutación de filas adecuada para seguir con el proceso de eliminación.

Queda todavía por demostrar que $\boldsymbol{PA}$ es factorizable. A partir de la ecuación 6.27 puede obtenerse la expresión de la última matriz obtenida con el algoritmo de Gauss con pivotamiento parcial

$$\boldsymbol{A}^{(n-1)} = \boldsymbol{G}^{(n-1)}\boldsymbol{P}_{n-1,\ell_{n-1}}\boldsymbol{G}^{(n-2)}\boldsymbol{P}_{n-2,\ell_{n-2}} \cdots \boldsymbol{G}^{(1)}\boldsymbol{P}_{1,\ell_1}\,\boldsymbol{A} \tag{6.28}$$

Si $k < i \leq j$, entonces existe una matriz $\widetilde{\boldsymbol{G}}^{(k)}$ tal que $\boldsymbol{P}_{ij}\boldsymbol{G}^{(k)} = \widetilde{\boldsymbol{G}}^{(k)}\boldsymbol{P}_{ij}$ en la que simplemente se han permutado los coeficientes $-m_{ik}$ y $-m_{jk}$. Por lo tanto, la ecuación 6.28 puede reescribirse como

$$\boldsymbol{A}^{(n-1)} = \boldsymbol{G}^{(n-1)}\widetilde{\boldsymbol{G}}^{(n-2)} \cdots \widetilde{\boldsymbol{G}}^{(1)}\boldsymbol{P}_{n-1,\ell_{n-1}}\boldsymbol{P}_{n-2,\ell_{n-2}} \cdots \boldsymbol{P}_{1,\ell_1}\,\boldsymbol{A} \tag{6.29}$$

que permite, con las definiciones adecuadas,

$$\begin{aligned} \boldsymbol{P} &= \boldsymbol{P}_{n-1,\ell_{n-1}} \boldsymbol{P}_{n-2,\ell_{n-2}} \cdots \boldsymbol{P}_{1,\ell_1} \\ \boldsymbol{U} &= \boldsymbol{A}^{(n-1)} \\ \boldsymbol{L} &= \left[\boldsymbol{G}^{(n-1)} \widetilde{\boldsymbol{G}}^{(n-2)} \cdots \widetilde{\boldsymbol{G}}^{(1)} \right]^{-1} \end{aligned} \tag{6.30}$$

volver a escribir la ecuación 6.28 como $\boldsymbol{PA} = \boldsymbol{LU}$ donde $\boldsymbol{L}$ es triangular inferior con diagonal unitaria y $\boldsymbol{U}$ es triangular superior y de esta forma finalizar la demostración. ■

Esta demostración puede extenderse al caso de Gauss con pivotamiento total empleando matrices de permutación $\boldsymbol{P}$ y $\boldsymbol{Q}$ tales que $\boldsymbol{P}\,\boldsymbol{A}\,\boldsymbol{Q}$ sea factorizable como $\boldsymbol{LU}$. Así mismo, todas las propiedades anteriores de existencia y unicidad de la descomposición se verifican también para $\boldsymbol{U}$ triangular superior con diagonal unitaria en vez de $\boldsymbol{L}$ con diagonal unitaria.

Por otro lado, el determinante de la matriz original se obtiene, de nuevo, como un subproducto del proceso de eliminación por el método de Gauss,

$$\det(\boldsymbol{A}) = \pm \det(\boldsymbol{A}^{(n-1)}) = \pm \prod_{i=1}^{n} a_{ii}^{(i-1)}$$

donde el signo depende del número de permutaciones realizadas.

Para finalizar este apartado, donde se han formalizado las diferentes variantes del método de Gauss, conviene conocer bajo qué condiciones puede aplicarse el método de Gauss sin pivotamiento. Esto es así porque sólo en esta variante (sin pivotamiento) pueden emplearse esquemas de almacenamiento específicos para algunos tipos de matrices muy frecuentes en cálculo numérico (por ejemplo, matrices en banda y en *skyline*, simétricas o no). Como se verá en el capítulo siguiente, se consigue un ahorro computacional considerable mediante estos esquemas de almacenamiento matricial, tanto en número de operaciones como en memoria necesaria. Para ello, el siguiente teorema presenta la condición necesaria y suficiente para aplicar el método de Gauss sin pivotamiento. Se denomina $\boldsymbol{A}_{[m]11}$ al menor principal de $\boldsymbol{A}$ de orden m. Es decir, los coeficientes de $\boldsymbol{A}_{[m]11}$ son los a_{ij} de $\boldsymbol{A}$ para $i = 1, \ldots, m$ y $i = 1, \ldots, m$. Por extensión $\boldsymbol{A}^{(k)}{}_{[m]11}$ será el menor de orden m de $\boldsymbol{A}^{(k)}$. Las tres cajas que completan la matriz $\boldsymbol{A}$ se denotan por $\boldsymbol{A}_{[m]12}$, $\boldsymbol{A}_{[m]21}$ y $\boldsymbol{A}_{[m]22}$, de forma que

$$\boldsymbol{A} = \begin{array}{l} \\ m \text{ filas} \\ (n-m) \text{ filas} \end{array} \begin{array}{c} \begin{array}{cc} m \text{ columnas} & (n-m) \text{ columnas} \end{array} \\ \begin{pmatrix} \boldsymbol{A}_{[m]11} & \boldsymbol{A}_{[m]12} \\ \boldsymbol{A}_{[m]21} & \boldsymbol{A}_{[m]22} \end{pmatrix} \end{array}$$

Teorema: La condición necesaria y suficiente para que una matriz no singular $\boldsymbol{A}$ pueda descomponerse en la forma $\boldsymbol{A} = \boldsymbol{LU}$ es que $\det(\boldsymbol{A}_{[m]11}) \neq 0$ para cualquier $m = 1, \ldots, n$.

Demostración: Para demostrar que es condición suficiente hay que mostrar que una vez obtenida la matriz $\boldsymbol{A}^{(k)}$, el proceso de eliminación puede seguir; es decir,

que el pivote $a^{(k)}_{k+1,k+1}$ es no nulo. Para ello supóngase que $\det(\boldsymbol{A}_{[m]11}) \neq 0 \; \forall m = 1, \ldots, n$. En particular esto se verifica para $m = 1$; es decir, $a^{(0)}_{1,1} = \det(\boldsymbol{A}_{[1]11}) \neq 0$ y, por lo tanto, la primera etapa del algoritmo de Gauss puede aplicarse. Para las siguientes etapas, supóngase que $a^{(i)}_{i+1,i+1} \neq 0$ para $i = 0, \ldots, k-1$; en consecuencia, ya se ha obtenido la matriz $\boldsymbol{A}^{(k)}$ descrita en 6.15 y que puede escribirse según la expresión 6.27, pero sin permutaciones, bajo la forma

$$\begin{aligned} \boldsymbol{A}^{(k)} &= \begin{pmatrix} \boldsymbol{A}^{(k)}{}_{[k+1]11} & \boldsymbol{A}^{(k)}{}_{[k+1]12} \\ \boldsymbol{A}^{(k)}{}_{[k+1]21} & \boldsymbol{A}^{(k)}{}_{[k+1]22} \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{G}^{(k)}{}_{[k+1]11} & \boldsymbol{0} \\ \boldsymbol{G}^{(k)}{}_{[k+1]21} & \boldsymbol{G}^{(k)}{}_{[k+1]22} \end{pmatrix} \begin{pmatrix} \boldsymbol{G}^{(k-1)}{}_{[k+1]11} & \boldsymbol{0} \\ \boldsymbol{G}^{(k-1)}{}_{[k+1]21} & \boldsymbol{G}^{(k-1)}{}_{[k+1]22} \end{pmatrix} \cdots \\ &\quad \cdots \begin{pmatrix} \boldsymbol{G}^{(1)}{}_{[k+1]11} & \boldsymbol{0} \\ \boldsymbol{G}^{(1)}{}_{[k+1]21} & \boldsymbol{G}^{(1)}{}_{[k+1]22} \end{pmatrix} \begin{pmatrix} \boldsymbol{A}_{[k+1]11} & \boldsymbol{A}_{[k+1]12} \\ \boldsymbol{A}_{[k+1]21} & \boldsymbol{A}_{[k+1]22} \end{pmatrix} \end{aligned} \tag{6.31}$$

donde se ha realizado la misma partición en las matrices $\boldsymbol{G}$ mostrando explícitamente sus menores principales y su estructura triangular inferior. A partir de 6.31 es fácil ver que

$$\boldsymbol{A}^{(k)}{}_{[k+1]11} = \boldsymbol{G}^{(k)}{}_{[k+1]11} \, \boldsymbol{G}^{(k-1)}{}_{[k+1]11} \, \ldots \, \boldsymbol{G}^{(1)}{}_{[k+1]11} \, \boldsymbol{A}_{[k+1]11}.$$

Tomando ahora determinantes y recordando que todas las matrices $\boldsymbol{G}^{(i)}{}_{[k+1]11}$ tienen diagonales unitarias y que $\det(\boldsymbol{A}_{[k+1]11}) \neq 0$ por hipótesis, es evidente que $\det(\boldsymbol{A}^{(k)}{}_{[k+1]11}) \neq 0$. Por lo tanto queda demostrado que el pivote $a^{(k)}_{k+1,k+1}$ es no nulo y el proceso de eliminación puede seguir. Por inducción se obtendrá la descomposición de $\boldsymbol{A}$ en $\boldsymbol{LU}$.

Para demostrar la condición necesaria en primer lugar se verá que si $\boldsymbol{A} = \boldsymbol{LU}$ entonces $\boldsymbol{A}_{[m]11} = \boldsymbol{L}_{[m]11}\boldsymbol{U}_{[m]11}$. Para ello basta reutilizar la partición de las matrices realizada anteriormente, es decir

$$\begin{pmatrix} \boldsymbol{A}_{[m]11} & \boldsymbol{A}_{[m]12} \\ \boldsymbol{A}_{[m]21} & \boldsymbol{A}_{[m]22} \end{pmatrix} = \begin{pmatrix} \boldsymbol{L}_{[m]11} & \boldsymbol{0} \\ \boldsymbol{L}_{[m]21} & \boldsymbol{L}_{[m]22} \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_{[m]11} & \boldsymbol{U}_{[m]12} \\ \boldsymbol{0} & \boldsymbol{U}_{[m]22} \end{pmatrix}$$

y efectuar el producto por cajas de las matrices. Además, al ser $\boldsymbol{A}$ no singular por hipótesis, ni $\boldsymbol{L}$ ni $\boldsymbol{U}$ lo serán y por lo tanto $\boldsymbol{L}_{[m]11}$ y $\boldsymbol{U}_{[m]11}$ tienen determinantes no nulos. El producto de estas dos últimas matrices tampoco tendrá determinante nulo; es decir, $\boldsymbol{A}_{[m]11}$ es no singular. Todo ello puede realizarse para cualquier valor de $m = 1, \ldots, n$ y de esta forma la condición queda demostrada. ■

Este resultado muestra que el método de Gauss sin pivotamiento puede emplearse si y sólo si todos los menores principales de $\boldsymbol{A}$ son no singulares. Esta condición es obviamente muy difícil de verificar a priori. De hecho, en la mayoría de los casos, es el propio proceso de Gauss el que indica si es necesario o no realizar pivotamiento de filas. De cualquier forma, es importante señalar una extensión del teorema anterior para dos clases de matrices extraordinariamente comunes en las ciencias de la ingeniería.

Observación: Si la matriz $\boldsymbol{A}$ es simétrica y definida positiva (negativa) entonces sus menores principales también son simétricos y definidos positivos (negativos), es decir, son no singulares, y, por consiguiente, se puede aplicar el método de Gauss sin pivotamiento. □

Para demostrar que $\boldsymbol{A}_{[m]11}$ (menor principal de orden m de la matriz $\boldsymbol{A}$) es definido positivo (negativo) cuando $\boldsymbol{A}$ es definida positiva (negativa), basta trabajar con vectores cualesquiera tales que sólo las primeras m componentes sean no nulas, es decir, del tipo: $({\boldsymbol{x}_{[m]}}^T\ \boldsymbol{0}^T)$.

Observación: Si la matriz $\boldsymbol{A}$ es estrictamente diagonalmente dominante entonces sus menores principales son no singulares, y, por consiguiente, se puede aplicar el método de Gauss sin pivotamiento. □

Para demostrarlo, basta recordar que todos los menores de $\boldsymbol{A}$ son diagonalmente dominantes y que toda matriz diagonalmente dominante es no singular.

6.2.4 Métodos de descomposición

INTRODUCCIÓN

Los métodos de descomposición (o factorización) se fundamentan en las ideas básicas descritas en el apartado anterior, donde se ha demostrado que toda matriz regular $\boldsymbol{A}$ puede, con las permutaciones adecuadas, descomponerse en el producto de una matriz triangular inferior $\boldsymbol{L}$ por una matriz triangular superior $\boldsymbol{U}$. Supóngase, para mayor claridad de la exposición, que las permutaciones no son necesarias (véase el último teorema del apartado anterior y las observaciones subsiguientes). Es decir, se dispone de dos matrices $\boldsymbol{L}$ y $\boldsymbol{U}$ tales que $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$. Entonces, el sistema de ecuaciones original $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ puede escribirse como

$$\begin{cases} \boldsymbol{L}\boldsymbol{y} = \boldsymbol{b} \\ \boldsymbol{U}\boldsymbol{x} = \boldsymbol{y} \end{cases} \tag{6.32}$$

donde puede observarse claramente que resolver el sistema original es equivalente a realizar una sustitución hacia adelante para determinar $\boldsymbol{y}$ y una sustitución hacia atrás para hallar la solución $\boldsymbol{x}$.

El objetivo será, por consiguiente, desarrollar algoritmos que, de forma eficiente, permitan descomponer la matriz original en el producto de matrices triangulares (y, en algunos métodos, también matrices diagonales) para acabar resolviendo *sistemas de ecuaciones con solución inmediata.*

En el subapartado anterior (Gauss compacto) ya se plantea, de hecho, un método de descomposición. En ese caso, la matriz $\boldsymbol{L}$ se obtiene de forma simple, pero la matriz $\boldsymbol{U}$ requiere de las operaciones clásicas del método de Gauss, ecuaciones (6.18). Este método compacto se

empleó con frecuencia a finales de los años cincuenta para resolver manualmente los sistemas inducidos por las primeras aplicaciones del método de los elementos finitos en ingeniería. Con la aparición de los ordenadores, las mismas operaciones del método de Gauss empezaron a automatizarse. Por desgracia, los primeros ordenadores disponían de poca memoria y, por tanto, de poca precisión (los cálculos habituales hoy en día con reales de 64 bits o con enteros de 16 bits entonces eran un lujo); aparecieron los primeros problemas graves de propagación de errores de redondeo y, por tanto, se invirtieron muchos esfuerzos para estudiar su origen y para desarrollar diversas técnicas que disminuyeran su influencia en el resultado final (Householder, 1964; Wilkinson, 1965). Los resultados de dichas investigaciones fueron muy extensos, pues abarcaban desde el establecimiento de una notación estándar en análisis numérico de matrices hasta las acotaciones de dichos errores, pero, en particular, dieron lugar a los métodos de descomposición.

La sistematización de las modificaciones que realiza el método de Gauss compacto sobre la matriz original $\boldsymbol{A}$ dio lugar al método de Doolittle. El método de Crout resulta de la misma sistematización pero ahora tomando la matriz $\boldsymbol{U}$ con diagonal unitaria (en vez de la matriz $\boldsymbol{L}$). En el próximo subapartado se describe en detalle este método desarrollado por Prescott D. Crout en 1941 para, según el autor, evaluar determinantes y (sólo en segundo lugar) resolver sistemas lineales. Los demás métodos de descomposición son casos particulares de los anteriores. Por consiguiente, los métodos de descomposición son simplemente una *sistematización del método de Gauss*. Es decir, consisten en organizar las operaciones que se realizan en el método de Gauss de una forma distinta y, por tanto, las propiedades estudiadas en el apartado anterior (unicidad de la descomposición, necesidad de pivotamiento, etc.) son idénticas en todos los casos. A pesar de ello, tienen un enfoque conceptual distinto (eliminar la triangular inferior *versus* descomponer en producto de triangulares) y esta sutil diferencia induce ciertas particularidades que conviene tener presentes:

1. Los métodos de descomposición son especialmente indicados para resolver *sucesivamente* varios sistemas lineales con la misma matriz y distintos términos independientes, sin necesidad, como se había comentado anteriormente para el método de Gauss original, de "recordar" las operaciones de fila realizadas durante el proceso de eliminación. De hecho, la matriz triangular inferior $\boldsymbol{L}$ representa una forma compacta de recordar dichas operaciones, como se ha visto para la forma compacta del método de Gauss. Por tanto, una vez descompuesta la matriz $\boldsymbol{A}$ en el producto de triangulares, basta realizar las dos sustituciones indicadas en 6.32 para tantos vectores $\boldsymbol{b}$ como se desee. En contrapartida, cuando todos los términos independientes son conocidos de antemano y se quiere resolver todos los sistemas *simultáneamente*, la simplicidad que existía en el método de Gauss original para realizar las operaciones de fila sobre todos los términos independientes en paralelo se ha perdido. En la práctica, cuando se opta por un método de descomposición, aunque se conozcan todos los términos independientes de antemano, se procede de forma secuencial resolviendo consecutivamente para cada término independiente dos sistemas triangulares.

2. La compacidad de las operaciones de los métodos de descomposición permite mejoras en la precisión de los resultados empleando únicamente unas pocas variables de precisión alta. Esta ventaja de los métodos de descomposición respecto de los métodos de eliminación ha perdido importancia con el progresivo abaratamiento de la memoria.

Método de Crout

A continuación se presenta el método de Crout en su versión estándar, es decir, sin pivotamiento y para matrices llenas. El método de Doolittle tiene un desarrollo paralelo que no se considera necesario realizar aquí. En el método de Crout se realiza la descomposición de la matriz $\boldsymbol{A}$ en una $\boldsymbol{L}$ por una $\boldsymbol{U}$, esta última con diagonal unitaria.

Este método tiene un planteamiento recursivo en el que se descomponen sucesivamente los menores principales de la matriz $\boldsymbol{A}$. Se empieza por el menor principal de orden 1, luego el de orden 2 y así sucesivamente hasta el menor de orden n, es decir, la matriz original. En esta ocasión, los menores principales de orden m de la matriz $\boldsymbol{A}$ se denotan por $\boldsymbol{A}_{[m]}$. La descomposición del menor principal de orden 1 es sencilla, puesto que $\boldsymbol{A}_{[1]} = a_{11}$ y se ha tomado $\boldsymbol{U}$ con diagonal unitaria: $l_{11} = a_{11}$ y $u_{11} = 1$. Suponiendo ahora descompuesto el menor de orden k, es decir, $\boldsymbol{A}_{[k]} = \boldsymbol{L}_{[k]}\boldsymbol{U}_{[k]}$, interesa estudiar cómo se descompone el siguiente menor principal $\boldsymbol{A}_{[k+1]}$.

Conviene primero escribir $\boldsymbol{A}_{[k+1]}$ en función de $\boldsymbol{A}_{[k]}$ y de los coeficientes de $\boldsymbol{A}$ necesarios:

$$\boldsymbol{A}_{[k+1]} = \begin{pmatrix} \boldsymbol{A}_{[k]} & \boldsymbol{c}_{[k+1]} \\ \boldsymbol{f}^T_{[k+1]} & a_{k+1,k+1} \end{pmatrix} \tag{6.33}$$

donde $\boldsymbol{c}_{[k+1]}$ y $\boldsymbol{f}_{[k+1]}$ son vectores de $\mathbb{R}^k$:

$$\boldsymbol{c}_{[k+1]} = \begin{pmatrix} a_{1,k+1} \\ a_{2,k+1} \\ \vdots \\ a_{k,k+1} \end{pmatrix} \qquad \boldsymbol{f}_{[k+1]} = \begin{pmatrix} a_{k+1,1} \\ a_{k+1,2} \\ \vdots \\ a_{k+1,k} \end{pmatrix} \tag{6.34}$$

Seguidamente, se establece la descomposición de $\boldsymbol{A}_{[k+1]}$ como

$$\begin{gathered} \boldsymbol{A}_{[k+1]} = \boldsymbol{L}_{[k+1]}\,\boldsymbol{U}_{[k+1]} \\ \begin{pmatrix} \boldsymbol{A}_{[k]} & \boldsymbol{c}_{[k+1]} \\ \boldsymbol{f}^T_{[k+1]} & a_{k+1,k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{L}_{[k]} & \boldsymbol{0} \\ \boldsymbol{l}^T_{[k+1]} & l_{k+1,k+1} \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_{[k]} & \boldsymbol{u}_{[k+1]} \\ \boldsymbol{0}^T & 1 \end{pmatrix} \end{gathered} \tag{6.35}$$

donde aparecen los siguientes vectores de $\mathbb{R}^k$:

$$\boldsymbol{l}_{[k+1]} = \begin{pmatrix} l_{k+1,1} \\ l_{k+1,2} \\ \vdots \\ l_{k+1,k} \end{pmatrix} \qquad \boldsymbol{u}_{[k+1]} = \begin{pmatrix} u_{1,k+1} \\ u_{2,k+1} \\ \vdots \\ u_{k,k+1} \end{pmatrix} \qquad \boldsymbol{0} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{6.36}$$

Y, por último, tras proceder a multiplicar las matrices $\boldsymbol{L}_{[k+1]}$ y $\boldsymbol{U}_{[k+1]}$ en la ecuación 6.35, se obtienen las ecuaciones necesarias para la descomposición del menor principal $\boldsymbol{A}_{[k+1]}$. Es decir,

$$\begin{aligned} \boldsymbol{L}_{[k]}\, \boldsymbol{u}_{[k+1]} &= \boldsymbol{c}_{[k+1]} \\ \boldsymbol{U}^T_{[k]}\, \boldsymbol{l}_{[k+1]} &= \boldsymbol{f}_{[k+1]} \\ l_{k+1,k+1} &= a_{k+1,k+1} - \boldsymbol{l}^T_{[k+1]} \cdot \boldsymbol{u}_{[k+1]} \end{aligned} \tag{6.37}$$

Nótese que los vectores $\boldsymbol{u}_{[k+1]}$ y $\boldsymbol{l}_{[k+1]}$ se obtienen resolviendo (mediante sustitución hacia adelante) sistemas con matrices triangulares inferiores $\boldsymbol{L}_{[k]}$ y $\boldsymbol{U}^T_{[k]}$.

El algoritmo de descomposición de Crout se obtiene finalmente repitiendo las ecuaciones 6.37 para $k = 1, \ldots, n-1$, sustituyendo las definiciones de los vectores por las ecuaciones 6.34 y 6.36, y planteando explícitamente las dos sustituciones hacia adelante. Es decir:

$$\begin{aligned} & l_{11} = a_{11} \qquad\qquad u_{11} = 1 \\ & k = 1, \ldots, n-1 \\ & \left\{ \begin{array}{lll} u_{1,k+1} & = a_{1,k+1} \,/\, l_{11} & \\ u_{i,k+1} & = \left(a_{i,k+1} - \displaystyle\sum_{j=1}^{i-1} l_{ij}\, u_{j,k+1} \right) \,/\, l_{ii} & i = 2, \ldots, k \\ l_{k+1,1} & = a_{k+1,1} & \\ l_{k+1,i} & = a_{k+1,i} - \displaystyle\sum_{j=1}^{i-1} u_{ji}\, l_{k+1,j} & i = 2, \ldots, k \\ u_{k+1,k+1} & = 1 & \\ l_{k+1,k+1} & = a_{k+1,k+1} - \displaystyle\sum_{i=1}^{k} l_{k+1,i}\, u_{i,k+1} & \end{array} \right. \end{aligned} \tag{6.38}$$

Como ya se ha indicado, se ha supuesto que la matriz $\boldsymbol{A}$ es tal que permite su descomposición sin pivotamiento; si esto no fuera así, algún $l_{k+1,k+1}$ se anularía lo cual impediría la descomposición del siguiente menor, puesto que se produciría una división por cero. Las técnicas de pivotamiento también pueden emplearse con estos métodos y son muy extendidas a pesar de que complican considerablemente el algoritmo expuesto en la ecuación 6.38.

También conviene observar que el número de operaciones necesarias para la descomposición (ecuación 6.38) y posterior resolución de los sistemas triangulares (ecuación 6.32) coincide exactamente con el método de Gauss. A pesar de ello, cada elemento de $\boldsymbol{L}$ y $\boldsymbol{U}$ se evalúa de forma compacta con las ecuaciones descritas anteriormente. Por este motivo, puede reducirse la

propagación de errores de redondeo si los sumatorios que aparecen en las expresiones de $u_{i,k+1}$ y $l_{k+1,i}$ se evalúan con una precisión mayor (sólo es necesario un número con precisión alta). Esta técnica de mejora de la precisión fue importante en su día, pero ha caído en desuso debido al abaratamiento de la memoria. Además, para algunos sistemas lineales, la combinación de variables de precisión normal y de precisión alta puede llegar a producir más errores de redondeo que trabajar solamente con variables de precisión normal.

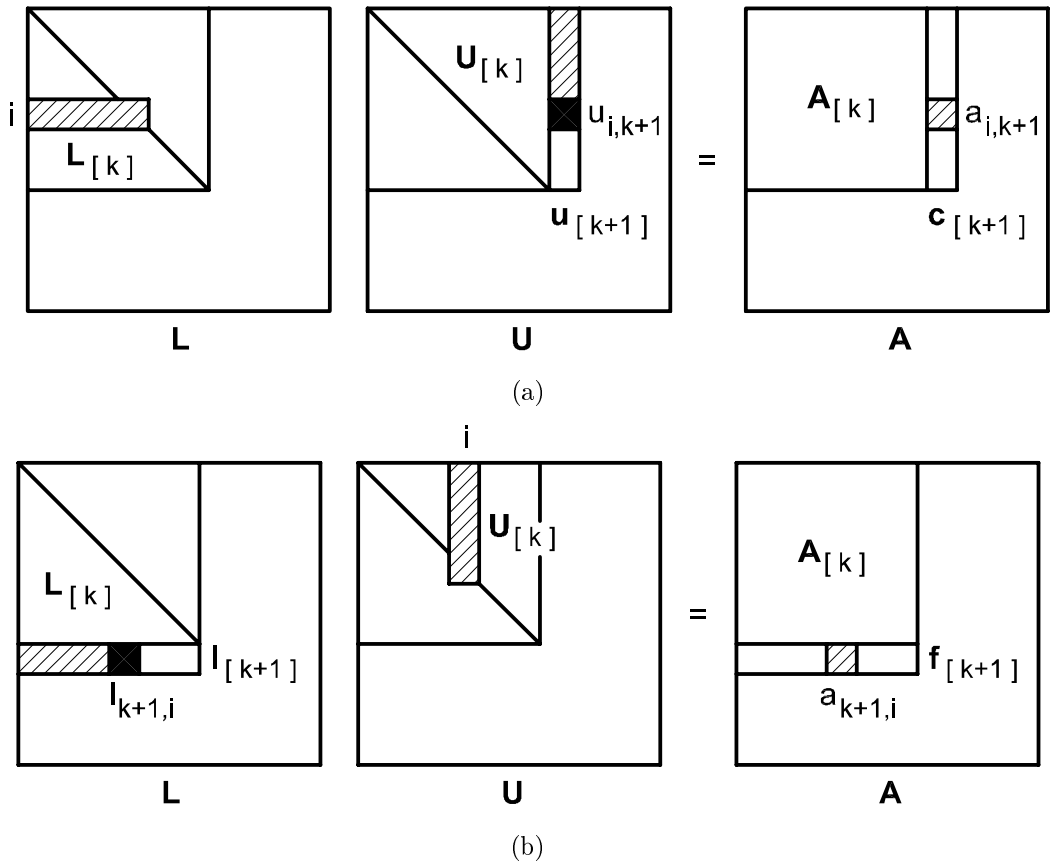


Fig. 6.2 Esquema gráfico de la descomposición en el método de Crout. Obtención de: a) la matriz triangular superior; b) matriz triangular inferior. Para calcular el elemento en negro son necesarios los elementos tramados.

Por último, conviene señalar que, según se aprecia en la figura 6.2a, para evaluar un elemento $u_{i,k+1}$ de $\boldsymbol{U}$ únicamente es necesario conocer el elemento correspondiente $a_{i,k+1}$ de $\boldsymbol{A}$, los de $\boldsymbol{U}$ que se encuentren encima de él en su propia columna ($u_{s,k+1}$, $s = 1, \ldots, i-1$) y la descomposición previa del menor de $\boldsymbol{A}$ de orden k. Es decir, no son necesarios ninguno de los coeficientes de $\boldsymbol{A}$ correspondientes a la columna $k+1$ por encima de $u_{i,k+1}$ ni del menor citado. En otras palabras, el elemento $a_{i,k+1}$ es necesario para calcular el $u_{i,k+1}$, pero no para

calcular los siguientes elementos de $\boldsymbol{U}$. Por consiguiente, se puede almacenar $\boldsymbol{U}$ "sobre" (es decir, empleando los mismos espacios de memoria) la matriz triangular superior de $\boldsymbol{A}$. Pueden deducirse conclusiones similares para la matriz $\boldsymbol{L}$ (véase la figura 6.2b). Es decir, el método de Crout está perfectamente diseñado para poder reducir las necesidades de memoria almacenando las dos matrices triangulares en las mismas posiciones de memoria que ocupaba originalmente la matriz $\boldsymbol{A}$ (la diagonal de $\boldsymbol{U}$ por ser unitaria no es necesario guardarla). Más aún; como se comentará en el capítulo 7, el almacenamiento de $\boldsymbol{A}$, y por tanto el de $\boldsymbol{L}$ y $\boldsymbol{U}$, puede reducirse en el caso de que la matriz $\boldsymbol{A}$ tenga estructura en banda o *skyline*. En realidad, los esquemas de almacenamiento para estos tipos de matrices se diseñaron para los métodos directos y conviene emplearlos siempre que la dimensión de las matrices exceda las pocas decenas.

Problema 6.2:
El objetivo de este problema es comprobar que el método de Crout puede implementarse reservando espacio de memoria únicamente para *una* matriz y *un* vector. Tal y como se acaba de comentar, las matrices triangulares $\boldsymbol{L}$ y $\boldsymbol{U}$ se pueden almacenar sobre la matriz $\boldsymbol{A}$. Se pide:

a) Comprobar que en la fase de sustituciones del método de Crout, ecuación 6.32, el vector $\boldsymbol{y}$ puede almacenarse sobre $\boldsymbol{b}$, y el vector $\boldsymbol{x}$ puede almacenarse sobre $\boldsymbol{y}$ (es decir, que basta reservar espacio de memoria para un vector).

b) Escribir el pseudocódigo del método de Crout (descomposición y sustituciones) empleando únicamente una matriz y un vector. ●

Método de Cholesky

Un caso bastante usual por el gran número de aplicaciones que se encuentran en el marco de la resolución numérica de ecuaciones en derivadas parciales (métodos de diferencias finitas, elementos finitos, etc.) es el de sistemas con matrices *simétricas y definidas positivas*. Como ya se ha visto anteriormente, este es un caso en el que no es necesario pivotar (véase la observación al respecto al final del análisis matricial del método de Gauss), y además conviene explotar el carácter simétrico de $\boldsymbol{A}$ tanto desde el punto de vista del número de operaciones como del almacenamiento (véase el problema 6.1). El método de Cholesky se propone precisamente utilizar esta información previa para realizar una descomposición más eficaz que el método de Crout.

De nuevo, la descomposición se realiza consecutivamente sobre todos los menores de $\boldsymbol{A}$ y ahora el objetivo es encontrar $\boldsymbol{L}$ tal que $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$. Es decir, la incógnita es una única matriz triangular inferior $\boldsymbol{L}$, y su traspuesta $\boldsymbol{L}^T$ hace de matriz triangular superior. El menor de orden 1 es, de nuevo, muy simple: $l_{11} = \sqrt{a_{11}}$, y los siguientes se obtienen a partir de otra relación de recurrencia. Descompuesto el menor de orden k, $\boldsymbol{A}_{[k]} = \boldsymbol{L}_{[k]}\boldsymbol{L}_{[k]}^T$, se establece la descomposición de $\boldsymbol{A}_{[k+1]}$ como

$$\begin{gathered}\boldsymbol{A}_{[k+1]} = \boldsymbol{L}_{[k+1]}\,\boldsymbol{L}_{[k+1]}^T \\ \begin{pmatrix} \boldsymbol{A}_{[k]} & \boldsymbol{f}_{[k+1]} \\ \boldsymbol{f}_{[k+1]}^T & a_{k+1,k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{L}_{[k]} & \boldsymbol{0} \\ \boldsymbol{l}_{[k+1]}^T & l_{k+1,k+1} \end{pmatrix} \begin{pmatrix} \boldsymbol{L}_{[k]}^T & \boldsymbol{l}_{[k+1]} \\ \boldsymbol{0}^T & l_{k+1,k+1} \end{pmatrix}\end{gathered} \tag{6.39}$$

Nótese que, por ser $\boldsymbol{A}$ una matriz simétrica, los vectores $\boldsymbol{c}_{[k+1]}$ y $\boldsymbol{f}_{[k+1]}$ definidos en la ecuación 6.34 coinciden. A partir de 6.39 se obtienen las ecuaciones necesarias para la descomposición

del menor principal $\boldsymbol{A}_{[k+1]}$:

$$\begin{aligned} \boldsymbol{L}_{[k]}\, \boldsymbol{l}_{[k+1]} &= \boldsymbol{f}_{[k+1]} \\ l_{k+1,k+1} &= \sqrt{a_{k+1,k+1} - \boldsymbol{l}_{[k+1]}^T \cdot \boldsymbol{l}_{[k+1]}} \end{aligned} \tag{6.40}$$

Detallando la sustitución hacia adelante para el cálculo de $\boldsymbol{l}_{[k+1]}$ se llega al algoritmo:

$$\begin{aligned} &l_{11} = \sqrt{a_{11}} \\ &k = 1, \ldots, n-1 \\ &\begin{cases} l_{k+1,1} &= a_{k+1,1} \;/\; l_{11} \\ l_{k+1,i} &= \left(a_{k+1,i} - \displaystyle\sum_{j=1}^{i-1} l_{ij}\, l_{k+1,j} \right) \;/\; l_{ii} \qquad i = 2, \ldots, k \\ l_{k+1,k+1} &= \sqrt{a_{k+1,k+1} - \displaystyle\sum_{i=1}^{k} l_{k+1,i}^2} \end{cases} \end{aligned} \tag{6.41}$$

Para concluir, es preciso comprobar que este algoritmo puede utilizarse para cualquier matriz simétrica y definida positiva. El algoritmo sólo fallaría si se se realizaran divisiones por cero o se calcularan raíces cuadradas de números negativos.

Teorema: La descomposición de Cholesky dada por la ecuación 6.41 puede realizarse para cualquier matriz $\boldsymbol{A}$ simétrica y definida positiva.

Demostración: Por ser $\boldsymbol{A}$ simétrica y definida positiva, $a_{11} > 0$ y $l_{11} = \sqrt{a_{11}} > 0$. A partir de aquí, se procede recursivamente: se supone que se tiene la descomposición $\boldsymbol{A}_{[k]} = \boldsymbol{L}_{[k]}\boldsymbol{L}_{[k]}^T$ (es decir, que se han ido obteniendo coeficientes l_{ii} positivos, con $i = 1, \ldots, k$), y se comprueba que puede hacerse la descomposición $\boldsymbol{A}_{[k+1]} = \boldsymbol{L}_{[k+1]}\boldsymbol{L}_{[k+1]}^T$. Las ecuaciones 6.39 y 6.40 permiten expresar el determinante del menor $\boldsymbol{A}_{[k+1]}$ como $\det(\boldsymbol{A}_{[k+1]}) = \det(\boldsymbol{L}_{[k]})^2(a_{k+1,k+1} - \boldsymbol{l}_{[k+1]}^T \cdot \boldsymbol{l}_{[k+1]})$. Se verifica que $\det(\boldsymbol{A}_{[k+1]}) > 0$ (por ser $\boldsymbol{A}$ simétrica y definida positiva) y que $\det(\boldsymbol{L}_{[k]})^2 > 0$ (por ser positivos todos sus elementos diagonales). En consecuencia, $a_{k+1,k+1} - \boldsymbol{l}_{[k+1]}^T \cdot \boldsymbol{l}_{[k+1]}$ es positivo, y al extraer la raíz cuadrada se obtiene $l_{k+1,k+1} > 0$. ■

Métodos $\boldsymbol{LDU}$ y $\boldsymbol{LDL}^T$

Como se ha visto, el método de Crout consiste en descomponer la matriz $\boldsymbol{A}$ en el producto de dos matrices triangulares, una de las cuales (la matriz triangular superior $\boldsymbol{U}$) tiene unos en la diagonal. Si se desea, puede conseguirse que ambas matrices ($\boldsymbol{L}$ y $\boldsymbol{U}$) tengan diagonales unitarias, a cambio de añadir una matriz diagonal $\boldsymbol{D}$. Se obtiene entonces la descomposición

generalizada $\boldsymbol{A} = \boldsymbol{LDU}$. Pueden obtenerse las matrices $\boldsymbol{L}$, $\boldsymbol{D}$ y $\boldsymbol{U}$ de forma recursiva, tal y como se ha visto para el método de Crout.

Asimismo, la descomposición de Cholesky $\boldsymbol{A} = \boldsymbol{LL}^T$ puede generalizarse en $\boldsymbol{A} = \boldsymbol{LDL}^T$. De nuevo, a cambio de añadir una matriz diagonal $\boldsymbol{D}$, puede imponerse que $\boldsymbol{L}$ tenga diagonal unitaria.

Para ambos métodos, una vez efectuada la descomposición, la solución $\boldsymbol{x}$ del sistema lineal se obtiene resolviendo tres sistemas con solución inmediata: dos triangulares y uno diagonal.

Problema 6.3:
A partir de lo visto para los métodos de Crout y de Cholesky en los subapartados anteriores,
a) Generalizar la ecuación 6.38 al método $\boldsymbol{LDU}$.
b) Generalizar la ecuación 6.41 al método $\boldsymbol{LDL}^T$. ●

El rango de aplicación de las descomposiciones $\boldsymbol{LU}$ y $\boldsymbol{LDU}$ es el mismo. No ocurre lo mismo para las descomposiciones simétricas $\boldsymbol{LL}^T$ y $\boldsymbol{LDL}^T$: solamente las matrices simétricas y definidas positivas pueden escribirse como $\boldsymbol{A} = \boldsymbol{LL}^T$; en cambio, cualquier matriz simétrica que no requiera pivotamiento (por ejemplo, las matrices simétricas y definidas negativas) puede expresarse como $\boldsymbol{A} = \boldsymbol{LDL}^T$.

6.3 Bibliografía

BREUER, S.; ZWAS, G. *Numerical Mathematics. A Laboratory Approach.* Cambridge University Press, 1993.

CIARLET, P.G. *Introduction à l'Analyse Numérique Matricielle et à l'Optimisation.* Masson, 1982.

HOUSEHOLDER, A.S. *The Theory of Matrices in Numerical Analysis.* Dover Publications Inc., 1964.

PRESS, W.H.; FLANNERY, B.P.; TEUKOLSKY, S.A.; VETTERLING, W.T. *Numerical Recipes. The Art of Scientific Computing.* Cambridge University Press, 1986.

RALSTON, A.; RABINOWITZ, P. *A First Course in Numerical Analysis.* McGraw-Hill, 1978.

STEWART G.W. *Afternotes on Numerical Analysis.* Society for Industrial Applied Mathematics, 1996.

WILKINSON, J.H. *The Algebraic Eigenvalue Problem.* Oxford University Press, 1965.

7 Programación y aspectos computacionales de los sistemas lineales de ecuaciones

Objetivos

- Estudiar y analizar el dimensionamiento de matrices.
- Desarrollar diversos programas FORTRAN para sistemas triviales.
- Realizar varias consideraciones generales sobre la memoria e introducir el concepto de dimensionamiento dinámico.
- Presentar los esquemas de almacenamiento para matrices diagonales, triangulares y en banda.

7.1 Programación

7.1.1 Dimensionamiento de matrices

En el álgebra numérica lineal es necesario trabajar con vectores y matrices. Estos vectores y matrices deben tener su representación en el ordenador para poder programar los distintos algoritmos.

En primer lugar, es necesario asignar a cada vector y matriz el nombre que le corresponda. Por ejemplo, un vector dado $\boldsymbol{v} \in \mathbb{R}^n$ puede denominarse `vect`, mientras que la matriz $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ se llamará `amat`. Si $n = 3$, las componentes de $\boldsymbol{v}$, v_1, v_2 y v_3, vienen representadas en el programa por `vect(1)`, `vect(2)` y `vect(3)`. De forma similar, si ahora se especifica $m = 2$, se tiene que los elementos (coeficientes) de la matriz $\boldsymbol{A}$, a_{11}, a_{12}, a_{13}, a_{21}, a_{22} y a_{23}, se escriben en FORTRAN como `amat(1,1)`, `amat(1,2)`, `amat(1,3)`, `amat(2,1)`, `amat(2,2)` y `amat(2,3)`.

El número de subíndices que como máximo puede tener una matriz depende del compilador empleado, aunque es usual que no pueda exceder de siete subíndices. En el ejemplo que se muestra se han empleando nombres, tanto para el vector como la matriz, tales que por defecto el compilador interpreta que son matrices de números reales. De cualquier modo, las matrices, de la misma manera que todas las variables en FORTRAN, pueden definirse como se desee: `INTEGER*2`, `INTEGER*4`, `REAL*4`, `REAL*8`, `COMPLEX*8`, `COMPLEX*16`, `CHARACTER` y `LOGICAL`. Los subíndices de las matrices son, por el contrario, siempre variables enteras.

Una vez se tienen definidas las componentes de un vector o bien los coeficientes de la matriz, se puede operar con ellos tal como se haría con variables estándares. Por ejemplo, si se desea calcular el módulo de $\boldsymbol{v}$, `vmod`, basta escribir:

```
c___Modulo de vector
      vmod = SQRT( vect(1)*vect(1) + vect(2)*vect(2) + vect(3)*vect(3) )
```

mientras que si se desea evaluar el vector $\boldsymbol{u} \in \mathbb{R}^2$, denominado `uvec`, como producto de $\boldsymbol{A}$ por $\boldsymbol{v}$, es decir $\boldsymbol{u} = \boldsymbol{A}\boldsymbol{v}$, entonces:

```
c___Producto de matriz por vector
      uvec(1) = amat(1,1)*vect(1) + amat(1,2)*vect(2) +
     .          amat(1,3)*vect(3)
      uvec(2) = amat(2,1)*vect(1) + amat(2,2)*vect(2) +
     .          amat(2,3)*vect(3)
```

La ventaja de trabajar con subíndices es precisamente que no resulta necesario explicitar cada elemento como se ha hecho. Como resulta fácil de imaginar, si las dimensiones n o m fueran valores habituales en las aplicaciones (del orden de varios miles) la programación de operaciones tan simples como evaluar el módulo de un vector o el producto de matriz por vector resultaría algo engorroso. Empleando la instrucción `DO` ambos ejemplos pueden reescribirse como:

```
c___Modulo de vector
      ndim = 3
      vmod = 0.0e0
      do 10 i=1,ndim
        vmod = vmod + vect(i)*vect(i)
   10 continue
      vmod = SQRT(vmod)
```

donde se ha introducido la variable `ndim` que indica la dimensión de $\boldsymbol{v}$ y que permite emplear el mismo código para cualquier dimensión deseada. El segundo programa sería:

```
c___Producto de matriz por vector
      ndim = 3
      mdim = 2
      do 10 i=1,mdim
        uvec(i) = 0.0e0
        do 10 j=1,ndim
          uvec(i) = uvec(i) + amat(i,j)*vect(j)
   10 continue
```

Cuando en un programa se emplean variables con subíndices (vectores y/o matrices) es necesario proporcionar la siguiente información:

1. ¿Qué variables tienen subíndice?
2. ¿Cuántos subíndices tiene cada variable?
3. ¿Cuál es el rango de valores de cada subíndice?

Para responder a estas preguntas existe la sentencia no ejecutable que se inicia con la palabra `dimension`. Esta sentencia debe situarse al inicio del programa (en la zona de sentencias no ejecutables), desde luego antes de utilizar la variable correspondiente. Esta instrucción puede afectar a todo tipo de variables (enteras, reales, etc.), previamente definidas, y pueden escribirse tantas instrucciones `dimension` como sea necesario. Por ejemplo:

```
dimension amat(2,3),vect(3),uvec(2)
dimension bmat(0:10,-90:1),cmat(23:230,-314:-157,-1:1,23,-5:-5)
```

En la primera instrucción se muestra el dimensionamiento de la matriz y los vectores empleados en los ejemplos anteriores y, en la segunda, otro dimensionamiento de matrices para que se pueda observar la gran libertad disponible para indicar el rango de valores de los subíndices. El límite inferior de los subíndices es un entero negativo, nulo o positivo; su valor por defecto es uno. El límite superior puede ser, de nuevo, un entero negativo, nulo o positivo, siempre que sea superior o igual al límite inferior. Para referirse a las componentes de la matriz se emplean subíndices entre los límites designados. De esta forma, `cmat(100,-300,0,10,-5)` tiene sentido, mientras que `cmat(1,-300,0,10,-5)` es una posición de memoria, en principio, desconocida.

En realidad, la instrucción `DIMENSION` se emplea cuando las variables han sido definidas o bien utilizan su definición implícita (por ejemplo serán enteras aquellas que empiecen por las letras de `i` a `n`). De hecho, existe una manera abreviada de definirlas y dimensionarlas. Por ejemplo, las siguientes instrucciones

```
real*4 amat, uvec, bmat
real*8 vect
character*2 cmat
dimension amat(2,3), vect(3),uvec(2)
dimension bmat(0:10,-90:1),cmat(23:230,-314:-157,-1:1,23,-5:-5)
```

son equivalentes a

```
real*4 amat(2,3), uvec(2), bmat(0:10,-90:1)
real*8 vect(3)
character*2 cmat(23:230,-314:-157,-1:1,23,-5:-5)
```

A modo de ejemplo, en el programa 7.1 primero se generan una matriz y un vector (la primera es una matriz de Hilbert y el segundo en cada componente contiene la suma de la fila correspondiente de la matriz), a continuación se calcula su producto, el módulo del vector resultante y finalmente se escriben los datos y los resultados en un archivo.

```
c
c        Este programa calcula el producto de una matriz por un
c        vector y el modulo del vector
c
c_____________________________________________________________________
      implicit real*8 (a-h,o-z)
      parameter (na_max = 100)
      dimension a(na_max,na_max),v(na_max),w(na_max)

      write (6,*) ' Entra el orden de la matriz: '
      read (5,*) ndim

c___Generacion de la matriz (matriz de Hilbert)
      do 10 i=1, ndim
        do 10 j=1, ndim
         a(i,j)= 1.0d0/(dfloat(i+j-1))
   10 continue

c___Generacion del vector (suma de las columnas)
      do 20 i=1, ndim
        v(i)=0.0d0
        do 20 j=1, ndim
          v(i)= v(i) + a(i,j)
   20 continue

c___Calculo del producto de matriz por vector
      do 30 i=1,ndim
        w(i)=0.0d0
        do 30 j=1,ndim
            w(i)=w(i) + a(i,j)*v(j)
   30 continue

c___Calculo del modulo de un vector
      xmodulo=0.0d0
      do 40 i=1,ndim
        xmodulo=xmodulo+w(i)*w(i)
   40 continue
      xmodulo=dsqrt(xmodulo)

c___Escritura de los resultados
      open (unit=10,file='p7_1.res',status='new')

      write (10,50)
      do 60 i=1,ndim
        write (10,70) (a(i,j),j=1,ndim)
   60 continue
```

```
      write (10,80)
      do 90 i=1,ndim
        write (10,100) v(i)
   90 continue

      write (10,110)
      do 120 i=1,ndim
        write (10,100) w(i)
  120 continue

      write (10,130) xmodulo

      close (10)

   50 format (' La matriz de entrada es: '/)
   70 format (5(1x,1pe14.6))
   80 format (/' El vector de entrada es: '/)
  100 format (1x,1pe14.6)
  110 format (/' El vector producto es: '/)
  130 format (/' El modulo del vector producto es: ',pe14.6)

      stop
      end
```

Prog. 7.1 Cálculo del producto de una matriz por un vector y del módulo del vector

En el programa 7.1 se ha introducido un nueva sentencia: es el denominado `DO` implícito. En el fragmento

```
    do 60 i=1,ndim
      write (10,70) (a(i,j),j=1,ndim)
60 continue
```

aparece, por una parte, el bloque `DO` que se ha presentado anteriormente (ver apartado 3.7.3). Mediante el contador `i`, el programa escribe una fila de la matriz `a` en cada línea del archivo de resultados (recuérdese que cada instrucción `write` produce automáticamente un salto de línea). Por otra parte, a través del contador `j`, *implícitamente* se escriben, para cada valor de `i`, todas las columnas de la matriz `c`.

En la tabla 7.1 se muestra el fichero de resultados creado por el programa 7.1 para el caso `ndim=4`.

Tabla 7.1 Fichero de resultados para `ndim=4`

```
La matriz de entrada es:

  1.000000E+00   5.000000E-01   3.333333E-01   2.500000E-01
  5.000000E-01   3.333333E-01   2.500000E-01   2.000000E-01
  3.333333E-01   2.500000E-01   2.000000E-01   1.666667E-01
  2.500000E-01   2.000000E-01   1.666667E-01   1.428571E-01

El vector de entrada es:

  2.083333E+00
  1.283333E+00
  9.500000E-01
  7.595238E-01

El vector producto es:

  3.231548E+00
  1.858849E+00
  1.331865E+00
  1.044337E+00

El modulo del vector producto es:   4.094231E+00
```

7.1.2 Programación estructurada: subrutinas

El programa 7.1 permite realizar el producto de la matriz de Hilbert por un vector cuyas componentes son la suma de las filas de dicha matriz; si se desea evaluar el producto de otra matriz por otro vector se deberá escribir un programa de nuevo. En realidad, el programador podría estar interesado en tener un programa más general que calcule los productos de una matriz por un vector cualquiera. Para ello es necesario agrupar y sistematizar los grupos de sentencias que realicen una tarea concreta, por ejemplo, obtener la matriz o calcularse una cierta norma. Al ejercicio de programar agrupando las sentencias que realizan una tarea concreta se le llama *programación estructurada*.

Además, en un mismo código pueden existir tareas repetitivas, por ejemplo, calcular el producto escalar de dos vectores. En este caso, se tendría que repetir tantas veces como se necesite el grupo de sentencias que realizan el cálculo del módulo de un vector (ver el ejemplo anterior). Por consiguiente, a pesar de tener una programación supuestamente estructurada, el programador debe reescribir varias veces un mismo grupo de sentencias.

Para evitar estos problemas se pueden definir las funciones externas (`FUNCTION`). Por ejemplo, para el módulo de un vector se define la función externa `XMODULO`. Pero estas funciones externas tienen la limitación de que sólo retornan un valor; es decir, con una función externa resulta complicado realizar el producto de una matriz por un vector, puesto que ahora el resultado es un vector.

Para solventar estos problemas se definen las subrutinas: `SUBROUTINE`. De hecho una subrutina se debe interpretar como un subprograma que tiene casi total independencia del programa que la requiere: las variables pueden tener nombres distintos en el programa principal y en cada uno de los subprogramas; pueden compilarse de forma independiente; pueden tener sus módulos de entrada y salida de datos... En resumen, al igual que las `FUNCTION`s, las `SUBROUTINE`s son independientes del programa principal, pero es relativamente fácil establecer una buena *comunicación* entre todas ellas.

Las funciones y la subrutinas empiezan por una sentencia con `FUNCTION` o `SUBROUTINE` y deben terminar con una `END` (todas ellas sentencias no ejecutables que indican al compilador dónde empieza y termina cada módulo). El nombre de la función o de la rutina va seguido de paréntesis que contienen los argumentos, separados por comas si hay más de uno.

`FUNCTION` *nombre_función(argumento1, argumento2,...)*

`SUBROUTINE` *nombre_subrutina(argumento1, argumento2,...)*

Mientras que con una `FUNCTION` el nombre de la misma debe aparecer al menos una vez en sus sentencias para asignarle el valor correspondiente, en el caso de una rutina su nombre no aparecerá por no estar asociado a ningún valor concreto: todos los resultados se definen en término de los argumentos y puede haber cualquier número de argumentos. Las sentencias de una subrutina no se ejecutan al introducir simplemente su nombre en una sentencia del programa, como ocurre con las funciones, sino que es necesario emplear una sentencia

`CALL` *nombre_subrutina(argumento1, argumento2,...)*

para que la rutina se ejecute. Los argumentos que se necesiten para ejecutar las instrucciones de la rutina tendrán los valores correspondientes al momento en que se efectúa la sentencia `CALL`. De la misma forma, los argumentos asociados a los resultados tendrán después de la sentencia `CALL` los valores asignados en la rutina. Es importante resaltar que la última sentencia que se ejecuta en una `FUNCTION` o `SUBROUTINE` es la sentencia `RETURN` que devuelve el control al programa principal.

Como ejemplo de lo expuesto anteriormente, en el programa 7.2 se ha reescrito el programa que calcula el producto de una matriz por un vector y el módulo de este último empleando funciones y rutinas. Como se podrá observar, dentro de las funciones y rutinas es necesario volver a indicar qué variables tienen subíndices y cuáles no.

```
c
c        Este programa calcula el producto de una matriz por un
c        vector y el modulo del vector mediante funciones y
c        rutinas
c
c_____________________________________________________________________
      implicit real*8 (a-h,o-z)
      parameter (na_max = 100)
      dimension a(na_max,na_max),v(na_max),w(na_max)

      write (6,*) ' Entra el orden de la matriz: '
      read (5,*) ndim

c___Lectura o generacion de la matriz y del vector
      call get_mat_vec (ndim,a,v)

c___Calculo del producto de matriz por vector
      call product (ndim,a,v,w)

c___Calculo del modulo de un vector
      x=xmodulo(ndim,w)

c___Escritura de los resultados
      call write_resul(ndim,a,v,w,x)

      stop
      end


c____________________Lectura o generacion de la matriz y del vector
      subroutine get_mat_vec (n,a,v)
      implicit real*8 (a-h,o-z)
      dimension a(n,n),v(n)

c___Generacion de la matriz (matriz de Hilbert)
      do 10 i=1, n
        do 10 j=1, n
         a(i,j)= 1.0d0/(dfloat(i+j-1))
   10 continue

c___Generacion del vector (suma de las columnas)
      do 20 i=1, n
        v(i)=0.0d0
        do 20 j=1, n
         v(i)= v(i) + a(i,j)
   20 continue
```

```
      return
      end


c_____________________Calculo del producto de matriz por vector
      subroutine product (n,a,v,w)
      implicit real*8 (a-h,o-z)
      dimension a(n,n),v(n),w(n)

      do 10 i=1,n
        w(i)=0.0d0
        do 10 j=1,n
            w(i)=w(i) + a(i,j)*v(j)
   10 continue

      return
      end


c_____________________Calculo del modulo de un vector
      real*8 function xmodulo(n,b)
      implicit real*8 (a-h,o-z)
      dimension b(n)

      xmodulo=0.0d0
      do 10 i=1,n
        xmodulo=xmodulo+b(i)*b(i)
   10 continue
      xmodulo=dsqrt(xmodulo)

      return
      end


c_____________________Escritura de los resultados
      subroutine write_resul(n,a_mat,b,c,x)
      implicit real*8 (a-h,o-z)
      dimension a_mat(n,n),b(n),c(n)

      open (unit=10,file='p7_2.res',status='new')

      write (10,10)
      do 20 i=1,n
        write (10,30) (a_mat(i,j),j=1,n)
   20 continue
```

```
      write (10,40)
      do 50 i=1,n
        write (10,60) b(i)
   50 continue

      write (10,70)
      do 80 i=1,n
        write (10,60) c(i)
   80 continue

      write (10,90) x

      close (10)

   10 format (' La matriz de entrada es: '/)
   30 format (5(1x,1pe14.6))
   40 format (/' El vector de entrada es: '/)
   60 format (1x,1pe14.6)
   70 format (/' El vector producto es: '/)
   90 format (/' El modulo del vector producto es: ',pe14.6)

      return
      end
```

Prog. 7.2 Cálculo del producto de una matriz por un vector y del módulo del vector mediante funciones y rutinas

Problema 7.1:
Modificar el programa 7.2 de forma que en lugar de generar una matriz de Hilbert genere una matriz de Vandermonde. La definición de los términos de la matriz de Vandermonde es:

$$a_{ij} \;=\; (x_i)^{j-1} \qquad i,j = 1,..,n$$

donde $x_1,..,x_n$, son n números reales distintos entre sí. Como puede observarse, se obtienen diferentes matrices de Vandermonde para diferentes valores de $x_1,..,x_n$. En particular se pide:

a) Tomar $x_i = 10^{-i+1}$, con $i = 1,..,n$, y presentar los resultados obtenidos con $n = 4$.
b) Tomar $x_i = i$, con $i = 1,..,n$, y presentar los resultados obtenidos con $n = 4$. ●

7.2 Sistemas con solución inmediata: programación

7.2.1 Matriz diagonal

Como ya se ha visto anteriormente, la matriz $\boldsymbol{A}$ se escribe como:

$$\boldsymbol{A} = \boldsymbol{D} = \begin{pmatrix} d_{11} & 0 & \cdots & \cdots & 0 \\ 0 & d_{22} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & d_{nn} \end{pmatrix}$$

y la solución se obtiene directamente

$$x_i = \frac{b_i}{d_{ii}} \qquad\qquad i = 1, \ldots, n$$

Por lo tanto, es muy sencillo realizar la siguiente subrutina para matrices diagonales.

```
c_____________Solucion de Sistemas Lineales con matriz DIAGONAL
      subroutine solve_diag(ndim, dmat, b, x)
      dimension dmat(ndim), b(ndim), x(ndim)

      do 10 i=1,ndim
        x(i) = b(i)/dmat(i)
   10 continue

      return
      end
```

Conviene resaltar algunas cuestiones: en primer lugar esta rutina presupone que la matriz $\boldsymbol{D}$ que le llega es regular (la rutina no comprueba que cada `dmat(i)` sea no nulo), porque esta verificación puede resultar cara (una sentencia lógica para cada componente) y normalmente la regularidad de $\boldsymbol{D}$ es conocida a priori; a pesar de ello, esta verificación es absolutamente necesaria si no se sabe con certeza que $\boldsymbol{D}$ es regular. En segundo lugar, esta rutina se ha diseñado para que, a gusto del programador, los resultados (el vector $\boldsymbol{x}$) se guarden en un vector distinto o no del vector $\boldsymbol{b}$. En función de lo que se desee, la sentencia que *llama* a (hace que se ejecuten la instrucciones de) `SOLVE_DIAG` es:

```
      call solve_diag(ndim, dmat, b, x)
```

si se han dimensionados los dos vectores $\boldsymbol{b}$ y $\boldsymbol{x}$, y además se desean guardar ambos. O bien,

```
      call solve_diag(ndim, dmat, b, b)
```

si sólo $\boldsymbol{b}$ ha sido dimensionado y se puede escribir el resultado, $\boldsymbol{x}$, de resolver el sistema sobre el término independiente. Se pierde la información original de $\boldsymbol{b}$ pero hay un ahorro de memoria (un vector de dimensión `ndim`).

7.2.2 Matriz triangular inferior

En este caso, la estructura de la matriz del sistema es

$$\boldsymbol{A} = \boldsymbol{L} = \begin{pmatrix} l_{11} & 0 & \cdots & \cdots & 0 \\ l_{21} & l_{22} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ l_{n1} & \cdots & \cdots & l_{n,n-1} & l_{nn} \end{pmatrix}$$

y el algoritmo de resolución que se denomina *sustitución hacia adelante* es (ver subapartado 6.2.2)

$$\begin{aligned} x_1 &= b_1 \;/\; l_{11} \\ x_i &= (b_i - \sum_{j=1}^{i-1} l_{ij} x_j) \;/\; l_{ii} \qquad i = 2, \ldots, n \end{aligned}$$

Una subrutina que permite aplicar este algoritmo es la siguiente:

```
c_____________Solucion de Sistemas Lineales con matriz TRIANGULAR INFERIOR
      subroutine solve_tl(ndim, tlmat, b, x)
      dimension tlmat(ndim,ndim), b(ndim), x(ndim)

      x(1) = b(1) / tlmat(1,1)
      do 10 i=2,ndim
        x(i) = b(i)
        do 20 j=1,i-1
          x(i) = x(i) - tlmat(i,j)*x(j)
   20   continue
        x(i) = x(i) / tlmat(i,i)
   10 continue

      return
      end
```

Conviene observar que en esta rutina se han anidado los bucles `DO` de manera natural siguiendo el algoritmo expuesto: para cada fila i se suman las columnas en j. Sin embargo, también podría plantearse el algoritmo con un bucle primero en las columnas, en j, y luego por filas i. Es decir, la sustitución hacia adelante puede ser reprogramada de la siguiente forma:

```
c_____________Solucion de Sistemas Lineales con matriz TRIANGULAR INFERIOR
      subroutine solve_tl(ndim, tlmat, b, x)
      dimension tlmat(ndim,ndim), b(ndim), x(ndim)

      do 10 i=1,ndim
        x(i) = b(i)
   10 continue

      do 20 j=1,ndim-1
        x(j) = x(j) / tlmat(j,j)
        do 30 i=j+1,ndim
          x(i) = x(i) - tlmat(i,j)*x(j)
   30   continue
   20 continue
      x(ndim) = x(ndim) / tlmat(ndim,ndim)

      return
      end
```

En el apartado 7.4 quedarán claras las implicaciones prácticas que pueden representar ambas formas de implementar este algoritmo. A pesar de todo es importante observar que en ambos casos esta previsto almacenar toda la matriz $\boldsymbol{L}$. Es decir, se almacenan todos los ceros de la triangular superior. Este despilfarro de memoria es innecesario y debe ser corregido, como se verá más adelante.

Finalmente, se presenta una variante del bucle `DO` muy conveniente para escribir matrices. Una rutina que sólo escriba los términos no nulos de la matriz `tlmat(ndim,ndim)` podría ser la siguiente:

```
c_____________Escritura de una matriz TRIANGULAR INFERIOR
      subroutine write_tl(ndim, tlmat)
      dimension tlmat(ndim,ndim)

      do 30 i=1,ndim
        write(6,1000) (tlmat(i,j), j=1,i)
   30 continue

 1000 format(11(1pe12.6))

      return
      end
```

7.3 Consideraciones sobre la memoria

7.3.1 Tipos de memoria

Como ya se vio en el primer tema, "Introducción al uso de los ordenadores", existen, desde el punto de vista del *hardware*, diversos tipos de memoria. En particular, interesa recordar la memoria RAM y la memoria *caché*. La primera porque es donde residen los datos, los programas que ejecuta el usuario y parte del sistema operativo. Y la segunda porque es la que almacena los datos antes de que los utilice la CPU.

Puesto que la memoria RAM es finita, existe, en principio, una limitación clara al número de datos que como máximo se pueden manipular simultáneamente en un ordenador. Por ejemplo, si se desea resolver un sistema lineal de ecuaciones con $n = 5000$ incógnitas y se almacenan todos los coeficientes de la matriz, $n^2 = 25 \cdot 10^6$, como reales de ocho bytes, `REAL*8`, sería necesario disponer de $2 \cdot 10^8$bytes, es decir (dividiendo por 1024^2) de 191Mbytes, sólo para almacenar la matriz. Es evidente que este número, que no tiene en cuenta ni otros vectores, ni las instrucciones del programa, ni el sistema operativo, excede con creces la memoria disponible en la gran mayoría de los ordenadores.

Por suerte, en los años sesenta se desarrolló una aportación fundamental en ciencias de la computación: la *memoria virtual*. La idea es sencilla pero su implementación es complicada. La memoria que el usuario tiene a su disposición no coincide con la memoria RAM del ordenador. El usuario dispone de una cierta cantidad de *memoria virtual*. Esta memoria está dividida en bloques de tamaño relativamente modestos llamados *páginas*. Puesto que la memoria virtual es mayor que la memoria RAM, la mayoría de las páginas de memoria virtual se almacenan en dispositivos alternativos, normalmente discos. Sólo unas pocas páginas de la memoria virtual se encuentran *activas* en RAM. Los dispositivos alternativos son mucho más lentos que la memoria RAM pero permiten aumentar considerablemente las capacidades de memoria direccionable por el usuario.

Cuando una instrucción de un programa referencia una cierta posición de la memoria, existen dos posibilidades:

1. Que la página que contiene esa posición de memoria se encuentre en RAM (*un acierto*). En este caso, se accede a esta posición de memoria inmediatamente.
2. Que la página que contiene esa posición de memoria *no* se encuentre en RAM (*un fallo*). En este caso, el sistema selecciona una de las páginas activas y la cambia por la página de memoria que contenga la información deseada.

Cada *fallo* es caro, puesto que implica la selección de la página adecuada, su búsqueda en el disco, su lectura y finalmente el intercambio con la página activa en RAM. Además, en sistemas multiusuario todos estos procesos implican interacciones entre la CPU y los dispositivos alternativos de almacenamiento (discos) que se ven retrasadas por las acciones de los demás usuarios. Es decir, conviene evitar, en la medida de lo posible, los fallos. Normalmente, las posiciones de memoria cercanas tienen grandes posibilidades de pertenecer a la misma página. Por consiguiente, los programadores deben procurar, para mejorar la eficiencia de sus códigos, que los datos que se vayan a emplear consecutivamente estén en posiciones de memoria lo más próximas posible.

El intercambio de información que se produce entre la memoria RAM y el disco también existe entre la memoria *caché* y la memoria RAM. Puesto que por la memoria *caché* pasa la

información que va a ser empleada en la CPU, también se pueden definir los aciertos (cuando la siguiente posición de memoria deseada ya se encuentra en *caché*) y los fallos (cuando la posición de memoria deseada no está en memoria *caché*).

En resumen, el concepto de *proximidad* de la información es fundamental para evitar sobrecostos en el manejo de la memoria.

7.3.2 Dimensionamiento dinámico

Si se recuerdan las primeras sentencias de los programas que realizan el cálculo del producto de una matriz por un vector (programas 7.1 y 7.2):

```
...
implicit real*8 (a-h,o-z)
parameter (na_max = 100)
...
dimension a(na_max,na_max), v(na_max), w(na_max)
...
```

se puede observar que para no tener que recompilar y *linkar* el programa cada vez que se desea trabajar con matrices de tamaños distintos, se ha definido un parámetro `na_max`. Este parámetro indica el tamaño máximo admisible sin necesidad de modificar el programa. En este caso, se supone que no van a analizarse matrices de orden superior a 100. Para ello se reserva en memoria espacio suficiente para almacenar la matriz `a` de 100×100 coeficientes reales de ocho bytes, posteriormente se reserva el espacio, que como máximo ocupará el vector `v` (100 `REAL*8` más), y finalmente otro tanto para el vector `w`.

Después de reservar el espacio que como máximo puede necesitarse, el programa requiere el orden que *en realidad* va a ser empleado:

```
...
write (6,*) ' Entra el orden de la matriz: '
read (5,*) ndim
...
```

Si, por ejemplo, se introduce para `ndim` el valor 4, el programa sólo utiliza las primeras 16 posiciones de memoria de la matriz `a`. El vector `v`, de 4 componentes, se encontrará almacenado detrás de la matriz `a`. Es decir, hay $100^2 - 16$ números reales de ocho bytes, que no van a ser usados. Entre los coeficientes de `a` y las componentes de `v` existen, por lo tanto, un gran número de posiciones de memoria. Es obvio que en este caso no se verifica la condición de proximidad entre posiciones de memoria (ver figura 7.1).

Para evitar este problema se emplea el *dimensionamiento dinámico*. El objetivo es almacenar las matrices y los vectores consecutivamente dejando las posiciones de memoria inutilizadas al final del espacio reservado (que es el mismo que antes).

Los programas 7.1 y 7.2 habían reservado un espacio equivalente a: $100^2+100+100 = 10\,200$ reales de ocho bytes. Ahora se reservarán las mismas posiciones de memoria (más una por comodidad):

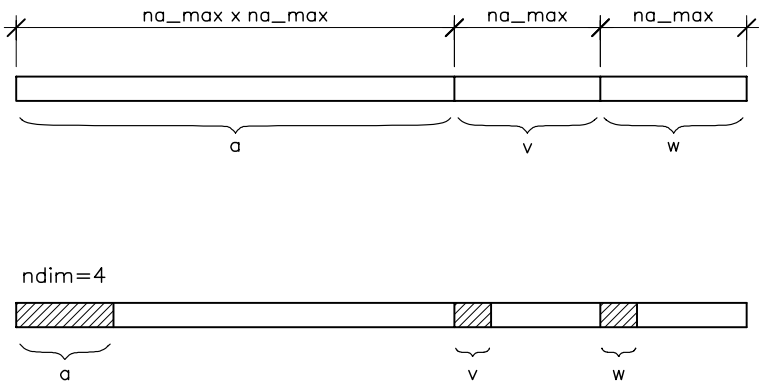


Fig. 7.1 Reserva no consecutiva de espacio de memoria

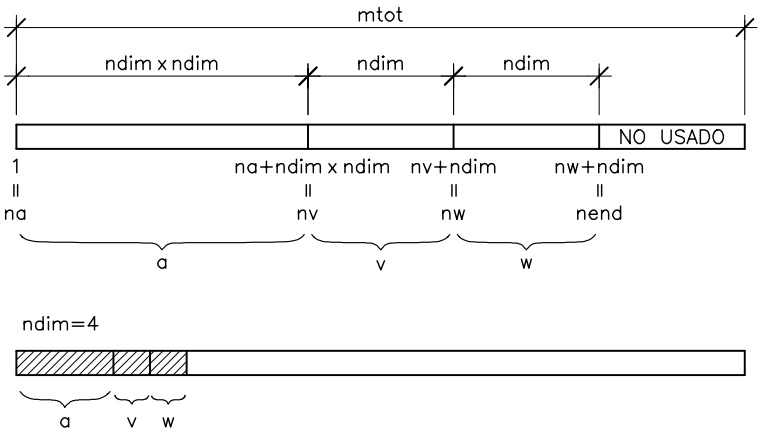


Fig. 7.2 Reserva consecutiva de espacio de memoria

```
...
implicit real*8 (a-h,o-z)
parameter (mtot = 10201)
...
dimension dd(mtot)
write (6,*) ' Entra el orden de la matriz: '
read (5,*) ndim
...
```

pero en un único vector denominado `dd` que debe contener la matriz `a` y los vectores `v` y `w`. La matriz `a` ocupará las primeras `ndim`×`ndim` posiciones de `dd`, el vector `v` las siguientes `ndim`, y por último el vector `w` utilizará las `ndim` siguientes. Quedan libres, por lo tanto, las últimas `mtot - ndim`^{2} `- ndim - ndim` posiciones del vector `dd`; si `ndim`= 4 quedan libres 10 177 consecutivas

y al final de `dd`. Con esto se ha conseguido que los coeficientes de `a`, las componentes de `v` y las componentes de `w` estén *próximas*, independientemente de `ndim` (ver figura 7.2).

Queda, sin embargo, el poder enviar a las funciones o rutinas la información adecuada: la matriz y los vectores por separado. Para ello es necesario saber dónde empieza cada matriz y vector. Esto se realiza por medio de *punteros*: `na` será el puntero de la matriz `a` (donde empieza la matriz `a` en `dd`), `nv` será el puntero de `v`, y `nw` el de `w`. Obsérvese que todos ellos son enteros. Para definir los punteros se realizan las siguientes instrucciones:

```
      ...
c___Definicion de punteros
      na   = 1              ! matriz a (que ocupa ndim*ndim posiciones)
      nv   = na + ndim*ndim ! vector v despues de a (y ocupa ndim posiciones)
      nw   = nv + ndim      ! vector w despues de v
      nend = nw + ndim
c___Verificacion de espacio necesario
      if (nend .gt. mtot) then
        write(5,*) '   ERROR >>> Dimensionamiento Insuficiente !'
        write(5,*) '             se requieren',nend,' posiciones'
        stop
      endif
      ...
```

Por último, es imprescindible establecer la comunicación con las funciones y rutinas: es necesario que se les transmitan la matriz y los vectores por separado. Para ello, es conveniente imaginar que el argumento asociado a una matriz, o a un vector, sólo transmite a la rutina la posición del primer elemento de esta matriz, o vector. Posteriormente, dentro de la rutina y con la instrucción `DIMENSION`, se reconoce el espacio que necesita esta matriz, o vector. Así, por ejemplo,

```
      ...
      call get_mat_vec (ndim,a,v)
      ...
```

es equivalente a

```
      ...
      call get_mat_vec (ndim,a(1,1),v(1))
```

puesto que ambas pasan la posición del mismo elemento de la matriz `a`. Es en realidad en la rutina, cuando se escribe

```
      ...
      subroutine get_mat_vec (n,a,v)
      implicit real*8 (a-h,o-z)
      dimension a(n,n),v(n)
      ...
```

donde se le da el carácter de matriz de dimensión `ndim` a la posición de memoria transmitida.

Gracias a esta propiedad, en el caso de *dimensionamiento dinámico*, sólo es necesario transmitir la posición del primer elemento correspondiente a la matriz o vector deseado. Por ejemplo:

```
      ...
      call get_mat_vec (ndim,dd(na),dd(nv))
      ...
```

A continuación se muestra el programa principal para el cálculo del producto matriz por vector modificado según el dimensionamiento dinámico.

```
c
c        Este programa calcula el producto de una matriz por un
c        vector y el modulo del vector mediante funciones y
c        rutinas utilizando dimensionamiento dinamico
c-----------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      parameter (mtot = 10201)
      dimension dd(mtot)

      write (6,*) ' Entra el orden de la matriz: '
      read (5,*) ndim

c___Definicion de punteros
      na   = 1
      nv   = na + ndim*ndim
      nw   = nv + ndim
      nend = nw + ndim

c___Verificacion de espacio necesario
      if (nend .gt. mtot) then
        write(5,*) '   ERROR >>> Dimensionamiento Insuficiente !'
        write(5,*) '             se requieren',nend,' posiciones'
        stop
      endif

c___Lectura o generacion de la matriz y del vector
      call get_mat_vec (ndim,dd(na),dd(nv))

c___Calculo del producto de matriz por vector
      call product (ndim,dd(na),dd(nv),dd(nw))

c___Calculo del modulo de un vector
      x=xmodulo(ndim,dd(nw))

c___Escritura de los resultados
      call write_resul(ndim,dd(na),dd(nv),dd(nw),x)

      stop
      end
```

Prog. 7.3 Programa principal mediante dimensionamiento dinámico

7.4 Almacenamiento de matrices

7.4.1 Almacenamiento por defecto en FORTRAN

En FORTRAN las variables con dos subíndices se almacenan por columnas. Por ejemplo, en la matriz

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

se almacenan los coeficientes siguiendo el orden que imponen las columnas, es decir:

$$a_{11} \quad a_{21} \quad a_{31} \quad a_{41} \quad a_{12} \quad a_{22} \quad a_{32} \quad a_{42} \quad a_{13} \quad a_{23} \quad a_{33} \quad a_{43} \quad a_{14} \quad a_{24} \quad a_{34} \quad a_{44}$$

En el caso de que la variable tenga más de dos subíndices, el almacenamiento sigue la misma filosofía. Por ejemplo, una variable dimensionada como `c(ni,nj,nk)`, tendrá en primer lugar los elementos `c(1,1,1)`, `c(2,1,1)`, ... , `c(ni,1,1)`, después vendrán los `c(1,2,1)`, `c(2,2,1)`, ... , `c(ni,2,1)`, hasta `c(1,nj,1)`, `c(2,nj,1)`, ... , `c(ni,nj,1)`, para seguir con `c(1,1,2)`, `c(2,1,2)`, ... , `c(ni,1,2)`, después `c(1,2,2)`, `c(2,2,2)`, ... , `c(ni,2,2)`, hasta `c(1,nj,2)`, `c(2,nj,2)`, ... , `c(ni,nj,2)`, hasta llegar finalmente a los últimos elementos: `c(1,nj,nk)`, `c(2,nj,nk)`, ... , `c(ni,nj,nk)`.

Es importante tener en cuenta el hecho de que el lenguaje FORTRAN almacena las matrices por columnas para el diseño de los algoritmos. Para ilustrar este punto se van a analizar los algoritmos propuestos en el apartado 7.2.2 para resolver sistemas con matrices triangulares inferiores. En este caso, se particulariza, para matrices de orden 4. Es decir,

$$\boldsymbol{L} = \begin{pmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{pmatrix}$$

Por consiguiente, esta matriz se almacena en el ordenador como

$$l_{11} \quad l_{21} \quad l_{31} \quad l_{41} \quad 0 \quad l_{22} \quad l_{32} \quad l_{42} \quad 0 \quad 0 \quad l_{33} \quad l_{43} \quad 0 \quad 0 \quad 0 \quad l_{44}$$

Ahora, si se observan en detalle los bucles del primer algoritmo propuesto se comprueba que el orden en que se accede a los coeficientes de $\boldsymbol{L}$ es

1	2	4	7		3	5	8			6	9				10
l_{11}	l_{21}	l_{31}	l_{41}	0	l_{22}	l_{32}	l_{42}	0	0	l_{33}	l_{43}	0	0	0	l_{44}

Como puede verse, los accesos a los elementos van saltando de un sitio a otro. Es evidente que en este caso no se accede a los elementos de $\boldsymbol{L}$ ni de forma secuencial ni con proximidad. Este efecto de saltar entre posiciones lejanas de la memoria se ve acentuado al aumentar el orden de

la matriz. En cambio, si se estudia el otro algoritmo propuesto, es fácil comprobar que a los elementos de $\boldsymbol{L}$ se accede como sigue:

$$\begin{array}{cccccccccccccccc} 1 & 2 & 3 & 4 & & 5 & 6 & 7 & & & 8 & 9 & & & & 10 \\ l_{11} & l_{21} & l_{31} & l_{41} & 0 & l_{22} & l_{32} & l_{42} & 0 & 0 & l_{33} & l_{43} & 0 & 0 & 0 & l_{44} \end{array}$$

Parece obvio que de esta forma el acceso a los coeficientes es más secuencial: se progresa a lo largo de las columnas de $\boldsymbol{L}$ siguiendo el almacenamiento natural del FORTRAN.

A pesar de todo el almacenamiento de los ceros es claramente un despilfarro de memoria y además separa elementos que se necesitan de forma consecutiva. Más adelante se analiza cómo evitar este almacenamiento innecesario.

Por último conviene resaltar que el almacenamiento por columnas que por defecto se tiene en FORTRAN no tiene por qué reproducirse con otros lenguajes de programación. Por ejemplo, el lenguaje `C` almacena por defecto las matrices por filas. En ese caso, el primer algoritmo propuesto resulta más eficiente.

7.4.2 Almacenamiento por filas y por columnas

Hasta ahora se ha estudiado el almacenamiento que de manera natural proporciona el FORTRAN, pero el usuario puede, si lo desea, modificarlo según su interés. A continuación se presentan las dos maneras de almacenamiento de matrices llenas en un vector: por columnas y por filas.

Almacenamiento por columnas

En algunas ocasiones (cuando, por ejemplo, se mezclan rutinas en `C` y en FORTRAN) puede ocurrir que el programador desee efectuar el almacenamiento de una matriz en un vector. En primer lugar se estudia el caso en que se desee realizar un almacenamiento por columnas. Es decir que la matriz $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, que es una matriz rectangular de m filas y n columnas,

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1,n-1} & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2,n-1} & a_{2n} \\ \vdots & \vdots & & \vdots & \vdots \\ a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} & a_{m-1,n} \\ a_{m1} & a_{m2} & \cdots & a_{m,n-1} & a_{mn} \end{pmatrix}$$

se desea almacenar en un vector $\boldsymbol{c} \in \mathbb{R}^{n \cdot m}$ introduciendo consecutivamente las columnas de $\boldsymbol{A}$. Es decir

$$\begin{aligned} \boldsymbol{c}^T &= (c_1\ ,\ c_2\ ,\ldots,\ c_m\ ,c_{m+1}\,,c_{m+2}\,,\ldots,c_{2\cdot m}\,,\ldots,\ c_k\ ,\ldots,c_{(n-1)\cdot m+1}\,,c_{(n-1)\cdot m+2}\,,\ldots,c_{n\cdot m}) \\ &= (a_{11}\,,a_{21}\,,\ldots,a_{m1}\,,\ a_{12}\ ,\ a_{22}\ ,\ldots,a_{m2}\,,\ldots,a_{ij}\,,\ldots,\quad a_{1n}\quad,\quad a_{2n}\quad,\ldots,a_{mn}) \end{aligned}$$

Como puede verse, en la componente c_k de $\boldsymbol{c}$ se almacena el elemento a_{ij} de $\boldsymbol{A}$. La posición k se determina a partir de i y de j como

$$k = (j-1) \cdot m + i$$

En el caso, poco probable, de que dada la posición k en $\boldsymbol{c}$ se quiera conocer a qué fila i y columna j de $\boldsymbol{A}$ se corresponde, es necesario ejecutar el siguiente algoritmo:

```
i = mod(k,m)
j = (k-i)/m + 1
if (i.eq.0) then
  i = m
  j = j - 1
endif
```

Almacenamiento por filas

Si por el contrario el programador desea efectuar un almacenamiento por filas, entonces la matriz $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ se guarda en un vector $\boldsymbol{f} \in \mathbb{R}^{m\cdot n}$ introduciendo consecutivamente las filas de $\boldsymbol{A}$. Es decir,

$$\begin{aligned}\boldsymbol{f}^T &= (f_1\ ,\ f_2\ ,\ldots,\ f_n\ ,f_{n+1}\,,f_{n+2}\,,\ldots,f_{2\cdot n}\,,\ldots,\ f_k\ ,\ldots,f_{(m-1)\cdot n+1}\,,f_{(m-1)\cdot n+2}\,,\ldots,f_{m\cdot n})\\ &=(a_{11}\,,a_{12}\,,\ldots,a_{1n}\,,\ a_{21}\ ,\ a_{22}\ ,\ldots,a_{2n}\,,\ldots,a_{ij}\,,\ldots,\quad a_{m1}\quad ,\quad a_{m2}\quad ,\ldots,a_{mn})\end{aligned}$$

Ahora, el elemento a_{ij} se almacena en la componente f_k de $\boldsymbol{f}$ y la posición k se determina a partir de i y de j como

$$k = (i-1)\cdot n + j$$

mientras que para recuperar i y j a partir de k el algoritmo es simplemente:

```
j = mod(k,n)
i = (k-i)/n + 1
if (j.eq.0) then
  j = n
  i = i - 1
endif
```

Para ver un ejemplo de lo expuesto anteriormente, se estudia el producto $\boldsymbol{A}\,\boldsymbol{B} = \boldsymbol{C}$ siendo $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times l}$ y, obviamente, $\boldsymbol{C} \in \mathbb{R}^{m\times l}$. Los elementos de $\boldsymbol{C}$ se obtienen a partir de

$$c_{ij} = \sum_{k=1}^{n} a_{ik}\ b_{kj} \qquad i = 1,\ldots,m \quad j = 1,\ldots,l$$

Si $\boldsymbol{A}$ y $\boldsymbol{C}$ se almacenan por filas y $\boldsymbol{B}$ se almacena por columnas, el algoritmo para efectuar el producto es el siguiente:

```
...
subroutine AporBenC(m,n,l,fa,cb,fc)
dimension fa(m*n), cb(n*l), fc(m*l)
...
do i = 1, m
  iapos = (i-1)*n
  icpos = (i-1)*l
  do j = 1, l
    jbpos = (j-1)*n
    kcpos = icpos + j
    fc( kcpos ) = dot( n, fa(iapos+1), cb(jbpos+1) )
  enddo
enddo
...
```

7.4.3 Matrices simétricas o matrices triangulares

Si se desea almacenar matrices simétricas basta conservar los elementos de la triangular inferior o de la triangular superior. Por lo tanto los esquemas que se exponen a continuación son válidos tanto para matrices simétricas como para matrices triangulares.

MATRIZ TRIANGULAR SUPERIOR

Sea la matriz $\boldsymbol{U}$ que se muestra a continuación

$$\boldsymbol{U} = \begin{array}{c} \\ \\ \\ \\ i \\ \\ \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & & & i & & & j & & \end{array} \\ \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1i} & \cdots & u_{1j} & \cdots & & u_{1n} \\ 0 & u_{22} & & \vdots & & \vdots & & & \vdots \\ \vdots & \ddots & \ddots & & & & & & \\ 0 & & \ddots & u_{ii} & \cdots & u_{ij} & \cdots & & u_{in} \\ & & & \ddots & \ddots & & & & \\ \vdots & & & & \ddots & \vdots & & & \vdots \\ & & & & \ddots & \ddots & & & \\ 0 & & & & & \ddots & u_{jj} & \cdots & u_{jn} \\ & & & & & & \ddots & \ddots & \vdots \\ \vdots & & & & & & & \ddots & u_{n-1,n} \\ 0 & \cdots & & 0 & \cdots & 0 & \cdots & 0 & u_{nn} \end{pmatrix} \end{array}$$

En este caso es usual el almacenamiento por columnas de esta matriz. Para evitar el almacenamiento de los ceros, el vector que se define es

$$\begin{aligned}\boldsymbol{c}^T = &(c_1\ ,\ c_2\ ,\ c_3\ ,\ c_4\ ,\ c_5\ ,\ c_6\ ,\ c_7\ ,\ldots,\quad\ ,\ldots,c_k\ ,\ldots,\quad\ ,\quad\ ,\ldots,c_{\frac{n(n+1)}{2}})\\ =&(u_{11}\,,u_{12}\,,u_{22}\,,u_{13}\,,u_{23}\,,u_{33}\,,u_{14}\,,\ldots,u_{44}\,,\ldots,u_{ij}\,,\ldots,u_{1n}\,,u_{2n}\,,\ldots,\quad u_{nn})\end{aligned}$$

y la ecuación que determina k a partir de i y de j es

$$\begin{aligned}k &= 1+2+3+4+\ldots+(j-1)+i\\ &= \frac{j(j-1)}{2}+i\end{aligned}$$

Obviamente, esta fórmula sólo tiene sentido para los elementos no nulos de $\boldsymbol{U}$, es decir para $i \leq j$.

El algoritmo de recuperación (dado k deducir i y j) puede escribirse como

```
i = 1
j = 0
do while ( i.gt.j )
  j = j+1
  i = k-j*(j-1)/2
enddo
```

7.4.4 Matrices en banda

Se dice que una matriz $\boldsymbol{B}$ es una matriz en banda con

1. *Semiancho inferior* l si todos los elementos de $\boldsymbol{B}$ por debajo de la subdiagonal l son nulos, es decir, $b_{ij} = 0$ si $i - j > l$.
2. *Semiancho superior* u si todos los elementos de $\boldsymbol{B}$ por encima de la subdiagonal u son nulos. Es decir, $b_{ij} = 0$ si $j - i > u$.

Esta matriz puede esquematizarse como sigue

$$\boldsymbol{B} = \begin{pmatrix} b_{11} & b_{12} & \ldots & b_{1,1+u} & & & & & \\ b_{21} & b_{22} & & & \ddots & & & & \\ \vdots & & \ddots & & & \ddots & & & \\ b_{1+l,1} & & & \ddots & & & \ddots & & \\ & \ddots & & & \ddots & & & b_{n-u,n} & \\ & & \ddots & & & \ddots & & \vdots & \\ & & & \ddots & & & b_{n-1,n-1} & b_{n-1,n} & \\ & & & & b_{n,n-l} & \cdots & b_{n,n-1} & b_{nn} & \end{pmatrix}$$

En este caso lo primero que se hace es una transformación de la matriz anterior a una matriz

rectangular $\boldsymbol{R}$ con n filas y $l+1+u$ columnas

$$\boldsymbol{R} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1,l+u} & r_{1,l+1+u} \\ r_{21} & r_{22} & \cdots & r_{2,l+u} & r_{2,l+1+u} \\ \vdots & \vdots & & \vdots & \vdots \\ r_{n-1,1} & r_{n-1,2} & \cdots & r_{n-1,l+u} & r_{n-1,l+1+u} \\ r_{n1} & r_{n2} & \cdots & r_{n,l+u} & r_{n,l+1+u} \end{pmatrix}$$

Los elementos de $\boldsymbol{B}$ quedan en esta matriz como

$$\boldsymbol{R} = \begin{pmatrix} & & & b_{11} & \cdots & \cdots & b_{1,1+u} \\ & & b_{21} & b_{22} & \cdots & \cdots & b_{2,2+u} \\ & \iddots & & \vdots & & & \vdots \\ b_{1+l,1} & \cdots & \cdots & b_{1+l,1+l} & \cdots & \cdots & b_{1+l,1+l+u} \\ \vdots & & & \vdots & & & \vdots \\ \vdots & & & \vdots & & & b_{n-u,n} \\ \vdots & & & \vdots & & \iddots & \\ \vdots & & & b_{n-1,n-1} & b_{n-1,n} & & \\ b_{n,n-l} & \cdots & b_{n,n-1} & b_{nn} & & & \end{pmatrix}$$

Conviene observar que este esquema de almacenamiento será ventajoso en la medida que se verifique: $n^2 \gg n \cdot (l+1+u)$, es decir, $n \gg (l+1+u)$. En realidad se almacenan $\frac{(l+1)l}{2} + \frac{(u+1)u}{2}$ elementos que a priori se sabe que son nulos, pero no resulta fácil evitar considerarlos sin complicar en exceso los algoritmos de almacenamiento. Además resulta fácil demostrar que para $l \ll n$ y $u \ll n$ el número de ceros innecesariamente almacenados en $\boldsymbol{R}$ es muy inferior a los que quedan fuera de la banda de $\boldsymbol{B}$, $\frac{(n-l-1)(n-l)}{2} + \frac{(n-u-1)(n-u)}{2}$.

Por último, se plantean las fórmulas de almacenamiento: dado un elemento genérico b_{ij} de $\boldsymbol{B}$, este elemento se almacena en $r_{\alpha\beta}$ de $\boldsymbol{R}$, y los subíndices α y β se evalúan como

$$\begin{aligned} \alpha &= i \\ \beta &= 1 + l + (j - i) \end{aligned}$$

En este caso, la recuperación de i y de j es también trivial

$$\begin{aligned} i &= \alpha \\ j &= (\alpha + \beta) - (1 + l) \end{aligned}$$

En la práctica, la matriz rectangular $\boldsymbol{R}$ se almacena por filas o por columnas. En el primer caso se tendrá almacenada $\boldsymbol{B}$ *por filas* y el el segundo *por diagonales*.

Por ejemplo, si $\boldsymbol{R}$ se almacena por columnas ($\boldsymbol{B}$ por diagonales) el algoritmo de almacenamiento que dados los subíndices i y j de un elemento de la banda de $\boldsymbol{B}$, indica la componente k en un vector, es

$$\begin{aligned} k &= (\beta - 1) \cdot n + \alpha \\ &= [l + (j - i)] \cdot n + i \end{aligned}$$

Problema 7.2:
Se desea encontrar la solución del sistema lineal de ecuaciones $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, donde $\boldsymbol{A}$ es una matriz de orden n, simétrica, en banda y con coeficientes:

$$\begin{aligned}
a_{11} &= 5 \\
a_{ii} &= 6 \qquad i = 2,..,n-1 \\
a_{nn} &= 5 \\
a_{ij} &= -4 \qquad i = 1,..,n \qquad j = \max(1, i-1), \min(n, i+1) \\
a_{ij} &= 1 \qquad i = 1,..,n \qquad j = \max(1, i-2), \min(n, i+2)
\end{aligned}$$

y $\boldsymbol{b}$ es un vector con coeficientes:

$$b_i = \frac{h^4}{EI}\left(p_a + \frac{p_b - p_a}{b-a} h\ i\right) \qquad i = 1,..,n$$

donde E, I, a, b, p_a, p_b son constantes del problema y $h = (b-a)/(n+1)$. Se pide:

a) Escribir un programa en FORTRAN que utilice dimensionamiento dinámico, que trate la matriz $\boldsymbol{A}$ como *matriz simétrica en banda* con $u = l = 2$, que tenga una estructura modular, y que conste de las siguientes subrutinas:
- Lectura de datos de un fichero (n, a, b, E, I, p_a, p_b).
- Definición de punteros.
- Generación de $\boldsymbol{A}$ y $\boldsymbol{b}$.
- Resolución del sistema $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ mediante el método de Gauss adaptado al esquema de almacenamiento óptimo definido para $\boldsymbol{A}$.
- Escritura del resultado $\boldsymbol{x}$ en un fichero.

b) Resolver el sistema $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ para los datos $a = 0$, $b = 1$, $p_a = 1$, $p_b = 1$, $E = 10^5$, $I = 10^{-5}$ y
1. $n = 5$
2. $n = 19$
3. $n = 99$

c) Importar los resultados del apartado b desde Excel y generar, para cada valor de n, una lista de pares ordenados $\{t_i, x_i\}$ con $i = 0,..,n+1$, según las siguientes relaciones:

$$\begin{aligned}
t_i &= a + \frac{b-a}{n+1}\ i \qquad i = 0,..,n+1 \\
x_0 &= 0 \\
x_{n+1} &= 0
\end{aligned}$$

y x_i, con $i = 1,..,n$, iguales a los resultados obtenidos a partir del programa. Presentar en un solo gráfico de Excel las tres series de pares ordenados $\{t_i, x_i\}$.

●

Problema 7.3:
Comparar el coste computacional (número de operaciones y memoria necesaria) de la resolución mediante el método de Gauss de un sistema de ecuaciones de

orden n, $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, en el que la matriz $\boldsymbol{A}$ es simétrica y en banda con $u = l = 2$ (véase el problema 7.2), considerando las dos alternativas siguientes:

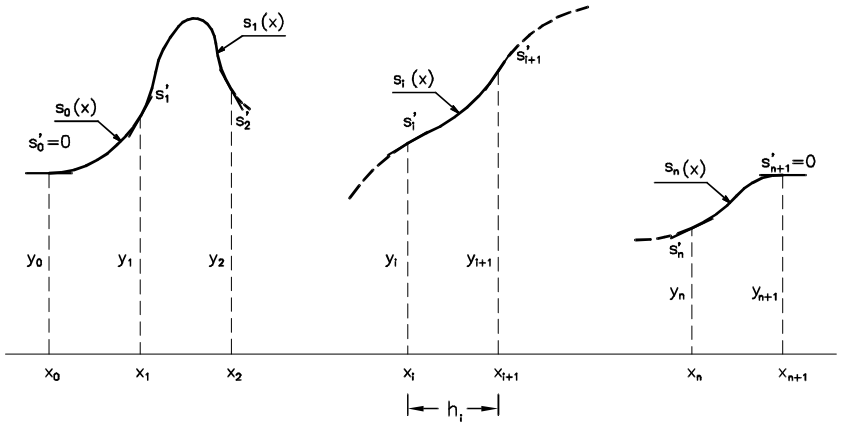
a) Se trata la matriz $\boldsymbol{A}$ como *matriz llena* y se utiliza el método de Gauss estándar.

b) Se almacena la matriz $\boldsymbol{A}$ de forma óptima y se utiliza el método de Gauss adaptado al esquema de almacenamiento.

Concretar los resultados obtenidos con n igual a 10, 100 y 1000. ●

Problema 7.4:

Un ingeniero está diseñando el trazado en alzado de una montaña rusa para un nuevo parque temático. Los datos de diseño son $n+2$ puntos (x_i, y_i), con $i = 0, \ldots, n+1$, que corresponden a $n+2$ puntos de apoyo de la vía (véase la figura).



Para definir el trazado de la vía, el ingeniero decide utilizar una cúbica en cada uno de los $n+1$ tramos $[x_i, x_{i+1}]$ con $i = 0, \ldots, n$. Cada cúbica puede expresarse como

$$s_i(x) = \left[h_i\left(s'_i + s'_{i+1}\right) - 2t_i\right]\left(\frac{x - x_i}{h_i}\right)^3 + \left[3t_i - h_i\left(s'_{i+1} + 2s'_i\right)\right]\left(\frac{x - x_i}{h_i}\right)^2 + s'_i\left(x - x_i\right) + y_i$$

donde $h_i = x_{i+1} - x_i$, $t_i = y_{i+1} - y_i$ y s'_i es la pendiente en el apoyo i-ésimo. Las únicas pendientes conocidas a priori son las de los dos apoyos extremos: $s'_0 = s'_{n+1} = 0$. Las pendientes de los apoyos interiores se calculan imponiendo la continuidad de la segunda derivada (curvatura) en dichos apoyos. Para cumplir esta condición, deben verificarse las siguientes relaciones entre las pendientes (para $i = 1, \ldots, n$):

$$\frac{h_i}{h_i + h_{i-1}} s'_{i-1} + 2s'_i + \frac{h_{i-1}}{h_i + h_{i-1}} s'_{i+1} = 3\frac{h_i}{h_i + h_{i-1}}\left(\frac{h_i}{h_{i-1}} t_{i-1} + \frac{h_{i-1}}{h_i} t_i\right)$$

Estas n ecuaciones pueden escribirse como un sistema lineal de ecuaciones tridiagonal (es decir, en banda y con semianchos $l = u = 1$) de dimensión n. Se pide:

a) Justificar razonadamente que el sistema lineal puede resolverse mediante los métodos de Gauss o de Crout *sin necesidad de pivotar*.

La particularización del método de Crout a matrices tridiagonales se denomina método de Thomas. Puesto que no hay que pivotar, puede emplearse un esquema de almacenamiento para matrices en banda (véase el subapartado 7.4.4).

b) Escribir el pseudocódigo del método de Thomas empleando una matriz rectangular de n filas y 3 columnas, que contiene la matriz tridiagonal al principio y los factores $\boldsymbol{L}$ y $\boldsymbol{U}$ al final.

c) ¿En qué orden se accede a los coeficientes $r_{\alpha\beta}$ de la matriz $\boldsymbol{R}$? Justificar razonadamente que, teniendo en cuenta la paginación, sería preferible trabajar con una matriz rectangular de 3 filas y de n columnas (en lugar de n filas y 3 columnas).

d) Escribir un programa en FORTRAN para resolver el sistema lineal tridiagonal dado mediante el método de Thomas. El programa debe *1)* emplear una matriz rectangular de 3 filas y n columnas, *2)* tener estructura modular (subrutinas), *3)* leer los datos (n y los puntos (x_i, y_i), con $i = 0, \ldots, n+1$) de un archivo de datos, y *4)* escribir los resultados (las pendientes $s'_1, s'_2, \ldots, s'_n$ en los apoyos interiores) en un archivo de resultados.

e) Utilizar el programa para resolver el caso con $n = 7$, $x_0 = 0$, $x_1 = 5$, $x_2 = 15$, $x_3 = 25$, $x_4 = 40$, $x_5 = 48$, $x_6 = 55$, $x_7 = 63$, $x_8 = 73$, $y_0 = 0$, $y_1 = 0.5$, $y_2 = 6$, $y_3 = -1$, $y_4 = 2$, $y_5 = 1.75$, $y_6 = 4$, $y_7 = 0.5$, $y_8 = 0$. Dibujar el trazado de la vía en una gráfica de Excel, teniendo en cuenta que en cada tramo la función es una cúbica distinta. Comentar los resultados obtenidos.

●

7.4.5 Almacenamiento en *skyline*

Algunos de los sistemas lineales de ecuaciones que se obtienen al resolver numéricamente problemas de ingeniería se caracterizan por presentar un ancho de banda muy importante pero con muchos elementos nulos en su interior. A modo de ejemplo, en la figura 7.3 se muestra una matriz de orden 7000 donde se han marcado en negro los términos no nulos. Como puede observarse, el semiancho de banda sería muy grande (es de 1737) y aun cuando se utilizara un almacenamiento en banda, se estarían guardando excesivos elementos (en realidad se almacenarían $(1737 + 1737 + 1) \times 7000 = 24\ 325\ 000$ elementos).

El almacenamiento en *skyline* de una matriz se realiza:

1. En la parte triangular superior por columnas. Se guardan únicamente todos los elementos comprendidos entre el primer término no nulo y el término de la diagonal.
2. En la parte triangular inferior por filas. Se guardan únicamente todos los elementos comprendidos entre el primer término no nulo y el término de la diagonal.

Por ejemplo, en la figura 7.4 se han marcado en negro todos los elementos que se deben guardar cuando la matriz de la figura 7.3 se almacena en *skyline* (el número total de elementos almacenados es 1 114 757, lo que representa un ahorro muy importante).

Como es usual, si la matriz es simétrica sólo se deben almacenar los elementos de la triangular superior o inferior. En adelante, tan sólo se presentará el almacenamiento en *skyline* para

matrices triangulares superiores. Su extensión a matrices no simétricas se deja como ejercicio para el lector.

Una de las propiedades más importantes de este tipo de almacenamiento consiste en que los métodos directos de resolución de sistemas lineales de ecuaciones conservan este tipo de esquemas. Esto hace que el almacenamiento en *skyline* sea ampliamente utilizado.

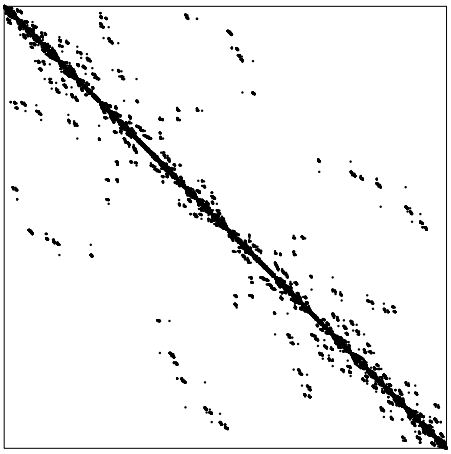


Fig. 7.3 Elementos no nulos de una matriz

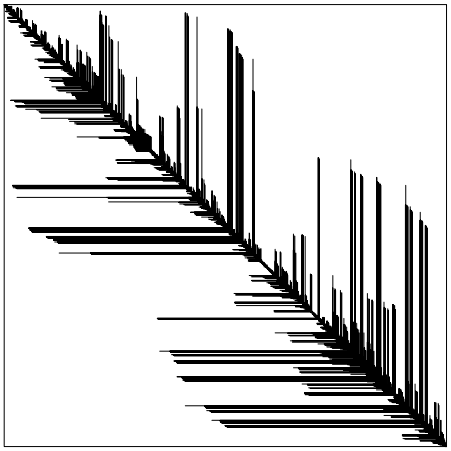


Fig. 7.4 Almacenamiento en skyline *de la matriz mostrada en la figura 7.4*

El almacenamiento de la matriz se realiza mediante dos vectores. El primero (vector $\boldsymbol{c}$) contiene los elementos de la matriz. Ésta se guarda por columnas, y para cada de ellas (empezando por la primera) se almacena desde el primer elemento no nulo hasta la diagonal. El

segundo (vector $\boldsymbol{l}$) es un vector de punteros de n componentes, donde n es el orden de la matriz del sistema. En la i-ésima componente de este vector se almacena la posición que ocupa el elemento de la diagonal a_{ii} en el vector $\boldsymbol{c}$. Por ejemplo, si la matriz del sistema es

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & & a_{13} & & a_{15} & & & \\ 0 & a_{22} & a_{23} & & a_{25} & & & \\ \vdots & \ddots & a_{33} & a_{34} & a_{35} & & & \\ \vdots & & \ddots & a_{44} & a_{45} & & a_{47} & \\ \vdots & & & \ddots & a_{55} & & a_{57} & \\ \vdots & & & & \ddots & a_{66} & a_{67} & \\ \vdots & & & & & \ddots & a_{77} & \\ \vdots & & & & & & \ddots & \ddots \end{pmatrix}$$

el vector $\boldsymbol{c}$ contiene los coeficientes

$$\boldsymbol{c}^T = (a_{11}\,, a_{22}\,, a_{13}\,, a_{23}\,, a_{33}\,, a_{34}\,, a_{44}\,, a_{15}\,, a_{25}\,, a_{35}\,, a_{45}\,, a_{55}\,, a_{66}\,, a_{47}\,, a_{57}\,, a_{67}\,, a_{77}\,, \ldots)$$

mientras que los siete primeros elementos del vector de punteros, $\boldsymbol{l}$, valen

$$\boldsymbol{l}^T = (1\,, 2\,, 5\,, 7\,, 12\,, 13\,, 17\,, \ldots \qquad\qquad)$$

De las definiciones anteriores se desprende que el número de elementos que se deben almacenar en la columna $j+1$ es

$$l_{j+1} - l_j$$

Dado un elemento genérico de la matriz a_{ij}, éste se almacena en una posición k del vector $\boldsymbol{c}$ que viene dada por

$$k \;=\; l_j \;-\; (j - i)$$

Por el contrario, el algoritmo de recuperación (conocido k encontrar i y j) puede expresarse como

```
      do 10 j=2,n
        if (l(j).ge.k) goto 10
      enddo
10    continue
      i= j-(l(j)-k)
```

Problema 7.5:
Sea $\boldsymbol{A}$ una matriz simétrica y definida positiva, almacenada en *skyline* (únicamente la triangular superior, gracias a la simetría de $\boldsymbol{A}$). El objetivo de este problema es adaptar el método de Cholesky a este esquema de almacenamiento. Para ello, se pide:

a) Comprobar, a partir del pseudocódigo del método de Cholesky dado en la ecuación 6.41, que la matriz triangular superior $\boldsymbol{L}^T$ tiene el mismo *skyline* que la parte superior de la matriz $\boldsymbol{A}$ (en consecuencia, $\boldsymbol{L}^T$ puede almacenarse sobre $\boldsymbol{A}$, empleando el mismo esquema de almacenamiento en *skyline).*

b) Escribir el pseudocódigo del método de Cholesky adaptado a matrices en *skyline.* ●

7.4.6 Almacenamiento compacto

Una matriz $\boldsymbol{A}$ se denomina *vacía* si la mayoría de sus elementos son cero. Puede ocurrir, además, que la disposición de los elementos no nulos desaconseje el empleo de los esquemas de almacenamiento vistos hasta ahora. Si, por ejemplo, la matriz $\boldsymbol{A}$ tiene elementos no nulos muy alejados de la diagonal, el ancho de banda puede ser muy parecido al orden n de la matriz y el esquema de almacenamiento en banda no permite ahorrar espacio de memoria.

En estos casos puede utilizarse un esquema de *almacenamiento compacto*, que consiste en guardar únicamente los elementos de $\boldsymbol{A}$ distintos de cero.

Almacenamiento comprimido por filas

El *almacenamiento comprimido por filas* es un método general para guardar matrices vacías, que no hace ninguna hipótesis sobre la distribución de los elementos no nulos. Sea τ el número de coeficientes no nulos de $\boldsymbol{A}$. La idea es guardar estos elementos en un vector $\boldsymbol{f}$ de τ componentes, recorriendo la matriz $\boldsymbol{A}$ por filas. Así, por ejemplo, para la matriz

$$\boldsymbol{A} = \begin{pmatrix} 8 & 0 & 0 & 3 & 0 \\ 2 & 7 & 0 & 0 & 1 \\ 0 & 1 & 9 & 0 & 6 \\ 0 & 0 & -4 & 5 & 0 \\ 0 & 4 & 0 & 0 & -6 \end{pmatrix} \qquad \tau = 12$$

el vector $\boldsymbol{f}$ es

$$\boldsymbol{f}^T = (8, 3, 2, 7, 1, 1, 9, 6, -4, 5, 4, -6)$$

Desde luego, el vector $\boldsymbol{f}$ por sí solo no basta para conocer la matriz $\boldsymbol{A}$. Hace falta, además, conocer la posición de las componentes de $\boldsymbol{f}$ en la matriz $\boldsymbol{A}$. Para ello, se utilizan dos vectores más: $\boldsymbol{m}$ (de τ componentes) y $\boldsymbol{l}$ (de $n+1$ componentes). En $\boldsymbol{m}$ se almacenan los índices de columna j de los elementos del vector $\boldsymbol{f}$. Es decir, si $f_k = a_{ij}$, entonces $m_k = j$. El vector $\boldsymbol{l}$ contiene punteros que indican la posición en $\boldsymbol{f}$ del primer elemento de cada fila. Es decir, si $f_k = a_{ij}$, entonces $l_i \leq k < l_{i+1}$. Por comodidad, se define $l_{n+1} = \tau + 1$ (de esta forma, la expresión también es válida para la última fila).

Para la matriz $\boldsymbol{A}$ del ejemplo, el vector $\boldsymbol{m}$ es

$$\boldsymbol{m}^T = (1, 4, 1, 2, 5, 2, 3, 5, 3, 4, 2, 5)$$

y el vector $\boldsymbol{l}$ es

$$\boldsymbol{l}^T = (1, 3, 6, 9, 11, 13)$$

Por ejemplo, para $k = 9$ se obtiene $f_9 = -4$, $m_9 = 3$ y $l_4 \leq 9 < l_5$. Esto significa que el coeficiente de la fila 4 y la columna 3 de la matriz $\boldsymbol{A}$ vale -4.

Así pues, la matriz $\boldsymbol{A}$ se almacena mediante tres vectores: uno de números reales (en un caso general), $\boldsymbol{f}$, y dos de números enteros, $\boldsymbol{m}$ y $\boldsymbol{l}$.

Es muy importante observar que los esquemas de almacenamiento compacto (como el que se acaba de presentar) *no* pueden emplearse en la resolución de un sistema lineal $\boldsymbol{Ax} = \boldsymbol{b}$ mediante métodos directos. Si el algoritmo de eliminación gaussiana se aplica a la matriz $\boldsymbol{A}$ del ejemplo, el coeficiente $a_{24}^{(0)} = 0$ se transforma en $a_{24}^{(1)} \neq 0$ después del primer paso (compruébese). Pero en los vectores $\boldsymbol{f}$, $\boldsymbol{m}$ y $\boldsymbol{l}$ no hay espacio para $a_{24}^{(1)}$, puesto que se han guardado únicamente los elementos no nulos de la matriz $\boldsymbol{A}$ *original*.

PRODUCTO DE MATRIZ POR VECTOR

En cambio, los esquemas de almacenamiento compacto son muy indicados si es necesario emplear la matriz $\boldsymbol{A}$ como un operador lineal (es decir, para efectuar productos de matriz por vector). Para ilustrarlo, se describe a continuación el producto de una matriz vacía por un vector mediante un algoritmo adaptado al almacenamiento comprimido por filas. Sea $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, es decir,

$$y_i = \sum_{j=1}^{n} a_{ij} x_j \qquad i = 1, \ldots, n$$

Dado que la matriz $\boldsymbol{A}$ es vacía, basta con efectuar los productos $a_{ij}x_j$ con $a_{ij} \neq 0$. Mediante los vectores $\boldsymbol{f}$, $\boldsymbol{m}$ y $\boldsymbol{l}$, esto puede hacerse con las siguientes instrucciones FORTRAN:

```
do i=1,n
  y(i) = 0.d0
  do j=l(i),l(i+1)-1
    y(i) = y(i) + f(j)*x(m(j))
  enddo
enddo
```

Nótese que, con la ayuda del vector $\boldsymbol{l}$, en el bucle `DO`—`ENDDO` interior (en j) se recorren *solamente* los elementos no nulos de la fila i (almacenados en el vector $\boldsymbol{f}$), y se multiplican por las componentes correspondientes del vector $\boldsymbol{x}$ (detectadas mediante el vector $\boldsymbol{m}$).

Problema 7.6:
El *almacenamiento comprimido por columnas* es un esquema para matrices vacías. Igual que en el almacenamiento comprimido por filas, una matriz $\boldsymbol{A}$ de orden n y τ elementos no nulos se guarda en tres vectores: 1) un vector $\boldsymbol{c}$, de τ componentes, que contiene los coeficientes no nulos de $\boldsymbol{A}$ por columnas; 2) un vector $\boldsymbol{m}$, de τ componentes, con los índices de fila de las componentes del vector $\boldsymbol{c}$; 3) un vector $\boldsymbol{l}$, de $n+1$ componentes, que contiene punteros que indican la posición en $\boldsymbol{c}$ del primer elemento de cada columna de $\boldsymbol{A}$. Se pide:

a) ¿Cuáles son los vectores $\boldsymbol{c}$, $\boldsymbol{m}$ y $\boldsymbol{l}$ para la matriz con $n = 5$ y $\tau = 12$ empleada para ilustrar el almacenamiento comprimido por filas?

b) Detallar el algoritmo de multiplicación de una matriz $\boldsymbol{A}$ guardada según un almacenamiento comprimido por columnas y un vector $\boldsymbol{x}$, $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. ●

7.5 Bibliografía

BORSE G. J. *Programación FORTRAN77 con aplicaciones de cálculo numérico en ciencias e ingeniería.* Anaya, 1989.

BREUER, S.; ZWAS, G. *Numerical Mathematics. A Laboratory Approach.* Cambridge University Press, 1993.

ELLIS T.M.R. *FORTRAN 77 Programming.* Addison–Wesley Publishing Company, 1990.

GOLUB, G.H.; VAN LOAN, C.F. *Matrix Computations.* Segunda edición, The John Hopkins University Press, 1990.

STEWART G.W. *Afternotes on Numerical Analysis.* Society for Industrial Applied Mathematics, 1996.

8 Aplicaciones al cálculo integral

Objetivos

- Describir dos técnicas numéricas para calcular integrales definidas $\int_a^b f(x)\mathrm{d}x$: el método de las aproximaciones rectangulares y la regla compuesta del trapecio.
- Estudiar y comparar los dos métodos mediante algunos ejemplos numéricos.
- Presentar la extensión para el cálculo de volúmenes.

8.1 Introducción

En muchos problemas de ingeniería interesa calcular la integral definida de la función $f(x)$ en el intervalo $[a, b]$,

$$I = \int_a^b f(x)\mathrm{d}x \tag{8.1}$$

Esta integral puede interpretarse como el área de la región limitada por la curva $y = f(x)$ y las rectas $y = 0$, $x = a$ e $x = b$ (véase la figura 8.1).

De manera más rigurosa, la integral I puede definirse a partir de las *aproximaciones rectangulares* (superior e inferior). Para ello, se divide el intervalo $[a, b]$ en n subintervalos iguales, de longitud $h = \frac{b-a}{n}$, mediante los puntos $x_0 = a$, $x_1 = a + h$, $x_2 = a + 2h$, ..., $x_i = a + ih$, ..., $x_n = b$ (véase la figura 8.2). A continuación se construyen los rectángulos "superior" e "inferior" para cada subintervalo $[x_i, x_{i+1}]$.

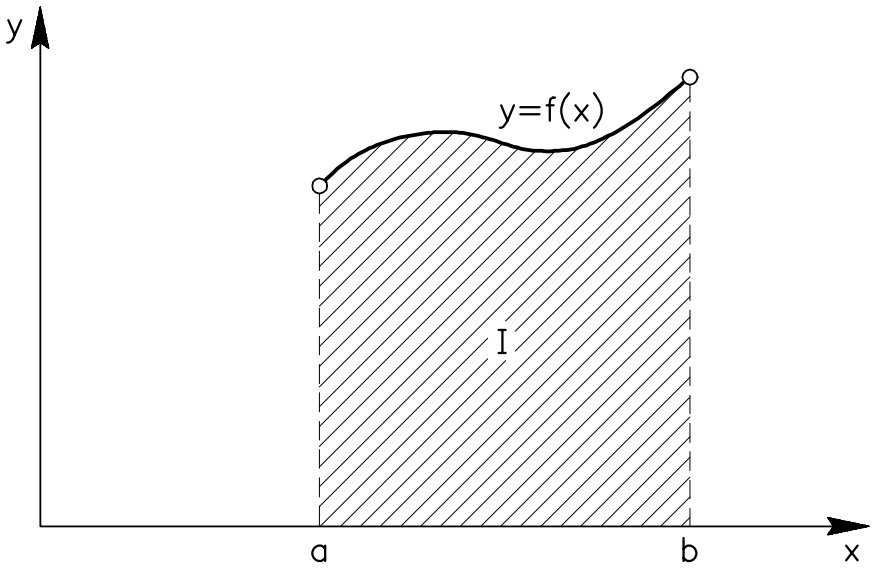


Fig. 8.1 Interpretación gráfica de la integral definida

Supóngase, para fijar ideas, que la función $f(x)$ es *creciente* en el intervalo $[a, b]$, tal como ocurre en la figura 8.2. En ese caso, la altura del rectángulo inferior es $f(x_i)$ (extremo izquierdo) y la altura del rectángulo superior es $f(x_{i+1})$ (extremo derecho).

La *aproximación rectangular inferior* se define como la suma de las áreas de todos los rectángulos inferiores

$$I_{\inf}(h) = hf(x_0) + hf(x_1) + \ldots + hf(x_{n-1}) = h\sum_{i=0}^{n-1} f(x_i) \tag{8.2}$$

y, análogamente, la *aproximación rectangular superior* es la suma de las áreas de todos los rectángulos superiores

$$I_{\sup}(h) = hf(x_1) + hf(x_2) + \ldots + hf(x_n) = h\sum_{i=0}^{n-1} f(x_{i+1}) \tag{8.3}$$

En la figura 8.2 se ha supuesto que la función f es creciente. Si la función f fuera decreciente en lugar de creciente, la altura del rectángulo inferior sería $f(x_{i+1})$ (extremo derecho) y la altura del rectángulo superior sería $f(x_i)$ (extremo izquierdo). En consecuencia, es necesario intercambiar las definiciones de $I_{\inf}(h)$ e $I_{\sup}(h)$ dadas en las ecuaciones 8.2 y 8.3, y tomar

$$I_{\inf}(h) = h\sum_{i=0}^{n-1} f(x_{i+1}) \qquad ; \qquad I_{\sup}(h) = h\sum_{i=0}^{n-1} f(x_i)$$

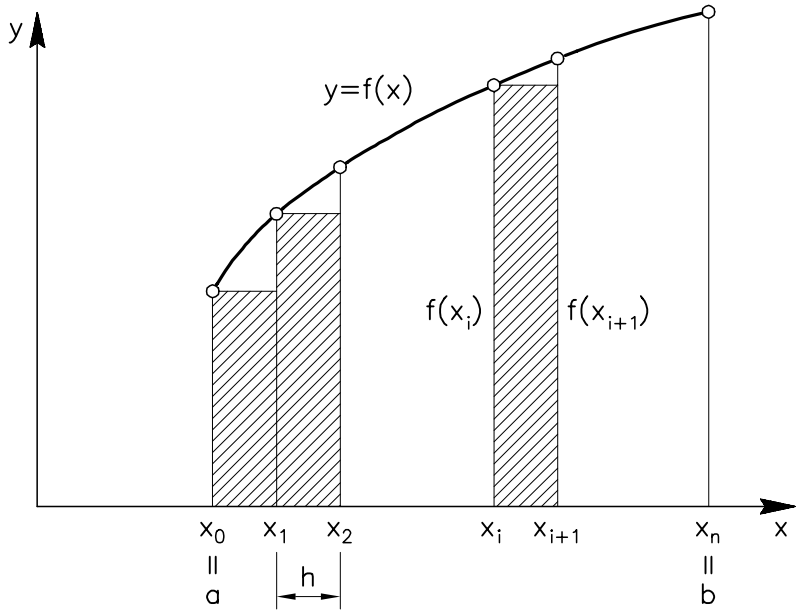


Fig. 8.2a Aproximación rectangular inferior

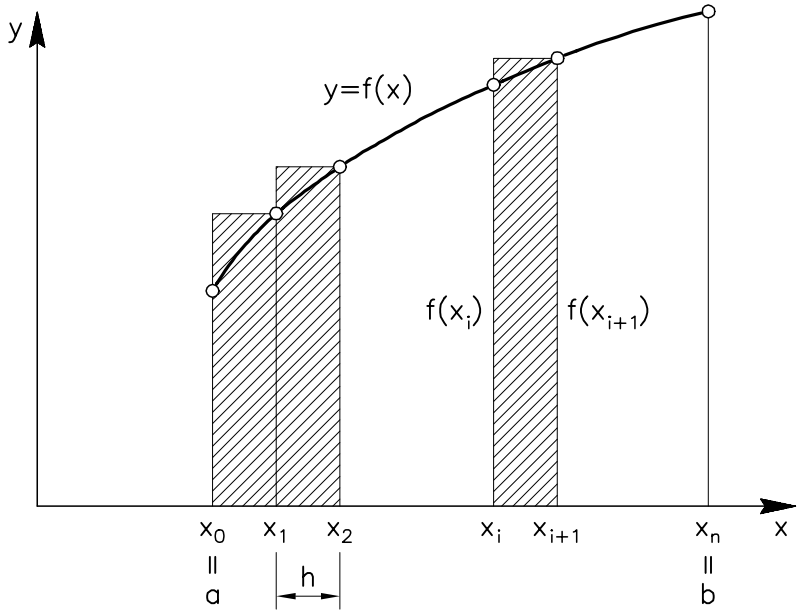


Fig. 8.2b Aproximación rectangular superior

Por último, si la función f no fuera monótona, sería necesario determinar para cada subintervalo $[x_i, x_{i+1}]$ cuál es la altura del rectángulo inferior y cuál la del rectángulo superior, para modificar adecuadamente las expresiones de $I_{\inf}(h)$ e $I_{\sup}(h)$.

A partir de las ecuaciones 8.2 y 8.3, la integral definida I se define como el límite de las aproximaciones rectangulares superior e inferior para h $\to 0$

$$I = \lim_{h\to 0} I_{\inf}(h) = \lim_{h\to 0} I_{\sup}(h)$$

Nótese que $I_{\inf}(h)$ e $I_{\sup}(h)$ son, de hecho, las sumas inferior y superior que se utilizan en la teoría de integración de Riemann (asociadas a la partición del intervalo $[a, b]$ que se muestra en la figura 8.2).

8.2 El método de las aproximaciones rectangulares

En algunos casos, la integral I de la ecuación 8.1 puede calcularse analíticamente, obteniendo una primitiva de la función f y evaluándola en los extremos del intervalo, a y b. En otros casos de interés práctico, sin embargo, es imposible (o muy farragoso) obtener una primitiva de f. Se hace entonces necesario emplear una *técnica numérica* para el cálculo de la integral I.

Una primera posibilidad es trabajar directamente con las aproximaciones rectangulares empleadas en el apartado anterior para la definición de I. En lugar de tomar el límite para $h \to 0$, se trabaja con h finito y se calcula $I_{\inf}(h)$ e $I_{\sup}(h)$, a partir de las ecuaciones 8.2 y 8.3. Para cualquier valor de h, el valor exacto de la integral, I, se halla comprendido entre $I_{\inf}(h)$ e $I_{\sup}(h)$:

$$I_{\inf}(h) < I < I_{\sup}(h)$$

Además, a medida que se toma h cada vez más pequeño, las aproximaciones $I_{\inf}(h)$ e $I_{\sup}(h)$ se parecen cada vez más, y así se consigue "atrapar" el valor exacto I en un intervalo $[I_{\inf}(h), I_{\sup}(h)]$ tan pequeño como se quiera.

De hecho, la diferencia entre las aproximaciones rectangulares superior e inferior es una *cota del error absoluto* cometido al aproximar la integral I. Teniendo en cuenta las ecuaciones 8.2 y 8.3, puede escribirse

$$E < I_{\sup}(h) - I_{\inf}(h) = h\Big(f(x_n) - f(x_0)\Big) = h\Big(f(b) - f(a)\Big) \tag{8.4}$$

La ecuación 8.4 indica que el método de las aproximaciones rectangulares es *lineal*, ya que el error E es $\mathcal{O}(h)$: a medida que h tiende a cero, el error tiende *linealmente* a cero.

Así pues, el método numérico de las aproximaciones rectangulares está íntimamente relacionado con la propia definición teórica de la integral de Riemann.

Como ejemplo de aplicación, se procede a calcular numéricamente la integral $\int_0^{\pi/2} \sin(x)\mathrm{d}x$. Desde luego, esta integral puede resolverse analíticamente sin ninguna dificultad:

$$\int_0^{\pi/2} \sin(x)\mathrm{d}x = -\cos(x)\Big|_0^{\pi/2} = 1 \tag{8.5}$$

Se trata únicamente de un problema sencillo que se emplea para ilustrar el funcionamiento del método. Se escribe un programa en FORTRAN que calcula esta integral (programa 8.1, apartado 8.5). En la tabla 8.1 se muestra el valor de las aproximaciones rectangulares inferior y superior para distintos valores de h.

Tabla 8.1 Cálculo de $\int_0^{\pi/2} \sin(x)\mathrm{d}x$ *por el método de las aproximaciones rectangulares*

n	h	I_{inf}	I_{sup}
1	1.57080D+00	0.0000000	1.5707963
2	7.85398D-01	0.5553604	1.3407585
5	3.14159D-01	0.8346821	1.1488414
10	1.57080D-01	0.9194032	1.0764828
100	1.57080D-02	0.9921255	1.0078334
1000	1.57080D-03	0.9992144	1.0007852
10000	1.57080D-04	0.9999215	1.0000785
100000	1.57080D-05	0.9999921	1.0000079
1000000	1.57080D-06	0.9999992	1.0000008
10000000	1.57080D-07	0.9999999	1.0000001

Tal y como estaba previsto, la columna I_{inf} de la tabla 8.1 tiende al valor exacto $I = 1$ desde abajo (es decir, con $I_{\mathrm{inf}} < 1$), mientras que la columna I_{sup} tiende a $I = 1$ desde arriba (con $I_{\mathrm{sup}} > 1$). Es importante notar que el error decrece muy lentamente a medida que n aumenta (trabajando con diez millones de subintervalos, se obtiene todavía un error relativo de 10^{-7}). Este fenómeno es debido a que el método de aproximaciones rectangulares es *lineal* (ecuación 8.4).

8.3 El método compuesto del trapecio

En el apartado anterior se han utilizado dos aproximaciones rectangulares (una inferior y una superior) del área comprendida entre $y = f(x)$, $y = 0$, $x = x_i$ y $x = x_{i+1}$. Intuitivamente, parece mejor utilizar como aproximación el área A_i del *trapecio* $PQRS$ (figura 8.3), que puede calcularse como

$$A_i = h\frac{f(x_i) + f(x_{i+1})}{2}$$

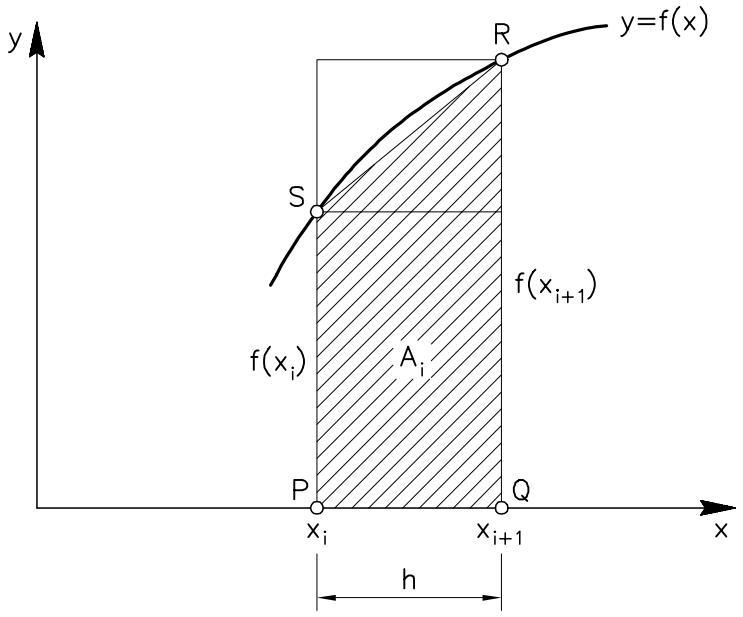


Fig. 8.3 Aproximación mediante un trapecio

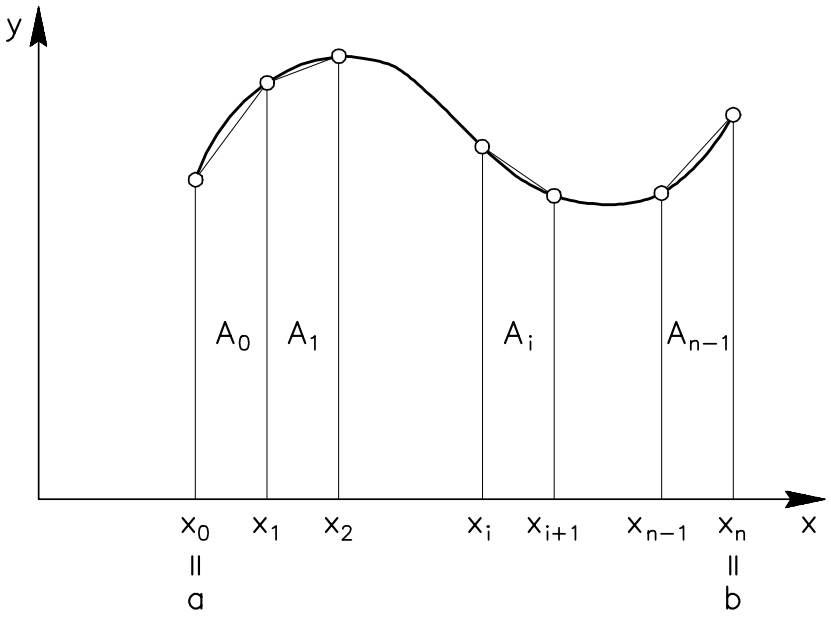


Fig. 8.4 El método compuesto del trapecio

El *método compuesto del trapecio* consiste en aproximar la integral I por la suma $I_{\mathrm{T}}(h)$ de las áreas de todos los trapecios (véase la figura 8.4):

$$I_{\mathrm{T}}(h) = A_0 + A_1 + A_2 + \ldots A_{n-1} = h\left[\frac{f(x_0)}{2} + f(x_1) + f(x_2) + \ldots + f(x_{n-1}) + \frac{f(x_n)}{2}\right] \quad (8.6)$$

En la figura 8.4 puede verse que este método permite tratar funciones no monótonas de manera directa, sin necesidad de distinguir entre los tramos crecientes y los tramos decrecientes.

Se puede demostrar que el error que se comete al aproximar la integral exacta I por I_{T} (ecuación 8.6) es $\mathcal{O}(h^2)$. Esto indica que el método compuesto del trapecio es *cuadrático*: a medida que h tiende a cero, el error tiende *cuadráticamente* a cero. Para ver cuál es la implicación práctica de este resultado teórico, se utiliza el método compuesto del trapecio para calcular numéricamente la integral 8.5. Para ello se emplea el programa 8.2 (apartado 8.5). La tabla 8.2 recoge el valor del $I_{\mathrm{T}}(h)$ para distintos valores de h.

Tabla 8.2 Cálculo de $\int_0^{\pi/2} \sin(x)\mathrm{d}x$ *por el método compuesto del trapecio*

n	h	I_{T}
1	1.57080D+00	0.7853982
2	7.85398D-01	0.9480594
5	3.14159D-01	0.9917618
10	1.57080D-01	0.9979430
100	1.57080D-02	0.9999794
1000	1.57080D-03	0.9999998
10000	1.57080D-04	1.0000000

Comparando las tablas 8.1 y 8.2 queda claro que el método compuesto del trapecio tiende al valor exacto $I = 1$ mucho más rápidamente que el método de las aproximaciones rectangulares. Para obtener ocho cifras correctas con el método del trapecio, por ejemplo, basta tomar diez mil subintervalos. Esta mayor rapidez se debe a que el método compuesto del trapecio es *cuadrático*, mientras que el método de las aproximaciones rectangulares es solamente *lineal*.

Problema 8.1: En este ejercicio se propone verificar experimentalmente el orden de convergencia de las dos técnicas numéricas presentadas para el cálculo de integrales definidas: el método de las aproximaciones rectangulares (convergencia lineal) y el método compuesto del trapecio (convergencia cuadrática).

a) Construir una tabla donde aparezca el error absoluto cometido al calcular numéricamente la integral definida $\int_0^{\pi/2} \sin(x)\mathrm{d}x$ mediante las dos técnicas mencionadas, para los valores de n que aparecen en las tablas 8.1 y 8.2.

b) Para cada una de las dos técnicas, representar el error absoluto E (calculado a partir del valor exacto de la integral y de los resultados del apartado anterior) en función del número de subintervalos, n. Emplear una escala log-log (logaritmo de E versus logaritmo de n).

c) Obtener una expresión teórica general (es decir, independiente de la integral definida que se está calculando) de la relación entre el logaritmo de E y el logaritmo de n en los casos

1. Método lineal: $E = \mathcal{O}(h) = \mathcal{O}(\frac{1}{n})$
2. Método cuadrático: $E = \mathcal{O}(h^2) = \mathcal{O}(\frac{1}{n^2})$

donde h es el tamaño de los subintervalos.

d) ¿Concuerda la expresión teórica del apartado c con las relaciones obtenidas en el apartado b? Razonar adecuadamente la respuesta. ●

Problema 8.2: Un ingeniero está proyectando una carretera, y necesita calcular el volumen de movimiento de tierras en el tramo comprendido entre los puntos kilométricos 1730 y 1810. Dispone para ello de perfiles transversales cada cinco metros. En cada perfil se han medido con un planímetro las áreas de desmonte A_{D} y terraplén A_{T} (véase la tabla adjunta).

Áreas de desmonte y terraplén. Perfiles transversales cada 5 m

Punto kilométrico (m)	Área desmonte (m^2)	Área terraplén (m^2)
1730	2.51	0.05
1735	1.32	0.61
1740	1.12	0.82
1745	0.85	0.95
1750	0.63	1.21
1755	0.05	1.35
1760	0.00	1.56
1765	0.00	2.58
1770	0.00	2.41
1775	0.25	2.21
1780	0.56	1.90
1785	0.85	1.50
1790	0.94	0.85
1795	1.57	0.34
1800	1.83	0.11
1805	2.61	0.00
1810	2.57	0.20

A partir de estos datos, los volúmenes de desmonte V_{D} y terraplén V_{T} pueden calcularse como

$$V_{D} = \int_{1730}^{1810} A_{D}(x)\mathrm{d}x \qquad ; \qquad V_{T} = \int_{1730}^{1810} A_{T}(x)\mathrm{d}x$$

Escribir un programa en FORTRAN que calcule, mediante el método compuesto del trapecio: 1) el volumen de desmonte V_D; 2) el volumen de terraplén V_T; 3) el balance de tierras (diferencia entre ambos volúmenes). Los datos deben leerse de un fichero de datos, y los resultados deben escribirse por pantalla y en un fichero de resultados. ●

8.4 Extensión al cálculo de volúmenes

En este apartado se generaliza el método compuesto del trapecio para el cálculo de integrales dobles del tipo

$$I = \int_a^b \int_c^d f(x,y)\mathrm{d}x\mathrm{d}y \tag{8.7}$$

Esta integral puede interpretarse como el volumen de la región limitada por la superficie $z = f(x,y)$ y los planos $z = 0$, $x = a$, $x = b$, $y = c$ e $y = d$ (véase la figura 8.5).

La integral 8.7 puede reescribirse como

$$I = \int_a^b \left[\int_c^d f(x,y)\mathrm{d}y \right] \mathrm{d}x = \int_a^b g(x)\mathrm{d}x \tag{8.8}$$

con

$$g(x) = \int_c^d f(x,y)\mathrm{d}y \tag{8.9}$$

A la vista de las ecuaciones 8.8 y 8.9, puede calcularse numéricamente la integral I aplicando *dos veces* el método compuesto del trapecio: una en dirección y y otra en dirección x. Se utiliza para ello una cuadrícula de nm rectángulos, con pasos $h_x = \frac{b-a}{n}$ en dirección x y $h_y = \frac{d-c}{m}$ en dirección y (véase la figura 8.6). Los nodos de esta cuadrícula son puntos (x_i, y_j), con $x_i = a + ih_x$ e $y_j = c + jh_y$.

Para un valor de x fijo ($x = x_i$), se calcula $g(x_i)$ (ecuación 8.9), empleando el método compuesto del trapecio en dirección y:

$$g(x_i) = h_y \left[\frac{f(x_i,y_0)}{2} + f(x_i,y_1) + f(x_i,y_2) + \ldots + f(x_i,y_{m-1}) + \frac{f(x_i,y_m)}{2} \right] \tag{8.10}$$

Una vez se ha calculado $g(x_i)$ para $i = 0,\ldots,n$ según la fórmula 8.10, se emplea el método compuesto del trapecio en dirección x para evaluar $I = \int_a^b g(x)\mathrm{d}x$:

$$I = \int_a^b g(x)\mathrm{d}x = h_x \left[\frac{g(x_0)}{2} + g(x_1) + g(x_2) + \ldots + g(x_{n-1}) + \frac{g(x_n)}{2} \right]$$

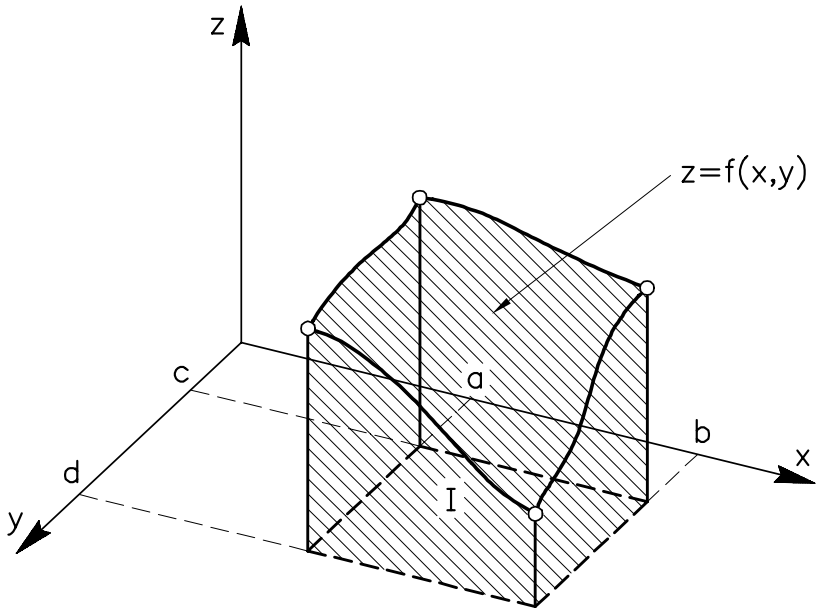


Fig. 8.5 Interpretación gráfica de la integral doble

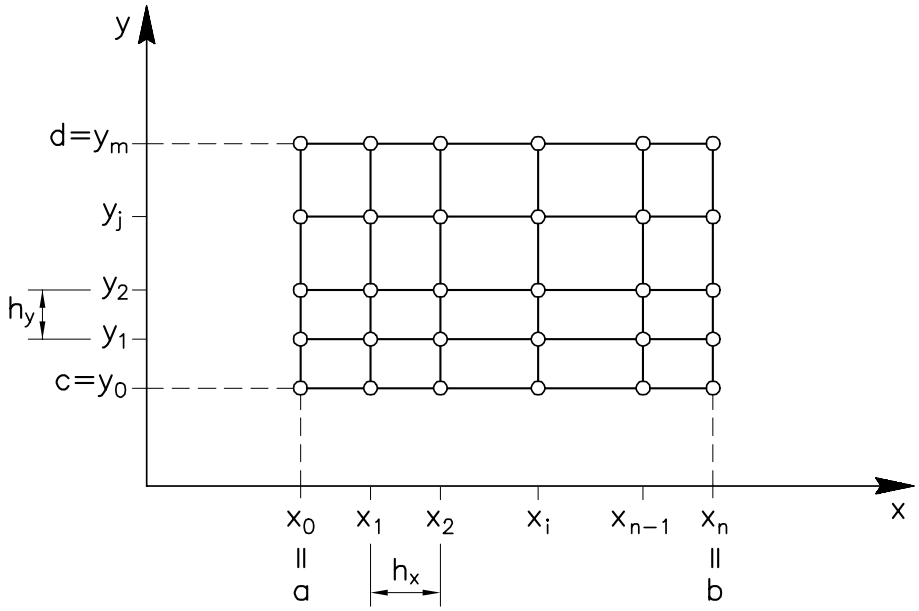


Fig. 8.6 Cuadrícula para el método compuesto del trapecio

8.5 Apéndice

```
c
c        Este programa calcula la integral de sin(x) entre 0 y pi/2
c        por el metodo de las APROXIMACIONES RECTANGULARES
c_____________________________________________________________________
      implicit real*8 (a-h,o-z)
      pi = 4.d0*atan(1.d0)

c___Numero n de subintervalos
      write (6,100)
      read  (5,*) n
 100  format (1x,'n= ')

c___Valor de h
      h = 0.5d0*pi/dble(n)

c___Calculo de las aproximaciones inferior y superior
      a_sup = 0.d0
      x = 0.d0

c___Extremo izquierdo a=0
      a_inf = dsin(x)

c___Puntos interiores
      do 10 i = 1,n-1
        x = x+h
        a_inf = a_inf + dsin(x)
        a_sup = a_sup + dsin(x)
  10  continue

c___Extremo derecho b=pi/2
      a_sup = a_sup + dsin(0.5d0*pi)

c___Factor comun h
      a_inf = h*a_inf
      a_sup = h*a_sup

c___Salida de resultados
      write (6,200) n, h , a_inf , a_sup
 200  format (1x,i8,1x,1pd12.5,1x,0pf10.7,1x,0pf10.7)
      stop
      end
```

Prog. 8.1 Método de las aproximaciones rectangulares

```
c
c        Este programa calcula la integral de sin(x) entre 0 y pi/2
c        por el METODO COMPUESTO DEL TRAPECIO
c
c_____________________________________________________________________
      implicit real*8 (a-h,o-z)
      pi = 4.d0*atan(1.d0)

c___Numero n de subintervalos
      write (6,100)
      read  (5,*) n
 100  format (1x,'n= ')

c___Valor de h
      h = 0.5d0*pi/dble(n)

c___Calculo de la aproximacion a_t
      x   = 0.d0

c___Extremo izquierdo a=0
      a_t = 0.5d0*dsin(x)

c___Puntos interiores
      do 10 i = 1,n-1
        x = x+h
        a_t = a_t + dsin(x)
  10  continue

c___Extremo derecho b=pi/2
      a_t = a_t + 0.5d0*dsin(0.5d0*pi)

c___Factor comun h
      a_t = h*a_t

c_____Salida de resultados
      write (6,200) n, h , a_t
 200  format (1x,i8,1x,1pd12.5,1x,0pf10.7)

      stop
      end
```

Prog. 8.2 Método compuesto del trapecio

8.6 Bibliografía

BORSE, G.J. *Programación FORTRAN77 con aplicaciones de cálculo numérico en ciencias e ingeniería.* Anaya, 1989.

BREUER, S.; ZWAS, G. *Numerical Mathematics. A Laboratory Approach.* Cambridge University Press, 1993.

CHAPRA, S.C.; CANALE, R.P. *Métodos numéricos para ingenieros con aplicaciones en computadores personales.* McGraw-Hill, 1988.

HOFFMAN, J.D. *Numerical Methods for Engineers and Scientists.* McGraw-Hill, 1992.

9 Aplicaciones al cálculo diferencial

Objetivos

- Comentar algunos conceptos básicos sobre ecuaciones diferenciales ordinarias (EDOs): expresión matemática de una EDO, condiciones iniciales, orden de una EDO, sistemas de EDOs, reducción de una EDO de orden n a un sistema de n EDOs de primer orden.
- Describir dos técnicas numéricas para resolver sistemas de EDOs de primer orden: método de Euler y método de Heun.
- Estudiar ambos métodos mediante su aplicación a algunos problemas de ingeniería.

9.1 Introducción

9.1.1 Ecuación diferencial ordinaria de primer orden

Una gran cantidad de problemas de la física y la ingeniería pueden modelarse matemáticamente mediante *ecuaciones diferenciales*. Como problema modelo, considérese el caso más sencillo: una *ecuación diferencial ordinaria (EDO) de primer orden*,

$$\frac{\mathrm{d}y}{\mathrm{d}x}(x) = f(x,y) \qquad \text{en} \qquad x \in [a,b] \tag{9.1a}$$

complementada con la *condición inicial*

$$y(a) = \alpha \tag{9.1b}$$

La incógnita del problema es la función y de una variable x, definida en el intervalo $[a,b]$. La función $f(x,y)$ y el escalar α son datos. La ecuación 9.1a es *diferencial* porque aparece la

derivada $\frac{\mathrm{d}y}{\mathrm{d}x}$ de la función incógnita y; *ordinaria* porque solamente aparecen derivadas totales, y no derivadas parciales; *de primer orden* porque únicamente aparece la derivada primera, y no derivadas de orden superior. Bajo ciertas condiciones de regularidad, el problema dado por las ecuaciones 9.1a y 9.1b tiene solución única.

En ciertos casos, esta solución puede hallarse analíticamente de manera sencilla. Tómese, por ejemplo,

$$\left.\begin{aligned} \frac{\mathrm{d}y}{\mathrm{d}x}(x) &= cy \qquad \text{en} \qquad x \in [0,1] \\ y(0) &= \alpha \end{aligned}\right\} \tag{9.2}$$

donde las constantes c e α son conocidas. La solución analítica de la ecuación 9.2 es

$$y(x) = \alpha \exp(cx)$$

(verifíquese). En otros muchos casos, sin embargo, la EDO no puede integrarse analíticamente y es necesario emplear alguna técnica numérica.

9.1.2 Ecuaciones diferenciales ordinarias de orden superior a uno

Puede ocurrir que en la ecuación diferencial aparezcan derivadas de y de orden superior a uno. Considérese, por ejemplo, una EDO de orden n, que se escribe como

$$\frac{\mathrm{d}^n y}{\mathrm{d}x^n} = f(x, y, \frac{\mathrm{d}y}{\mathrm{d}x}, \frac{\mathrm{d}^2 y}{\mathrm{d}x^2}, \ldots, \frac{\mathrm{d}^{n-1} y}{\mathrm{d}x^{n-1}}) \qquad \text{en} \qquad x \in [a,b] \tag{9.3}$$

Esta EDO involucra a la función y y a sus n primeras derivadas, puesto que la derivada n-ésima depende, según una función conocida f, de x, y y las $n-1$ primeras derivadas. Para que el problema tenga solución única, son necesarias n condiciones adicionales sobre la función incógnita y. Estas condiciones adicionales se llaman *condiciones iniciales* si están dadas en un mismo punto del intervalo $[a,b]$ o *condiciones de contorno* si están dadas en más de un punto del intervalo $[a,b]$.

Un caso habitual de condiciones iniciales es que la función y y sus $n-1$ primeras derivadas tengan valores prescritos conocidos α_0, α_1, α_2, $\ldots$, α_{n-1} en el extremo a:

$$y(a) = \alpha_0 \quad ; \quad \frac{\mathrm{d}y}{\mathrm{d}x}(a) = \alpha_1 \quad ; \quad \frac{\mathrm{d}^2 y}{\mathrm{d}x^2}(a) = \alpha_2 \quad ; \quad \ldots ; \quad \frac{\mathrm{d}^{n-1} y}{\mathrm{d}x^{n-1}}(a) = \alpha_{n-1} \tag{9.4}$$

Si se complementa la EDO (ecuación 9.3) con las condiciones iniciales (ecuación 9.4), se obtiene un *problema de valor inicial*: se tiene información sobre la función y en el punto $x = a$ (condiciones iniciales), y hay que integrar la EDO para hallar la evolución de la función y (es decir, su valor en todo el intervalo $[a,b]$).

Si, por el contrario, la EDO se complementa con condiciones de contorno, se tiene un *problema de contorno*. Como ejemplo de condiciones de contorno, supóngase que la función y tiene su valor prescrito en n puntos del intervalo $[a,b]$. Los problemas de contorno, que no serán tratados aquí, se resuelven numéricamente transformándolos en problemas de valor inicial.

9.1.3 Reducción de una EDO de orden n a un sistema de n EDOs de primer orden

Una EDO de orden n puede transformarse en un sistema de n EDOs (con n funciones incógnita), todas ellas de primer orden. Es decir, puede reducirse el orden de las derivadas a costa de aumentar el número de incógnitas.

Esta transformación es necesaria puesto que las técnicas numéricas para la resolución de EDOs (en particular, los dos métodos que se estudiarán en los apartados siguientes) están diseñadas para resolver problemas de primer orden.

La idea básica de la transformación es tratar explícitamente como funciones incógnita a las $n-1$ primeras derivadas de la función y. Se utiliza la notación

$$y_{(i)} \equiv \frac{\mathrm{d}^{i-1}y}{\mathrm{d}x^{i-1}} \qquad \text{para} \qquad i = 1, \ldots, n$$

con el convenio de que la derivada cero de la función es la propia función. Las n funciones $y_{(i)}$ son, por lo tanto,

$$\left.\begin{aligned} y_{(1)} &\equiv y \\ y_{(2)} &\equiv \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y_{(1)}}{\mathrm{d}x} \\ y_{(3)} &\equiv \frac{\mathrm{d}^2y}{\mathrm{d}x^2} = \frac{\mathrm{d}y_{(2)}}{\mathrm{d}x} \\ &\vdots \\ y_{(n)} &\equiv \frac{\mathrm{d}^ny}{\mathrm{d}x^n} = \frac{\mathrm{d}y_{(n-1)}}{\mathrm{d}x} \end{aligned}\right\} \tag{9.5}$$

En las ecuaciones 9.5 se muestra también la relación que existe entre las funciones $y_{(i)}$: la *derivada primera* de $y_{(i)}$ es la siguiente función, $y_{(i+1)}$. Esta relación es una consecuencia inmediata de la definición de las $y_{(i)}$ como derivadas sucesivas de y.

Con ayuda de las ecuaciones 9.5, la EDO de orden n de la ecuación 9.3 puede reescribirse como

$$\frac{\mathrm{d}y_{(n)}}{\mathrm{d}x}dx = f(x, y_{(1)}, y_{(2)}, y_{(3)}, \ldots, y_{(n)}) \qquad \text{en} \qquad x \in [a, b] \tag{9.6}$$

donde simplemente se ha hecho un cambio de notación. Si se añade la ecuación 9.6 a las ecuaciones 9.5 (exceptuando la primera) y se transforman también las condiciones iniciales

(ecuación 9.4), se obtiene

$$\left.\begin{aligned} \frac{\mathrm{d}y_{(1)}}{\mathrm{d}x} &= y_{(2)} \\ \frac{\mathrm{d}y_{(2)}}{\mathrm{d}x} &= y_{(3)} \\ &\vdots \\ \frac{\mathrm{d}y_{(n-1)}}{\mathrm{d}x} &= y_{(n)} \\ \frac{\mathrm{d}y_{(n)}}{\mathrm{d}x} &= f(x, y_{(1)}, y_{(2)}, y_{(3)}, \ldots, y_{(n)}) \\ \\ y_{(1)}(a) &= \alpha_0 \\ y_{(2)}(a) &= \alpha_1 \\ &\vdots \\ y_{(n)}(a) &= \alpha_{n-1} \end{aligned}\right\} \tag{9.7}$$

Se ha conseguido el objetivo perseguido: las ecuaciones 9.7 son un sistema de n EDOs (con n funciones incógnita, $y_{(i)}$ para $i = 1, \ldots, n$) de primer orden, puesto que sólo aparecen derivadas primeras de las $y_{(i)}$.

Para compactar las ecuaciones 9.7, puede emplearse notación vectorial. Se definen los vectores $\boldsymbol{y}$, $\boldsymbol{f}$ e $\boldsymbol{\alpha}$ de dimensión n como

$$\boldsymbol{y} = \begin{Bmatrix} y_{(1)} \\ y_{(2)} \\ \vdots \\ y_{(n-1)} \\ y_{(n)} \end{Bmatrix} \quad ; \quad \boldsymbol{f} = \begin{Bmatrix} y_{(2)} \\ y_{(3)} \\ \vdots \\ y_{(n)} \\ f(x, y_{(1)}, y_{(2)}, \ldots, y_{(n)}) \end{Bmatrix} \quad ; \quad \boldsymbol{\alpha} = \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1)} \end{Bmatrix} \tag{9.8}$$

Con estos vectores, el sistema de ecuaciones 9.7 puede ponerse finalmente como

$$\left.\begin{aligned} \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}x} &= \boldsymbol{f}(x, \boldsymbol{y}) \qquad \text{en} \qquad x \in [a, b] \\ \boldsymbol{y}(a) &= \boldsymbol{\alpha} \end{aligned}\right\} \tag{9.9}$$

Es importante notar que la ecuación 9.9 es muy similar a la ecuación 9.1. La única diferencia es que ahora se trabaja con vectores en lugar de con escalares. Esta similitud será muy útil en los apartados siguientes. En primer lugar, se presentan el método de Euler (apartado 9.2) y el método de Heun (apartado 9.3) para resolver una EDO de primer orden (ecuación 9.1); luego se hace la generalización a sistemas de n EDOs de primer orden (ecuación 9.9), simplemente cambiando y, f y α por $\boldsymbol{y}$, $\boldsymbol{f}$ e $\boldsymbol{\alpha}$ (apartado 9.4).

9.2 El método de Euler

El método de Euler se presenta mediante el problema modelo dado por la ecuación 9.1. El primer paso es dividir el intervalo $[a, b]$ en m subintervalos de longitud $h = \frac{b-a}{m}$ mediante puntos x_i, con $i = 0, \ldots, m$ (véase la figura 9.1).

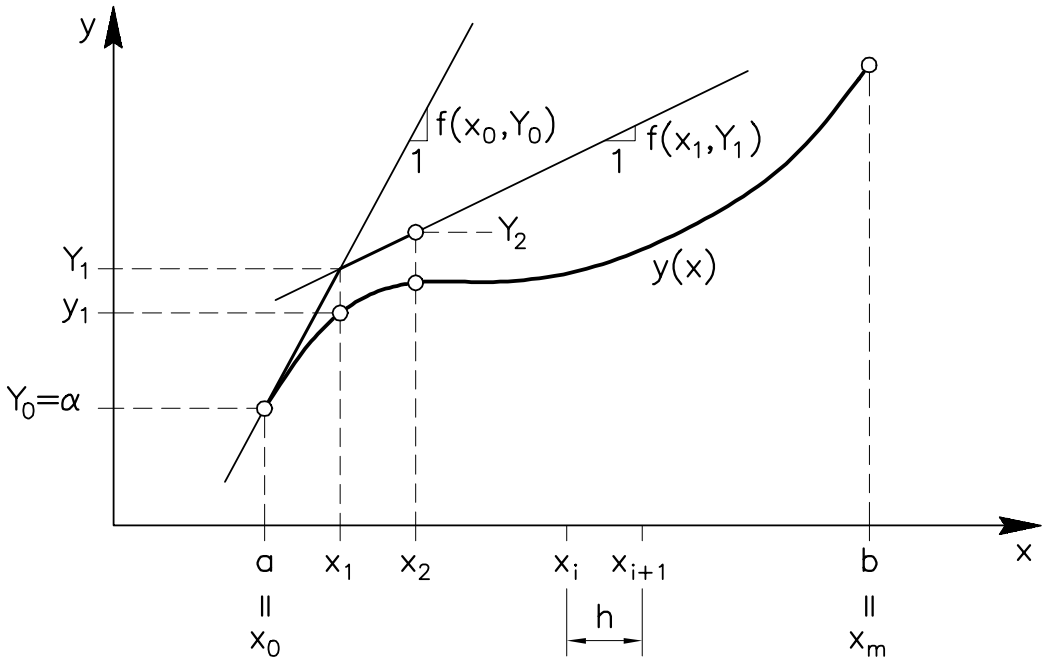


Fig. 9.1 Interpretación gráfica del método de Euler

La ecuación 9.1a debe verificarse para todos los puntos de $[a, b]$. Si se particulariza para un punto x_i de la discretización, resulta

$$\frac{\mathrm{d}y}{\mathrm{d}x}(x_i) = f(x_i, y_i) \tag{9.10}$$

donde se emplea la notación $y_i \equiv y(x_i)$. La idea básica del método de Euler es aproximar la derivada en el punto x_i mediante un *cociente incremental* hacia adelante, utilizando los puntos x_i y x_{i+1}:

$$\frac{\mathrm{d}y}{\mathrm{d}x}(x_i) \approx \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \frac{y_{i+1} - y_i}{h} \tag{9.11}$$

La ecuación 9.11 puede interpretarse geométricamente como aproximar la recta *tangente* a la función y en x_i por la recta *secante* que pasa por los puntos (x_i, y_i) y (x_{i+1}, y_{i+1}) (véase la figura 9.2).

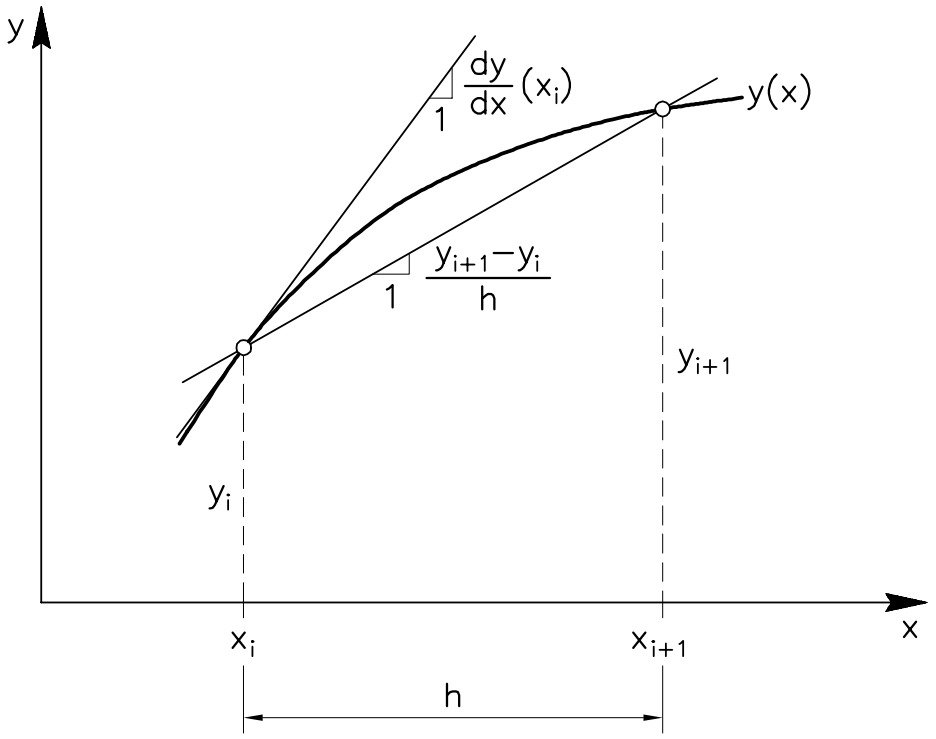


Fig. 9.2 Aproximación secante a la recta tangente

Si el valor de la derivada en x_i que aparece en la ecuación 9.10 se sustituye por la aproximación dada por la ecuación 9.11 se llega a

$$\frac{y_{i+1} - y_i}{h} \approx f(x_i, y_i)$$

de donde puede despejarse y_{i+1} como

$$y_{i+1} \approx y_i + hf(x_i, y_i) \tag{9.12}$$

El signo $\approx$ que aparece en la ecuación 9.12 está causado por la aproximación efectuada en la ecuación 9.11. Para poder sustituirlo por un signo $=$ es necesario trabajar con *valores aproximados* Y de la función y, y escribir

$$Y_{i+1} = Y_i + hf(x_i, Y_i) \tag{9.13}$$

La ecuación 9.13 permite obtener Y_{i+1} (aproximación a y_{i+1}) a partir de Y_i (aproximación a y_i). Este esquema de avance, que debe inicializarse con la condición inicial $Y_0 = \alpha$, constituye el *método de Euler*:

$$\begin{aligned} Y_0 &= \alpha \\ Y_{i+1} &= Y_i + hf(x_i, Y_i) \qquad \text{para} \qquad i = 0, \ldots, m-1 \end{aligned} \tag{9.14}$$

La interpretación gráfica del método de Euler puede verse en la figura 9.1. A partir de los datos $x_0 = a$ e $Y_0 = \alpha$, se calcula $f(x_0, \alpha)$, que es la pendiente de la tangente a la función y en x_0. Se avanza según esta tangente una distancia h, hasta llegar al punto x_1; se obtiene así el valor de Y_1. A partir de x_1 e Y_1 se evalúa $f(x_1, Y_1)$, que es una aproximación a la pendiente $f(x_1, y_1)$ de la tangente a y en x_1. Esta pendiente $f(x_1, Y_1)$ permite avanzar hasta x_2 y obtener Y_2, y así sucesivamente. De esta manera, la función incógnita y se aproxima mediante una línea quebrada, llamada *poligonal de Euler*. En la figura 9.1 se aprecia claramente la diferencia entre y_i (valor exacto) e Y_i (valor aproximado), que en general sólo coinciden para $i = 0$ (condición inicial).

A continuación se muestra un ejemplo de aplicación del método de Euler. Se trata de un problema sencillo con solución analítica, que servirá de referencia para la solución numérica calculada con distintos valores del número m de subintervalos.

Un pilar de hormigón de longitud $L = 4$ m debe resistir una carga $P = 100$ KN y su peso propio (peso específico del hormigón: $\gamma = 25$ KN/m^3). Para conseguir que la tensión normal sea uniforme, se proyecta un pilar de sección transversal variable $S(x)$, con sección en la cabeza del pilar de $S_0 = 0.07$ m^2 (véase la figura 9.3). Combinando algunas nociones de física y cálculo diferencial puede comprobarse (se deja como ejercicio al lector interesado) que la función $S(x)$ debe verificar

$$\frac{\mathrm{d}S(x)}{\mathrm{d}x} = \frac{\gamma S_0}{P} S(x) \qquad \text{en} \qquad x \in [0, L]$$

con la condición inicial

$$S(x_0) = S_0$$

Este problema es muy parecido al ejemplo de la ecuación 9.2, tomando $c = \gamma S_0 / P$. La solución analítica es, en consecuencia,

$$S(x) = S_0 \exp\left(\frac{\gamma S_0}{P} x\right) \tag{9.15}$$

Supóngase que el objetivo es calcular la sección en la base del pilar, $S(x = L)$ con $L = 4$ m, a efectos de diseñar la cimentación. Sustituyendo los valores $S_0 = 0.07$ m^2, $\gamma = 25$ KN/m^3, $P = 100$ KN y $x = 4$ m en la ecuación 9.15 se halla la solución analítica $S(L) = 0.075076$ m^2.

Con la ayuda de un programa en FORTRAN (programa 9.1, apartado 9.5), se resuelve numéricamente el problema mediante el método de Euler para distinto número de subintervalos, m. Se llega a los resultados de la tabla 9.1. Puede verse como el valor numérico de la sección de la base del pilar tiende al valor exacto a medida que m aumenta.

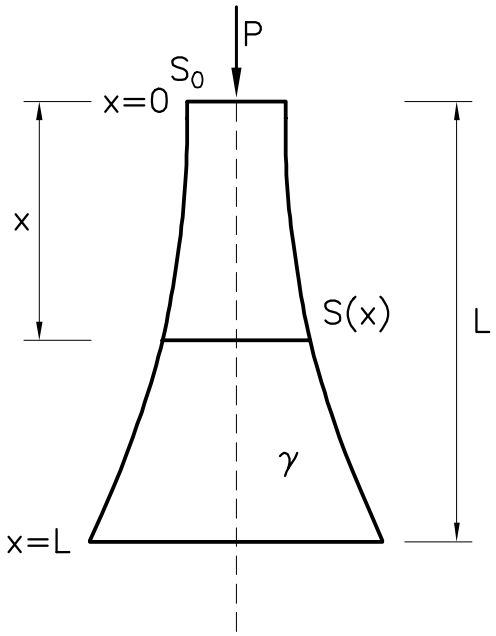


Fig. 9.3 Pilar de sección variable $S(x)$

Tabla 9.1 Cálculo de la sección en la base del pilar mediante el método de Euler

m	$S(x=4)$
1	0.074900
2	0.074986
5	0.075039
10	0.075057
100	0.075074
1000	0.075075
10000	0.075076

9.3 El método de Heun

En el método de Euler que se acaba de presentar, el avance de la solución se produce según una pendiente calculada al principio de cada paso, es decir, en el extremo izquierdo x_i (véase la ecuación 9.13). Parece intuitivamente mejor tomar un valor más representativo de la *pendiente media* de la función y en todo el intervalo $[x_i, x_{i+1}]$. Esta es la estrategia utilizada por una familia de métodos numéricos para la resolución de EDOs, entre los que se encuentra el *método de Heun*.

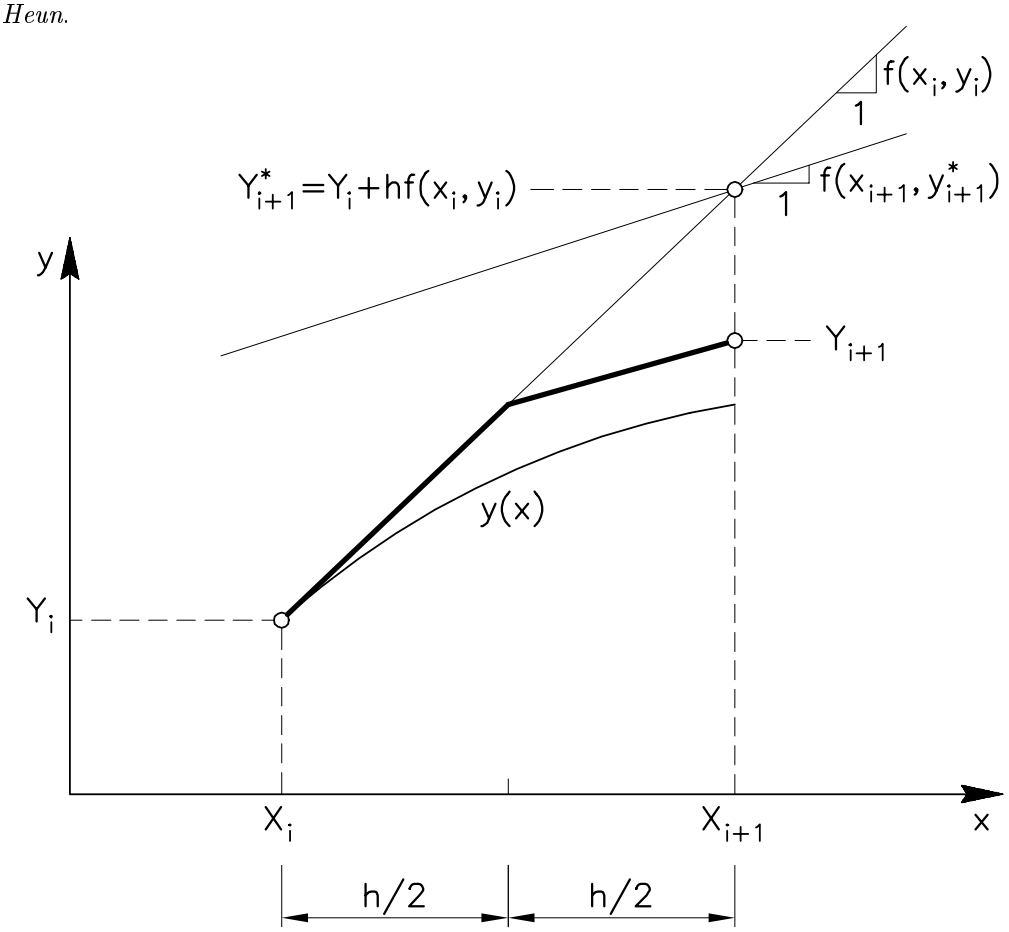


Fig. 9.4 Interpretación gráfica del método de Heun

En este método, la solución avanza en cada paso según el promedio de dos pendientes: la pendiente en el punto inicial (x_i, Y_i) y la pendiente en un punto final auxiliar obtenido según el método de Euler, (x_{i+1}, Y^*_{i+1}), con $Y^*_{i+1} = Y_i + hf(x_i, Y_i)$. En consecuencia, la fórmula de

avance es

$$\begin{aligned} Y^*_{i+1} &= Y_i + hf(x_i, Y_i) \\ Y_{i+1} &= Y_i + \frac{h}{2}\left[f(x_i, Y_i) + f(x_{i+1}, Y^*_{i+1})\right] \end{aligned} \tag{9.16}$$

Junto con la condición inicial $Y_0 = \alpha$, esta fórmula proporciona el método de Heun:

$$\begin{aligned} Y_0 &= \alpha \\ Y^*_{i+1} &= Y_i + hf(x_i, Y_i) \\ Y_{i+1} &= Y_i + \frac{h}{2}\left[f(x_i, Y_i) + f(x_{i+1}, Y^*_{i+1})\right] \qquad \text{para} \qquad i = 0, \ldots, m-1 \end{aligned} \tag{9.17}$$

La interpretación gráfica del método de Heun se muestra en la figura 9.4 para un intervalo genérico $[x_i, x_{i+1}]$. Puede verse que el avance consta de tres fases: *a)* determinación de la pendiente $f(x_i, Y_i)$ al principio de paso y obtención del valor auxiliar Y^*_{i+1}; *b)* determinación de la pendiente $f(x_{i+1}, Y^*_{i+1})$ en el punto auxiliar; *c)* avance de medio intervalo con cada una de las pendientes y obtención del valor final Y_{i+1}.

Si se emplea el método de Heun para resolver el problema del pilar (figura 9.3) con distintos valores de m (número de subintervalos), se obtienen los resultados de la tabla 9.2.

Tabla 9.2 Cálculo de la sección en la base del pilar mediante el método de Heun

m	$S(x = 4)$
1	0.075072
2	0.075075
5	0.075075
10	0.075076

Comparando las tablas 9.1 y 9.2, puede verse que el método de Heun tiene una convergencia mucho más rápida que el método de Euler.

Problema 9.1: En este ejercicio se propone investigar experimentalmente el orden de convergencia de los métodos de Euler y Heun, con ayuda del ejemplo del pilar de hormigón (figura 9.3).

a) Para cada uno de los métodos, calcular el error absoluto E en la sección de la base del pilar, para distintos valores de m (aprovechando y completando los datos de las tablas 9.1 y 9.2).

b) Representar el error E en función de m, en escala log-log.

c) ¿Cuál es el orden de convergencia del método de Euler? ¿Y del método de Heun? ●

9.4 Extensión a un sistema de EDOs de primer orden

Considérese ahora el caso de un sistema de n EDOs de primer orden (ecuación 9.9):

$$\left.\begin{aligned} \frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}x} &= \boldsymbol{f}(x,\boldsymbol{y}) \qquad \text{en} \qquad x \in [a,b] \\ \boldsymbol{y}(a) &= \boldsymbol{\alpha} \end{aligned}\right\}$$

Este sistema puede ser directamente el modelo matemático de un fenómeno físico, o puede provenir de la transformación de una EDO de orden n, tal y como se ha comentado en el subapartado 9.1.3.

Los métodos de Euler y de Heun pueden extenderse sin ninguna dificultad para tratar este sistema. Se trata simplemente de sustituir los *escalares* α, Y_0, Y_i, Y_{i+1} y f que aparecen en las ecuaciones 9.14 y 9.17 por los *vectores* $\boldsymbol{\alpha}$, $\boldsymbol{Y}_0$, $\boldsymbol{Y}_i$, $\boldsymbol{Y}_{i+1}$ y $\boldsymbol{f}$. Así, por ejemplo, para el método de Euler se obtiene:

$$\begin{aligned} \boldsymbol{Y}_0 &= \boldsymbol{\alpha} \\ \boldsymbol{Y}_{i+1} &= \boldsymbol{Y}_i + h\boldsymbol{f}(x_i,\boldsymbol{Y}_i) \qquad \text{para} \qquad i = 1,\ldots,n \end{aligned} \tag{9.18}$$

Teniendo en cuenta la ecuación 9.8, la ecuación 9.18 puede escribirse, componente a componente, como

$$\left.\begin{aligned} \begin{Bmatrix} Y_{(1)}(x_0) \\ Y_{(2)}(x_0) \\ \vdots \\ Y_{(n)}(x_0) \end{Bmatrix} &= \begin{Bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{Bmatrix} \\ \begin{Bmatrix} Y_{(1)}(x_{i+1}) \\ Y_{(2)}(x_{i+1}) \\ \vdots \\ Y_{(n-1)}(x_{i+1}) \\ Y_{(n)}(x_{i+1}) \end{Bmatrix} &= \begin{Bmatrix} Y_{(1)}(x_i) \\ Y_{(2)}(x_i) \\ \vdots \\ Y_{(n-1)}(x_i) \\ Y_{(n)}(x_i) \end{Bmatrix} + h \begin{Bmatrix} Y_{(2)}(x_i) \\ Y_{(3)}(x_i) \\ \vdots \\ Y_{(n)}(x_i) \\ f(x, Y_{(1)}(x_i), Y_{(2)}(x_i), \ldots, Y_{(n)}(x_i)) \end{Bmatrix} \end{aligned}\right\} \tag{9.19}$$

donde $Y_{(j)}(x_i)$ es la aproximación a $y_{(j)}(x_i)$ (valor de la función $y_{(j)}$ en el punto x_i).

Es importante darse cuenta de que en la ecuación 9.19 se tratan todas las ecuaciones del sistema de forma conjunta. Desde luego, sería totalmente incorrecto intentar resolver el sistema tratando las ecuaciones una a una, por separado, empleando el método de Euler para una EDO visto en el apartado 9.2.

Problema 9.2: Generalizar el método de Heun (apartado 9.3) al caso de sistemas de EDOs. ●

Problema 9.3: En el diseño de una estación depuradora de aguas residuales es necesario estudiar una reacción química, donde las concentraciones y_A e y_B de los reactivos A y B varían en el tiempo según un sistema de dos ecuaciones diferenciales ordinarias de primer orden

$$\left.\begin{aligned} \frac{dy_A}{dt} &= -\kappa\ y_A\ y_B \\ \frac{dy_B}{dt} &= -\kappa\ y_A\ y_B^2 \end{aligned}\right\} \qquad \text{para} \qquad t \in [0,1]$$

con condiciones iniciales $y_A(0) = y_B(0) = 1$ y siendo $\kappa = 24$ la constante cinética de la reacción.

a) Escribir un programa en FORTRAN que resuelva el sistema de EDOs mediante el método de Euler y el método de Heun.

b) Resolver el problema mediante los dos métodos y para diferentes valores de m (número de subintervalos). Comentar los resultados obtenidos.

c) ¿Para qué valor del tiempo t el reactivo A ha reducido su concentración a la mitad de la concentración inicial $y_A(0)$? ●

9.5 Apéndice

```
c
c        Este programa resuelve la ecuacion diferencial ordinaria
c                     y'(x) = cy(x) para x en [a,b]
c                     y(a)  = y0
c        por el METODO DE EULER
c        Aplicacion: problema de la columna con tension uniforme
c
c---------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
c___Carga vertical P
      write (6,100)
      read  (5,*) p
 100  format (1x,'P= ')

c___Peso especifico gamma
      write (6,200)
      read  (5,*) gamma
 200  format (1x,'gamma= ')

c___Longitud de la columna hlong
      write (6,300)
      read  (5,*) hlong
 300  format (1x,'longitud= ')
```

```
c___Seccion de la cabeza (extremo superior) de la columna
      write (6,400)
      read  (5,*) s0
 400  format (1x,'Seccion extremo superior= ')

c___Numero de intervalos m
      write (6,500)
      read  (5,*) m
 500  format (1x,'Numero de intervalos= ')

c___Calculos previos
c___Extremos a, b del intervalo
      a = 0.d0
      b = hlong

c___Parametro c de la EDO
      c = gamma*s0/p

c___Paso h (discretizacion)
      h = (b-a)/dble(m)

c___Metodo de Euler
c___Inicializacion
      i = 0
      x = a
      y = s0

c___Bucle
      do 10 i=1,m
        f = c*y
        y = y + h*f
        x = x + h
  10  continue

c___Salida de resultados
      write (6,600) y
 600  format (1x, 'La seccion en la base del pilar es=', d12.5)

      stop
      end
```

Prog. 9.1 Método de Euler

```
c
c        Este programa resuelve la ecuacion diferencial ordinaria
c                     y'(x) = cy(x) para x en [a,b]
c                     y(a)  = y0
c        por el METODO DE HEUN
c        Aplicacion: problema de la columna con tension uniforme
c
c-----------------------------------------------------------------
      implicit real*8 (a-h,o-z)

        .
        .
        .

c___Metodo de Heun
c___Inicializacion
      i = 0
      x = a
      y = s0

c___Bucle
      do 10 i=1,m
        f = c*y
        yaux = y + h*f
        x = x + h
        faux = c*yaux
        y = y + 0.5d0*h*(f+faux)
  10  continue

        .
        .
        .

      stop
      end
```

Prog. 9.2 Método de Heun

Las únicas diferencias entre los programas 9.1 y 9.2 se encuentran en el interior del bucle, dado que es precisamente aquí donde se aplica la definición del método de resolución escogido. La lectura de datos, la inicialización de las variables y la escritura de resultados, sustituidos por puntos suspensivos en el listado del programa 9.2, pueden realizarse de la misma manera en ambos programas.

9.6 Bibliografía

BORSE, G.J. *Programación FORTRAN77 con aplicaciones de cálculo numérico en ciencias e ingeniería.* Anaya, 1989.

CHAPRA, S.C.; CANALE, R.P. *Métodos numéricos para ingenieros con aplicaciones en computadores personales.* McGraw-Hill, 1988.

HOFFMAN, J.D. *Numerical Methods for Engineers and Scientists.* McGraw-Hill, 1992.

10 Resolución de los problemas propuestos

Objetivos

- Comentar los aspectos más interesantes de los problemas propuestos.
- Proporcionar al lector el resultado numérico de los problemas.

10.1 Problemas del capítulo 2

Problema 2.1

Para conseguir que en la columna `C` de la hoja de cálculo aparezcan las ordenadas de la función $y = 3x^2 - 2x - 3$ basta con realizar dos de los pasos descritos en el apartado 2.6.

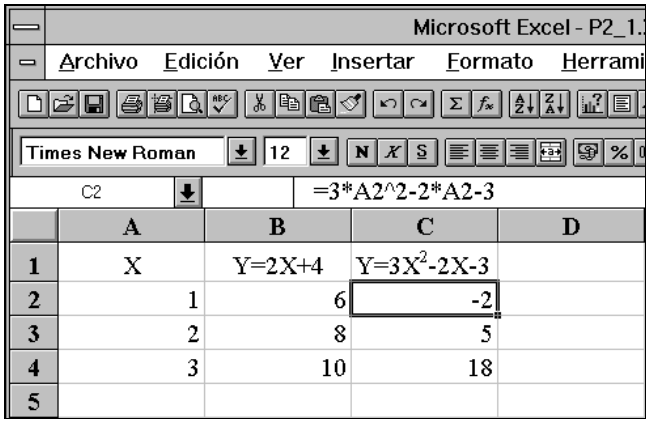


Fig. 2.1.1 Evaluación de la función $y = 3x^2 - 2x - 3$

1. En primer lugar es necesario introducir en la casilla `C2` la fórmula correspondiente a la mencionada función según se puede observar en la figura 2.1.1.

2. A continuación, debe arrastrarse dicha fórmula en sentido vertical tantas casillas como abscisas (en este caso 3) se tengan.

Problema 2.2

En este caso, partiendo de los resultados del problema anterior, se trata de modificar el rango de las abscisas para que las funciones $y = 2x + 4$ e $y = 3x^2 - 2x - 3$ queden definidas en el intervalo [-2,2].

En primer lugar, conviene borrar (empleando, por ejemplo, la selección de casillas y la tecla *suprimir*) los cálculos de las celdas `B3`, `B4`, `C3` y `C4`. A continuación, puede introducirse como valor constante el escalar `-2` en la casilla `A2` que indica el extremo inferior del intervalo. Automáticamente, los valores de las casillas `B2` y `C2` cambiarán a los valores correctos, esto es `0` y `13`, según se puede apreciar en la figura 2.2.1.

La más sencilla de las posibilidades para definir la serie de abscisas creciente de 0.5 en 0.5 a partir de -2 consiste en programar en la casilla `B3` la fórmula que se muestra en la figura 2.2.1 (`= A2 + 0,5`) y a continuación arrastrar dicha celda en sentido descendente hasta alcanzar el extremo superior del intervalo. Desgraciadamente, no se trata de un proceso automático, en el sentido de que el usuario debe decidir el número de casillas que arrastra (hasta llegar al extremo superior del intervalo, 2).

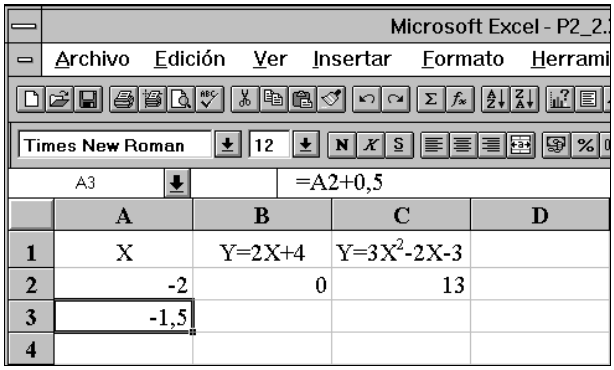
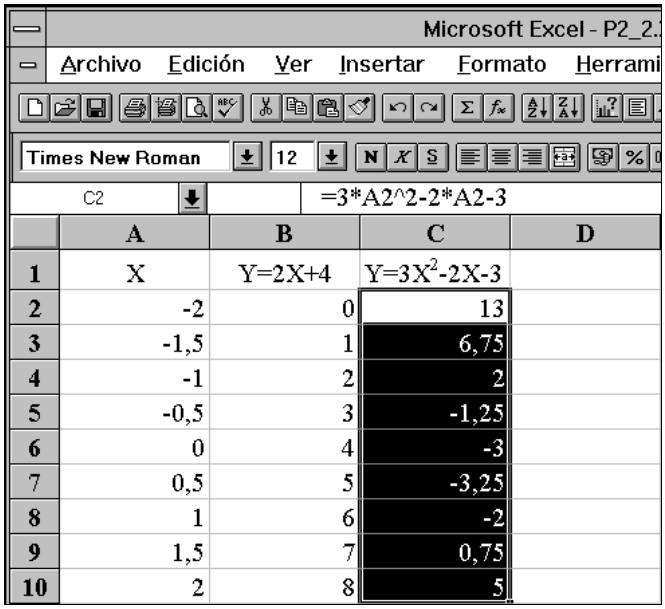


Fig. 2.2.1 Programación de las abscisas

Finalmente, bastaría con arrastrar las casillas `B2` y `C2` en sentido descendente para conseguir el recálculo de las ordenadas correspondientes a ambas funciones. El resultado se puede observar

en la figura 2.2.2.



	A	B	C	D
1	X	Y=2X+4	$Y=3X^2-2X-3$	
2	-2	0	13	
3	-1,5	1	6,75	
4	-1	2	2	
5	-0,5	3	-1,25	
6	0	4	-3	
7	0,5	5	-3,25	
8	1	6	-2	
9	1,5	7	0,75	
10	2	8	5	

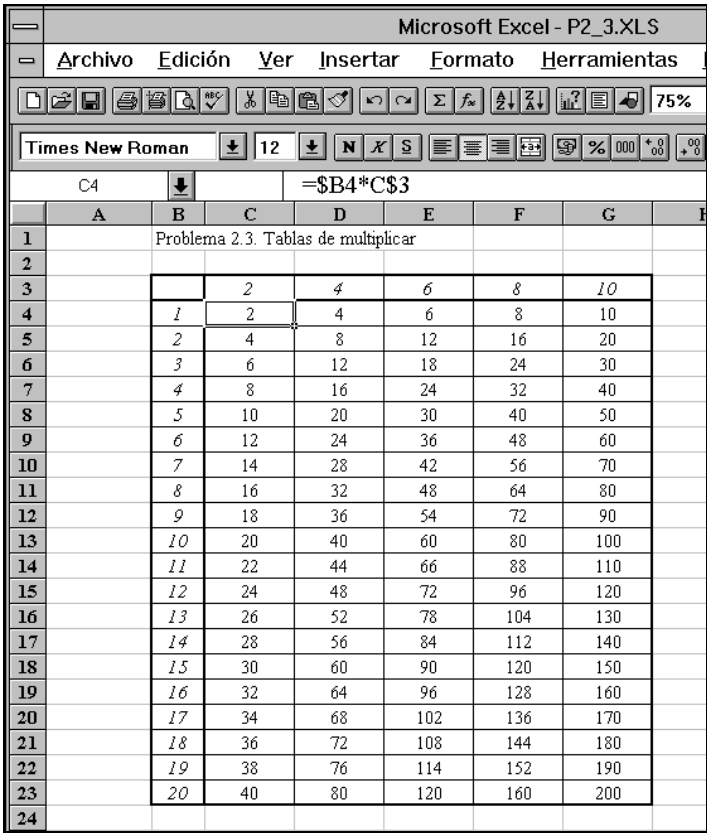
Fig. 2.2.2 Evaluación de las funciones $y = 2x + 4$ *e* $y = 3x^2 - 2x - 3$

Problema 2.3

Para construir la tabla de multiplicar, se comenzará programando por ejemplo en la fila `3` y en la columna `B` los valores de los argumentos que se deben multiplicar. Para ello se puede emplear lo aprendido en el problema 2.2. Así, las casillas `C3` y `B4` deberán contener los valores iniciales de ambas series de datos, `2` en ambos casos.

Seguidamente, deberá introducirse en la casilla `D3` la fórmula `= C3 + 2` y arrastrarla en sentido horizontal hasta alcanzar la columna `G`. De la misma manera, la celda `B5` deberá contener la fórmula `= B4 + 1`, que será arrastrada en sentido vertical hasta la fila `23`, según consta en la figura 2.3.1.

Finalmente, queda por escribir la fórmula de multiplicación. Una de las posibilidades consiste en programar en la casilla `C4` la expresión `= $B4 * C$3`. El bloqueo alternativo de las filas para el primer argumento (el que está a lo largo de la columna `B`) y de las columnas para el segundo argumento (el que está situado a lo largo de la fila `3`) permite que esta única fórmula arrastrada en sentido horizontal y vertical proporcione el resultado correcto para todas las casillas, de manera que resulta innecesaria la introducción de ninguna otra fórmula.



	A	B	C	D	E	F	G
1		Problema 2.3. Tablas de multiplicar					
2							
3			*2*	*4*	*6*	*8*	*10*
4		*1*	2	4	6	8	10
5		*2*	4	8	12	16	20
6		*3*	6	12	18	24	30
7		*4*	8	16	24	32	40
8		*5*	10	20	30	40	50
9		*6*	12	24	36	48	60
10		*7*	14	28	42	56	70
11		*8*	16	32	48	64	80
12		*9*	18	36	54	72	90
13		*10*	20	40	60	80	100
14		*11*	22	44	66	88	110
15		*12*	24	48	72	96	120
16		*13*	26	52	78	104	130
17		*14*	28	56	84	112	140
18		*15*	30	60	90	120	150
19		*16*	32	64	96	128	160
20		*17*	34	68	102	136	170
21		*18*	36	72	108	144	180
22		*19*	38	76	114	152	190
23		*20*	40	80	120	160	200
24							

Fig. 2.3.1 Tabla de multiplicar

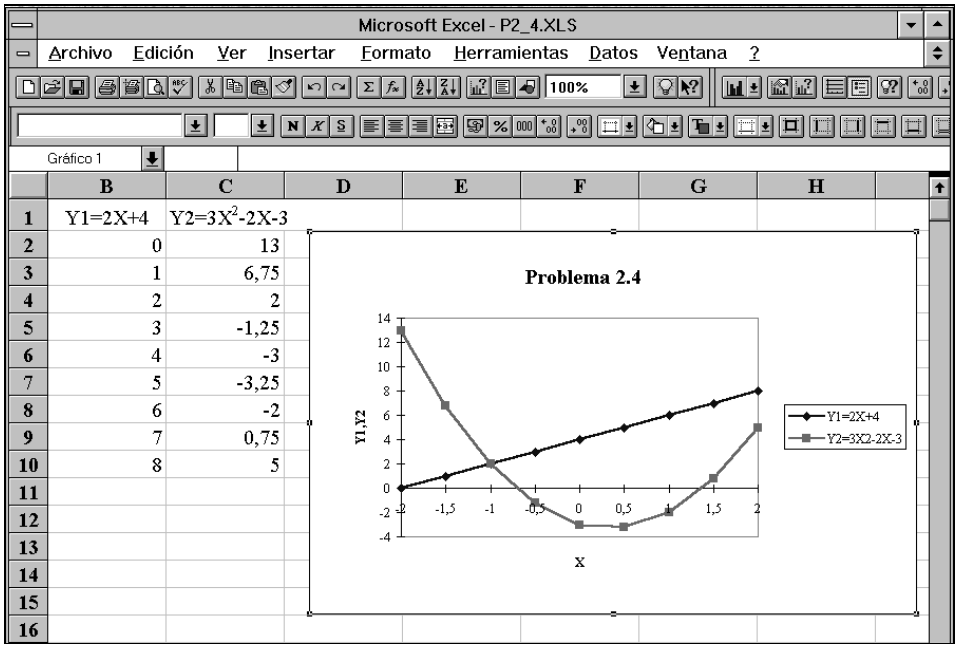
En la figura 2.3.1 se muestra la expresión a introducir en la celda `C4` para confeccionar las tablas de multiplicación.

Problema 2.4

A partir de la hoja de cálculo obtenida en el problema 2.3 basta invocar al *Asistente para gráficos* para obtener el dibujo deseado y estimar los puntos de corte.

Los resultados pueden apreciarse en la figura 2.4.1. En ella se observa cómo uno de los dos puntos de corte entre la recta $y = 2x + 4$ y la parábola $y = 3x^2 - 2x - 3$ es $x = -1$, mientras que el otro queda fuera del rango de definición del dibujo.

Ello puede solventarse ampliando el dominio de x y modificando posteriormente el área de dibujo, con lo que se obtiene el segundo punto de corte, $x = 2.33$.



	B	C
1	Y1=2X+4	Y2=3X^2-2X-3
2	0	13
3	1	6,75
4	2	2
5	3	-1,25
6	4	-3
7	5	-3,25
8	6	-2
9	7	0,75
10	8	5

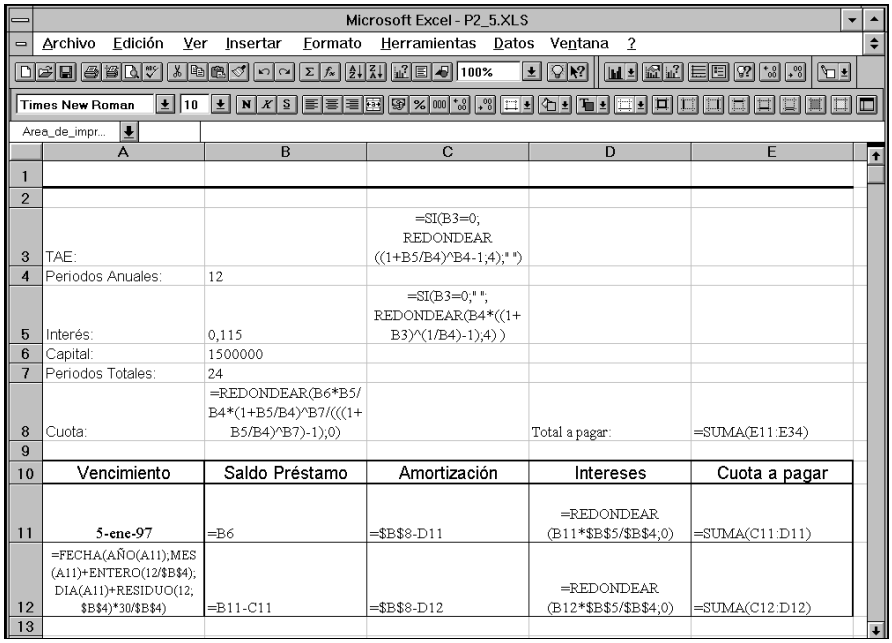
Fig. 2.4.1 Representación gráfica de las funciones $y = 2x + 4$ e $y = 3x^2 - 2x - 3$

Problema 2.5

Las instrucciones necesarias para confeccionar la hoja de cálculo se encuentran suficientemente detalladas a lo largo de la explicación del problema, si bien en algún caso puede resultar conveniente complementar las citadas indicaciones con las que puede proporcionar el *Asistente de funciones* de Excel. Para conocer el significado y uso concreto de algunas de las funciones que aquí se usan basta con emplear la opción *Insertar–Función* que se encuentra bajo el menú desplegable *Insertar*. Los formatos de las celdas en que aparecen expresiones porcentuales o monetarias se obtienen modificando el formato original de presentación de los números. Ello se consigue empleando la opción *Formato–Celdas–Número* del menú desplegable *Formato*.

Con respecto a la pregunta final acerca de si está bien calculada la última cuota que se debe pagar, la respuesta es que no. Ello se debe al hecho de que esta cuota ha sido calculada en la casilla `B8` empleando la función `redondear`. De esta manera se produce un desfase entre el total teórico que hay que devolver y el real devuelto. Para comprobarlo, basta evaluar la fórmula de la casilla `B8` (*Cuota*) sin emplear la función `redondear`. En ese caso, la cuota mensual que

hay pagar es de 70 260.47 pesetas, lo que al cabo de 24 mensualidades representan un total de 1 686 251.35 pesetas. En consecuencia, la diferencia con respecto a la casilla `E8` (*Total a pagar*) asciende a poco más de 11 pesetas a favor del banco, que deberían ser abonadas en el último pago –con lo que éste pasaría a ser de 70 271 pesetas– posibilitando la total extinción del crédito. En la figura 2.5.1 se pueden apreciar las fórmulas introducidas en las diversas casillas de la hoja de cálculo para obtener los mencionados resultados.



	A	B	C	D	E
1					
2					
3	TAE:		=SI(B3=0; REDONDEAR ((1+B5/B4)^B4-1;4);" ")		
4	Periodos Anuales:	12			
5	Interés:	0,115	=SI(B3=0;" "; REDONDEAR(B4*((1+ B3)^(1/B4)-1);4))		
6	Capital:	1500000			
7	Periodos Totales:	24			
8	Cuota:	=REDONDEAR(B6*B5/ B4*(1+B5/B4)^B7/(((1+ B5/B4)^B7)-1);0)		Total a pagar:	=SUMA(E11:E34)
9					
10	Vencimiento	Saldo Préstamo	Amortización	Intereses	Cuota a pagar
11	5-ene-97	=B6	=$B$8-D11	=REDONDEAR (B11*$B$5/$B$4;0)	=SUMA(C11:D11)
12	=FECHA(AÑO(A11);MES (A11)+ENTERO(12/$B$4); DIA(A11)+RESIDUO(12; $B$4)*30/$B$4)	=B11-C11	=$B$8-D12	=REDONDEAR (B12*$B$5/$B$4;0)	=SUMA(C12:D12)
13					

Fig. 2.5.1 Fórmulas para el cálculo de las tasas de amortización de un crédito

10.2 Problemas del capítulo 3

Problema 3.1

Al ejecutar el programa se obtiene que las dos variables valen 0. Este resultado implica que al realizar una división entre variables enteras y almacenar el resultado en otra variable entera, el FORTRAN escoge la *parte entera* del cociente (es decir, *no* redondea al entero más cercano).

Problema 3.2

En la sentencia `OPEN` el archivo de resultados `RESUL.RES` se abre con `STATUS='NEW'`. Al ejecutar el programa 3.7 por segunda vez, el archivo `RESUL.RES` ya existe y se produce un error. Existen varias maneras de evitar este error. La más sencilla consiste en renombrar o borrar el archivo de resultados generado durante la primera ejecución antes de ejecutar el programa por segunda vez. Una alternativa es declarar el fichero `RESUL.RES` con `STATUS='UNKNOWN'`. De esta forma, si el fichero no existe el programa lo crea, y si ya existe escribe encima de la información ya existente; esto quiere decir que sólo se conservan los resultados correspondientes a la última ejecución del programa y que, por lo tanto, se *pierden* los resultados de las anteriores ejecuciones.

Problema 3.3

El programa 3.3.1 resuelve las cuestiones planteadas de una forma concisa y sistemática. La justificación teórica de los algoritmos utilizados para cada una de las cuestiones se describe a continuación. Sea una circunferencia de radio r entero. Si (i, j) es un punto perteneciente a dicha circunferencia, entonces se verifican simultáneamente las condiciones siguientes:

$$i \leq r \tag{3.3.1}$$

$$j \leq r \tag{3.3.2}$$

$$i^2 + j^2 = r^2 \tag{3.3.3}$$

Para identificar *todos* los puntos de coordenadas enteras que pertenecen a la circunferencia, se empieza por *recorrer* todos los puntos de coordenadas enteras que verifican simultáneamente las condiciones 3.3.1 y 3.3.2. Para cada uno de estos puntos se evalúa la expresión $i^2 + j^2$; si se verifica la igualdad 3.3.3, entonces se tiene también que

$$(-i)^2 + j^2 = r^2 \quad ; \quad i^2 + (-j)^2 = r^2 \quad ; \quad (-i)^2 + (-j)^2 = r^2$$

es decir, que los puntos $(-i, j)$, $(i, -j)$ y $(-i, -j)$ también satisfacen la condición de pertenencia a la circunferencia. Es interesante notar que, dado el punto de coordenadas (i, j), el punto $(-i, j)$ se obtiene por simetría del anterior respecto al eje de ordenadas, el punto $(i, -j)$ se obtiene por simetría respecto al eje de abscisas y el punto $(-i, -j)$ es el simétrico del (i, j) respecto al origen.

Para obtener los puntos interiores a la circunferencia, se procede del mismo modo pero sustituyendo la condición 3.3.3 por la siguiente:

$$i^2 + j^2 \leq r^2 \tag{3.3.4}$$

A nivel de programación, es interesante notar la existencia de bucles `DO-ENDDO` anidados, que se utilizan para recorrer todos los puntos (i, j) que sean *candidatos* a solución.

Por lo que respecta a la escritura de los resultados, se pueden observar dos procedimientos distintos. Por una parte, las coordenadas de cada uno de los puntos se escriben en pantalla mediante varias sentencias `WRITE` diferentes; cada sentencia `WRITE` produce la escritura de un punto, *más* un salto de línea. En cambio, cuando se accede al archivo de resultados se escriben todos los puntos de una vez, utilizando una sola sentencia `WRITE`; en este caso, el salto de línea se controla mediante la instrucción `FORMAT` correspondiente.

```
c
c        Este programa detecta los puntos con coordenadas
c        enteras: 1) sobre una circunferencia
c                 2) en el interior de una circunferencia
c_____________________________________________________________________
c___Introduccion del valor del radio
 10   write (6,*) ' Valor del radio : '
      read (5,*) nrad
      if ((nrad.le.0) .or. (nrad.gt.100)) goto 10
      nrad2 = nrad*nrad

c___Apertura del archivo de resultados
      open (unit=20,file='circunferencia.res',status='unknown')

c___Obtencion de puntos sobre la circunferencia
      write (6,*) ' Puntos sobre la circunferencia '
      write (6,*)
      write (20,*) ' Puntos sobre la circunferencia '
      write (20,*)

      do i=0,nrad
         do j=0,nrad
            if ( (i*i+j*j) .eq. nrad2) then
               write (6,*)    ' '
               write (6,100)  i, j
               write (6,100)  i,-j
               write (6,100) -i, j
               write (6,100) -i,-j
               write (20,200)  i, j, i,-j, -i, j, -i,-j
            endif
         enddo
      enddo
```

```
      write (6,*)

c___Obtencion de puntos interiores a la circunferencia
      write (6,*) ' Puntos interiores a la circunferencia '
      write (6,*)
      write (20,*) ' Puntos interiores a la circunferencia '
      write (20,*)

      do i=0,nrad
         do j=0,nrad
            if ( (i*i+j*j) .lt. nrad2) then
               write (6,*)    ' '
               write (6,100)  i, j
               write (6,100)  i,-j
               write (6,100) -i, j
               write (6,100) -i,-j
               write (20,200)  i, j, i,-j, -i, j, -i,-j
            endif
         enddo
      enddo

      close (20)

 100  format (5x,'El punto es : (',i4,',',i4,')')
 200  format (4(5x,'El punto es : (',i4,',',i4,')'/))

      stop
      end
```

Prog. 3.3.1 Puntos de coordenadas enteras en una circunferencia

En la tabla 3.3.1 se presenta el fichero de resultados `CIRCUNFERENCIA.RES` obtenido para $r = 2$. Se puede observar que los puntos que tienen una coordenada nula aparecen dos veces, y que el centro de la circunferencia (que tiene ambas coordenadas nulas) aparece cuatro veces. La justificación de este fenómeno se expone a continuación. Sea $(a, 0)$ un punto de la circunferencia. El programa realiza tres simetrías de este punto para obtener tres nuevos puntos de la circunferencia; en este caso, dichas simetrías dan por resultados los puntos de coordenadas $(-a, 0)$, $(a, -0)$ y $(-a, -0)$. Así pues, el número total de puntos *diferentes* detectados no es cuatro sino dos, ya que $(a, 0) \equiv (a, -0)$ y $(-a, 0) \equiv (-a, -0)$. Para puntos del tipo $(0, b)$ (pertenecientes al eje de ordenadas), la situación es análoga. Finalmente, cuando $a = b = 0$, es decir, cuando el punto detectado es el origen de coordenadas, las tres simetrías dan por resultado el punto original $(0, 0)$. Por lo que respecta a la detección de puntos interiores a la circunferencia, se aplican los mismos razonamientos.

Tabla 3.3.1 Fichero de resultados para $r = 2$

```
                Puntos sobre la circunferencia

                   El punto es : (   0,   2)
                   El punto es : (   0,  -2)
                   El punto es : (   0,   2)
                   El punto es : (   0,  -2)

                   El punto es : (   2,   0)
                   El punto es : (   2,   0)
                   El punto es : (  -2,   0)
                   El punto es : (  -2,   0)

                Puntos interiores a la circunferencia

                   El punto es : (   0,   0)
                   El punto es : (   0,   0)
                   El punto es : (   0,   0)
                   El punto es : (   0,   0)

                   El punto es : (   0,   1)
                   El punto es : (   0,  -1)
                   El punto es : (   0,   1)
                   El punto es : (   0,  -1)

                   El punto es : (   1,   0)
                   El punto es : (   1,   0)
                   El punto es : (  -1,   0)
                   El punto es : (  -1,   0)

                   El punto es : (   1,   1)
                   El punto es : (   1,  -1)
                   El punto es : (  -1,   1)
                   El punto es : (  -1,  -1)
```

Problema 3.4

El programa 3.4.1 permite evaluar un polinomio $p_4(x)$ mediante la regla de Horner. Cabe comentar dos aspectos. Por una parte, los cálculos se realizan en simple precisión, puesto que no se declara explícitamente ninguna variable. Conviene notar que la variable entera `I` se convierte al formato `REAL*4` a la hora de realizar con ella operaciones aritméticas que involucren variables reales; esta conversión se lleva a cabo mediante la instrucción `FLOAT`. Por otra parte, mediante

la variable `INTER` el usuario puede seleccionar el número de intervalos (y, por tanto, el número de puntos) en los que se evaluará el polinomio.

```
c
c        Evaluacion de un polinomio mediante la REGLA DE HORNER
c_____________________________________________________________________
c___Entrada del los datos
      write (6,*) ' Introduce el limite inferior : '
      read (5,*) x_inf
      write (6,*) ' Introduce el limite superior : '
      read (5,*) x_sup
      write (6,*) ' Introduce el numero de intervalos : '
      read (5,*) inter

c___Calculo del paso
      deltax = (x_sup - x_inf) / float(inter)

c___Apertura del archivo de resultados
      open (unit=20,file='horner.res',status='unknown')

c___Cabecera del archivo
      write (20,50) 'X','P4(X)'
      write (20,50) '==========','=========='

c___Evaluacion del polinomio
      do i=0,inter
         x = x_inf + float(i)*deltax
         y= (( (2.0*x + 20.0)*x + 70.0)*x + 100.0)*x + 48.0
         write (20,100)  x, y
      enddo

      close (20)

  50  format (2x,A15,3x,A15)
 100  format (2x,f15.7,3x,f15.7)

      stop
      end
```

Prog. 3.4.1 Evaluación de un polinomio mediante la regla de Horner

En la tabla 3.4.1 se presenta el archivo de resultados `HORNER.RES`, producido por el programa 3.4.1 al evaluar el polinomio $p_4(x) = 2x^4 - 20x^3 + 70x^2 - 100x + 48$ mediante la regla de

Horner con `INTER = 6` (pasos de $\Delta x = 0.5$).

Tabla 3.4.1 Resultados obtenidos para `INTER = 6`

X	P4(X)
-4.0000000	0.0000000
-3.5000000	-1.8750000
-3.0000000	0.0000000
-2.5000000	1.1250000
-2.0000000	0.0000000
-1.5000000	-1.8750000
-1.0000000	0.0000000

10.3 Problemas del capítulo 4

Problema 4.1

APARTADO A)

En el apartado 4.2.1 se ha detallado el almacenamiento de los números enteros en un ordenador cuando se reserva un bit para el signo. Sin embargo, en las variables *unsigned* no se debe reservar dicha posición porque el número siempre será positivo. A continuación se calculará cuál es el mayor y menor número entero cuando éste se representa en sistema binario mediante S posiciones (ver figura 4.1.1).

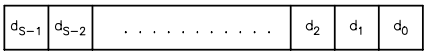


Fig. 4.1.1 Representación de un número unsigned integer *mediante* S *posiciones*

De acuerdo con la expresión 4.1 la representación es

$$\left(d_{S-1}\, d_{S-2}\, \ldots\, d_2\, d_1\, d_0\right)_2 \;=\; \left(d_{S-1}2^{S-1} + d_{S-2}2^{S-2} + \ldots + d_2 2^2 + d_1 2^1 + d_0 2^0\right)$$

En estas condiciones, el número entero máximo (en valor absoluto) que se puede almacenar es (ver figura 4.1.2)

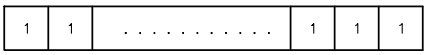


Fig. 4.1.2 unsigned integer *máximo*

y su valor es

$$\left|N_{máx}\right| \;=\; 1\cdot 2^{S-1} + 1\cdot 2^{S-2} + \ldots + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 \;=\; 1\,\frac{1-2^S}{1-2} \;=\; 2^S - 1$$

Del mismo modo, la figura 4.1.3 muestra el almacenamiento del número entero no nulo más próximo a cero. Su valor es lógicamente el mismo que para las variables `INTEGER` de FORTRAN (ver expresión 4.3)

$$\left|N_{mín}\right| \;=\; 1\cdot 2^0 \;=\; 1$$

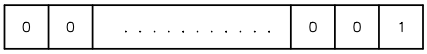


Fig. 4.1.3 unsigned integer *no nulo mínimo*

Obsérvese que, evidentemente, el número mayor que se puede almacenar con `S` posiciones de memoria sin incluir el signo es el mismo que el número mayor que se puede almacenar con `S + 1` posiciones incluyendo el signo. Ya que, en ambos casos, el número de posiciones que almacenan dígitos (sin el signo) es `S`. La fórmula del número máximo se podría haber deducido de la expresión 4.2 del apartado 4.2.1 simplemente sustituyendo `S` por `S + 1`.

APARTADO B)

El mayor *unsigned integer* de 16 bits es ($\mathtt{S} = 16$)

$$\left|N_{máx}\right| \;=\; 2^{16} - 1 \;=\; 65535$$

El mayor *unsigned integer* de 32 bits es ($\mathtt{S} = 32$)

$$\left|N_{máx}\right| \;=\; 2^{32} - 1 \;=\; 4\,294\,967\,295$$

El mayor `INTEGER*4` del lenguaje FORTRAN es (substituir $\mathtt{S} = 32$ en la expresión 4.2)

$$\left|N_{máx}\right| \;=\; 2^{32-1} - 1 \;=\; 2\,147\,483\,647$$

Por lo tanto, el *unsigned integer* de 32 bits permite almacenar el mayor número entero.

APARTADO C)

A partir de los resultados del apartado anterior es fácil comprobar que sólo el *unsigned integer* de 32 bits permite almacenar el número $k = 3\,125\,587\,976$. En cambio, ninguno de los tres tipos de variables enteras permite almacenar el número π, puesto que se trata de un número real no entero.

Problema 4.2

El pseudocódigo correspondiente a la resolución de una ecuación de segundo grado es

```
1.  Entrar a, b y c
2.  dis = b^2 - 4ac
3.  Si dis ≥ 0 entonces
        x1 = (-b + √dis) / (2a)
        x2 = (-b - √dis) / (2a)
        Escribir x1 y x2
    Si no
        z1 = (cmplx(-b) + √cmplx(dis)) / cmplx(2a)
        z2 = (cmplx(-b) - √cmplx(dis)) / cmplx(2a)
        Escribir z1 y z2
    Fin de si
4.  Parar
```

La figura 4.2.1 muestra el diagrama de flujo correspondiente al algoritmo de Euclides para determinar el máximo común divisor de m y n.

Problema 4.3

Apartado a)

Tras la inicialización de las variables (que permite la entrada en el bucle), dentro del bucle de iteraciones la variable `a` toma por valor las sucesivas potencias de 2, `a` $= (1/2)^i$. El valor del contador `i` se incrementa en cada iteración y se calcula la potencia de $1/2$ correspondiente multiplicando la potencia del paso anterior por `half` $= 1/2$, `a` $= (1/2)^i = (1/2)^{i-1} * (1/2)$, con la instrucción `a` $\leftarrow$ `a*half`. Por lo tanto, durante las iteraciones la variable `b` toma los valores `b` $= 1 +$ `a` $= 1 + (1/2)^i$ para los distintos valores de la variable `i`.

El bucle sigue iterando mientras se cumple la condición `b > 1`. En consecuencia, sólo se podrá salir del bucle en el momento en que `b` $= 1 +$ `a` se confunda con 1. Así, cuando se sale del bucle, la variable `a` contiene la primera potencia de $1/2$ menor que la precisión de las variables con que se trabaja. Es decir, el primer término del sumatorio que no va a afectar a la suma total S_n. Por lo tanto, una vez fuera del bucle, la precisión de las variables utilizadas se puede calcular como `2*a` o $(1/2)^{i-1}$ (los valores de la iteración anterior, que todavía cumplían la condición).

Es importante observar que con este algoritmo se puede calcular la precisión de las variables que se utilizan. No se debe confundir con el número mínimo almacenable. Cuando se sale del bucle la variable `a` tiene un valor menor que la precisión (1+`a` se confunde con 1) pero todavía

se mantiene por encima del número mínimo almacenable (`a` no se confunde con 0 y tiene sentido calcular `2*a` para obtener el valor de la precisión).

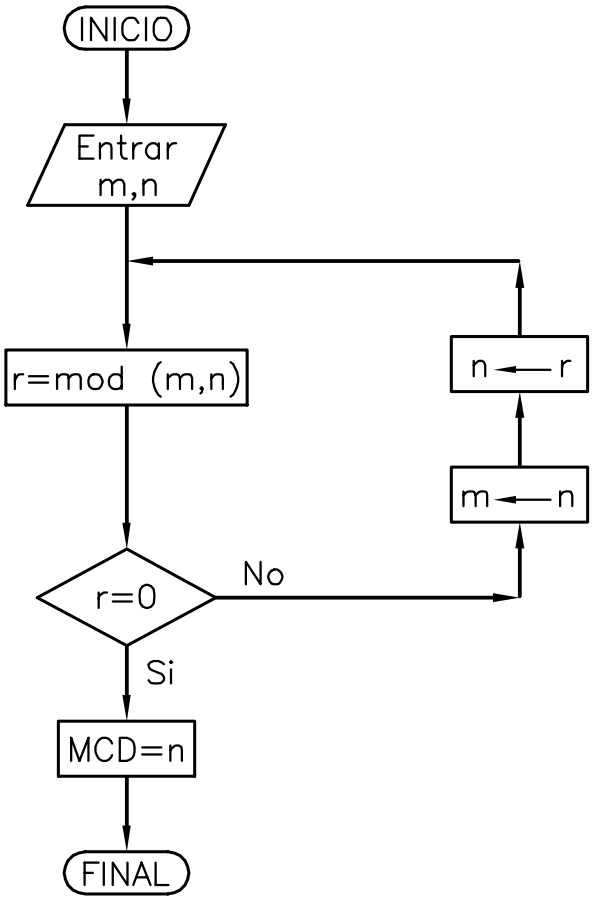


Fig. 4.2.1 Diagrama de flujo correspondiente al algoritmo de Euclides

Apartado b)

El programa 4.3.1 presenta el código correspondiente al algoritmo planteado, tanto en simple como en doble precisión.

En este programa se ha introducido la sentencia `DO WHILE`. Su sintaxis es

```
DO WHILE ( condición )
        bloque
ENDDO
```

```
c
c        Evaluacion de la precision de la maquina
c---------------------------------------------------------------------
      implicit real*4 (a-h)
      implicit real*8 (o-z)

c___Resolucion del problema utilizando variables REAL*4
      a = 1.e0
      b = 2.e0
      half=0.5e0
      i = 0
      do while (b .gt. 1.e0)
        a = a * half
        i = i + 1
        b = 1.e0 + a
      enddo
      write(6,*) ' '
      write(6,*) '   RESULTADOS CALCULADOS CON VARIABLES REAL*4 '
      write(6,*) '     mantisa   =',i-1
      write(6,*) '     precision =',2.e0**(1-i),2.e0*a

c___Resolucion del problema utilizando variables REAL*8
      x = 1.d0
      y = 2.d0
      zhalf=0.5d0
      i = 0
      do while (y .gt. 1.d0)
        x = x * zhalf
        i = i + 1
        y = 1.d0 + x
      enddo
      write(6,*) ' '
      write(6,*) '   RESULTADOS CALCULADOS CON VARIABLES REAL*8 '
      write(*,*) '     mantisa   =',i-1
      write(*,*) '     precision =',2.d0**(1-i),2.d0*x

      stop
      end
```

Prog. 4.3.1 Cálculo de la precisión mediante las potencias negativas del número 2

y se interpreta como: ejecutar reiteradamente el bloque de sentencias que hay entre la sentencia `DO WHILE` y la sentencia `ENDDO` mientras la *condición* sea verdadera. Por lo tanto, mientras `(b.gt.1.e0)` o `(y.gt.1.d0)` se multiplica `a` por `half`, se incrementa `i` en 1 y se actualiza el

valor de `b`.

En la tabla 4.3.1 se presentan los resultados obtenidos al ejecutar el programa en un ordenador personal. Los resultados obtenidos concuerdan con lo expuesto en el capítulo 4 (apartado 4.4.1).

Tabla 4.3.1 Resultados correspondientes al cálculo de las potencias negativas del número 2

```
RESULTADOS CALCULADOS CON VARIABLES REAL*4
  mantisa   =          23
  precision =  1.1920929E-07  1.1920929E-07

RESULTADOS CALCULADOS CON VARIABLES REAL*8
  mantisa   =          52
  precision =  2.2204460492503131E-16  2.2204460492503131E-16
```

Problema 4.4

Apartado a)

La figura 4.4.1 muestra el diagrama de las operaciones que deben realizarse para calcular el perímetro de la elipse. Así mismo, también se han numerado los resultados parciales y se han incluido los coeficientes que afectan a cada operación. El error en cada una de las operaciones es

$$
\begin{aligned}
R_1 &= r_a + r_a + r_1 = 2r_a + r_1 \\
R_2 &= r_b + r_b + r_2 = 2r_b + r_2 \\
R_3 &= \frac{a^2}{a^2+b^2} R_1 + \frac{b^2}{a^2+b^2} R_2 + r_3 \\
R_4 &= R_3 - \tilde{r}_2 + r_4 \\
R_5 &= \frac{1}{2} R_3 + r_5 \\
R_6 &= \tilde{r}_2 + r_\pi + r_6 \\
R_7 &= R_5 + R_6 + r_7
\end{aligned}
$$

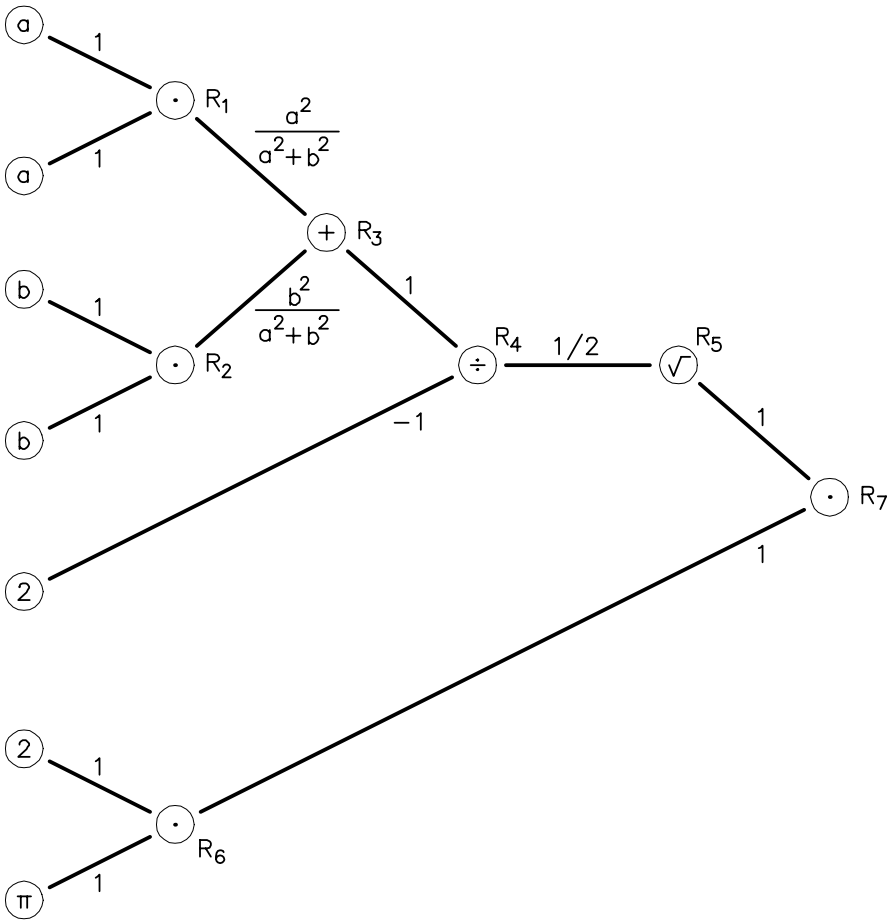


Fig. 4.4.1 Propagación del error en el cálculo del perímetro de la elipse

donde r_1, r_2, r_3, r_4, r_5, r_6 y r_7 representan el error de almacenamiento de cada una de las operaciones intermedias. En el resultado anterior hay que comentar dos resultados. Por una parte, el error de almacenamiento del número 2, $\tilde{r}_2$, es nulo, puesto que se trata de un numero entero. Por otra, el coeficiente de propagación del error debido a la raíz cuadrada, $f(x) = \sqrt{x}$, queda determinado por la expresión 4.15 del subapartado 4.5.6

$$r_{\sqrt{x}} = x\,\frac{f'(x)}{f(x)}\,r_x = x\,\frac{1/(2\sqrt{x})}{\sqrt{x}}r_x = \frac{1}{2}r_x$$

Si los errores inherentes se designan por r^i y los errores de almacenamiento por r^a, la expresión

del error relativo en P es

$$R_7 = \frac{1}{2}\left[\frac{a^2}{a^2+b^2}(2r_a + r_1) + \frac{b^2}{a^2+b^2}(2r_b + r_2) + r_3\right] + r_\pi + \frac{1}{2}r_4 + r_5 + r_6 + r_7$$

$$= \frac{1}{2}\left[\frac{a^2}{a^2+b^2}(2(r_a^i + r_a^a) + r_1) + \frac{b^2}{a^2+b^2}(2(r_b^i + r_b^a) + r_2) + r_3\right]$$

$$+ r_\pi + \frac{1}{2}r_4 + r_5 + r_6 + r_7$$

APARTADO B)

La cota del error del resultado final es

$$|R_7| \leq \frac{1}{2}\left[\frac{a^2}{a^2+b^2}(2(|r_a^i| + |r_a^a|) + |r_1|) + \frac{b^2}{a^2+b^2}(2(|r_b^i| + |r_b^a|) + |r_2|) + |r_3|\right]$$

$$+ |r_\pi| + \frac{1}{2}|r_4| + |r_5| + |r_6| + |r_7|$$

APARTADO C)

Puesto que el operario realiza las operaciones en base diez, coma flotante, utilizando tres dígitos para la mantisa (sin incluir el signo) y redondeando por aproximación, todos los errores relativos de almacenamiento son menores en valor absoluto que

$$r = \frac{1}{2}10^{1-3} = 0.005$$

Además, la cota del error inherente de los semiejes a y b es

$$|r_a^i|,\ |r_b^i| \leq r^i = 0.025$$

Por consiguiente, la cota del error total cometido por el operario es

$$|R_7| \leq \frac{1}{2}\left[\frac{a^2}{a^2+b^2}(2r^i + 3r) + \frac{b^2}{a^2+b^2}(2r^i + 3r) + r\right] + \frac{1}{2}r + 4r$$

$$= r^i + \frac{13}{2}r = 5.75 \cdot 10^{-2}$$

Por lo tanto, el operario puede cometer un error superior a los límites de diseño y no puede asegurar el cálculo correcto del perímetro. Sin embargo, la expresión anterior permite deducir la precisión que debe tener la cinta métrica para poder calcular el perímetro de la elipse con un error relativo inferior a $r_p = 5 \cdot 10^{-2}$. En efecto, debe verificarse que

$$r^i + \frac{13}{2}r \leq r_p$$

o equivalentemente

$$r^i \leq r_p - \frac{13}{2}r = 1.75 \cdot 10^{-2}$$

Problema 4.5

APARTADO a)

La figura 4.5.1 muestra los diagramas de propagación del error correspondientes a las tres alternativas.

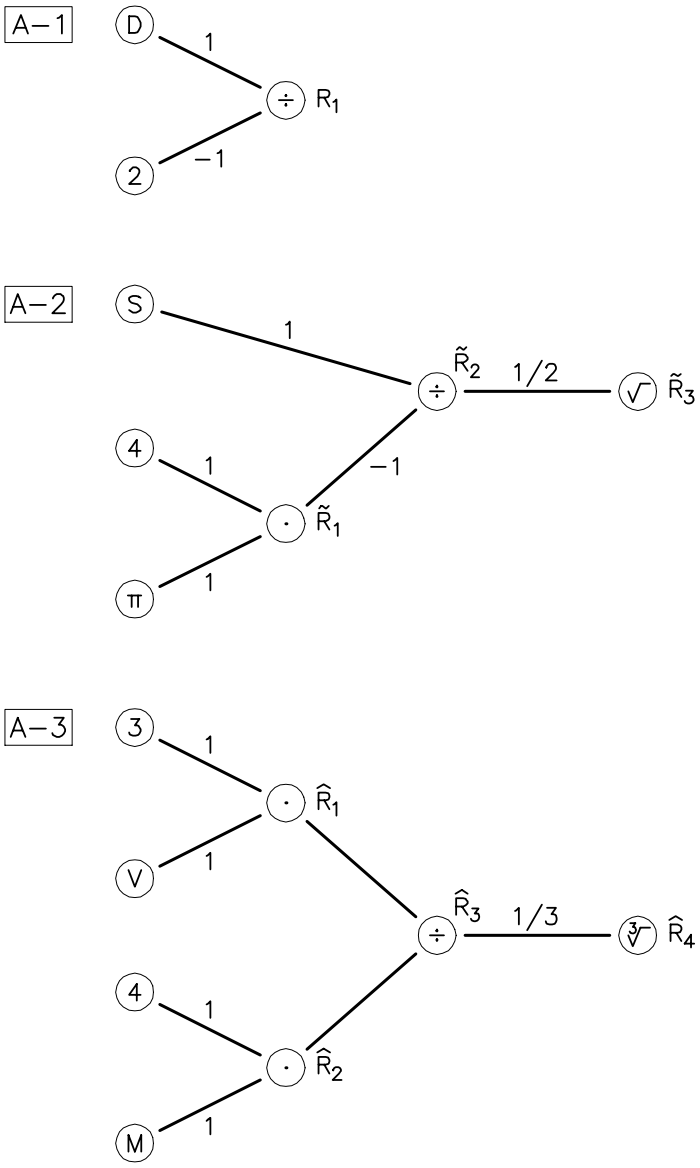


Fig. 4.5.1 Propagación del error en el cálculo del radio de la esfera

En las siguientes expresiones los superíndices ^{i} y ^{a} indicarán error inherente y error de almacenamiento respectivamente. La expresión del error final mediante la primera alternativa es

$$R_1 = r_D - r_2^a + r_1 = (r_D^i + r_D^a) - r_2^a + r_1$$

donde r_2^a es nulo por tratarse del error de almacenamiento de un número entero. El error de la segunda alternativa es

$$\begin{aligned}
\tilde{R}_1 &= r_4^a + r_\pi + \tilde{r}_1 \\
\tilde{R}_2 &= r_S - \tilde{R}_1 + \tilde{r}_2 \\
\tilde{R}_3 &= \frac{1}{2}\tilde{R}_2 + \tilde{r}_3 = \frac{1}{2}\left[(r_S^i + r_S^a) - r_\pi - \tilde{r}_1 + \tilde{r}_2\right] + \tilde{r}_3
\end{aligned}$$

donde r_4^a es nulo por tratarse del error de almacenamiento de un número entero y el cálculo del coeficiente de propagación del error debido a la raíz cuadrada se ha realizado como en el problema 4.4. Por último, el error de la tercera alternativa es

$$\begin{aligned}
\hat{R}_1 &= r_3^a + r_V + \hat{r}_1 \\
\hat{R}_2 &= r_4^a + r_\pi + \hat{r}_2 \\
\hat{R}_3 &= \hat{R}_1 - \hat{R}_2 + \hat{r}_3 \\
\hat{R}_4 &= \frac{1}{3}\hat{R}_3 + \hat{r}_4 = \frac{1}{3}\left[(r_V^i + r_V^a) + \hat{r}_1 - r_\pi - \hat{r}_2 + \hat{r}_3\right] + \hat{r}_4
\end{aligned}$$

Como en los casos anteriores, r_3^a y r_4^a son nulos por tratarse del error de almacenamiento de un número entero. Así mismo, el cálculo del coeficiente de propagación del error debido a la raíz tercera se ha realizado, como en el problema 4.4, a partir de la ecuación 4.15.

APARTADO B)

A partir de las expresiones anteriores, la cota del error para cada una de las alternativas es

$$\begin{aligned}
R_1 &= (|r_D^i| + |r_D^a|) + |r_1| \\
\tilde{R}_3 &= \frac{1}{2}\left[(|r_S^i| + |r_S^a|) + |r_\pi| + |\tilde{r}_1| + |\tilde{r}_2|\right] + |\tilde{r}_3| \\
\hat{R}_4 &= \frac{1}{3}\left[(|r_V^i| + |r_V^a|) + |\hat{r}_1| + |r_\pi| + |\hat{r}_2| + |\hat{r}_3|\right] + |\hat{r}_4|
\end{aligned}$$

APARTADO c)

Los errores inherentes de las medidas verifican

$$\left.\begin{array}{l} |r_D^i| \\ |r_S^i| \\ |r_V^i| \end{array}\right\} \leq r^i = 10^{-3}$$

Además, todos los errores de almacenamiento en valor absoluto son menores que

$$r = \frac{1}{2}2^{1-23} = 1.192 \cdot 10^{-7}$$

puesto que se trabaja mediante variables `REAL*4`. Por consiguiente, se cumple

$$|R_1| \leq r^i + 2r = 1.00024 \cdot 10^{-3}$$
$$|\tilde{R}_3| \leq \frac{1}{2}r^i + 3r = 5.0036 \cdot 10^{-4}$$
$$|\hat{R}_4| \leq \frac{1}{3}(r^i + 8r) = 3.3365 \cdot 10^{-4}$$

Por lo que el ingeniero sólo puede utilizar la tercera alternativa para calcular el radio A con la precisión requerida.

Problema 4.6

Sean x e y las distancias desde los anclajes hasta la intersección de la pila y el tablero. Entonces, la longitud del cable se puede calcular como (teorema de Pitágoras)

$$L = \sqrt{x^2 + y^2}$$

Haciendo un estudio de la propagación del error análogo al realizado en el problema 4.4, se puede obtener la expresión del error relativo en la longitud de los cables siguiente:

$$r_L = \frac{1}{2}\left[\frac{x^2}{x^2+y^2}(2r_x + r_1) + \frac{y^2}{x^2+y^2}(2r_y + r_2) + r_3\right] + r_4$$

donde r_1, r_2, r_3 y r_4 son los errores de almacenamiento producidos con cada una de las operaciones, y r_x, r_y son los errores en x e y respectivamente (que incluirán error inherente y error de almacenamiento en el caso más general). La cota del error es

$$|r_L| = \frac{1}{2}\left[\frac{x^2}{x^2+y^2}(2|r_x| + |r_1|) + \frac{y^2}{x^2+y^2}(2|r_y| + |r_2|) + |r_3|\right] + |r_4|$$

Si se trabaja con infinitas cifras significativas correctas, sólo intervienen en la propagación del error los errores inherentes en los datos x e y ($r_1 = \ldots = r_4 = 0$). Suponiendo que la cota del error inherente es la misma en ambos casos

$$r_x,\ r_y \leq r^i$$

la cota del error se reescribe como

$$|r_L| \leq \frac{1}{2}\left[\frac{x^2}{x^2+y^2}2r^i + \frac{y^2}{x^2+y^2}2r^i\right] = r^i$$

Un error absoluto de 0.25 m en L representa un error relativo de $0.25/100 = 2.5 \cdot 10^{-3}$. Por lo tanto, para asegurar la precisión deseada es necesario que la cota del error inherente verifique

$$r^i \leq 2.5 \cdot 10^{-3}$$

Es necesario tomar las medidas con la misma precisión que se desea en el resultado (0.25 % en este caso).

Si la posición de los anclajes se conoce exactamente, pues son números enteros, el error en los datos (r_x y r_y) es nulo, puesto que no hay error inherente y no se produce ningún error al almacenar un entero. Así mismo, x^2, y^2 y x^2+y^2 son números enteros, y pueden calcularse sin error de redondeo. El *único* error de redondeo se comete al calcular $\sqrt{x^2+y^2}$.

Si el ingeniero trabaja en base 10 y con dos dígitos de precisión (es decir, con dos cifras significativas correctas), la cota del error relativo de redondeo por aproximación es

$$r = \frac{1}{2}10^{-2} = 5 \cdot 10^{-3}$$

En estas condiciones la cota del error (debido exclusivamente al error de almacenamiento del resultado de la última operación –la raíz cuadrada–) es

$$|r_L| \leq r = 5 \cdot 10^{-3} > 2.5 \cdot 10^{-3}$$

y, por lo tanto, es posible que los cálculos del ingeniero no tengan la precisión necesaria y que los cables pedidos no sirvan.

10.4 Problemas del capítulo 5

Problema 5.1

APARTADO A)

Los ceros de la función $f(\theta)$ son cuatro: $\theta_1 = 0$, $\theta_2 = \pi/2$, $\theta_3 = \pi$ y $\theta_4 = 3\pi/2$, y corresponden a impactos de la bola P en las intersecciones de los ejes coordenados con la circunferencia. Conviene notar que la raíz $\theta_3 = \pi$, a pesar de ser solución del problema numérico, carece de sentido físico, ya que para este valor del ángulo la bola P chocaría con la bola Q antes de impactar con la banda de la mesa.

APARTADO B)

Las raíces $\theta_2 = \pi/2$, $\theta_3 = \pi$ y $\theta_4 = 3\pi/2$ se capturan tomando, por ejemplo, los intervalos iniciales $[1.3, 1.7]$, $[3.0, 3.3]$ y $[4.5, 4.9]$ respectivamente. Se puede comprobar fácilmente que se produce un cambio de signo de f en cada uno de los mencionados intervalos, condición suficiente para iniciar el método de la bisección. En la tabla 5.1.1 se muestran los resultados obtenidos para el primero de los casos citados.

Al intentar capturar la solución $\theta_1 = 0$ el programa genera un error de ejecución en la fase de control de convergencia, puesto que la evaluación de la condición 5.8 conduce a una división por 0 en el cálculo de r^k (ecuación 5.7). Para que el programa consiga capturar cualquiera de las raíces θ_i, sin excepción, es suficiente con adoptar la condición 5.9 en lugar de la 5.8. A nivel de programación, esto se lleva a cabo sustituyendo la línea correspondiente al cálculo de la variable `REL_X` por

```
abs_x = x - pmedio
```

y modificando la condición de control de convergencia

```
if ((abs(fx).lt.tol_f) .and. (abs(rel_x).lt.tol_x))
```

por la siguiente

```
      if ((abs(fx).lt.tol_f) .and.
     .       (abs(abs_x) .lt. (tol_x * abs(pmedio) + tol_e)))
```

donde `TOL_E` es la tolerancia E que aparece en la ecuación 5.9 y debe ser introducida por el usuario junto con los demás datos.

Como comentario final cabe comentar que, para este problema en particular y utilizando la

versión original del programa 5.1, no es posible capturar la raíz $\theta_1 = 0$, pero sí la raíz $\theta = 2\pi$, que es físicamente equivalente; esto se consigue inicializando el método de la bisección con el intervalo $[6.0, 6.4]$, por ejemplo.

Tabla 5.1.1 Resultados del problema del billar circular; método de la bisección con $x^0 = 1.3$, $a = 1.7$

```
Iteracion  Extremo a   Aproximacion x     f(x)     Error relativo en x
=========  ==========  ==============  ==========  ===================
    0       1.3000000     1.7000000    -5.848D-02       1.333D-01
    1       1.7000000     1.5000000     3.211D-02      -6.250D-02
    2       1.5000000     1.6000000    -1.326D-02       3.226D-02
    3       1.6000000     1.5500000     9.440D-03      -1.587D-02
    4       1.5500000     1.5750000    -1.908D-03       8.000D-03
    5       1.5750000     1.5625000     3.766D-03      -3.984D-03
    6       1.5750000     1.5687500     9.290D-04      -1.988D-03
    7       1.5687500     1.5718750    -4.897D-04       9.950D-04
    8       1.5718750     1.5703125     2.196D-04      -4.973D-04
    9       1.5703125     1.5710938    -1.350D-04       2.487D-04
   10       1.5710938     1.5707031     4.231D-05      -1.243D-04
   11       1.5707031     1.5708984    -4.635D-05       6.217D-05
   12       1.5707031     1.5708008    -2.022D-06       3.109D-05
   13       1.5708008     1.5707520     2.014D-05      -1.554D-05
   14       1.5708008     1.5707764     9.061D-06      -7.771D-06
   15       1.5708008     1.5707886     3.519D-06      -3.886D-06
   16       1.5708008     1.5707947     7.486D-07      -1.943D-06
   17       1.5707947     1.5707977    -6.368D-07       9.714D-07
   18       1.5707977     1.5707962     5.592D-08      -4.857D-07
   19       1.5707962     1.5707970    -2.904D-07       2.429D-07
   20       1.5707962     1.5707966    -1.173D-07       1.214D-07
   21       1.5707962     1.5707964    -3.067D-08       6.071D-08
   22       1.5707964     1.5707963     1.263D-08      -3.036D-08
   23       1.5707963     1.5707963    -9.020D-09       1.518D-08
   24       1.5707963     1.5707963     1.803D-09      -7.589D-09
   25       1.5707963     1.5707963    -3.608D-09       3.795D-09

Convergencia en la iteracion 25
Solucion para theta=  1.5707963
```

Problema 5.2

APARTADO A)

Efectivamente, el método de la bisección inicializado con $a = 2$, $x^0 = 0$ converge sin problemas. Los resultados obtenidos empleando el programa 5.1 con $tol_x = 0.5 \cdot 10^{-8}, TOL_f = 0.5 \cdot 10^{-8}$ se muestran en la tabla 5.2.1.

Tabla 5.2.1 Resultados del cálculo de $\sqrt{2}$ por el método de la bisección con $a = 2$, $x^0 = 0$

Iteracion	Extremo a	Aproximacion x	f(x)	Error relativo en x
0	2.0000000	0.0000000	-2.000D+00	-1.000D+00
1	2.0000000	1.0000000	-1.000D+00	-3.333D-01
2	1.0000000	1.5000000	2.500D-01	2.000D-01
3	1.5000000	1.2500000	-4.375D-01	-9.091D-02
4	1.5000000	1.3750000	-1.094D-01	-4.348D-02
5	1.3750000	1.4375000	6.641D-02	2.222D-02
6	1.4375000	1.4062500	-2.246D-02	-1.099D-02
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
27	1.4142136	1.4142136	-2.631D-08	-5.268D-09
28	1.4142136	1.4142136	-5.237D-09	-2.634D-09
29	1.4142136	1.4142136	5.300D-09	1.317D-09
30	1.4142136	1.4142136	3.154D-11	6.585D-10

```
Convergencia en la iteracion 30
La raiz cuadrada de  2.0000000 es  1.4142136
```

APARTADO B)

En la expresión del método de Newton (ecuación 5.14) puede verse que, para obtener la nueva aproximación a la solución x^{k+1}, es necesario dividir por la derivada de la función evaluada en la aproximación anterior, $f'(x^k)$. En este caso, dicha derivada tiene la expresión $f'(x^k) = 2\ x^k$. Esto quiere decir que, para $k = 0$, se obtendrá $f'(x^0) = 2\ x^0 = 0$. El programa produce un error de ejecución cuando, al intentar calcular x^1, se realiza una división por 0.

Gráficamente, la interpretación de este fenómeno es muy sencilla. Para obtener x^1, se debe trazar la recta tangente a la curva $y = f(x) = x^2 - 2$ por el punto $(x^0, f(x^0))$ y obtener su intersección con el eje de abscisas. Como $x^0 = 0$, dicha recta tangente tiene pendiente

$f'(x^0) = 0$, es decir, es una recta horizontal. Al no cortar esta recta al eje de las x es imposible obtener la nueva aproximación x^1.

Este comportamiento ilustra que, si bien el método de Newton, en general, es más rápido que el método de la bisección, a veces es también menos robusto.

Problema 5.3

APARTADO A)

El programa 5.3.1 permite hallar las raíces del problema del billar circular mediante el método de Newton. Este programa se ha obtenido por modificación del programa 5.2 (método de la bisección para el problema del billar circular –véase el capítulo 5–). Las únicas diferencias (reflejadas en el listado) entre estos dos programas son: 1) la entrada de datos (para el método de Newton sólo es necesaria una aproximación inicial), 2) el bloque donde se obtiene la nueva aproximación a la solución a partir de la aproximación calculada en la iteración anterior, y 3) las funciones externas de evaluación de la función f y de su derivada f'.

```
c
c        Metodo de Newton para el problema del billar circular
c_____________________________________________________________________

c___Aproximacion inicial x0
      write (6,200)
      read  (5,*) x_actual
 200  format (1x,'Aproximacion inicial')
           .
           .
           .
c___Nueva aproximacion
      fx = funcion(x_actual,r,xp,yp,xq,yq)
      dx = derivada(x_actual,r,xp,yp,xq,yq)
      x_nuevo = x_actual - (fx/dx)
           .
           .
           .
c___________________Function funcion(x)
      real*8 function funcion(x,r,xp,yp,xq,yq)
      implicit real*8 (a-h,o-z)

      sx=dsin(x)
      cx=dcos(x)

      distp=dsqrt( (r*cx - xp)*(r*cx - xp) +
     .             (r*sx - yp)*(r*sx - yp) )
```

```
      distq=dsqrt( (r*cx - xq)*(r*cx - xq) +
     .             (r*sx - yq)*(r*sx - yq) )
      funcion = (xp*sx - yp*cx)/distp + (xq*sx - yq*cx)/distq

      return
      end

c_____________________Function derivada(x)
      real*8 function derivada(x,r,xp,yp,xq,yq)
      implicit real*8 (a-h,o-z)

      sx=dsin(x)
      cx=dcos(x)

      dpx=(r*cx - xp)*(r*cx - xp)
      dpy=(r*sx - yp)*(r*sx - yp)
      dp =dpx+dpy
      distp=dsqrt(dp)

      sump=( (xp*cx + yp*sx) * distp -
     .       (xp*sx - yp*cx) *
     .       2.d0*r*((r*sx-yp)*cx - (r*cx-xp)*sx) /
     .       (2*distp)
     .     ) / dp

      dqx=(r*cx - xq)*(r*cx - xq)
      dqy=(r*sx - yq)*(r*sx - yq)
      dq=dqx+dqy
      distq=dsqrt(dq)
      sumq=( (xq*cx + yq*sx) * distq -
     .       (xq*sx - yq*cx) *
     .       2.d0*r*((r*sx-yq)*cx - (r*cx-xq)*sx) /
     .       (2*distq)
     .     ) / dq

      derivada = sump + sumq

      return
      end
```

Prog. 5.3.1 Método de Newton para el problema del billar circular

APARTADO B)

A modo de ejemplo, se muestran los resultados obtenidos al inicializar el método de Newton con $x^0 = 1.3$ (véase la tabla 5.3.1). Para este valor de la aproximación inicial se converge a la raíz $\theta_2 = \pi/2$. Los valores de las tolerancias son los mismos que para los ejemplos anteriores ($tol_x = 0.5 \cdot 10^{-8}, TOL_f = 0.5 \cdot 10^{-8}$).

Tabla 5.3.1 Resultados del problema del billar circular; método de Newton con $x^0 = 1.3$

```
Iteracion   Aproximacion x       f(x)       Error relativo en x
=========   ==============    ==========    ===================
    0          1.3000000       1.213D-01         1.765D-01
    1          1.5786010      -3.543D-03        -4.969D-03
    2          1.5707962       7.773D-08         1.090D-07
    3          1.5707963      -1.110D-16        -1.414D-16

Convergencia en la iteracion  3
El valor del angulo theta es  1.5707963
```

Problema 5.4

APARTADO A)

El programa 5.4.1 permite hallar las raíces del problema del billar circular mediante el método de la secante. Este programa se ha obtenido por modificación del programa 5.3.1 (método de Newton para el problema del billar circular). Las únicas diferencias (reflejadas en el listado) entre estos dos programas son: 1) la entrada de datos (ahora son necesarias dos aproximaciones iniciales), 2) el bloque donde se obtiene la nueva aproximación a la solución a partir de la aproximación calculada en la iteración anterior, y 3) las funciones externas `FUNCTION` (para el método de Newton es necesario realizar dos evaluaciones funcionales por iteración, una de f y otra de f', mientras que para el método de la secante es suficiente con evaluar f). Por lo que respecta al punto 3, sólo es necesario eliminar del programa 5.3.1 la función externa `DERIVADA(X)`.

APARTADO B)

A modo de ejemplo, se muestran los resultados obtenidos al inicializar el método de la secante con $x^0 = 1.3$ y $x^1 = 1.58$ (véase la tabla 5.4.1). Se ha escogido para x^1 la segunda aproximación calculada por el método de Newton (ver resolución del problema 5.3). De nuevo, se obtiene sin problemas la raíz $\theta_2 = \pi/2$. Los valores de las tolerancias son los mismos que para los ejemplos anteriores ($tol_x = 0.5 \cdot 10^{-8}, TOL_f = 0.5 \cdot 10^{-8}$).

```
c
c        Metodo de la secante para el problema del billar circular
c---------------------------------------------------------------------

c___Aproximaciones iniciales
      write (6,200)
      read  (5,*) x_actual
 200  format (1x,'Aproximacion inicial x0')

      write (6,300)
      read  (5,*) x_nuevo
 300  format (1x,'Aproximacion inicial x1')
          .
          .
          .
c___Nueva aproximacion
      f_nuevo   = f(x_nuevo,r,xp,yp,xq,yq)
      pendiente = (f_nuevo-f_actual)/(x_nuevo-x_actual)
      x_actual  = x_nuevo
      x_nuevo = x_actual - (f_nuevo/pendiente)
```

Prog. 5.4.1 Método de la secante para el problema del billar circular

Tabla 5.4.1 Resultados del problema del billar circular; método de la secante con $x^0 = 1.3, x^1 = 1.58$

```
Iteracion   Aproximacion x       f(x)       Error relativo en x
=========   ==============    ==========    ===================
    0          1.3000000       1.213D-01         1.772D-01
    1          1.5800000      -4.178D-03        -5.937D-03
    2          1.5706746       5.524D-05         7.747D-05
    3          1.5707963      -8.315D-10        -1.166D-09

Convergencia en la iteracion  3
El valor del angulo theta es  1.5707963
```

Para evitar un error de ejecución al intentar capturar la raíz $\theta_1 = 0$, es suficiente con introducir en el programa un criterio de convergencia en errores *absolutos* (véase la resolución del problema 5.2).

Problema 5.5

En la figura 5.5.1 se muestran las curvas de convergencia (número de iteración *versus* error relativo en escala logarítmica) correspondientes a los resultados de las tablas 5.1.1, 5.3.1 y 5.4.1 (raíz capturada: $\theta_2 = \pi/2$). Como se puede observar, el método de la bisección es el que presenta un decrecimiento más lento del error, precisando 25 iteraciones para llegar a convergencia. El método de Newton es el más rápido; para este método, el error relativo decrece mucho en muy pocas iteraciones (el criterio de convergencia se verifica ya en la tercera iteración). El método de la secante presenta un comportamiento intermedio entre los dos anteriores.

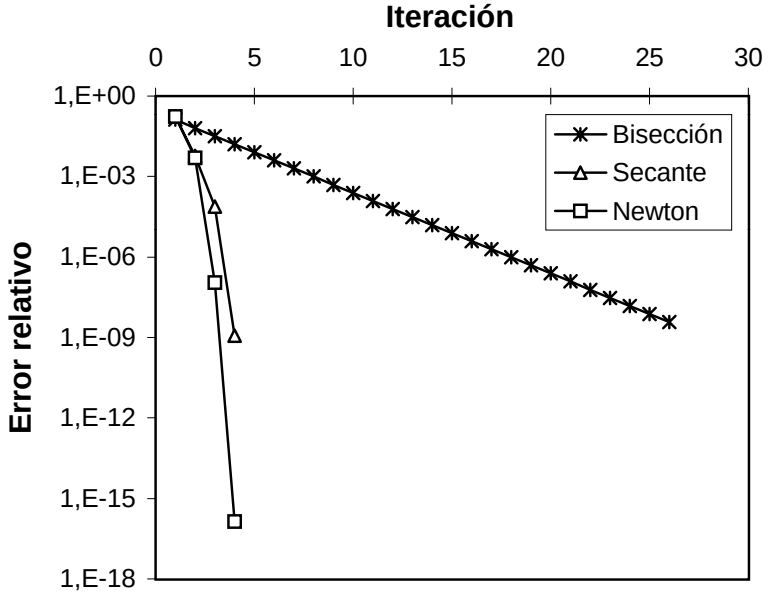


Fig. 5.5.1 Comparación de los tres métodos para el problema del billar circular

Problema 5.6

Los programas 5.1, 5.3 y 5.4 del apartado 5.7 utilizan respectivamente los métodos de la bisección, de Newton y de la secante. A partir de ellos puede confeccionarse el programa pedido en el enunciado.

La función $f(x) = x^2 - 2x + 1 = (x-1)^2$ tiene una única raíz $x^* = 1$. Esta raíz es doble y, por lo tanto, se verifica $f(x^*) = 0$ y $f'(x^*) = 0$. La función toma únicamente valores positivos (ver figura 5.6.1) y, en consecuencia, es imposible el cálculo de la raíz mediante el método de bisección.

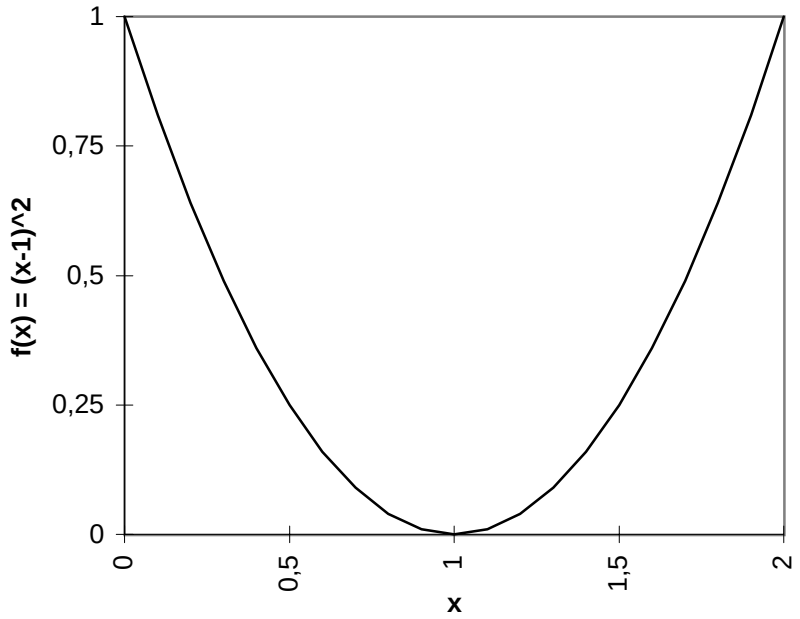


Fig. 5.6.1 Función $f(x) = (x-1)^2$

Trabajando con aproximación inicial $x^0 = 2$ y $tol_x = 0.5 \cdot 10^{-8}, TOL_f = 0.5 \cdot 10^{-8}$, se obtienen los resultados de las tablas 5.6.1 (método de Newton) y 5.6.2 (método de la secante). Para el método de la secante, se toma como aproximación x^1 la obtenida con el método de Newton.

Tabla 5.6.1 Cálculo de la raíz de $f(x) = (x-1)^2$ *mediante el método de Newton*

Iteracion	Aproximacion x	funcion(x)	Error relativo en x
0	2.0000000	1.000D+00	3.333D-01
1	1.5000000	2.500D-01	2.000D-01
2	1.2500000	6.250D-02	1.111D-01
3	1.1250000	1.563D-02	5.882D-02
	.		
	.		
25	1.0000000	8.882D-16	1.490D-08
26	1.0000000	2.220D-16	7.451D-09
27	1.0000000	0.000D+00	0.000D+00

Convergencia en la iteracion 27
La raiz es 1.0000000

Tabla 5.6.2 Cálculo de la raíz de $f(x) = (x-1)^2$ *mediante el método de la secante*

Iteracion	Aproximacion x	funcion(x)	Error relativo en x
0	2.0000000	1.000D+00	3.333D-01
1	1.5000000	2.500D-01	1.250D-01
2	1.3333333	1.111D-01	1.111D-01
3	1.2000000	4.000D-02	6.667D-02
	.		
	.		
35	1.0000000	1.776D-15	1.678D-08
36	1.0000000	6.661D-16	1.007D-08
37	1.0000000	2.220D-16	5.034D-09
38	1.0000000	0.000D+00	0.000D+00

```
Convergencia en la iteracion 38
La raiz es  1.0000000
```

La figura 5.6.2 muestra el error relativo (en escala logarítmica) en función del número de iteraciones.

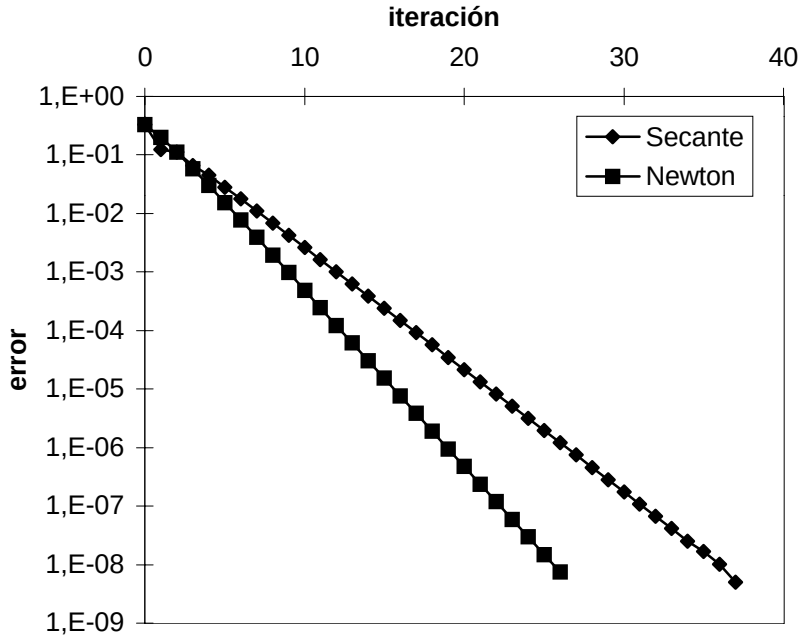


Fig. 5.6.2 Comparación de dos métodos para el cálculo de la raíz de $f(x) = (x-1)^2$

Los métodos de Newton y de la secante presentan típicamente una convergencia rápida (véase la figura 5.5.1 del problema anterior, por ejemplo). Para este problema, sin embargo, la convergencia es más lenta de lo habitual (tablas 5.6.1 y 5.6.2, figura 5.6.2). La razón es que $f(x) = (x-1)^2$ tiene una raíz *doble*. Con las sucesivas iteraciones se consigue una sucesión de valores x^k cada vez más cercanos a la raíz x^*. Al tratarse de una raíz doble, los valores de la derivada $f'(x^k)$ cada vez son más cercanos a $f'(x^*) = 0$ y, en consecuencia, al aplicar la definición del método de Newton, se divide por números muy cercanos a 0. Esto provoca que el método pierda su rapidez. En el caso del método de la secante ocurre algo similar. No existe el cálculo de la derivada, pero sí se calcula una aproximación de ésta que también se acercará a cero con cada iteración.

10.5 Problemas del capítulo 6

Problema 6.1

APARTADO A)

La fase de eliminación del algoritmo de Gauss sin pivotamiento se escribe (ecuación 6.18) como

$$\begin{aligned} a_{ij}^{(k)} &= a_{ij}^{(k-1)} - m_{ik}a_{kj}^{(k-1)} = a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}a_{kj}^{(k-1)} \\ b_{i}^{(k)} &= b_{i}^{(k-1)} - m_{ik}b_{k}^{(k-1)} = b_{i}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}b_{k}^{(k-1)} \end{aligned} \qquad \begin{cases} k = 1,\ldots,n-1 \\ i = k+1,\ldots,n \\ j = k+1,\ldots,n \end{cases}$$

En cada paso (para cada valor de k) se pasa de una matriz $\boldsymbol{A}^{(k-1)}$ a una matriz $\boldsymbol{A}^{(k)}$, anulando los elementos de la columna k-ésima por debajo de la diagonal. Al hacerlo, sólo se modifican los coeficientes de la submatriz llena de orden $n-k$ (filas y columnas $k+1$ hasta n), que queda (véase la ecuación 6.15)

$$\begin{pmatrix} a_{k+1,k+1}^{(k)} & \cdots & a_{k+1,n}^{(k)} \\ \vdots & \ddots & \vdots \\ a_{n,k+1}^{(k)} & \cdots & a_{nn}^{(k)} \end{pmatrix}$$

Sea $\boldsymbol{A} \equiv \boldsymbol{A}^{(0)}$ una matriz simétrica. Para aprovechar la simetría en el algoritmo de Gauss, basta tener en cuenta que estas submatrices llenas *se mantienen simétricas* durante toda la fase de eliminación. Por ejemplo, para $k = 1$, la submatriz de orden $n-1$

$$\begin{pmatrix} a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \ddots & \vdots \\ a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{pmatrix}$$

se calcula como (ecuación 6.12)

$$a_{ij}^{(1)} = a_{ij}^{(0)} - m_{i1}a_{1j}^{(0)} = a_{ij}^{(0)} - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}}a_{1j}^{(0)} \qquad \begin{cases} i = 2,\ldots,n \\ j = 2,\ldots,n \end{cases}$$

A partir de la simetría de $\boldsymbol{A}$ ($a_{ij}^{(0)} = a_{ji}^{(0)}$), puede deducirse directamente que $a_{ij}^{(1)} = a_{ji}^{(1)}$. En efecto,

$$a_{ji}^{(1)} = a_{ji}^{(0)} - \frac{a_{j1}^{(0)}}{a_{11}^{(0)}}a_{1i}^{(0)} = a_{ij}^{(0)} - \frac{a_{1j}^{(0)}}{a_{11}^{(0)}}a_{i1}^{(0)} = a_{ij}^{(1)} \qquad \begin{cases} i = 2,\ldots,n \\ j = 2,\ldots,n \end{cases}$$

La demostración se completa por inducción. A partir de $a_{ij}^{(k-1)} = a_{ji}^{(k-1)}$, se comprueba fácilmente que $a_{ij}^{(k)} = a_{ji}^{(k)}$.

Puesto que las submatrices llenas son simétricas, basta trabajar con los elementos de la submatriz triangular superior ($i \leq j$). Esto se consigue reescribiendo el algoritmo de Gauss (fase de eliminación) como

$$\begin{aligned} a_{ij}^{(k)} &= a_{ij}^{(k-1)} - m_{ik} a_{kj}^{(k-1)} = a_{ij}^{(k-1)} - \frac{a_{ki}^{(k-1)}}{a_{kk}^{(k-1)}} a_{kj}^{(k-1)} \\ b_{i}^{(k)} &= b_{i}^{(k-1)} - m_{ik} b_{k}^{(k-1)} = b_{i}^{(k-1)} - \frac{a_{ki}^{(k-1)}}{a_{kk}^{(k-1)}} b_{k}^{(k-1)} \end{aligned} \qquad \begin{cases} k = 1, \ldots, n-1 \\ i = k+1, \ldots, n \\ j = i, \ldots, n \end{cases}$$

Respecto del algoritmo original (matrices no necesariamente simétricas), se han introducido únicamente dos modificaciones: 1) en el cálculo de m_{ik} se emplea el elemento $a_{ki}^{(k-1)}$ del triángulo superior en lugar del elemento $a_{ik}^{(k-1)}$ (del mismo valor, dada la simetría) del triángulo inferior; 2) el índice de columna j va desde i (y no desde $k+1$) hasta n. Con estos dos cambios se utilizan solamente coeficientes de la matriz triangular superior.

Apartado b)

Al pasar de $\boldsymbol{A}^{(k-1)}$ a $\boldsymbol{A}^{(k)}$ con el algoritmo propuesto en el apartado anterior es necesario calcular $(n-k+1)(n-k)/2$ elementos $a_{ij}^{(k)}$ y $n-k$ componentes $b_i^{(k)}$. Para ello es necesario realizar las siguientes operaciones:

$$\begin{cases} (n-k)(n-k+3)/2 & \text{sumas} \\ (n-k)(n-k+3)/2 & \text{productos} \\ n-k & \text{divisiones} \end{cases}$$

Sumando para los $n-1$ pasos ($k = 1, \ldots, n-1$), se obtiene un total de

$$\begin{cases} \displaystyle\sum_{k=1}^{n-1} \frac{(n-k)(n-k+3)}{2} = \frac{n^3+3n^2-4n}{6} & \text{sumas} \\ \displaystyle\sum_{k=1}^{n-1} \frac{(n-k)(n-k+3)}{2} = \frac{n^3+3n^2-4n}{6} & \text{productos} \\ \displaystyle\sum_{k=1}^{n-1} n-k = \frac{n(n-1)}{2} & \text{divisiones} \end{cases}$$

para la fase de eliminación. Si se añaden las operaciones de la fase de sustitución hacia atrás (véase el subapartado 6.2.2), se obtiene que el número de operaciones necesarias para resolver un sistema lineal con matriz $\boldsymbol{A}$ simétrica (mediante el algoritmo de Gauss adaptado desarrollado el el apartado *a*) es $T_{GSim} = \frac{2n^3+15n^2-11n}{6}$. Tal y como era de esperar, este número es aproximadamente la *mitad* del correspondiente al método de Gauss estándar, $T_G = \frac{4n^3+9n^2-7n}{6}$. En efecto, en el caso general se hacen del orden de $\frac{2n^3}{3}$ operaciones, y en el caso simétrico del orden de $\frac{n^3}{3}$.

La simetría de la matriz puede aprovecharse también a efectos de almacenamiento, empleando el esquema propuesto en el subapartado 7.4.3. La idea básica es guardar solamente los coeficientes del triángulo superior. De esta forma, los requisitos de memoria para el caso simétrico son también aproximadamente la mitad que para el caso general.

En resumen, el coste computacional del método de Gauss sin pivotamiento para matrices simétricas es aproximadamente la mitad que para matrices no simétricas, tanto en número de operaciones como en espacio de memoria.

APARTADO C)

El algoritmo desarrollado en el apartado *a no* puede emplearse si es necesario pivotar. El pivotamiento (es decir, la permutación de filas) rompe la simetría de las submatrices llenas de orden $n-k$. Es necesario entonces emplear el algoritmo de Gauss con pivotamiento estándar, y almacenar la matriz $\boldsymbol{A}$ como matriz llena.

Problema 6.2

APARTADO A)

El vector auxiliar $\boldsymbol{y}$ se calcula resolviendo el sistema triangular inferior $\boldsymbol{Ly} = \boldsymbol{b}$ mediante una sustitución hacia adelante (ecuación 6.9):

$$y_1 = b_1 \;/\; l_{11}$$
$$y_i = \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right) \;/\; l_{ii} \qquad i = 2, \ldots, n$$

Nótese que la componente b_i (para $i = 1, \ldots, n$) sólo es necesaria para calcular la incógnita y_i. En consecuencia, una vez calculada y_i puede sobreescribirse encima de b_i (es decir, y_i puede almacenarse en la posición de memoria que ocupaba b_i).

Pueden hacerse las mismas consideraciones para el sistema triangular superior $\boldsymbol{Ux} = \boldsymbol{y}$ que permite obtener la solución $\boldsymbol{x}$ del sistema $\boldsymbol{Ax} = \boldsymbol{b}$. En la sustitución hacia atrás (ecuación 6.7),

$$x_n = y_n$$
$$x_i = y_i - \sum_{j=i+1}^{n} u_{ij} y_j \qquad i = n-1, n-2, \ldots, 1$$

puede sobrescribirse x_i encima de y_i. En conclusión, basta con reservar espacio de memoria para un único vector, que contiene los términos independientes $\boldsymbol{b}$ al principio, el vector auxiliar $\boldsymbol{y}$ luego y las incógnitas $\boldsymbol{x}$ al final. Desde luego, esta estrategia de ahorro de memoria sólo es válida en el caso (muy habitual) en que no sea necesario disponer del vector $\boldsymbol{b}$ en la memoria una vez resuelto el sistema lineal. Si se quisiera conservar $\boldsymbol{b}$, habría que trabajar con dos vectores (uno para $\boldsymbol{b}$, y otro para $\boldsymbol{y}$ y $\boldsymbol{x}$).

APARTADO B)

Se ha visto que las matrices $\boldsymbol{L}$ y $\boldsymbol{U}$ pueden guardarse encima de $\boldsymbol{A}$ (figura 6.2) y que los vectores $\boldsymbol{y}$ y $\boldsymbol{x}$ pueden guardarse encima de $\boldsymbol{b}$. Esto permite escribir la fase de descomposición del algoritmo de Crout como

$$
\begin{array}{l}
k = 1, \ldots, n-1 \\
\left\{ \begin{array}{lll}
a_{1,k+1} & = a_{1,k+1} \;/\; a_{11} & \\
a_{i,k+1} & = \left(a_{i,k+1} - \displaystyle\sum_{j=1}^{i-1} a_{ij}\; a_{j,k+1} \right) \;/\; a_{ii} & i = 2, \ldots, k \\
a_{k+1,i} & = a_{k+1,i} - \displaystyle\sum_{j=1}^{i-1} a_{ji}\; a_{k+1,j} & i = 2, \ldots, k \\
a_{k+1,k+1} & = a_{k+1,k+1} - \displaystyle\sum_{i=1}^{k} a_{k+1,i}\; a_{i,k+1} &
\end{array} \right.
\end{array}
$$

Estas ecuaciones se obtienen partiendo de la ecuación 6.38 y *1)* suprimiendo las asignaciones $l_{11} = a_{11}$ y $l_{k+1,1} = a_{k+1,1}$ (innecesarias, puesto que $\boldsymbol{L}$ se escribe encima de $\boldsymbol{A}$), *2)* suprimiendo las asignaciones $u_{ii} = 1$ (los unos de la diagonal de $\boldsymbol{U}$ no se almacenan) y *3)* reemplazando l_{ij} y u_{ij} por a_{ij} en las expresiones restantes.

En cuanto a la fase de sustituciones, se obtiene

$$
\begin{array}{ll}
b_1 = b_1 \;/\; a_{11} & \\
b_i = \left(b_i - \displaystyle\sum_{j=1}^{i-1} a_{ij} b_j \right) \;/\; a_{ii} & i = 2, \ldots, n \\
b_i = b_i - \displaystyle\sum_{j=i+1}^{n} a_{ij} b_j & i = n-1, n-2, \ldots, 1
\end{array}
$$

Estas ecuaciones se obtienen partiendo de las vistas en el apartado *a* y *1)* reemplazando l_{ij} y u_{ij} por a_{ij}, *2)* reemplazando x_i e y_i por b_i, y *3)* suprimiendo la asignación (innecesaria) $x_n = y_n$.

Problema 6.3

Apartado a)

Siguiendo el procedimiento visto para el método de Crout (ecuación 6.35), la descomposición del menor $\boldsymbol{A}_{[k+1]}$ puede escribirse como

$$\boldsymbol{A}_{[k+1]} = \boldsymbol{L}_{[k+1]}\,\boldsymbol{D}_{[k+1]}\,\boldsymbol{U}_{[k+1]}$$

$$\begin{pmatrix} \boldsymbol{A}_{[k]} & \boldsymbol{c}_{[k+1]} \\ \boldsymbol{f}^T_{[k+1]} & a_{k+1,k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{L}_{[k]} & \boldsymbol{0} \\ l^T_{[k+1]} & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{D}_{[k]} & \boldsymbol{0} \\ \boldsymbol{0}^T & d_{k+1,k+1} \end{pmatrix} \begin{pmatrix} \boldsymbol{U}_{[k]} & \boldsymbol{u}_{[k+1]} \\ \boldsymbol{0}^T & 1 \end{pmatrix}$$

A partir de esta expresión, se obtienen las siguientes ecuaciones para calcular los vectores $\boldsymbol{l}_{[k+1]}$ y $\boldsymbol{u}_{[k+1]}$ y el escalar $d_{k+1,k+1}$:

$$\begin{aligned} \boldsymbol{L}_{[k]}\,\boldsymbol{D}_{[k]}\,\boldsymbol{u}_{[k+1]} &= \boldsymbol{c}_{[k+1]} \\ \boldsymbol{U}^T_{[k]}\,\boldsymbol{D}_{[k]}\,\boldsymbol{l}_{[k+1]} &= \boldsymbol{f}_{[k+1]} \\ d_{k+1,k+1} &= a_{k+1,k+1} - \boldsymbol{l}^T_{[k+1]}\boldsymbol{D}_{[k]}\boldsymbol{u}_{[k+1]} \end{aligned}$$

Para calcular el vector $\boldsymbol{u}_{[k+1]}$, hay que resolver primero un sistema triangular inferior $\boldsymbol{L}_{[k]}\boldsymbol{v}_{[k+1]} = \boldsymbol{c}_{[k+1]}$ (donde $\boldsymbol{v}_{[k+1]}$ es un vector auxiliar) y luego un sistema diagonal $\boldsymbol{D}_{[k]}\boldsymbol{u}_{[k+1]} = \boldsymbol{v}_{[k+1]}$. Nótese, sin embargo, que no es necesario reservar espacio de memoria para el vector auxiliar $\boldsymbol{v}_{[k+1]}$, porque puede aprovecharse el espacio destinado para el vector $\boldsymbol{u}_{[k+1]}$. Pueden hacerse consideraciones muy similares respecto del cálculo del vector $\boldsymbol{l}_{[k+1]}$. Así pues, la descomposición $\boldsymbol{LDU}$ puede calcularse como con las siguientes ecuaciones (análogas a las ecuaciones 6.38 vistas para la descomposición $\boldsymbol{LU}$):

$$l_{11} = 1 \qquad d_{11} = a_{11} \qquad u_{11} = 1$$

$$k = 1, \ldots, n-1$$

$$\left\{\begin{array}{lll}
u_{1,k+1} = a_{1,k+1} & & \\
u_{i,k+1} = a_{i,k+1} - \displaystyle\sum_{j=1}^{i-1} l_{ij}\; u_{j,k+1} & & i = 2, \ldots, k \\
u_{i,k+1} = u_{i,k+1} \;/\; d_{ii} & & i = 1, \ldots, k \\
 & & \\
l_{k+1,1} = a_{k+1,1} & & \\
l_{k+1,i} = a_{k+1,i} - \displaystyle\sum_{j=1}^{i-1} u_{ji}\; l_{k+1,j} & & i = 2, \ldots, k \\
l_{i,k+1} = l_{i,k+1} \;/\; d_{ii} & & i = 1, \ldots, k \\
 & & \\
l_{k+1,k+1} = 1 & & \\
u_{k+1,k+1} = 1 & & \\
d_{k+1,k+1} = a_{k+1,k+1} - \displaystyle\sum_{i=1}^{k} l_{k+1,i}\; d_{ii}\; u_{i,k+1} & &
\end{array}\right.$$

Existen implementaciones más eficientes de este algoritmo. Golub y Van Loan (véanse las referencias del apartado 7.5), por ejemplo, proponen una versión basada en recorrer las matrices triangulares por columnas (y no por filas) al realizar las sustituciones hacia adelante, tal y como se comenta en el subapartado 7.2.2.

Al igual que el método de Crout (véase problema 6.2), este método puede implementarse reservando espacio para una única matriz, que contiene la matriz $\boldsymbol{A}$ al principio y los factores $\boldsymbol{L}$, $\boldsymbol{U}$ (sin la diagonal, por ser unitaria) y $\boldsymbol{D}$ después de la descomposición.

Apartado b)

El proceso es el mismo que en el apartado anterior. La descomposición del menor $\boldsymbol{A}_{[k+1]}$ es

$$\boldsymbol{A}_{[k+1]} = \boldsymbol{L}_{[k+1]}\, \boldsymbol{D}_{[k+1]}\, \boldsymbol{L}_{[k+1]}^T$$

$$\begin{pmatrix} \boldsymbol{A}_{[k]} & \boldsymbol{f}_{[k+1]} \\ \boldsymbol{f}_{[k+1]}^T & a_{k+1,k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{L}_{[k]} & \boldsymbol{0} \\ \boldsymbol{l}_{[k+1]}^T & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{D}_{[k]} & \boldsymbol{0} \\ \boldsymbol{0}^T & d_{k+1,k+1} \end{pmatrix} \begin{pmatrix} \boldsymbol{L}_{[k]}^T & \boldsymbol{l}_{[k+1]} \\ \boldsymbol{0}^T & 1 \end{pmatrix}$$

que lleva a las siguientes ecuaciones para el cálculo del vector $\boldsymbol{l}_{[k+1]}$ y del escalar $d_{k+1,k+1}$:

$$\begin{aligned} \boldsymbol{L}_{[k]}\, \boldsymbol{D}_{[k]}\, \boldsymbol{l}_{[k+1]} &= \boldsymbol{f}_{[k+1]} \\ d_{k+1,k+1} &= a_{k+1,k+1} - \boldsymbol{l}_{[k+1]}^T \boldsymbol{D}_{[k]} \boldsymbol{l}_{[k+1]} \end{aligned}$$

Detallando las operaciones necesarias, se obtiene el siguiente algoritmo:

$$\begin{aligned} & l_{11} = 1 \qquad d_{11} = a_{11} \\ & k = 1, \ldots, n-1 \\ & \quad \begin{cases} l_{k+1,1} &= a_{k+1,1} \\ l_{k+1,i} &= a_{k+1,i} - \displaystyle\sum_{j=1}^{i-1} l_{ij}\, l_{k+1,j} \qquad i = 2, \ldots, k \\ l_{k+1,i} &= l_{k+1,i} \;/\; d_{ii} \qquad i = 1, \ldots, k \\ l_{k+1,k+1} &= 1 \\ d_{k+1,k+1} &= a_{k+1,k+1} - \displaystyle\sum_{i=1}^{k} d_{ii} l_{k+1,i}^2 \end{cases} \end{aligned}$$

10.6 Problemas del capítulo 7

Problema 7.1

Las modificaciones realizadas en el programa 7.2 para los casos en que $x_i = 10^{-i+1}$ y $x_i = i$ están respectivamente en los programas 7.1.1 y 7.1.2. Tan sólo ha sido necesario modificar parte de la subrutina `get_mat_vec`. La única consideración destacable es que las operaciones de asignación de valores a las componentes del vector $\boldsymbol{x}$ deben convertir correctamente el valor de las variables enteras en reales de doble precisión. La tabla 7.1.1 muestra el fichero de resultados para el caso en que $x_i = 10^{-i+1}$ y $n = 4$, y la tabla 7.1.2 presenta los resultados para $x_i = i$ y $n = 4$.

```
c____________________Lectura o generacion de la matriz y del vector
      subroutine get_mat_vec (n,a,v)
      implicit real*8 (a-h,o-z)
      dimension a(n,n),v(n)

c___Generacion de la matriz (matriz de Vandermonde)
      do 10 i=1, n
        s=10.d0**(1-i)
        do 10 j=1, n
         a(i,j)= s**(j-1)
   10 continue
```

Prog. 7.1.1 Modificación de la subrutina `get_mat_vec` *para el caso* $x_i = 10^{-i+1}$

```
c____________________Lectura o generacion de la matriz y del vector
      subroutine get_mat_vec (n,a,v)
      implicit real*8 (a-h,o-z)
      dimension a(n,n),v(n)

c___Generacion de la matriz (matriz de Vandermonde)
      do 10 i=1, n
        s=dfloat(i)
        do 10 j=1, n
         a(i,j)= s**(j-1)
   10 continue
```

Prog. 7.1.2 Modificación de la subrutina `get_mat_vec` *para el caso* $x_i = i$

Tabla 7.1.1 Fichero de resultados del caso $x_i = 10^{-i+1}$

```
La matriz de entrada es:

  1.000000E+00   1.000000E+00   1.000000E+00   1.000000E+00
  1.000000E+00   1.000000E-01   1.000000E-02   1.000000E-03
  1.000000E+00   1.000000E-02   1.000000E-04   1.000000E-06
  1.000000E+00   1.000000E-03   1.000000E-06   1.000000E-09

El vector de entrada es:

  4.000000E+00
  1.111000E+00
  1.010101E+00
  1.001001E+00

El vector producto es:

  7.122102E+00
  4.122202E+00
  4.011212E+00
  4.001112E+00

El modulo del vector producto es:   9.990776E+00
```

Problema 7.2

APARTADO A)

El programa 7.2.1 sirve para resolver el sistema lineal pentadiagonal y simétrico planteado. El programa está realizado con dimensionamiento dinámico y consta de las subrutinas indicadas en el enunciado.

Se puede observar en el programa principal que la solución del sistema, $\boldsymbol{x}$, se almacena en la misma posición que ocupa el vector de términos independientes, $\boldsymbol{b}$. Esto se puede realizar porque el vector $\boldsymbol{b}$ no se necesita para realizar cálculos tras la resolución del problema, y de esta forma se ahorra espacio de memoria.

La matriz $\boldsymbol{A}$ se almacena mediante un esquema de almacenamiento óptimo por diagonales. Al ser $\boldsymbol{A}$ simétrica se ha considerado solamente la diagonal principal y las dos diagonales superiores. De esta forma se requieren un total de $3n$ posiciones de memoria para almacenar la matriz, cantidad muy inferior a las n^2 posiciones necesarias si se almacena como como matriz

llena. Este almacenamiento óptimo se puede realizar gracias a que en la resolución del sistema no es necesario pivotar.

Tabla 7.1.2 Fichero de resultados del caso $x_i = i$

```
La matriz de entrada es:

  1.000000E+00   1.000000E+00   1.000000E+00   1.000000E+00
  1.000000E+00   2.000000E+00   4.000000E+00   8.000000E+00
  1.000000E+00   3.000000E+00   9.000000E+00   2.700000E+01
  1.000000E+00   4.000000E+00   1.600000E+01   6.400000E+01

El vector de entrada es:

  4.000000E+00
  1.500000E+01
  4.000000E+01
  8.500000E+01

El vector producto es:

  1.440000E+02
  8.740000E+02
  2.704000E+03
  6.144000E+03

El modulo del vector producto es:   6.770891E+03
```

El método de Gauss ha sido adaptado al almacenamiento óptimo definido para la matriz $\boldsymbol{A}$. La adaptación se ha realizado en dos pasos. En primer lugar se ha adaptado el algoritmo a matrices en banda con semianchos superior e inferior iguales a 2. Este primer paso consiste en modificar los dos bucles internos en la fase de eliminación (con contadores `i` y `j`) de forma que sólo recorran los elementos necesarios (en ambos casos de `k+1` a `k+2`), y en separar las operaciones que corresponden a `k=n-1` del bucle principal. El segundo paso consiste en considerar que la matriz es simétrica, y por lo tanto es suficiente trabajar con la parte triangular superior. Este segundo paso implica que, para matrices pentadiagonales, sólo es necesario realizar operaciones de fila sobre tres coeficientes de la matriz para cada valor de `k`, con `k` de `1` a `n-2`, y por tanto se ha sustituido el bucle en `i` por la asignación directa `i=k+1` e `i=k+2`. Respecto de la sustitución hacia atrás sólo es importante resaltar que los bucles se han modificado de forma adecuada, respetando el almacenamiento de la matriz. Por último, cabe indicar que se ha utilizado la función `KPOS` para relacionar la notación matricial de $\boldsymbol{A}$ con el almacenamiento vectorial utilizado.

```
c
c          Este programa soluciona el sistema de ecuaciones
c                        A*x = b
c          para el caso en que A es una matriz pentadiagonal
c          simetrica y utilizando dimensionamiento dinamico
c_____________________________________________________________

      implicit real*8 (a-h,o-z)
      parameter(mtot=3997)
      dimension dd(mtot)

      call lecdat(n)
      call punter(n,na,nb,mtot)
      call gendat(dd(na),dd(nb),n)
      call gaus5s(dd(na),dd(nb),n)
      call escres(dd(nb),n)
      end

c____________________Lectura de datos
      subroutine lecdat(n)
      implicit real*8 (a-h,o-z)
      common /datos/ act,bct,ect,yoct,pact,pbct

      open(unit=10,file='pentadiagonal.dat',status='old')
      read(10,'(I5)') n
      read(10,'(5(E12.6,/),E12.6)')act,bct,ect,yoct,pact,pbct
      close(10)
      return
      end

c____________________Definicion de punteros
      subroutine punter(n,na,nb,mtot)
      implicit real*8 (a-h,o-z)

      na  =1
      nb  =na+3*n
      nend=nb+n
      if (nend.gt.mtot) then
         write(*,*)' Dimensionamiento insuficiente'
         write(*,*)' se requieren',nend,' posiciones'
         stop
      endif
      return
      end

c____________________Generacion de la matriz y el vector
      subroutine gendat(a,b,n)
      implicit real*8 (a-h,o-z)
      dimension a(5*n),b(n)
      common /datos/ act,bct,ect,yoct,pact,pbct
```

```
c___Generar Matriz A

      a(1)    = 5.D0
      a(n+1)  =-4.D0
      a(2*n+1)= 1.D0
      do i=2,n-2
         a(    i)= 6.D0
         a(  n+i)=-4.D0
         a(2*n+i)= 1.D0
      enddo
      a(n-1)  = 6.D0
      a(2*n-1)=-4.D0
      a(n)    = 5.D0

c___Generar Vector b

      h   =(bct-act)/dfloat(n+1)
      aux1=h**4/(ect*yoct)*pact
      aux2=h**5/(ect*yoct)*(pbct-pact)/(bct-act)
      do i=1,n
         b(i)=aux1+aux2*dfloat(i)
      enddo
      return
      end

c____________________Algoritmo de Gauss adaptado
      subroutine gaus5s(a,b,n)
      implicit real*8 (a-h,o-z)
      dimension a(3*n), b(n)
      kpos(i,j) = n*(j-i)+i

c___Eliminacion

      do k=1,n-2
          i=k+1
          fact=a(kpos(k,i))/a(kpos(k,k))
          b(i)=b(i)-fact*b(k)
          do j=k+1,k+2
            a(kpos(i,j))=a(kpos(i,j))-fact*a(kpos(k,j))
          enddo
          i=k+2
          fact=a(kpos(k,i))/a(kpos(k,k))
          b(i)=b(i)-fact*b(k)
          j=k+2
          a(kpos(i,j))=a(kpos(i,j))-fact*a(kpos(k,j))
      enddo
      fact=a(kpos(n-1,n))/a(kpos(n-1,n-1))
      b(n)=b(n)-fact*b(n-1)
      a(kpos(n,n))=a(kpos(n,n))-fact*a(kpos(n-1,n))
```

```
c___Sustitucion hacia atras

      b(n)=b(n)/a(kpos(n,n))
      b(n-1)=(b(n-1)-a(kpos(n-1,n))*b(n))/a(kpos(n-1,n-1))
      do i=n-2,1,-1
        do j=i+1,i+2
          b(i)=b(i)-a(kpos(i,j))*b(j)
        enddo
        b(i)=b(i)/a(kpos(i,i))
      enddo
      return
      end

c____________________Escritura de los resultados
      subroutine escres(x,n)
      implicit real*8 (a-h,o-z)
      dimension x(n)

      open(unit=11,file='t7_p73.res',status='unknown')
      write(11,'(6x,a3)'       )' x '
      write(11,'(<n>(E15.9,/))')(x(i),i=1,n)
      close(11)
      return
      end
```

Prog. 7.2.1 Resolución de sistemas lineales pentadiagonales simétricos

Apartado b)

El programa 7.2.1 se emplea para resolver sistemas lineales pentadiagonales con los datos indicados. En la tabla 7.2.1 se muestran los resultados obtenidos con $n = 5$.

Tabla 7.2.1 Resultado para $n = 5$

```
       x
 .675154321E-02
 .115740741E-01
 .133101852E-01
 .115740741E-01
 .675154321E-02
```

APARTADO c)

En la figura 7.2.1 se muestra la gráfica realizada en Excel en la que se presentan las tres series de pares ordenados $\{t_i, x_i\}$, con $i = 0, .., n+1$, correspondientes a $n = 5$, 19 y 99.

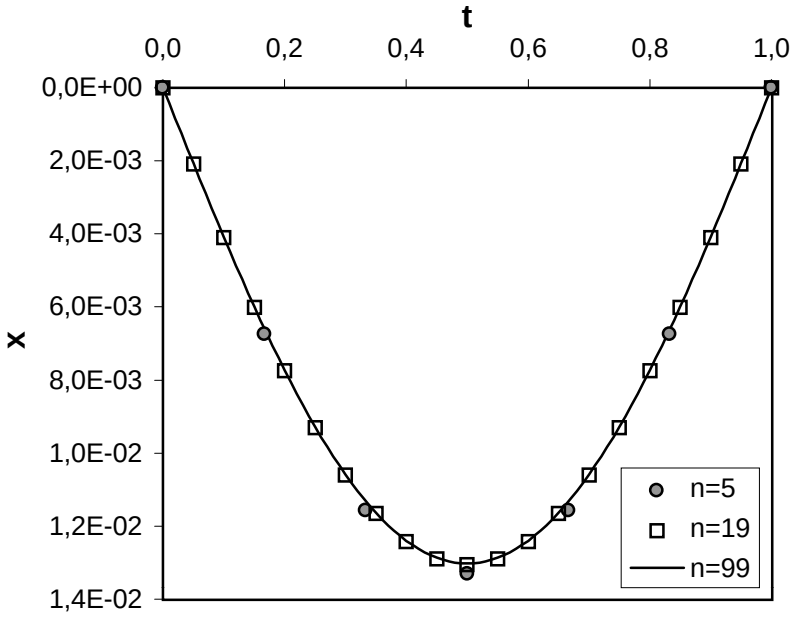


Fig. 7.2.1 Gráfica de las tres series de pares ordenados $\{t_i, x_i\}$, *con* $i = 0, .., n+1$, *correspondientes a* $n = 5$, 19 *y* 99

Para interpretar los resultados obtenidos es necesario explicar brevemente el problema físico asociado al sistema lineal de ecuaciones del enunciado. Este sistema se ha obtenido en la resolución numérica de la ecuación diferencial ordinaria que modela la deformada de una viga bajo una carga repartida. Los pares ordenados $\{t_i, x_i\}$ corresponden a una serie de puntos uniformemente distribuidos según el eje de la viga, $\{t_i\}$, y sus respectivas flechas (desplazamientos verticales), $\{x_i\}$. Cuanto más fina es la partición según el eje de la viga (por tanto, con n mayor), más correcta es la aproximación de la deformada obtenida. Esto se debe a dos motivos: porque al tener más puntos se puede representar mejor la deformada, y porque al aumentar n cada uno de los puntos se obtiene con una precisión mayor.

Con los datos del enunciado se calcula la deformada de una viga de longitud unidad bajo una carga uniforme de valor unidad. En el gráfico de la figura 7.2.1 se observa cómo las soluciones para $n = 19$ y $n = 99$ son prácticamente iguales, mientras que con $n = 5$ los puntos quedan un poco desplazados.

Por último, puesto que la ecuación de la deformada de una viga bajo carga uniforme es conocida, se puede comparar la solución numérica obtenida mediante la solución del sistema de ecuaciones con la solución analítica. La flecha en el punto medio de la viga es

$$x(t = \frac{b-a}{2}) \;=\; \frac{5}{384}\,\frac{p(b-a)^4}{EI}$$

Se puede verificar que el error relativo cometido con $n = 99$ es del orden de 10^{-4}, y con $n = 999$ del orden de 10^{-6}.

Problema 7.3

En la resolución de este problema se considera que se trabaja con números reales en doble precisión y que el vector $\boldsymbol{x}$ se almacena en el mismo sitio que el vector $\boldsymbol{b}$. En primer lugar se compara el número de reales que es necesario almacenar en cada alternativa. Posteriormente se compara el número de operaciones en coma flotante (por tanto, entre variables reales) que se realizan, y finalmente se concretan los resultados para $n = 10$, 100 y 1000.

Respecto de la primera alternativa (apartado *a*), la matriz $\boldsymbol{A}$ considerada como *matriz llena* ocupa un total de n^2 posiciones de memoria, y los vectores $\boldsymbol{b}$ y $\boldsymbol{x}$ ocupan en total n posiciones más. Por tanto es necesario almacenar $n^2 + n$ números reales en doble precisión. En la segunda alternativa (apartado *b*) se ha considerado que la matriz $\boldsymbol{A}$ esta almacenada de forma óptima por diagonales. Por ser una matriz simétrica es suficiente almacenar las tres diagonales superiores; por tanto necesita $3n$ posiciones de memoria. Los vectores $\boldsymbol{b}$ y $\boldsymbol{x}$ ocupan la misma cantidad de memoria que en la primera alternativa. Por tanto basta almacenar $4n$ números reales en doble precisión.

Respecto del número de operaciones entre variables reales que se realizan, hay que considerar las dos fases del método de Gauss: eliminación y sustitución hacia atrás. Como se ha visto en el capítulo 6, la alternativa *a* requiere un total de $(4n^3 + 9n^2 - 7n)/6$ operaciones en coma flotante. En cambio, la alternativa *b* requiere tan sólo $17n - 25$ operaciones. Para simplificar el cálculo de este número de operaciones, se presenta en el programa 7.3.1 una versión compacta de la subrutina `gaus5s` del programa 7.2.1. Ambas subrutinas corresponden al mismo algoritmo de Gauss adaptado a matrices pentadiagonales. En la nueva versión (programa 7.3.1) es más difícil ver la correspondencia con las distintas partes del algoritmo de Gauss estándar, pero es más sencillo contar el número de operaciones en coma flotante realizadas, y obtener el resultado indicado.

En la tabla 7.3.1 se puede observar cómo al aumentar n las diferencias entre ambas alternativas son cada vez más elevadas. El coste computacional de la alternativa *a* desaconseja claramente su uso para valores de n del orden de 100 o superiores.

```
      subroutine gaus5s_b(a,b,n)
      implicit real*8 (a-h,o-z)
      dimension a(3*n), b(n)

c___Eliminacion

      do k=1,n-2
          fact    =a(n+k)/a(k)
          b(k+1)  =b(k+1)-fact*b(k)
          a(k+1)  =a(k+1)-fact*a(n+k)
          a(n+k+1)=a(n+k+1)-fact*a(2*n+k)
          fact  =a(2*n+k)/a(k)
          b(k+2)=b(k+2)-fact*b(k)
          a(k+2)=a(k+2)-fact*a(2*n+k)
      enddo
      fact=a(2*n-1)/a(n-1)
      b(n)=b(n)-fact*b(n-1)
      a(n)=a(n)-fact*a(2*n-1)

c___Sustitucion hacia atras

      b(n)=b(n)/a(n)
      b(n-1)=(b(n-1)-a(2*n-1)*b(n))/a(n-1)
      do i=n-2,1,-1
        b(i)=(b(i)-a(n+i)*b(i+1)-a(2*n+i)*b(i+2))/a(i)
      enddo
      return
      end
```

Prog. 7.3.1 Modificación de la subrutina `gaus5s` *del programa 7.2.1*

Tabla 7.3.1 Comparación del coste computacional de las dos alternativas definidas en el problema 7.3

	Alternativa a		Alternativa b	
	Núm. operaciones	Memoria	Núm. operaciones	Memoria
General	$(4n^3+9n^2-7n)/6$	n^2+n	$17n-25$	$4n$
$n=10$	638	880 b	145	320 b
$n=100$	$682\cdot 10^3$	79 Kb	1675	13 Kb
$n=1000$	$668\cdot 10^6$	7.6 Mb	$17\cdot 10^3$	31 Kb

Problema 7.4

APARTADO A)

La matriz tridiagonal del sistema es diagonalmente dominante: para cada fila, el coeficiente diagonal es 2, y la suma de los dos coeficientes no diagonales $h_i/(h_i+h_{i-1})$ y $h_{i-1}/(h_i+h_{i-1})$ es 1. Por este motivo, el pivotamiento no es necesario.

APARTADO B)

El método de Thomas se obtiene particularizando el método de Crout al caso $u = l = 1$. Puesto que $\boldsymbol{A}$ es una matriz triadiagonal, los vectores $\boldsymbol{c}_{[k+1]}$ y $\boldsymbol{f}_{[k+1]}$ definidos en la ecuación 6.34 sólo tienen la última componente distinta de cero. Al resolver los sistemas triangulares inferiores indicados en la ecuación 6.37, se obtienen vectores $\boldsymbol{u}_{[k+1]}$ y $\boldsymbol{l}_{[k+1]}$ con la misma propiedad. Así pues, los factores $\boldsymbol{L}$ y $\boldsymbol{U}$ conservan los semianchos de banda de la matriz $\boldsymbol{A}$.

Con estas consideraciones, las ecuaciones 6.38 pueden reducirse a

$$
\begin{aligned}
&l_{11} = a_{11} \qquad\qquad u_{11} = 1\\
&k = 1, \ldots, n-1\\
&\quad \begin{cases} u_{k,k+1} &= a_{k,k+1} \;/\; l_{kk} \\ l_{k+1,k} &= a_{k+1,k} \\ l_{k+1,k+1} &= a_{k+1,k+1} - l_{k+1,k}\; u_{k,k+1} \end{cases}
\end{aligned}
$$

El almacenamiento puede optimizarse empleando una matriz rectangular $\boldsymbol{R}$ de n filas y 3 columnas (véase el subapartado 7.4.4), que contiene la matriz tridiagonal $\boldsymbol{A}$ antes de la descomposición y los factores $\boldsymbol{L}$ y $\boldsymbol{U}$ después. La relación entre los índices α y β (matriz $\boldsymbol{R}$) y los índices i y j (matrices $\boldsymbol{A}$, $\boldsymbol{L}$ y $\boldsymbol{U}$) es $\alpha = i$ y $\beta = 2 + j - i$. Las ecuaciones anteriores pueden reescribirse como

$$
\begin{aligned}
&k = 1, \ldots, n-1\\
&\quad \begin{cases} r_{k3} &= r_{k3} \;/\; r_{k2} \\ r_{k+1,2} &= r_{k+1,2} - r_{k1}\; r_{k3} \end{cases}
\end{aligned}
$$

Nótese que se han suprimido algunas asignaciones que son innecesarias por el hecho de trabajar con una única matriz (véase el problema 6.2).

En cuanto a las sustituciones, siguiendo el procedimiento indicado en el problema 6.2, se obtiene

$$
\begin{aligned}
&b_1 = b_1 \;/\; r_{12}\\
&b_i = (b_i - r_{i1} b_{i-1}) \;/\; r_{i2} \qquad i = 2, \ldots, n\\
&b_i = b_i - r_{i3} b_{i+1} \qquad i = n-1, n-2, \ldots, 1
\end{aligned}
$$

Apartado c)

Tanto en la descomposición como en las sustituciones, se accede a los coeficientes de la matriz $\boldsymbol{R}$ por filas. En cambio, la matriz rectangular $\boldsymbol{R}$ está almacenada por columnas (esquema de almacenamiento por defecto en FORTRAN, véase el subapartado 7.4.1). Esto no resulta adecuado desde el punto de vista de la paginación, puesto que se trabaja con posiciones no consecutivas de memoria. Podría optarse por almacenar la matriz $\boldsymbol{R}$ por filas en un vector $\boldsymbol{f}$, según se indica en el subapartado 7.4.2. Sin embargo, esto complicaría un poco más el esquema de almacenamiento. Por este motivo, se prefiere trabajar con una matriz rectangular de 3 filas y n columnas (la traspuesta de la matriz $\boldsymbol{R}$ de n filas y 3 columnas considerada hasta ahora). Esto puede hacerse simplemente permutando los índices en las ecuaciones del apartado b. Para esta nueva matriz, el almacenamiento por columnas ya resulta adecuado.

Apartado d)

El programa 7.4.1 permite resolver sistemas tridiagonales mediante el método de Thomas. La matriz del sistema se almacena en una matriz rectangular de 3 filas y n columnas (véanse las subrutinas `build_mat_vec`, `descom_LU` y `subs`).

```
c
c  Este programa calcula la solucion de un sistema tridiagonal
c  de ecuaciones mediante el metodo de Thomas.
c
c  Se emplea dimensionamiento dinamico y un esquema de
c  almacenamiento por diagonales
c
c  El fichero de datos rusa.dat contiene:
c    *Linea 1: n (numero de puntos base interiores)
c    *Lineas 2,...,n+3: x,f (valores de x e y en los apoyos)
c
c-----------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      parameter (mtot = 1000)
      dimension dd(mtot)

c___Ficheros de datos y de resultados
      open (unit=2, file='rusa.dat', status='unknown')
      open (unit=4, file='rusa.res', status='unknown')

c___Lectura del numero de puntos base interiores
      read (2,*) n

c___Definicion de los punteros
      call puntero(n,nx,nf,nh,nt,na,nb,mtot)
```

```
c___Lectura de los datos
      call get_data(n,dd(nx),dd(nf))

c___Generacion de la matriz y el vector de terminos independientes
      call build_mat_vec(n,dd(nx),dd(nf),dd(nh),dd(nt),dd(na),dd(nb))

c___Descomposicion LU
      call descom_LU(n,dd(na))

c___Sustituciones hacia adelante y hacia atras
      call subs(n,dd(na),dd(nb))

c___Escritura de los resultados
      call write_resul(n,dd(nb))

      stop
      end
c_______________________________________Definicion de los punteros
      subroutine puntero (n,nx,nf,nh,nt,na,nb,mtot)
      implicit real*8 (a-h,o-z)

nx   = 1
nf   = nx + (n+2)
nh   = nf + (n+2)
nt   = nh + (n+1)
na   = nt + (n+1)
nb   = na + (3*n)
nend = nb +  n

      if (nend .gt. mtot) then
        write(5,*) '   ERROR >>> Dimensionamiento insuficiente !'
        write(5,*) '             se requieren',nend,' posiciones'
        stop
      endif

return
end

c_________________________________________________Lectura de los datos
      subroutine get_data (n,x,f)
      implicit real*8 (a-h,o-z)
      dimension x(0:n+1),f(0:n+1)

      do 10 i=0,n+1
  read (2,*) x(i),f(i)
   10 continue

      close (2)

      return
      end
```

```
c____Generacion de la matriz y el vector de terminos independientes
      subroutine build_mat_vec (n,x,f,h,t,r,b)
      implicit real*8 (a-h,o-z)
      dimension x(0:n+1),f(0:n+1),h(0:n),t(0:n),r(3,n),b(n)

      do 20 i=0,n
          h(i)=x(i+1)-x(i)
          t(i)=f(i+1)-f(i)
   20 continue

      r(1,1)=0.d0
      do 50 i=2,n
          r(1,i)=h(i)/(h(i)+h(i-1))
   50 continue
      do 60 i=1,n
          r(2,i)=2.d0
          b(i)=3.d0*(h(i)/(h(i-1)+h(i)))*(t(i-1)/h(i-1))
          b(i)=b(i)+3.d0*(h(i-1)/(h(i-1)+h(i)))*(t(i)/h(i))
   60 continue
      do 70 i=1,n-1
          r(3,i)=h(i-1)/(h(i)+h(i-1))
   70 continue
      r(3,n)=0.d0

      return
      end

c_____________________________________________________Descomposicion LU
      subroutine descom_LU(n,r)
      implicit real*8 (a-h,o-z)
      dimension r(3,n)
      parameter(tol=1.d-5)

      if(abs(r(2,1)).lt.tol) call cero_pivote(1)
      r(3,1)=r(3,1)/r(2,1)
      do i=2,n-1
          r(2,i)=r(2,i)-r(1,i)*r(3,i-1)
          if(abs(r(2,i)).lt.tol) call cero_pivote(i)
          r(3,i)=r(3,i)/r(2,i)
      end do
      r(2,n)=r(2,n)-r(1,n)*r(3,n-1)

      return
      end

      subroutine cero_pivote(ieq)

      write(6,900)ieq
  900 format(3x,'Pivote igual a cero,ecuacion numero: 'i5)

      stop
      end
```

```
c_________________________Sustituciones hacia adelante y hacia atras
      subroutine subs(n,r,b)
      implicit real*8 (a-h,o-z)
      dimension r(3,n),b(n)

      b(1)=b(1)/r(2,1)
      do i=2,n
          b(i)=(b(i)-r(1,i)*b(i-1))/r(2,i)
      end do

      do i=n-1,1,-1
          b(i)=b(i)-r(3,i)*b(i+1)
      end do

      return
      end
c_______________________________________Escritura de los resultados
      subroutine write_resul (n,b)
      implicit real*8 (a-h,o-z)
      dimension b(n)

      write(4,100) 0,0.d0
      do 60 i=1,n
        write (4,100) i,b(i)
   60 continue
      write(4,100) n+1,0.d0

  100 format (1x,'La pendiente en el apoyo ',i3,' es ',1pd13.6)

close (4)

      return
      end
```

Prog. 7.4.1 Resolución de sistemas tridiagonales con el método de Thomas

Apartado e)

Para los datos indicados en el enunciado, se obtiene el archivo de resultados de la tabla 7.4.1. Para dibujar el trazado de la vía en una gráfica de Excel, debe tenerse en cuenta que las cúbicas $s_i(x)$ (es decir, sus coeficientes) son distintas en cada uno de los $n+1$ intervalos. Se obtiene entonces la gráfica de la figura 7.4.1.

Tabla 7.4.1 Pendientes en los apoyos de la montaña rusa

```
La pendiente en el apoyo   0 es  0.000000D+00
La pendiente en el apoyo   1 es  3.890094D-01
La pendiente en el apoyo   2 es -8.405664D-02
La pendiente en el apoyo   3 es -5.027829D-01
La pendiente en el apoyo   4 es  8.999935D-02
La pendiente en el apoyo   5 es  2.184029D-01
La pendiente en el apoyo   6 es -1.550568D-02
La pendiente en el apoyo   7 es -3.936095D-01
La pendiente en el apoyo   8 es  0.000000D+00
```

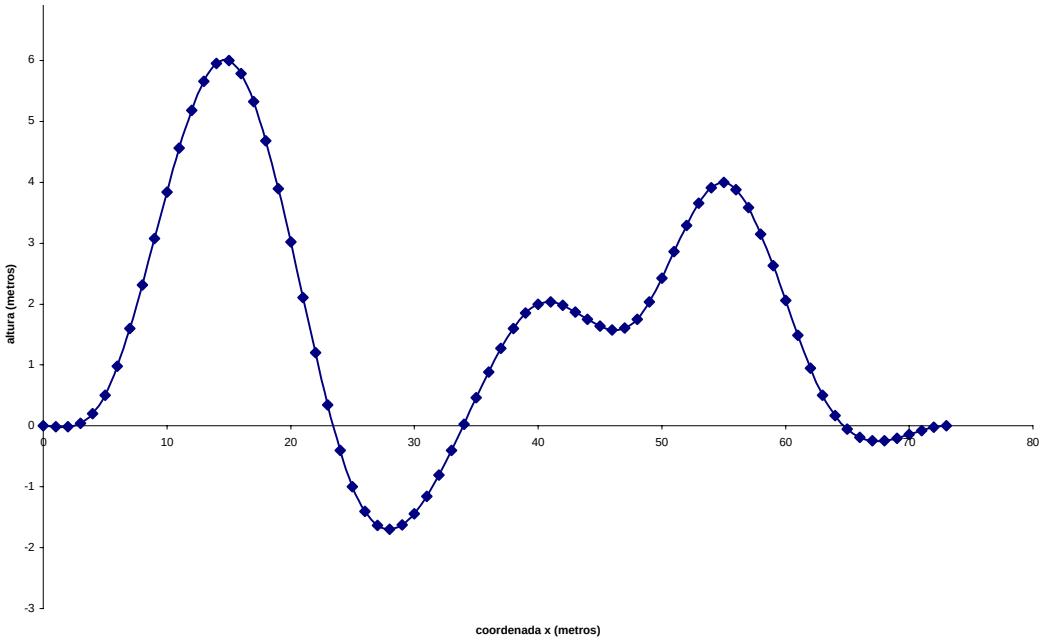


Fig. 7.4.1 Trazado en alzado de la montaña rusa

La gráfica tiene en los puntos de apoyo las pendientes prescritas en la tabla 7.4.1. Puesto que el sistema tridiagonal tiene solución única, este trazado es el único que verifica las condiciones de diseño: una cúbica en cada intervalo, pendientes nulas en los extremos y continuidad $\mathcal{C}^2$.

Problema 7.5

APARTADO A)

El vector $\boldsymbol{l}_{[k+1]}$, que contiene los elementos por encima de la diagonal de la columna k-ésima de la matriz $\boldsymbol{L}^T$, se calcula resolviendo el sistema triangular inferior $\boldsymbol{L}_{[k]}\ \boldsymbol{l}_{[k+1]} = \boldsymbol{f}_{[k+1]}$. Durante el proceso de sustitución hacia adelante, se irán obteniendo ceros hasta encontrar el primer elemento no nulo del vector $\boldsymbol{f}_{[k+1]}$ (k-ésima columna de la matriz $\boldsymbol{A}$). En consecuencia, el *skyline* de las matrices $\boldsymbol{L}^T$ y $\boldsymbol{A}$ es el mismo.

APARTADO B)

Para adaptar el algoritmo de Cholesky a matrices en *skyline*, se siguen tres pasos. En primer lugar, las ecuaciones 6.41 se reescriben de manera que *1)* se trabaje con elementos del triángulo superior (y no del inferior), y *2)* la matriz $\boldsymbol{L}^T$ se sobreescriba encima de $\boldsymbol{A}$. Se obtiene entonces

$$
\begin{array}{l}
a_{11} = \sqrt{a_{11}} \\
k = 1, \ldots, n-1 \\
\quad \left\{ \begin{array}{lll}
a_{1,k+1} & = a_{1,k+1} \ / \ a_{11} & \\
a_{i,k+1} & = \left(a_{i,k+1} - \displaystyle\sum_{j=1}^{i-1} a_{ji}\ a_{j,k+1} \right) \ / \ a_{ii} & i = 2, \ldots, k \\
a_{k+1,k+1} & = \sqrt{a_{k+1,k+1} - \displaystyle\sum_{i=1}^{k} a_{i,k+1}^2} &
\end{array} \right.
\end{array}
$$

En segundo lugar, el algoritmo se adapta al perfil en *skyline* eliminando las operaciones innecesarias, pero sin variar el esquema de almacenamiento. El primer elemento no nulo del vector $\boldsymbol{l}_{[k+1]}$ está en la fila $ip = k+2-l_{k+1}+l_k$. En consecuencia, pueden modificarse los rangos de los bucles y obtener

$$
\begin{array}{l}
a_{11} = \sqrt{a_{11}} \\
k = 1, \ldots, n-1 \\
\quad \left\{ \begin{array}{lll}
ip & = k + 2 - l_{k+1} + l_k & \\
a_{ip,k+1} & = a_{ip,k+1} \ / \ a_{ip,ip} & \\
a_{i,k+1} & = \left(a_{i,k+1} - \displaystyle\sum_{j=ip}^{i-1} a_{ji}\ a_{j,k+1} \right) \ / \ a_{ii} & i = ip+1, \ldots, k \\
a_{k+1,k+1} & = \sqrt{a_{k+1,k+1} - \displaystyle\sum_{i=ip}^{k} a_{i,k+1}^2} &
\end{array} \right.
\end{array}
$$

En tercer lugar, el esquema de almacenamiento se adapta al perfil en *skyline*. Para ello, hay que tener en cuenta que el elemento a_{ij} se almacena en la m-ésima componente del vector $\boldsymbol{c}$, con $m = l(j) - (j - i)$ (véase el subapartado 7.4.5):

$$
\begin{array}{l}
c(1) = \sqrt{c(1)} \\
lkk = 1 \\
k = 1, \ldots, n-1 \\
\left\{ \begin{array}{l}
lk = lkk \\
lkk = l(k+1) \\
ip = k + 2 - l_{k+1} + l_k \\
lip = l(ip) \\
c(lk+1) = c(lk+1) \;/\; c(lip) \\
i = ip+1, \ldots, k \\
\quad \left\{ \begin{array}{l}
li = l(i) \\
c(lkk - k - 1 + i) = \\
\quad \left(c(lkk - k - 1 + i) - \displaystyle\sum_{j=ip}^{i-1} c(li - i + j)\; c(lkk - k - 1 + j) \right) \;/\; c(li)
\end{array} \right. \\
c(lkk) = \sqrt{c(lkk) - \displaystyle\sum_{i=ip}^{k} \left[c(lkk - k - 1 + i) \right]^2}
\end{array} \right.
\end{array}
$$

Se han empleado las variables enteras auxiliares lk, lkk, lip y li para minimizar el número de veces que se accede a posiciones de memoria del vector de punteros $\boldsymbol{l}$.

Problema 7.6

APARTADO A)

Recorriendo la matriz $\boldsymbol{A}$ por columnas y guardando sus elementos no nulos en $\boldsymbol{c}$, se obtiene

$$\boldsymbol{c}^T = (8, 2, 7, 1, 4, 9, -4, 3, 5, 1, 6, -6)$$

El vector $\boldsymbol{m}$ de índices de fila de las componentes de $\boldsymbol{c}$ es

$$\boldsymbol{m}^T = (1, 2, 2, 3, 5, 3, 4, 1, 4, 2, 3, 5)$$

y el vector $\boldsymbol{l}$ de punteros de la posición en $\boldsymbol{c}$ del primer elemento de cada columna de $\boldsymbol{A}$ es

$$\boldsymbol{l}^T = (1, 3, 6, 8, 10, 13)$$

APARTADO B)

El producto de una matriz $\boldsymbol{A}$ llena por un vector $\boldsymbol{x}$ puede escribirse como

```
do i=1,n
  y(i) = 0.d0
enddo
do j=1,n
  do i=1,n
    y(i)=y(i)+a(i,j)*x(j)
  enddo
enddo
```

Con la ordenación elegida para los bucles (el exterior en j y el interior en i), la matriz $\boldsymbol{A}$ se recorre por columnas. Esto permite adaptar el algoritmo de multiplicación a un esquema de almacenamiento comprimido por columnas:

```
do i=1,n
  y(i) = 0.d0
enddo
do j=1,n
  do i=l(j),l(j+1)-1
    y(m(i)) = y(m(i)) + c(i)*x(j)
  enddo
enddo
```

Nótese que, con la ayuda del vector $\boldsymbol{l}$, en el bucle `DO`—`ENDDO` interior (en i) se recorren solamente los elementos no nulos de la columna j (almacenados en el vector $\boldsymbol{c}$).

10.7 Problemas del capítulo 8

Problema 8.1

Apartado a)

Los programas 8.1 y 8.2 (apartado 8.5) permiten calcular numéricamente la integral indefinida $\int_0^{\pi/2} \sin(x)\mathrm{d}x$ mediante el método de las aproximaciones rectangulares y el método compuesto del trapecio. Modificando en estos programas las instrucciones de escritura de resultados, de manera que se escriba por pantalla el error absoluto, y ejecutando los programas para los distintos valores de n que aparecen en las tablas 8.1 y 8.2, se obtienen los resultados de la tabla 8.1.1.

Tabla 8.1.1 Error absoluto en el cálculo de $\int_0^{\pi/2} \sin(x)\mathrm{d}x$ *por el método de las aproximaciones rectangulares (*E_{inf}, E_{sup}*) y por el método compuesto del trapecio (*E_{T}*)*

n	h	E_{inf}	E_{sup}	E_{T}
1	1.57080D+00	0.10000D+01	0.57080D+00	0.21460D+00
2	7.85298D+00	0.44464D+00	0.34076D+00	0.51941D-01
5	3.14159D+00	0.16532D+00	0.14884D+00	0.82382D-02
10	1.57080D-01	0.80597D-01	0.76483D-01	0.20570D-02
100	1.57080D-02	0.78745D-02	0.78334D-02	0.20562D-04
1000	1.57080D-03	0.78560D-03	0.78519D-03	0.20562D-06
10000	1.57080D-04	0.78542D-04	0.78538D-04	0.20561D-08

En esta tabla se puede observar cómo la convergencia a la solución analítica del problema, $\int_0^{\pi/2} \sin(x)\mathrm{d}x = 1$, es considerablemente mejor para el método compuesto del trapecio.

Apartado b)

A partir de los resultados del apartado anterior se puede obtener la figura 8.1.1 donde se representa logaritmo de E *versus* logaritmo de n para las dos técnicas mencionadas.

Apartado c)

Para un método lineal, el error en función del número de subintervalos n se comporta según la expresión

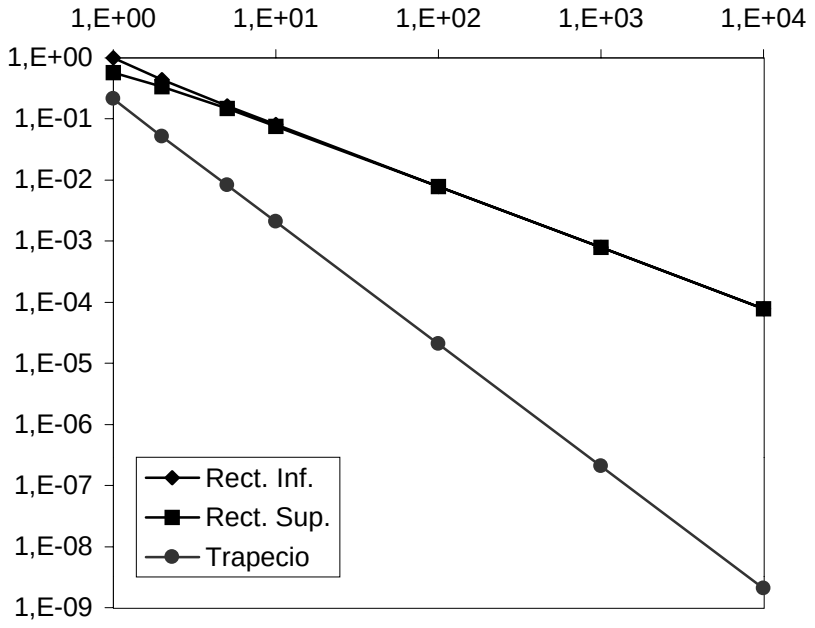


Fig. 8.1.1 Comparación de dos métodos para el cálculo de $\int_0^{\pi/2} \sin(x)\mathrm{d}x$

$$E = \frac{C}{n}$$

donde C es una constante independiente de n. Tomando logaritmos a ambos lados se obtiene la expresión equivalente

$$\log E = K - \log n$$

con $K = \log C$ independiente de n. Así, para un método lineal, si se representa $\log E$ en función de $\log n$ se obtiene una recta con pendiente -1.

Análogamente, para un método cuadrático el error se comporta según

$$E = \frac{C}{n^2}$$

donde C es una constante independiente de n. Y tomando logaritmos a ambos lados se obtiene la expresión

$$\log E = K - 2\log n$$

Si para un método cuadrático se representa $\log E$ en función de $\log n$ se obtiene una recta con pendiente menos -2.

APARTADO D)

El resultado teórico concuerda con el gráfico de la figura 8.1.1. Con ambos métodos se obtiene una recta que relaciona $\log E$ con $\log n$. Para cada una de las rectas se puede calcular su pendiente considerando dos parejas de valores $\{\log n, \log E\}$ cualesquiera. Con el método de las aproximaciones rectangulares (lineal) se comprueba que las dos rectas (correspondientes a las aproximaciones con rectángulos inferiores o rectángulos superiores), casi superpuestas, tienen pendiente -1, mientras que para el método compuesto del trapecio (cuadrático) la recta tiene pendiente -2.

Problema 8.2

El programa 8.2 (apartado 8.5) calcula la integral definida de la función $\sin(x)$ en el intervalo $[0, \pi/2]$ mediante el método compuesto del trapecio. Puede servir fácilmente como base para construir un programa que calcule integrales definidas de una función cualquiera en un intervalo cualquiera.

Para el problema planteado no se dispone de la definición de la función, sólo se dispone de su valor en una serie de puntos equiespaciados, pero esta diferencia respecto del programa 8.2 se soluciona simplemente sustituyendo la evaluación de la función por la lectura del valor correspondiente del archivo de datos. Al no conocer la definición de la función, la integración numérica es obligada. Por otro lado, el número de subintervalos queda ya determinado (número de datos menos uno) y, por lo tanto, la precisión con que se puede obtener la solución es limitada.

El programa 8.2.1 (inspirado en el programa 8.2) calcula los volúmenes V_D y V_T a partir de los valores conocidos de las funciones A_D y A_T (que se leen de los archivos de datos `AREAS_D.DAT` y `AREAS_T.DAT` respectivamente).

```
c
c        Este programa calcula los volumenes de desmonte y terraplen
c        por el METODO COMPUESTO DEL TRAPECIO a partir de las areas.
c        (datos equiespaciados)
c-----------------------------------------------------------------------
      implicit real*8 (a-h,o-z)

c___Asignacion de las unidades de lectura y escritura
      n_lec = 1
      n_esc = 2

c___Calculo del volumen de desmonte
      open(unit=n_lec,file='areas_d.dat',status='old')
      V_d = volumen(n_lec)
      close(n_lec)

c___Calculo del volumen de terraplen
```

```
      open(unit=n_lec,file='areas_t.dat',status='old')
      V_t = volumen(n_lec)
      close(n_lec)

c___Calculo del balance de tierras
      balance = V_d - V_t

c___Escritura de resultados
      open(unit=n_esc,file='volumen.res',status='new')
      write (n_esc,200) V_d,V_t,balance
      close(n_esc)
      write (6,200) V_d,V_t,balance

 200  format(/,1x,'Volumen de desmonte  = ',0pf12.7,/,
     .         1x,'Volumen de terraplen = ',0pf12.7,/,
     .         1x,'Balance de tierras   = ',0pf12.7,/)
      stop
      end


c____________________Calculo del volumen
      real*8 function volumen(n_lec)
      implicit real*8 (a-h,o-z)

c___Numero n de subintervalos
      read  (n_lec,*) n

c___Extremo izquierdo
      read(n_lec,*) a

c___Extremo derecho
      read(n_lec,*) b

c___Valor de h
      h = (b-a)/dble(n)

c___Calculo de la aproximacion V

C___valor en el extremo izquierdo
      read(n_lec,*) area
      V = 0.5d0*area

c___Puntos interiores
      do 10 i = 1,n-1
        read(n_lec,*) area
        V = V + area
  10  continue
```

```
c___Extremo derecho
      read(n_lec,*) area
      V = V + 0.5d0*area

c___Factor comun h
      volumen = V*h

      return
      end
```

Prog. 8.2.1 Método compuesto del trapecio para el balance de tierras

El listado incluye la `FUNCTION VOLUMEN` que dada una unidad de lectura `N_LEC`, correspondiente al archivo de datos que contiene las áreas, retorna el valor del volumen. Lee del archivo el número de subintervalos y los extremos de integración y, con ayuda de un bucle, lee los valores de las áreas y añade su contribución a la integral según el método compuesto del trapecio. Esta misma `FUNCTION` se puede utilizar para calcular la integral de cualquier función a partir de sus valores en puntos equiespaciados (introducidos en un archivo).

El programa principal utiliza dos veces la `FUNCTION VOLUMEN`. Para el cálculo de V_D abre el archivo que contiene la áreas de desmonte, con unidad de lectura `N_LEC`, y utiliza la función para el cálculo del volumen. Luego, abriendo el archivo que contiene las áreas de terraplén también con unidad `N_LEC`, calcula el volumen V_T con ayuda de la función. Finalmente, calcula el balance de tierras como diferencia de volúmenes y escribe los resultados por pantalla y en el archivo `VOLUMEN.RES`, que se muestra en la tabla 8.2.1.

Tabla 8.2.1 Balance de tierras

```
Volumen de desmonte  =   75.6000000
Volumen de terraplen =   92.6250000
Balance de tierras   =  -17.0250000
```

Obsérvese que la función implementada en el programa 8.2.1 calcula la integral a partir de los datos de un archivo sólo en el caso de que los puntos sean equiespaciados. En un caso más general, con puntos no necesariamente equiespaciados, cada subintervalo tendría un tamaño diferente, tal como se muestra en la figura 8.4 (apartado 8.3), y sería necesario utilizar una fórmula más general para el método compuesto del trapecio. En cada subintervalo el área del trapecio sería

$$A_i = h_i \frac{f(x_i) + f(x_{i+1})}{2}$$

donde $h_i = x_{i+1} - x_i$ no puede tomarse como factor común, y el área total se calcularía como

$$I_T = A_0 + A_1 + A_2 + \ldots A_{n-1}$$

10.8 Problemas del capítulo 9

Problema 9.1

Apartado a)

Los programas 9.1 y 9.2 (apartado 9.5) permiten calcular numéricamente la sección de la base del pilar mediante el método de Euler y el método de Heun. Modificando en estos programas las instrucciones de escritura de resultados, de manera que se escriba por pantalla el error absoluto, y ejecutando los programas para distintos valores de m, se pueden obtener los resultados de la tabla 9.1.1.

Tabla 9.1.1 Error absoluto en el cálculo de la sección del pilar por los métodos de Euler y de Heun

m	Euler	Heun
1	0.17557D-03	0.40727D-05
2	0.89823D-04	0.10452D-05
5	0.36438D-04	0.16988D-06
10	0.18306D-04	0.42693D-07
100	0.18385D-05	0.42896D-09
1000	0.18393D-06	0.42915D-11

A la hora de calcular el error absoluto, en la modificación de los programas se debe calcular la solución analítica evaluando la expresión 9.15 del apartado 9.2 para $x = L$. Dado que se va a ejecutar cada uno de los programas varias veces, puede resultar cómodo asignar los valores de las constantes del problema a las variables correspondientes ($L = 4$, $P = 100$, $S_0 = 0.07$,...) en lugar de leer estos valores del teclado.

En la tabla 9.1.1 se puede observar cómo la convergencia a la solución analítica del problema es considerablemente más rápida para el método de Heun que para el método de Euler.

Apartado b)

A partir de los resultados del apartado anterior se puede obtener el gráfico de la figura 9.1.1, donde se representa logaritmo de E *versus* logaritmo de m para los dos métodos.

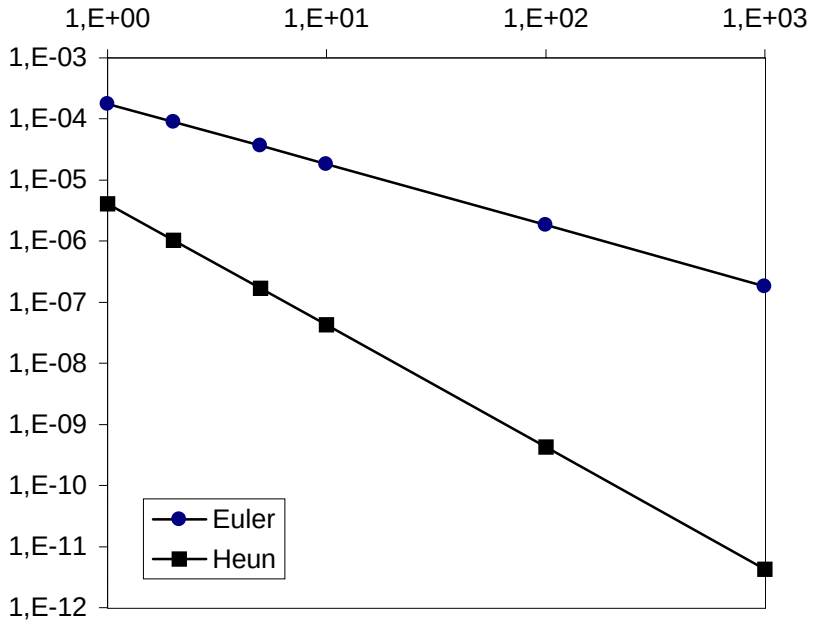


Fig. 9.1.1 Comparación de dos métodos de integración numérica

Apartado c)

En el problema 8.1 se comprobó que, al representar $\log E$ en función de $\log m$, se obtiene una recta de pendiente -1 para un método lineal ($E = \mathcal{O}(\frac{1}{m})$) y una recta de pendiente -2 para un método cuadrático ($E = \mathcal{O}(\frac{1}{m^2})$).

Si se calculan las pendientes de las dos rectas de la figura 9.1.1 (tomando dos parejas de valores $\{\log m, \log E\}$) se obtienen valores de -1 para el método de Euler y de -2 para el método de Heun. Esto es debido a que el primer método es lineal, mientras que el segundo método es cuadrático.

Problema 9.2

El sistema de ecuaciones diferenciales ordinarias de dimensión n

$$\left.\begin{aligned}
\frac{\mathrm{d}y_{(1)}}{\mathrm{d}t} &= f_{(1)}(t, y_{(1)}, y_{(2)}, \ldots, y_{(n)}) \\
\frac{\mathrm{d}y_{(2)}}{\mathrm{d}t} &= f_{(2)}(t, y_{(1)}, y_{(2)}, \ldots, y_{(n)}) \qquad \text{en} \qquad t \in [a,b] \\
&\vdots \\
\frac{\mathrm{d}y_{(n)}}{\mathrm{d}t} &= f_{(n)}(t, y_{(1)}, y_{(2)}, \ldots, y_{(n)}) \\
y_{(1)}(a) &= \alpha_{(1)} \\
y_{(2)}(a) &= \alpha_{(2)} \\
&\vdots \\
y_{(n)}(a) &= \alpha_{(n)}
\end{aligned}\right\}$$

puede reescribirse con notación vectorial como

$$\left.\begin{aligned}
\mathrm{d}\boldsymbol{y} &= \boldsymbol{f}(t, \boldsymbol{y}) \qquad \text{en} \qquad t \in [a,b] \\
\boldsymbol{y}(a) &= \boldsymbol{\alpha}
\end{aligned}\right\}$$

donde $\boldsymbol{y} = (y_{(1)}, y_{(2)}, \ldots, y_{(n)})^T$ y $\boldsymbol{f}(t, \boldsymbol{y}) = (f_{(1)}(t, \boldsymbol{y}), f_{(2)}(t, \boldsymbol{y}), \ldots, f_{(n)}(t, \boldsymbol{y}))^T$. La extensión del método de Heun a este problema es

$$\begin{aligned}
\boldsymbol{Y}_0 &= \boldsymbol{\alpha} \\
\boldsymbol{Y}^*_{i+1} &= \boldsymbol{Y}_i + h\boldsymbol{f}(t_i, \boldsymbol{Y}_i) \\
\boldsymbol{Y}_{i+1} &= \boldsymbol{Y}_i + \frac{h}{2}\left[\boldsymbol{f}(t_i, \boldsymbol{Y}_i) + \boldsymbol{f}(t_{i+1}, \boldsymbol{Y}^*_{i+1})\right] \qquad \text{para} \qquad i = 1, \ldots, n
\end{aligned}$$

donde simplemente se han sustituido los escalares α, Y_0, Y_i, Y_{i+1}, Y^*_{i+1} y f en la definición del método de Heun (ecuación 9.17, apartado 9.3) por los vectores $\boldsymbol{\alpha}$, $\boldsymbol{Y}_0$, $\boldsymbol{Y}_i$, $\boldsymbol{Y}_{i+1}$, $\boldsymbol{Y}^*_{i+1}$ y $\boldsymbol{f}$.

El método de Heun puede escribirse, componente a componente, como

$$\left.\begin{aligned}
\begin{Bmatrix} Y_{(1)}(t_0) \\ Y_{(2)}(t_0) \\ \vdots \\ Y_{(n)}(t_0) \end{Bmatrix} &= \begin{Bmatrix} \alpha_{(1)} \\ \alpha_{(2)} \\ \vdots \\ \alpha_{(n)} \end{Bmatrix} \\
\begin{Bmatrix} Y^*_{(1)}(t_{i+1}) \\ Y^*_{(2)}(t_{i+1}) \\ \vdots \\ Y^*_{(n)}(t_{i+1}) \end{Bmatrix} &= \begin{Bmatrix} Y_{(1)}(t_i) \\ Y_{(2)}(t_i) \\ \vdots \\ Y_{(n)}(t_i) \end{Bmatrix} + h \begin{Bmatrix} f_{(1)}(t_i, \boldsymbol{Y}_i) \\ f_{(2)}(t_i, \boldsymbol{Y}_i) \\ \vdots \\ f_{(n)}(t_i, \boldsymbol{Y}_i) \end{Bmatrix} \\
\begin{Bmatrix} Y_{(1)}(t_{i+1}) \\ Y_{(2)}(t_{i+1}) \\ \vdots \\ Y_{(n)}(t_{i+1}) \end{Bmatrix} &= \begin{Bmatrix} Y_{(1)}(t_i) \\ Y_{(2)}(t_i) \\ \vdots \\ Y_{(n)}(t_i) \end{Bmatrix} + \frac{h}{2} \left(\begin{Bmatrix} f_{(1)}(t_i, \boldsymbol{Y}_i) \\ f_{(2)}(t_i, \boldsymbol{Y}_i) \\ \vdots \\ f_{(n)}(t_i, \boldsymbol{Y}_i) \end{Bmatrix} + \begin{Bmatrix} f_{(1)}(t_{i+1}, \boldsymbol{Y}^*_{i+1}) \\ f_{(2)}(t_{i+1}, \boldsymbol{Y}^*_{i+1}) \\ \vdots \\ f_{(n)}(t_{i+1}, \boldsymbol{Y}^*_{i+1}) \end{Bmatrix} \right)
\end{aligned}\right\}$$

donde $\boldsymbol{Y}_i = (Y_{(1)}(t_i), Y_{(2)}(t_i), \ldots, Y_{(n)}(t_i))^T$ y $\boldsymbol{Y}^*_{i+1} = (Y^*_{(1)}(t_{i+1}), Y^*_{(2)}(t_i), \ldots, Y^*_{(n)}(t_{i+1}))^T$.

Problema 9.3

El sistema de dos ecuaciones diferenciales ordinarias puede reescribirse en forma vectorial como

$$\left.\begin{array}{rl} \mathrm{d}\boldsymbol{y} = \boldsymbol{f}(t,\boldsymbol{y}) & \text{en} \quad t \in [0,1] \\ \boldsymbol{y}(0) = \boldsymbol{\alpha} & \end{array}\right\}$$

con

$$\boldsymbol{y} = \left\{\begin{array}{l} y_{(1)} \\ y_{(2)} \end{array}\right\} = \left\{\begin{array}{l} y_A \\ y_B \end{array}\right\} \quad ; \quad \boldsymbol{f}(t,\boldsymbol{y}) = \boldsymbol{f}(t, y_{(1)}, y_{(2)}) = \left\{\begin{array}{l} -k y_{(1)} y_{(2)} \\ -k y_{(1)} y^2_{(2)} \end{array}\right\} \quad ; \quad \boldsymbol{\alpha} = \left\{\begin{array}{l} 1 \\ 1 \end{array}\right\}$$

Apartado a)

El programa 9.3.1 resuelve este problema mediante el método de Euler y el método de Heun. El programa está escrito de forma modular, de manera que puede resolver cualquier sistema de ecuaciones diferenciales ordinarias simplemente modificando la definición de la subrutina que proporciona los valores de las derivadas, `calcula_f`. En todo el programa se utiliza la notación vectorial comentada en el apartado 9.4 (método de Euler) y en la resolución del problema 9.2 (método de Heun).

```
c
c        Este programa resuelve el sistema de n ecuaciones
c        diferenciales ordinarias
c                   y'=f(y,t)  para t en [a,b]
c                   y(a)=alpha
c        (donde y es un vector con n componentes)
c        por el METODO DE EULER y el METODO DE HEUN.
c        Aplicacion: problema de la estacion depuradora de aguas
c-----------------------------------------------------------------------
      implicit real*8 (a-h,o-z)
      parameter(maxdim=10)
      dimension alpha(maxdim), y(maxdim)

c___Extremos a,b del intervalo
      write(6,100)
      read(5,*) a,b
 100  format(/,2x,'Extremos del intervalo = ',$)

c___Numero de subintervalos m
      write(6,200)
      read(5,*) m
 200  format(/,2x,'Numero de subintervalos = ',$)
```

```
c___Numero de componentes del vector y
      write(6,300)
      read(5,*) n
 300  format(/,2x,'Dimension del sistema de edo''s = ',$)

c___Valores iniciales, alpha
      write(6,400) n
      read(5,*) (alpha(i),i=1,n)
 400  format(/,1x,i2,1x,'valores iniciales: ',$)

c___Eleccion del metodo de resolucion
      write(6,500)
      read(5,*) metodo
 500  format(/,2x,'Metodo de resolucion:
     .       ',/,7x,'(1) EULER',/,7x,'(2) HEUN')

c___Llamada a la subrutina del metodo correspondiente
      if(metodo.eq.1) call euler(a,b,m,alpha,y,n)
      if(metodo.eq.2) call heun(a,b,m,alpha,y,n)

      stop
      end

c____________________metodo de Euler
      subroutine euler(a,b,m,alpha,y,n)
      implicit real*8 (a-h,o-z)
      parameter(maxdim=10)
      dimension alpha(n), y(n), f(maxdim)

c___Apertura del archivo de resultados
      open(unit=1,file='euler.res',status='unknown')
      write(1,*) 'METODO DE EULER'

c___Paso h (discretizacion)
      h = (b-a)/dble(m)

c___Inicializacion
      t = a
      do i=1,n
        y(i) = alpha(i)
      enddo
      write(1,100) ipas,t,y

c___Bucle
      do ipas=1,m
        call calcula_f(y,t,f)
        do i=1,n
```

```
           y(i) = y(i) + h*f(i)
         enddo
         t = t + h
         write(1,100) ipas,t,y
       enddo

c___Cierre del archivo de resultados
       close(1)

100    format(1x,i7,2x,'t =',f7.5,5x,
     .       'y = ( ',<n-1>(f12.8,','),f12.8,')')
       return
       end

c____________________metodo de Heun
       subroutine heun(a,b,m,alpha,y,n)
       implicit real*8 (a-h,o-z)
       parameter(maxdim=10)
       dimension alpha(n), y(n), f(maxdim)
       dimension yaux(maxdim), faux(maxdim)

c___Apertura del archivo de resultados
       open(unit=1,file='heun.res',status='unknown')
       write(1,*) 'METODO DE HEUN'

c___Paso h (discretizacion)
       h = (b-a)/dble(m)

c___Inicializacion
       t = a
       do i=1,n
         y(i) = alpha(i)
       enddo
       write(1,100) ipas,t,y

c___Bucle
       do ipas=1,m
         call calcula_f(y,t,f)
         do i=1,n
            yaux(i) = y(i) + h*f(i)
         enddo
         t = t + h
         call calcula_f(yaux,t,faux)
         do i=1,n
            y(i) = y(i) + 0.5d0*h*(f(i)+faux(i))
         enddo
         write(1,100) ipas,t,y
```

```
      enddo

c___Cierre del archivo de resultados
      close(1)

100   format(1x,i7,2x,'t =',f7.5,5x,
     .       'y = ( ',<n-1>(f12.8,','),f12.8,')')
      return
      end

c____________________Definicion del vector de derivadas
      subroutine calcula_f(y,t,f)
      implicit real*8 (a-h,o-z)
      dimension y(2), f(2)

      f(1) = -24.d0*y(1)*y(2)
      f(2) = -24.d0*y(1)*(y(2)**2)

      return
      end
```

Prog. 9.3.1 Resolución de un sistema de ecuaciones diferenciales ordinarias mediante los métodos de Euler y de Heun

El programa 9.3.1 contiene dos subrutinas para el cálculo de la solución: la rutina `euler` y la rutina `heun`. En ambos casos los datos de entrada para la subrutina son los extremos del intervalo de cálculo $[a, b]$, el número de subintervalos m, la dimensión n del sistema de ecuaciones diferenciales ordinarias y el vector de condiciones iniciales $\boldsymbol{\alpha}$. Una vez ejecutada cualquiera de las dos subrutinas, el vector `y` contiene el vector solución $\boldsymbol{Y}_n \simeq \boldsymbol{y}(b)$. Durante el cálculo se escriben los cálculos intermedios $\boldsymbol{Y}_i \simeq \boldsymbol{y}(x_i)$, $i = 1, \ldots n$ en el archivo `euler.res` o en el archivo `heun.res` respectivamente.

La implementación de cada uno de los métodos es similar a la de los programas 9.1 (Euler) y 9.2 (Heun). Simplemente hay que sustituir las asignaciones de variables por bucles para recorrer las componentes de los vectores correspondientes, cuando sea necesario. Por ejemplo, la asignación `y = y + h*f` se debe reescribir como

```
do i=1,n
   y(i) = y(i) + h*f(i)
enddo
```

dado que tanto `y` como `f` son ahora vectores.

Una vez implementadas las subrutinas, el programa principal simplemente lee las constantes

que definen el problema y que se deben pasar a las subrutinas de cálculo, pide al usuario el método de resolución y llama a la rutina correspondiente.

APARTADO B)

Ejecutando el programa 9.3.1 para diferentes valores del número de subintervalos m se obtienen los resultados de la tabla 9.3.1.

Tabla 9.3.1 Solución numérica en el instante $t = 1$ para los métodos de Euler y Heun

m	Euler	Heun
10	overflow	overflow
100	(0.00006882, 0.34668445)	(0.00006748, 0.36787680)
1000	(0.00006549, 0.36624375)	(0.00006602, 0.36791095)
10000	(0.00006596, 0.36774095)	(0.00006601, 0.36790380)
100000	(0.00006601, 0.36788748)	(0.00006601, 0.36790373)

Como solución de referencia se tomará la obtenida con el método de mayor orden de convergencia (Heun) y para la discretización más fina ($m = 100\,000$ subintervalos). Es razonable suponer que esta solución es la más precisa de las reflejadas en la tabla 9.3.1. Esta solución se utilizará para hacer una comparación de la convergencia de ambos métodos.

En la tabla 9.3.1 se puede observar cómo para $m = 10$ el valor de h no es suficientemente pequeño como para poder aproximar correctamente la solución con ninguno de los dos métodos. Es necesario un número mayor de puntos para poder capturar aproximadamente la solución. Pero, para valores suficientemente grandes de m, se puede observar la convergencia de ambos métodos a la solución exacta; ésta es considerablemente mejor para el método de Heun.

La tabla 9.3.2 muestra el logaritmo decimal del error en $y_B(1)$ para ambos métodos y para distintos valores de m (comparando con el resultado de referencia $y_B(1) = 0.36790373$). Puede observarse cómo $\log E$ en función de $\log m$ corresponde a una recta de pendiente -1 para el método de Euler (lineal) y pendiente -2 para el método de Heun (cuadrático), tal como se comentó en el problema 9.1.

Tabla 9.3.2 Logaritmo de E para la componente y_B en $t = 1$ con los métodos de Euler y Heun

m	$\log m$	Euler	Heun
100	2	-1.67326937951316	-4.56984776155033
1000	3	-2.77989886255826	-5.14130282722359
10000	4	-3.78839376954350	-7.12364816033703

APARTADO B)

La solución de referencia (método de Heun, $m = 100\ 000$) se recoge en la tabla 9.3.3.

Tabla 9.3.3 Solución de referencia: método de Heun, $m = 100\ 000$

```
METODO DE HEUN
     0  t = 0.00000     y = (   1.00000000,  1.00000000)
              .
              .
              .
  3854  t = 0.03854     y = (   0.50009555,  0.60658862)
  3855  t = 0.03855     y = (   0.50002276,  0.60654446)
  3856  t = 0.03856     y = (   0.49994998,  0.60650032)
  3857  t = 0.03857     y = (   0.49987721,  0.60645619)
              .
              .
              .
100000  t = 1.00000     y = (   0.00006601,  0.36790373)
```

De la tabla 9.3.3 puede deducirse que el reactivo A reduce su concentración a la mitad en el instante $t = 0.03855$ aproximadamente. Si fuese necesario obtener el instante de tiempo con una mayor precisión se deberían hacer los cálculos con un número de subintervalos m mayor. De todas formas, con los resultados de que se dispone se puede intentar ajustar un poco más el instante de tiempo a partir de los resultados en los instantes $t = 0.03855$ y $t = 0.03856$. Suponiendo que $y_A(t)$ se comporta como una recta en este pequeño intervalo

$$y_A(t) = y_A(0.03855) + \frac{y_A(0.03856) - y_A(0.03855)}{0.00001}(t - 0.03855) = 0.50002276 - 7.278(t - 0.03855)$$

se puede imponer $y_A(t) = 0.5$

$$0.50002276 - 7.278(t - 0.03855) = 0.5$$

con lo que se obtiene el instante de tiempo

$$t = 0.038553$$