Departamento de Física, Facultad de Ciencias, Universidad de Chile.
Las Palmeras 3425, Ñuñoa. Casilla 653, Correo 1, Santiago
FONO: 562 678 7276 FAX: 562 271 2973
E-MAIL: secretaria@fisica.ciencias.uchile.cl

Apuntes de un curso de

FÍSICA MATEMÁTICA

José Rogan C.
Víctor Muñoz G.

Índice

Primer Curso

INTRODUCCIÓN A LA FÍSICA MATEMÁTICA

Capítulo 1

Elementos del sistema operativo UNIX.

versión final 2.3-19 de agosto de 2002

1.1 Introducción.

En este capítulo se intentará dar los elementos básicos para poder trabajar en un ambiente UNIX. Sin pretender cubrir todos los aspectos del mismo, nuestro interés se centra en dar las herramientas al lector para que pueda realizar los trabajos del curso bajo este sistema operativo. Como comentario adicional, concientemente se ha evitado la traducción de gran parte de la terminología técnica teniendo en mente que documentación disponible se encuentre por lo general en inglés y nos interesa que el lector sea capaz se reconocer los términos.

El sistema operativo UNIX es el más usado en investigación científica, tiene una larga historia y muchas de sus ideas y métodos se encuentran presentes en otros sistemas operativos. Algunas de las características relevantes del UNIX moderno son:

- Memoria grande, lineal y virtual: Un programa en una máquina de 32 Bits puede acceder y usar direcciones hasta los 4 GB en un máquina de sólo 4 MB de RAM. El sistema sólo asigna memoria auténtica cuando le hace falta, en caso de falta de memoria de RAM, se utiliza el disco duro (*swap*).

- Multitarea (*Multitasking*): Cada programa tiene asignado su propio "espacio" de memoria. Es **imposible** que un programa afecte a otro sin usar los servicios del sistema operativo. Si dos programas escriben en la misma dirección de memoria cada uno mantiene su propia "idea" de su contenido.

- Multiusuario: Más de una persona puede usar la máquina al mismo tiempo. Programas de otros usuarios continúan ejecutándose a pesar de que un nuevo usuario entre a la máquina.

- Casi todo tipo de dispositivo puede ser accedido como un archivo.

- Existen muchos aplicaciones diseñadas para trabajar desde la línea de comandos. Además, la mayoría de las aplicaciones permiten que la salida de una pueda ser la entrada de la otra.

- Permite compartir dispositivos (como disco duro) entre una red de máquinas.

Por su naturaleza de multiusuario, **nunca** se debe apagar una máquina UNIX[1], ya que una máquina apagada sin razón puede matar trabajos de días, perder los últimos cambios de tus archivos e ir degradando el sistema de archivos en dispositivos como el disco duro.

Entre los sistemas operativos UNIX actuales cabe destacar:

- Linux: esta disponible para: Intel x86; Motorola 68k, en particular, para las estaciones Sun3, computadores personales Apple Macintosh, Atari y Amiga; Sun SPARC; Alpha; Motorola/IBM PowerPC; ARM, máquinas NetWinder; Sun UltraSPARC; MIPS CPUs, máquinas SGI y estaciones Digital; HP PA-RISC; IA-64, arquitectura Intel de 64-bits; S/390, servidores IBM S/390 y SuperH procesadores Hitachi SuperH.
- SunOS[2]: disponible para la familia 68K así como para la familia SPARC de estaciones de trabajo SUN
- Solaris[3] : disponible para la familia SPARC de SUN así como para la familia x86.
- OSF1[4]: disponible para Alpha.
- Ultrix: disponible para VAX de Digital
- SYSVR4[5]: disponible para la familia x86, vax.
- IRIX: disponible para MIPS.
- AIX[6]: disponible para RS6000 de IBM y PowerPC.

1.2 Ingresando al sistema.

En esta sección comentaremos las operaciones de comienzo y fin de una sesión en UNIX así como la modificación de la contraseña (que a menudo no es la deseada por el usuario, y que por lo tanto puede olvidar con facilidad).

1.2.1 Terminales.

Para iniciar una sesión es necesario poder acceder a un terminal. Pueden destacarse dos tipos de terminales:

- Terminal de texto: consta de una pantalla y de un teclado. Como indica su nombre, en la pantalla sólo es posible imprimir caracteres de texto.
- Terminal gráfico: Consta de pantalla gráfica, teclado y *mouse*. Dicha pantalla suele ser de alta resolución. En este modo se pueden emplear ventanas que emulan el comportamiento de un terminal de texto (`xterm` o `gnome-terminal`).

[1] Incluyendo el caso en que la máquina es un PC normal corriendo Linux u otra versión de UNIX.
[2] SunOS 4.1.x también se conoce como Solaris 1.
[3] También conocido como SunOS 5.x, solaris 2 o Slowaris :-).
[4] También conocido como Dec Unix.
[5] También conocido como Unixware y Novell-Unix.
[6] También conocido como Aches.

1.2.2 Login.

El primer paso es encontrar un terminal libre donde aparezca el *login prompt* del sistema:

```
Debian GNU/Linux 2.2 hostname tty2

hostname login:
```

También pueden ocurrir un par de cosas cosas:

- La pantalla no muestra nada.
  - Comprobar que la pantalla esté encendida.
  - Pulsar alguna tecla o mover el *mouse* para desactivar el protector de pantalla.
- Otra persona ha dejado una sesión abierta. En este caso existe la posibilidad de intentar en otra máquina o bien finalizar la sesión de dicha persona (si ésta no se halla en las proximidades).

  Una vez que se haya superado el paso anterior de encontrar el *login prompt* se procede con la introducción de tu *Username* al *prompt* de *login* y después la contraseña (*password*) adecuado.

1.2.3 Passwords.

El *password* puede ser cualquier secuencia de caracteres a elección. Deben seguirse las siguientes pautas:

- Debe ser fácil de recordar por uno mismo. Si se olvida, deberá pasarse un mal rato diciéndole al administrator de sistema que uno lo ha olvidado.
- Para evitar que alguna persona no deseada obtenga la *password* y tenga libre acceso a los archivos de tu cuenta:
  - Las mayúsculas y minúsculas no son equivalentes sin embargo se recomienda que se cambie de una a otra.
  - Los caracteres numéricos y no alfabéticos también ayudan. Debe tenerse sin embargo la precaución de usar caracteres alfanuméricos que se puedan encontrar en todos los terminales desde los que se pretenda acceder.
  - Las palabras de diccionario deben ser evitadas.
- Debes cambiarlo si crees que tu *password* es conocido por otras personas, o descubres que algún intruso[7] está usando tu cuenta.
- El *password* debe de ser cambiado con regularidad.

[7] Intruso es cualquier persona que no sea el usuario.

La orden para cambiar el password en UNIX es `passwd`. A menudo cuando existen varias máquinas que comparten recursos (disco duro, impresora, correo electrónico, ...), para facilitar la administración de dicho sistema se unifican los recursos de red (entre los que se hayan los usuarios de dicho sistema) en una base de datos común. Dicho sistema se conoce como NIS (*Network Information Service*)[8]. Si el sistema empleado dispone de este servicio, la modificación de la contraseña en una máquina supone la modificación en todas las máquinas que constituyan el dominio NIS.

1.2.4 Cerrando la sesión.

Es importante que nunca se deje abierta una sesión, pues algún "intruso" podría tener libre acceso a archivos de tu propiedad y manipularlos de forma indeseable para ti. Para evitar todo esto basta teclear `logout` ó `exit` y habrá acabado tu sesión de UNIX en dicha máquina[9].

1.3 Archivos y directorios.

Aunque haya diferentes distribuciones y cada una traiga sus programas, la estructura básica de directorios y archivos es más o menos la misma en todas:

```
/-|--> bin
  |--> boot
  |--> cdrom
  |--> dev
  |--> etc
  |--> floppy
  |--> home
  |--> lib
  |--> mnt
  |--> proc
  |--> root
  |--> sbin
  |--> tmp
  |--> usr -|--> X11
  |         |--> bin
  |         |--> include
  |         |--> lib
  |         |--> local -|--> bin
  |         |           |--> lib
  |         |--> man
  |         |--> src --> linux
  |         |--> doc
  |--> var --> adm
```

[8] Antiguamente se conocía como YP (*Yellow Pages*), pero debido a un problema de marca registrada de *United Kingdom of British Telecomunications* se adoptaron las siglas NIS.

[9] En caso que se estuviera trabajando bajo X-Windows debes cerrar la sesión con `Log out of Gnome`.

El árbol que observamos muestra un típico árbol de directorios en Linux. Pueden variar, sin embargo, algunos de los nombres dependiendo de la distribución o versión de Linux que se esté usando. Algunos directorios destacados son:

- `/home` - Espacio reservado para las cuentas de los usuarios.
- `/bin`, `/usr/bin` - Binarios (ejecutables) básicos de UNIX.
- `/etc`, aquí se encuentran los archivos de configuración de todo el software de la máquina.
- `/proc`, es un sistema de archivo virtual. Contiene archivos que residen en memoria pero no en el disco duro. Hace referencia a los programas que se están corriendo en el momento en el sistema.
- `/dev` (*device*) (dispositivo). Aquí se guardan los controladores de dispositivos. Se usan para acceder a los dispositivos físicos del sistema y recursos como discos duros, *modems*, memoria, *mouse*, etc. Algunos dispositivos:
  - `hd`: `hda1` será el disco duro IDE, primario (`a`), y la primera partición (`1`).
  - `fd`: Así tambien, los archivos que empiecen con las letras `fd` se referirán a los controladores de las disketteras: `fd0` sería la primera diskettera, `fd1` sería la segunda y así sucesivamente.
  - `ttyS`: se usan para acceder a los puertos seriales como por ejemplo, `ttyS0` es el puerto conocido como `com1`.
  - `sd`: son los dispositivos SCSI. Su uso es muy similar al del `hd`.
  - `lp`: son los puertos paralelos. `lp0` es el puerto conocido como `LPT1`.
  - `null`: este es usado como un agujero negro, ya que todo lo que se dirige allí desaparece.
  - `tty`: hacen referencia a cada una de las consolas virtuales. Como es de suponerse, `tty1` será la primera consola virtual, `tty2` la segunda, etc.
- `/usr/local` - Zona con las aplicaciones no comunes a todos los sistemas UNIX, pero no por ello menos utilizadas. En `/usr/doc` se puede encontrar información relacionada con dicha aplicación (en forma de páginas de manual, texto, html o bien archivos dvi, Postscript o pdf). También encontramos archivos de ejemplo, tutoriales, *HOWTO*, etc.

1.4 Órdenes básicas.

Para ejecutar un comando, basta con teclear su nombre (también debes tener permiso para hacerlo). Las opciones o modificadores empiezan normalmente con el carácter `-` (p. ej. `ls -l`). Para especificar más de una opción, se pueden agrupar en una sola cadena de caracteres (`ls -l -h` es equivalente a `ls -lh`). Algunos comandos aceptan también opciones dadas por palabras completas, en cuyo caso usualmente comienzan con `--` (`ls --color=auto`).

1.4.1 Archivos y directorios.

En un sistema computacional la información se encuentra en archivos que la contienen (tabla de datos, texto ASCII, fuente en lenguaje C, fortran o C++, ejecutable, imagen, mp3, figura, resultados de simulación, ...). Para organizar toda la información se dispone de una entidad denominada directorio, que permite el almacenamiento en su interior tanto de archivos como de otros directorios[10]. Se dice que la estructura de directorios en UNIX es jerárquica o arborescente, debido a que todos los directorios nacen en un mismo punto (denominado directorio raíz). De hecho la zona donde uno trabaja es un nodo de esa estructura de directorios, pudiendo uno a su vez generar una estructura por debajo de ese punto. Un archivo se encuentra situado siempre en un directorio y su acceso se realiza empleando el camino que conduce a él en el Árbol de Directorios del Sistema. Este camino es conocido como el *path*. El acceso a un archivo se puede realizar empleando:

- Path Absoluto, aquel que empieza con `/`

  Por ejemplo : `/etc/printcap`

- Path Relativo, aquel que NO empieza con `/`

  Por ejemplo : `../examples/rc.dir.01`

- Nombres de archivos y directorios pueden usar un máximo de 255 caracteres, cualquier combinación de letras y símbolos (el carácter `/` no se permite).

Los caracteres comodín pueden ser empleados para acceder a un conjunto de archivos con características comunes. El signo `*` puede sustituir cualquier conjunto de caracteres[11] y el signo `?` cualquier caracter individual. Por ejemplo:[12]

```
bash$ ls
f2c.1          flexdoc.1    rcmd.1    rptp.1   zforce.1
face.update.1  ftptool.1    rlab.1    rxvt.1   zip.1
faces.1        funzip.1     robot.1   zcat.1   zipinfo.1
flea.1         fvwm.1       rplay.1   zcmp.1   zmore.1
flex.1         rasttoppm.1  rplayd.1  zdiff.1  znew.1
bash$ ls rp*
rplay.1        rplayd.1     rptp.1
bash$ ls *e??
face.update.1  zforce.1     zmore.1
```

Los archivos cuyo nombre comiencen por `.` se denominan **ocultos**, así por ejemplo en el directorio de partida de un usuario.

```
bash$ ls -a  user
.                .alias       .fvwmrc  .login    .xinitrc
..               .cshrc       .joverc  .profile
.Xdefaults       .enviroment  .kshrc   .tcshrc
```

[10] Normalmente se acude a la imagen de una carpeta que puede contener informes, documentos o bien otras carpetas, y así sucesivamente.

[11] Incluido el punto '.', UNIX no es DOS.

[12] `bash$` es el prompt en todos los ejemplos.

Algunos caracteres especiales para el acceso a archivos son:

.	Directorio actual
..	Directorio superior en el árbol
~	Directorio `$HOME`
~user	Directorio `$HOME` del usuario user

1.4.2 Ordenes relacionadas con directorios.

`ls` (LiSt)
Este comando permite listar los archivos de un determinado directorio. Si no se le suministra argumento, lista los archivos y directorios en el directorio actual. Si se añade el nombre de un directorio el listado es del directorio suministrado. Existen varias opciones que modifican su funcionamiento entre las que destacan:

- `-l` (Long listing) proporciona un listado extenso, que consta de los permisos[13] de cada archivo, el usuario el tamaño del archivo, ...
- `-a` (list All) lista también los archivos ocultos.
- `-R` (Recursive) lista recursivamente el contenido de todos los directorios que se encuentre.
- `-t` ordena los archivos por tiempo de modificación.
- `-S` ordena los archivos por tamaño.
- `-r` invierte el sentido de un orden.
- `-p` agrega un carácter al final de cada nombre de archivo, indicando el tipo de archivo (por ejemplo, los directorios son identificados con un `/` al final).

`pwd` (Print Working Directory)
Este comando proporciona el nombre del directorio actual.
`cd` (Change Directory)
Permite moverse a través de la estructura de directorios. Si no se le proporciona argumento se provoca un salto al directorio `$HOME`. El argumento puede ser un nombre absoluto o relativo de un directorio. `cd -` vuelve al último directorio visitado.
`mkdir` (MaKe DIRectory)
Crea un directorio con el nombre (absoluto o relativo) proporcionado.
`rmdir` (ReMove DIRectory)
Elimina un directorio con el nombre (absoluto o relativo) suministrado. Dicho directorio debe de estar vacío.

[13] Se comentará posteriormente este concepto.

1.4.3 Visitando archivos.

Este conjunto de órdenes permite visualizar el contenido de un archivo sin modificar su contenido.

`cat`
Muestra por pantalla el contenido de un archivo que se suministra como argumento.

`more`
Este comando es análogo a la anterior, pero permite la paginación.

`less`
Es una versión mejorada del anterior. Permite moverse en ambas direcciones. Otra ventaja es que no lee el archivo entero antes de arrancar.

1.4.4 Copiando, moviendo y borrando archivos.

`cp` (CoPy)
copia un archivo(s) con otro nombre y/o a otro directorio. Veamos algunas opciones:

- `-i` (interactive), impide que la copia provoque una pérdida del archivo destino si éste existe[14].
- `-R` (recursive), copia un directorio y toda la estructura que cuelga de él.

`mv` (MoVe)
Mover un archivo(s) a otro nombre y/o a otro directorio. Dispone de opciones análogas al caso anterior.

`rm` (ReMove)
Borrar un archivo(s). En caso de que el argumento sea un directorio y se haya sumnistrado la opción `-r`, es posible borrar el directorio y todo su contenido. La opción `-i` pregunta antes de borrar.

1.4.5 Espacio de disco.

El recurso de almacenamiento en el disco es siempre limitado, a continuación se comentan un par de comandos relacionados con la ocupación de este recurso:

`du` (Disk Usage)
Permite ver el espacio de disco ocupado (en bloques de disco[15]) por el archivo o directorio suministrado como argumento. La opción `-s` impide que cuando se aplique recursividad en un directorio se muestren los subtotales. La opción `-h` imprime los tamaños en un formato fácil de leer (Human readable).

`df` (Disk Free)
Muestra los sistemas de archivos que estan montados en el sistema, con las cantidades totales, usadas y disponibles para cada uno. `df -h` muestra los tamaños en formato fácil de leer.

[14] Muchos sistemas tienen esta opción habilitada a través de un alias, para evitar equivocaciones.
[15] 1 bloque normalmente es 1 Kbyte.

1.4.6 Links.

`ln` (LiNk)
Permite realizar un enlace (link) entre dos archivos o directorios. Un enlace puede ser:

- *hard link*: se puede realizar sólo entre archivos del mismo sistema de archivos. El archivo enlazado apunta a la zona de disco donde se halla el archivo original. Por tanto, si se elimina el archivo original, el enlace sigue teniendo acceso a dicha información. Es el enlace por omisión.
- *symbolic link*: permite enlazar archivos/directorios[16] de diferentes sistemas de archivos. El archivo enlazado apunta al nombre del original. Así si se elimina el archivo original el enlace apunta hacia un nombre sin información asociada. Para realizar este tipo de enlace debe emplearse la opción `-s`.

Un enlace permite el uso de un archivo en otro directorio distinto del original sin necesidad de copiarlo, con el consiguiente ahorro de espacio.

1.4.7 Protección de archivos.

Dado que el sistema de archivos UNIX es compartido por un conjunto de usuarios, surge el problema de la necesidad de privacidad. Sin embargo, dado que existen conjuntos de personas que trabajan en común, es necesario la posibilidad de que un conjunto de usuarios puedan tener acceso a una serie de archivos (que puede estar limitado para el resto de los usuarios). Cada archivo y directorio del sistema dispone de un propietario, un grupo al que pertenece y unos **permisos**. Existen tres tipos fundamentales de permisos:

- **lectura** (**r**-*Read*): en el caso de un archivo significa poder examinar el contenido del mismo; en el caso de un directorio significa poder entrar en dicho directorio.
- **escritura** (**w**-*Write*): en el caso de un archivo significa poder modificar su contenido; en el caso de un directorio es crear un archivo o directorio en su interior.
- **ejecución** (**x**-*eXecute*): en el caso de un archivo significa que ese archivo se pueda ejecutar (binario o archivo de procedimientos); en el caso de un directorio es poder ejecutar alguna orden dentro de él.

Se distinguen tres grupos de personas sobre las que especificar permisos:

- **user**: el usuario propietario del archivo.
- **group**: el grupo propietario del archivo (excepto el usuario). Como ya se ha comentado, cada usuario puede pertenecer a uno o varios grupos y el archivo generado pertenece a uno de los mismos.
- **other**: el resto de los usuarios (excepto el usuario y los usuarios que pertenezcan al grupo)

[16] Debe hacerse notar que los directorios sólo pueden ser enlazados simbólicamente.

También se puede emplear *all* que es la unión de todos los anteriores. Para visualizar las protecciones de un archivo o directorio se emplea la orden `ls -l`, cuya salida es de la forma:

`-rw-r--r-- ...otra información... nombre`

Los 10 primeros caracteres muestran las protecciones de dicho archivo:

- El primer carácter indica el tipo de archivo de que se trata:
  - archivo
  - `d` directorio
  - `l` enlace (*link*)
  - `c` dispositivo de caracteres (p.e. puerta serial)
  - `b` dispositivo de bloques (p.e. disco duro)
  - `s` socket (conexión de red)
- Los caracteres 2, 3, 4 son los permisos de usuario
- Los caracteres 5, 6, 7 son los permisos del grupo
- Los caracteres 8, 9, 10 son los permisos del resto de usuarios

  Así en el ejemplo anterior `-rw-r--r--` se trata de un archivo donde el usuario puede leer y escribir, mientras que el grupo y el resto de usuarios sólo pueden leer. Estos suelen ser los permisos por omisión para un archivo creado por un usuario. Para un directorio los permisos por omisión suelen ser:`drwxr-xr-x` donde se permite al usuario "entrar" en el directorio y ejecutar órdenes desde él.

`chmod` (CHange MODe)

Esta orden permite modificar los permisos de un archivo.

`chmod` *permisos files*

Existen dos modos de especificar los permisos:

- Modo absoluto o modo numérico. Se realiza empleando un número que resulta de la `OR` binario de los siguientes modos:

400	lectura por el propietario.
200	escritura por el propietario.
100	ejecución (búsqueda) por el propietario.
040	lectura por el grupo.
020	escritura por el grupo.
010	ejecución (búsqueda) por el grupo.
004	lectura por el resto.
002	escritura por el resto.
001	ejecución (búsqueda) por el resto.
4000	Set User ID, cuando se ejecuta el proceso corre con los permisos del dueño del archivo.

Por ejemplo:

```
chmod 640 *.txt
```

Permite la lectura y escritura por el usuario, lectura para el grupo y ningún permiso para el resto, de un conjunto de archivos que acaban en `.txt`

- Modo simbólico o literal. Se realiza empleando una cadena (o cadenas separadas por comas) para especificar los permisos. Esta cadena se compone de los siguientes tres elementos: `who operation permission`

  - `who` : es una combinación de:
    - `u` : user
    - `g` : group
    - `o` : others
    - `a` : all (equivalente a `ugo`)

    Si se omite este campo se supone `a`, con la restricción de no ir en contra de la máscara de creación (`umask`).
  - `operation`: es una de las siguientes operaciones:
    - `+` : añadir permiso.
    - `-` : eliminar permiso.
    - `=` : asignar permiso, el resto de permisos de la misma categoría se anulan.
  - `permission`: es una combinación de los caracteres:
    - `r` : *read*.
    - `w` : *write*.
    - `x` : *execute*.
    - `s` : en ejecución usar los permisos de dueño.

  Por ejemplo:
  `chmod u+x tarea`
  Permite la ejecución por parte del usuario[17] del archivo `tarea`.

  `chmod u=rx, go=r *.txt`
  Permite la lectura y ejecución del usuario, y sólo la lectura por parte del grupo y el resto de usuarios. La opción `-r` hace que la orden se efectúe recursivamente.

`umask`
Esta es una orden intrínseca del Shell que permite asignar los permisos que se desea tengan los archivos y directorios por omisión. El argumento que acompaña a la orden es un número octal que aplicará una XOR sobre los permisos por omisión (`rw-rw-rw-`) para archivos y (`rwxrwxrwx`) para directorios. El valor por omisión de la máscara es `022` que habilita al usuario para lectura-escritura, al grupo y al resto para lectura. Sin argumentos muestra el valor de la máscara.

[17] Un error muy frecuente es la creación de un archivo de órdenes (*script file*) y olvidar y permitir la ejecución del mismo.

`chgrp` (CHange GRouP)
Cambia el grupo propietario de una serie de archivos/directorios
`chgrp` *grupo files*
El usuario que efectúa esta orden debe de pertenecer al grupo mencionado.
`chown` (CHange OWNer)
Cambia el propietario y el grupo de una serie de archivos/directorios
`chown` *user:group files*
La opción `-r` hace que la orden se efectúe recursivamente.
`id`
Muestra la identificación del usuario[18], así como el conjunto de grupos a los que el usuario pertenece.

```
user@hostname:~$ id
uid=1000(user) gid=1000(group) groups=1000(group),25(floppy),29(audio)
user@hostname:~$
```

1.4.8 Filtros.

Existe un conjunto de órdenes en UNIX que permiten el procesamiento de archivos de texto. Se denominan **filtros** (*Unix Filters*) porque normalmente se trabaja empleando redirección recibiendo datos por su `stdin`[19] y retornándolos modificados por su `stdout`[20].

Para facilitar la comprensión de los ejemplos siguientes supondremos que existe dos archivo llamado `mylist.txt` y `yourlist.txt` que tienen en su interior:

```
mylist.txt       yourlist.txt

1 190            1 190
2 280            2 281
3 370            3 370
```

`awk`
Es un procesador de archivos de texto que permite la manipulación de las líneas de forma tal que tome decisiones en función del contenido de la misma. Ejemplo, supongamos que tenemos nuestro archivo `mylist.txt` con sus dos columnas

```
user@hostname:~$ awk '{print $2, $1 }'  mylist.txt
190 1
280 2
370 3
user@hostname:~$
```

Imprime esas dos columnas en orden inverso.
`cat`
Es el filtro más básico, copia la entrada a la salida.

[18] A pesar de que el usuario se identifica por una cadena denominada *username*, también existe un número denominado UID que es un identificativo numérico de dicho usuario.
[19] Entrada estándar.
[20] Salida estándar.

```
user@hostname:~$ cat  mylist.txt
1 190
2 280
3 370
user@hostname:~$
```

`cut`

Para un archivo compuesto por columnas de datos, permite escribir sobre la salida cierto intervalo de columnas. La opción `-b N-M` permite indicar el intervalo en bytes que se escribiran en la salida.

```
user@hostname:~$ cut -b 3-4 mylist.txt
19
28
37
user@hostname:~$
```

`diff`

Permite comparar el contenido de dos archivos

```
user@hostname:~$ diff mylist.txt yourlist.txt
2c2
< 2 280
---
> 2 281
user@hostname:~$
```

Hay una diferencia entre los archivos en la segunda fila.

`find`

Permite la búsqueda de un archivo en la estructura de directorios

`find . -name file.dat -print`

Comenzando en el directorio actual recorre la estructura de directorios buscando el archivo `file.dat`, cuando lo encuentre imprime el path al mismo.

`find . -name '*~' -exec rm '{}' \;`

Busca en la estructura de directorios un archivo que acabe en `~` y lo borra. `xargs` ordena repetir orden para cada argumento que se lea desde *stdin*. Permite el uso muy eficiente de `find`.

`find . -name '*.dat' -print | xargs mv ../data \;`

Busca en la estructura de directorios todos los archivos que acaben en `.dat`, y los mueve al directorio `../data`.

`grep`

Permite la búsqueda de una cadena de caracteres en uno o varios archivos, imprimiendo el nombre del archivo y la línea en que encuentra la cadena.

```
user@hostname:~$ grep 1 *list.txt
mylist.txt:1 190
yourlist.txt:1 190
yourlist.txt:2 281
user@hostname:~$
```

Algunas opciones útiles

- `-c` Elimina la salida normal y sólo cuenta el número de apariciones de la cadena en cada archivos.
- `-i` Ignora para la comparación entre la cadena dada y el archivo si la cadena está en mayúsculas o minúsculas.
- `-n` Incluye el número de líneas en que aparece la cadena en la salida normal.
- `-r` La busqueda la hace recursiva.
- `-v` Invierte la busqueda mostrando todas las líneas donde no aparece al cadena pedida.

`head`
Muestra las primeras diez líneas de un archivo.
`head -30 file` Muestra las 30 primeras líneas de *file*.

```
user@hostname:~$ head -1  mylist.txt
1 190
user@hostname:~$
```

`tail`
Muestra las diez últimas líneas de un archivo.
`tail -30 file` Muestra las 30 últimas líneas de *file*.
`tail +30 file` Muestra desde la línea 30 en adelante de *file*.

```
user@hostname:~$ tail -1  mylist.txt
3 370
user@hostname:~$
```

`tar`
Este comando permite la creación/extracción de archivos contenidos en un único archivo denominado `tarfile` (o `tarball`). Este `tarfile` suele ser luego comprimido con `gzip` la versión de compresión **gnu**[21] o bien con `bzip2`.

La acción a realizar viene controlada por el primer argumento:

- `c` (Create) creación
- `x` (eXtract) extracción
- `t` (lisT) mostrar contenido
- `r` añadir al final
- `u` (Update) añadir aquellos archivos que no se hallen en el tarfile o que hayan sido modificados con posterioridad a la versión que aparece.

[21] **gnu** es un acrónimo recursivo, significa: **gnu**'s Not UNIX! **gnu** es el nombre del producto de la *Free Software Foundation*, una organización dedicada a la creación de programas compatible con UNIX (y mejorado respecto a los estándars) y de libre distribución. La distribución de Linux **gnu** es **debian**.

A continuación se colocan algunas de las opciones:

- `v` Verbose (indica qué archivos son agregados a medida que son procesados)
- `z` Comprimir o descomprimir el contenido con `gzip`.
- `I` Comprimir o descomprimir el contenido con `bzip2`.
- `f` File: permite especificar el archivo para el tarfile.

Veamos algunos ejemplos:
`tar cvf simul.tar *.dat`
Genera un archivo `simul.tar` que contiene todos los archivos que terminen en `.dat` del directorio actual. A medida que se va realizando indica el tamaño en bloques de cada archivo añadido modo *verbose*.
`tar czvf simul.tgz *.dat`
Igual que en el caso anterior, pero el archivo generado simul.tgz ha sido comprimido empleando gzip.
`tar tvf simul.tar`
Muestra los archivos contenidos en el tarfile `simul.tar`.
`tar xvf simul.tar`
Extrae todos los archivos contenidos en el tarfile `simul.tar`.

`wc` (*Word Count*) Contabiliza el número de líneas, palabras y caracteres de un archivo.

```
user@hostname:~$ wc  mylist.txt
      3       6      18 mylist.txt
user@hostname:~$
```

El archivo tiene 3 líneas, 6 palabras, al considerar cada número como una palabra *i.e.* 1 es la primera palabra y 190 la segunda, y finalmente 18 caracteres. ¿Cuáles son los 18 caracteres?

1.4.9 Otros usuarios y máquinas

`users` `who` `w`
Para ver quién está conectado en la máquina.

`ping`
Verifica si una máquina está conectada a la red y si el camino de Internet hasta la misma funciona correctamente.

`finger`
`finger user`, muestra información[22] sobre el usuario `user` en la máquina local.
`finger user@hostname`, muestra información sobre un usuario llamado `user` en una máquina `hostname`.
`finger @hostname`, muestra los usuarios conectados de la máquina `hostname`.

[22]La información proporcionada es el nombre de completo del usuario, las últimas sesiones en dicha máquina, si ha leido o no su correo y el contenido de los archivos `.project` y `.plan` del usuario.

1.4.10 Fecha

`cal`
Muestra el calendario del mes actual. Con la opción `-y` y el año presenta el calendario del año completo.

`date`
Muestra el día y la hora actual.

1.4.11 Transferencia a diskettes.

La filosofía de diferentes unidades (A:, B:, ...) difiere de la estructura única del sistema de archivos que existe en UNIX. Son varias las alternativas que existen para la transferencia de información a diskette.

- Una posibilidad es disponer de una máquina WIN9X con ftp instalado y acceso a red. Empleando dicha aplicación se pueden intercambiar archivos entre un sistema y el otro.

- Existe un conjunto de comandos llamados `mtools` disponible en multitud plataformas, que permiten el acceso a diskettes en formato WIN9X de una forma muy eficiente.

  `mdir a:` Muestra el contenido de un diskette en `a:`.

  `mcopy file a:` Copia el archivo `file` del sistema de archivos UNIX en un diskette en `a:`.

  `mcopy a:file file` Copia el archivo `a:file` del diskette en el sistema de archivos UNIX con el nombre `file`.

  `mdel a:file` Borra el archivo `a:file` del diskette.

  Con `a:` nos referimos a la primera diskettera `/dev/fd0`, luego el archivo que se encuentra en el diskette su nombre se compone de `a:filename`. Si se desea emplear el caracter comodín para un conjunto de archivos del diskette debe de rodearse de dobles comillas el mismo para evitar la actuación del *shell* (p.e. `mcopy ‘‘a:*.dat’’`). La opción `-t` realiza la conversión necesaria entre UNIX y WIN9X, que se debe realizar **sólo** en archivos de texto.

- Una alternativa final es montar el dispositivo `/dev/fd0` en algún directorio, típicamente `/floppy`, considerando el tipo especial de sistema de archivos que posee `vfat` y luego copiar y borrar usando comandos UNIX. Hay que hacer notar que esta forma puede estar restringida sólo a `root`, el comando: `mount -t vfat /dev/fd0 /floppy`

1.4.12 Diferencias entre los sistemas.

Cuando se transfieren archivos de texto entre DOS y UNIX sin las precauciones adecuadas pueden aparecer los siguientes problemas:

- En DOS los nombres de los archivos pueden tener un máximo de 8 caracteres y una extensión de 3 caracteres. En UNIX no existe restricción respecto a la longitud del

nombre, y aunque pueden llevar extensión, no es obligatorio. También pueden tener más de una extensión `algo.v01.tar.gz` esto complica mucho a otros sistemas que tienen limitaciones en los nombres.

- El cambio de línea en DOS se compone de *Carriage Return* y *Line Feed*. Sin embargo, en UNIX sólo existe el *Carriage Return*. Así un archivo de UNIX visto desde DOS parece una única línea. El caso inverso es la aparición del carácter `^M` al final de cada línea. Además, el fin de archivo en DOS es `^Z` y en UNIX es `^D`.

- La presencia de caracteres con código ASCII por encima del 127 (ASCII extendido) suele plantear problemas. Debido a que en DOS dicho código depende de la asignación hecha, que a su vez depende del país.

1.5 Shells.

El sistema operativo UNIX soporta varios intérpretes de comandos o *shells*, que ayudan a que la interacción con el sistema sea lo más cómoda y amigable posible. La elección de cuál es el *shell* más cómoda es algo personal; en este punto sólo indicaremos las cuatro más significativas y populares:

- **sh** : Bourne SHell, el *shell* básico, no pensado para uso interactivo.

- **csh** : C-SHell, *shell* con sintaxis como el lenguaje "C". El archivo de configuración es `.cshrc` (en el directorio `$HOME`).

- **tcsh** : alTernative C-Shell (Tenex-CSHell), con editor de línea de comando. El archivo de configuración es `.tcshrc`, o en caso de no existir, `.cshrc` (en el directorio `$HOME`).

- **bash** : Bourne-Again Shell, con lo mejor de sh, ksh y tcsh. El archivo de configuración es `.bash_profile` cuando se entra a la cuenta por primera vez, y después el archivo de configuración es `.bashrc` siempre en el directorio `$HOME`. La línea de comando puede ser editada usando comandos (secuencias de teclas) del editor `emacs`. Es el *shell* por defecto de Linux.

Si queremos cambiar de *shell* en un momento dado, sólo será necesario que tecleemos el nombre del mismo y estaremos usando dicho *shell*. Si queremos usar de forma permanente otro *shell* del que tenemos asignado por omisión[23] podemos emplear la orden `chsh` que permite realizar esta acción.

En los archivos de configuración se encuentran las definiciones de las variables de entorno (*enviroment variables*) como camino de búsqueda `PATH`, los aliases y otras configuraciones personales. Veamos unos caracteres con especial significado para el Shell:

- ` ‘ ` [24]permite que el output de un comando reemplace al nombre del comando. Por ejemplo: `echo ‘pwd‘` imprime por pantalla el nombre del directorio actual.

[23]Por omisión se asigna bash.

[24]Acento agudo o inclinado hacia atrás, *backquote*.

```
user@hostname:~$ echo ‘pwd‘
/home/user
user@hostname:~$
```

- ’ [25] preserva el significado literal de cada uno de los caracteres de la cadena que delimita.

```
user@hostname:~$ echo ’Estoy en ‘pwd‘’
Estoy en ‘pwd‘
user@hostname:~$
```

- ” [26] preserva el significado literal de todos los caracteres de la cadena que delimita, salvo $, ‘, \.

```
user@hostname:~$ echo "Estoy en ‘pwd‘"
Estoy en /home/user
user@hostname:~$
```

- ; permite la ejecución de más de una orden en una sola línea de comando.

```
user@hostname:~$ mkdir mydir; cd mydir; cp *.txt . ; cd ..
user@hostname:~$
```

1.5.1 Variables de entorno.

Las variables de entorno permiten la configuración, por defecto, de muchos programas cuando ellos buscan datos o preferencias. Se encuentran definidas en los archivos de configuración anteriormente mencionados. Para referenciar a las variables se debe poner el símbolo $ delante, por ejemplo, para mostrar el camino al directorio por defecto del usuario `user`:

```
user@hostname:~$ echo $HOME
/home/user
user@hostname:~$
```

Las variables de entorno más importantes son:

- `HOME` - El directorio por defecto del usuario.
- `PATH` - El camino de búsqueda, una lista de directorios separado con ‘:’ para buscar programas.
- `EDITOR` - El editor por defecto del usuario.
- `DISPLAY` - Bajo el sistema de X windows, el nombre de máquina y pantalla que está usando. Si esta variable toma el valor `:0` el despliegue es local.

[25] Acento usual o inclinado hacia adelante, *single quote*.
[26] *double quote*.

- `TERM` - El tipo de terminal. En la mayoría de los casos bajo el sistema X windows se trata de `xterm` y en la consola en Linux es `linux`, en otros sistemas puede ser `vt100`.
- `SHELL` - La *shell* por defecto.
- `MANPATH` - Camino para buscar páginas de manuales.
- `PAGER` - Programa de paginación de texto (`less` o `more`).
- `TMPDIR` - Directorio para archivos temporales.

1.5.2 Redirección.

Cuando un programa espera que se teclee algo, aquello que el usuario teclea se conoce como el *Standard Input*: `stdin`. Los caracteres que el programa retorna por pantalla es lo que se conoce como *Standard Output*: `stdout` (o *Standard Error*: `stderr`[27]). El signo `<` permite que un programa reciba el `stdin` desde un archivo en vez de la interacción con el usuario. Por ejemplo: `mail root < file`, invoca el comando `mail` con argumento (destinatario del mail) `root`, siendo el contenido del mensaje el contenido del archivo `file` en vez del texto que usualmente teclea el usuario. Más a menudo aparece la necesidad de almacenar en un archivo la salida de un comando. Para ello se emplea el signo `>`. Por ejemplo, `man bash > file`, invoca el comando `man` con argumento (información deseada) `bash` pero indicando que la información debe ser almacenada en el archivo `file` en vez de ser mostrada por pantalla.

En otras ocasiones uno desea que la salida de un programa sea la entrada de otro. Esto se logra empleando los denominados *pipes*, para ello se usa el signo `|`. Este signo permite que el `stdout` de un programa sea el `stdin` del siguiente. Por ejemplo:

```
zcat  manual.gz | more
```

Invoca la orden de descompresión de `zcat` y conduce el **flujo** de caracteres hacia el paginador `more`, de forma que podamos ver página a página el archivo descomprimido. A parte de los símbolos mencionados existen otros que permiten acciones tales como:

- `>>` Añadir el `stdout` al final del archivo indicado (*append*).[28]
- `>&` o `&>` (sólo `csh`, `tcsh` y `bash`) Redireccionar el `stdout` y `stderr`. Con `2>` redirecciono sólo el `stderr`.
- `>>&` Igual que `>&` pero en modo *append*.
- `>>!` Igual que `>>` pero con la adición que funciona también cuando el archivo no existe.

1.5.3 Ejecución de comandos.

- Si el comando introducido es propio del *shell* (*built-in*), se ejecuta directamente.
- En caso contrario:

[27] Si estos mensajes son de error.
[28] En `bash` si el archivo no existe es creado.

- Si el comando contiene `/`, el *shell* lo considera un `PATH` e intenta resolverlo (entrar en cada directorio especificado para encontrar el comando).
- En caso contrario el *shell* busca en una tabla *hash table* que contiene los nombres de los comandos que se han encontrado en los directorios especificados en la variable `PATH`, cuando ha arrancado el *shell*.

1.5.4 Aliases.

Para facilitar la entrada de algunas órdenes o realizar operaciones complejas, los *shells* interactivos permiten el uso de aliases. La orden `alias` permite ver qué aliases hay definidos y también definir nuevos. Es corriente definir el alias `rm =‘rm -i’`, de esta forma la orden siempre pide confirmación para borrar un archivo. Si alguna vez quieres usar `rm` sin alias sólo hace falta poner delante el símbolo \, denominado *backslash* . Por ejemplo `\rm` elimina los alias aplicados a `rm`. Otro ejemplo, bastante frecuente (en `tcsh/csh`) podría ser (debido a la complejidad de la orden): `alias ffind 'find . -name \!* -print'` Para emplearlo: `ffind tema.txt` el resultado es la búsqueda recursiva a partir del directorio actual de un archivo que se llame `tema.txt`, mostrando el camino hasta el mismo.

1.5.5 Las shells csh y tcsh.

Son dos de los Shells interactivos más empleados. Una de las principales ventajas de `tcsh` es que permite la edición de la línea de comandos, y el acceso a la historia de órdenes usando las teclas de cursores.[29]

Comandos propios.

Los comandos propios o intrínsecos *Built-In Commands* son aquéllos que proporciona el propio *shell* [30].

`alias name def`
Asigna el nombre `name` al comando `def`.

```
foreach var (wordlist)
   commands
end
```

La variable `var` se asigna sucesivamente a los valores de cadena `wordlist`, y se ejecutan el conjunto de comandos. El contenido de dicha variable puede ser empleado en los comandos: `$var`.

`history`

[29] `bash` también lo permite.

[30] A diferencia de los comandos que provienen de un ejecutable situado en alguno de los directorios de la variable `PATH`.

Muestra las últimas órdenes introducidas en el *shell*. Algunos comandos relacionados con el *Command history* son:

- `!!`
  Repite la última orden.
- `!n`
  Repite la orden n-ésima.
- `!string`
  Repite la orden más reciente que empiece por la cadena `string`.
- `!?string`
  Repite la orden más reciente que contenga la cadena `string`.
- `^str1^str2` o `!!:s/str1/str2/`
  (*substitute*) Repite la última orden reemplanzando la primera ocurrencia de la cadena `str1` por la cadena `str2`.
- `!!:gs/str1/str2/`
  (*global substitute*) Repite la última orden reemplazando todas las ocurrencias de la cadena `str1` por la cadena `str2`.
- `!$`
  Es el último argumento de la orden anterior que se haya tecleado.

`pushd`
Cambia de directorio, recordando el directorio actual.

`popd`
Retorna al directorio desde donde se hizo `pushd` la última vez.

`repeat count command`
Repite `count` veces el comando `command`.

`rehash`
Rehace la tabla de comandos (*hash table*).

`set variable = VALUE`
Asigna el valor de una variable del *shell*.

`set variable`
Muestra el valor de la variable

`setenv VARIABLE VALUE`
Permite asignar el valor de una variable de entorno.

`source file`
Ejecuta las órdenes del fichero `file` en el *shell* actual.

`unset variable`

Desasigna el valor de una variable del *shell*.

`unsetenv VARIABLE VALUE`

Permite desasignar el valor de una variable de entorno.

`umask value`

Asigna la máscara para los permisos por omisión.

`unalias name`

Elimina un alias asignado.

Variables propias del shell.

Existe un conjunto de variables denominadas *shell variables*, que permiten modificar el funcionamiento del *shell*.

`filec` (*FILE Completion*)

Es una variable *toggle* que permite que el *shell* complete automáticamente el nombre de un archivo o un directorio[31]. Para ello, si el usuario introduce sólo unos cuantos caracteres de un archivo y pulsa el `TAB` el *shell* completa dicho nombre. Si sólo existe una posibilidad, el completado es total y el *shell* deja un espacio tras el nombre. En caso contrario hace sonar un pitido. Pulsando `Ctrl-D` el *shell* muestra las formas existentes para completar.

`prompt`

Es una variable de cadena que contiene el texto que aparece al principio de la línea de comandos.

`savehist`

Permite definir el número de órdenes que se desea se almacenen al abandonar el *shell*. Esto permite recordar las órdenes que se ejecutaron en la sesión anterior.

1.5.6 Las shell sh y bash.

Sólo `bash` puede considerarse un *shell* interactivo, permitiendo la edición de la línea de comandos, y el acceso a la historia de órdenes (*readline*). En uso normal (historia y editor de línea de comandos) `BASH` es compatible con `TCSH` y `KSH`. El modo de completado (*file completion*) es automático (usando `TAB` sólo) si el *shell* es interactivo.

Comandos propios del shell.

Los comandos `umask`, `source`, `pushd`, `popd`, `history`, `unalias`, `hash` [32], funcionan igual que en la *shell* `TCSH`.

`help`

Ayuda interna sobre los comandos del *shell*.

[31] `bash` permite no sólo completar ficheros/directorios sino también comandos.

[32] En `bash/sh` la *hash table* se va generando dinámicamente a medida que el usuario va empleando las órdenes. Así el arranque del *shell* es más rápido, y el uso de orden equivalente `hash -r` casi nunca hace falta.

`VARIABLE=VALUE`

Permite asignar el valor de una variable de entorno. Para que dicha variable sea "heredada" es necesario emplear: `export VARIABLE` o bien combinarlas: `export VARIABLE=VALUE`.

`alias`

En `bash alias` sólo sirve para substitución simple de una cadena por otra, Por ejemplo: `alias ls='ls -F'`. Para crear aliases con argumentos se usan funciones.

Las funciones se definen con `()` y los comandos a realizar entre llaves `{}`. El empleo de los argumentos se realiza mediante `$0`, . . . ,`$n`, siendo `$#` el número de argumentos. Por ejemplo:

```
setenv() {
if [ $# -gt 1 ]; then
export $1=$2
 else
env
 fi
}+
```

o bien,
`setenv () { if [ $# -gt 1 ]; then export $1=$2 ; else env; fi;}`
define una función igual que el `setenv` de `tcsh`.

El siguiente código define una función equivalente al alias `ffind` de `tcsh`:

```
ffind() {
if [ $# != 1 ]; then
echo Error, falta argumento
else
find . -name \$1 -print
fi
}
```

Las funciones pueden usar todas las órdenes de *shell* y de UNIX presentando una forma muy potente para construir aliases.

1.6 Ayuda y documentación.

Para obtener ayuda sobre comandos de UNIX, se puede emplear la ayuda *on-line*, en la forma de páginas de manual. Asi `man comando` proporciona la ayuda sobre el `comando` deseado. Por ejemplo, para leer el manual de los shells, puedes entrar: `man sh csh tcsh bash` la orden formatea las páginas y te permite leer los manuales en el orden pedido. En el caso de `bash` se puede usar el comando `help`, por ejemplo, `help alias`. Además, para muchos comandos y programas se puede obtener información y tipeando `info comando`. Finalmente, algunos comandos tienen una opción de ayuda (`--help`), para recordar rápidamente las opciones más comunes disponibles (`ls --help`).

1.7 Procesos.

En una máquina existen una multitud de procesos que pueden estar ejecutándose simultánemente. La mayoría de ellos no corresponden a ninguna acción realizada por el usuario y no merecen que se les preste mayor atención. Estos procesos corresponden a programas ejecutados en el arranque del sistema y tienen que ver con el funcionamiento global del servidor. En general, los programas suelen tener uno de estos dos modos de ejecución:

- **foreground**: Son aquellos procesos que requieren de la interacción y/o atención del usuario mientras se están ejecutando, o bien en una de sus fases de ejecución (*i.e.* Introducción de datos). Así por ejemplo, una consulta de una página de manual es un proceso que debe ejecutarse claramente en *foreground*.
- **background**: Son aquellos procesos que no requieren de la interacción con el usuario para su ejecución. Si bien el usuario desearía estar informado cuando éste proceso termine. Un ejemplo de este caso sería la impresión de un archivo.

Sin embargo, esta división que a primera vista pueda parecer tan clara y concisa, a menudo en la práctica aparece la necesidad de conmutar de un modo al otro, detención de tareas indeseadas, etc. Así por ejemplo, puede darse el caso de que estemos leyendo una página de manual y de repente necesitemos ejecutar otra tarea. Un proceso viene caracterizado por:

- *process number*
- *job number*

Veamos algunas de las órdenes más frecuentes para la manipulación de procesos:

- `comando &` Ejecución de un comando en el *background*. [33]
- `Ctrl-Z` Detiene el proceso que estuviera ejecutándose en el *foreground* y lo coloca detenido en el *background*.
- `Ctrl-C` Termina un proceso que estaba ejecutándose en *foreground*.
- `Ctrl-\` Termina de forma definitiva un proceso que estaba ejecutándose en *foreground*.
- `ps x` Lista todos los procesos que pertenezcan al usuario, incluyendo los que no están asociados a un terminal.
- `jobs` Lista los procesos que se hayan ejecutado desde el *shell* actual, mostrando el *job number*.
- `fg (job number)` Pasa a ejecución en *foreground* un proceso que se hallase en *background*.
- `bg (job number)` Pasa a ejecución en *background* un proceso que se hallase detenido con `Ctrl-Z`.

[33] Por omisión un comando se ejecuta siempre en el *foreground*.

- `kill (process number)` Envía una señal[34] a un proceso UNIX. En particular para enviar la señal de término a un programa, damos el comando `kill -KILL`, pero no hace falta al ser la señal por defecto.

Cuando se intenta abandonar una sesión con algún proceso aún detenido en el *background* del *shell*, se informa de ello con un mensaje del tipo: `There are stopped jobs` si no importa, el usuario puede intentar abandonar de nuevo el *shell* y éste matará los *jobs*, o puedes utilizar `fg` para traerlos al *foreground* y ahí terminar el mismo.

1.8 Editores.

Un editor es un programa que permite crear y/o modificar un archivo. Existen multitud de editores diferentes, y al igual que ocurre con los *shells*, cada usuario tiene alguno de su predilección. Mencionaremos algunos de los más conocidos:

- **vi** - El editor standard de UNIX.

- **emacs (xemacs)** - Editor muy configurable escrito en lenguaje Lisp. Existen multitud de modos para este editor (lector de mail, news, www, ...) que lo convierten en un verdadero *shell* para multitud de usuarios. Las últimas versiones del mismo permiten la ejecución desde X-windows o terminal indistintamente con el mismo binario. Posee un tutorial en línea, comando `C-H t` dentro del editor. El archivo de configuración personalizada es: `$HOME/.emacs`.

- **jove** - Basado en Emacs, (Jonathan's Own Version of Emacs). Posee tutorial en una utilidad asociada: `teachjove`. El archivo de configuración personalizada es: `$HOME/.joverc`.

- **jed** - Editor configurable escrito en S-Lang. Permite la emulación de editores como emacs, edt[35] y Wordstar. Posee una ayuda en línea `C-H C-H`. El archivo de configuración personalizada es: `$HOME/.jedrc`.

- **gedit** - Editor por defecto de gnome.

- **xjed** - Versión de jed para el X-windows system. Presenta como ventaja que es capaz de funcionar en muchos modos: lenguaje C, Fortran, TeX, etc, reconociendo palabras clave y signos de puntuación, empleando un colorido distinto para ellos. El archivo de configuración personalizada es el mismo que el de de jed.

Dado que los editor del tipo de **gedit** disponen de menús autoexplicativos, daremos a continuación unas ligeras nociones sólo de `vi` y `emacs`.

[34] Para ver las señales disponibles entra la orden `kill -l` (l por *list*).
[35] Para usuarios VMS.

1.8.1 El editor vi.

El `vi` es un editor de texto muy poderoso pero un poco difícil de usar. Lo importante de este editor es que se puede encontrar en cualquier sistema UNIX y sólo hay unas pocas diferencias entre un sistema y otro. Explicaremos lo básico solamente. Comencemos con el comando para invocarlo:

```
localhost:/# vi
```

```
~
~
~
/tmp/vi.9Xdrxi: new file: line 1
```

La sintaxis para editar un archivo es:

```
localhost:/# vi nombre.de.archivo
```

```
~
~
~
nombre.de.archivo: new file: line 1
```

Insertar y borrar texto en vi.

Cuando se inicia el `vi`, editando un archivo, o no, se entra en un modo de órdenes, es decir, que no se puede empezar a escribir directamente. Si se quiere entrar en modo de inserción de texto se debe presionar la tecla `i`. Entrando en el modo de inserción, se puede empezar a escribir. Para salir del modo de inserción de texto y volver al modo de órdenes se aprieta `ESC`.

```
Aqui ya estamos escribiendo porque apretamos
la tecla 'i' al estar en modo ordenes.
~
~
```

La tecla `a` en el modo de órdenes también entra en modo de inserción de texto, pero en vez de comenzar a escribir en la posición del cursor, empieza un espacio después.

La tecla `o` en el modo de órdenes inserta texto pero desde la línea que sigue a la línea donde se está ubicado.

Para borrar texto, hay que salir al modo órdenes, y presionar la tecla `x` que borrará el texto que se encuentre sobre el cursor. Si se quiere borrar las líneas enteras, entonces se debe presionar dos veces la tecla `d` sobre la línea que deseo eliminar. Si se presionan las teclas `dw` se borra la palabra sobre la que se está ubicado.

La letra `R` sobre una palabra se puede escribir encima de ella. Esto es una especie de modo de inserción de texto pero sólo se podrá modificar la palabra sobre la que se está situado. La tecla `~` cambia de mayúscula a minúscula la letra sobre la que se está situado.

Moverse dentro de vi.

Estando en modo ordenes podemos movernos por el archivo que se está editando usando las flechas hacia la izquierda, derecha, abajo o arriba. Con la tecla `0` nos movemos al comienzo de la línea y con la tecla `$` nos movemos al final de la misma.

Con las teclas `w` y `b` nos movemos al comienzo de la siguiente palabra o al de la palabra anterior respectivamente. Para moverme hacia la pantalla siguiente la combinacion de teclas `CTRL F` y para volver a la pantalla anterior `CTRL B`. Para ir hasta el principio del archivo se presiona la tecla `G`.

Opciones de comandos.

Para entrar al menú de comandos se debe presionar la tecla `:` en el modo de órdenes. Apareceran los dos puntos (`:`). Aquí se pueden ingresar ordenes para guardar, salir, cambiar de archivo entre otras cosas. Veamos algunos ejemplos:

- `:w` Guardar los cambios.
- `:w otherfile.txt` Guardar con el nuevo nombre `otherfile.txt`
- `:wq` Guardar los cambios y salir.
- `:q!` Salir del archivo sin guardar los cambios.
- `:e file1.txt` Si deseo editar otro archivo al que se le pondrá por nombre `file1.txt`.
- `:r file.txt` Si se quiere insertar un archivo que ya existente, por ejemplo `file.txt`.
- `:r! comando` Si se quiere ejecutar algún comando del *shell* y que su salida aparezca en el archivo que se está editando.

1.8.2 Editores modo emacs.

El editor `GNU Emacs`, escrito por Richard Stallman de la *Free Software Foundation*, es uno de los que tienen mayor aceptación entre los usuarios de UNIX, estando disponible bajo licencia **GNU GPL**[36] para una gran cantidad de arquitecturas. También existe otra versión de emacs llamada `XEmacs` totalmente compatible con la anterior pero presentando mejoras significativas respecto al `GNU Emacs`. Dentro de los "inconvenientes" que presenta es que no viene por defecto incluido en la mayoría de los sistemas UNIX. Las actuales distribuciones de Linux y en particular Debian GNU/Linux contienen ambas versiones de emacs, tanto `GNU Emacs` como `XEmacs`, como también versiones de `jove`, `jed`, `xjed` y muchos otros editores.

Los editores tipo emacs se parecen mucho y en su mayoría sus comandos son los mismos. Para ejemplificar este tipo de editores nos centraremos en `XEmacs`, pero los comandos y descripciones se aplican casi por igual a todos ellos. Los editores tipo emacs constan de tres zonas:

- La zona de edición: donde aparece el texto que está siendo editado y que ocupa la mayor parte de la pantalla.
- La zona de información: es una barra que esta situada en la penúltima línea de la pantalla.
- La zona de introducción de datos: es la última línea de la pantalla.

Emacs es un editor que permite la edición visual de un archivo (en constraste con el modo de edición de `vi`). El texto se agrega o modifica en la zona de edición, usando las teclas disponibles en el teclado.

Además, existen una serie de comandos disponibles para asistir en esta tarea.

La mayoría de los comandos de emacs se realizan empleando la tecla de `CONTROL` o la tecla `META`[37]. Emplearemos la nomenclatura: `C-key` para indicar que la tecla `key` debe de ser pulsada junto con `CONTROL` y `M-key` para indicar que la tecla `META` debe de ser pulsada junto a `key`. En este último caso NO es necesario pulsar simultáneamente las teclas `ESC` y `key`, pudiendo pulsarse secuencialmente `ESC` y luego `key`, sin embargo, si se usa `ALT` como `META` deben ser pulsadas simultáneamente. Observemos que en un teclado normal hay unos 50 caracteres (letras y números). Usando `SHIFT` se agregan otros 50. Así, usando `CONTROL` y `META`, hay unos $50 \cdot 4 = 200$ comandos disponibles. Además, existen comandos especiales llamados *prefijos*, que modifican el comando siguiente. Por ejemplo, `C-x` es un prefijo, y si `C-s` es un comando (de búsqueda en este caso), `C-x C-s` es otro (grabar archivo). Así, a través de un prefijo, se duplican el número de comandos disponibles sólo con el teclado, hasta llegar a unos $200 \cdot 2 = 400$ comandos en total.

Aparte de estos comandos accesibles por teclas, algunos de los cuales comentaremos a continuación, existen comandos que es posible ejecutar por nombre, haciendo así el número de comandos disponibles virtualmente infinito.

Revisemos los comandos más usuales, ordenados por tópico.

[36] La licencia de GNU, da el permiso de libre uso de los programas con su fuentes, pero los autores mantienen el *Copyright* y no es permitido distribuir los binarios sin acesso a sus fuentes, los programas derivados de dichos fuentes heredan la licencia GNU.

[37] Dado que la mayoría de los teclados actuales no poseen la tecla `META` se emplea ya sea `ESC` o `ALT`.

Abortar y deshacer

En cualquier momento, es posible abortar la operación en curso, o deshacer un comando indeseado:

C-g	abortar
C-x u	deshacer

Archivos

C-x C-f	cargar archivo
C-x i	insertar archivo
C-x C-s	grabar archivo
C-x C-w	grabar con nombre
C-x C-c	salir

Ventanas

`Emacs` permite dividir la pantalla en varias ventanas. En cada ventana se puede editar texto e ingresar comandos independientemente. Esto es útil en dos situaciones: a) si necesitamos editar un solo archivo, pero necesitamos ver su contenido en dos posiciones distintas (por ejemplo, el comienzo y el final de archivos muy grandes); y b) si necesitamos editar o ver varios archivos simultáneamente. Naturalmente, aunque son independientes, sólo es posible editar un archivo a la vez. A la ventana en la cual se encuentra el cursor en un momento dado le llamamos la "ventana actual".

C-x 2	dividir ventana actual en 2 partes, con línea horizontal
C-x 3	dividir ventana actual en 2 partes, con línea vertical
C-x 1	sólo 1 ventana (la ventana actual, eliminando las otras)
C-x 0	elimina sólo la ventana actual
C-x o	cambia el cursor a la siguiente ventana

El cambio del cursor a una ventana cualquiera se puede hacer también rápidamente a través del *mouse*.

Comandos de movimiento

Algunos de estos comandos tienen dos teclas asociadas, como se indica a continuación.

C-b o ←	izquierda un carácter	**C-f** o →	derecha un carácter
C-p o ↑	arriba una línea	**C-n** o ↓	abajo una línea
C-a o `Home`	principio de la línea	**C-e** o `End`	fin de la línea
M-< o **C-**`Home`	principio del documento	**M->** o **C-**`End`	fin del documento
M-f o **M-**→	avanza una palabra	**M-b** o **M-**←	retrocede una palabra
C-v o `Page Up`	avanza una página	**M-v** o `Page Down`	retrocede una página
M-g (número)	salta a la línea (número)	**C-l**	refresca la pantalla

Comandos de inserción y borrado

Al ser un editor en modo visual, las modificaciones se pueden hacer en el texto sin necesidad de entrar en ningún modo especial.

C-d o `Delete`	borra un carácter después del cursor
`Backspace`	borra un carácter antes del cursor
C-k	borra desde la posición del cursor hasta el fin de línea (no incluye el cambio de línea)
M-d	borra desde el cursor hacia adelante, hasta que termina una palabra
M-`Backspace`	borra desde el cursor hacia atrás, hasta que comienza una palabra
C-o	Inserta una línea en la posición del cursor

Mayúsculas y minúsculas

M-u	Cambia a mayúscula desde la posición del cursor hasta el fin de la palabra
M-l	Cambia a minúscula desde la posición del cursor hasta el fin de la palabra
M-c	Cambia a mayúscula el carácter en la posición del cursor y a minúscula hasta el fin de la palabra

Por ejemplo, veamos el efecto de cada uno de estos comandos sobre la palabra `EmAcS`, si el cursor está sobre la letra `E` (¡el efecto es distinto si está sobre cualquier otra letra!):

M-u : `EmAcS` ⟶ `EMACS`
M-l : `EmAcS` ⟶ `emacs`
M-c : `EmAcS` ⟶ `Emacs`

Transposición

Los siguientes comandos toman como referencia la posición actual del cursor. Por ejemplo, **C-t** intercambia el carácter justo antes del cursor con el carácter justo después.

C-t	Transpone dos caracteres
M-t	Transpone dos palabras
C-x C-t	Transpone dos líneas

Búsqueda y reemplazo

C-s	Búsqueda hacia el fin del texto
C-r	Búsqueda hacia el inicio del texto
M-%	Búsqueda y sustitución (pide confirmación cada vez)
M-&	Búsqueda y sustitución (sin confirmación)

Definición de regiones y reemplazo

Uno de los conceptos importantes en `emacs` es el de región. Para ello, necesitamos dos conceptos auxiliares: el *punto* y la *marca*. El punto es simplemente el cursor. Específicamente, es el punto donde *comienza* el cursor. Así, si el cursor se encuentra sobre la letra `c` en `emacs`, el punto está entre la `a` y la `c`. La marca, por su parte, es una señal que se coloca en algún punto del archivo con los comandos apropiados. La *región* es el espacio comprendido entre el punto y la marca.

Para colocar una marca basta ubicar el cursor en el lugar deseado, y teclear **C-**`Space` o **C-@**. Esto coloca la marca donde está el punto (en el ejemplo del párrafo anterior, quedaría entre las letras `a` y `c`. Una vez colocada la marca, podemos mover el cursor a cualquier otro lugar del archivo (hacia atrás o hacia adelante respecto a la marca). Esto define una cierta ubicación para el punto, y, por tanto, queda definida la región automáticamente.

La región es una porción del archivo que se puede manipular como un todo. Una región se puede borrar, copiar, pegar en otro punto del archivo o incluso en otro archivo; una región se puede imprimir, grabar como un archivo distinto; etc. Así, muchas operaciones importantes se pueden efectuar sobre un bloque del archivo.

Por ejemplo, si queremos duplicar una región, basta con definir la región deseada (poniendo la marca y el punto donde corresponda) y teclear **M-w**. Esto copia la región a un buffer temporal (llamado *kill buffer*). Luego movemos el cursor al lugar donde queremos insertar el texto duplicado, y hacemos **C-y**. Este comando toma el contenido del *kill buffer* y lo inserta en el archivo. El resultado final es que hemos duplicado una cierta porción del texto.

Si la intención era mover dicha porción, el procedimiento es el mismo, pero con el comando **C-w** en vez de **M-w**. **C-w** también copia la región a un *kill buffer*, pero borra el texto de la pantalla.

Resumiendo:

C-`Space` o **C-@**	Comienzo de región
M-w	Copia región
C-w	Corta región
C-y	Pega región

El concepto de *kill buffer* es mucho más poderoso que lo explicado recién. En realidad, muchos comandos, no sólo **M-w** y **C-w**, copian texto en un *kill buffer*. En general, cualquier comando que borre más de un carácter a la vez, lo hace. Por ejemplo, **C-k** borra una línea. Lo que hace no es sólo borrarla, sino además copiarla en un *kill buffer*. Lo mismo ocurre con los comandos que borran palabras completas (**M-d**, **M-**`Backspace`), y muchos otros. Lo interesante es que **C-y** funciona también en todos esos casos: **C-y** lo único que hace es tomar el último texto colocado en un *kill buffer* (resultado de la última operación que borró más de un carácter a la vez), y lo coloca en el archivo. Por lo tanto, no sólo podemos copiar o mover "regiones", sino también palabras o líneas. Más aún, el *kill buffer* no es borrado con el **C-y**, así que ese mismo texto puede ser duplicado muchas veces. Continuará disponible con **C-y** mientras no se ponga un nuevo texto en el *kill buffer*.

Además, `emacs` dispone no de uno sino de muchos *kill buffers*. Esto permite recuperar texto borrado hace mucho rato. En efecto, cada vez que se borra más de un carácter de una vez, se una un nuevo *kill buffer*. Por ejemplo, consideremos el texto:

```
La primera linea del texto,
la segunda linea,
y finalmente la tercera.
```

Si en este párrafo borramos la primera línea (con **C-k**), después borramos la primera palabra de la segunda (con **M-d**, por ejemplo), y luego la segunda palabra de la última, entonces habrá tres *kill buffers* ocupados:

buffer 1 : `La primera linea del texto,`
buffer 2 : `la`
buffer 3 : `finalmente`

Al colocar el cursor después del punto final, **C-y** toma el contenido del último *kill buffer* y lo coloca en el texto:

```
 segunda linea,
y  la tercera. finalmente
```

Si se teclea ahora **M-y**, el último texto recuperado, `finalmente`, es reemplazado por el penúltimo texto borrado, y que está en el *kill buffer* anterior:

```
 segunda linea,
y  la tercera. la
```

Además, la posición de los *kill buffers* se rota:

buffer 1 : `finalmente`
buffer 2 : `La primera linea del texto,`
buffer 3 : `la`

Sucesivas aplicaciones de **M-y** después de un **C-y** rotan sobre todos los *kill buffers* (que pueden ser muchos). El editor, así, conserva un conjunto de las últimas zonas borradas durante la edición, pudiendo recuperarse una antigua a pesar de haber seleccionado una nueva zona, o borrado una nueva palabra o línea. Toda la información en los *kill buffers* se pierde al salir de `emacs` (**C-c**).

Resumimos entonces los comandos para manejo de los *kill buffers*:

C-y	Copia el contenido del último *kill buffer* ocupado
M-y	Rota los *kill buffers* ocupados

Definición de macros

La clave de la configurabilidad de `emacs` está en la posibilidad de definir nuevos comandos que modifiquen su comportamiento o agreguen nuevas funciones de acuerdo a nuestras necesidades. Un modo de hacerlo es a través del archivo de configuración `$HOME/.emacs`, para lo

cual se sugiere leer la documentación disponible en la distribución instalada. Sin embargo, si sólo necesitamos un nuevo comando en la sesión de trabajo actual, un modo más simple es definir una *macro*, un conjunto de órdenes que son ejecutados como un solo comando. Los comandos relevantes son:

C-x (	Comienza la definición de una macro
C-x)	Termina la definición de una macro
C-x e	Ejecuta una macro definida

Todas las sucesiones de teclas y comandos dados entre **C-x (** y **C-x)** son recordados por `emacs`, y después pueden ser ejecutados de una vez con **C-x e**.

Como ejemplo, consideremos el siguiente texto, con los cinco primeros lugares del ránging ATP (sistema de entrada) al 26 de marzo de 2002:

```
1 hewitt, lleyton (Aus)
2 kuerten, gustavo (Bra)
3 ferrero, juan (Esp)
4 kafelnikov, yevgeny (Rus)
5 haas, tommy (Ger)
```

Supongamos que queremos: (a) poner los nombres y apellidos con mayúscula (como debería ser); (b) poner las siglas de países sólo en mayúsculas.

Para definir una macro, colocamos el cursor al comienzo de la primera línea, en el `1`, y damos **C-x (**. Ahora realizamos todos los comandos necesarios para hacer las tres tareas solicitadas para el primer jugador solamente: **M-f** (avanza una palabra, hasta el espacio antes de `hewitt`; **M-c M-c** (cambia a `Hewitt, Lleyton`); **M-u** (cambia a `AUS`); `Home` (vuelve el cursor al comienzo de la línea); ↓ (coloca el cursor al comienzo de la línea siguiente, en el `2`). Los dos últimos pasos son importantes, porque dejan el cursor en la posición correcta para ejecutar el comando nuevamente. Ahora terminamos la definición con **C-x)**. Listo. Si ahora ejecutamos la macro, con **C-x e**, veremos que la segunda línea queda modificada igual que la primera, y así podemos continuar hasta el final:

```
1 Hewitt, Lleyton (AUS)
2 Kuerten, Gustavo (BRA)
3 Ferrero, Juan (ESP)
4 Kafelnikov, Yevgeny (RUS)
5 Haas, Tommy (GER)
```

Comandos por nombre

Aparte de los ya comentados existen muchas otras órdenes que no tienen necesariamente una tecla asociada (*bindkey*) asociada. Para su ejecución debe de teclearse previamente:

`M-x`

y a continuación en la zona inferior de la pantalla se introduce el comando deseado. Empleando el `TAB` se puede completar dicho comando (igual que en `bash`).

De hecho, esto sirve para cualquier comando, incluso si tiene tecla asociada. Por ejemplo, ya sabemos **M-g** `n` va a la línea n del documento. Pero esto no es sino el comando `goto-line`, y se puede también ejecutar tecleando: `M-x goto-line n`.

Repetición

Todos los comandos de `emacs`, tanto los que tienen una tecla asociada como los que se ejecutan con nombre, se pueden ejecutar más de una vez, anteponiéndoles un argumento numérico con

`M-(number)`

Por ejemplo, si deseamos escribir 20 letras `e`, basta teclear `M-20 e`. Esto es particularmente útil con las macros definidos por el usuario. En el ejemplo anterior, con el ránking ATP, después de definir la macro quedamos en la línea 2, y en vez de ejecutar **C-x e** 4 veces, podemos teclear `M-4 C-x e`, con el mismo resultado, pero en mucho menos tiempo.

Para terminar la discusión de este editor, diremos que es conveniente conocer las secuencias de control básico de `emacs`:
`C-a, C-e, C-k, C-y, C-w, C-t, C-d`, etc.,
porque funcionan para editar la línea de comandos en el *shell*, como también en muchos programas de texto y en ventanas de diálogo de las aplicaciones *X Windows*. A su vez, los editores `jed`, `xjed`, `jove` también usan por defecto estas combinaciones.

1.9 El sistema X Windows.

El *X Windows system* es el sistema estándar de ventanas en las estaciones de trabajo. Es corriente que el sistema de ventanas sea arrancando automáticamente cuando la máquina parte. En caso contrario, la orden para arrancarlo es `startx`. En el sistema *X Windows* deben distinguirse dos conceptos:

- **server** : Es un programa que se encarga de escribir en el dispositivo de video y de capturar las entradas (por teclado, ratón, etc). Asimismo se encarga de mantener los recursos y preferencias de las aplicaciones. Sólo puede existir un server para cada pantalla.
- **client** : Es cualquier aplicación que se ejecute en el sistema *X Windows*. No hay límite (en principio) en el número de clientes que pueden estarse ejecutando simultáneamente. Los clientes pueden ser locales o remotos.

.
Window Manager (WM) Es un cliente con "privilegios especiales": Controla el comportamiento (forma, tamaño, ...) del resto de clientes. Existen varios, destacando:

- **fvwm** : *F* Virtual Window Manager*, el instalado por defecto.
- **olwm** : *Open Look Window Manager*, propio de las estaciones de trabajo SUN.
- **twm** : *Tab Window Manager*, suministrado con la distribución X11R* del MIT.

- **icewm** : *Ice Window Manager*, uno de los *window managers* gnome compatible.

El *look and feel* (o GUI) de *X Windows* es extremadamente configurable, y puede parecer que dos máquinas son muy distintas, pero esto se debe al `WM` que se esté usando y no a que las aplicaciones sean distintas.

Para configurar tu sesión es necesario saber qué programas estás usando y ver las páginas de manual. Los archivos pricipales son:

- `.xinitrc` ó `.xsession` archivo leído al arrancar *X Windows*. Aquí se pueden definir los programas que aparecen al inicio de tu sesión.
- `.fvwmrc` archivo de configuración del `fvwm`. Ver las páginas del manual de `fvwm`.
- `.olwmrc` archivo de configuración del `olwm`. Ver las páginas del manual de `olwm`.
- `.Xdefaults` Configuración general de las aplicaciones de X Windows. Aquí puedes definir los *resources* que encontrarás en los manuales de las aplicaciones de X.

En caso de que tengas que correr una aplicación de X que no esté disponible en la máquina que estás usando, eso no representa ningún problema. Las órdenes necesarias son (por ejemplo, para arrancar un `gnome-terminal` remoto):

```
userA@hostname1:~$ xhost +hostname2
hostname2 being added to access control list
user@hostname1:~$ ssh userB@hostname2
userB@hostname2's password:
userB@hostname2:~$ export DISPLAY=hostname1:0
userB@hostname2:~$ gnome-terminal &
```

Si todo está previamente configurado, es posible que no haga falta dar la *password*.

Cuando quieres salir, normalmente puedes encontrar un icono con la opción `Log out`, en un menú o panel de la pantalla.

1.10 Uso del ratón.

El ratón es un dispositivo esencial en el uso de programas X, sin embargo, la función que realiza en cada uno de ellos no está normalizada.

Comentaremos la pauta seguida por la mayoría de las aplicaciones, pero debe tenerse presente que es muy frecuente encontrar aplicaciones que no las respetan.[38]

- **Botón izquierdo** (LB): Seleccionar. Comienza el bloque de selección.
- **Botón central** (MB): Pegar. Copia la selección en la posición del cursor.
- **Botón derecho** (RB): Habitualmente ofrece un menu para partir aplicaciones.

[38] Las aplicaciones que son conscientes de un uso anormal y están realizadas por programadores inteligentes, muestran en pantalla la función de cada botón cuando son posibles varias alternativas.

Existen dos modos para determinar cuál es la **ventana activa**, aquella que recibe las entradas de teclado:

- *Focus Follows Mouse*: La ventana que contenga al ratón es la que es activa. No usado por defecto actualmente.
- *Click To Focus*: La ventana seleccionada es la activa. El modo que esté activo depende de la configuración del *Window Manager*.

1.11 Internet.

En esta sección denominaremos `unix1` a la máquina local (desde donde ejecutamos la orden) y `unix2` a la máquina remota (con la que interaccionamos). Ambos son los `hostnames` de las respectivas máquinas. Existen algunos conceptos que previamente debemos comentar:

- `IP-number`: es un conjunto de 4 números separados por puntos (p.e. 146.83.57.34) que se asocia a cada máquina. No puede haber dos máquinas conectadas en la misma red con el mismo número.
- `hostname`: es el nombre que tiene asociada la máquina (p.e. `macul`). A este nombre se le suelen añadir una serie de sufijos separados por puntos que constituye el denominado dominio (p.e. `macul.ciencias.uchile.cl`). Una máquina por tanto puede tener más de un nombre reconocido (se habla en este caso de alias). Se denomina resolución a la identificación entre un `hostname` y el `IP-number` correspondiente. La consulta se realiza inicialmente en el archivo `/etc/hosts`, donde normalmente se guardan las identificaciones de las máquinas más comunmente empleadas. En caso de que no se lograse se accede al servicio `DNS` (*Domain Name Service*), que permite la identificación (resolución) entre un `hostname` y un `IP-number`.
- `mail-address`: es el nombre que se emplea para enviar correo electrónico. Este nombre puede coincidir con el nombre de una máquina, pero se suele definir como un alias, con objeto de que la dirección no deba de cambiarse si la máquina se estropea o se cambia por otra.

1.11.1 Acceso a la red.

Existen muchos programas para la conexión de la red, los más usados son:

- `telnet unix2`, hace un login en la máquina `unix2`, debe ingresarse el usuario y su respectiva `passwd`. Adermás, permite especificar el puerto en conexión en la máquina remota.
- `ssh nombre@unix2`, muy similar a `telnet` pero se puede especificar el usuario, si no se especifica se usa el nombre de la cuenta local. Además, la `passwd` pasa encriptada a través de la red.

- `scp file1 usuario2@unix2:path/file`, copia el archivo `file1`, del usuario1, que se encuentra en el directorio local en la máquina `unix1` en la cuenta del `usuario2` en la máquina `unix2` en `$HOME/path/file`. Si no se especifica el nombre del usuario se usa el nombre de la cuenta local. Si se quiere copiar el archivo `file2` del `usuario3` en `unix2` en la cuenta actual de `unix1` el comando sería: `scp usuario3@unix2:file2 .`. Antes de realizar cualquiera de las copias el sistema preguntará por la `passwd` del usuario en cuestión en la máquina `unix2`. Nuevamente, la `passwd` pasa encriptada a través de la red.

- `talk usuario1@unix2`, Intenta hacer una conexión para hablar con el `usuario1` en la máquina `unix2`. Existen varias versiones de `talk` en los diferentes sistemas operativos, de forma que no siempre es posible establecer una comunicación entre máquinas con sistemas operativos diferentes.

- `ftp unix2`, (file transfer protocol) aplicación para copiar archivos entre máquinas de una red. `ftp` exige un nombre de cuenta y password para la máquina remota. Algunas de las opciones más empleadas (una vez establecida la conexión) son:

  - `bin`: Establece el modo de comunicación binario. Es decir, transfiere una imagen exacta del archivo.
  - `asc`: Establece el modo de comunicación `ascii`. Realiza las conversiones necesarias entre las dos máquinas en comunicación. Es el modo por defecto.
  - `cd`: Cambia directorio en la máquina remota.
  - `lcd`: Cambia directorio en la máquina local.
  - `ls`: Lista el directorio remoto.
  - `!ls`: Lista el directorio local.
  - `prompt` : No pide confirmación para transferencia múltiple de archivos.
  - `get rfile [lfile]`: transfiere el archivo `rfile` de la máquina remota a la máquina local denominándolo `lfile`. En caso de no suministrarse el segundo argumento supone igual nombre en ambas máquinas.
  - `put lfile [rfile]` : transfiere el archivo `lfile` de la máquina local a la máquina remota denominándolo `rfile`. En caso de no suministrarse el segundo argumento supone igual nombre en ambas máquinas. También puede usarse `send`.
  - `mget rfile` : igual que `get`, pero con más de un archivo (`rfile` puede contener carácteres comodines).
  - `mput lfile` : igual que `put`, pero con más de un archivo (`lfile` puede contener carácteres comodines).

  Existen versiones mejoras de `ftp` con muchas más posibilidades, por ejemplo, `ncftp`. También existen versiones gráficas de clientes `ftp` donde la elección de archivo, el sentido de la transferencia y el modo de esta se elige con el *mouse* (p.e. `wxftp`).

- `rlogin -l nombre unix2`, (*remote login*), hace un `login` a la máquina `unix2` como el usuario `nombre` por defecto, sin los argumentos `-l nombre` `rlogin` usa el nombre de la cuenta local. Normalmente `rlogin` pide el *password* de la cuenta remota, pero con el uso del archivo `.rhosts` o `/etc/hosts.equiv` esto no es siempre necesario.

- `rsh -l nombre unix2 orden`, (*remote shell*), ejecuta la orden orden en la máquina `unix2` como usuario `nombre`. Es necesario que pueda entrar en la máquina remota sin *password* para ejecutar una orden remota. Sin especificar orden actúa como `rlogin`.

1.11.2 El correo electrónico.

El correo electrónico (`e-mail`) es un servicio para el envío de mensajes entre usuarios, tanto de la misma máquina como de diferentes máquinas.

Direcciones de correo electrónico.

Para mandar un `e-mail` es necesario conocer la dirección del destinatario. Esta dirección consta de dos campos que se combinan intercalando entre ellos el @ (*at*): `user@domain`

- `user` : es la identificación del usuario (*i.e.* `login`) en la máquina remota.

- `domain` : es la máquina donde recibe correo el destinatario. A menudo, es frecuente que si una persona tiene acceso a un conjunto de máquinas, su dirección de correo no corresponda con una máquina sino que corresponda a un alias que se resolverá en un nombre específico de máquina en forma oculta para el que envía.

Si el usuario es local no es necesario colocar el campo `domain` (ni tampoco el @).

Nomenclatura.

Veamos algunos conceptos relacionados con el correo electrónico:

- **Subject** : Es una parte de un mensaje que piden los programas al comienzo y sirve como título para el mensaje.

- **Cc** (Carbon Copy) : Permite el envío de copias del mensaje que está siendo editado a terceras personas.

- **Reply** : Cuando se envía un mensaje en respuesta a otro se suele añadir el comienzo del *subject:* `Re:`, con objeto de orientar al destinatario sobre el tema que se responde. Es frecuente que se incluya el mensaje al que se responde para facilitar al destinatario la comprensión de la respuesta.

- **Forward** : Permite el envío de un mensaje (con modificaciones o sin ellas) a una tercera persona.

- **Forwarding Mail** : Permite a un usuario que disponga de cuentas en varias máquinas no relacionadas, de concentrar su correo en una cuenta única[39]. Para ello basta con tener un archivo `$HOME/.forward` que contenga la dirección donde desea centralizar su correo.
- **Mail group** : Un grupo de correo es un conjunto de usuarios que reciben el correo dirigido a su grupo. Existen órdenes para responder a un determinado correo recibido por esa vía de forma que el resto del grupo sepa lo que ha respondido un miembro del mismo.
- **In-Box** : Es el archivo donde se almacena el correo que todavía no ha sido leído por el usuario. Suele estar localizado en `/var/spool/mail/user`.
- **Mailer-Daemon** : Cuando existe un problema en la transmisión de un mensaje se recibe un mensaje proviniente del *Mailer-Daemon* que indica el problema que se ha presentado.

Aplicación mail.

Es posiblemente la aplicación más simple. Para la lectura de mail teclear simplemente: `mail` y a continuación aparece un índice con los diferentes mensajes recibidos. Cada mensaje tiene una línea de identificación con número. Para leer un mensaje basta teclear su número y a continuación `return`. Para enviar un mensaje: `mail (address)` se pregunta por el `Subject:` y a continuación se introduce el mensaje. Para acabar se teclea sólo un punto en una línea o bien `Ctr-D`. Por último, se pregunta por `Cc:`. Es posible personalizar el funcionamiento mediante el archivo `$HOME/.mailrc`. Para enviar un archivo de texto a través del correo se suele emplear la redirección de entrada: `mail (address) < file`.

1.11.3 Ftp anonymous.

Existen servidores que permiten el acceso por `ftp` a usuarios que no disponen de cuenta en dichas máquinas. Para ello se emplea como `login` de entrada el usuario `anonymous` y como `passwd` la dirección de *e-mail* personal. Existen servidores que no aceptan conexiones desde máquinas que no están declaradas correctamente en el servicio de nombre (*dns*), así como algunas que no permiten la entrada a usuarios que no se identifican correctamente. Dada la sobrecarga que existe, muchos de los servidores tienen limitado el número de usuarios que pueden acceder simultáneamente.

1.11.4 WWW.

WWW son las siglas de *World-Wide Web*. Este servicio permite el acceso a información entrelazada (dispone de un texto donde un término puede conducir a otro texto): *hyperlinks*. Los archivos están realizados en un lenguaje denominado *html*. Para acceder a este servicio es necesario disponer de un lector de dicho lenguaje conocido como *browser* o navegador. Destacan actualmente: Netscape, Mozilla, Opera y el simple pero muy rápido Lynx.

[39] Este comando debe usarse con conocimiento pues en caso contrario podría provocar un *loop* indefinido y no recibir nunca correo.

1.12 Impresión.

Cuando se quiere obtener una copia impresa de un archivo se emplea el comando `lpr`.

`lpr file` - Envía el archivo `file` a la cola de impresión por defecto. Si la cola está activada, la impresora lista y ningún trabajo por encima del enviado, nuestro trabajo será procesado de forma automática.

A menudo existen varias posibles impresoras a las que poder enviar los trabajos. Para seleccionar una impresora en concreto (en vez de la por defecto) se emplea el modificador: `lpr -Pimpresora`, siendo `impresora` el nombre lógico asignado a esta otra impresora. Para recibir una lista de las posibles impresoras de un sistema así como su estado se puede emplear el comando `/usr/sbin/lpc status`. La lista de impresoras y su configuración también está disponible en el archivo `/etc/printcap`.

Otras órdenes para la manipulación de la cola de impresión son:

- `lpq [-Pprinter]`, permite examinar el estado de una determinada cola (para ver la cantidad de trabajos sin procesar de ésta, por ejemplo).
- `lprm [-Pprinter] jobnumber`, permite eliminar un trabajo de la cola de impresión.

Uno de los lenguajes de impresión gráfica más extendidos en la actualidad es *PostScript*. La extensión de los archivos *PostScript* empleada es `.ps`. Un archivo *PostScript* puede ser visualizado e imprimirso mediante los programas: `gv, gnome-gv` o `ghostview`. Por ello muchas de las impresoras actuales sólo admiten la impresión en dicho formato.

En caso de desear imprimir un archivo `ascii` deberá previamente realizarse la conversión a *PostScript* empleando la orden `a2ps`: `a2ps file.txt` Esta orden envía a la impresora el archivo `ascii file.txt` formateado a 2 páginas por hoja. Otro programa que permite convertir un archivo `ascii` en postscript es `enscript`.

Otro tipo de archivos ampliamente difundido y que habitualmente se necesita imprimir es el conocido como *Portable Document Format*. Este tipo de archivo posee una extensión `.pdf` y pueden ser visualizados e impresos usando aplicaciones tales como: `acroread, gv` o `xpdf`.

1.13 Compresión.

A menudo necesitamos comprimir un archivo para disminuir su tamaño, o bien crear un respaldo (*backup*) de una determinada estructura de directorios. Se comentan a continuación una serie de comandos que permiten ejecutar dichas acciones:

El compresor `compress` esta relativamente fuera de uso, pero es la estádar de UNIX.

- `compress file` : comprime el archivo, creando el archivo `file.Z`, destruye el archivo original.
- `uncompress file.Z` : descomprime el archivo, creando el archivo `file`, destruye el archivo original.

- `zcat file.Z` : muestra por el `stdout` el contenido descomprimido del archivo (sin destruir el original).

Otra alternativa de compresor mucho más usada es `gzip` el compresor de `GNU` que posee una mayor razón de compresión que `compress`. Veamos los comandos:

- `gzip file` : comprime el archivo, creando el archivo `file.gz`, destruye el archivo original.

- `gunzip file.gz` : descomprime el archivo, creando el archivo `file`, destruye el archivo original.

- `zless file.gz` : muestra por el `stdout` el contenido descomprimido del archivo paginado por `less`.

La extensión empleada en los archivos comprimidos con `gzip` suele ser `.gz` pero a veces se usa `.gzip`. Adicionalmente el programa `gunzip` también puede descomprimir archivos creados con `compress`.

La opción con mayor tasa de compresión que `gzip` es `bzip2` y su descompresor `bunzip2`. La extensión usada en este caso es `.bz2`. El *kernel* de Linux se distribuye en formato `bzip2`.

Existe también versiones de los compresores compatibles con otros sistemas operativos: `zip`, `unzip`, `unarj`, `lha` y `zoo`.

En caso que se desee crear un archivo comprimido con una estructura de directorios debe ejecutarse la orden:
`tar cvzf nombre.tgz directorio`
o bien
`tar cvIf nombre.tgz directorio`
En el primer caso comprime con `gzip` y en el segundo con `bzip2`. Para descomprimir y reestablecer la estructura de directorio almacenada se usan los comandos:
`tar xvzf nombre.tgz directorio`
si se realizó la compresión con `gzip` o bien
`tar xvIf nombre.tgz directorio`
si se realizó la compresión con `bzip2`.

1.14 Compilación y debugging.

1.14.1 Compiladores.

El comando para usar el compilador de lenguaje C es `gcc`, para usar el compilador de C++ es `g++` y para usar el compilador de fortran 77 es `g77`. Centrémosnos en el compilador de C++, los demás funcionan en forma muy similar. Su uso más elemental es:
`g++ filename.cc`
Esto compila el archivo `filename.cc` y crea un archivo ejecutable que se denomina `a.out` por omisión. Existen diversas opciones para el compilador, sólo comentaremos una pocas.

- `-c` realiza sólo la compilación pero no el *link*:
  `g++ -c filename.cc`
  genera el archivo `filename.o` que es código objeto.

- `-o exename` define el nombre del ejecutable creado, en lugar del por defecto `a.out`.
  `g++ -o outputfile filename.cc`

- `-lxxx` incluye la librería `/usr/lib/libxxx.a` en la compilación.
  `g++ filename.cc -lm`
  En este caso se compila con la librería matemática verb+libm.a+.

- `-g` permite el uso de un debugger posteriormente.

- `-On` optimización de grado `n` que puede tomar valores de 1 (por defecto) a 3. El objetivo inicial del compilador es reducir el tiempo de la compilación. Con `-On`, el compilador trata de reducir el tamaño del ejecutable y el tiempo de ejecución, con `n` se aumenta el grado de optimización.

- `-Wall` Notifica todos los posibles *warnings* en el código que está siendo compilado.

El compilador `gcc` (*the GNU C compiler*) es compatible ANSI.

1.15 make & Makefile.

Frecuentemente los programas están compuestos por diferentes subprogramas que se hayan contenidos en diferentes archivos. La orden de compilación necesaria puede ser muy engorrosa, y a menudo no es necesario volver a compilar todos los archivos, sino sólo aquellos que hayan sido modificados. UNIX dispone de una orden denominada `make` que evita los problemas antes mencionados y permite el mantenimiento de una biblioteca personal de programas. Este comando analiza qué archivos fuentes han sido modificados después de la última compilación y evita recompilaciones innecesarias.

En su uso más simple sólo es necesario suministrar una lista de dependencias y/o instrucciones a la orden `make` en un archivo denominado `Makefile`. Una dependencia es la relación entre dos archivos de forma que un archivo se considera actualizado siempre que el otro tenga una fecha de modificación inferior a éste. Por ejemplo, si el archivo `file.cc` incluye el archivo `file.h`, no se puede considerar actualizado el archivo `file.o` si el archivo `file.cc` o el archivo `file.h` ha sido modificado después de la última compilación. Se dice que el archivo `file.o` depende de `file.cc` y el archivo `file.cc` depende de archivo `file.h`. El Makefile se puede crear con un editor de texto y tiene el siguiente aspecto para establecer una dependencia:

```
# Esto es un ejemplo de Makefile.
# Se pueden poner comentarios tras un caracter hash (#).

FILE1: DEP1 DEP2
        comandos para generar FILE1
ETIQUETA1: FILE2
```

```
FILE2: DEP3 DEP4
        comandos para generar FILE2
ETIQUETA2:
        comandos
```

Se comienza con un destino, seguido de dos puntos (`:`) y los prerrequisitos o dependencias necesarios. También puede ponerse una etiqueta y como dependencia un destino, o bien una etiqueta y un o más comandos. Si existen muchos prerrequisitos, se puede finalizar la línea con un backslash (\) y continuar en la siguiente línea.

En la(s) línea(s) siguiente(s) se escriben uno o más comandos. Cada línea se considera como un comando independiente. Si se desea utilizar múltiples líneas para un comando, se debería poner un backslash (\) al final de cada línea del comando. El comando `make` conectará las líneas como si hubieran sido escritas en una única línea. En esta situación, se deben separar los comandos con un punto y coma (`;`) para prevenir errores en la ejecución de el *shell*. **Los comandos deben ser indentados con un tabulador, no con 8 espacios**

Make lee el `Makefile` y determina para cada archivo destino (empezando por el primero) si los comandos deben ser ejecutados. Cada destino, junto con los prerrequisitos o dependencias, es denominado una regla.

Si `make` se ejecuta sin argumentos, sólo se ejecutará el primer destino. Veamos un ejemplo:

```
file.o: file.cc file.h
        g++ -c file.cc
```

En este caso se comprueba las fechas de las última modificaciones de los archivo `file.cc` y `file.h`, si estas fecha son más recientes que la del archivo `file.o` se procede a la compilación.

El comando `make` se puede suministrar con un argumento, que indica la etiqueta situada a la izquierda de los dos puntos. Así en el ejemplo anterior podría invocarse `make file.o`..

Gracias a las variables, un `Makefile` se puede simplificar significativamente. Las variables se definen de la siguiente manera:
`VARIABLE1=valor1`
`VARIABLE2=valor2`
Una variable puede ser utilizada en el resto del `Makefile` refiriendonos a ella con la expresión `$(VARIABLE)`. Por defecto, `make` sabe las órdenes y dependencias (reglas implícitas) para compilar un archivo `*.cc` y producir un archivo `*.o`, entonces basta especificar solamente los dependecias que make no puede deducir a partir de los nombres de los archivos, por ejemplo:

```
OUTPUTFILE = prog
OBJS = prog.o misc.o aux.o
INCLUDESMISC = misc.h aux.h
INCLUDESFILE = foo.h $(INCLUDESMISC)
LIBS = -lmylib -lg++ -lm

prog.o: $(INCLUDESFILE)
misc.o: $(INCLUDESMISC)
aux.o: aux.h

$(OUTPUTFILE): $(OBJS)
```

```
        gcc $(OBJS) -o $(OUTPUTFILE) $(LIBS)
```

Las reglas patrones son reglas en las cuales se especifican multiples destinos y construye el nombre de las dependencias para cada blanco basada en el nombre del blanco. La sintaxis de una regla patrón:

```
destinos ... : destino patron:dependencias patrones
        comandos
```

La lista de destinos especifica aquellos sobre los que se aplicará la regla. El destino patrón y las dependencias patrones dicen cómo calcular las dependencias para cada destino. Veamos un ejemplo:

```
objects = foo.o \
          bar.o

all: $(objects)

$(objects): %.o: %.cc
        $(CXX) -c $(CFLAGS) $< -o $@
```

Cada uno de los destinos (`foo.o bar.o`) es comparado con el destino patrón `%.o` para extraer parte de su nombre. La parte que se extrae es conocida como el tronco o *stem*, `foo` y `bar` en este caso. A partir del tronco y la regla patrón de las dependencias `%.cc make` construye los nombres completos de ellas (`foo.cc bar.cc`). Además, en el comando del ejemplo anterior aparecen un tipo de variables especiales conocidas como automáticas. La variable `$<` mantiene el nombre de la dependencia actual y la variable `$@` mantiene el nombre del destino actual. Finalmente un ejemplo completo:

```
# Makefile for C++ program EMBAT
# Embedded Atoms Molecular Dynamics
#
# You should do a "make" to compile
############################################################

#Compiler name
CXX = g++
#Linker name
LCC = g++

#Compiler flags you want to use
MYFLAGS = -Wall
OPFLAGS = -O4

CPPFLAGS = $(MYFLAGS) $(OPFLAGS)

#Library flags you want to use
```

```
MATHLIBS = -lm
CLASSLIBS= -lg++
LIBFLAGS = $(MATHLIBS) $(CLASSLIBS)
#############################################################

OBJFILES = embat.o lee2.o crea100.o errores.o\
           archivos.o azar.o lista.o rij.o\
           escribir2.o velver.o cbc.o force.o\
           fMetMet.o fDf.o relaja.o funciones.o\
           initvel.o leerfns.o spline.o

embat : $(OBJFILES)
        $(LCC) -o embat $(OBJFILES) $(LIBFLAGS)

$(OBJFILES): %.o:%.cc
        $(CXX) -c $(CPPFLAGS) $< -o $@

#############################################################
clean:
        rm embat  $(OBJFILES)
#############################################################
```

Capítulo 2

Una breve introducción a C++.

versión final 3.5-19 de agosto de 2002

En este capítulo se intentará dar los elementos básicos del lenguaje de programación C++. No se pretende más que satisfacer las mínimas necesidades del curso, sirviendo como un ayuda de memoria de los tópicos abordados, para futura referencia. Se debe consignar que no se consideran todas las posibilidades del lenguaje y las explicaciones están reducidas al mínimo.

2.1 Estructura básica de un programa en C++.

2.1.1 El programa más simple.

El primer ejemplo de todo manual es el que permite escribir "Hola" en la pantalla.

```
//
// Los comentarios comienzan con //
//
#include <iostream.h>
main()
{
    cout << "Hola." << endl;
}
```

Las tres primeras líneas corresponden a comentarios, todo lo que está a la derecha de los caracteres `//` son comentarios y no serán considerados en la compilación. En la línea siguiente se incluye un archivo de cabecera, o *header*, con la instrucción de preprocesador `#include`. El nombre del archivo se puede escribir como `<nombre.h>` o bien `"nombre.h"`. En el primer caso el archivo `nombre.h` será buscado en el *path* por defecto para los `include`, típicamente `/usr/include` o `/usr/include/g++-3/` en el caso de *headers* propios de C++; en el segundo caso la búsqueda se hace en el directorio local. También podríamos incluir un *path* completo cuando se ocupan las comillas. En nuestro ejemplo se incluye el archivo `iostream.h`, en el cual se hacen las definiciones adecuadas para el manejo de la entrada y salida en C++. Este archivo es necesario para enviar luego un mensaje a pantalla.

La función `main` es donde comienza a ejecutarse el programa; siempre debe haber una función `main` en nuestro programa. Los paréntesis vacíos `()` indican que el `main` no tiene

argumentos de entrada (más adelante se verá que puede tenerlos). Lo que está encerrado entre llaves `{}` corresponde al cuerpo de la función `main`. Cada una de las líneas termina con el carácter `;`. El identificador predefinido `cout` representa la salida a pantalla. El operador `<<` permite que lo que está a su derecha se le dé salida por el dispositivo que está a su izquierda, en este caso `cout`. Si se quiere enviar más de un objeto al dispositivo que está al inicio de la línea agregamos otro operador `<<` y en este caso lo que está a la derecha del operador se agregará a lo que está a la izquierda y todo junto será enviado al dispositivo. En nuestro caso se ha enviado `endl`, un objeto predefinido en el archivo `iostream.h` que corresponde a un cambio de línea, el cual será agregado al final del mensaje.

Si escribimos nuestro primer programa en el editor `xemacs` con el nombre de `primero.cc` las instruciones para editarlo, compilarlo y correrlo serán:

```
jrogan@pucon:~/tmp$ xemacs primero.cc
jrogan@pucon:~/tmp$ g++ -o primero primero.cc
jrogan@pucon:~/tmp$ ./primero
Hola.
jrogan@pucon:~/tmp$
```

Luego de la compilación, un archivo ejecutable llamado `primero` es creado en el directorio actual. Si el directorio actual no está en el `PATH`, nuestro programa debe ser ejecutado anteponiendo `./`. Si está en el `PATH`, para ejecutarlo basta escribir `primero`. (Para agregar el directorio local al `PATH` basta editar el archivo `~/.bashrc` agregarle una línea como `PATH="${PATH}:."` y ejecutar en la línea de comando `source ~/.bashrc` para que los cambios tengan efecto.)

2.1.2 Definición de funciones.

Las funciones en C++ son muy importantes, pues permiten aislar parte del código en una entidad separada. Esto es un primer paso a la *modularización* de nuestro programa, es decir, a la posibilidad de escribirlo en partes que puedan ser editadas de modo lo más independiente posible. Ello facilita enormemente la creación de código complicado, pues simplifica su modificación y la localización de errores. Nos encontraremos frecuentemente con este concepto.

Aprovecharemos de introducir las funciones modificando el primer programa de manera que se delegue la impresión del mensaje anterior a una función independiente:

```
//
// Segunda version incluye funcion adicional
//
#include <iostream.h>

void PrintHola()
{
    cout << "Hola." << endl;
}
```

```
main()
{
    PrintHola();
}
```

La función debe estar definida antes de que sea ocupada, por eso va primero en el código fuente. Como ya se dijo antes la ejecución del programa comienza en la función `main` a pesar de que no está primera en el código fuente. Los paréntesis vacíos indican que la función `PrintHola` no tiene argumentos y la palabra delante del nombre de la función indica el tipo de dato que devuelve. En nuestro caso la palabra `void` indica que no devuelve nada a la función `main`.

Una alternativa al código anterior es la siguiente:

```
#include <iostream.h>

void PrintHola();

main()
{
    PrintHola();
}

void PrintHola()
{
    cout << "Hola." << endl;
}
```

En esta versión se ha separado la *declaración* de la función de su *implementación*. En la declaración se establece el nombre de la función, los argumentos que recibe, y el tipo de variable que entrega como resultado. En la implementación se da explícitamente el código que corresponde a la función. Habíamos dicho que una función debe estar definida antes que sea ocupada. En verdad, basta conque la función esté declarada. La implementación puede ir después (como en el ejemplo anterior), o incluso en un archivo distinto, como veremos más adelante. La separación de declaración e implementación es otro paso hacia la modularización de nuestro programa.

2.1.3 Nombres de variables.

Nuestros datos en los programas serán almacenados en objetos llamados variables. Para referirnos a ellas usamos un nombre que debe estar de acuerdo a las siguientes reglas:

- Deben comenzar con una letra (mayúsculas y minúsculas son distintas).
- Pueden contener números.
- Pueden contener el símbolo `_` (underscore).
- Longitud arbitraria.

– No pueden corresponder a una de las palabras reservadas de C++[1]:

```
asm       delete  if         return    try
auto      do      inline     short     typedef
break     double  int        signed    union
case      else    long       sizeof    unsigned
catch     enum    new        static    virtual
char      extern  operator   struct    void
class     float   private    switch    volatile
const     for     protected  template  while
continue  friend  public     this
default   goto    register   throw
```

2.1.4 Tipos de variables.

Todas las variables a usar deben ser *declaradas* de acuerdo a su tipo. Por ejemplo, si usamos una variable `i` que sea un número entero, debemos, antes de usarla, declararla, y sólo entonces podemos asignarle un valor:

```
int i;
i=10;
```

Esta necesidad de declarar cada variable a usar se relaciona con la característica de C++ de ser fuertemente "tipeado"[2]. Algunos de los errores más habituales en programación se deben al intento de asignar a variables valores que no corresponden a sus tipos originales. Si bien esto puede no ser muy grave en ciertos contextos, a medida que los programas se vuelven más complejos puede convertirse en un verdadero problema. El compilador de C++ es capaz de detectar los usos indebidos de las variables pues conoce sus tipos, y de este modo nuestro código se vuelve más seguro.

Es posible reunir las acciones de declaración e inicialización en una misma línea:

```
int i=10;
```

o declarar más de una variable del mismo tipo simultáneamente, e inicializar algunas en la misma línea:

```
int r1, r2, r3 = 10;
```

A veces se requiere que una variable no varíe una vez que se le asigna un valor. Por ejemplo, podríamos necesitar definir el valor de $\pi = 3.14159...$, y naturalmente no nos gustaría que, por un descuido, a esa variable se le asignara otro valor en alguna parte del programa. Para asegurarnos de que ello no ocurra, basta agregar el modificador `const` a la variable:

```
const float pi = 3.14159;
```

[1] A esta tabla hay que agregar algunas palabras adicionales, presentes en versiones más recientes de C++, como `namespace` y `using`

[2] Una traducción libre del término inglés *strongly typed*.

Para números reales se puede usar la notación exponencial. Por ejemplo, `1.5e-3` representa el número $1.5 \cdot 10^{-3}$.

Una variable puede ser declarada sólo una vez, pero naturalmente se le pueden asignar valores en un número arbitrario de ocasiones.

Los tipos de variables disponibles son[3]:

`int`	Enteros entre $-2^{15} = -32768$ y $2^{15} - 1 = 32767$ o entre $-2^{31} = -2147483648$ y $2^{31} - 1 = 2147483647$
`short int`	
`short`	Enteros entre -2^{15} y $2^{15} - 1$.
`long int`	
`long`	Enteros entre -2^{31} y $2^{31} - 1$.
`unsigned int`	Enteros entre 0 y $2^{16} - 1$ o entre 0 y $2^{32} - 1$.
`unsigned short`	Enteros entre 0 y $2^{16} - 1$.
`unsigned long`	Enteros entre 0 y $2^{32} - 1$.
`char`	Caracteres.
`float`	Reales en los intervalos $[-1.7 \cdot 10^{38}, -0.29 \cdot 10^{-38}]$, $[0.29 \cdot 10^{-38}, 1.7 \cdot 10^{38}]$ (Precisión de unos 7 dígitos decimales.)
`double`	Reales en los mismos intervalos que `float`, pero con precisión de 16 decimales, o en los intervalos $[-0.9 \cdot 10^{308}, -0.86 \cdot 10^{-308}]$ y $[0.86 \cdot 10^{-308}, 0.9 \cdot 10^{308}]$, con precisión de 15 decimales.

Las variables tipo `char` alojan caracteres, debiendo inicializarse en la forma:

```
char c = 'a';
```

Además de las letras mayúsculas y minúsculas, y símbolos como `&`, `(`, `:`, etc., hay una serie de caracteres especiales (*escape codes*) que es posible asignar a una variable `char`. Ellos son:

newline	`\n`
horizontal tab	`\t`
vertical tab	`\v`
backspace	`\b`
carriage return	`\r`
form feed	`\f`
alert (bell)	`\a`
backslash	`\\`
single quote	`\'`
double quote	`\"`

Por ejemplo, la línea:

```
cout << "Primera columna\t Segunda columna\n
```

[3]Los valores de los rangos indicados son simplemente representativos y dependen de la máquina utilizada. Además, estos valores no corresponden exactamente a las versiones más recientes de C++.

```
          Segunda linea" << endl;
```

corresponde al output

```
Primera columna         Segunda columna
Segunda linea
```

2.1.5 Ingreso de datos desde el teclado.

El *header* `iostream.h` define un objeto especial llamado `cin` que está asociado al teclado o `stdin`. Con el operador `>>` asignamos la entrada en el dispositivo de la izquierda a la variable de la derecha; una segunda entrada requiere de otro operador `>>` y de otra variable. En el siguiente ejemplo veremos una declaración simultánea de dos variables del mismo tipo `i` y `j`, un mensaje a pantalla con las instrucciones a seguir, el ingreso de dos variables desde el teclado y luego su escritura en la pantalla.

```
#include <iostream.h>
main()
{
    int i, j ;
    cout << "Ingrese dos numeros enteros: " ;
    cin >> i >> j ;
    cout << "Los dos numeros ingresados fueron: " << i <<" "<< j << endl ;
}
```

2.1.6 Operadores aritméticos.

Existen operadores binarios (i.e., que actúan sobre dos variables, una a cada lado del operador) para la suma, la resta, la multiplicación y la división:

```
+   -   *   /
```

2.1.7 Operadores relacionales.

Los símbolos para los operadores relacionales de igualdad, desigualdad, menor, menor o igual, mayor y mayor o igual son:

```
==    !=    <    <=    >    >=
```

Para las relaciones lógicas `AND`, `OR` y `NOT`:

```
&&    ||    !
```

2.1.8 Asignaciones.

a) Asignación simple. Podemos asignar a una variable un valor explícito, o el valor de otra variable:

```
i = 1;
j = k;
```

Una práctica habitual en programación es iterar porciones del código. La iteración puede estar determinada por una variable cuyo valor aumenta (disminuye) cada vez, hasta alcanzar cierto valor máximo (mínimo), momento en el cual la iteración se detiene. Para que una variable `x` aumente su valor en `2`, por ejemplo, basta escribir:

```
x = x + 2;
```

Si `x` fuera una variable matemática normal, esta expresión no tendría sentido. Esta expresión es posible porque el compilador interpreta a `x` de modo distinto a cada lado del signo igual: a la derecha del signo igual se usa el valor contenido en la variable `x` (por ejemplo, 10); a la izquierda del signo igual se usa la dirección de memoria en la cual está alojada la variable `x`. De este modo, la asignación anterior tiene el efecto de colocar en la dirección de memoria que contiene a `x`, el valor que tiene `x` más `2`. En general, todas las variables tienen un *rvalue* y un *lvalue*: el primero es el valor usado a la derecha (*right*) del signo igual en una asignación, y el segundo es el valor usado a la izquierda (*left*) —es decir, su dirección de memoria.

b) Asignación compuesta.

La expresión `x = x + 2` se puede reemplazar por `x += 2`.

Existen los operadores `+=` `-=` `*=` `/=`

c) Operadores de incremento y decremento.

La expresión `x = x + 1` se puede reescribir `x += 1` o bien `x++`.

Análogamente, existe el operador `--`. Ambos operadores unarios, `++` y `--` pueden ocuparse como prefijos o sufijos sobre una variable y su acción difiere en ambos casos. Como prefijo la operación de incremento o decremento se aplica antes de que el valor de la variable sea usado en la evaluación de la expresión. Como sufijo el valor de la variable es usado en la evaluación de la expresión antes que la operación de incremento o decremento. Por ejemplo, supongamos que inicialmente $x = 3$. Entonces la instrucción `y=x++` hace que $y = 3$, $x = 4$; por su parte, `y=++x` hace que $y = 4$, $x = 4$.

Con estas consideraciones, deberíamos poder convencernos de que la salida del siguiente programa es `3 2 2-1 1 1` :

```
// Ejemplo de operadores unarios ++ y --.
#include <iostream.h>
void main()
{
```

```
  int y ; int x = (y = 1) ;
  int w = ++x + y++;
  cout << w <<" " << x << " " << y << "-" ;
  w = x-- - --y;
  cout <<  w << " " << x << " " << y << endl ;
}
```

Los operadores para asignación compuesta, y los de incremento y decremento, no son sólo abreviaciones. En realidad hay que preferirlas porque implican optimizaciones en el ejecutable resultante.

2.1.9 Conversión de tipos.

Una consecuencia de que C++ sea fuertemente "tipeado" es que no se pueden hacer operaciones binarias con objetos de tipos distintos. En la siguiente expresión,

```
int i = 3;
float x = 43.8;
cout << "Suma = " << x + i << endl;
```

el computador debe sumar dos variables de tipos distintos, y en principio la operación es imposible. La estrategia para resolver este problema es convertir ambas variables a un tipo común antes de efectuar la suma (en inglés, decimos que hacemos un *cast* de un tipo a otro. Existen dos modos de proceder:

a) Conversión explícita.

   Si `i` es un `int`, por ejemplo, entonces `float(i)` la convierte en `float`. Así, el programa anterior se puede reescribir:

   ```
   int i = 3;
   float x = 43.8;
   cout << "Suma = " << x + float(i) << endl;
   ```

   Ahora la suma es claramente entre dos variables `float`, y se puede realizar. Sin embargo, esto es bastante tedioso, por cuanto el programador debe realizar el trabajo de conversión personalmente cada vez que en su código se desee sumar un real con un número entero.

b) Conversión implícita.

   En este caso, el compilador realiza las conversiones de modo automático, prefiriendo siempre la conversión desde un tipo de variable de menor precisión a uno de mayor precisión (de `int` a `double`, de `short` a `int`, etc.). Así, a pesar de lo que dijimos, el código anterior habría funcionado en su forma original. Evidentemente esto es muy cómodo, porque no necesitamos hacer una conversión explícita cada vez que sumamos un entero con un real. Sin embargo, debemos estar conscientes de que esta comodidad sólo es posible porque ocurren varias cosas: primero, el compilador detecta el intento de

operar sobre dos variables que no son del mismo tipo; segundo, el compilador detecta, en sus reglas internas, la posibilidad de cambiar uno de los tipos (`int` en este caso) al otro (`float`); tercero, el compilador realiza la conversión, y finalmente la operación se puede llevar a cabo. Entender este proceso nos permitirá aprovechar las posibilidades de la convesión implícita de tipos cuando nuestro código involucre tipos de variables más complicados, y entender varios mensajes de error del compilador.

Es interesante notar cómo las conversiones implícitas de tipos pueden tener consecuencias insospechadas. Consideremos las tres expresiones:

i) `x = (1/2) * (x + a/x) ;`

ii) `x = (0.5) * (x + a/x) ;`

iii) `x = (x + a/x)/2 ;`

Si inicialmente `x=0.5` y `a=0.5`, por ejemplo, i) entrega el valor `x=0`, mientras ii) y iii) entregan el valor `x=1.5`. Lo que ocurre es que `1` y `2` son enteros, de modo que `1/2 = 0`. De acuerdo a lo que dijimos, uno esperaría que en i), como conviven números reales con enteros, los números enteros fueran convertidos a reales y, por tanto, la expresión tuviera el resultado esperado, 1.5. El problema es la *prioridad* de las operaciones. No todas las operaciones tienen igual prioridad (las multiplicaciones y divisiones se realizan antes que las sumas y restas, por ejemplo), y esto permite al compilador decidir cuál operación efectuar primero. Cuando se encuentra con operaciones de igual prioridad (dos multiplicaciones, por ejemplo), se procede a efectuarlas de izquierda a derecha.

Pues bien, en i), la primera operación es `1/2`, una división entre enteros, i.e. cero. En ii) no hay problema, porque todas son operaciones entre reales. Y en iii) la primera operación es el paréntesis, que es una operación entre reales. Al dividir por 2 éste es convertido a real antes de calcular el resultado.

i) aún podría utilizarse, cambiando el prefactor del paréntesis a `1.0/2.0`, una práctica que sería conveniente adoptar como standard cuando queremos utilizar enteros dentro de expresiones reales, para evitar errores que pueden llegar a ser muy difíciles de detectar.

2.2 Control de flujo.

2.2.1 `if`, `if... else`, `if... else if.`

Las construcciones siguientes permiten controlar el flujo del programa en base a si una expresión lógica es verdadera o falsa.

a) En el caso de la sentencia `if` se evaluará la expresión `(a==b)`, si ella es cierta ejecutará la o las líneas entre los paréntesis de llave y si la expresión es falsa el programa se salta esa parte del código.

```
if (a==b) {
    cout << "a es igual a b" << endl;
}
```

En este y en muchos de los ejemplos que siguen, los paréntesis cursivos son opcionales. Ellos indican simplemente un grupo de instrucciones que debe ser tratado como una sola instrucción. En el ejemplo anterior, los paréntesis cursivos después del `if` (o después de un `while`, `for`, etc. más adelante) indican el conjunto de instrucciones que deben o no ejecutarse dependiendo de si cierta proposición es verdadera o falsa. Si ese conjunto de instrucciones es una sola, se pueden omitir los paréntesis:

```
if (a==b) cout << "a es igual a b" << endl;
```

b) En el caso `if... else` hay dos acciones mutuamente excluyentes. La sentencia `if (c!=b)` evaluará la expresión `(c!=b)`. Si ella es cierta ejecutará la o las líneas entre los paréntesis de llave que le siguen, saltándose la o las líneas entre los paréntesis de llave que siguen a la palabra clave `else`. Si la expresión es falsa el programa se salta la primera parte del código y sólo ejecuta la o las líneas entre los paréntesis de llave que siguen a `else`.

```
if (c!=d) {
    cout << "c es distinto de d" << endl;
}
else {
    cout << "c es igual a d" << endl;
}
```

c) En el último caso se evaluará la expresión que acompaña al `if` y si ella es cierta se ejecutará la o las líneas entre los paréntesis de llave que le siguen, saltándose todo el resto de las líneas entre los paréntesis de llave que siguen a las palabras claves `else if` y `else`. Si la primera expresión es falsa el programa se salta la primera parte del código y evalúa la expresión que acompaña al primer `else if` y si ella es cierta ejecutará la o las líneas entre los paréntesis de llave que le siguen, saltándose todo el resto de las líneas entre los paréntesis que siguen a otros eventuales `else if` o al `else`. Si ninguna de las expresiones lógicas resulta cierta se ejecutará la o las líneas entre los paréntesis que siguen al `else`.

```
if (e > f) {
    cout << "e es mayor que f" << endl;
}
else if (e == f) {
    cout << "e es igual a f" << endl;
}
else {
    cout << "e es menor que f" << endl;
}
```

Para C++, una expresión verdadera es igual a `1`, y una falsa es igual a `0`. Esto es, cuando escribimos `if(e>f)`, y `e>f` es falsa, en realidad estamos diciendo `if(0)`. A la inversa, `0`

es considerada una expresión falsa, y cualquier valor no nulo es considerado una expresión verdadera. Así, podríamos hacer que una porción del código siempre se ejecute (o nunca) poniendo directamente `if(1)` o `if(0)`, respectivamente.

Naturalmente, lo anterior no tiene mucho sentido, pero un error habitual (y particularmente difícil de detectar) es escribir `a = b` en vez de `a == b` en una expresión lógica. Esto normalmente trae consecuencias indeseadas, pues la asignación `a = b` es una función que se evalúa siempre al nuevo valor de `a`. En efecto, una expresión como `a=3` siempre equivale a verdadero, y `a=0` siempre equivale a falso. Por ejemplo, en el siguiente programa:

```
#include <iostream.h>
void main(){
  int k=3;
  if (k==3){
    cout << "k es igual a 3" << endl;
  }
  k=4;
  if (k=3){
    cout << "k es igual a 3" << endl;
  }
}
```

la salida siempre es:

```
k es igual a 3
k es igual a 3
```

aunque entre los dos `if` el valor de `k` cambia.

2.2.2 Expresión condicional.

Una construcción `if else` simple, que sólo asigna un valor distinto a una misma variable según si una proposición es verdadera o falsa, es muy común en programación. Por ejemplo:

```
if (a==b) {
    c = 1;
} else {
    c = 0;
}
```

Existen dos maneras de compactar este código. Éste se puede reemplazar por

```
if (a==b) c = 1;
else c = 0;
```

Sin embargo, esto no es recomendable por razones de claridad al leer el código. Una expresión más compacta y clara, se consigue usando el operador ternario `? :`

```
c = (a==b) ? 1 : 0;
```

Como en el caso de los operadores de incremento y decremento, el uso del operador `?` es preferible para optimizar el ejecutable resultante.

2.2.3 `switch`.

La instrucción `switch` permite elegir múltiples opciones a partir del valor de una variable entera. En el ejemplo siguiente tenemos que si `i==1` la ejecución continuará a partir del caso `case 1:`, si `i==2` la ejecución continuará a partir del caso `case 2:` y así sucesivamente. Si `i` toma un valor que no está enumerado en ningún `case` y existe la etiqueta `default`, la ejecución continuará a partir de ahí. Si no existe `default`, la ejecución continúa luego del último paréntesis cursivo.

```
switch (i)
{
case 1:
     {
         cout << "Caso 1." << endl;
     }
     break;
case 2:
     {
         cout << "Caso 2." << endl;
     }
     break;
default:
     {
         cout << "Otro caso." << endl;
     }
     break;
}
```

La instrucción `break` permite que la ejecución del programa salte a la línea siguiente después de la serie de instrucciones asociadas a `switch`. De esta manera sólo se ejecutarán las líneas correspondiente al `case` elegido y no el resto. Por ejemplo, si `i==1` veríamos en pantalla solo la línea `Caso 1.` En el otro caso, si no existieran los `break`, y también `i==1`, entonces veríamos en pantalla las líneas `Caso 1.`, `Caso 2.` y `Otro caso.` La instrucción `default` es opcional.

2.2.4 `for`.

Una instrucción que permite repetir un bloque de instrucciones un número definido de veces es el `for`. Su sintaxis comienza con una o varias inicializaciones, luego una condición lógica de continuación mientras sea verdadera, y finalmente una o más expresiones que se evalúan vuelta por vuelta no incluyendo la primera vez. Siguiendo al `for(...)` viene una instrucción o un bloque de ellas encerradas entre paréntesis de llave. En el ejemplo siguiente la variable entera `i` es inicializada al valor 1, luego se verifica que la condición lógica sea cierta y se ejecuta el bloque de instrucciones. A la vuelta siguiente se evalúa la expresión a la extrema derecha (suele ser uno o más incrementadores), se verifica que la condición lógica se mantenga cierta y se ejecuta nuevamente el bloque de instrucciones. Cuando la condición lógica es falsa

se termina el *loop*, saltando la ejecución a la línea siguiente al parentésis que indica el fin del bloque de instrucciones del `for`. En este ejemplo, cuando `i=4` la condición de continuación será falsa y terminará la ejecución del `for`.

```
for (int i = 1; i < 4; i++) {
    cout << "Valor del indice: " << i << endl;
}
```

El output correspondiente es:

```
Valor del indice: 1
Valor del indice: 2
Valor del indice: 3
```

Cualquier variable declarada en el primer argumento del `for` es local al *loop*. En este caso, la variable `i` es local, y no interfiere con otras posibles variables `i` que existan en nuestro código.

`for` es una instrucción particularmente flexible. En el primer y tercer argumento del `for` se puede colocar más de una instrucción, separadas por comas. Esto permite, por ejemplo, involucrar más de una variable en el ciclo. El código:

```
for (int i=0, k=20; (i<10) && (k<50); i++, k+=6) {
    cout << "i + k = " << i + k << endl;
}
```

resulta en el output:

```
i + k = 20
i + k = 27
i + k = 34
i + k = 41
i + k = 48
```

Además, la condición de continuación (segundo argumento del `for`), no tiene por qué depender de las variables inicializadas en el primer argumento. Y el tercer argumento no tiene por qué ser un incremento o decremento de las variables del *loop*; puede ser cualquier expresión que queramos ejecutar cada vez que un ciclo termina. En el siguiente ejemplo, además de incrementar los contadores en cada ciclo, se envía un mensaje a pantalla:

```
for (int i=1, k=2;k<20 && i<20;k++, i+=2, cout << "Fin iteracion" << endl){
   cout << " i = " << i << endl;
   cout << " k = " << k << endl;
}
```

El resultado:

```
 i = 1, k = 2
Fin iteracion
 i = 3, k = 3
Fin iteracion
 i = 5, k = 4
```

Todos los argumentos del `for` son opcionales (no los `;`), por lo cual se puede tener un `for` que carezca de inicialización y/o de condición de continuación y/o de una expresión que se evalúe en cada iteración.

Un caso típico en que se aprovecha la opcionalidad de los argumentos del `for` es para tener un *loop* infinito, que puede servir para dejar el programa en pausa indefinida. Para salir del *loop* (y en general, para detener cualquier programa en C++), hay que presionar `^C`:

```
for (; ; ) cout << "Este es un loop infinito, ^C para detenerlo"<< endl;
```

Se puede además, salir abruptamente del *loop* con `break`. El código:

```
  for(int indice=0; indice<10; indice++) {
    int cuadrado = indice*indice ;
    cout << indice << " " ;
    if(cuadrado > 10 ) break ;
  }
  cout << endl;
```

da la salida a pantalla:

```
0 1 2 3 4
```

aun cuando la condición de continuación permite que `indice` llegue hasta `9`.

Finalmente, las variables involucradas en el `for` pueden ser modificadas dentro del ciclo. Por ejemplo, modifiquemos uno de los ejemplos anteriores, cambiando la variable `k` en medio del ciclo:

```
for (int i=1, k=2;k<5 && i<8;k++, i+=2, cout << "Fin iteracion" << endl){
  cout << " i = " << i  << ", k = " << k << endl;
  k = k+5;
}
```

El resultado es:

```
 i = 1, k = 2
Fin iteracion
```

En vez de pasar por el ciclo tres veces como ocurría originalmente, ahora, al cabo del primer ciclo, $k = 2 + 5 = 7 > 5$, y el programa sale del *loop*.

En general no es una buena práctica modificar las variables internas del ciclo en medio de él, porque no es muy ordenado, y el desorden normalmente conduce a los errores en programación, pero ocasionalmente puede ser útil hacer uso de esta libertad que proporciona el lenguaje.

2.2.5 `while`.

La instrucción `while` permite repetir un bloque de instrucciones encerradas entre paréntesis de llave mientras la condición lógica que acompaña al `while` se mantenga cierta. La condición es evaluada antes de que comience la primera iteración; si es falsa en ésta o en una posterior evaluación no se ejecuta el bloque de instrucciones que le siguen y se continúa la ejecución en la línea siguiente al parentésis que indica el fin del bloque asociado al `while`. Hay que notar que la instrucción `while` podría no ejecutarse ni una sola vez si la condición no se cumple inicialmente. Un ejemplo simple:

```
int i=1;
while (i < 3) {
    cout << i++ << " ";
}
```

que da por resultado: `1 2 3`.

En el siguiente *loop*, la salida será: `5 4 3 2 1` (¿Por qué?)

```
int k=5 ;
while(k) {
    cout << k-- <<" ";
}
```

2.2.6 `do... while`.

La instrucción `do... while` es análoga a `while`, salvo que la condición lógica es evaluada después de la primera iteración. Por tanto, el bloque se ejecuta al menos una vez, siempre. Un ejemplo simple:

```
do {
    cout << i++ << endl;
} while (i<=20);
```

Podemos construir de otra manera un *loop* infinito usando `do while`

```
do {
    cout << "Este es un segundo loop infinito, ^C para detenerlo"<< endl;
} while (1);
```

2.2.7 `goto`.

Existe también en C++ una instrucción `goto` que permite saltar de un punto a otro del programa (`goto salto;` permite saltar a la línea que contiene la instrucción `salto:`). Sin embargo, se considera una mala técnica de programación usar `goto`, y siempre se puede diseñar un programa evitándolo. Es altamente no recomendable, pero si su utilización simplifica el código se puede usar.

2.3 Funciones.

Las funciones nos permiten programar partes del procedimiento por separado. Un ejemplo simple de ellas lo vimos en la subsección 2.1.2.

2.3.1 Funciones tipo `void`.

Un caso especial de funciones es aquél en que el programa que llama la función no espera que ésta le entregue ningún valor al terminar. Por ejemplo, en la subsección 2.1.2, la función `PrintHola` simplemente imprime un mensaje en pantalla. El resto del programa no necesita de ningún resultado parcial proveniente de la ejecución de dicha función. La definición de estas funciones debe ir precedida de la palabra `void`, como en el ejemplo citado.

2.3.2 `return`.

Si deseamos definir una función que calcule una raíz cuadrada, evidentemente esperamos que la función nos entregue un resultado: el valor de la raíz cuadrada. En este caso hay que traspasar el valor de una variable desde la función al programa que la llamó. Esto se consigue con `return`. Veamos un ejemplo muy simple:

```
int numero(){
    int i = 3;
    return i;
}

main(){
    cout << "Llamamos a la funcion" << endl;
    cout << "El numero es: " << numero() << endl;
    int i = 5;
    i = i + numero();
    cout << "El numero mas 5 es: " << i << endl;
}
```

En este caso, la función simplemente entrega el valor de la variable interna `i`, es decir 3, el cual puede ser usado para salida en pantalla o dentro de operaciones matemáticas corrientes.

Separando declaración e implementación de la función, el ejemplo anterior se escribe:

```
int numero();

main(){ ... }

int numero(){
   int i = 3;
   return i;
}
```

Dos observaciones útiles:

a) La declaración de la función lleva antepuesto el tipo de variable que la función entrega. En el ejemplo, la variable entregada es un entero, `i`, y la declaración debe ser por tanto: `int numero()`. Podemos tener funciones tipo `double`, `char`, `long`, etc., de acuerdo al tipo de variable que corresponde a `return`.

b) La variable `i` que se usa dentro de `main()` y la que se usa dentro de `numero()` son distintas. A pesar de que tienen el mismo nombre, se pueden usar independientemente como si se llamaran distinto. Se dice que `i` es una variable *local*.

Después de `return` debe haber una expresión que se evalúe a una variable del tipo correspondiente, ya sea explícitamente o a través de un *cast* implícito. Las siguientes funciones devuelven un `double` al programa:

```
double f1(){
  double l = 3.0;
  return l;
}

double f2(){
  double l = 3.0, m = 8e10;
  return l*m;
}

double f3(){
  int l = 3;
  return l;
}
```

Sin embargo, la siguiente función hará que el compilador emita una advertencia, pues se está tratando de devolver un `double` donde debería ser un `int`, y la conversión implica una pérdida de precisión:

```
int f4(){
  double l=3.0;
  return l;
}
```

Naturalmente, podemos modificar la función anterior haciendo una conversión explícita antes de devolver el valor: `return int(l)`.

2.3.3 Funciones con parámetros.

Volviendo al ejemplo de la raíz cuadrada, nos gustaría llamar a esta función con un parámetro (el número al cual se le va a calcular la raíz cuadrada). Consideremos por ejemplo una función que necesita un solo parámetro, de tipo `int`, y cuyo resultado es otro `int`:

```
int funcion(int i){
    i+=4;
```

```
    return i;
}

main(){
    int i = 3;
    cout << "El valor de la funcion es " << funcion(i)
         << endl;
    cout << "El valor del parametro es " << i << endl;
}
```

El resultado en pantalla es:

```
El valor de la funcion es 7
El valor del parametro es 3
```

La función `funcion` entrega el valor del parámetro más 4. Usamos el mismo nombre (`i`) para las variables en `main` y `funcion`, pero son variables locales, así que no interfieren. Lo importante es notar que cuando se llama la función, la reasignación del valor de `i` (`i+=4`) ocurre sólo para la variable local en `funcion`; el parámetro externo mantiene su valor.

Separando declaración e implementación el ejemplo anterior se escribe:

```
int funcion(int);

main(){...}

int funcion(int i){
   i+=4;
   return i;
}
```

Si nuestra función necesita más parámetros, basta separarlos con comas, indicando para cada uno su tipo:

```
int funcion2(int,double);
void funcion3(double,int,float);
double funcion4(float);
```

El compilador verifica cuidadosamente que cada función sea llamada con el número de parámetros adecuados, y que cada parámetro corresponda al tipo especificado. En los ejemplos anteriores, `funcion2` debe ser llamada siempre con dos argumentos, el primero de los cuales es `int` y el segundo `double`. Como siempre, puede ser necesario un *cast* implícito (si se llama `funcion2` con el segundo argumento `int`, por ejemplo), pero si no existe una regla de conversión automática (llamando a `funcion2` con el primer argumento `double`, por ejemplo), el compilador enviará una advertencia. Además, el compilador verifica que el valor de retorno de la función sea usado como corresponde. Por ejemplo, en las dos líneas:

```
double m = funcion2(2,1e-3);
int k = funcion4(0.4);
```

la primera compilará exitosamente (pero hay un *cast* implícito), y la segunda dará una advertencia.

Existen dos modos de transferir parámetros a una función:

a) Por valor. Se le pasan los parámetros para que la función que es llamada copie sus valores en sus propias variables locales, las cuales desaparecerán cuando la función termine y no tienen nada que ver con las variables originales.

   Hasta ahora, en todos los ejemplos de esta subsección el traspaso de parámetros ha sido por valor. En la funcion `int funcion(int)`, en el código de la página 65, lo que ha ocurrido es que la función copia el *valor* de la variable externa `i` en una nueva variable (que también se llama `i`, pero está en otra dirección de memoria). El valor con el que trabaja la función es la copia, manteniendo inalterada la variable original.

b) Por referencia. Se le pasa la dirección de memoria de los parámetros. La función llamada puede modificar el valor de tales variables.

   La misma función de la página 65 puede ser modificada para que el paso de parámetros sea por referencia, modificando la declaración:

```
int funcion(int &);

main()
{
    int i = 3;
    cout << "El valor de la funcion es " << funcion(i)
         << endl;
    cout << "El valor del parametro es " << i << endl;
}
int funcion(int & i)
{
    i+=4;
    return i;
}
```

   En vez de traspasarle a `funcion` el valor del parámetro, se le entrega la *dirección* de memoria de dicha variable. Debido a ello, `funcion` puede modificar el valor de la variable. El resultado en pantalla del último programa será:

```
El valor de la funcion es 7
El valor del parametro es 7
```

   Debido a que las variables dejan de ser locales, el paso de parámetros por reerencia debe ser usado con sabiduría. De hecho el ejemplo presentado es poco recomendable. Peor aún, el problema es no sólo que las variables dejan de ser locales, sino que *es imposible saber que no lo son* desde el `main`. En efecto, el `main` en ambas versiones de `funcion` es el mismo. Lo único que cambió es la declaración de la función. Puesto que un

usuario normal usualmente no conoce la declaración e implementación de cada función que desea usar (pues pueden haber sido hechas por otros programadores), dejamos al usuario en la indefensión.

Por otro lado, hay al menos dos situaciones en que el paso de referencia es la única opción viable para entregar los parámetros. Un caso es cuando hay que cuidar el uso de la memoria. Supongamos que una función necesita un parámetros que es una matriz de 10 millones de filas por 10 millones de columnas. Seguramente estaremos llevando al límite los recursos de nuestra máquina, y sería una torpeza pasarle la matriz por valor: ello involucraría, primero, duplicar la memoria utilizada, con el consiguiente riesgo de que nuestro programa se interrumpa; y segundo, haría el programa más lento, porque la función necesitaría llenar su versión local de la matriz elemento por elemento. Es decir, nada de eficiente. En esta situación, el paso por referencia es lo adecuado.

Un segundo caso en que el paso por referencia es recomendable es cuando efectivamente nuestra intención es cambiar el valor de las variables. El ejemplo típico es el intercambio de dos variables entre sí, digamos `a1=1` y `a2=3`. Luego de ejecutar la función queremos que `a1=3` y `a1=1`. El siguiente código muestra la definición y el uso de una función para esta tarea, y por cierto requiere el paso de parámetros por referencia:

```
#include <iostream.h>
void  swap(int &,int &);

void main(){

  int i = 3, k=10;
  swap(i,k);
  cout <<  "Primer argumento: " << i << endl;
  cout <<  "Segundo argumento: " << k << endl;
}

void swap(int & j,int & p){
  int temp = j;
  j = p;
  p = temp;
}
```

El output es:

```
Primer argumento: 10
Segundo argumento: 3
```

En el ejemplo de la matriz anterior, sería interesante poder pasar el parámetro por referencia, para ahorrar memoria y tiempo de ejecución, pero sin correr el riesgo de que nuestra matriz gigantesca sea modificada por accidente. Afortunadamente existe el modo de hacerlo, usando una palabra que ya hemos visto antes: `const`. En el siguiente código:

```
int f5(const int &);
main(){...}
int f5(const int & i){...};
```

`f5` recibirá su único argumento por referencia, pero, debido a la presencia del modificador `const`, el compilador avisará si se intenta modificar el argumento en medio del código de la función.

2.3.4 Parámetros por defecto.

C++ permite que omitamos algunos parámetros de la función llamada, la cual reemplaza los valores omitidos por otros predeterminados. Tomemos por ejemplo la función `int funcion(int);` de la subsección 2.3.3, y modifiquémosla de modo que si no le entregamos parámetros, asuma que el número entregado fue 5:

```
int funcion(int i = 5){
    i+=4;
    return i;
}

main(){
    cout << "El resultado default es " << funcion() << endl;
    int i = 3;
    cout << "Cuando el parametro vale " << i <<
            " el resultado es " << funcion(i) << endl;
}
```

El output correspondiente es:

```
El resultado default es 9
Cuando el parametro vale 3 el resultado es 7
```

Separando declaración e implementación:

```
int funcion(int = 5);
main(){...}
int funcion(int i){
    i+=4;
    return i;
}
```

Si una función tiene n argumentos, puede tener $m \leq n$ argumentos opcionales. La única restricción es que, en la declaración e implementación de la función, los parámetros opcionales ocupen los últimos lugares m lugares:

```
void f1(int,int = 4);
int f2(double,int = 4, double = 8.2);
double f3(int = 3,double = 0.0, int = 0);
```

En este caso, `f1(2)`, `f1(2,8)`, `f2(2.3,5)`, `f3(3)`, `f3()`, y muchas otras, son todas llamadas válidas de estas funciones. Cada vez, los parámetros no especificados son reemplazados por sus valores predeterminados.

2.3.5 Ejemplos de funciones: raíz cuadrada y factorial.

Raíz cuadrada.

Con lo visto hasta ahora, ya podemos escribir un programa que calcule la raíz cuadrada de una función. Para escribir una función, debemos tener claro qué se espera de ella: cuántos y de qué tipo son los argumentos que recibirá, que tipo de valor de retorno deberá tener, y, por cierto, un nombre adecuado. Para la raiz cuadrada, es claro que el argumento es un número. Pero ese número podría ser un entero o un real, y eso al compilador no le da lo mismo. En este punto nos aprovechamos del *cast* implícito: en realidad, basta definir la raíz cuadrada con argumento `double`; de este modo, si se llama la función con un argumento `int`, el compilador convertirá automáticamente el `int` en `double` y nada fallará. En cambio, si la definiéramos para `int` y la llamamos con argumento `double`, el compilador se quejaría de que no sabe efectuar la conversión. Si el argumento es `double`, evidentemente esperamos que el valor de retorno de la función sea también un `double`. Llamando a la función `raiz`, tenemos la declaración:

```
double raiz(double);
```

Debido a la naturaleza de la función raiz cuadrada, `raiz()` no tendría sentido, y por tanto no corresponde declararla con un valor default.

Ahora debemos pensar en cómo calcular la raíz cuadrada. Usando una variante del método de Newton-Raphson, se obtiene que la secuencia

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$$

converge a $\sqrt{a}$ cuando $n \to \infty$. Por tanto, podemos calcular la raíz cuadrada con aproximaciones sucesivas. El cálculo terminará en el paso N, cuando la diferencia entre el cuadrado de la aproximación actual, x_N, y el valor de a, sea menor que un cierto número pequeño: $|\, x_N - a \,| < \epsilon \ll 1$. El valor de ϵ determinará la precisión de nuestro cálculo. Un ejemplo de código lo encontramos a continuación:

```
#include <iostream.h>
#include <math.h>

double raiz(double);

void main(){

  double r;

  cout.precision(20);
```

```
  cout << "Ingrese un numero: " << endl;
  cin >> r;
  cout << raiz(r) << endl;

}

double raiz(double a){
  double x, dx = 1e3, epsilon = 1e-8;

  while (fabs(dx)>epsilon){
    x = (x + a/x)/2;
    dx = x*x - a;
    cout << "x = " << x << ", precision = " << dx << endl;
   }
  return x;
}
```

Luego de la declaración de la función `raiz`, está `main`, y al final la implementación de `raiz`. En `main` se pide al usuario que ingrese un número, el cual se aloja en la variable `r`, y se muestra en pantalla el valor de su raiz cuadrada. La instrucción `cout.precision(20)` permite que la salida a pantalla muestre el resultado con 20 cifras significativas.

En la implementación de la función hay varios aspectos que observar. Se ha llamado `x` a la variable que contendrá las sucesivas aproximaciones a la raíz. Al final del ciclo, `x` contendrá el valor (aproximado) de la raiz cuadrada. `dx` contiene la diferencia entre el cuadrado de `x` y el valor de `a`. `epsilon` es el número (pequeño) que determina si la aproximación es satisfactoria o no.

El ciclo está dado por una instrucción `while`, y se ejecuta mientras `dx>epsilon`, es decir termina cuando `dx` es suficientemente pequeño. El valor absoluto del real `dx` se obtiene con la función matemática `fabs`, disponible en el header `math.h` incluido al comienzo del programa. Observar que inicialmente `dx=1e3`, esto es un valor muy grande; esto permite que la condición del `while` sea siempre verdadera, y el ciclo se ejecuta al menos una vez.

Dentro del ciclo, se calcula la nueva aproximación, y se envía a pantalla un mensaje con la aproximación actual y la precisión alcanzada (dada por `dx`). Eventualmente, cuando la aproximación es suficientemente buena, se sale del ciclo y la función entrega a `main` el valor de `x` actual, que es la última aproximación calculada.

Factorial.

Otro ejemplo útil es el cálculo del factorial, definido para numeros naturales:

$$n! = n \cdot (n-1) \cdots 2 \cdot 1 \ , \quad 0! \equiv 1 \ .$$

La estrategia natural es utilizar un ciclo `for`, determinado por una variable entera i, que va desde 1 a n, guardando los resultados en una variable auxiliar que contiene el producto de todos los números naturales desde 1 hasta i:

```
#include <iostream.h>

int factorial(int);

int main(){
  int n=25 ;
  cout << "El factorial de " << n << " es: " << factorial(n) << endl;
}

int factorial(int i)
{
  int f =1;
  for (int j=1;j<=i;j++){
    f = f*j;
  }
  return f;
}
```

Observar que la variable auxiliar `f` que contiene el producto de los primeros i números naturales debe ser inicializada a 1. Si se inicializara a 0, `factorial(n)` sería 0 para todo `n`.

Esta función no considera el caso `n=0`, pero al menos para el resto de los naturales funcionará bien.

2.3.6 Alcance, visibilidad, tiempo de vida.

Con el concepto de función hemos apreciado que es posible que coexistan variables con el mismo nombre en puntos distintos del programa, y que signifiquen cosas distintas. Conviene entonces tener en claro tres conceptos que están ligados a esta propiedad:

Alcance (*scope*) La sección del código durante la cual el nombre de una variable puede ser usado. Comprende desde la declaración de la variable hasta el final del cuerpo de la función donde es declarada.

Si la variable es declarada dentro de una función es *local*. Si es definida fuera de todas las funciones (incluso fuera de `main`), la variable es *global*.

Visibilidad Indica cuáles de las variables actualmente al alcance pueden ser accesadas. En nuestros ejemplos (subsección 2.3.3), la variable `i` en `main` aún está al alcance dentro de la función `funcion`, pero no es visible, y por eso es posible reutilizar el nombre.

Tiempo de vida Indica cuándo las variables son creadas y cuándo destruidas. En general este concepto coincide con el alcance (las variables son creadas cuando son declaradas y destruidas cuando la función dentro de la cual fueron declaradas termina), salvo porque es posible definir: (a) variables *dinámicas*, que no tienen alcance, sino sólo tiempo de vida; (b) variables *estáticas*, que conservan su valor entre llamadas sucesivas de una función (estas variables tienen tiempo de vida mayor que su alcance). Para declarar estas últimas se usa un modificador `static`.

El efecto del modificador `static` se aprecia en el siguiente ejemplo:

```
#include <iostream.h>

int f();

int main(){

  cout << f() << endl;
  cout << f() << endl;

}

int f(){
  int x=0;
  x++;
  return x;
}
```

La función `f` simplemente toma el valor inicial de `x` y le suma 1. Como cada vez que la función es llamada la variable local `x` es creada e inicializada, el resultado de este programa es siempre un 1 en pantalla:

```
1
1
```

Ahora modifiquemos la función, haciendo que `x` sea una variable estática:

```
#include <iostream.h>

int f();

int main(){

  cout << f() << endl;
  cout << f() << endl;

}

int f(){
  static int x=0;
  x++;
  return x;
}
```

Ahora, al llamar a `f` por primera vez, la variable `x` es creada e inicializada, pero no destruida cuando la función termina, de modo que conserva su valor cuando es llamada por segunda vez:

```
1
2
```

Veamos un ejemplo de una variable estática en el cálculo del factorial:

```
int factorial2(int i=1){
    static int fac = 1;
    fac*=i;
    return fac ;
}
main (){
  int n=5;
  int m=n;
  while(n>0)  factorial2(n--);
  cout << "El factorial de "<< m << " es = " << factorial2() << endl;
}
```

La idea es, si se desea calcular el factorial de 5, por ejemplo, es llamar a la función `factorial2` una vez, con argumento $n = 5$, y después disminuir n en 1. Dentro de la función, una variable estática (`fac`) aloja el valor $1 * 5 = 5$. Luego se llama nuevamente con $n = 4$, con lo cual `fac=1*5*4`, y así sucesivamente, hasta llegar a $n = 1$, momento en el cual `fac=1*5*4*3*2*1`. Al disminuir n en 1 una vez más, la condición del `while` es falsa y se sale del ciclo. Al llamar una vez más a `factorial2`, esta vez sin argumentos, el programa asume que el argumento tiene el valor predeterminado 1, y así el resultado es `1*5*4*3*2*1*1`, es decir 5!.

Observemos el uso del operador de decremento en este programa: `factorial2(n--)` llama a la función con argumento `n` y *después* disminuye `n` en 1. Ésto es porque el operador de decremento está actuando como sufijo, y es equivalente a las dos instrucciones:

```
factorial2(n);
n--;
```

Si fuera un prefijo [`factorial2(n--)`], primero disminuiría `n` en 1, y llamaría luego a `factorial2` con el nuevo valor de `n`

Este ejemplo de cálculo del factorial ilustra el uso de una variable estática, que aloja los productos parciales de los números enteros, pero no es un buen ejemplo de una función que calcule el factorial, porque de hecho esta función no lo calcula: es `main` quien, a través de sucesivas llamadas a `factorial2`, calcula el factorial, pero la función en sí no.

2.3.7 Recursión.

C++ soporta un tipo especial de técnica de programación, la recursión, que permite que una función se llame a sí misma (esto es no trivial, por cuanto si definimos, digamos, una función `f`, dentro del cuerpo de la implementación no hay ninguna declaración a una función `f`, y por tanto en principio no se podría usar `f` porque dicho nombre no estaría en *scope*; C++ permite soslayar este hecho). La recursión permite definir de modo muy compacto una función que calcule el factorial de un número entero `n`.

```
int factorial3(int n){
    return (n<2) ? 1: n * factorial(n-1);
}

int main(){
  int n=5;
  cout << "El factorial de "<< m << " es = " << factorial3(n) << endl;
}
```

En este tercer ejemplo, el factorial de n es definido en función del factorial de $n-1$. Se ha usado la expresión condicional (operador `?`) para compactar aún más el código. Por ejemplo, al pedir el factorial de 5 la función se pregunta si $5 < 2$. Esto es falso, luego, la función devuelve a `main` el valor `5*factorial3(4)`. A su vez, `factorial3(4)` se pregunta si $4 < 2$; siendo falso, devuelve *a la función que la llamó* (es decir, a `factorial3(5)`), el valor `4*factorial3(3)`. El proceso sigue hasta que `factorial(2)` llama a `factorial3(1)`. En ese momento, $1 < 2$, y la función `factorial3(1)`, en vez de llamar nuevamente al factorial, devuelve a la función que la llamó el valor 1. No hay más llamadas a `factorial3`, y el proceso de recursión se detiene. El resultado final es que `main` recibe el valor `factorial3(5)` = `5*factorial3(4)` = $\cdots$ = `5*4*3*2*factorial3(1)` = `5*4*3*2*1`= 120.

Este tercer código para el cálculo del factorial sí considera el caso $n = 0$, y además es más eficiente, al ser más compacto.

La recursión debe ser empleada con cuidado. Es importante asegurarse de que existe una condición para la cual la recursión se detenga, de otro modo, caeríamos en una recursión infinita que haría inútil nuestro programa. En el caso del factorial, pudimos verificar que dicha condición existe, por tanto el programa es finito. En situaciones más complicadas puede no ser tan evidente, y es responsabilidad del programador —como siempre— revisar que todo esté bajo control.

2.3.8 Funciones internas.

Exiten muchas funciones previamente implementadas en C++ almacenadas en distintas bibliotecas. Una de las bibliotecas importante es la matemática. Para usarla uno debe incluir el archivo de *header* `<math.h>` y luego al compilar agregar al final del comando de compilación `-lm`:

```
g++ -o <salida> <fuente>.cc -lm
```

si se desea crear un ejecutable `<salida>` a partir del código en `<fuente>.cc`.

Veamos algunas de estas funciones:

`pow(x,y)`	Eleva a potencia, x^y
`fabs(x)`	Valor absoluto
`sqrt(x)`	Raíz cuadrada
`sin(x) cos(x)`	Seno y coseno
`tan(x)`	Tangente
`atan(x)`	Arcotangente de x en $[-\pi, \pi]$
`atan2(y, x)`	Arcotangente de y/x en $[-\pi, \pi]$
`exp(x)`	Exponencial
`log(x) log10(x)`	Logaritmo natural y logaritmo en base 10
`floor(x)`	Entero más cercano hacia abajo (e.g. `floor(3.2)=3`)
`ceil(x)`	Entero más cercano hacia arriba (e.g. `ceil(3.2)=4`)
`fmod(x,y)`	El resto de x/y (e.g. `fmod(7.3, 2)=1.3`)

Para elevar a potencias enteras, es más conveniente usar la forma explícita en vez de la función `pow`, i.e. calcular `x^3` como `x*x*x` es más eficiente computacionalmente que `pow(x,3)`, debido a los algoritmos que usa `pow` para calcular potencias. Éstos son más convenientes cuando las potencias no son enteras, en cuyo caso no existe una forma explícita en términos de productos.

2.4 Punteros.

Una de las ventajas de C++ es permitir el acceso directo del programador a zonas de memoria, ya sea para crearlas, asignarles un valor o destruirlas. Para ello, además de los tipos de variables ya conocidos (`int`, `double`, etc.), C++ proporciona un nuevo tipo: el *puntero*. El puntero no contiene el valor de una variable, sino la dirección de memoria en la cual dicha variable se encuentra.

Un pequeño ejemplo nos permite ver la diferencia entre un puntero y la variable a la cual ese puntero "apunta":

```
void main(){
    int i = 42;
    int * p = &i;
    cout << "El valor del puntero es: " << p << endl;
    cout << "Y apunta a la variable: " << *p << endl;
}
```

En este programa definimos una variable `i` entera. Al crear esta variable, el programa reservó un espacio adecuado en algún sector de la memoria. Luego pusimos, en esa dirección de memoria, el valor `42`. En la siguiente línea creamos un puntero a `i`, que en este caso denominamos `p`. Los punteros no son punteros a cualquier cosa, sino punteros a un tipo particular de variable. Ello es manifiesto en la forma de la declaración: `int * p`. En la misma línea asignamos a este puntero un valor. Ese valor debe ser también una dirección de memoria, y para eso usamos `&i`, que es la dirección de memoria donde está `i`. Ya hemos visto antes el uso de `&` para entregar una dirección de memoria, al estudiar paso de parámetros a funciones por referencia (2.3.3).

Al ejecutar este programa vemos en pantalla los mensajes:

```
El valor del puntero es: 0xbffff9d8
Y apunta a la variable: 42
```

Primero obtenemos un número hexadecimal imposible de determinar *a priori*, y que corresponde a la dirección de memoria donde quedó ubicada la variable `i`. La segunda línea nos da el valor de la variable que está en esa dirección de memoria: 42. Puesto que `*` aplicado a un puntero entrega el contenido de esa dirección de memoria, se le denomina *operador de dereferenciación*.

En este ejemplo, hemos creado un puntero que contiene la dirección de memoria de una variable preexistente: declaramos una variable, esa variable queda en alguna dirección de memoria, y después asignamos esa dirección de memoria a un puntero. En este caso, podemos referirnos a la variable tanto por su nombre (`i`) como por su puntero asociado (`p_i`).

También es posible crear directamente una dirección de memoria, sin necesidad de crear una variable antes. En este caso, la única forma de manipular este objeto es a través de su puntero, porque no existe ninguna variable y por tanto ningún nombre asociado a él. Esto se hace con el operador `new`. El mismo ejemplo anterior puede ser reescrito usando sólo punteros:

```
void main(){
    int * p = new int;
    *p = 42;
    cout << "El valor del puntero es: " << p << endl;
    cout << "Y apunta a la variable: " << *p << endl;
    delete p;
}
```

La primera línea crea un nuevo puntero a `int` llamado `p`. `new` verifica que haya suficiente memoria para alojar un nuevo `int`, y si es así reserva ese espacio de memoria. En `p` queda la dirección de la memoria reservada. Esto es equivalente a la declaración `int i;` del programa anterior, salvo que ahora la única manera de accesar esa dirección de memoria es a través del puntero `p`. A continuación se coloca *dentro* de esa dirección (observar la presencia del operador de dereferenciación `*`) el número `42`. El programa manda a pantalla la misma información que la versión anterior, salvo que seguramente el valor de `p` será distinto.

Finalmente, ya que el puntero no volverá a ser usado, la dirección de memoria debe ser liberada para que nuestro u otros programas puedan utilizarla. Ello se realiza con el operador `delete`. Todo puntero creado con `new` debe ser, cuando ya no se utilice, borrado con `delete`. Ello evitará desagradables problemas en nuestro programa debido a fuga de memoria (*memory leak*).

Los punteros tienen gran importancia cuando de manejar datos dinámicos se trata, es decir, objetos que son creados durante la ejecución del programa, en número imposible de predecir al momento de compilar. Por ejemplo, una aplicación X-windows normal que crea una, dos, tres, etc. ventanas a medida que uno abre archivos. En este caso, cada ventana es un objeto dinámico, creado durante la ejecución, y la única forma de manejarlo es a través de un puntero a ese objeto, creado con `new` cuando la ventana es creada, y destruido con `delete` cuando la ventana es cerrada.

2.5 Matrices.

2.5.1 Declaración e inicialización.

Podemos declarar (e inicializar inmediatamente) matrices de enteros, reales de doble precisión, caracteres, etc., según nuestras necesidades.

```
int a[5];
double r[3] = {3.5, 4.1, -10.8};
char palabra[5];
```

Una vez declarada la matriz (digamos `a[5]`), los valores individuales se accesan con `a[i]`, con `i` desde 0 a 4. Por ejemplo, podemos inicializar los elementos de la matriz así:

```
a[0] = 3;
a[3] = 5; ...
```

o si queremos ingresarlos desde el teclado:

```
for (i = 0; i < 5; i++){
    cin >> a[i];
}
```

Y si deseamos escribirlos en pantalla:

```
for (i = 0; i < 5; i++){
    cout >> a[i];
}
```

2.5.2 Matrices como parámetros de funciones.

Si deseamos, por ejemplo, diseñar una función que mande los elementos de una matriz a pantalla, necesitamos entregarle como parámetro la matriz que va a utilizar. Para ello se agrega `[]` luego del nombre de la variable, para indicar que se trata de una matriz:

```
void PrintMatriz(int, double []);

void main(){
     double matriz[5] = {3.5, 5.2, 2.4, -0.9, -10.8};
     PrintMatriz(5, matriz);
}

void PrintMatriz(int i, double a[]){
     for (int j = 0; j < i; j++){
          cout << "Elemento " << j << " = " << a[j] << endl;
     }
}
```

Observemos que la función debe recibir dos parámetros, uno de los cuales es la dimensión de la matriz. Esto se debe a que cuando las matrices son usadas como parámetros la información de su dimensión no es traspasada, y debe ser comunicada independientemente. Una ligera optimización al programa anterior es modificar `main` a:

```
main()
{
     const int dim = 5;
     double matriz[dim] = {3.5, 5.2, 2.4, -0.9, -10.8};
     PrintMatriz(dim, matriz);
}
```

De este modo, si eventualmente cambiamos de opinión y deseamos trabajar con matrices de longitud distinta, sólo hay que modificar una línea de código (la primera) en todo el programa, el cual puede llegar a ser bastante largo por cierto. (En el ejemplo, también habría que cambiar la línea de inicialización de la matriz, porque asume que la matriz requiere sólo 5 elementos, pero de todos modos debería ser clara la enorme conveniencia.) Podemos reescribir este programa con un comando de preprocesador para hacer la definición de la dimensión:

```
#include <iostream.h>
#define DIM 5
main(){
     double matriz[DIM] = {3.5, 5.2, 2.4, -0.9, -10.8};
     PrintMatriz(DIM, matriz);
}
```

Sin embargo, ninguna de estas alternativas resuelve el problema de que el compilador espera que la dimensión de una matriz sea un entero constante, determinado en el momento de la compilación (no de la ejecución).

2.5.3 Asignación dinámica.

La reserva de memoria para la matriz podemos hacerla en forma dinámica ocupando el operador `new` que pedirá al sistema la memoria necesaria, si está disponible el sistema se la asignará. Como con cualquier puntero, una vez desocupado el arreglo debemos liberar la memoria con el comando `delete`.

```
#include <iostream.h>
main()
{
     cout<<"Ingrese la dimension deseada :" ;
     int dim ;
     cin >> dim ;
     double * matriz = new double[dim] ; // Reserva la memoria
     for(int i=0; i < dim; i++) {
        cout << "Ingrese elemento "<< i <<" : ";
        cin >> matriz[i] ;
```

```
     }

     for (int i=0;i<dim;i++){
        cout << matriz[i] << ", ";
     }
     cout << endl;

     delete [] matriz       // Libera la memoria reservada
}
```

Este ejemplo permite apreciar una gran ventaja del uso de punteros, al permitirnos liberarnos de definir la dimensión de una matriz como una constante. Aquí, `dim` es simplemente un `int`. La asignación dinámica permite definir matrices cuya dimensión se determina recién durante la ejecución.

Observemos finalmente que la liberación de memoria, en el caso de arreglos, se hace con el operador `delete []`, no `delete` como en los punteros usuales.

2.5.4 Matrices multidimensionales.

Es fácil declarar e inicializar matrices de más de una dimensión:

```
double array[10][8];
int array[2][3] = {{1, 2, 3},
                   {4, 5, 6}};
```

Una operación usual es definir primero las dimensiones de la matriz, y luego llenar sus elementos uno por uno (o desplegarlos en pantalla), recorriendo la matriz ya sea por filas o por columnas. Hay que tener cuidado del orden en el cual uno realiza las operaciones. En el siguiente código, definimos una matriz de 10 filas y 3 columnas, la llenamos con ceros elemento por elemento, y luego inicializamos tres de sus elementos a números distintos de cero. Finalmente desplegamos la matriz resultante en pantalla:

```
#include <iostream.h>

int main(){
  const int dimx=3, dimy=10;

  double a[dimy][dimx];

   for (int i=0;i<dimy;i++){
     for (int j=0;j<dimx;j++){
        a[i][j]=0;
     }
     cout << endl;
   }
```

```
  a[0][0]=1;
  a[3][2]=2;
  a[9][2]=3;

   for (int i=0;i<dimy;i++){
     for (int j=0;j<dimx;j++){
       cout << a[i][j] << ", ";
     }
     cout << endl;
   }
}
```

Inicializar los elementos a cero inicialmente es particularmente relevante. Si no, la matriz se llenaría con elementos aleatorios.

También es posible definir arreglos bidimensionales dinámicamente. En el siguiente ejemplo, se define una matriz de 200 filas y 400 columnas, inicializándose sus elementos a cero, y finalmente se borra:

```
int main()
{

  int width;
  int height;

  width = 200;
  height = 400;


  double ** matriz = new double * [width];

  for (int i=0;i<width;i++){
    matriz[i] = new double[height];
  }

  for (int i=0;i<width;i++){
    for (int j=0;j<height;j++){
      matriz[i][j] = 0;
    }
  }

  for (int i=0;i<width;i++){
    delete [] matriz[i];
  }
  delete [] matriz;

}
```

Primero se crea, con `new`, un puntero (`matriz`) de dimensión 200, que representará las filas. Cada uno de sus elementos (`matriz[i]`), a su vez, será un nuevo puntero, de dimensión 400, representando cada columna. Por tanto, `matriz` debe ser un puntero a puntero (de dobles, en este caso), y así es definido inicialmente (`double ** ...`). Esto puede parecer extraño a primera vista, pero recordemos que los punteros pueden ser punteros a cualquier objeto, en particular a otro puntero. Luego se crean los punteros de cada columna (primer ciclo `for`). A continuación se llenan los elementos con ceros (segundo ciclo `for`). Finalmente, se libera la memoria, en orden inverso a como fue asignada: primero se libera la memoria de cada columna de la matriz (`delete [] matriz[i]`, tercer ciclo `for`), y por último se libera la memoria del puntero a estos punteros (`delete [] matriz`).

2.5.5 Matrices de caracteres: cadenas (strings).

Una palabra, frase o texto más largo es representado internamente por C++ como una matriz de `char`s. A esto se le llama "cadena" (string). Sin embargo, esto ocasiona un problema, pues las matrices deben ser definidas con dimensión constante (a menos que sean definidas dinámicamente), y las palabras pueden tener longitud arbitraria. La convención de C++ para resolver el problema es aceptar que una cadena tiene longitud arbitraria, pero debe indicar dónde termina. Esto se hace con el `char` nulo: `'\0'`. Así, para asignar a la variable `palabra` el valor "Hola", debe definirse como una matriz de dimensión 5 (una más que el número de letras):

```
char palabra[5] = {'H', 'o', 'l', 'a', '\0'};
```

Para escribir "Hola" en pantalla basta recorrer los elementos de `palabra` uno a uno:

```
for (i = 0; i < 5; i++)
{
    cout << palabra[i];
}
```

Si tuviéramos que hacer esto cada vez que queremos escribir algo a pantalla no sería muy cómodo. Por ello, también podemos escribir "Hola" en pantalla simplemente con `cout << "Hola"`, y de hecho ése fue el primer ejemplo de este capítulo. De hecho, la declaración de `palabra` podría haberse escrito:

```
char palabra[5] = "Hola";
```

Esto ya es bastante más cómodo, aunque persiste la inconsistencia de definir `palabra` con dimensión 5, cuando en realidad al lado derecho de la asignación hay un objeto con sólo 4 elementos (visibles).

Éste y otros problemas asociados con el manejo convencional de cadenas en C++ se resuelven incluyendo el header `string`.

string

El código anterior se puede reescribir:

```
#include <iostream.h>
#include <string>

int main(){
  string palabra = "Hola";
  cout << texto1 << endl;
}
```

Observar que la línea a incluir es `#include <string>`, *sin la extensión ".h"*. Al incluir `string`, las cadenas pueden ser declaradas como objetos tipo `string` en vez de arreglos de `char`. El hecho de que ya no tengamos que definir a priori la dimensión de la cadena es una gran ventaja. De hecho, permite ingresar palabras desde el teclado trivialmente, sin preocuparse de que el input del usuario sea demasiado grande (tal que supere la dimensión del arreglo que podamos haber declarado inicialmente) o demasiado corto (tal que se traduzca en un despilfarro de memoria por reservar más memoria para el arreglo de la que realmente se necesita):

```
#include <iostream.h>
#include <string>

int main(){
  string palabra;
  cin >> texto1;
}
```

Además, este nuevo tipo `string` permite acceder a un sinnúmero de funciones adicionales que facilitan enormemente el manejo de cadenas. Por ejemplo, las cadenas se pueden sumar, donde la suma de cadenas `a` y `b` está definida (siguiendo la intuición) como la cadena que resulta de poner `b` a continuación de `a`:

```
#include <iostream.h>
#include <string>

int main(){
  string texto1 = "Primera palabra";
  string texto2 = "Segunda palabra";
  cout << texto1 << endl << texto2 << endl;
  cout << texto1 + ", " + texto2  << endl;
  // La ultima linea es equivalente a:
  // string texto3 = texto1 + ", " + texto2;
  // cout << texto3 << endl;
}
```

El output de este programa será: `Primera palabra, Segunda palabra`.

Dijimos que es muy fácil ingresar una cadena desde el teclado, pues no es necesario definir la dimensión desde el comienzo. Sin embargo, el código anterior, usando `cin`, no es muy general, porque el input termina cuando el usuario ingresa el primer cambio de

línea o el primer espacio. Esto es muy cómodo cuando queremos ingresar una serie de valores (por ejemplo, para llenar un arreglo), pues podemos ingresarlos ya sea en la forma: `1<Enter> 2<Enter> 3<Enter>`, etc., o `1 2 3`, etc, pero no es óptimo cuando deseamos ingresar texto, que podría constar de más de una palabra y, por tanto, necesariamente incluiría espacios (por ejemplo, al ingresar el nombre y apellido de una persona). Sin explicar demasiado por qué, digamos que la solución a este problema es utilizar una función asociada a `cin` llamada `gets`, y que espera input desde el teclado hasta que el usuario dé el primer cambio de línea. Un ejemplo simple lo encontramos en el siguiente código:

```
#include <iostream.h>
#include <string>

int main(){
  string texto1 = "El resultado es: " ;
  char * texto2;

  cin.gets(&texto2);
  string texto
  cout << texto1 + string(texto2) << endl;
}
```

Observamos que `gets` acepta en realidad un argumento que es un puntero a puntero de caracteres (`texto2` fue declarado como un puntero a `char`, y `gets` es llamado con el argumento `&texto2`, que es la dirección de memoria asociada a `texto2`, i.e. el puntero que apunta a `texto2`.)

De este modo, `gets` espera input desde el teclado hasta el primer cambio de línea, y asigna la cadena ingresada a `texto2`. Sin embargo, si después queremos utilizar `texto2` en conjunto con otras cadenas (definidas como `string`), será necesario convertirla explícitamente a `string` (ver sección 2.1.9). En nuestro código, deseábamos sumar `texto1` con `texto2` y enviar el resultado a pantalla.

2.6 Manejo de archivos

Una operación usual en todo tipo de programas es la interacción con archivos. Ya sea que el programa necesite conocer ciertos parámetros de configuración, hacer análisis estadístico sobre un gran número de datos generados por otro programa, entregar las coordenadas de los puntos de una trayectoria para graficarlos posteriormente, etc., lo que se requiere es un modo de ingresar datos desde, o poner datos en, archivos. En C++ ello se efectúa incluyendo el header `fstream.h`.

2.6.1 Archivos de salida

Observemos el siguiente programa:

```
#include <iostream.h>
```

```
#include <fstream.h>

int main(){
  ofstream nombre_logico("nombre_fisico.dat");
  int i = 3, j;
  cout << i << endl;
  nombre_logico << i << endl;
  cout << "Ingrese un numero entero: ";
  cin >> j;
  cout << i << endl;
  nombre_logico << j << endl;
  nombre_logico.close();
}
```

La primera línea de `main` define un objeto de tipo `ofstream` (*output file stream*). Esto corresponde a un archivo de salida. Dentro de `main` este archivo será identificado por una variable llamada `nombre_logico`, y corresponderá a un archivo en el disco duro llamado `nombre_fisico.dat`. Naturalmente, el identificador `nombre_logico` puede ser cualquier nombre de variable válido para C++, y `nombre_fisico.dat` puede ser cualquier nombre de archivo válido para el sistema operativo. En particular, se pueden también dar nombres que incluyan *paths* absolutos o relativos:

```
ofstream nombre_logico_1("/home/vmunoz/temp/nombre_fisico.dat");
ofstream nombre_logico_2("../nombre_fisico.dat");
```

Las líneas tercera y sexta de `main` envían a `nombre_logico` (es decir, escribe en `nombre_fisico.dat`), las variables `i` y `j`. Observar la analogía que existe entre estas operaciones y las que envían la misma información a pantalla.[4] Si ejecutamos el programa y en el teclado ingresamos el número `8`, al finalizar la ejecución el archivo `nombre_fisico.dat` tendrá los dos números escritos:

```
3
8
```

Finalmente, el archivo creado debe ser cerrado (`nombre_logico.close()`). Si esta última operación se omite en el código, no habrá errores de compilación, y el programa se encargará de cerrar por sí solo los archivos abiertos durante su ejecución, pero un buen programador debiera tener cuidado de cerrarlos explícitamente. Por ejemplo, un mismo programa podría desear utilizar un mismo archivo más de una vez, o varios programas podrían querer acceder al mismo archivo, y si no se ha insertado un `close` en el punto adecuado esto podría provocar problemas.

El archivo indicado al declarar la variable de tipo `ofstream` tiene modo de escritura, para permitir la salida de datos hacia él. Si no existe un archivo llamado `nombre_fisico.dat` es creado; si existe, los contenidos antiguos se pierden y son reemplazados por los nuevos. No

[4]Esta analogía no es casual y se entiende con el concepto de *clases* (Sec. 2.8). `fstream` e `iostream` definen clases que heredan sus propiedades de un objeto abstracto base, común a ambas, y que en el caso de `iostream` se concreta en la salida estándar —pantalla—, y en el de `fstream` en un archivo.

siempre deseamos este comportamiento. A veces deseamos agregar la salida de un programa a un archivo de texto ya existente. En ese caso la declaración del archivo es diferente, para crear el archivo en modo *"append"*:

```
#include <iostream.h>
#include <fstream.h>

int main(){

  ofstream nombre_logico("nombre_fisico.dat",ios::app);
  int i = 3;

  nombre_logico << i << endl;
  nombre_logico.close();
}
```

Si ejecutamos este programa y el archivo `nombre_fisico.dat` no existe, será creado. El resultado será un archivo con el número `3` en él. Al ejecutarlo por segunda vez, los datos se ponen a continuación de los ya existentes, resultando el archivo con el contenido:

```
3
3
```

La línea del tipo `ofstream a("b")` es equivalente a una del tipo `int i=3`, declarando una variable (`a`/`i`) de un cierto tipo (`ofstream`/`int`) y asignándole un valor simultáneamente `"b"`/`3`. Como para los tipos de variables predefinidos de C++, es posible separar declaración y asignación para una variable de tipo `ofstream`:

```
ofstream a;
a.open("b");
```

es equivalente a `ofstream a("b")`. Esto tiene la ventaja de que podríamos usar el mismo nombre lógico para identificar dos archivos físicos distintos, usados en distintos momentos del programa:

```
ofstream a;
a.open("archivo1.txt");
// Codigo en que "archivo1.txt" es utilizado
a.close();
a.open("archivo2.txt");
// Ahora "archivo2.txt" es utilizado
a.close();
```

Observar la necesidad del primer `close`, que permitirá liberar la asociación de `a` a un nombre físico dado, y reutilizar la variable lógica en otro momento.

En los ejemplos hemos escrito solamente variables de tipo `int` en los archivos. Esto por cierto no es restrictivo. Cualquiera de los tipos de variables de C++ —`float`, `double`, `char`, etc.— se puede enviar a un archivo del mismo modo. Dicho esto, en el resto de esta sección seguiremos usando como ejemplo el uso de `int`.

2.6.2 Archivos de entrada

Ya sabemos que enviar datos a un archivo es tan fácil como enviarlos a pantalla. ¿Cómo hacemos ahora la operación inversa, de leer datos desde un archivo? Como es de esperar, es tan fácil como leerlos desde el teclado. Para crear un archivo en modo de lectura, basta declararlo de tipo `ifstream` (*input file stream*). Por ejemplo, si en `nombre_logico.dat` tenemos los siguientes datos:

```
3
6
9
12
```

el siguiente programa,

```
#include <iostream.h>
#include <fstream.h>

int main(){

  ifstream nombre_logico("nombre_fisico.dat");
  int i, j,k,l;

  nombre_logico >> i >> j >> k >> l;

  cout << i << "," << j << "," << k << "," << l << endl;

  nombre_logico.close();
}
```

será equivalente a asignar `i=3`, `j=6`, `k=9`, `l=12`, y luego enviar los datos a pantalla. Observar que la sintaxis para ingresar datos desde un archivo, `nombre_logico >> i`, es idéntica a `cin >> i`, para hacerlo desde el teclado. Al igual que `cin`, espacios en blanco son equivalentes a cambios de línea, de modo que el archivo podría haber sido también:

```
3 6 9 12
```

Por cierto, el ingreso de datos desde un archivo se puede hacer con cualquier técnica, por ejemplo, usando un `for`:

```
  ifstream nombre_logico("nombre_fisico.dat");
  int i;
  for (int j=0;j<10;j++){
    nombre_logico >> i;
    cout << i << ",";
  }
  nombre_logico.close();
}
```

Como con `ofstream`, es posible separar declaración e implementación:

```
ifstream a;
a.open("b");
a.close();
```

2.6.3 Archivos de entrada y salida

Ocasionalmente nos encontraremos con la necesidad de usar un mismo archivo, en el mismo programa, a veces para escribir datos, y otras veces para leer datos. Por ejemplo, podríamos tener una secuencia de datos en un archivo, leerlos, y de acuerdo al análisis de esos datos agregar más datos a continuación del mismo archivo, o reemplazar los datos ya existentes con otros. Necesitamos entonces un tipo de variable flexible, que pueda ser usado como entrada y salida. Ese tipo es `fstream`. Todo lo que hemos dicho para `ofstream` y `ifstream` por separado es cierto simultánemente para `fstream`.[5] Para especificar si el archivo debe ser abierto en modo de escritura o lectura, `open` contiene el argumento `ios::out` o `ios::in`, respectivamente. Por ejemplo, el siguiente código escribe el número `4` en un archivo, y luego lo lee desde el mismo archivo:

```
#include <iostream.h>
#include <fstream.h>

int main(){
  fstream nombre_logico;
  nombre_logico.open("nombre_fisico.dat",ios::out);
  int i = 4,j;

  nombre_logico << i << endl;
  nombre_logico.close();

  nombre_logico.open("nombre_fisico.dat",ios::in);

  nombre_logico >> j;
  j = j++;
  cout << j << endl;
  nombre_logico.close();
}
```

Las dos primeras líneas de `main` separan declaración y asignación, y son equivalentes a `fstream nombre_logico("nombre_fisico.dat",ios::out);`, pero lo hemos escrito así para hacer evidente la simetría entre el uso del archivo como salida primero y como entrada después.

De lo anterior, se deduce que:

[5]Nuevamente, este hecho se debe al concepto de clases que subyace a las definiciones de estos tres tipos de variables; `fstream` es una clase derivada a la vez de `ofstream` y de `ifstream`, heredando las propiedades de ambas.

```
fstream archivo_salida("salida.dat",ios::out);
fstream archivo_entrada("entrada.dat",ios::in);
```

es equivalente a

```
ofstream archivo_salida("salida.dat");
ifstream archivo_entrada("entrada.dat");
```

2.7 `main` como función

Para ejecutar un programa compilado en C++, escribimos su nombre en el prompt:

```
user@host:~/$ programa
```

Si el mismo usuario desea ejecutar alguno de los comandos del sistema operativo, debe hacer lo mismo:

```
user@host:~/$ ls
```

Sin embargo, `ls` es en realidad el nombre de un archivo ejecutable en el directorio `/bin`, de modo que en realidad no hay diferencias entre nuestro programa y un comando del sistema operativo en ese sentido. Sin embargo, éstos pueden recibir argumentos y opciones. Por ejemplo, para ver todos los archivos que comienzan con `l` en el directorio local basta con darle a `ls` el argumento `l*`: `ls l*`. Si queremos ordenar los archivos en orden inverso de modificación, basta dar otro argumento, en forma de opción: `ls -tr l*`. Se ve entonces que los argumentos de un archivo ejecutable permiten modificar el comportamiento del programa de modos específicos.

¿Es posible hacer lo mismo con archivos ejecutables hechos por el usuario? La respuesta es sí, y para eso se usan los argumentos del `main`. Recordemos que `main` es una función, pero hasta el momento no hemos aprovechado esa característica. Simplemente sabemos que el programa empieza a ejecutarse en la líñea donde está la función `main`. Además, siempre hemos escrito esa línea como `main()`. Sin embargo, `main`, como cualquier función, es capaz de aceptar argumentos. Específicamente, acepta dos argumentos, el primero es un entero (que cuenta el número de argumentos que `main` recibió), y el segundo es un puntero a un arreglo de caracteres (que contiene los distintos argumentos, en forma de cadenas de caracteres, que se le entregaron.

Por ejemplo:

```
#include <iostream.h>
main( int argc, char * argv[])
{
     for(int i = 0; i < argc; i++) cout << argv[i] << endl ;
}
```

Si llamamos a este programa `argumentos`, obtenemos distintas salidas al llamarlo con distintos argumentos:

```
user@host:~/$ argumentos
argumentos
user@host:~/$ argumentos ap k
argumentos
ap
k
user@host:~/$ argumentos -t -s arg1
argumentos
-t
-s
arg1
```

Observar que el primer argumento del programa es siempre el nombre del propio programa.

Naturalmente, éste es un ejemplo muy simple. Es tarea del programador decidir cómo manejar cada una de las opciones o argumentos que se le entregan al programa desde la línea de comandos, escribiendo el código correspondiente.

Un segundo aspecto con el cual no hemos sido sistemáticos es que `main`, como toda función, tiene un tipo de retorno. En el caso de `main`, ese tipo debe ser `int`. Este `int` es entregado al sistema operativo, y puede servir para determinar si el programa se ejecutó con normalidad o si ocurrió algo anormal. Podríamos hacer ese valor de retorno igual a 0 o 1, respectivamente. Así, la siguiente estructura es correcta:

```
int main(){
   // Codigo

   return 0;
}
```

En este caso, el programa entrega siempre el valor 0 al sistema operativo.

Los códigos del tipo:

```
main(){
   // Codigo
}
```

o

```
void main(){
   // Codigo
}
```

también compilan, pero el compilador emite una advertencia si es llamado con la opción `-Wall` (*Warning all*). En el primer caso, la advertencia es:

```
warning: ANSI C++ forbids declaration ‘main’ with no type
```

En el segundo:

```
return type for ‘main’ changed to ‘int’
```

En general, siempre es conveniente compilar con la opción `-Wall`, para lograr que nuestro código esté realmente correcto (`g++ -Wall <archivo>.cc -o <archivo>`).

2.8 Clases.

C++ dispone de una serie de tipos de variables con los cuales nos está permitido operar: `int`, `double`, `char`, etc. Creamos variables de estos tipos y luego podemos operar con ellos:

```
int x, y;
x = 3;
y = 6;
int z = x + y;
```

No hay, sin embargo, en C++, una estructura predefinida que corresponda a números complejos, vectores de dimensión n o matrices, por ejemplo. Y sin embargo, nos agradaría disponer de números complejos que pudiéramos definir como

```
z = (3,5);
w = (6,8);
```

y que tuvieran sentido las expresiones

```
a = z + w;
b = z * w;
c = z / w;
d = z + 3;
e = modulo(z);
f = sqrt(z);
```

Todas estas expresiones son completamente naturales desde el punto de vista matemático, y sería bueno que el lenguaje las entendiera. Esto es imposible en el estado actual, pues, por ejemplo, el signo `+` es un operador que espera a ambos lados suyos un número. Sumar cualquier cosa con cualquier cosa no significa nada necesariamente, así que sólo está permitido operar con números. Pero los humanos sabemos que los complejos son números. ¿Cómo decírselo al computador? ¿Cómo convencerlo de que sumar vectores o matrices es también posible matemáticamente, y que el mismo signo `+` debería servir para todas estas operaciones?

La respuesta es: a través del concepto de *clases*. Lo que debemos hacer es definir una clase de números complejos. Llamémosla `Complejo`. Una vez definida correctamente, `Complejo` será un tipo más de variable que el compilador reconocerá, igual que `int`, `double`, `char`, etc. Y será tan fácil operar con los `Complejos` como con todos los tipos de variables preexistentes. Esta facilidad es la base de la extensibilidad de que es capaz C++, y por tanto de todas las propiedades que lo convierten en un lenguaje muy poderoso.

Las clases responden a la necesidad del programador de construir objetos o tipos de datos que respondan a sus necesidades. Si necesitamos trabajar con vectores de 5 coordenadas, será natural definir una clase que corresponda a vectores con 5 coordenadas; si se trata de un programa de administración de personal, la clase puede corresponder a un empleado, con sus datos personales como elementos.

Si bien es cierto uno puede trabajar con clases en el contexto de orientación al procedimiento, las clases muestran con mayor propiedad su potencial con la orientación al objeto, donde cada objeto corresponde a una clase. Por ejemplo, para efectuar una aplicación para Windows, la ventana principal, las ventanas de los archivos abiertos, la barra de menú, las cajas de diálogo, los botones, etc., cada uno de estos objetos estará asociado a una clase.

2.8.1 Definición.

Digamos que queremos una clase para representar los empleados de una empresa. Llamémosla `Persona`. La convención aceptada es que los nombres de las clases comiencen con mayúscula. Esto es porque las clases, recordemos, corresponderán a tipos de variables tan válidos como los internos de C++ (`int`, `char`, etc.). Al usar nombres con mayúscula distiguimos visualmente los nombres de un tipo de variable interno y uno definido por el usuario.

La estructura mínima de la definición de la clase `Persona` es:

```
class Persona
{

};
```

Todas las características de la clase se definen entre los parentésis cursivos.

2.8.2 Miembros.

Se denomina *miembros* de una clase a todas las variables y funciones declaradas dentro de una clase. Por ejemplo, para personas, es natural caracterizarlas por su nombre y su edad. Y si se trata de empleados de una empresa, es natural también tener una función que entregue su sueldo:

```
class Persona
{
     char nombre[20];
     int edad;
     double sueldo();
}
```

Los miembros de una clase pueden tener cualquier nombre, excepto el nombre de la propia clase dentro de la cual se definen, ese nombre está reservado.

2.8.3 Miembros públicos y privados.

Una clase distingue información (datos o funciones) privada (accesible sólo a otros miembros de la misma clase) y pública (accesible a funciones externas a la clase). La parte privada corresponde a la estructura interna de la clase, y la parte pública a la implementación (típicamente funciones), que permite la interacción de la clase con el exterior.

Consideremos ahora nuestro deseo de tener una clase que represente números complejos. Un número complejo tiene dos números reales (parte real e imaginaria), y ésos son elementos privados, es decir, parte de su estructura interna. Sin embargo, nos gustaría poder modificar y conocer esas cantidades. Eso sólo puede hacerse a través de funciones públicas.

```
class Complejo
{
private:
```

```
    double real, imaginaria;
public:
    void setreal(double);
    void setimag(double);
    double getreal();
    double getimag();
};
```

En este ejemplo, los miembros privados son sólo variables, y los miembros públicos son sólo funciones. Éste es el caso típico, pero puede haber variables y funciones de ambos tipos.

2.8.4 Operador de selección (.).

Hemos definido una clase de números complejos y funciones que nos permiten conocer y modificar las partes real e imaginaria. ¿Cómo se usan estos elementos? Consideremos el siguiente programa de ejemplo:

```
class Complejo
{
private:
    double real, imaginaria;
public:
    void setreal(double);
    void setimag(double);
    double getreal();
    double getimag();
};

void main()
{
    Complejo z, w;

    z.setreal(3);
    z.setimag(2.8);
    w.setreal(1.5);
    w.setimag(5);
    cout << "El primer numero complejo es: " << z.getreal()
         << " + i*" << z.getimag() << endl;
    cout << "El segundo es: " << w.getreal() << " + i*"
         << z.getimag() << endl;
}
```

Vemos en la primera línea de `main` cómo la clase `Complejo` se usa del mismo modo que usaríamos `int` o `double`. Ahora `Complejo` es un tipo de variable tan válido como los tipos predefinidos por C++. Una vez definida la variable, el operador de selección (`.`) permite acceder a las funciones públicas correspondientes a la clase `Complejo`, aplicadas a la variable

particular que nos interesa: `z.setreal(3)` pone en la parte real del `Complejo z` el número 3, y `w.setreal(1.5)` hace lo propio con `w`.

2.8.5 Implementación de funciones miembros.

Ya sabemos cómo declarar funciones miembros en el interior de la clase y cómo usarlas. Ahora veamos cómo se implementan.

```
void Complejo::setreal(double x)
{
    real = x;
}

void Complejo::setimag(double x)
{
    imaginaria = x;
}

double Complejo::getreal()
{
    return real;
}

double Complejo::getimag()
{
    return imaginaria;
}
```

Como toda función, primero va el tipo de la función (`void` o `double` en los ejemplos), luego el nombre de la función y los argumentos. Finalmente la implementación. Lo diferente es que el nombre va precedido del nombre de la clase y el operador "`::`" .

2.8.6 Constructor.

Al declarar una variable, el programa crea el espacio de memoria suficiente para alojarla. Cuando se trata de variables de tipos predefinidos en C++ esto no es problema, pero cuando son tipos definidos por el usuario C++ debe saber cómo construir ese espacio. La función que realiza esa tarea se denomina *constructor*.

El constructor es una función pública de la clase, que tiene el mismo nombre que ella. Agreguemos un constructor a la clase `Complejo`:

```
class Complejo
{
private:
    double real,imaginaria;
public:
```

```
    Complejo(double,double);
    void setreal(double);
    void setimag(double);
    double getreal();
    double getimag();
};

Complejo::Complejo (double x, double y)
: real(x), imaginaria(y)
{}
```

Definir el constructor de esta manera nos permite crear en nuestro programa variables de tipo `Complejo` y asignarles valores sin usar `setreal()` o `setimag()`:

```
Complejo z (2, 3.8);
Complejo w = Complejo(6.8, -3);
```

En el constructor se inicializan las variables internas que nos interesa inicializar al momento de crear un objeto de esta clase.

Si una de las variables internas a inicializar es una cadena de caracteres, hay que inicializarla de modo un poco distinto. Por ejemplo, si estamos haciendo una clase `Persona` que sólo tenga el nombre de una persona, entonces podemos definir la clase y su constructor en la forma:

```
class Persona
{
private:
    char nombre[20];
public:
    Persona(char []);
};
Persona::Persona(a[])
{
   strcpy(nombre,a);
}
```

Si uno no especifica el constructor de una clase C++ crea uno default, pero en general será insuficiente para cualquier aplicación realmente práctica. Es una mala costumbre ser descuidado y dejar estas decisiones al computador.

2.8.7 Destructor.

Así como es necesario crear espacio de memoria al definir una variable, hay que deshacerse de ese espacio cuando la variable deja de ser necesaria. En otras palabras, la clase necesita también un *destructor*. Si la clase es `Complejo`, el destructor es una función pública de ella, llamada `~Complejo`.

```
class Complejo
{
private:
    double real, imaginaria;
public:
    Complejo(double,double);
    ~Complejo(void);
    void setreal(double);
    void setimag(double);
    double getreal();
    double getimag();
};

Complejo::Complejo (double x, double y): real(x), imaginaria(y)
{
}

Complejo::~Complejo(void)
{
}
```

Como con los constructores, al omitir un destructor C++ genera un *default*, pero es una mala costumbre ... , etc.

2.8.8 Matrices de clases.

Una clase es un tipo de variable como cualquier otra de las predefinidas en C++. Es posible construir matrices con ellas, del mismo modo que uno tiene matrices de enteros o caracteres. La única diferencia con las matrices usuales es que no se pueden sólo declarar, sino que hay que inicializarlas simultáneamente. Por ejemplo, si queremos crear una matriz que contenga 2 números complejos, la línea

```
Complejo z[2];
```

es incorrecta, pero sí es aceptable la línea

```
Complejo z[2] = {Complejo(3.5,-0.8), Complejo(-2,4)};
```

2.9 Sobrecarga.

Para que la definición de nuevos objetos sea realmente útil, hay que ser capaz de hacer con ellos muchas acciones que nos serían naturales. Como ya comentamos al introducir el concepto de clase, nos gustaría sumar números complejos, y que esa suma utilizara el mismo signo `+` de la suma usual. O extraerles la raíz cuadrada, y que la operación sea tan fácil como escribir `sqrt(z)`. Lo que estamos pidiendo es que el operador `+` o la función `sqrt()` sean *polimórficos*, es decir, que actúe de distinto modo según el tipo de argumento que se

le entregue. Si `z` es un real, `sqrt(z)` calculará la raíz de un número real; si es complejo, calculará la raíz de un número complejo.

La técnica de programación mediante la cual podemos definir funciones polimórficas se llama *sobrecarga*.

2.9.1 Sobrecarga de funciones.

Digamos que la raíz cuadrada de un número complejo $a + ib$ es $(a/2) + i(b/2)$. (Es más complicado en realidad, pero no queremos escribir las fórmulas ahora.)

Para sobrecargar la función `sqrt()` de modo que acepte números complejos basta definirla así:

```
Complejo sqrt(Complejo z)
{
     return Complejo (z.getreal()/2, z.getimag()/2);
}
```

Observemos que definimos una función `sqrt` que acepta argumentos de tipo `Complejo`, y que entrega un número del mismo tipo. Cuando pidamos la raíz de un número, el computador se preguntará si el número en cuestión es un `int`, `double`, `float` o `Complejo`, y según eso escogerá la versión de `sqrt` que corresponda.

Con la definición anterior podemos obtener la raíz cuadrada de un número complejo simplemente con las instrucciones:

```
Complejo z(1,3);
Complejo raiz = sqrt(z);
```

2.9.2 Sobrecarga de operadores.

¿Cómo le decimos al computador que el signo `+` también puede aceptar números complejos? La respuesta es fácil, porque para C++ un operador no es sino una función, y la acción de sobrecargar que ya vimos sirve en este caso también. La sintaxis es:

```
Complejo operator + (Complejo z, Complejo w)
{
    return Complejo (z.getreal() + w.getreal(),
                     z.getimag() + w.getimag());
}
```

2.9.3 Coerción.

Sabemos definir $a + b$, con a y b complejos. Pero ¿qué pasa si a o b son enteros? ¿O reales? Pareciera que tendríamos que definir no sólo

```
Complejo operator + (Complejo a, Complejo b);
```

sino también todas las combinaciones restantes:

```
Complejo operator + (Complejo a, int b);
Complejo operator + (Complejo a, float b);
Complejo operator + (int a, Complejo b);
```

etcétera.

En realidad esto no es necesario. Por cierto, un número real es un número complejo con parte imaginaria nula, y es posible hacerle saber esto a C++, usando la posibilidad de definir funciones con parámetros default. Basta declarar (en el interior de la clase) el constructor de los números complejos como

```
Complejo (double, double = 0);
```

Esto permite definir un número complejo con la instrucción:

```
Complejo c = Complejo(3.5);
```

resultando el número complejo $3.5 + i \cdot 0$. Y si tenemos una línea del tipo:

```
Complejo c = Complejo(3,2.8) + 5;
```

el computador convertirá implícitamente el entero 5 a `Complejo` (sabe cómo hacerlo porque el constructor de números complejos acepta también un solo argumento en vez de dos), y luego realizará la suma entre dos complejos, que es entonces la única que es necesario definir.

2.10 Herencia.

Herencia es el mecanismo mediante el cual es posible definir clases a partir de otras, preservando parte de las propiedades de la primera y agregando o modificando otras.

Por ejemplo, si definimos la clase `Persona`, toda `Persona` tendrá una variable miembro que sea su `nombre`. Si definimos una clase `Hombre`, también será `Persona`, y por tanto debería tener `nombre`. Pero además puede tener `esposa`. Y ciertamente no toda `Persona` tiene `esposa`. Sólo un `Hombre`.

C++ provee mecanismos para implementar estas relaciones lógicas y poder definir una clase `Hombre` a partir de `Persona`. Lo vemos en el siguiente ejemplo:

```
class Persona
{
private:
    char nombre[20];
public:
    Persona(char [] = "");
    ~Persona(void);
    char getname();
}

class Hombre : public Persona
{
```

```
private:
    char esposa[20];
public:
    Hombre(char a[]) : Persona(a)
    { };
    char getwife();
    void setwife();
}
```

Primero definimos una clase `Persona` que tiene `nombre`. Luego definimos una clase `Hombre` *a partir de* `Persona` (con la línea `class Hombre : public Persona`). Esto permite de modo automático que `Hombre` tenga también una variable `nombre`. Y finalmente, dentro de la clase `Hombre`, se definen todas aquellas características adicionales que una `Persona` no tiene pero un `Hombre` sí: `esposa`, y funciones miembros para modificar y obtener el nombre de ella.

Un ejemplo de uso de estas dos clases:

```
Persona cocinera("Maria");
Hombre panadero("Claudio");
panadero.setwife("Estela");

cout << cocinera.getname() << endl;
cout << panadero.getname() << endl;
cout << panadero.getwife() << endl;
```

Observemos que `panadero` también tiene una función `getname()`, a pesar de que la clase `Hombre` no la define explícitamente. Esta función se ha *heredado* de la clase de la cual `Hombre` se ha derivado, `Persona`.

Capítulo 3

Gráfica.

versión final 1.1-010821

En este capítulo queremos mostrar algunas de las posibilidades gráficas presentes en Linux. Cubriendo temas como la visualización, conversión, captura y creación de archivos gráficos. Sólo mencionaremos las aplicaciones principales en cada caso centrándonos en sus posibilidades más que en su utilización específica, ya que la mayoría posee una interfase sencilla de manejar y con amplia documentación.

3.1 Visualización de archivos gráficos.

Si disponemos de un archivo gráfico conteniendo algún tipo de imagen lo primero que es importante determinar es en que tipo de formato gráfico está codificada. Existen un número realmente grande de diferentes tipos de codificaciones de imágenes, cada una de ellas se considera un formato gráfico. Por razones de reconocimiento inmediato del tipo de formato gráfico se suelen incluír en el nombre del archivo, que contiene la imagen, una trío de letras finales, conocidas como la extensión, que representan el formato. Por ejemplo: bmp, tiff, jpg, ps, eps, fig, gif entre muchas otras.

¿De qué herramientas disponemos en Linux para visualizar estas imágenes? La respuesta es que en Linux se dipone de variadas herramientas para este efecto.

Si se trata de archivos de tipo *PostScript* o *Encapsulated PostScript*, identificados por la extensión `ps` o `eps`, existen las aplicaciones `gv`, `gnome-gv` o `kghostview`, todos programas que nos permitirán visualizar e imprimir este tipo de archivos. Si los archivos son tipo *Portable Document Format*, con extensión `pdf`, tenemos las aplicaciones `gv`, `acroread` o `xpdf`, Con todas ellas podemos ver e imprimir dicho formato. Una mención especial requieren los archivos *DeVice Independent* con extensión `dvi` ya que son el resultado de la compilación de un documento TEX o LATEX, para este tipo de archivo existen las aplicaciones `xdvi` y `kdvi` entre otras. La primera aplicación sólo permite visualizar estos archivos y no imprimirlos, para hacer esto, los archivos se transforman a `ps` y se imprime como cualquier otro Postscript.

Para la gran mayoría de formatos gráficos más conocidos y usualmente usados para almacenar fotos existen otra serie se programas especializados en visualización que son capaces de entender la mayoría de los formatos más usados. Entre estos programas podemos mencionar: Eye de Gnome (`eog`), Electric Eyes (`eeyes`), `kview` o `display`. Podemos mencionar que aplicaciones como `display` entienden sobre ochenta formatos gráficos distintos entre los que se encuentran `ps`, `eps`, `pdf`, `fig`, `html`, entre muchos otros.

3.2 Modificando imágenes

Si queremos modificaciones como rotaciones, ampliaciones, cortes, cambios de paleta de colores, filtros o efectos sencillos `display` es la herramienta precisa. Pero si se desea intervenir la imagen en forma profesional, el programa `gimp` es el indicado. El nombre `gimp` viene de *GNU Image Manipulation Program*. Se puede usar esta aplicación para editar y manipular imágenes. Pudiendo cargar y salvar en una variedad de formatos, lo que permite usarlo para convertir entre ellos. Gimp puede también ser usado como programa de pintar, de hecho posee una gran variedad de herramientas en este sentido tales como brocha de aire, lápiz clonador, tijeras inteligentes, curvas bezier, etc. Además, permite incluír *plugins* que realizan gran variedad de manipulaciones de imagen. Como hecho anecdótico podemos mencionar que la imagen oficial de Tux, el pingüino mascota de Linux, fue creada en `gimp`.

3.3 Conversión entre formatos gráficos.

El problema de transformar de un formato a otro es una situación usual en el trabajo con archivos gráficos. Muchos software tienen salidas muy restringidas en formato o con formatos arcaicos (`gif`) y se presenta la necesidad de convertir estos archivos de salida en otros formatos que nos sean más manejables o prácticos. Como ya se mencionó, `gimp` puede ser usado para convertir entre formatos gráficos. También `display` permite este hecho. Sin embargo, en ambos casos la conversión es vía menús, lo cual lo hace engorroso para un gran número de conversiones e imposible para conversiones de tipo automático. Existe un programa llamado `convert` que realiza conversiones desde la línea de comando. Este programa junto con `display`, `import` y varios otros forman la *suite* gráfica *ImageMagick* una de las más importantes en UNIX en general y en especial en Linux y que ya ha sido migrada a otras plataformas. Además de la clara ventaja de automatización que proporciona `convert`, posee otro aspecto interesante, puede convertir un grupo de imágenes asociadas en una secuencia de animación o película. Veamos la sintaxis para este programa:

```
user@host:~/imagenes$convert cockatoo.tiff cockatoo.jpg

user@host:~/secuencias$convert -delay 20 dna.* dna.gif
```

En el primer caso convierte el archivo `cockatoo` de formato `tiff` a formato `jpg`. En el segundo a partir de un conjunto de archivos `gif` numerados correlativamente crea una secuencia animada con imágenes que persisten por 20 centésimas de segundos en un formato conocido como `gif` animado, muy usado en sitios en internet.

3.4 Captura de pantalla.

A menudo se necesita guardar imágenes que sólo se pueden generar a tiempo de ejecución, es decir, mientras corre nuestro programa genera la imagen pero no tiene un mecanismo propio para exportarla o salvarla como imagen. En este caso necesitamos capturar la pantalla y poderla almacenar en un archivo para el cual podamos elegir el formato. Para estos efectos

existe un programa, miembro también de la *suite ImageMagick*, llamado `import` que permite hacer el trabajo. La sintaxis es

```
import figure.eps
```

```
import -window root root.jpeg
```

En el primer caso uno da el comando en un terminal y queda esperando hasta que uno toque alguna de las ventanas, la cual es guardada en este caso en un archivo `figure.eps` en formato *PostScript*. La extensión le indica al programa qué formato usar para almacenar la imagen. En el segundo caso uno captura la pantalla completa en un archivo `root.jpeg`. Este comando puede ser dado desde la consola de texto para capturar la imagen completa en la pantalla gráfica.

3.5 Creando imágenes.

Para imágenes artísticas sin duda la alternativa es `gimp`, todo le que se dijo respecto a sus posibilidades para modificar imágenes se aplica también en el caso de crearlas. En el caso de necesitar imágenes más bien técnicas como esquemas o diagrámas o una ilustración para aclarar un problema la alternativa de `xfig` es muy poderosa. El programa `xfig` es una herramienta manejada por menús que permite dibujar y manipular objetos interactivamente. Las imágenes pueden ser salvadas, en formato `xfig`, y posteriormente editadas. La documentación del programa está en `html` y es fácil de accesar y muy completa. La gran ventaja de `xfig` es que trabaja con objetos y no con bitmaps. Además, puede exportar las imágenes a una gran cantidad de formatos: LaTeX, *Metafont*, *PostScript* o *Encapsulated PostScript* o bien `gif`, `jpeg` y muchos otros.

Habitualmente los dibujos necesarios para ilustrar problemas en Física en tareas, pruebas y apuntes son realizados con este software, exportados a *PostScript* e incluídos en los respectivos archivos LaTeX. También existe una herramienta extremadamente útil que permite convertir un archivo *PostScript*, generado de cualquier manera, a un archivo `fig` que puede ser editado y modificado. Esta aplicación que transforma se llama `pstoedit` y puede llegar a ser realmente práctica.

Una aparente limitación de `xfig` es que se podría pensar que no podemos incluír curvas analíticas, es decir, si necesitamos ilustrar una función gaussiana no podemos pretender "dibujarla" con las herramientas de que dispone `xfig` ¿cómo resolver este problema?, simple veremos que un software que gráfica funciones analíticas como `gnuplot` permite exportar en formato `fig` luego `xfig` puede leer el archivo y editarlo. Además `xfig` permite importar e incluír imágenes del tipo *bitmap*, agregando riqueza a los diagramas que puede generar.

Una característica destacable del programa es que trabaja por capas, las cuales son tratadas independientemente, uno puede poner un objeto sobre otro o por debajo de otro logrando diferentes efectos. Algunos programas de presentación gráficos basados en LaTeX y `pdf` están utilizando esta capacidad para lograr animaciones de imágenes.

Finalmente este programa permite construir una biblioteca de objetos reutilizables ahorrando mucho trabajo. Por ejemplo, si uno dibuja los elementos de un circuito eléctrico y los almacena en el lugar de las bibliotecas de imágenes podrá incluír estos objetos en futuros trabajos. El programa viene con varias bibliotecas de objetos listas para usar.

3.6 Graficando funciones y datos.

Existen dos aplicaciones que permiten graficar datos de un archivo: `gnuplot` y `xmgrace`. La primera está basada en la línea de comando y permite gráficos en 2 y 3 dimensiones, pudiendo ademas graficar funciones directamente sin pasar por un archivo de datos. La segunda es una aplicación basada en menús que permite un resultado final de mucha calidad y con múltiples variantes, su debilidad es que sólo hace gráficos bidimensionales.

El programa `gnuplot` se invoca de la línea de comando y da un *prompt* en el mismo terminal desde el cual se puede trabajar, veamos una sesión de `gnuplot`:

```
jrogan@huelen:~$ gnuplot

        G N U P L O T
        Linux version 3.7
        patchlevel 1
        last modified Fri Oct 22 18:00:00 BST 1999

        Copyright(C) 1986 - 1993, 1998, 1999
        Thomas Williams, Colin Kelley and many others

        Type ‘help‘ to access the on-line reference manual
        The gnuplot FAQ is available from
        <http://www.ucc.ie/gnuplot/gnuplot-faq.html>

        Send comments and requests for help to <info-gnuplot@dartmouth.edu>
        Send bugs, suggestions and mods to <submit@bugs.debian.org>

Terminal type set to 'x11'
gnuplot> plot sqrt(x)
gnuplot> set xrange[0:5]
gnuplot> set xlabel" eje de las x"
gnuplot> replot
gnuplot> set terminal postscript
Terminal type set to 'postscript'
Options are 'landscape noenhanced monochrome dashed defaultplex "Helvetica" 14'
gnuplot> set output "mygraph.ps"
gnuplot> replot
gnuplot> set terminal X
Terminal type set to 'X11'
Options are '0'
gnuplot> set xrange[-2:2]
gnuplot> set yrange[-2:2]
gnuplot> splot exp(-x*x-y*y)
gnuplot> plot "myfile.dat" w l
gnuplot> exit
jrogan@huelen:~$
```

En el caso de `xmgrace` es mucho más directo manejarlo ya que está basado en menús. Además existe abundante documentación del software. Próximamente, estará disponible bajo licencia gnu el software *SciGraphica*. Esta es una aplicación de visualización y análisis de data científica que permite el despliege de gráficos en 2 y 3 dimensiones además de poder exportar el resultado en formato *PostScript*. Realmente esta aplicación nació como un intento de clonar el programa comercial `origen` no disponible para Linux.

3.7 Graficando desde nuestros programas.

Finalmente para poder graficar desde nuestro propio programa necesitamos alguna biblioteca gráfica, en nuestros caso usaremos las bibliotecas `plplot`. El comando de compilación incluido en un *script* será:

```
#!/bin/bash
export PLPLOT_LIB="/usr/lib/plplot"
g++ -o $1 $1.cc -lplplot  -lplcxx -ltclmatrix  -ltk8.2 -ltcl8.2  \
-L/usr/X11R6/lib -lX11 -lm
```

Veamos algunos ejemplos:

```
#include <iostream.h>
#include <math.h>
#include <unistd.h>
#include <plplot.h>

main()
{
  cout << "Ingrese el numero de puntos : ";
  int n =0 ;
  cin >> n ;
  float * x = new float[n] ;
  float * y = new float[n] ;
  plsdev("xwin") ;
  plinit() ;
  plenv(-M_PI, M_PI, -1, 1, 0, 0 ) ;

  for( int ind=0; ind < n; ind++) {
    float arg = -M_PI+ 2.0*M_PI*float(ind)/float(n-1) ;
    x[ind] = arg ;
    y[ind] = cos(arg) ;
  }
  plline(n, x, y) ;
  plend() ;
  cout << endl ;
  delete [] x ;
  delete [] y ;
```

```
}
```

Este programa grafica la función seno con un número de puntos dado. Veamos un caso tridimensional

```
#include <iostream.h>
#include <math.h>
#include <plplot.h>
#include <stdlib.h>
#define nX    100              /* Data points in x */
#define nY    100              /* Datat points in y */


main()
{
  float * x = new float[nX] ;
  float * y = new float[nY] ;

  float * * z = (float **) malloc(nX * sizeof(float *));
  for (int indP = 0; indP < nX; indP++) {
    z[indP] = (float *) malloc(nY * sizeof(float));
  }
  plsdev("xwin") ;
  plinit() ;
  float xmin2d =-2.9 ;
  float xmax2d = 2.9 ;
  float ymin2d = -2.9 ;
  float ymax2d = 2.9 ;
  plenv(xmin2d, xmax2d, ymin2d, ymax2d, 0, -2 ) ;
  float basex =2.5 ;
  float basey =2.5 ;
  float height = 2.5 ;
  float xmin =-2.0 ;
  float xmax = 2.0 ;
  float ymin = -2.0 ;
  float ymax = 2.0 ;
  float zmin = 0.0 ;
  float zmax = 1.0 ;
  float alt = 30.0 ;
  float az = 30.0 ;
  plw3d(basex, basey, height, xmin, xmax, ymin, ymax, zmin, zmax, alt, az) ;

  for (int indX=0; indX <nX; indX++) {
    for( int indY=0; indY < nY; indY++) {
      float argX= -2.0 + 4.0*float(indX)/float(nX-1) ;
      float argY= -2.0 + 4.0*float(indY)/float(nY-1) ;
```

```
      x[indX]= argX ;
      y[indY]= argY ;
      float r=argX*argX+argY*argY ;
      z[indX][indY] = sin(M_PI*r)/(M_PI*r) ;
  }
         plbox3("bnstu", "x axis", 0.0, 0,
              "bnstu", "y axis", 0.0, 0,
              "bcdmnstuv", "z axis", 0.0, 0);

  plot3d(x, y, z, nX, nY, 1, 1) ;
  plend() ;
  cout << endl ;
  delete [] x ;
  delete [] y ;
}
```

Ahora un caso de gráficos multiple:

```
#include <iostream.h>
#include <math.h>
#include <unistd.h>
#include <plplot.h>
#define nX    100              /* Data points in x */
#define nY    100              /* Datat points in y */

main()
{
   plssub(2,2) ;
   int n =1000 ;
  float * x = new float[nX] ;
  float * y = new float[nY] ;
  float * x2 = new float[n] ;
  float * y2 = new float[n] ;
  plsdev("xwin") ;
  plinit() ;
  plenv(-M_PI, M_PI, -1, 1, 0, 0 ) ;
  for( int ind=0; ind < n; ind++) {
    float arg = -M_PI+ 2.0*M_PI*float(ind)/float(n-1) ;
    x2[ind] = arg ;
    y2[ind] = cos(arg) ;
  }
  plline(n, x2, y2) ;
  pllab( "Hola", "chao", "nada") ;

  plenv(-M_PI, M_PI, -1, 1, 0, 0 ) ;
  for( int ind=0; ind < n; ind++) {
```

```
  float arg = -M_PI+ 2.0*M_PI*float(ind)/float(n-1) ;
  x2[ind] = arg ;
  y2[ind] = tan(arg) ;
}
plline(n, x2, y2) ;
  plenv(-M_PI, M_PI, 0, 1, 0, 0 ) ;
for( int ind=0; ind < n; ind++) {
  float arg = -M_PI+ 2.0*M_PI*float(ind)/float(n-1) ;
  x2[ind] = arg ;
  y2[ind] = exp(-arg*arg) ;
}
plline(n, x2, y2) ;

   float * * z = (float **) malloc(nX * sizeof(float *));
   for (int indP = 0; indP < nX; indP++) {
          z[indP] = (float *) malloc(nY * sizeof(float));
   }

   float xmin2d =-2.9 ;
   float xmax2d = 2.9 ;
   float ymin2d = -2.9 ;
   float ymax2d = 2.9 ;
   plenv(xmin2d, xmax2d, ymin2d, ymax2d, 0, -2 ) ;
   float basex =2.5 ;
   float basey =2.5 ;
   float height = 2.5 ;
   float xmin =-2.0 ;
   float xmax = 2.0 ;
   float ymin = -2.0 ;
   float ymax = 2.0 ;
   float zmin = 0.0 ;
   float zmax = 1.0 ;
   float alt = 30.0 ;
   float az = 30.0 ;

   plw3d(basex, basey, height, xmin, xmax, ymin, ymax, zmin, zmax, alt, az) ;

   for (int indX=0; indX <nX; indX++) {
      for( int indY=0; indY < nY; indY++) {
         float argX= -2.0 + 4.0*float(indX)/float(nX-1) ;
         float argY= -2.0 + 4.0*float(indY)/float(nY-1) ;
         x[indX]= argX ;
         y[indY]= argY ;
         float r=argX*argX+argY*argY ;
         z[indX][indY] = sin(M_PI*r)/(M_PI*r) ;
```

```
        }
     }
        plbox3("bnstu", "x axis", 0.0, 0,
               "bnstu", "y axis", 0.0, 0,
               "bcdmnstuv", "z axis", 0.0, 0);

          plot3d(x, y, z, nX, nY, 1, 1) ;

  plend() ;
  cout << endl ;
  delete [] x ;
  delete [] y ;
}
```

Finalmente una primitiva animación

```
#include <iostream.h>
#include <math.h>
#include <unistd.h>
#include <plplot.h>

main()
{
  int n =1000 ;
  float * x = new float[n] ;
  float * y = new float[n] ;
  plsdev("xwin") ;
  plinit() ;
  plenv(-M_PI, M_PI, -1, 1, 0, -2 ) ;

  for (int indt=0; indt <5000; indt++) {
    for( int ind=0; ind < n; ind++) {
      float arg = -M_PI+ 2.0*M_PI*float(ind)/float(n-1) ;
      x[ind] = arg ;
      y[ind] = cos(arg-M_PI*indt/160.0) ;
    }
    plbop() ;
    plcol(6) ;
    plline(n, x, y) ;
    usleep(50000) ;
  }
  plend() ;
  cout << endl ;
  delete [] x ;
  delete [] y ;
}
```

Capítulo 4

Una breve introducción a Octave/Matlab

versión final 2.0-19 agosto 2002

4.1 Introducción

Octave es un poderoso software para análisis numérico y visualización. Muchos de sus comandos son compatibles con Matlab. En estos apuntes revisaremos algunas características de estos programas. En realidad, el autor de este capítulo ha sido usuario durante algunos años de Matlab, de modo que estos apuntes se han basado en ese conocimiento, considerando los comandos que le son más familiares de Matlab. En la mayoría de las ocasiones he verificado que los comandos descritos son también compatibles con Octave, pero ocasionalmente se puede haber omitido algo . . . .

Matlab es una abreviación de *Matrix Laboratory*. Los elementos básicos con los que se trabaja con matrices. Todos los otros tipos de variables (vectores, texto, polinomios, etc.), son tratados como matrices. Esto permite escribir rutinas optimizadas para el trabajo con matrices, y extender su uso a todos los otros tipos de variables fácilmente.

4.2 Interfase con el programa

Con Octave/Matlab se puede interactuar de dos modos: un modo interactivo, o a través de *scripts*. Al llamar a Octave/Matlab (escribiendo `octave` en el prompt, por ejemplo), se nos presenta un prompt. Si escribimos `a=1`, el programa responderá `a=1`. Alternativamente, podemos escribir `a=3;` (con punto y coma al final), y el programa no responderá (elimina el eco), pero almacena el nuevo valor de `a`. Si a continuación escribimos `a`, el programa responderá `a=3`. Hasta este punto, hemos usado el modo interactivo.

Alternativamente, podemos introducir las instrucciones anteriores en un archivo, llamado, por ejemplo, `prueba.m`. En el prompt, al escribir `prueba`, y si nuestro archivo está en el path de búsqueda del programa, las líneas de `prueba.m` serán ejecutadas una a una. Por ejemplo, si el archivo consta de las siguientes cuatro líneas:

```
a=3;
a
```

```
a=5
a
```

el programa responderá con

```
a=3
a=5
a=5
```

`prueba.m` corresponde a un *script*. Todas las instrucciones de Octave/Matlab pueden ejecutarse tanto en modo interactivo como desde un *script*. En Linux se puede ejecutar un archivo de comandos Octave de modo *stand-alone* incluyendo en la primera línea:
`#!/usr/bin/octave -q`.

4.3 Tipos de variables

4.3.1 Escalares

A pesar de que éstos son sólo un tipo especial de matrices (ver subsección siguiente), conviene mencionar algunas características específicas.

- Un número sin punto decimal es tratado como un entero exacto. Un número con punto decimal es tratado como un número en doble precisión. Esto puede no ser evidente en el output. Por *default*, `8.4` es escrito en pantalla como `8.4000`. Tras la instrucción `format long`, sin embargo, es escrito como `8.40000000000000`. Para volver al formato original, basta la instrucción `format`.
- Octave/Matlab acepta números reales y complejos. La unidad imaginaria es `i`: `8i` y `8*i` definen el mismo número complejo. Como `i` es una varible habitualmente usada en iteraciones, también está disponible `j` como un sinónimo. Octave/Matlab distinguen entre mayúsculas y minúsculas.
- Octave/Matlab representa de manera especial los infinitos y cantidades que no son números. `inf` es infinito, y `NaN` es un no-número (Not-a-Number). Por ejemplo, escribir `a=1/0` no arroja un error, sino un mensaje de advertencia, y asigna a `a` el valor `inf`. Análogamente, `a=0/0` asigna a `a` el valor `NaN`.

4.3.2 Matrices

Este tipo de variable corresponde a escalares, vectores fila o columna, y matrices convencionales.

Construcción

Las instrucciones:

```
a = [1 2 ; 3 4]
```

ó

```
a = [1, 2; 3, 4]
```

definen la matriz $\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$. Las comas (opcionales) separan elementos de columnas distintas, y los punto y coma separan elementos de filas distintas. El vector fila $(1 \quad 2)$ es

```
b = [1 2]
```

y el vector columna $\begin{pmatrix}1\\ 2\end{pmatrix}$ es

```
c = [1;2]
```

Un número se define simplemente como `d = [3]` ó `d = 3`.

Nota importante: Muchas funciones de Octave/Matlab en las páginas siguientes aceptan indistintamente escalares, vectores filas, vectores columnas, o matrices, y su output es un escalar, vector o matriz, respectivamente. Por ejemplo, `log(a)` es un vector fila si `a` es un vector fila (donde cada elemento es el logaritmo natural del elemento correspondiente en `a`), y un vector columna si `a` es un vector columna. En el resto de este manual *no se advertira* este hecho, y se pondrán ejemplos con un solo tipo de variable, en el entendido que el lector está conciente de esta nota.

Acceso y modificación de elementos individuales

Accesamos los elementos de cada matriz usando los índices de filas y columnas, que parten de uno. Usando la matriz `a` antes definida, `a(1,2)` es 2. Para modificar un elemento, basta escribir, por ejemplo, `a(2,2) = 5`. Esto convierte a la matriz en $\begin{pmatrix}1 & 2\\ 3 & 5\end{pmatrix}$. En el caso especial de vectores filas o columnas, basta un índice. (En los ejemplos anteriores, `b(2)` = `c(2)` = 2.)

Una característica muy importante del programa es que toda matriz es redimensionada automáticamente cuando se intenta modificar un elemento que sobrepasa las dimensiones actuales de la matriz, llenando con ceros los lugares necesarios. Por ejemplo, si `b = [1 2]`, y en seguida intentamos la asignación `b(5) = 8`, `b` es automáticamente convertido al vector fila de 5 elementos `[1 2 0 0 8]`.

Concatenación de matrices

Si $a = \begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$, $b = \begin{pmatrix}5 & 6\end{pmatrix}$, $c = \begin{pmatrix}7\\ 8\end{pmatrix}$, entonces

`d = [a c]`	$d = \begin{pmatrix}1 & 2 & 7\\ 3 & 4 & 8\end{pmatrix}$
`d = [a; b]`	$d = \begin{pmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{pmatrix}$
`d = [a [0; 0] c]`	$d = \begin{pmatrix}1 & 2 & 0 & 7\\ 3 & 4 & 0 & 8\end{pmatrix}$

4.3.3 Strings

Las cadenas de texto son casos particulares de vectores fila, y se construyen y modifican de modo idéntico.

Construcción

Las instrucciones

```
t = ['un buen texto']
t = ["un buen texto"]
t = 'un buen texto'
t = "un buen texto"
```

definen el mismo string `t`.

Acceso y modificación de elementos individuales

```
r = t(4)
t(9) = 's'
```

```
r = 'b'
texto = 'un buen sexto'
```

Concatenación

```
t = 'un buen texto';
t1 = [t ' es necesario']
```

```
t1 = 'un buen texto es necesario'
```

4.3.4 Estructuras

Las estructuras son extensiones de los tipos de variables anteriores. Una estructura consta de distintos campos, y cada campo puede ser una matriz (es decir, un escalar, un vector o una matriz), o una string.

Construcción

Las líneas

```
persona.nombre = 'Eduardo'
persona.edad = 30
persona.matriz_favorita = [2 8;10 15];
```

definen una estructura con tres campos, uno de los cuales es un string, otro un escalar, y otro una matriz:

```
persona =
{
nombre = 'Eduardo';
edad = 30;
matriz_favorita = [2 8; 10 15];
}
```

Acceso y modificación de elementos individuales

```
s = persona.nombre
persona.nombre = 'Claudio'
persona.matriz_favorita(2,1) = 8
```

```
 s = 'Eduardo'
persona =
{
nombre = 'Claudio';
edad = 30;
matriz_favorita = [2 8; 8 15];
}
```

4.4 Operadores básicos

4.4.1 Operadores aritméticos

Los operadores `+`, `-`, `*` corresponden a la suma, resta y multiplicación convencional de matrices. Ambas matrices deben tener la misma dimensión, a menos que una sea un escalar. Un escalar puede ser sumado, restado o multiplicado de una matriz de cualquier dimensión.

`.*` y `./` permiten multiplicar y dividir elemento por elemento. Por ejemplo, si

$$a = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad b = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

entonces

```
c = a.*b
```
$$c = \begin{pmatrix} 5 & 12 \\ 21 & 32 \end{pmatrix}$$

```
c = a./b
```
$$c = \begin{pmatrix} 0.2 & 0.3333 \\ 0.42857 & 0.5 \end{pmatrix}$$

Si `b` es un escalar, `a.*b` y `a./b` equivalen a `a*b` y `a/b`.
`a^b` es `a` elevado a `b`, si `b` es un escalar. `a.^b` eleva cada elemento de `a` a `b`.
`a'` es la matriz $\mathtt{a}^\dagger$ (traspuesta y conjugada)
`a.'` es la matriz traspuesta de `a`.

4.4.2 Operadores relacionales

Los siguientes operadores están disponibles:
`<  <=  >  >=  ==  ~=`

El resultado de estas operaciones es 1 (verdadero) ó 0 (falso). Si uno de los operandos es una matriz y el otro un escalar, se compara el escalar con cada elemento de la matriz. Si ambos operandos son matrices, el test se realiza elemento por elemento; en este caso, las matrices deben ser de igual dimensión. Por ejemplo,

```
a = [1 2 3];
b = [4 2 1];
c = (a<3);
d = (a>=b);
```
$c = (1, 1, 0)$
$d = (0, 1, 1)$

4.4.3 Operadores lógicos

Los siguientes símbolos corresponden a los operadores `AND`, `OR` y `NOT`:

`&` `|` `~`

El resultado de estas operaciones es 1 (verdadero) ó 0 (falso).

4.4.4 El operador :

Es uno de los operadores fundamentales. Permite crear vectores y extraer submatrices.

: crea vectores de acuerdo a las siguientes reglas:

`j:k`	es lo mismo que `[j,j+1,...,k]`, si `j<=k`.
`j:i:k`	es lo mismo que `[j,j+i,j+2*i,...,k]`, si `i>0` y `j<k`, o si `i<0` y `j>k`.

: extrae submatrices de acuerdo a las siguientes reglas:

`A(:,j)`	es la `j`-ésima columna de `A`.
`A(i,:)`	es la `i`-ésima fila de `A`.
`A(:,:)`	es `A`.
`A(:,j:k)`	es `A(:,j)`, `A(:,j+1)`, ... , `A(:,k)`.
`A(:)`	son todos los elementos de `A`, agrupados en una única columna.

4.4.5 Operadores de aparición preferente en scripts

Los siguientes operadores es más probable que aparezcan durante la escritura de un *script* que en modo interactivo.

`%` : Comentario. El resto de la línea es ignorado.

`...` : Continuación de línea. Si una línea es muy larga y no cabe en la pantalla, o por alguna otra razón se desea dividir una línea, se puede usar el operador `...` . Por ejemplo,

```
m = [1 2 3 ...
4 5 6];
```

es equivalente a

```
m = [1 2 3 4 5 6];
```

4.5 Comandos matriciales básicos

Antes de revisar una a una diversas familias de comandos disponibles, y puesto que las matrices son el elemento fundamental en Octave/Matlab, en esta sección reuniremos algunas de las funciones más frecuentes sobre matrices, y cómo se realizan en Octave/Matlab.

Op. aritmética	`+, -, *, .*, /`	(ver subsección 4.4.1)
Conjugar	`conj(a)`	
Trasponer	`a.'`	
Trasponer y conjugar	`a'`	
Invertir	`inv(a)`	
Autovalores, autovectores	`[v,d]=eig(a)`	(ver subsección 4.6.5)
Determinante	`det(a)`	
Extraer elementos	`:`	(ver subsección 4.4.4)
Traza	`trace(a)`	
Dimensiones	`size(a)`	
Exponencial	`exp(a)`	(elemento por elemento)
	`expm(a)`	(exponencial matricial)

4.6 Comandos

En esta sección revisaremos diversos comandos de uso frecuente en Octave/Matlab. Esta lista no pretende ser exhaustiva (se puede consultar la documentación para mayores detalles), y está determinada por mi propio uso del programa y lo que yo considero más frecuente debido a esa experiencia. Insistimos en que ni la lista de comandos es exhaustiva, ni la lista de ejemplos o usos de cada comando lo es. Esto pretende ser sólo una descripción de los aspectos que me parecen más importantes o de uso más recurrente.

4.6.1 Comandos generales

`clear` Borra variables y funciones de la memoria

`clear`	Borra todas las variables en memoria
`clear a`	Borra la variable `a`

`disp` Presenta matrices o texto

`disp(a)` presenta en pantalla los contenidos de una matriz, sin imprimir el nombre de la matriz. `a` puede ser una string.

```
disp('     c1       c2');
disp([.3 .4]);
```

$c1$	$c2$
0.30000	0.40000

`load, save` Carga/Guarda variables desde el disco

`save fname a b`	Guarda las variables `a` y `b` en el archivo `fname`
`load fname`	Lee el archivo `fname`, cargando las definiciones de variables en él definidas.

`size,length` Dimensiones de una matriz/largo de un vector

Si a es una matrix de $n \times m$:

`d = size(a)`	`d = [m,n]`
`[m,n] = size(a)`	Aloja en `m` el número de filas, y en `n` el de columnas

Si b es un vector de n elementos, `length(b)` es n.

`who` Lista de variables en memoria

`quit` Termina Octave/Matlab

4.6.2 Como lenguaje de programación

Control de flujo

`for`

```
n=3;                                    a=[1 4 9]
for i=1:n
    a(i)=i^2;
end
```

Para Octave el vector resultante es columna en vez de fila.

Observar el uso del operador `:` para generar el vector `[1 2 3]`. Cualquier vector se puede utilizar en su lugar: `for i=[2 8 9 -3]`, `for i=10:-2:1` (equivalente a `[10 8 6 4 2]`), etc. son válidas. El ciclo `for` anterior se podría haber escrito en una sola línea así:

```
for i=1:n, a(i)=i^2; end
```

`if, elseif, else`

Ejemplos:

a)
```
if a~=b, disp(a); end
```

b)
```
if a==[3 8 9 10]
   b = a(1:3);
end
```

c)
```
if a>3
   clear a;
elseif a<0
   save a;
else
   disp('Valor de a no considerado');
end
```

Naturalmente, `elseif` y `else` son opcionales. En vez de las expresiones condicionales indicadas en el ejemplo pueden aparecer cualquier función que dé valores `1` (verdadero) ó `0` (falso).

`while`

```
while s
    comandos
end
```

Mientras `s` es `1`, se ejecutan los `comandos` entre `while` y `end`. `s` puede ser cualquier expresión que dé por resultado `1` (verdadero) ó `0` (falso).

`break`

Interrumpe ejecución de ciclos `for` o `while`. En *loops* anidados, `break` sale del más interno solamente.

Funciones lógicas

Además de expresiones construidas con los operadores relacionales `==`, `<=`, etc., y los operadores lógicos `&`, `|` y `~`, los comandos de control de flujo anteriores admiten cualquier función cuyo resultado sea `1` (verdadero) ó `0` (falso). Particularmente útiles son funciones como las siguientes:

`all(a)`	`1` si todos los elementos de `a` son no nulos, y `0` si alguno es cero
`any(a)`	`1` si alguno de los elementos de `a` es no nulo
`isempty(a)`	`1` si `a` es matriz vacía (`a=[]`)

Otras funciones entregan matrices de la misma dimensión que el argumento, con unos o ceros en los lugares en que la condición es verdadera o falsa, respectivamente:

`finite(a)`	`1` donde `a` es finito (no `inf` ni `NaN`)
`isinf(a)`	`1` donde `a` es infinito
`isnan(a)`	`1` donde `a` es un `NaN`

Por ejemplo, luego de ejecutar las líneas

```
x = [-2 -1 0 1 2];
y = 1./x;
a = finite(y);
b = isinf(y);
c = isnan(y);
```

se tiene

```
a = [1 1 0 1 1]
b = [0 0 1 0 0]
c = [0 0 0 0 0]
```

Otra función lógica muy importante es `find`:

`find(a)`	Encuentra los índices de los elementos no nulos de `a`.

Por ejemplo, si ejecutamos las líneas

```
x=[11 0 33 0 55];
z1=find(x);
z2=find(x>0 & x<40);
```

obtendremos

```
z1 = [1 3 5]
z2 = [1 3]
```

`find` también puede dar dos resultados de salida simultáneamente (más sobre esta posibilidad en la sección 4.6.2), en cuyo caso el resultado son los pares de índices (índices de fila y columna) para cada elemento no nulo de una matriz

```
y=[1 2 3 4 5;6 7 8 9 10];
[z3,z4]=find(y>8);
```

da como resultado

```
z3 = [2;2];
z4 = [4;5];
```

`z3` contiene los índice de fila y `z4` los de columna para los elementos no nulos de la matriz `y>8`. Esto permite construir, por ejemplo, la matriz `z5=[z3 z4]` $= \begin{pmatrix} 2 & 4 \\ 2 & 5 \end{pmatrix}$, en la cual cada fila es la posición de `y` tal que la condición `y>8` es verdadera (en este caso, es verdadera para los elementos `y(2,4)` e `y(2,5)`).

Funciones definidas por el usuario

Octave/Matlab puede ser fácilmente extendido por el usuario definiendo nuevas funciones que le acomoden a sus propósitos. Esto se hace a través del comando `function`.

Podemos definir (en modo interactivo o dentro de un *script*), una función en la forma

```
function nombre (argumentos)
   comandos
endfunction
```

`argumentos` es una lista de argumentos separados por comas, y `comandos` es la sucesión de comandos que serán ejecutados al llamar a `nombre`. La lista de argumentos es opcional, en cuyo caso los paréntesis redondos se pueden omitir.

A mediano y largo plazo, puede ser mucho más conveniente definir las funciones en archivos especiales, listos para ser llamados en el futuro desde modo interactivo o desde cualquier *script*. Esto se hace escribiendo la definición de una función en un *script* con extensión `.m`. Cuando Octave/Matlab debe ejecutar un comando o función que no conoce, por ejemplo, `suma(x,y)`,busca en los archivos accesibles en su path de búsqueda un archivo llamado `suma.m`, lo carga y ejecuta la definición contenida en ese archivo.

Por ejemplo, si escribimos en el *script* `suma.m` las líneas

```
function s=suma(x,y)
s = x+y;
```

el resultado de `suma(2,3)` será `5`.

Las funciones así definidas pueden entregar más de un argumento si es necesario (ya hemos visto algunos ejemplos con `find` y `size`). Por ejemplo, definimos una función que efectúe un análisis estadístico básico en `stat.m`:

```
function [mean,stdev] = stat(x)
n = length(x);
mean = sum(x)/n;
stdev = sqrt(sum((x-mean).^2/n));
```

Al llamarla en la forma `[m,s] = stat(x)`, si `x` es un vector fila o columna, en `m` quedará el promedio de los elementos de `x`, y en `s` la desviación estándard.

Todas las variables dentro de un *script* que define una función son locales, a menos que se indique lo contrario con `global`. Por ejemplo, si un *script* `x.m` llama a una función `f`, y dentro de `f.m` se usa una variable `a` que queremos sea global, ella se debe declarar en la forma `global a` tanto en `f.m` como en el *script* que la llamó, `x.m`, y en todo otro *script* que pretenda usar esa variable global.

4.6.3 Matrices y variables elementales

Matrices constantes importantes

Las siguientes son matrices que se emplean habitualmente en distintos contextos, y que es útil tener muy presente:

`eye(n)`	Matriz identidad de $n \times n$
`ones(m,n)`	Matriz de $m \times n$, con todos los elementos igual a 1.
`rand(m,n)`	Matriz de $m \times n$ de números al azar, distribuidos uniformemente.
`randn(m,n)`	Igual que `rand`, pero con distribución normal (Gaussiana).
`zeros(m,n)`	Igual que `ones`, pero con todos los elementos 0.

Matrices útiles para construir ejes o mallas para graficar

`v = linspace(min,max,n)`	Vector cuyo primer elemento es `min`, su último elemento es `max`, y tiene `n` elementos equiespaciados.
`v = logspace(min,max,n)`	Análogo a `linspace`, pero los `n` elementos están espaciados logarítmicamente.
`[X,Y] = meshgrid(x,y)`	Construye una malla del plano x-y. Las filas de `X` son copias del vector `x`, y las columnas de `Y` son copias del vector `y`.

Por ejemplo:

```
x = [1 2 3];
y = [4 5];
[X,Y] = meshgrid(x,y);
```

da

$$X = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} , \qquad Y = \begin{pmatrix} 4 & 4 & 4 \\ 5 & 5 & 5 \end{pmatrix} .$$

Notemos que al tomar sucesivamente los pares ordenados (`X(1,1),Y(1,1)`), (`X(1,2),Y(1,2)`), (`X(1,3),Y(1,3)`), etc., se obtienen todos los pares ordenados posibles tales que el primer elemento está en `x` y el segundo está en `y`. Esta característica hace particularmente útil el comando `meshgrid` en el contexto de gráficos de funciones de dos variables (ver secciones 4.6.7, 4.6.7).

Constantes especiales

Octave/Matlab proporciona algunos números especiales, algunos de los cuales ya mencionamos en la sección 4.3.1.

`i`, `j`	Unidad imaginaria ($\sqrt{-1}$)
`inf`	Infinito
`NaN`	Not-A-Number
`pi`	El número π ($= 3.1415926535897\ldots$)

Funciones elementales

Desde luego, Octave/Matlab proporciona todas las funciones matemáticas básicas. Por ejemplo:

a) Funciones sobre números reales/complejos

`abs`	Valor absoluto de números reales, o módulo de números imaginarios
`angle`	Ángulo de fase de un número imaginario
`conj`	Complejo conjugado
`real`	Parte real
`imag`	Parte imaginaria
`sign`	Signo
`sqrt`	Raíz cuadrada

b) Exponencial y funciones asociadas

`cos`, `sin`, etc.	Funciones trigonométricas
`cosh`, `sinh`, etc.	Funciones hiperbólicas
`exp`	Exponencial
`log`	Logaritmo

c) Redondeo

`ceil`	Redondear hacia $+\infty$
`fix`	Redondear hacia cero
`floor`	Redondear hacia $-\infty$
`round`	Redondear hacia el entero más cercano

Funciones especiales

Además, Octave/Matlab proporciona diversas funciones matemáticas especiales. Algunos ejemplos:

`bessel`	Función de Bessel
`besselh`	Función de Hankel
`beta`	Función beta
`ellipke`	Función elíptica
`erf`	Función error
`gamma`	Función gamma

Así, por ejemplo, `bessel(alpha,X)` evalúa la función de Bessel de orden `alpha`, $J_\alpha(x)$, para cada elemento de la matriz X.

4.6.4 Polinomios

Octave/Matlab representa los polinomios como vectores fila. El polinomio

$$p = c_n x^n + \cdots + c_1 x + c_0$$

es representado en Octave/Matlab en la forma

```
p = [c_n, ..., c1, c0]
```

Podemos efectuar una serie de operaciones con los polinomios así representados.

`poly(x)`	Polinomio cuyas raíces son los elementos de `x`.
`polyval(p,x)`	Evalúa el polinomio `p` en `x` (en los elementos de `x` si éste es un vector)
`roots(p)`	Raíces del polinomio `p`

4.6.5 Álgebra lineal (matrices cuadradas)

Unos pocos ejemplos, entre los comandos de uso más habitual:

`det`	Determinante
`rank`	Número de filas o columnas linealmente independientes
`trace`	Traza
`inv`	Matriz inversa
`eig`	Autovalores y autovectores
`poly`	Polinomio característico

Notar que `poly` es la misma función de la sección 4.6.4 que construye un polinomio de raíces dadas. En el fondo, construir el polinomio característico de una matriz es lo mismo,

y por tanto tiene sentido asignarles la misma función. Y no hay confusión, pues una opera sobre vectores y la otra sobre matrices cuadradas.

El uso de todos estos comandos son autoexplicativos, salvo `eig`, que se puede emplear de dos modos:

```
d = eig(a)
[V,D] = eig(a)
```

La primera forma deja en `d` un vector con los autovalores de `a`. La segunda, deja en `D` una matriz diagonal con los autovalores, y en `V` una matiz cuyas columnas son los autovalores, de modo que `A*V = V*D`. Por ejemplo, si `a =[1 2; 3 4]`, entonces

$$d = \begin{pmatrix} 5.37228 \\ -0.37228 \end{pmatrix}$$

y

$$D = \begin{pmatrix} 5.37228\ldots & 0 \\ 0 & -0.37228\ldots \end{pmatrix}, \qquad V = \begin{pmatrix} 0.41597\ldots & -0.82456\ldots \\ 0.90938\ldots & 0.56577\ldots \end{pmatrix}.$$

La primera columna de `V` es el autovector de `a` asociado al primer autovalor, $5.37228\ldots$.

4.6.6 Análisis de datos y transformada de Fourier

En Octave/Matlab están disponibles diversas herramientas para el análisis de series de datos (estadística, correlaciones, convolución, etc.). Algunas de las operaciones básicas son:

a) Máximos y mínimos
Si `a` es un vector, `max(a)` es el mayor elemento de `a`. Si es una matriz, `max(a)` es un vector fila, que contiene el máximo elemento para cada columna.

```
a = [1 6 7; 2 8 3; 0 4 1]
b = max(a)
```

$b = (2 \quad 8 \quad 7)$

Se sigue que el mayor elemento de la matriz se obtiene con `max(max(a))`.

`min` opera de modo análogo, entregando los mínimos.

b) Estadística básica
Las siguientes funciones, como `min` y `max`, operan sobre vectores del modo usual, y sobre matrices entregando vectores fila, con cada elemento representando a cada columna de la matriz.

`mean`	Valor promedio
`median`	Mediana
`std`	Desviación standard
`prod`	Producto de los elementos
`sum`	Suma de los elementos

c) Orden
`sort(a)` ordena los elementos de `a` en orden ascendente si `a` es un vector. Si es una matriz, ordena cada columna.

```
b = sort([1 3 9; 8 2 1; 4 -3 0]);
```
$$b = \begin{pmatrix} 1 & -3 & 0 \\ 4 & 2 & 1 \\ 8 & 3 & 9 \end{pmatrix}$$

d) Transformada de Fourier

Por último, es posible efectuar transformadas de Fourier directas e inversas, en una o dos dimensiones. Por ejemplo, `fft` y `ifft` dan la transformada de Fourier y la transformada inversa de `x`, usando un algoritmo de *fast Fourier transform* (FFT). Específicamente, si `X=fft(x)` y `x=ifft(X)`, y los vectores son de largo `N`:

$$X(k) = \sum_{j=1}^{N} x(j)\omega_N^{(j-1)(k-1)} \ ,$$
$$x(j) = \frac{1}{N}\sum_{k=1}^{N} X(k)\omega_N^{-(j-1)(k-1)} \ ,$$

donde $\omega_N = e^{-2\pi i/N}$.

4.6.7 Gráficos

Una de las características más importantes de Matlab son sus amplias posibilidades gráficas. Algunas de esas características se encuentran también en Octave. En esta sección revisaremos el caso de gráficos en dos dimensiones, en la siguiente el caso de tres dimensiones, y luego examinaremos algunas posibilidades de manipulación de gráficos.

Gráficos bidimensionales

Para graficar en dos dimensiones se usa el comando `plot`. `plot(x,y)` grafica la ordenada `y` versus la abscisa `x`. `plot(y)` asume abscisa `[1,2,...n]`, donde `n` es la longitud de `y`.

Ejemplo: Si `x=[2 8 9]`, `y=[6 3 2]`, entonces
`plot(x,y)`

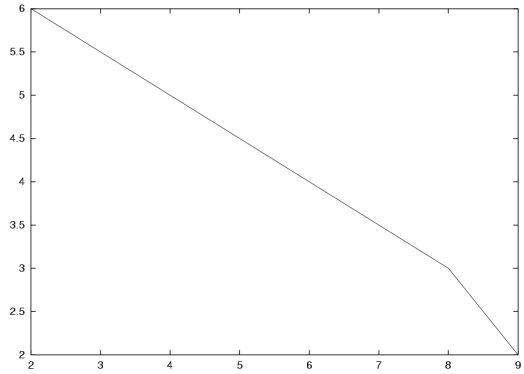


Figura 4.1: Gráfico simple.

Por *default*, Octave utiliza `gnuplot` para los gráficos. Por *default*, los puntos se conectan con una línea roja en este caso. El aspecto de la línea o de los puntos puede ser modificado. Por ejemplo, `plot(x,y,'ob')` hace que los puntos sean indicados con círculos (`'o'`) azules (`'b'`, *blue*). Otros modificadores posibles son:

`-`	línea (*default*)	`r`	red
`.`	puntos	`g`	green
`@`	otro estilo de puntos	`b`	blue
`+`	signo más	`m`	magenta
`*`	asteriscos	`c`	cyan
`o`	círculos	`w`	white
`x`	cruces		

Dos o más gráficos se pueden incluir en el mismo output agregando más argumentos a `plot`. Por ejemplo: `plot(x1,y1,'x',x2,y2,'og',x3,y3,'.c')`.

Los mapas de contorno son un tipo especial de gráfico. Dada una función $z = f(x, y)$, nos interesa graficar los puntos (x, y) tales que $f = c$, con c alguna constante. Por ejemplo, consideremos

$$z = xe^{-x^2-y^2} \ , \quad x \in [-2, 2], \ y \in [-2, 3] \ .$$

Para obtener el gráfico de contorno de z, mostrando los niveles $z = -.3$, $z = -.1$, $z = 0$, $z = .1$ y $z = .3$, podemos usar las instrucciones:

```
x = -2:.2:2;
y = -2:.2:3;
[X,Y] = meshgrid(x,y);
Z = X.*exp(-X.^2-Y.^2);
contour(Z.',[-.3 -.1 0 .1 .3],x,y);   # Octave por default (gnuplot)
contour(x, y, Z.',[-.3 -.1 0 .1 .3]); # Octave con plplot y Matlab
```

Las dos primeras líneas definen los puntos sobre los ejes x e y en los cuales la función será evaluada. En este caso, escojimos una grilla en que puntos contiguos están separados por `.2`. Para un mapa de contorno, necesitamos evaluar la función en todos los pares ordenados (x, y) posibles al escoger x en `x` e y en `y`. Para eso usamos `meshgrid` (introducida sin mayores explicaciones en la sección 4.6.3). Luego evaluamos la función [`Z` es una matriz, donde cada elemento es el valor de la función en un par ordenado (x, y)], y finalmente construimos el mapa de contorno para los niveles deseados.

Gráficos tridimensionales

También es posible realizar gráficos tridimensionales. Por ejemplo, la misma doble gaussiana de la sección anterior se puede graficar en tres dimensiones, para mostrarla como una superficie $z(x, y)$. Basta reemplazar la última instrucción, que llama a `contour`, por la siguiente:

```
mesh(X,Y,Z)
```

Observar que, mientras `contour` acepta argumentos dos de los cuales son vectores, y el tercero una matriz, en `mesh` los tres argumentos son matrices de la misma dimensión (usamos `X`, `Y`, en vez de `x`, `y`).

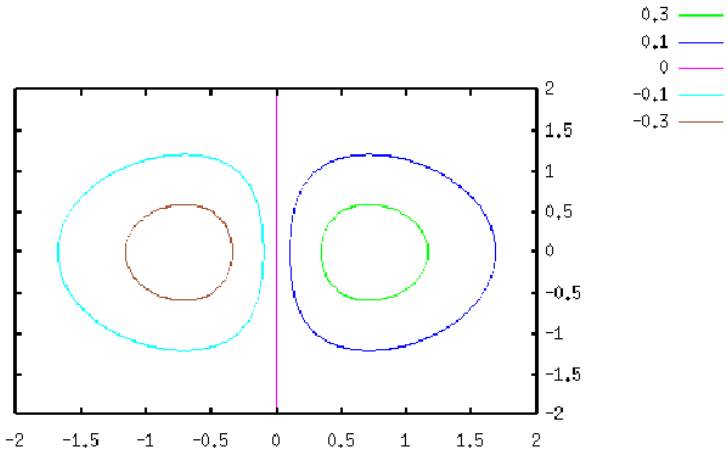


Figura 4.2: Curvas de contorno.

Nota importante: Otro modo de hacer gráficos bi y tridimensionales es con `gplot` y `gsplot` (instrucciones asociadas realmente no a Octave sino a `gnuplot`, y por tanto no equivalentes a instrucciones en Matlab). Se recomienda consultar la documentación de Octave para los detalles.

Manipulación de gráficos

Los siguientes comandos están disponibles para modificar gráficos construidos con Octave/Matlab:

a) Ejes

`axis([x1 y1 x2 y2])` Cambia el eje x al rango $(x1, x2)$, y el eje y al rango $(y1, y2)$.

b) Títulos

`title(s)`	Título (`s` es un string)
`xlabel(s)`	Título del eje x, y, z.
`ylabel(s)`	
`zlabel(s)`	

c) Grillas

`grid` Incluye o borra una grilla de referencia en un gráfico bidimensional. `grid ''on''` coloca la grilla y `grid ''off''` la saca. `grid` equivale a `grid ''on''`.

Al usar `gnuplot`, el gráfico mostrado en pantalla no es actualizado automáticamente. Para actualizarlo y ver las modificaciones efectuadas, hay que dar la instrucción `replot`.

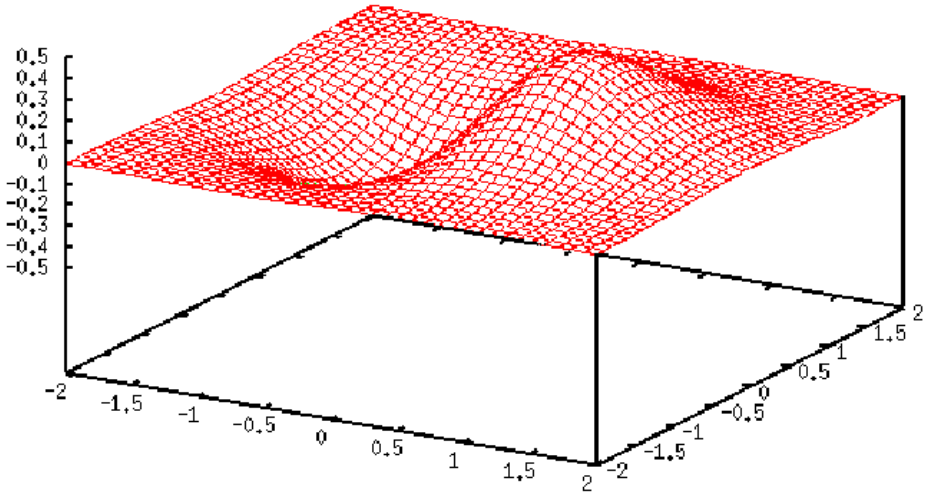


Figura 4.3: Curvas de contorno.

Los siguientes comandos permiten manipular las ventanas gráficas:

`hold`	Permite "congelar" la figura actual, de modo que sucesivos comandos gráficos se superponen sobre dicha figura (normalmente la figura anterior es reemplazada por la nueva). `hold on` activa este "congelamiento", y `hold off` lo desactiva. `hold` cambia alternativamente entre el estado `on` y `off`.
`closeplot`	Cierra la ventana actual.

Finalmente, si se desea guardar un gráfico en un archivo, se puede proceder del siguiente modo si Octave está generando los gráficos con `gnuplot` y se trabaja en un terminal con XWindows. Si se desea guardar un gráfico de la función $y = x^3$, por ejemplo:

```
x = linspace(1,10,30);
y = x.^3;
plot(x,y);
gset term postscript color
gset output ‘‘xcubo.ps’’
replot
gset term x11
```

Las tres primeras líneas son los comandos de Octave/Matlab convencionales para graficar. Luego se resetea el terminal a un terminal postscript en colores (`gset term postscript` si no deseamos los colores), para que el output sucesivo vaya en formato postscript y no a la pantalla. La siguiente línea indica que la salida es al archivo `xcubo.ps`. Finalmente, se redibuja el gráfico (con lo cual el archivo `xcubo.ps` es realmente generado), y se vuelve al terminal XWindows para continuar trabajando con salida a la pantalla.

Debemos hacer notar que no necesariamente el gráfico exportado a *Postscript* se verá igual al resultado que `gnuplot` muestra en pantalla. Durante la preparación de este manual,

nos dimos cuenta de ello al intentar cambiar los estilos de línea de `plot`. Queda entonces advertido el lector.

4.6.8 Strings

Para manipular una cadena de texto, disponemos de los siguientes comandos:

`lower`	Convierte a minúsculas
`upper`	Convierte a mayúsculas

Así, `lower('Texto')` da `'texto'`, y `upper('Texto')` da `'TEXTO'`.

Para comparar dos matrices entre sí, usamos `strcmp`:

`strcmp(a,b)`	1 si `a` y `b` son idénticas, 0 en caso contrario

Podemos convertir números enteros o reales en strings, y strings en números, con los comandos:

`int2str`	Convierte entero en string
`num2str`	Convierte número en string
`str2num`	Convierte string en número

Por ejemplo, podemos usar esto para construir un título para un gráfico:

```
s = ['Intensidad transmitida vs. frecuencia, n = ', num2str(1.5)];
title(s);
```

Esto pondrá un título en el gráfico con el texto:
`Intensidad transmitida vs. frecuencia, n = 1.5`.

4.6.9 Manejo de archivos

Ocasionalmente nos interesará grabar el resultado de nuestros cálculos en archivos, o utilizar datos de archivos para nuevos cálculos. El primer paso es abrir un archivo:

```
archivo = fopen('archivo.dat','w');
```

Esto abre el archivo `archivo.dat` para escritura (`'w'`), y le asigna a este archivo un número que queda alojado en la variable `archivo` para futura referencia.

Los modos de apertura posibles son:

`r`	Abre para lectura
`w`	Abre para escritura, descartando contenidos anteriores si los hay
`a`	Abre o crea archivo para escritura, agregando datos al final del archivo si ya existe
`r+`	Abre para lectura y escritura
`w+`	Crea archivo para lectura y escritura
`a+`	Abre o crea archivo para lectura y escritura, agregando datos al final del archivo si ya existe

En un archivo se puede escribir en modo binario:

`fread`	Lee datos binarios
`fwrite`	Escribe datos binarios

o en modo texto

`fgetl`	Lee una línea del archivo, descarta cambio de línea
`fgets`	Lee una línea del archivo, preserva cambio de l'ınea
`fprintf`	Escribe datos siguiendo un formato
`fscanf`	Lee datos siguiendo un formato

Referimos al lector a la ayuda que proporciona Octave/Matlab para interiorizarse del uso de estos comandos. Sólo expondremos el uso de `fprintf`, pues el formato es algo que habitualmente se necesita tanto para escribir en archivos como en pantalla, y `fprintf` se puede usar en ambos casos.

La instrucción

```
fprintf(archivo,'formato',A,B,...)
```

imprime en el archivo asociado con el identificador `archivo` (asociado al mismo al usar `fopen`, ver más arriba), las variables `A`, `B`, etc., usando el formato `'formato'`. `archivo=1` corresponde a la pantalla; si `archivo` se omite, el *default* es `1`, es decir, `fprintf` imprime en pantalla si `archivo=1` o si se omite el primer argumento.

`'formato'` es una string, que puede contener caracters normales, caracteres de escape o especificadores de conversión. Los caracteres de escape son:

`\n`	New line
`\t`	Horizontal tab
`\b`	Backspace
`\r`	Carriage return
`\f`	Form feed
`\\`	Backslash
`\'`	Single quote

Por ejemplo, la línea

```
fprintf('Una tabulacion\t y un {\'o}riginal\'\' cambio de linea\n aqui\n')
```

da como resultado

```
Una tabulacion       y un ''original'' cambio de linea
aqui
```

Es importante notar que por *default*, el cambio de línea al final de un `fprintf` no existe, de modo que, si queremos evitar salidas a pantalla o a archivo poco estéticas, siempre hay que terminar con un `\n`.

Los especificadores de conversión permiten dar formato adecuado a las variables numéricas `A`, `B`, etc. que se desean imprimir. Constan del caracter `%`, seguido de indicadores de ancho (opcionales), y caracteres de conversión. Por ejemplo, si deseamos imprimir el número π con 5 decimales, la instrucción es:

```
fprintf('Numero pi = %.5f\n',pi)
```

El resultado:

```
Numero pi = 3.14159
```

Los caracteres de conversión pueden ser

`%e`	Notación exponencial (Ej.: `2.4e-5`)
`%f`	Notación con punto decimal fijo (Ej.: `0.000024`)
`%g`	`%e` o `%f`, dependiendo de cuál sea más corto (los ceros no significativos no se imprimen)

Entre `%` y `e`, `f`, o `g` según corresponda, se pueden agregar uno o más de los siguientes caracteres, en este orden:

- Un signo menos (`-`), para especificar alineamiento a la izquierda (a la derecha es el *default*).
- Un número entero para especificar un ancho mínimo del campo.
- Un punto para separar el número anterior del siguiente número.
- Un número indicando la precisión (número de dígitos a la derecha del punto decimal).

En el siguiente ejemplo veremos distintos casos posibles. El output fue generado con las siguientes instrucciones, contenidas en un *script*:

```
a = .04395;
fprintf('123456789012345\n');
fprintf('a = %.3f.\n',a);
fprintf('a = %10.2f.\n',a);
fprintf('a = %-10.2f.\n',a);
fprintf('a = %4f.\n',a);
fprintf('a = %5.3e.\n',a);
fprintf('a = %f.\n',a);
fprintf('a = %e.\n',a);
fprintf('a = %g.\n',a);
```

El resultado:

```
12345678901234567890
a = 0.044.
a =       0.04.
a = 0.04      .
a = 0.043950.
a = 4.395e-02.
a = 0.043950.
a = 4.395000e-02.
a = 0.04395.
```

En la primera línea, se imprimen tres decimales. En la segunda, dos, pero el ancho mínimo es 10 caracteres, de modo que se alínea a la derecha el output y se completa con blancos. En la tercera línea es lo mismo, pero alineado a la izquierda. En la cuarta línea se ha especificado un ancho mínimo de 4 caracteres; como el tamaño del número es mayor, esto no tiene efecto y se imprime el número completo. En la quinta línea se usa notación exponencial, con tres decimal (nuevamente, el ancho mínimo especificado, 5, es menor que el ancho del output, luego no tiene efecto). Las últimas tres líneas comparan el output de `%f`, `%e` y `%g`, sin otras especificaciones.

Si se desean imprimir más de un número, basta agregar las conversiones adecuadas y los argumentos en `fprintf`. Así, la línea

```
fprintf('Dos numeros arbitrarios: %g y %g.\n',pi,exp(4));
```

da por resultado

```
Dos numeros arbitrarios: 3.14159 y 54.5982.
```

Si los argumentos numéricos de `fprintf` son matrices, el formato es aplicado a cada columna hasta terminar la matriz. Por ejemplo, el *script*

```
x = 1:5;
y1 = exp(x);
y2 = log(x);
a = [x; y1; y2];
fprintf = ('%g  %8g %8.3f\n',a);
```

da el output

```
1   2.71828    0.000
2   7.38906    0.693
3   20.0855    1.099
4   54.5982    1.386
5   148.413    1.609
```

Capítulo 5

El sistema de preparación de documentos TEX .

versión final 3.5-19 de agosto de 2002

5.1 Introducción.

TEX es un procesador de texto o, mejor dicho, un avanzado sistema de preparación de documentos, creado por Donald Knuth, que permite el diseño de documentos de gran calidad, conteniendo textos y fórmulas matemáticas. Años después, LATEX fue desarrollado por Leslie Lamport, facilitando la preparación de documentos en TEX, gracias a la definición de "macros" o conjuntos de comandos de fácil uso.

LATEX tuvo diversas versiones hasta la 2.09. Actualmente, LATEX ha recibido importantes modificaciones, siendo la distribución actualmente en uso y desarrollo LATEX 2_ε, una versión transitoria en espera de que algún día se llegue a la nueva versión definitiva de LATEX, LATEX3. En estas páginas cuando digamos LATEX nos referiremos a la versión actual, LATEX 2_ε. Cuando queramos hacer referencia a la versión anterior, que debería quedar progresivamente en desuso, diremos explícitamente LATEX 2.09.

5.2 Archivos.

El proceso de preparación de un documento LATEX consta de tres pasos:

1. Creación de un archivo con extensión `tex` con algún editor.

2. Compilación del archivo `tex`, con un comando del tipo `latex <archivo>.tex` o `latex <archivo>`. Esto da por resultado tres archivos adicionales, con el mismo nombre del archivo original, pero con extensiones distintas:

   (a) `dvi`. Es el archivo procesado que podemos ver en pantalla o imprimir. Una vez compilado, este archivo puede ser enviado a otro computador, para imprimir en otra impresora, o verlo en otro monitor, independiente de la máquina (de donde su extensión `dvi`, *device independent*).

(b) `log`. Aquí se encuentran todos los mensajes producto de la compilación, para consulta si es necesario (errores encontrados, memoria utilizada, mensajes de advertencia, etc.).

(c) `aux`. Contiene información adicional que por el momento no nos interesa.

3. Visión en pantalla e impresión del archivo procesado a través de un programa anexo (`xdvi` o `dvips`, por ejemplo), capaz de leer el `dvi`.

5.3 Input básico.

5.3.1 Estructura de un archivo.

En un archivo no pueden faltar las siguientes líneas:

```
\documentclass[12pt]{article}

\begin{document}

\end{document}
```

Haremos algunas precisiones respecto a la primera línea más tarde. Lo importante es que una línea de esta forma debe ser la primera de nuestro archivo. Todo lo que se encuentra antes de `\begin{document}` se denomina *preámbulo*. El texto que queramos escribir va entre `\begin{document}` y `\end{document}`. Todo lo que se encuentre después de `\end{document}` es ignorado.

5.3.2 Caracteres.

Pueden aparecer en nuestro texto todos los caracteres del código ASCII no extendido (teclado inglés usual): letras, números y los signos de puntuación:

```
.  :  ;  ,  ?  !  ‘  ’  (  )  [  ]  -  /  * @
```

Los caracteres especiales:

```
#  $  %  &  ~  _  ^  \  {  }
```

tienen un significado específico para LaTeX. Algunos de ellos se pueden obtener anteponiéndoles un *backslash*:

\# $ \$ % \% & \& { \{ } \}

Los caracteres

```
+  =  |  <  >
```

generalmente aparecen en fórmulas matemáticas, aunque pueden aparecer en texto normal. Finalmente, las comillas dobles (") casi nunca se usan.

Los espacios en blanco y el fin de línea son también caracteres (invisibles), que LaTeX considera como un mismo carácter, que llamaremos espacio, y que simbolizaremos ocasionalmente como ␣ .

Para escribir en castellano requeriremos además algunos signos y caracteres especiales:

ñ `\~n` á `\'a` í `\'{\i}` ü `\"u` ¡ `!‘` ¿ `?‘`

5.3.3 Comandos.

Todos los comandos comienzan con un backslash, y se extienden hasta encontrar el primer carácter que no sea una letra (es decir, un espacio, un número, un signo de puntuación o matemático, etc.).

5.3.4 Algunos conceptos de estilo.

LaTeX es consciente de muchas convenciones estilísticas que quizás no apreciamos cuando leemos textos bien diseñados, pero las cuales es bueno conocer para aprovecharlas.

a) Observemos la siguiente palabra: fino. Esta palabra fue generada escribiendo simplemente `fino`, pero observemos que las letras ‘f’ e ‘i’ no están separadas, sino que unidas artísticamente. Esto es una *ligadura*, y es considerada una práctica estéticamente preferible. LaTeX sabe esto e inserta este pequeño efecto tipográfico sin que nos demos cuenta.

b) Las comillas de apertura y de cierre son distintas. Por ejemplo: ‘insigne’ (comillas simples) o “insigne” (comillas dobles). Las comillas de apertura se hacen con uno o con dos acentos graves (`‘`), para comillas simples o dobles, respectivamente, y las de cierre con acentos agudos (`’`): `‘insigne’`, `‘‘insigne’’`. No es correcto entonces utilizar las comillas dobles del teclado e intentar escribir `"insigne"` (el resultado de esto es el poco estético ”insigne”).

c) Existen tres tipos de guiones:

Corto	Saint-Exupéry	`-`	(entre palabras, corte en sílabas al final de la línea)
Medio	páginas 1–2	`--`	(rango de números)
Largo	un ejemplo —como éste	`---`	(puntuación, paréntesis)

d) LaTeX inserta después de un punto seguido un pequeño espacio adicional respecto al espacio normal entre palabras, para separar sutilmente frases. Pero, ¿cómo saber que un punto termina una frase? El criterio que utiliza es que todo punto termina una frase cuando va precedido de una minúscula. Esto es cierto en la mayoría de los casos, así como es cierto que generalmente cuando un punto viene después de una mayúscula no hay fin de frase:

> China y U.R.S.S. estuvieron de acuerdo. Sin embargo ...

Pero hay excepciones:

> En la pág. 11 encontraremos noticias desde la U.R.S.S. Éstas fueron entregadas ...

Cuando estas excepciones se producen, nosotros, humanos, tenemos que ayudarle al computador, diciéndole que, aunque hay un punto después de la "g", no hay un fin de frase, y que el punto después de la última "S" sí termina frase. Esto se consigue así:

```
En la p\'ag.\ 11 encontraremos noticias desde la
U.R.S.S\@. \'Estas fueron entregadas...
```

d) Énfasis de texto:

Éste es un texto *enfatizado*.	`\'Este es un texto {\em enfatizado}.`
Otro texto *enfatizado*.	`Otro texto \emph{enfatizado}.`

Al enfatizar, pasamos temporalmente a un tipo de letra distinto, la *itálica*. Esta letra es ligeramente inclinada hacia adelante, lo cual puede afectar el correcto espaciado entre palabras. Comparemos, por ejemplo:

Quiero *hoy* mi recompensa.	`Quiero {\em hoy} mi recompensa.`
Quiero *hoy* mi recompensa.	`Quiero {\em hoy\/} mi recompensa.`
Quiero *hoy* mi recompensa.	`Quiero \emph{hoy} mi recompensa.`

La segunda y tercera frase tienen un pequeño espacio adicional después de "hoy", para compensar el espacio entre palabras perdido por la inclinación de la itálica. Este pequeño espacio se denomina *corrección itálica*, y se consigue usando `\emph`, o, si se usa `\em`, agregando `\/` antes de cerrar el paréntesis cursivo. La corrección itálica es innecesaria cuando después del texto enfatizado viene un punto o una coma. LaTeX advierte esto y omite el espacio adicional aunque uno lo haya sugerido.

5.3.5 Notas a pie de página.

```
Insertemos una nota a pie de p\'agina.\footnote{Como \'esta.}
```

LaTeX colocará una nota a pie de página[1] en el lugar apropiado.

[1] Como ésta.

5.3.6 Fórmulas matemáticas.

LaTeX distingue dos modos de escritura: un modo de texto, en el cual se escriben los textos usuales como los ya mencionados, y un modo matemático, dentro del cual se escriben las fórmulas. Cualquier fórmula *debe* ser escrita dentro de un modo matemático, y si algún símbolo matemático aparece fuera del modo matemático el compilador acusará un error.

Hay tres formas principales para acceder al modo matemático:

a) `$x+y=3$`

b) `$$xy=8$$`

c)
```
\begin{equation}
x/y=5
\end{equation}
```

Estas tres opciones generan, respectivamente, una ecuación en el texto: $x + y = 3$, una ecuación separada del texto, centrada en la página:

$$xy = 8$$

y una ecuación separada del texto, numerada:

$$x/y = 5 \tag{5.1}$$

Es importante notar que al referirnos a una variable matemática en el texto debemos escribirla en modo matemático:

Decir que la incógnita es x es incorrecto. No: la incógnita es x.

```
Decir que la inc{\'o}gnita es
x es incorrecto. No: la
inc{\'o}gnita es $x$.
```

5.3.7 Comentarios.

Uno puede hacer que el compilador ignore parte del archivo usando `%`. Todo el texto desde este carácter hasta el fin de la línea correspondiente será ignorado (incluyendo el fin de línea).

Un pequeño comentario.

```
Un peque{\~n}o co%  Texto ignorado
mentario.
```

5.3.8 Estilo del documento.

Las características generales del documento están definidas en el preámbulo. Lo más importante es la elección del *estilo*, que determina una serie de parámetros que al usuario normal pueden no importarle, pero que son básicas para una correcta presentación del texto: ¿Qué márgenes dejar en la página? ¿Cuánto dejar de sangría? ¿Tipo de letra? ¿Distancia entre líneas? ¿Dónde poner los números de página? Y un largo etcétera.

Todas estas decisiones se encuentran en un *archivo de estilo* (extensión `cls`). Los archivos standard son: `article`, `report`, `book` y `letter`, cada uno adecuado para escribir artículos cortos (sin capítulos) o más largos (con capítulos), libros y cartas, respectivamente.

La elección del estilo global se hace en la primera línea del archivo:[2]

```
\documentclass{article}
```

Esta línea será aceptada por el compilador, pero nos entregará un documento con un tamaño de letra pequeño, técnicamente llamado de 10 puntos ó 10pt (1pt = 1/72 pulgadas). Existen tres tamaños de letra disponibles: 10, 11 y 12 pt. Si queremos un tamaño de letra más grande, como el que tenemos en este documento, se lo debemos indicar en la primera línea del archivo:

```
\documentclass[12pt]{article}
```

Todas las decisiones de estilo contenidas dentro del archivo `cls` son modificables, existiendo tres modos de hacerlo:

a) Modificando el archivo `cls` directamente. Esto es poco recomendable, porque dicha modificación (por ejemplo, un cambio de los márgenes) se haría extensible a todos los archivos compilados en nuestro computador, y esto puede no ser agradable, ya sea que nosotros seamos los únicos usuarios o debamos compartirlo. Por supuesto, podemos deshacer los cambios cuando terminemos de trabajar, pero esto es tedioso.

b) Introduciendo comandos adecuados en el preámbulo. Ésta es la opción más recomendable y la más usada. Nos permite dominar decisiones específicas de estilo válidas sólo para el archivo que nos interesa.

c) Creando un nuevo archivo `cls`. Esto es muy recomendable cuando las modificaciones de estilo son abundantes, profundas y deseen ser reaprovechadas. Se requiere un poco de experiencia en LaTeX para hacerlo, pero a veces puede ser la única solución razonable.

En todo caso, la opción a usar en la gran mayoría de los casos es la b) (Sec. 5.9).

5.3.9 Argumentos de comandos.

Hemos visto ya algunos comandos que requieren argumentos. Por ejemplo: `\begin{equation}`, `\documentclass[12pt]{article}`, `\footnote{Nota}`. Existen dos tipos de argumentos:

1. **Argumentos obligatorios.** Van encerrados en paréntesis cursivos: `\footnote{Nota}`, por ejemplo. Es obligatorio que después de estos comandos aparezcan los paréntesis. A veces es posible dejar el interior de los paréntesis vacío, pero en otros casos el compilador reclamará incluso eso (`\footnote{}` no genera problemas, pero `\documentclass{}` sí es un gran problema).

   Una propiedad muy general de los comandos de LaTeX es que las llaves de los argumentos obligatorios se pueden omitir cuando dichos argumentos tienen sólo un carácter. Por ejemplo, `\~n` es equivalente a `\~{n}`. Esto permite escribir más fácilmente muchas expresiones, particularmente matemáticas, como veremos más adelante.

[2] En LaTeX 2.09 esta primera línea debe ser `\documentstyle[12pt]article`, y el archivo de estilo tiene extensión `sty`. Intentar compilar con LaTeX 2.09 un archivo que comienza con `\documentclass` da un error. Por el contrario, la compilación con LaTeX 2ε de un archivo que comienza con `\documentstyle` no genera un error, y LaTeX entra en un *modo de compatibilidad*. Sin embargo, interesantes novedades de LaTeX 2ε respecto a LaTeX 2.09 se pierden.

2. **Argumentos opcionales.** Van encerrados en paréntesis cuadrados. Estos argumentos son omitibles, `\documentclass[12pt]...`. Ya dijimos que `\documentclass{article}` es aceptable, y que genera un tamaño de letra de 10pt. Un argumento en paréntesis cuadrados es una opción que modifica la decisión default del compilador (en este caso, lo obliga a usar 12pt en vez de sus instintivos 10pt).

5.3.10 Título.

Un título se genera con:

```
\title{Una breve introducci\'on}
\author{V\'{\i}ctor Mu\~noz}
\date{30 de Junio de 1998}
\maketitle
```

`\title`, `\author` y `\date` pueden ir en cualquier parte (incluyendo el preámbulo) antes de `\maketitle`. `\maketitle` debe estar después de `\begin{document}`. Dependiendo de nuestras necesidades, tenemos las siguientes alternativas:

a) Sin título:

```
\title{}
```

b) Sin autor:

```
\author{}
```

c) Sin fecha:

```
\date{}
```

d) Fecha actual (en inglés): omitir `\date`.

e) Más de un autor:

```
\author{Autor_1 \and Autor_2 \and Autor_3}
```

Para artículos cortos, LaTeX coloca el título en la parte superior de la primera página del texto. Para artículos largos, en una página separada.

5.3.11 Secciones.

Los títulos de las distintas secciones y subsecciones de un documento (numerados adecuadamente, en negrita, como en este texto) se generan con comandos de la forma:

```
\section{Una secci\'on}
\subsection{Una subsecci\'on}
```

Los comandos disponibles son (en orden decreciente de importancia):

`\part`	`\subsection`	`\paragraph`
`\chapter`	`\subsubsection`	`\subparagraph`
`\section`		

Los más usados son `\chapter`, `\section`, `\subsection` y `\subsubsection`. `\chapter` sólo está disponible en los estilos `report` y `book`.

5.3.12 Listas.

Los dos modos usuales de generar listas:

a) Listas numeradas (ambiente `enumerate`):

1. Nivel 1, ítem 1.
2. Nivel 1, ítem 2.
   (a) Nivel 2, ítem 1.
      i. Nivel 3, ítem 1.
3. Nivel 1, ítem 3.

```
\begin{enumerate}
\item Nivel 1, \'{\i}tem 1.
\item Nivel 1, \'{\i}tem 2.
\begin{enumerate}
\item Nivel 2, \'{\i}tem 1.
\begin{enumerate}
\item Nivel 3, \'{\i}tem 1.
\end{enumerate}
\end{enumerate}
\item Nivel 1, \'{\i}tem 3.
\end{enumerate}
```

b) Listas no numeradas (ambiente `itemize`):

- Nivel 1, ítem 1.
- Nivel 1, ítem 2.
  – Nivel 2, ítem 1.
    ∗ Nivel 3, ítem 1.
- Nivel 1, ítem 3.

```
\begin{itemize}
\item Nivel 1, {\'\i}tem 1.
\item Nivel 1, {\'\i}tem 2.
\begin{itemize}
\item Nivel 2, {\'\i}tem 1.
\begin{itemize}
\item Nivel 3, {\'\i}tem 1.
\end{itemize}
\end{itemize}
\item Nivel 1, {\'\i}tem 3.
\end{itemize}
```

Es posible anidar hasta tres niveles de listas. Cada uno usa tipos distintos de rótulos, según el ambiente usado: números arábes, letras y números romanos para `enumerate`, y puntos, guiones y asteriscos para `itemize`. Los rótulos son generados automáticamente por cada `\item`, pero es posible modificarlos agregando un parámetro opcional:

a) Nivel 1, ítem 1.
b) Nivel 1, ítem 2.

```
\begin{enumerate}
\item[a)] Nivel 1, \'{\i}tem 1.
\item[b)] Nivel 1, \'{\i}tem 2.
\end{enumerate}
```

`\item` es lo primero que debe aparecer después de un `\begin{enumerate}` o `\begin{itemize}`.

5.3.13 Tipos de letras.

Fonts.

Los fonts disponibles por default en LaTeX son:

roman	*italic*	Small Caps
boldface	*slanted*	`typewriter`
sans serif		

Los siguientes modos de cambiar fonts son equivalentes:

texto	`{\rm texto}`	`\textrm{texto}`
texto	`{\bf texto}`	`\textbf{texto}`
texto	`{\sf texto}`	`\textsf{texto}`
texto	`{\it texto}`	`\textit{texto}`
texto	`{\sl texto}`	`\textsl{texto}`
Texto	`{\sc Texto}`	`\textsc{texto}`
`texto`	`{\tt texto}`	`\texttt{texto}`

`\rm` es el default para texto normal; `\it` es el default para texto enfatizado; `\bf` es el default para títulos de capítulos, secciones, subsecciones, etc.

`\textrm`, `\textbf`, etc., sólo permiten cambiar porciones definidas del texto, contenido entre los paréntesis cursivos. Con `\rm`, `\bf`, etc. podemos, omitiendo los paréntesis, cambiar el font en todo el texto posterior:

Un cambio local de fonts *y uno global, interminable e infinito* ...

```
Un cambio {\sf local} de fonts
\sl y uno global, interminable
e infinito...
```

También es posible tener combinaciones de estos fonts, por ejemplo, ***bold italic***, pero no sirven los comandos anteriores, sino versiones modificadas de `\rm`, `\bf`, etc.:

```
\rmfamily
\sffamily
\ttfamily
\mdseries
```

```
\bfseries
\upshape
\itshape
\slshape
\scshape
```

Por ejemplo:

texto	{\bfseries\itshape texto}
texto	{\bfseries\upshape texto} (= {\bf texto})
TEXTO	{\ttfamily\scshape texto}
texto	{\sffamily\bfseries texto}
texto	{\sffamily\mdseries texto} (= {\sf texto})

Para entender el uso de estos comandos hay que considerar que un font tiene tres *atributos*: `family` (que distingue entre `rm`, `sf` y `tt`), `series` (que distingue entre `md` y `bf`), y `shape` (que distingue entre `up`, `it`, `sl` y `sc`). Cada uno de los comandos `\rmfamily`, `\bfseries`, etc., cambia sólo uno de estos atributos. Ello permite tener versiones mixtas de los fonts, como un *slanted sans serif*, imposible de obtener usando los comandos `\sl` y `\sf`. Los defaults para el texto usual son: `\rmfamily`, `\mdseries` y `\upshape`.

Tamaño.

Los tamaños de letras disponibles son:

texto	`\tiny`	texto	`\normalsize`	texto	`\LARGE`
texto	`\scriptsize`	texto	`\large`	texto	`\huge`
texto	`\footnotesize`	texto	`\Large`	texto	`\Huge`
texto	`\small`				

Se usan igual que los comandos de cambio de font `\rm`, `\sf`, etc., de la sección 5.3.13.

`\normalsize` es el default para texto normal; `\scriptsize` para sub o supraíndices; `\footnotesize` para notas a pie de página.

5.3.14 Acentos y símbolos.

LaTeX provee diversos tipos de acentos, que se muestran en la Tabla 5.1 (como ejemplo consideramos la letra "o", pero cualquiera es posible, por supuesto). (Hemos usado acá el hecho de que cuando el argumento de un comando consta de un carácter, las llaves son omitibles.)

Otros símbolos especiales y caracteres no ingleses disponibles se encuentran en la Tabla 5.2.

ó	\'o	õ	\~o	ǒ	\v o	o̧	\c o
ò	\‘o	ō	\=o	ő	\H o	ọ	\d o
ô	\^o	ȯ	\. o	o͡o	\t{oo}	o̲	\b o
ö	\"o	ŏ	\u o	o̊	\r o		

Tabla 5.1: Acentos.

†	\dag	œ	\oe	ł	\l
‡	\ddag	Œ	\OE	Ł	\L
§	\S	æ	\ae	ß	\ss
¶	\P	Æ	\AE	SS	\SS
ⓒ	\copyright	å	\aa	¿	?‘
ⓐ	\textcircled a	Å	\AA	¡	!‘
␣	\textvisiblespace	ø	\o		
£	\pounds	Ø	\O		

Tabla 5.2: Símbolos especiales y caracteres no ingleses.

5.3.15 Escritura de textos en castellano.

LaTeX emplea sólo los caracteres ASCII básicos, que no contienen símbolos castellanos como ¿, ¡, ñ, etc. Ya hemos visto que existen comandos que permiten imprimir estos caracteres, y por tanto es posible escribir cualquier texto en castellano (y otros idiomas, de hecho).

Sin embargo, esto no resuelve todo el problema, porque en inglés y castellano las palabras se cortan en "sílabas" de acuerdo a reglas distintas, y esto es relevante cuando se debe cortar el texto en líneas. LaTeX tiene incorporados algoritmos para cortar palabras en inglés y, si se ha hecho una instalación especial de LaTeX en nuestro computador, también en castellano u otros idiomas (a través del programa **babel**, que es parte de la distribución standard de LaTeX 2_ε). En un computador con babel instalado y configurado para cortar en castellano basta incluir el comando `\usepackage[spanish]{babel}` en el preámbulo para poder escribir en castellano cortando las palabras en sílabas correctamente.[3]

Sin embargo, ocasionalmente LaTeX se encuentra con una palabra que no sabe cortar, en cuyo caso no lo intenta y permite que ella se salga del margen derecho del texto, o bien toma decisiones no óptimas. La solución es sugerirle a LaTeX la silabación de la palabra. Por ejemplo, si la palabra conflictiva es `matem\'aticas` (generalmente hay problemas con las palabras acentuadas), entonces basta con reescribirla en la forma: `ma\-te\-m\'a\-ti\-cas`. Con esto, le indicamos a LaTeX en qué puntos es posible cortar la palabra. El comando `\-` no tiene ningún otro efecto, de modo que si la palabra en cuestión no queda al final de la línea, LaTeX por supuesto ignora nuestra sugerencia y no la corta.

Consideremos el siguiente ejemplo:

[3]Esto resuelve también otro problema: los encabezados de capítulos o índices, por ejemplo, son escritos "Capítulo" e "Índice", en vez de "Chapter" e "Index", y cuando se usa el comando `\date`, la fecha aparece en castellano.

Podemos escribir matemáticas. O matemáticas.

```
Podemos escribir matem\'aticas.
O matem\'aticas.
```

Podemos escribir matemáticas. O matemáticas.

```
Podemos escribir
ma\-te\-m\'a\-ti\-cas.
O ma\-te\-m\'a\-ti\-cas.
```

En el primer caso, LaTeX decidió por sí mismo dónde cortar "matemáticas". Como es una palabra acentuada tuvo problemas y no lo hizo muy bien, pues quedó demasiado espacio entre palabras en esa línea. En el segundo párrafo le sugerimos la silabación y LaTeX pudo tomar una decisión más satisfactoria. En el mismo párrafo, la segunda palabra "matemáticas" también tiene sugerencias de corte, pero como no quedó al final de línea no fueron tomadas en cuenta.

5.4 Fórmulas matemáticas.

Hemos mencionado tres formas de ingresar al modo matemático: `$...$` (fórmulas dentro del texto), `$$...$$` (fórmulas separadas del texto, no numeradas) y `\begin{equation} ... \end{equation}` (fórmulas separadas del texto, numeradas). Los comandos que revisaremos en esta sección sólo pueden aparecer dentro del modo matemático.

5.4.1 Sub y supraíndices.

x^{2y} `x^{2y}` x^{y^2} `x^{y^{2}}` (ó `x^{y^2}`) x_1^y `x^y_1` (ó `x_1^y`)
x_{2y} `x_{2y}` x^{y_1} `x^{y_{1}}` (ó `x^{y_1}`)

`\textsuperscript` permite obtener supraíndices fuera del modo matemático:

La 3ª es la vencida.

```
La 3\textsuperscript{a}
es la vencida.
```

5.4.2 Fracciones.

a) Horizontales

$n/2$ `n/2`

b) Verticales

$\frac{1}{2}$ `\frac{1}{2}`, `\frac 1{2}`, `\frac{1}2` ó `\frac 12`

$x = \frac{y + z/2}{y^2+1}$ `x = \frac{y + z/2}{y^2+1}`

$\frac{x+y}{1 + \frac{y}{z+1}}$ `\frac{x+y}{1 + \frac y{z+1}}`

La forma a) es más adecuada y la preferida para fracciones dentro del texto, y la segunda para fórmulas separadas. `\frac` puede aparecer en fórmulas dentro del texto ($\frac{1}{2}$ con `$\frac 12$`), pero esto es inusual y poco recomendable estéticamente, salvo estricta necesidad.

5.4.3 Raíces.

$\sqrt{n}$	`\sqrt{n}` ó `\sqrt n`
$\sqrt{a^2 + b^2}$	`\sqrt{a^2 + b^2}`
$\sqrt[n]{2}$	`\sqrt[n]{2}`

5.4.4 Puntos suspensivos.

a) $\ldots$ `\ldots`

Para fórmulas como

$$a_1 a_2 \ldots a_n \qquad \texttt{a\_1 a\_2 \textbackslash ldots a\_n}$$

b) $\cdots$ `\cdots`

Entre símbolos como $+$, $-$, $=$:

$$x_1 + \cdots + x_n \qquad \texttt{x\_1 + \textbackslash cdots + x\_n}$$

c) $\vdots$ `\vdots`

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

d) $\ddots$ `\ddots`

$$I_{n\times n} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}$$

`\ldots` puede ser usado también en el texto usual:

Arturo quiso salir ... pero se detuvo.	`Arturo quiso salir\ldots pero se detuvo.`

No corresponde usar tres puntos seguidos (...), pues el espaciado entre puntos es incorrecto.

Minúsculas

α	\alpha	θ	\theta	o	o	τ	\tau
β	\beta	ϑ	\vartheta	π	\pi	υ	\upsilon
γ	\gamma	ι	\iota	ϖ	\varpi	ϕ	\phi
δ	\delta	κ	\kappa	ρ	\rho	φ	\varphi
ϵ	\epsilon	λ	\lambda	ϱ	\varrho	χ	\chi
ε	\varepsilon	μ	\mu	σ	\sigma	ψ	\psi
ζ	\zeta	ν	\nu	ς	\varsigma	ω	\omega
η	\eta	ξ	\xi				

Mayúsculas

Γ	\Gamma	Λ	\Lambda	Σ	\Sigma	Ψ	\Psi
Δ	\Delta	Ξ	\Xi	Υ	\Upsilon	Ω	\Omega
Θ	\Theta	Π	\Pi	Φ	\Phi		

Tabla 5.3: Letras griegas.

5.4.5 Letras griegas.

Las letras griegas se obtienen simplemente escribiendo el nombre de dicha letra (en inglés): `\gamma`. Para la mayúscula correspondiente se escribe la primera letra con mayúscula: `\Gamma`. La lista completa se encuentra en la Tabla 5.3.

No existen símbolos para α, β, η, etc. mayúsculas, pues corresponden a letras romanas (A, B, E, etc.).

5.4.6 Letras caligráficas.

Letras caligráficas mayúsculas $\mathcal{A}$, $\mathcal{B}$, ... , $\mathcal{Z}$ se obtienen con `\cal`. `\cal` se usa igual que los otros comandos de cambio de font (`\rm`, `\it`, etc.).

Sea $\mathcal{F}$ una función con $\mathcal{F}(x) > 0$.

```
Sea $\cal F$ una funci\'on
con ${\cal F}(x) > 0$.
```

No son necesarios los paréntesis cursivos la primera vez que se usan en este ejemplo, porque el efecto de `\cal` está delimitado por los `$`.

5.4.7 Símbolos matemáticos.

LaTeX proporciona una gran variedad de símbolos matemáticos (Tablas 5.4, 5.5, 5.6, 5.7).

La negación de cualquier símbolo matemático se obtiene con `\not`:

$x \not< y$

```
x \not < y
```

$a \notin \mathcal{M}$

```
a \not \in {\cal M}
```

[Notemos, sí, en la Tabla 5.5, que existe el símbolo $\neq$ (`\neq`).]

$\pm$	\pm	$\cap$	\cap	$\diamond$	\diamond	$\oplus$	\oplus
$\mp$	\mp	$\cup$	\cup	$\triangle$	\bigtriangleup	$\ominus$	\ominus
$\times$	\times	$\uplus$	\uplus	$\triangledown$	\bigtriangledown	$\otimes$	\otimes
$\div$	\div	$\sqcap$	\sqcap	$\triangleleft$	\triangleleft	$\oslash$	\oslash
$*$	\ast	$\sqcup$	\sqcup	$\triangleright$	\triangleright	$\odot$	\odot
$\star$	\star	$\vee$	\lor	$\bigcirc$	\bigcirc		
$\circ$	\circ	$\wedge$	\land	$\dagger$	\dagger		
$\bullet$	\bullet	$\setminus$	\setminus	$\ddagger$	\ddagger		
$\cdot$	\cdot	$\wr$	\wr	$\amalg$	\amalg		

Tabla 5.4: Símbolos de operaciones binarias.

$\leq$	\leq	$\geq$	\geq	$\equiv$	\equiv	$\models$	\models
$\prec$	\prec	$\succ$	\succ	$\sim$	\sim	$\perp$	\perp
$\preceq$	\preceq	$\succeq$	\succeq	$\simeq$	\simeq	$\mid$	\mid
$\ll$	\ll	$\gg$	\gg	$\asymp$	\asymp	$\parallel$	\parallel
$\subset$	\subset	$\supset$	\supset	$\approx$	\approx	$\bowtie$	\bowtie
$\subseteq$	\subseteq	$\supseteq$	\supseteq	$\cong$	\cong	$\neq$	\neq
$\smile$	\smile	$\sqsubseteq$	\sqsubseteq	$\sqsupseteq$	\sqsupseteq	$\doteq$	\doteq
$\frown$	\frown	$\in$	\in	$\ni$	\ni	$\propto$	\propto
$\vdash$	\vdash	$\dashv$	\dashv				

Tabla 5.5: Símbolos relacionales.

$\leftarrow$	\gets	$\longleftarrow$	\longleftarrow	$\uparrow$	\uparrow
$\Leftarrow$	\Leftarrow	$\Longleftarrow$	\Longleftarrow	$\Uparrow$	\Uparrow
$\rightarrow$	\to	$\longrightarrow$	\longrightarrow	$\downarrow$	\downarrow
$\Rightarrow$	\Rightarrow	$\Longrightarrow$	\Longrightarrow	$\Downarrow$	\Downarrow
$\leftrightarrow$	\Leftrightarrow	$\Longleftrightarrow$	\Longleftrightarrow	$\updownarrow$	\Updownarrow
$\mapsto$	\mapsto	$\longmapsto$	\longmapsto	$\nearrow$	\nearrow
$\hookleftarrow$	\hookleftarrow	$\hookrightarrow$	\hookrightarrow	$\searrow$	\searrow
$\leftharpoonup$	\leftharpoonup	$\rightharpoonup$	\rightharpoonup	$\swarrow$	\swarrow
$\leftharpoondown$	\leftharpoondown	$\rightharpoondown$	\rightharpoondown	$\nwarrow$	\nwarrow
$\rightleftharpoons$	\rightleftharpoons				

Tabla 5.6: Flechas

$\aleph$	\aleph	$\prime$	\prime	$\forall$	\forall	∞	\infty
$\hbar$	\hbar	$\emptyset$	\emptyset	$\exists$	\exists	$\triangle$	\triangle
$\imath$	\imath	∇	\nabla	$\neg$	\lnot	$\clubsuit$	\clubsuit
$\jmath$	\jmath	$\surd$	\surd	$\flat$	\flat	$\diamondsuit$	\diamondsuit
ℓ	\ell	$\top$	\top	$\natural$	\natural	$\heartsuit$	\heartsuit
$\wp$	\wp	$\bot$	\bot	$\sharp$	\sharp	$\spadesuit$	\spadesuit
$\Re$	\Re	$\|$	|	\	\backslash		
$\Im$	\Im	$\angle$	\angle	∂	\partial		

Tabla 5.7: Símbolos varios.

Σ	$\sum$	\sum	$\cap$	$\bigcap$	\bigcap	$\odot$	$\bigodot$	\bigodot
Π	$\prod$	\prod	$\cup$	$\bigcup$	\bigcup	$\otimes$	$\bigotimes$	\bigotimes
$\amalg$	$\coprod$	\coprod	$\sqcup$	$\bigsqcup$	\bigsqcup	$\oplus$	$\bigoplus$	\bigoplus
$\int$	$\int$	\int	$\vee$	$\bigvee$	\bigvee	$\oplus$	$\bigoplus$	\bigoplus
$\oint$	$\oint$	\oint	$\wedge$	$\bigwedge$	\bigwedge			

Tabla 5.8: Símbolos de tamaño variable.

Algunos símbolos tienen tamaño variable, según aparezcan en el texto o en fórmulas separadas del texto. Se muestran en la Tabla 5.8.

Estos símbolos pueden tener índices que se escriben como sub o supraíndices. Nuevamente, la ubicación de estos índices depende de si la fórmula está dentro del texto o separada de él:

$$\sum_{i=1}^{n} x_i = \int_0^1 f$$ `$$\sum_{i=1}^n x_i = \int_0^1 f $$`

$\sum_{i=1}^{n} x_i = \int_0^1 f$ `$\sum_{i=1}^n x_i = \int_0^1 f $`

5.4.8 Funciones tipo logaritmo.

Observemos la diferencia entre estas dos expresiones:

$x = logy$ `$x  = log y$`

$x = \log y$ `$x  = \log y$`

En el primer caso LaTeX escribe el producto de cuatro cantidades, l, o, g e y. En el segundo, representa correctamente nuestro deseo: el logaritmo de y. Todos los comandos de la Tabla 5.9 generan el nombre de la función correspondiente, en letras romanas.

Algunas de estas funciones pueden tener índices:

$$\lim_{n\to\infty} x_n = 0$$ `$$\lim_{n\to\infty} x_n = 0 $$`

$\lim_{n\to\infty} x_n = 0$ `$\lim_{n\to\infty} x_n = 0 $`

\arccos	\cos	\csc	\exp	\ker	\limsup	\min	\sinh
\arcsin	\cosh	\deg	\gcd	\lg	\ln	\Pr	\sup
\arctan	\cot	\det	\hom	\lim	\log	\sec	\tan
\arg	\coth	\dim	\inf	\liminf	\max	\sin	\tanh

Tabla 5.9: Funciones tipo logaritmo

(	(	)	)	↑	\uparrow
[	[	]	]	↓	\downarrow
{	\{	}	\}	↕	\updownarrow
⌊	\lfloor	⌋	\rfloor	⇑	\Uparrow
⌈	\lceil	⌉	\rceil	⇓	\Downarrow
⟨	\langle	⟩	\rangle	⇕	\Updownarrow
/	/	\	\backslash		
|	|	‖	\\|		

Tabla 5.10: Delimitadores

5.4.9 Matrices.

Ambiente `array`.

Se construyen con el ambiente `array`. Consideremos, por ejemplo:

$$\begin{array}{clr} a+b+c & uv & 27 \\ a+b & u+v & 134 \\ a & 3u+vw & 2.978 \end{array}$$

La primera columna está alineada al centro (`c`, center); la segunda, a la izquierda (`l`, left); la tercera, a la derecha (`r`, right). `array` tiene un argumento obligatorio, que consta de tantas letras como columnas tenga la matriz, letras que pueden ser `c`, `l` o `r` según la alineación que queramos obtener. Elementos consecutivos de la misma línea se separan con `&` y líneas consecutivas se separan con `\\`. Así, el ejemplo anterior se obtiene con:

```
\begin{array}{clr}
a+b+c & uv & 27 \\
a+b & u + v & 134 \\
a & 3u+vw & 2.978
\end{array}
```

Delimitadores.

Un delimitador es cualquier símbolo que actúe como un paréntesis, encerrando una expresión, apareciendo a la izquierda y a la derecha de ella. La Tabla 5.10 muestra todos los delimitadores posibles.

Para que los delimitadores tengan el tamaño correcto para encerrar la expresión correspondiente hay que anteponerles `\left` y `\right`. Podemos obtener así expresiones matriciales:

$$\left(\begin{array}{cc} a & b \\ c & d \end{array} \right)$$

```
\left(\begin{array}{cc}
a&b\\
c&d
\end{array}\right)
```

$$v = \left(\begin{array}{c} 1 \\ 2 \\ 3 \end{array} \right)$$

```
v = \left(\begin{array}{c}
1\\
2\\
3
\end{array}\right)
```

$$\Delta = \left| \begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right|$$

```
\Delta = \left|\begin{array}{cc}
a_{11} & a_{12}\\
a_{21} & a_{22}
\end{array}\right|
```

`\left` y `\right` deben ir de a pares, pero los delimitadores no tienen por qué ser los mismos:

$$\left(\begin{array}{c} a \\ b \end{array} \right[$$

```
\left(\begin{array}{c}
a\\
b
\end{array}\right[
```

Tampoco es necesario que los delimitadores encierren matrices. Comparemos, por ejemplo:

$$(\vec{A} + \vec{B}) = (\frac{d\vec{F}}{dx})_{x=a}$$

```
(\vec A + \vec B) =
( \frac{d \vec F}{dx} )_{x=a}
```

$$\left(\vec{A} + \vec{B}\right) = \left(\frac{d\vec{F}}{dx}\right)_{x=a}$$

```
\left(\vec A + \vec B\right) =
\left( \frac{d \vec F}{dx} \right)_{x=a}
```

El segundo ejemplo es mucho más adecuado estéticamente.

Algunas expresiones requieren sólo un delimitador, a la izquierda o a la derecha. Un punto (`.`) representa un delimitador invisible. Los siguientes ejemplos son típicos:

$$\int_a^b dx \frac{df}{dx} = f(x)\Big|_a^b$$

```
\left. \int_a^b  dx \frac{df}{dx} =
f(x) \right |_a^b
```

$$f(x) = \left\{ \begin{array}{cl} 0 & x < 0 \\ 1 & x > 0 \end{array} \right.$$

```
f(x) = \left\{ \begin{array}{cl}
0 & x<0 \\
1 & x>0
\end{array} \right.
```

Fórmulas de más de una línea.

`eqnarray` ofrece una manera de ingresar a modo matemático (en reemplazo de `$`, `$$` o `equation`) equivalente a un `array` con argumentos `{rcl}`:

$\hat{a}$	`\hat a`	$\acute{a}$	`\acute a`	$\bar{a}$	`\bar a`	$\dot{a}$	`\dot a`
$\check{a}$	`\check a`	$\grave{a}$	`\grave a`	$\vec{a}$	`\vec a`	$\ddot{a}$	`\ddot a`
$\breve{a}$	`\breve a`	$\tilde{a}$	`\tilde a`				

Tabla 5.11: Acentos matemáticos

$$\begin{aligned} x &= a+b+c+ \\ &\quad d+e \end{aligned}$$

```
\begin{eqnarray*}
x& = & a + b + c +\\
&& d + e
\end{eqnarray*}
```

El asterisco impide que aparezcan números en las ecuaciones. Si deseamos que numere cada línea como una ecuación independiente, basta omitir el asterisco:

$$\begin{aligned} x &= 5 \quad (5.2) \\ a+b &= 60 \quad (5.3) \end{aligned}$$

```
\begin{eqnarray}
x& = & 5 \\
a + b&= & 60
\end{eqnarray}
```

Si queremos que solamente algunas líneas aparezcan numeradas, usamos `\nonumber`:

$$\begin{aligned} x &= a+b+c+ \\ &\quad d+e \quad (5.4) \end{aligned}$$

```
\begin{eqnarray}
x& = & a + b + c + \nonumber\\
&& d + e
\end{eqnarray}
```

El comando `\eqnarray` es suficiente para necesidades sencillas, pero cuando se requiere escribir matemática de modo intensivo sus limitaciones comienzan a ser evidentes. Al agregar al preámbulo de nuestro documento la línea `\usepackage{amsmath}` quedan disponibles muchos comandos mucho más útiles para textos matemáticos más serios, como el ambiente `equation*`, `\split`, `\multline` o `\intertext`. En la sección 5.8.2 se encuentra una descripción de estos y otros comandos.

5.4.10 Acentos.

Dentro de una fórmula pueden aparecer una serie de "acentos", análogos a los de texto usual (Tabla 5.11).

Las letras i y j deben perder el punto cuando son acentuadas: $\vec{i}$ es incorrecto. Debe ser $\vec{\imath}$. `\imath` y `\jmath` generan las versiones sin punto de estas letras:

$$\vec{\imath} + \hat{\jmath}$$

```
\vec \imath + \hat \jmath
```

5.4.11 Texto en modo matemático.

Para insertar texto dentro de modo matemático empleamos `\mbox`:

$V_{\text{crítico}}$ `V_{\mbox{\scriptsize cr\'{\i}tico}}`

Bastante más óptimo es utilizar el comando `\text`, disponible a través de `amsmath` (sección 5.8.2).

5.4.12 Espaciado en modo matemático.

TEX ignora los espacios que uno escribe en las fórmulas y los determina de acuerdo a sus propios criterios. A veces es necesario ayudarlo para hacer ajustes finos. Hay cuatro comandos que agregan pequeños espacios dentro de modo matemático:

`\,`	espacio pequeño	`\:`	espacio medio
`\!`	espacio pequeño (negativo)	`\;`	espacio grueso

Algunos ejemplos de su uso:

$\sqrt{2}\,x$	`\sqrt 2 \, x`	en vez de	$\sqrt{2}x$
$n/\log n$	`n / \!\log n`	en vez de	$n/\log n$
$\int f\,dx$	`\int f \, dx`	en vez de	$\int f dx$

El último caso es quizás el más frecuente, por cuanto la no inserción del pequeño espacio adicional entre f y dx hace aparecer el integrando como el producto de tres variables, f, d y x, que no es la idea.

5.4.13 Fonts.

Análogamente a los comandos para texto usual (Sec. 5.3.13), es posible cambiar los fonts dentro del modo matemático:

(A, x)	`\mathrm{(A,x)}`
(A, x)	`\mathnormal{(A,x)}`
$(\mathcal{A}, \mathcal{B})$	`\mathcal{(A,B)}`
$(\mathbf{A}, \mathbf{x})$	`\mathbf{(A,x)}`
(A, x)	`\mathsf{(A,x)}`
$(\mathtt{A}, \mathtt{x})$	`\mathtt{(A,x)}`
(A, x)	`\mathit{(A,x)}`

(Recordemos que la letras tipo `\cal` sólo existen en mayúsculas.)

Las declaraciones anteriores permiten cambiar los fonts de letras, dígitos y acentos, pero no de los otros símbolos matemáticos:

$\mathbf{\tilde{A} \times 1}$ `\mathbf{\tilde A \times 1}`

Como en todo ambiente matemático, los espacios entre caracteres son ignorados:

Hola `\mathrm{H o l a}`

Finalmente, observemos que `\mathit` corresponde al font itálico, en tanto que `\mathnormal` al font matemático usual, que es también itálico ... o casi:

$different$	`$different$`
$different$	`$\mathnormal{different}$`

different	`$\mathit{different}$`
different	`\textit{different}`

5.5 Tablas.

`array` nos permitió construir matrices en modo matemático. Para tablas de texto existe `tabular`, que funciona de la misma manera. Puede ser usado tanto en modo matemático como fuera de él.

Nombre	:	Juan Pérez
Edad	:	26
Profesión	:	Estudiante

```
\begin{tabular}{lcl}
Nombre&:&Juan P\'erez\\
Edad&:&26\\
Profesi\'on&:&Estudiante
\end{tabular}
```

Si deseamos agregar líneas verticales y horizontales para ayudar a la lectura, lo hacemos insertando `|` en los puntos apropiados del argumento de `tabular`, y `\hline` al final de cada línea de la tabla:

Item	Gastos
Vasos	$ 500
Botellas	$ 1300
Platos	$ 500
Total	$ 2300

```
\begin{tabular}{|l|r|}\hline
Item&Gastos\\ \hline
Vasos& \$ 500 \\
Botellas & \$ 1300 \\
Platos & \$ 500 \\ \hline
Total& \$ 2300 \\ \hline
\end{tabular}
```

5.6 Referencias cruzadas.

Ecuaciones, secciones, capítulos y páginas son entidades que van numeradas y a las cuales podemos querer referirnos en el texto. Evidentemente no es óptimo escribir explícitamente el número correspondiente, pues la inserción de una nueva ecuación, capítulo, etc., su eliminación o cambio de orden del texto podría alterar la numeración, obligándonos a modificar estos números dispersos en el texto. Mucho mejor es referirse a ellos de modo simbólico y dejar que TEX inserte por nosotros los números. Lo hacemos con `\label` y `\ref`.

La ecuación de Euler

$$e^{i\pi} + 1 = 0 \qquad (5.5)$$

reúne los números más importantes. La ecuación (5.5) es famosa.

```
La ecuaci\'on de Euler
\begin{equation}
\label{euler}
 e^{i\pi} + 1 = 0
\end{equation}
re\'une los n\'umeros
m\'as importantes.
La ecuaci\'on (\ref{euler})
es famosa.
```

El argumento de `\label` (reiterado luego en `\ref`) es una etiqueta simbólica. Ella puede ser cualquier secuencia de letras, dígitos o signos de puntuación. Letras mayúsculas y minúsculas son diferentes. Así, `euler`, `eq:euler`, `euler_1`, `euler1`, `Euler`, etc., son etiquetas válidas y distintas. Podemos usar `\label` dentro de `equation`, `eqnarray` y `enumerate`.

También podemos referenciar páginas con `\pageref`:

Ver página 154 para más detalles.

[Texto en pág. 154]

El significado de la vida ...

```
Ver p\'agina
\pageref{significado}
para m\'as detalles.
...
El significado
\label{significado}
de la vida...
```

LaTeX puede dar cuenta de las referencias cruzadas gracias al archivo `aux` (auxiliar) generado durante la compilación.

Al compilar por primera vez el archivo, en el archivo `aux` es escrita la información de los `\label` encontrados. Al compilar por segunda vez, LaTeX lee el archivo `aux` e incorpora esa información al `dvi`. (En realidad, también lo hizo la primera vez que se compiló el archivo, pero el `aux` no existía entonces o no tenía información útil.)

Por tanto, para obtener las referencias correctas hay que compilar dos veces, una para generar el `aux` correcto, otra para poner la información en el `dvi`. Toda modificación en la numeración tendrá efecto sólo después de compilar dos veces más. Por cierto, no es necesario preocuparse de estos detalles a cada momento. Seguramente compilaremos muchas veces el archivo antes de tener la versión final. En todo caso, LaTeX avisa, tras cada compilación, si hay referencias inexistentes u otras que pudieron haber cambiado, y sugiere compilar de nuevo para obtener las referencias correctas. (Ver Sec. 5.14.2.)

5.7 Texto centrado o alineado a un costado.

Los ambientes `center`, `flushleft` y `flushright` permiten forzar la ubicación del texto respecto a los márgenes. Líneas consecutivas se separan con `\\`:

Una línea centrada,
otra
y otra más.

Ahora el texto continúa

alineado a la izquierda

y finalmente

dos líneas
alineadas a la derecha.

```
\begin{center}
Una l\'{\i}nea centrada,\\
otra\\
y otra m\'as.
\end{center}
Ahora el texto contin\'ua
\begin{flushleft}
alineado a la izquierda
\end{flushleft}
y finalmente
\begin{flushright}
dos l\'{\i}neas\\
alineadas a la derecha.
\end{flushright}
```

5.8 Algunas herramientas importantes

Hasta ahora hemos mencionado escencialmente comandos disponibles en LaTeX standard. Sin embargo, éstos, junto con el resto de los comandos básicos de LaTeX, se vuelven insuficientes cuando se

trata de ciertas aplicaciones demasiado específicas, pero no inimaginables: si queremos escribir un texto de alta matemática, o usar LaTeX para escribir partituras, o para escribir un archivo `.tex` en un teclado croata . . . . Es posible que con los comandos usuales LaTeX responda a las necesidades, pero seguramente ello será a un costo grande de esfuerzo por parte del autor del texto. Por esta razón, las distribuciones modernas de LaTeX incorporan una serie de extensiones que hacen la vida un poco más fácil a los eventuales autores. En esta sección mencionaremos algunas extensiones muy útiles. Muchas otras no están cubiertas, y se sugiere al lector consultar la documentación de su distribución para saber qué otros paquetes se encuentran disponibles.

En general, las extensiones a LaTeX vienen contenidas en *paquetes* (*"packages"*, en inglés), en archivos `.sty`. Así, cuando mencionemos el paquete `amsmath`, nos referimos a características disponibles en el archivo `amsmath.sty`. Para que los comandos de un paquete `<package>.sty` estén disponibles, deben ser cargados durante la compilación, incluyendo en el preámbulo del documento la línea:

```
\usepackage{<package>}
```

Si se requiere cargar más de un paquete adicional, se puede hacer de dos formas:

```
\usepackage{<package1>,<package2>}
```

o

```
\usepackage{<package1>}
\usepackage{<package2>}
```

Algunos paquetes aceptan opciones adicionales (del mismo modo que la clase `article` acepta la opción `12pt`):

```
\usepackage[option1,option2]{<package1>}
```

Revisemos ahora algunos paquetes útiles.

5.8.1 babel

Permite el procesamiento de textos en idiomas distintos del inglés. Esto significa, entre otras cosas, que se incorporan los patrones de silabación correctos para dicho idioma, para cortar adecuadamente las palabras al final de cada línea. Además, palabras claves como "Chapter", "Index", "List of Figures", etc., y la fecha dada por `\date`, son cambiadas a sus equivalentes en el idioma escogido. La variedad de idiomas disponibles es enorme, pero cada instalación de LaTeX tiene sólo algunos de ellos incorporados. (Ésta es una decisión que toma el administrador del sistema, de acuerdo a las necesidades de los usuarios. Una configuración usual puede ser habilitar la compilación en inglés, castellano, alemán y francés.)

Ya sabemos como usar `babel` para escribir en castellano: basta incluir en el preámbulo la línea

```
\usepackage[spanish]{babel}
```

5.8.2 $\mathcal{AMS}$-LaTeX

El paquete `amsmath` permite agregar comandos para escritura de textos matemáticos profesionales, desarrollados originalmente por la American Mathematical Society. Si un texto contiene abundante matemática, entonces seguramente incluir la línea correspondiente en el preámbulo:

```
\usepackage{amsmath}
```

aliviará mucho la tarea. He aquí algunas de las características adicionales disponibles con $\mathcal{AMS}$-LaTeX.

Ambientes para ecuaciones

Con `equation*` generamos una ecuación separada del texto, no numerada:

$$x = 2y - 3$$

```
\begin{equation*}
x = 2y - 3
\end{equation*}
```

`multline` permite dividir una ecuación muy larga en varias líneas, de modo que la primera línea quede alineada con el margen izquierdo, y la última con el margen derecho:

$$\sum_{i=1}^{15} = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13 + 14 + 15 \quad (5.6)$$

```
\begin{multline}
\sum_{i=1}^{15} = 1 +2+3+4+5+\\
6+7+8+9+10+\\
11+12+13+14+15
\end{multline}
```

`align` permite reunir un grupo de ecuaciones consecutivas alineándolas (usando `&`, igual que la alineación vertical de `tabular` y `array`). `gather` hace lo mismo, pero centrando cada ecuación en la página independientemente.

$$a_1 = b_1 + c_1 \quad (5.7)$$
$$a_2 = b_2 + c_2 - d_2 + e_2 \quad (5.8)$$

```
\begin{align}
a_1 &= b_1 + c_1 \\
a_2 &= b_2 + c_2 - d_2 + e_2
\end{align}
```

$$a_1 = b_1 + c_1 \quad (5.9)$$
$$a_2 = b_2 + c_2 - d_2 + e_2 \quad (5.10)$$

```
\begin{gather}
a_1 = b_1 + c_1 \\
a_2 = b_2 + c_2 - d_2 + e_2
\end{gather}
```

Con `multline*`, `align*` y `gather*` se obtienen los mismos resultados, pero con ecuaciones no numeradas.

`split` permite escribir una sola ecuación separada en líneas (como `multline`), pero permite alinear las líneas con `&` (como `align`). `split` debe ser usado dentro de un ambiente como `equation`, `align` o `gather` (o sus equivalentes con asterisco):

$$\begin{aligned} a_1 &= b_1 + c_1 \\ &= b_2 + c_2 - d_2 + e_2 \end{aligned} \tag{5.11}$$

```
\begin{equation}
\begin{split}
a_1& = b_1 + c_1 \\
   & =  b_2 + c_2 - d_2 + e_2
\end{split}
\end{equation}
```

Espacio horizontal

`\quad` y `\qquad` insertan espacio horizontal en ecuaciones:

$$x > y \ , \quad \forall\, x \in A$$
$$x \leq z \ , \qquad \forall\, z \in B$$

```
\begin{gather*}
x > y \ , \quad \forall\, x \in A \\
x \leq z \ , \qquad \forall\, z \in B
\end{gather*}
```

Texto en ecuaciones

Para agregar texto a una ecuación, usamos `\text`:

$$x = 2^n - 1 \ , \quad \text{con } n \text{ entero}$$

```
\begin{equation*}
x = 2^n - 1 \ , \quad \text{con $n$ entero}
\end{equation*}
```

`\text` se comporta como un buen objeto matemático, y por tanto se pueden agregar subíndices textuales más fácilmente que con `\mbox` (ver sección 5.4.11):

$V_{\text{crítico}}$

```
$V_{\text{cr\'{\i}tico}}$
```

Referencia a ecuaciones

`\eqref` es equivalente a `\ref`, salvo que agrega los paréntesis automáticamente:

La ecuación (5.5) era la de Euler.

```
La ecuaci\'on \eqref{euler} era la de Euler.
```

Ecuaciones con casos

Ésta es una construcción usual en matemáticas:

$$f(x) = \begin{cases} 1 & \text{si } x < 0 \\ 0 & \text{si } x > 0 \end{cases}$$

```
f(x)=
\begin{cases}
1&\text{si $x<0$} \\
0&\text{si $x>0$}
\end{cases}
```

Notar cómo es más simple que el ejemplo con los comandos convencionales en la sección 5.4.9.

Texto insertado entre ecuaciones alineadas

Otra situación usual es insertar texto entre ecuaciones alineadas, preservando la alineación:

$$x_1 = a + b + c \ ,$$
$$x_2 = d + e \ ,$$

y por otra parte

$$x_3 = f + g + h \ .$$

```
\begin{align*}
x_1 &=  a + b + c  \ , \\
x_2 &=  d + e \ , \\
\intertext{y por otra parte}
x_3 &=  f + g + h \ .
\end{align*}
```

Matrices y coeficientes binomiales

La complicada construcción de matrices usando `array` (sección 5.4.9), se puede reemplazar con ambientes como `pmatrix` y `vmatrix`, y comandos como `\binom`.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

```
\begin{pmatrix}
a&b\\
c&d
\end{pmatrix}
```

$$\Delta = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$$

```
\Delta = \begin{vmatrix}
a_{11} & a_{12}\\
a_{21} & a_{22}
\end{vmatrix}
```

$$v = \binom{k}{2}$$

```
v = \binom{k}{2}
```

Podemos observar que el espaciado entre los paréntesis y el resto de la fórmula es más adecuado que el de los ejemplos en la sección 5.4.9.

Flechas extensibles

Las flechas en la tabla 5.6 vienen en ciertos tamaños predefinidos. `amsmath` proporciona flechas extensibles `\xleftarrow` y `\xrightarrow`, para ajustar sub o superíndices demasiado anchos. Además, tienen un argumento opcional y uno obligatorio, para colocar material sobre o bajo ellas:

$$A \xleftarrow{n+\mu-1} B \xrightarrow[T]{n\pm i-1} C \underset{U}{\longrightarrow} D$$

```
A \xleftarrow{n+\mu-1} B \xrightarrow[T]{n\pm i-1} C \xrightarrow[U]{} D
```

5.8.3 fontenc

Ocasionalmente, LaTeX tiene problemas al separar una palabra en sílabas. Típicamente, eso ocurre con palabras acentuadas, pues, debido a la estructura interna del programa, un carácter como la "á" en "matemáticas" no es tratado igual que los otros. Para solucionar el problema, y poder cortar en sílabas palabras que contengan letras acentuadas (además de acceder a algunos caracteres adicionales), basta incluir el paquete `fontenc`:

```
\usepackage[T1]{fontenc}
```

Técnicamente, lo que ocurre es que la codificación antigua para fonts es la `OT1`, que no contiene fonts acentuados, y que por lo tanto es útil sólo para textos en inglés. La codificación `T1` aumenta los fonts disponibles, permitiendo que los caracteres acentuados sean tratados en igual pie que cualquier otro.

5.8.4 enumerate

`enumerate.sty` define una muy conveniente extensión al ambiente `enumerate` de LaTeX. El comando se usa igual que siempre (ver sección 5.3.12), con un argumento opcional que determina el tipo de etiqueta que se usará para la lista. Por ejemplo, si queremos que en vez de números se usen letras mayúsculas, basta usar `\begin{enumerate}[A]`:

A Primer ítem.

B Segundo ítem.

Si queremos etiquetas de la forma "1.-", `\begin{enumerate}[1.-]`:

1.- Primer ítem.

2.- Segundo ítem.

Si deseamos insertar un texto que no cambie de una etiqueta a otra, hay que encerrarlo entre paréntesis cursivos (`\begin{enumerate}[{Caso} A:]`):

Caso A: Primer ítem.

Caso B: Segundo ítem.

5.8.5 Color.

A través de PostScript es posible introducir color en documentos LaTeX. Para ello, incluimos en el preámbulo el paquete `color.sty`:

```
\usepackage{color}
```

De este modo, está disponible el comando `\color`, que permite especificar un color, ya sea por nombre (en el caso de algunos colores predefinidos), por su código `rgb` (red-green-blue) o código `cmyk` (cian-magenta-yellow-black). Por ejemplo:

Un texto en azul
Un texto en un segundo color
Un texto en un tercer color

```
Un texto en {\color{blue} azul}

Un texto en un
{\color[rgb]{1,0,1} segundo color}

Un texto en un
{\color[cmyk]{.3,.5,.75,0} tercer color}
```

Los colores más frecuentes (azul, amarillo, rojo, etc.) se pueden dar por nombre, como en este ejemplo. Si se da el código `rgb`, se deben especificar tres números entre 0 y 1, que indican la cantidad

de rojo, verde y azul que constituyen el color deseado. En el ejemplo, le dimos máxima cantidad de rojo y azul, y nada de verde, con lo cual conseguimos un color violeta. Si se trata del código `cmyk` los números a especificar son cuatro, indicando la cantidad de cian, magenta, amarillo y negro. En el ejemplo anterior pusimos una cantidad arbitraria de cada color, y resultó un color café. Es evidente que el uso de los códigos `rgb` y `cmyk` permite explorar infinidad de colores.

Observar que `\color` funciona de modo análogo a los comandos de cambio de font de la sección 5.3.13, de modo que si se desea restringir el efecto a una porción del texto, hay que encerrar dicho texto entre paréntesis cursivos. Análogamente al caso de los fonts, existe el comando `\textcolor`, que permite dar el texto a colorear como argumento:

Un texto en azul
Un texto en un segundo color
Un texto en un tercer color

```
Un texto en \textcolor{blue}{azul}

Un texto en un
\textcolor[rgb]{1,0,1}{segundo color}

Un texto en un
\textcolor[cmyk]{.3,.5,.75,0}{tercer color}
```

5.9 Modificando el estilo de la página.

TEX toma una serie de decisiones por nosotros. Ocasionalmente nos puede interesar alterar el comportamiento normal. Disponemos de una serie de comandos para ello, los cuales revisaremos a continuación. Todos deben aparecer en el preámbulo, salvo en los casos que se indique.

5.9.1 Estilos de página.

a) Números de página.

Si se desea que los números de página sean arábicos (1, 2, 3 . . .):

```
\pagenumbering{arabic}
```

Para números romanos (i, ii, iii, . . .):

```
\pagenumbering{roman}
```

`arabic` es el default.

b) Estilo de página.

El comando `\pagestyle` determina dónde queremos que vayan los números de página:

`\pagestyle{plain}`	Números de página en el extremo inferior, al centro de la página. (Default para estilos `article`, `report`.)
`\pagestyle{headings}`	Números de página y otra información (título de sección, etc.) en la parte superior de la página. (Default para estilo `book`.)
`\pagestyle{empty}`	Sin números de página.

5.9.2 Corte de páginas y líneas.

TEX tiene modos internos de decidir cuándo cortar una página o una línea. Al preparar la versión final de nuestro documento, podemos desear coartar sus decisiones. En todo caso, no hay que hacer esto antes de preparar la versión verdaderamente final, porque agregar, modificar o quitar texto puede alterar los puntos de corte de líneas y páginas, y los cortes inconvenientes pueden resolverse solos.

Los comandos de esta sección no van en el preámbulo, sino en el interior del texto.

Corte de líneas.

En la página 143 ya vimos un ejemplo de inducción de un corte de línea en un punto deseado del texto, al dividir una palabra en sílabas.

Cuando el problema no tiene relación con sílabas disponemos de dos comandos:

`\newline`	Corta la línea y pasa a la siguiente en el punto indicado.
`\linebreak`	Lo mismo, pero justificando la línea para adecuarla a los márgenes.

Un corte de línea
no justificado a los márgenes
en curso.

```
Un corte de l\'{\i}nea\newline
no justificado a los m\'argenes
en curso.
```

Un corte de línea
justificado a los márgenes
en curso.

```
Un corte de l\'{\i}nea\linebreak
justificado a los m\'argenes
en curso.
```

Observemos cómo en el segundo caso, en que se usa `\linebreak`, la separación entre palabras es alterada para permitir que el texto respete los márgenes establecidos.

Corte de páginas.

Como para cortar líneas, existe un modo violento y uno sutil:

`\newpage`	Cambia de página en el punto indicado. Análogo a `\newline`.
`\clearpage`	Lo mismo, pero ajustando los espacios verticales en el texto para llenar del mejor modo posible la página.

`\clearpage`, sin embargo, no siempre tiene efectos visibles. Dependiendo de la cantidad y tipo de texto que quede en la página, los espacios verticales pueden o no ser ajustados, y si no lo son, el resultado termina siendo equivalente a un `\newpage`. TEX decide en última instancia qué es lo óptimo.

Adicionalmente, tenemos el comando:

`\enlargethispage{<longitud>}`	Cambia el tamaño de la página actual en la cantidad `<longitud>`.

(Las unidades de longitud que maneja TEX se revisan a continuación.)

Unidades de longitud y espacios.

a) Unidades.

TEX reconoce las siguientes unidades de longitud:

`cm`	centímetro
`mm`	milímetro
`in`	pulgada
`pt`	punto (1/72 pulgadas)
`em`	ancho de una "M" en el font actual
`ex`	altura de una "x" en el font actual

Las cuatro primeras unidades son absolutas; las últimas dos, relativas, dependiendo del tamaño del font actualmente en uso.

Las longitudes pueden ser números enteros o decimales, positivos o negativos:

```
1cm     1.6in     .58pt     -3ex
```

b) Cambio de longitudes.

TEX almacena los valores de las longitudes relevantes al texto en comandos especiales:

`\parindent`	Sangría.
`\textwidth`	Ancho del texto.
`\textheight`	Altura del texto.
`\oddsidemargin`	Margen izquierdo menos 1 pulgada.
`\topmargin`	Margen superior menos 1 pulgada.
`\baselineskip`	Distancia entre la base de dos líneas de texto consecutivas.
`\parskip`	Distancia entre párrafos.

Todas estas variables son modificables con los comandos `\setlength`, que le da a una variable un valor dado, y `\addtolength`, que le suma a una variable la longitud especificada. Por ejemplo:

`\setlength{\parindent}{0.3em}` (`\parindent` = 0.3 cm.)

`\addtolength{\parskip}{1.5cm}` (`\parskip` = `\parskip` + 1.5 cm.)

Por default, el ancho y altura del texto, y los márgenes izquierdo y superior, están definidos de modo que quede un espacio de una pulgada ($\simeq$ 2.56 cm) entre el borde del texto y el borde de la página.

Un problema típico es querer que el texto llene un mayor porcentaje de la página. Por ejemplo, para que el margen del texto en los cuatro costados sea la mitad del default, debemos introducir los comandos:

```
\addtolength{\textwidth}{1in}
\addtolength{\textheight}{1in}
\addtolength{\oddsidemargin}{-.5in}
\addtolength{\topmargin}{-.5in}
```

Las dos primeras líneas aumentan el tamaño horizontal y vertical del texto en 1 pulgada. Si luego restamos media pulgada del margen izquierdo y el margen superior, es claro que la distancia entre el texto y los bordes de la página sera de media pulgada, como deseábamos.

c) Espacios verticales y horizontales.

Se insertan con `\vspace` y `\hspace`:

`\vspace{3cm}`	Espacio vertical de 3 cm.
`\hspace{3cm}`	Espacio horizontal de 3 cm.

Algunos ejemplos:

Un primer párrafo de un pequeño texto.

Y un segundo párrafo separado del otro.

Tres palabras separadas del resto.

```
Un primer p\'arrafo de un
peque\~no texto.

\vspace{1cm}
Y un segundo p\'arrafo
separado del otro.

Tres\hspace{.5cm}palabras
\hspace{.5cm}separadas
del resto.
```

Si por casualidad el espacio vertical impuesto por `\vspace` debiese ser colocado al comienzo de una página, TEX lo ignora. Sería molesto visualmente que en algunas páginas el texto comenzara algunos centímetros más abajo que en el resto. Lo mismo puede ocurrir si el espacio horizontal de un `\hspace` queda al comienzo de una línea.

Los comandos `\vspace*{<longitud>}` y `\hspace*{<longitud>}` permiten que el espacio en blanco de la `<longitud>` especificada no sea ignorado. Ello es útil cuando invariablemente queremos ese espacio vertical u horizontal, aunque sea al comienzo de una página o una línea —por ejemplo, para insertar una figura.

5.10 Figuras.

Lo primero que hay que decir en esta sección es que LATEX es un excelente procesador de texto, tanto convencional como matemático. Las figuras, sin embargo, son un problema aparte.

LATEX provee un ambiente `picture` que permite realizar dibujos simples. Dentro de la estructura `\begin{picture}` y `\end{picture}` se pueden colocar una serie de comandos para dibujar líneas, círculos, óvalos y flechas, así como para posicionar texto. Infortunadamente, el proceso de ejecutar dibujos sobre un cierto umbral de complejidad puede ser muy tedioso para generarlo directamente.

Existe software (por ejemplo, `xfig`) que permite superar este problema, pudiéndose dibujar con el mouse, exportando el resultado al formato `picture` de LaTeX. Sin embargo, `picture` tiene limitaciones (no se pueden hacer líneas de pendiente arbitraria), y por tanto no es una solución óptima.

Para obtener figuras de buena calidad es imprescindible recurrir a lenguajes gráficos externos, y LaTeX da la posibilidad de incluir esos formatos gráficos en un documento. De este modo, tanto el texto como las figuras serán de la más alta calidad. Las dos mejores soluciones son utilizar Metafont o PostScript. Metafont es un programa con un lenguaje de programación gráfico propio. De hecho, los propios fonts de LaTeX fueron creados usando Metafont, y sus capacidades permiten hacer dibujos de complejidad arbitraria. Sin embargo, los dibujos resultantes no son trivialmente reescalables, y exige aprender un lenguaje de programación específico.

Una solución mucho más versátil, y adoptada como el estándar en la comunidad de usuarios de LaTeX, es el uso de PostScript. Como se mencionó brevemente en la sección 1.12, al imprimir, una máquina UNIX convierte el archivo a formato PostScript, y luego lo envía a la impresora. Pero PostScript sirve más que para imprimir, siendo un lenguaje de programación gráfico completo, con el cual podemos generar imágenes de gran calidad, y reescalables sin pérdida de resolución. Además, muchos programas gráficos permiten exportar sus resultados en formato PostScript. Por lo tanto, podemos generar nuestras figuras en alguno de estos programas (`xfig` es un excelente software, que satisface la mayor parte de nuestras necesidades de dibujos simples; `octave` o `gnuplot` pueden ser usados para generar figuras provenientes de cálculos científicos, etc.), lo cual creará un archivo con extensión `.ps` (PostScript) o `.eps` (PostScript encapsulado).[4] Luego introducimos la figura en el documento LaTeX, a través del paquete `graphicx`.

5.10.1 graphicx.sty

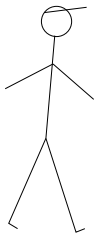
Si nuestra figura está en un archivo `figura.eps`, la instrucción a utilizar es:

```
\documentclass[12pt]{article}
\usepackage{graphicx}
\begin{document}
... Texto ...
\includegraphics[width=w, height=h]{figura.eps}
...
\end{document}
```

Los parámetros `width` y `height` son opcionales y puede omitirse uno para que el sistema escale de acuerdo al parámetro dado. Es posible variar la escala completa de la figura o rotarla usando comandos disponibles en `graphicx`.

[4] `eps` es el formato preferido, pues contiene información sobre las dimensiones de la figura, información que es utilizada por LaTeX para insertar ésta adecuadamente en el texto.

Una figura aquí:



puede hacer más agradable el
texto.

```
Una figura aqu\'{\i}:

\begin{center}
\includegraphics[height=3cm]{figura.eps}
\end{center}

puede hacer m\'as agradable
el texto.
```

En este ejemplo, indicamos sólo la altura de la figura (3cm). El ancho fue determinado de modo que las proporciones de la figura no fueran alteradas. Si no se especifica ni la altura ni el ancho, la figura es insertada con su tamaño natural.

Observemos también que pusimos la figura en un ambiente `center`. Esto no es necesario, pero normalmente uno desea que las figuras estén centradas en el texto.

5.10.2 Ambiente `figure`.

Insertar una figura es una cosa. Integrarla dentro del texto es otra. Para ello está el ambiente `figure`, que permite: (a) posicionar la figura automáticamente en un lugar predeterminado o especificado por el usuario; (b) numerar las figuras; y (c) agregar un breve texto explicativo junto a la figura.

Coloquemos la misma figura de la sección anterior dentro de un ambiente `figure`. El input:

```
\begin{figure}[h]
\begin{center}
\includegraphics[height=3cm]{figura.eps}
\end{center}
\caption{Un sujeto caminando.}
\label{caminando}
\end{figure}
```

da como resultado:

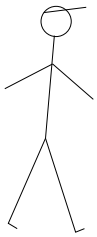


Figura 5.1: Un sujeto caminando.

`figure` delimita lo que en TEX se denomina un *objeto flotante*, es decir, un objeto cuya posición no está determinada *a priori*, y se ajusta para obtener los mejores resultados posibles. TEX considera (de acuerdo con la tradición), que la mejor posición para colocar una figura es al principio o al final

de la página. Además, lo ideal es que cada página tenga un cierto número máximo de figuras, que ninguna figura aparezca en el texto antes de que sea mencionada por primera vez, y que, por supuesto, las figuras aparezcan en el orden en que son mencionadas. Éstas y otras condiciones determinan la posición que un objeto flotante tenga al final de la compilación. Uno puede forzar la decisión de LaTeX con el argumento opcional de `figure`:

`t`	(*top)*	extremo superior de la página
`b`	(*bottom*)	extremo inferior de la página
`h`	(*here*)	aquí, en el punto donde está el comando
`p`	(*page of floats*)	en una página separada al final del texto

El argumento adicional `!` suprime, para ese objeto flotante específico, cualquier restricción que exista sobre el número máximo de objetos flotantes en una página y el porcentaje de texto mínimo que debe haber en una página.

Varios de estos argumentos se pueden colocar simultánemente, su orden dictando la prioridad. Por ejemplo,

```
\begin{figure}[htbp]
...
\end{figure}
```

indica que la figura se debe colocar como primera prioridad aquí mismo; si ello no es posible, al comienzo de página (ésta o la siguiente, dependiendo de los detalles de la compilación), y así sucesivamente.

Además, `figure` numera automáticamente la figura, colocando el texto "Figura N:", y `\caption` permite colocar una leyenda, centrada en el texto, a la figura. Puesto que la numeración es automática, las figuras pueden ser referidas simbólicamente con `\label` y `\ref` (sección 5.6). Para que la referencia sea correcta, `\label` debe estar dentro del argumento de `\caption`, o después, como aparece en el ejemplo de la Figura 5.1 (`\ref{caminando}`!).

Finalmente, notemos que la figura debió ser centrada explícitamente con `center`. `figure` no hace nada más que tratar la figura como un objeto flotante, proporcionar numeración y leyenda. El resto es responsabilidad del autor.

5.11 Cartas.

Para escribir cartas debemos emplear el estilo `letter` en vez del que hemos utilizado hasta ahora, `article`. Comandos especiales permiten escribir una carta, poniendo en lugares adecuados la dirección del remitente, la fecha, la firma, etc.

A modo de ejemplo, consideremos el siguiente input:

```
\documentclass[12pt]{letter}

\usepackage[spanish]{babel}

\begin{document}

\address{Las Palmeras 3425\\
\~Nu\~noa, Santiago}
\date{9 de Julio de 1998}
```

```
\signature{Pedro P\'erez \\ Secretario}

\begin{letter}{Dr.\ Juan P\'erez \\ Las Palmeras 3425 \\
\~Nu\~noa, Santiago}
\opening{Estimado Juan}

A\'un no tenemos novedades.

Parece incre\'{\i}ble, pero los recientes acontecimientos nos han superado,
a pesar de nuestros esfuerzos.  Esperamos que mejores tiempos nos
aguarden.

\closing{Saludos,}
\cc{Arturo Prat \\ Luis Barrios}

\end{letter}
\end{document}
```

El resultado se encuentra en la próxima página.

Las Palmeras 3425
Ñuñoa, Santiago

9 de Julio de 1998

Dr. Juan Pérez
Las Palmeras 3425
Ñuñoa, Santiago

Estimado Juan

Aún no tenemos novedades.

Parece increíble, pero los recientes acontecimientos nos han superado, a pesar de nuestros esfuerzos. Esperamos que mejores tiempos nos aguarden.

Saludos,

Pedro Pérez
Secretario

Copia a: Arturo Prat
Luis Barrios

Observemos que el texto de la carta está dentro de un ambiente `letter`, el cual tiene un argumento obligatorio, donde aparece el destinatario de la carta (con su dirección opcionalmente).

Los comandos disponibles son:

`\address{<direccion>}`	`<direccion>` del remitente.
`\signature{<firma>}`	`<firma>` del remitente.
`\opening{<apertura>}`	Fórmula de `<apertura>`.
`\closing{<despedida>}`	Fórmula de `<despedida>`.
`\cc{<copias>}`	Receptores de `<copias>` (si los hubiera).

Uno puede hacer más de una carta con distintos ambientes `letter` en un mismo archivo. Cada una tomará el mismo remitente y firma dados por `\address` y `\signature`. Si deseamos que `\address` o `\signature` valgan sólo para una carta particular, basta poner dichos comandos entre el `\begin{letter}` y el `\opening` correspondiente.

Por ejemplo, la siguiente estructura:

```
\documentclass[12pt]{letter}
\begin{document}
\address{<direccion remitente>}
\date{<fecha>}
\signature{<firma>}

\begin{letter}{<destinatario 1>}
\opening<apertura 1>
...
\end{letter}

\begin{letter}{<destinatario 2>}
\address{<direccion remitente 2>}
\signature{<firma 2>}
\opening<apertura 2>
...
\end{letter}

\begin{letter}{<destinatario 3>}
\opening<apertura 3>
...
\end{letter}
\end{document}
```

dará origen a tres cartas con la misma dirección de remitente y firma, salvo la segunda.

En todos estos comandos, líneas sucesivas son indicadas con `\\`.

5.12 LaTeX y el formato pdf.

Junto con PostScript, otro formato ampliamente difundido para la transmisión de archivos, especialmente a través de Internet, es el formato `pdf` (Portable Document Format). Para generar un

archivo `pdf` con LaTeX es necesario compilarlo con `pdflatex`. Así, `pdflatex <archivo>` generará un archivo `<archivo>.pdf` en vez del `<archivo>.dvi` generado por el compilador usual.

Si nuestro documento tiene figuras, sólo es posible incluirlas en el documento si están también en formato `pdf`. Por tanto, si tenemos un documento con figuras en PostScript, debemos introducir dos modificaciones antes de compilar con `pdflatex`:

a) Cambiar el argumento de `\includegraphics` (sección 5.10) de `<archivo_figura>.eps` a `<archivo_figura>.pdf`.

b) Convertir las figuras PostScript a `pdf` (con `epstopdf`, por ejemplo). Si tenemos una figura en el archivo `<archivo_figura>.eps`, entonces `epstopdf <archivo_figura>.eps` genera el archivo correspondiente `<archivo_figura>.pdf`.

Observar que el mismo paquete `graphicx` descrito en la sección 5.10 para incluir figuras PostScript permite, sin modificaciones, incluir figuras en `pdf`.

5.13 Modificando LaTeX.

Esta sección se puede considerar "avanzada". Normalmente uno se puede sentir satisfecho con el desempeño de LaTeX, y no es necesaria mayor intervención. A veces, dependiendo de la aplicación y del autor, nos gustaría modificar el comportamiento default. Una alternativa es definir nuevos comandos que sean útiles para nosotros. Si esos nuevos comandos son abundantes, o queremos reutilizarlos frecuentemente en otros documentos, lo conveniente es considerar crear un nuevo paquete o incluso una nueva clase. Examinaremos a continuación los elementos básicos de estas modificaciones.

5.13.1 Definición de nuevos comandos.

El comando `\newcommand`

Un nuevo comando se crea con:

```
\newcommand{<comando>}{<accion>}
```

El caso más sencillo es cuando una estructura se repite frecuentemente en nuestro documento. Por ejemplo, digamos que un sujeto llamado Cristóbal no quiere escribir su nombre cada vez que aparece en su documento:

Mi nombre es Cristóbal.
Sí, como oyes, Cristóbal.
Cristóbal Loyola.

```
\newcommand{\nombre}{Crist\'obal}
...
\begin{document}
...
Mi nombre es \nombre. S\'{\i}, como oyes,
\nombre. \nombre\ Loyola.
```

Un `\newcommand` puede aparecer en cualquier parte del documento, pero lo mejor es que esté en el preámbulo, de modo que sea evidente qué nuevos comandos están disponibles en el presente documento. Observemos además que la definición de un comando puede contener otros comandos (en este caso, `\'`). Finalmente, notamos que ha sido necesario agregar un espacio explícito con `\ `, al escribir "Cristóbal Loyola": recordemos que un comando comienza con un backslash y termina

con el primer carácter que no es letra. Por tanto, `\nombre Loyola` ignora el espacio al final de `\nombre`, y el output sería "CristóbalLoyola".

También es posible definir comandos que funcionen en modo matemático:

Sea $\dot{x}$ la velocidad, de modo que $\dot{x}(t) > 0$ si $t < 0$.

```
\newcommand{\vel}{\dot x}

Sea $\vel$ la velocidad, de modo que
$ \vel(t)> 0$ si $t<0$.
```

Como `\vel` contiene un comando matemático (`\dot`), `\vel` sólo puede aparecer en modo matemático.

Podemos también incluir la apertura de modo matemático en la definición de `\vel`: `\newcommand{\vel}{$\dot x$}`. De este modo, `\vel` (no `$\vel$`) da como output directamente $\dot{x}$. Sin embargo, esta solución no es óptima, porque la siguiente ocurrencia de `\vel` da un error. En efecto, si `\vel` = `$\dot x$`, entonces `$ \vel(t)>0$` = `$ $\dot x$> 0$`. En tal caso, LaTeX ve que un modo matemático se ha abierto y cerrado inmediatamente, conteniendo sólo un espacio entremedio, y luego, *en modo texto*, viene el comando `\dot`, que es matemático: LaTeX acusa un error y la compilación se detiene.

La solución a este problema es utilizar el comando `\ensuremath`, que asegura que haya modo matemático, pero si ya hay uno abierto, no intenta volverlo a abrir:

Sea $\dot{x}$ la velocidad, de modo que $\dot{x}(t) > 0$ si $t < 0$.

```
\newcommand{\vel}{\ensuremath{\dot x}}

Sea \vel\ la velocidad, de modo que
$ \vel(t)> 0$ si $t<0$.
```

Un caso especial de comando matemático es el de operadores tipo logaritmo (ver Tabla 5.9). Si queremos definir una traducción al castellano de `\sin`, debemos usar el comando `\DeclareMathOperator` disponible via `amsmath`:

Ahora podemos escribir en castellano, $\operatorname{sen} x$.

```
\usepackage{amsmath}
\DeclareMathOperator{\sen}{sen}
...
Ahora podemos escribir en castellano, $\sen x$.
```

A diferencia de `\newcommand`, `\DeclareMathOperator` sólo puede aparecer en el preámbulo del documento.

Un nuevo comando puede también ser usado para ahorrar tiempo de escritura, reemplazando comandos largos de LaTeX:

1. El primer caso.
2. Ahora el segundo.
3. Y el tercero.

```
\newcommand{\be}{\begin{enumerate}}
\newcommand{\ee}{\end{enumerate}}

\be
\item El primer caso.
\item Ahora el segundo.
\item Y el tercero.
\ee
```

Nuevos comandos con argumentos

Podemos también definir comandos que acepten argumentos. Si el sujeto anterior, Cristóbal, desea escribir cualquier nombre precedido de "Nombre:" en itálica, entonces puede crear el siguiente comando:

Nombre: Cristóbal
Nombre: Violeta

```
\newcommand{\nombre}[1]{\textit{Nombre:} #1}

\nombre{Crist\'obal}

\nombre{Violeta}
```

Observemos que `\newcommand` tiene un argumento opcional, que indica el número de argumentos que el nuevo comando va a aceptar. Esos argumentos se indican, dentro de la definición del comando, con `#1`, `#2`, etc. Por ejemplo, consideremos un comando que acepta dos argumentos:

$f(x,y) + f(x_3, y*) = 0$.

```
\newcommand{\fn}[2]{f(#1,#2)}

$$ \fn{x}{y} + \fn{x_3}{y*} = 0 \ . $$
```

En los casos anteriores, todos los argumentos son obligatorios. LaTeX permite definir comandos con un (sólo un) argumento opcional. Si el comando acepta n argumentos, el argumento opcional es el `#1`, y se debe indicar, en un segundo paréntesis cuadrado, su valor default. Así, podemos modificar el comando `\fn` del ejemplo anterior para que el primer argumento sea opcional, con valor default x:

$f(x,y) + f(x_3, y*) = 0$.

```
\newcommand{\fn}[2][x]{f(#1,#2)}

$$ \fn{y} + \fn[x_3]{y*} = 0 \ . $$
```

Redefinición de comandos

Ocasionalmente no nos interesa definir un nuevo comando, sino redefinir la acción de un comando preexistente. Esto se hace con `\renewcommand`:

La antigua versión de `ldots`: La antigua versi\'on de {\tt ldots}: \ldots
...
La nueva versión de `ldots`: •••

```
\renewcommand{\ldots}{\textbullet \textbullet
\textbullet}

La nueva versi\'on de {\tt ldots}: \ldots
```

Párrafos y cambios de línea dentro de comandos

En el segundo argumento de `\newcommand` o `\renewcommand` puede aparecer cualquier comando de LaTeX, pero ocasionalmente la aparición de líneas en blanco (para forzar un cambio de párrafo) puede provocar problemas. Si ello ocurre, podemos usar `\par`, que hace exactamente lo mismo. Además, la definición del comando queda más compacta:

```
\newcommand{\comandolargo}{\par Un nuevo comando que incluye un cambio de
  p\'arrafo, porque deseamos incluir bastante texto.\par \'Este es el
  nuevo p\'arrafo.\par}

Observemos en acci\'on el comando: \comandolargo Listo.
```

da como resultado:

Observemos en acción el comando:

Un nuevo comando que incluye un cambio de párrafo, porque deseamos incluir bastante texto.

Éste es el nuevo párrafo.

Listo.

Un ejemplo más útil ocurre cuando queremos asegurar un cambio de párrafo, por ejemplo, para colocar un título de sección:

Observemos en acción el comando:

Sección: Ejemplo

Listo.

```
\newcommand{\seccion}[1]{\par\vspace{.5cm}
{\bf Secci\'on: #1}\par\vspace{.5cm}}

Observemos en acci\'on el comando:
\seccion{Ejemplo} Listo.
```

Además de las líneas en blanco, los cambios de línea pueden causar problemas dentro de la definición de un nuevo comando. El ejemplo anterior, con el comando `\seccion`, es un buen ejemplo: notemos que cuando se definió, pusimos un cambio de línea después de `\vspace{.5cm}`. Ese cambio de línea es interpretado (como todos los cambios de línea) como un espacio en blanco, y es posible que, bajo ciertas circunstancias, ese espacio en blanco produzca un output no deseado. Para ello basta utilizar sabiamente el carácter `%`, que permite ignorar todo el resto de la línea, *incluyendo el cambio de línea*. Ilustremos lo anterior con los siguientes tres comandos, que subrayan (comando `\underline`) una palabra, y difieren sólo en el uso de `%` para borrar cambios de línea:

Notar la diferencia entre:
Un texto de prueba , Un texto de prueba , y Un texto de prueba.

```
\newcommand{\texto}{
Un texto de prueba
}
\newcommand{\textodos}{%
Un texto de prueba
}
\newcommand{\textotres}{%
Un texto de prueba%
}

Notar la diferencia entre:

\underline{\texto},
\underline{\textodos},
y
\underline{\textotres}.
```

`\texto` conserva espacios en blanco antes y después del texto, `\textodos` sólo el espacio en

blanco después del texto, y `\textotres` no tiene espacios en blanco alrededor del texto.

Nuevos ambientes

Nuevos ambientes en LaTeX se definen con `\newenvironment`:

```
\newenvironment{<ambiente>}{<comienzo ambiente>}{<final ambiente>}
```

define un ambiente `<ambiente>`, tal que `\begin{ambiente}` ejecuta los comandos `<comienzo ambiente>`, y `\end{ambiente}` ejecuta los comandos `<final ambiente>`.

Definamos un ambiente que, al comenzar, cambia el font a itálica, pone una línea horizontal (`\hrule`) y deja un espacio vertical de .3cm, y que al terminar cambia de párrafo, coloca `XXX` en sans serif, deja un nuevo espacio vertical de .3cm, y vuelve al font roman:

```
\newenvironment{na}{\it \hrule \vspace{.3cm}}{\par\sf XXX \vspace{.3cm}\rm}
```

Entonces, con

```
\begin{na}
  Hola a todos. Es un placer saludarlos en este d\'{\i}a tan especial.

Nunca esper\'e una recepci\'on tan calurosa.
\end{na}
```

obtenemos:

Hola a todos. Es un placer saludarlos en este día tan especial.
Nunca esperé una recepción tan calurosa.
XXX

Los nuevos ambientes también pueden ser definidos de modo que acepten argumentos. Como con `\newcommand`, basta agregar como argumento opcional a `\newenvironment` un número que indique cuántos argumentos se van a aceptar:

```
\newenvironment{<ambiente>}[n]{<comienzo ambiente>}{<final ambiente>}
```

Dentro de `<comienzo ambiente>`, se alude a cada argumento como `#1`, `#2`, etc. Los argumentos no pueden ser usados en los comandos de cierre del ambiente (`<final ambiente>`). Por ejemplo, modifiquemos el ambiente `na` anterior, de modo que en vez de colocar una línea horizontal al comienzo, coloque lo que le indiquemos en el argumento:

```
\newenvironment{na}[1]{\it #1 \vspace{.3cm}}{\par\sf XXX\hrule\vspace{.3cm}\rm}
```

Ahora usémoslo dos veces, cada una con un argumento distinto:

El mismo ejemplo anterior, ahora es

Hola a todos ...
XXX

Pero podemos ahora cambiar el comienzo:
XXX Hola a todos ...

XXX

```
El mismo ejemplo anterior, ahora es

\begin{na}{\hrule}
  Hola a todos...
\end{na}

Pero podemos ahora cambiar el comienzo:

\begin{na}{\it XXX}
  Hola a todos...
\end{na}
```

5.13.2 Creación de nuevos paquetes y clases

Si la cantidad de nuevos comandos y/o ambientes que necesitamos en nuestro documento es suficientemente grande, debemos considerar crear un nuevo paquete o una nueva clase. Para ello hay que tener clara la diferencia entre uno y otro. En general, se puede decir que si nuestros comandos involucran alterar la apariencia general del documento, entonces corresponde crear una nueva clase (`.cls`). Si, por el contrario, deseamos que nuestros comandos funcionen en un amplio rango de circunstancias, para diversas apariencias del documento, entonces lo adecuado es un paquete (`.sty`).

Consideremos por ejemplo la experiencia de los autores de estos apuntes. Para crear estos apuntes necesitamos básicamente la clase `book`, con ciertas modificaciones: márgenes más pequeños, inclusión automática de los paquetes `amsmath`, `babel` y `graphicx`, entre otros, y definición de ciertos ambientes específicos. Todo ello afecta la apariencia de este documento, cambiándola de manera apreciable, pero a la vez de un modo que en general no deseamos en otro tipo de documento. Por ello lo hemos compilado usando una clase adecuada, llamada `mfm2.cls`.

Por otro lado, uno de los autores ha necesitado escribir muchas tareas, pruebas y controles de ayudantía en su vida, y se ha convencido de que su trabajo es más fácil creando una clase `tarea.cls`, que sirve para esos tres propósitos, definiendo comandos que le permiten especificar fácilmente la fecha de entrega de la tarea, o el tiempo disponible para una prueba, los nombres del profesor y el ayudante, etc., una serie de comandos específicos para sus necesidades.

Sin embargo, tanto en este documento que usa `mfm2.cls`, como en las tareas y pruebas que usan `tarea.cls`, se utilizan algunos comandos matemáticos que no vienen con LaTeX, pero que son recurrentes, como `\sen` (la función seno en castellano), `\modulo` (el módulo de un vector), o `\TLaplace` (la transformada de Laplace). Para que estos comandos estén disponibles en cualquier tipo de documento, necesitamos reunirlos en un paquete, en este caso `addmath.sty`. De este modo, `mfm2.cls`, `tarea.cls` o cualquier otra clase pueden llamar a este paquete y utilizar sus comandos.

Estructura básica.

La estructura básica de un paquete o una clase es:

a) Identificación: Información general (nombre del paquete, fecha de creación, etc.). (Obligatoria.)

b) Declaraciones preliminares: Opcionales, dependiendo del paquete o clase en cuestión.

c) Opciones: Comandos relacionados con el manejo de las opciones con las cuales el paquete o clase pueden ser invocados. (Opcional.)

d) Más declaraciones: Aquí van los comandos que constituyen el cuerpo de la clase o paquete. (Obligatoria: si no hay ninguna declaración, el paquete o clase no hace nada, naturalmente.)

La identificación está consituida por las siguientes dos líneas, que deben ir al comienzo del archivo:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{<paquete>}[<fecha> <otra informacion>]
```

La primera línea indica a LaTeX que éste es un archivo para LaTeX 2_ε. La segunda línea especifica que se trata de un paquete, indicando el nombre del mismo (es decir, el nombre del archivo sin extensión) y, opcionalmente, la fecha (en formato `YYYY/MM/DD`) y otra información relevante. Por ejemplo, nuestro paquete `addmath.sty` comienza con las líneas:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{addmath}[1998/09/30 Macros matematicos adicionales (VM)]
```

Si lo que estamos definiendo es una clase, usamos el comando `\ProvidesClass`. Para nuestra clase `mfm2.cls`:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{mfm2}[2002/03/25 Estilo para apuntes MFM II (VM)]
```

A continuación de la identificación vienen los comandos que se desean incorporar a través de este paquete o clase.

Como hemos dicho, `addmath.sty` contiene muchos nuevos comandos matemáticos que consideramos necesario definir mientras escribíamos estos apuntes. Veamos los contenidos de una versión simplificada de dicho paquete:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{addmath}[1998/09/30 Macros matematicos adicionales (VM)]

\newcommand{\prodInt}[2]{\ensuremath \left(\, #1\, |\, #2\, \right ) }
\newcommand{\promedio}[1]{\langle #1 \rangle}
\newcommand{\intii}{\int_{-\infty}^{\infty}}
\newcommand{\grados}{\ensuremath{^\circ}}
\newcommand{\Hipergeometrica}[4]{{}_2F_1\left (#1, #2, #3\, ; #4\right )}
...
```

De este modo, incluyendo en nuestro documento el paquete con `\usepackage{addmath}`, varios nuevos comandos están disponibles:

$(x \mid y)$	`\prodInt{x}{y}`
$\langle x \rangle$	`\promedio{x}`
$\displaystyle\int_{-\infty}^{\infty} dz\, f(z)$	`\intii dz\, f(z)`
$\angle ABC = 90^\circ$	`\angle\, ABC = 90\grados`
${}_2F_1\,(a, b, c\,; d)$	`\Hipergeometrica{a}{b}{c}{d}`

Incluyendo otros paquetes y clases

Los comandos `\RequirePackage` y `\LoadClass` permiten cargar un paquete o una clase, respectivamente.[5]. Esto es de gran utilidad, pues permite construir un nuevo paquete o clase aprovechando la funcionalidad de otros ya existentes.

Así, nuestro paquete `addmath.sty` define bastantes comandos, pero nos gustaría definir varios más que sólo pueden ser creados con las herramientas de $\mathcal{AMS}$-LaTeX. Cargamos entonces en `addmath.sty` el paquete `amsmath` y otros relacionados, y estamos en condiciones de crear más comandos:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{addmath}[1998/09/30 Macros matematicos adicionales (VM)]

\RequirePackage{amsmath}
\RequirePackage{amssymb}
\RequirePackage{euscript}
...
\newcommand{\norma}[1]{\ensuremath \left\lVert\, #1 \,\right\rVert}
\newcommand{\intC}{{\sideset{^*}{}\int}}
\DeclareMathOperator{\senh}{senh}
...
```

Por ejemplo:

$\| x \|$	`\norma{x}`
${}^*\!\int dz\, f(z)$	`\intC dz \, f(z)`
$\operatorname{senh}(2y)$	`\senh (2y)`

La posibilidad de basar un archivo `.sty` o `.cls` en otro es particularmente importante para una clase, ya que contiene una gran cantidad de comandos y definiciones necesarias para compilar el documento exitosamente. Sin embargo, un usuario normal, aun cuando desee definir una nueva clase, estará interesado en modificar sólo parte del comportamiento. Con `\LoadClass`, dicho usuario puede cargar la clase sobre la cual se desea basar, y luego introducir las modificaciones necesarias, facilitando enormemente la tarea.

Por ejemplo, al preparar este documento fue claro desde el comienzo que se necesitaba esencialmente la clase `book`, ya que sería un texto muy extenso, pero también era claro que se requerían ciertas modificaciones. Entonces, en nuestra clase `mfm2.cls` lo primero que hacemos es cargar la clase `book`, más algunos paquetes necesarios (incluyendo nuestro `addmath`), y luego procedemos a modificar o añadir comandos:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{mfm2}[2002/03/25 Estilo para apuntes MFM II (VM)]

\LoadClass[12pt]{book}

\RequirePackage[spanish]{babel}
```

[5]Estos comandos sólo se pueden usar en un archivo `.sty` o `.cls` Para documentos normales, la manera de cargar un paquete es `\usepackage`, y para cargar una clase es `\documentclass`.

```
\RequirePackage{enumerate}
\RequirePackage{addmath}
```

En un archivo `.sty` o un `.cls` se pueden cargar varios paquetes con `\RequirePackage`. `\LoadClass`, en cambio, sólo puede aparecer en un `.cls`, y sólo es posible usarlo una vez (ya que normalmente clases distintas son incompatibles entre sí).

Manejo de opciones

En el último ejemplo anterior, la clase `mfm2` carga la clase `book` con la opción `12pt`. Esto significa que si nuestro documento comienza con `\documentclass{mfm2}`, será compilado de acuerdo a la clase `book`, en 12 puntos. No es posible cambiar esto desde nuestro documento. Sería mejor que pudiéramos especificar el tamaño de letra fuera de la clase, de modo que `\documentclass{mfm2}` dé un documento en 10 puntos, y `\documentclass[12pt]{mfm2}` uno en 12 puntos. Para lograr esto hay que poder pasar opciones desde la clase `mfm2` a `book`.

El modo más simple de hacerlo es con `\LoadClassWithOptions`. Si `mfm2.cls` ha sido llamada con opciones `<opcion1>,<opcion2>, etc.`, entonces `book` será llamada con las mismas opciones. Por tanto, basta modificar en `mfm2.cls` la línea `\LoadClass[12pt]{book}` por:

```
\LoadClassWithOptions{book}
```

`\RequirePackageWithOptions` es el comando análogo para paquetes. Si una clase o un paquete llaman a un paquete `<paquete_base>` y desean pasarle todas las opciones con las cuales han sido invocados, basta indicarlo con:

```
\RequirePackageWithOptions{<paquete_base>}
```

El ejemplo anterior puede ser suficiente en muchas ocasiones, pero en general uno podría llamar a nuestra nueva clase, `mfm2`, con opciones que no tienen nada que ver con `book`. Por ejemplo, podríamos llamarla con opciones `spanish,12pt`. En tal caso, debería pasarle `spanish` a babel, y `12pt` a book. Más aún, podríamos necesitar definir una *nueva* opción, que no existe en ninguna de las clases o paquetes cargados por `book`, para modificar el comportamiento de `mfm2.cls` de cierta manera específica no prevista. Estas dos tareas, discriminar entre opciones antes de pasarla a algún paquete determinado, y crear nuevas opciones, constituyen un manejo más avanzado de opciones. A continuación revisaremos un ejemplo combinado de ambas tareas, extraido de la clase con la cual compilamos este texto, `mfm2.cls`.

La idea es poder llamar a `mfm2` con una opción adicional `keys`, que permita agregar al `dvi` información sobre las etiquetas (dadas con `\label`) de ecuaciones, figuras, etc., que aparezcan en el documento (veremos la utilidad y un ejemplo de esto más adelante). Lo primero es *declarar* una nueva opción, con:

```
\DeclareOption{<opcion>}{<comando>}
```

`<opcion>` es el nombre de la nueva opción a declarar, y `<comando>` es la serie de comandos que se ejecutan cuando dicha opción es especificada.

Así, nuestro archivo `mfm2.cls` debe ser modificado:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{mfm2}[2002/03/25 Estilo para apuntes MFM II (VM)]
...
\DeclareOption{keys}{...}
```

```
...
\ProcessOptions\relax
...
```

Observamos que después de declarar la o las opciones (en este caso `keys`), hay que *procesarlas*, con `\ProcessOptions`.[6]

Las líneas anteriores permiten que `\documentclass{mfm2}` y `\documentclass[keys]{mfm2}` sean ambas válidas, ejecutándose o no ciertos comandos dependiendo de la forma utilizada.

Si ahora queremos que `\documentclass[keys,12pt]{mfm2}` sea una línea válida, debemos procesar `keys` dentro de `mfm2.cls`, y pasarle a `book.cls` las opciones restantes. El siguiente es el código definitivo:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{mfm2}[2002/03/25 Estilo para apuntes MFM II (VM)]
\newif\ifkeys\keysfalse
\DeclareOption{keys}{\keystrue}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{book}}
\ProcessOptions\relax
\LoadClass{book}

\RequirePackage[spanish]{babel}
\RequirePackage{amsmath}
\RequirePackage{theorem}
\RequirePackage{epsfig}
\RequirePackage{ifthen}
\RequirePackage{enumerate}
\RequirePackage{addmath}
\ifkeys\RequirePackage[notref,notcite]{showkeys}\fi

<nuevos comandos de la clase mfm2.cls>
```

Sin entrar en demasiados detalles, digamos que la opción `keys` tiene el efecto de hacer que una cierta variable lógica `\ifkeys`, sea verdadera (cuarta línea del código). La siguiente línea (`\DeclareOption*...`) hace que todas las opciones que no han sido procesadas (`12pt`, por ejemplo) se pasen a la clase `book`. A continuación se procesan las opciones con `\ProcessOptions`, y finalmente se carga la clase `book`.

Las líneas siguientes cargan todos los paquetes necesarios, y finalmente se encuentran todos los nuevos comandos y definiciones que queremos incluir en `mfm2.cls`.

Observemos que la forma particular en que se carga el paquete `showkeys`. Ésa es precisamente la función de la opción `keys` que definimos: `showkeys.sty` se carga con ciertas opciones sólo si se da la opción `keys`.

¿Cuál es su efecto? Consideremos el siguiente texto de ejemplo, en que `mfm2` ha sido llamada sin la opción `keys`:

[6] `\relax` es un comando de TeX que, esencialmente, no hace nada, ni siquiera introduce un espacio en blanco, y es útil incluirlo en puntos críticos de un documento, como en este ejemplo.

```
\documentclass[12pt]{mfm2}
\begin{document}
La opci\'on \verb+keys+ resulta muy \'util cuando tengo objetos numerados
autom\'aticamente, como una ecuaci\'on:
\begin{equation}
  \label{newton}
  \vec F = m \vec a \ .
\end{equation}
y luego quiero referirme a ella: Ec.\  \eqref{newton}.
```

En el primer caso, se ha compilado sin la opción `keys`, y en el segundo con ella. El efecto es que, si se usa un `\label` en cualquier parte del documento, aparece en el margen derecho una caja con el nombre de dicha etiqueta (en este caso, `newton`). Esto es útil para cualquier tipo de documentos, pero lo es especialmente en textos como estos apuntes, muy extensos y con abundantes referencias. En tal caso, tener un modo visual, rápido, de saber los nombres de las ecuaciones sin tener que revisar trabajosamente el archivo fuente es una gran ayuda. Así, versiones preliminares pueden ser compiladas con la opción `keys`, y la versión final sin ella, para no confesar al lector nuestra mala memoria o nuestra comodidad.

Caso 1: `\documentclass[12pt]{mfm2}`

La opción `keys` resulta muy útil cuando tengo objetos numerados automáticamente, como una ecuación:

$$\vec{F} = m\vec{a} \ . \tag{1}$$

y luego quiero referirme a ella: Ec. (1).

Caso 2: `\documentclass[keys,12pt]{mfm2}`

La opción `keys` resulta muy útil cuando tengo objetos numerados automáticamente, como una ecuación:

$$\vec{F} = m\vec{a} \ . \tag{1}$$ `newton`

y luego quiero referirme a ella: Ec. (1).

5.14 Errores y advertencias.

5.14.1 Errores.

Un mensaje de error típico tiene la forma:

```
LaTeX error. See LaTeX manual for explanation.
             Type H <return> for immediate help.
! Environment itemie undefined.
\@latexerr ...or immediate help.}\errmessage {#1}
                                                \endgroup
l.140 \begin{itemie}

?
```

La primera línea nos comunica que LaTeX ha encontrado un error. A veces los errores tienen que ver con procesos más internos, y son encontrados por TeX. Esta línea nos informa quién encontró el error.

La tercera línea comienza con un signo de exclamación. Éste es el indicador del error. Nos dice de qué error se trata.

Las dos líneas siguientes describen el error en términos de comandos de bajo nivel.

La línea 6 nos dice dónde ocurrió el error: la línea 140 en este caso. Además nos informa del texto conflictivo: `\begin{itemie}`.

En realidad, el mensaje nos indica dónde LaTeX advirtió el error por primera vez, que no es necesariamente el punto donde el error se cometió. Pero la gran mayoría de las veces la indicación es precisa. De hecho, es fácil darse cuenta, con la tercera línea
(`Environment itemie undefined`)
y la sexta (`\begin{itemie}`) que el error consistió en escribir `itemie` en vez de `itemize`. La información de LaTeX es clara en este caso y nos dice correctamente qué ocurrió y dónde.

Luego viene un `?`. LaTeX está esperando una respuesta de nosotros. Tenemos varias alternativas. Comentaremos sólo cuatro, típicamente usadas:

(a) `h <Enter>`

Solicitamos ayuda. TeX nos explica brevemente en qué cree él que consiste el error y/o nos da alguna recomendación.

(b) `x <Enter>`

Abortamos la compilación. Deberemos volver al editor y corregir el texto. Es la opción más típica cuando uno tiene ya cierta experiencia, pues el mensaje basta para reconocer el error.

(c) `<Enter>`

Ignoramos el error y continuamos la compilación. TeX hace lo que puede. En algunos casos esto no tiene consecuencias graves y podremos llegar hasta el final del archivo sin mayores problemas. En otros casos, ignorar el error puede provocar que ulteriores comandos —perfectamente válidos en principio— no sean reconocidos y, así, acumular

muchos errores más. Podemos continuar con `<Enter>` sucesivos hasta llegar al final de la compilación.

(d) `q <Enter>`

La acción descrita en el punto anterior puede llegar a ser tediosa o infinita. `q` hace ingresar a TEX en `batchmode`, modo en el cual la compilación prosigue ignorando todos los errores hasta el final del archivo, sin enviar mensajes a pantalla y por ende sin que debamos darle infinitos `<Enter>`.

Las opciones (c) y (d) son útiles cuando no entendemos los mensajes de error. Como TEX seguirá compilando haciendo lo mejor posible, al mirar el `dvi` puede que veamos más claramente dónde comenzaron a ir mal las cosas y, por tanto, por qué.

Como dijimos, LATEX indica exactamente dónde encontró el error, de modo que hemos de ponerle atención. Por ejemplo, si tenemos en nuestro documento la línea:

```
... un error inesperado\fotnote{En cualquier punto.}
puede decidir...
```

generaría el mensaje de error:

```
! Undefined control sequence.

l.249 ...un error inesperado\fotnote
                                    {En cualquier punto.}
?
```

En la línea de localización, LATEX ha cortado el texto justo después del comando inexistente. LATEX no sólo indica la línea en la cual detectó el error, sino el punto de ella donde ello ocurrió. (En realidad, hizo lo mismo —cortar la línea para hacer resaltar el problema— en el caso expuesto en la pág. 182, pero ello ocurrió en medio de comandos de bajo nivel, así que no era muy informativo de todos modos.)

Errores más comunes.

Los errores más comunes son:

a) Comando mal escrito.

b) Paréntesis cursivos no apareados.

c) Uso de uno de los caracteres especiales `#`, `$`, `%`, `&`, `_`, `{`, `}`, `~`, `^`, `\` como texto ordinario.

d) Modo matemático abierto de una manera y cerrado de otra, o no cerrado.

e) Ambiente abierto con `\begin...` y cerrado con un `\end...` distinto.

f) Uso de un comando matemático fuera de modo matemático.

g) Ausencia de argumento en un comando que lo espera.

h) Línea en blanco en ambiente matemático.

Algunos mensajes de error.

A continuación, una pequeña lista de errores (de LaTeX y TeX) en orden alfabético, y sus posibles causas.

`*`

Falta `\end{document}`. (Dar `Ctrl-C` o escribir `\end{document}` para salir de la compilación.)

`! \begin{...} ended by \end{...}`

Error e) de la Sec. 5.14.1. El nombre del ambiente en `\end{...}` puede estar mal escrito, sobra un `\begin` o falta un `\end`.

`! Double superscript` (o `subscript`).

Una expresión como `x^2^3` o `x_2_3`. Si se desea obtener x^{2^3} (x_{2_3}), escribir `{x^2}^3` (`{x_2}_3`).

`! Environment ... undefined.`

`\begin{...}` con un argumento que corresponde a un ambiente no definido.

`! Extra alignment tab has been changed.`

En un `tabular` o `array` sobra un `&`, falta un `\\`, o falta una `c`, `l` ó `r` en el argumento obligatorio.

`! Misplaced alignment tab character &.`

Un `&` aparece fuera de un `tabular` o `array`.

`! Missing $ inserted.`

Errores c), d), f), h) de la Sec. 5.14.1.

`! Missing {` (o `}`) `inserted.`

Paréntesis cursivos no apareados.

`! Missing \begin{document}.`

Falta `\begin{document}` o hay algo incorrecto en el preámbulo.

`! Missing number, treated as zero.`

Falta un número donde LaTeX lo espera: `\hspace{}`, `\vspace cm`, `\setlength{\textwidth}{a}`, etc.

`! Something's wrong -- perhaps a missing \item.`

Posiblemente la primera palabra después de un `\begin{enumerate}` o `\begin{itemize}` no es `\item`.

`! Undefined control sequence.`

Aparece una secuencia `\<palabra>`, donde `<palabra>` no es un comando.

5.14.2 Advertencias.

La estructura de una advertencia de LaTeX es:

`LaTeX warning. <mensaje>.`

Algunos ejemplos:

`Label '...' multiply defined.`

Dos `\label` tienen el mismo argumento.

`Label(s) may have changed. Rerun to get cross-references right.`

Los números impresos por `\ref` y `\pageref` pueden ser incorrectos, pues los valores correspondientes cambiaron respecto al contenido del `aux` generado en la compilación anterior.

`Reference '...' on page ... undefined.`

El argumento de un `\ref` o un `\pageref` no fue definido por un `\label`.

TeX también envía advertencias. Se reconocen porque no comienzan con `TeX warning`. Algunos ejemplos.

`Overfull \hbox ...`

TeX no encontró un buen lugar para cortar una línea, y puso más texto en ella que lo conveniente.

`Overfull \vbox ...`

TeX no encontró un buen lugar para cortar una página, y puso más texto en ella que lo conveniente.

`Underfull \hbox ...`

TeX construyó una línea con muy poco material, de modo que el espacio entre palabras puede ser excesivo.

`Underfull \vbox ...`

TeX construyó una página con muy poco material, de modo que los espacios verticales (entre párrafos) pueden ser excesivos.

Las advertencias de LaTeX siempre deben ser atendidas. Una referencia doblemente definida, o no compilar por segunda vez cuando LaTeX lo sugiere, generará un resultado incorrecto en el `dvi`. Una referencia no definida, por su parte, hace aparecer un signo **??** en el texto final. Todos resultados no deseados, por cierto.

Las advertencias de TeX son menos decisivas. Un overfull o underfull puede redundar en que alguna palabra se salga del margen derecho del texto, que el espaciado entre palabras en una línea sea excesivo, o que el espacio vertical entre párrafos sea demasiado. Los estándares de calidad de TeX son altos, y por eso envía advertencias frecuentemente. Pero generalmente los defectos en el resultado final son imperceptibles a simple vista, o por lo menos no son suficientes para molestarnos realmente. A veces sí, por supuesto, y hay que estar atentos. Siempre conviene revisar el texto y prestar atención a estos detalles, aunque ello sólo tiene sentido al preparar la versión definitiva del documento.

Capítulo 6

Ecuaciones diferenciales de primer orden.

versión 2.0-19 agosto 2002[1]

6.1 Introducción.

Al estudiar los fenómenos físicos, con frecuencia no es posible hallar de inmediato las expresiones explícitas de las magnitudes que caracterizan dicho fenómeno. Sin embargo, suele ser más fácil establecer la dependencia entre esas magnitudes y sus derivadas o sus diferenciales. Así obtenemos ecuaciones que contienen las funciones desconocidas, escalares o vectoriales, bajo el signo de derivadas o de diferenciales.

Las ecuaciones en las cuales la función desconocida, escalar o vectorial, se encuentra bajo el signo de derivada o de diferencial, se llaman *ecuaciones diferenciales*. Veamos algunos ejemplos de ecuaciones diferenciales:

1) $\dfrac{dx}{dt} = -kx$ es la ecuación de la desintegración radioctiva. (k es la constante de desintegración; x es la cantidad de sustancia no desintegrada en el tiempo t; la velocidad de desintegración $\dfrac{dx}{dt}$ es proporcional a la cantidad de sustancia que se desintegra).

2) $m\dfrac{d^2\vec{r}}{dt^2} = \vec{F}\left(t, \vec{r}, \dfrac{d\vec{r}}{dt}\right)$ es la ecuación del movimiento de un punto de masa m, bajo la influencia de una fuerza $\vec{F}$ dependiente del tiempo, de la posición, $\vec{r}$, y de su velocidad $\dfrac{d\vec{r}}{dt}$. La fuerza es igual al producto de la masa por al aceleración.

3) $\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} + \dfrac{\partial^2 u}{\partial z^2} = -4\pi\rho(x, y, z)$ es la ecuación de Poisson, la cual satisface el potencial electroestático $u(x, y, z)$ en presencia de la densidad de carga $\rho(x, y, z)$.

Si se indican los métodos para hallar las funciones incógnitas, determinadas por las ecuaciones diferenciales, se habrá hallado así la dependencia entre las magnitudes indicadas. La

[1]Este capítulo está basado en el primer capítulo del libro: *Ecuaciones diferenciales y cálculo variacional* de L. Elsgoltz, editorial MIR

búsqueda de las funciones desconocidas, determinadas por las ecuaciones diferenciales, es precisamente el problema fundamental de la teoría de ecuaciones diferenciales.

Si en una ecuación diferencial las funciones desconocidas, escalares o vectoriales, son funciones de una sola variable, la ecuación diferencial es llamada *ordinaria* (los dos primeros casos en el ejemplo anterior). Si, en cambio, la función desconocida es función de dos o más variables independientes, la ecuación diferencial se llama *ecuación en derivadas parciales* (el tercer caso en el ejemplo anterior).

Se denomina *orden* de la ecuación diferencial al grado de la derivada (o diferencial) máxima de la función desconocida, que figura en la ecuación.

Se llama *solución* de la ecuación diferencial a una función que, al ser sustituida en la ecuación diferencial, la convierte en una identidad.

Por ejemplo, la ecuación de la desintegración radiactiva

$$\frac{dx}{dt} = -kx \ , \tag{6.1}$$

tiene la solución

$$x(t) = ce^{-kt} \ , \tag{6.2}$$

dode c es una constante arbitraria.

Es evidente que la ecuación diferencial (6.1) aún no determina por completo la ley de desintegración $x = x(t)$. Para su completa determinación hay que conocer la cantidad de sustancia x_0 en un momento inicial t_0. Si x_0 es conocida, entonces tomando en cuenta la condición $x(t_0) = x_0$, de (6.2) hallamos la ley de desintegración radioactiva:

$$x(t) = x_0 e^{-k(t-t_0)} \ .$$

El proceso de determinar las soluciones de una ecuación diferencial se llama *integración* de la misma. En el ejemplo anterior hallamos fácilmente la solución exacta, pero, en casos más complejos, con frecuencia es necesario utilizar métodos aproximados de integración de dichas ecuaciones. Este tema lo abordaremos en la parte III de estos apuntes.

Veamos más detalladamente el problema más complejo mencionado antes, de la determinación de la trayectoria $\vec{r} = \vec{r}(t)$ de un punto material de masa m, bajo la acción de una fuerza dada $\vec{F}(t, \vec{r}, \dot{\vec{r}})$. Según la ecuación de Newton,

$$m\ddot{\vec{r}} = \vec{F}(t, \vec{r}, \dot{\vec{r}}) \ . \tag{6.3}$$

Por lo tanto, el problema se reduce a la integración de esta ecuación diferencial. Es evidente que la ley de movimiento aún no queda determinada por completo si se dan la masa m y la fuerza $\vec{F}$; hay que conocer también la posición inicial del punto

$$\vec{r}(t_0) = \vec{r}_0 \tag{6.4}$$

y su velocidad inicial

$$\dot{\vec{r}}(t_0) = \dot{\vec{r}}_0 \tag{6.5}$$

Obsérvese que la ecuación vectorial (6.3) de segundo orden puede sustituirse por un sistema equivalente de dos ecuaciones vectoriales de primer orden, si consideramos la velociadad $\vec{v}$ como una segunda función vectorial desconocida:

$$\frac{d\vec{r}}{dt} = \vec{v} \ , \qquad m\frac{d\vec{v}}{dt} = \vec{F}(t, \vec{r}, \vec{v}) \ . \tag{6.6}$$

Cada ecuación vectorial en el espacio tridimensional puede ser sustituida, proyectando sobre los ejes de coordenadas, por tres ecuaciones escalares. Por lo tanto, la ecuación (6.3) es equivalente a un sistema de tres ecuaciones escalares de segundo orden, y el sistema (6.6), a un sistema de seis ecuaciones escalares de primer orden.

Finalmente, podemos sustituir una ecuación vectorial (6.3) de segundo orden en el espacio tridimensional por una ecuación vectorial de primer orden en el espacio de seis dimensiones cuyas coordenadas son las tres del vector posición r_x, r_y y r_z, más las tres del vector velocidad v_x, v_y y v_z. Usualmente este espacio es llamado *espacio de fase*. El vector $\vec{R}(t)$ en dicho espacio con coordenadas $(r_x, r_y, r_z, v_x, v_y, v_z)$. En esta notación el sistema (6.6) toma la forma:

$$\frac{d\vec{R}}{dt} = \Phi(t, \vec{R}(t)) \ , \tag{6.7}$$

las proyecciones del vector Φ en el espacio de seis dimensiones son las correspondientes proyecciones de los segundos mienbros del sistema (6.6) en el espacio tridimensional.

Bajo esta interpretación, las condiciones iniciales (6.4) y (6.5) se sustituyen por la condición

$$\vec{R}(t_0) = \vec{R}_0 \ .$$

La solución de la ecuación (6.7) $\vec{R} = \vec{R}(t)$ será una trayectoria en el espacio de fase, en la que cada uno de sus puntos corresponde a cierto estado momentáneo del sistema.

6.2 Ecuaciones de primer orden.

Una ecuación diferencial ordinaria de primer orden, puede escribirse en la forma

$$\frac{dy}{dx} = f(x, y) \ .$$

Un ejemplo simple de tal ecuación,

$$\frac{dy}{dx} = f(x) \ ,$$

se analiza en el curso de cálculo integral. En este caso simple, la solución

$$y = \int f(x)\, dx + c \ ,$$

contiene una constante arbitraria , que puede determinarse si se conoce el valor de $y(x_0) = y_0$; entonces

$$y = y_0 + \int_{x_0}^{x} f(x')\,dx' \ .$$

Más adelante se mostrará que, bajo algunas limitaciones establecidas sobre la función $f(x, y)$, la ecuación

$$\frac{dy}{dx} = f(x, y) \ ,$$

tiene también una solución única, que satisface la condición $y(x_0) = y_0$, y su *solución general*, es decir, el conjunto de soluciones que contienen sin excepción todas las soluciones, depende de una constante arbitraria.

La ecuación diferencial $\dfrac{dy}{dx} = f(x, y)$ establece una dependencia entre las coordenadas de un punto y el coeficiente angular de la tangente $\dfrac{dy}{dx}$ a la gráfica de la solución en ese punto. Conociendo x e y, se puede calcular $\dfrac{dy}{dx}$. Por consiguiente, la ecuación diferencial de la forma considerada determina un campo de direcciones (figura 6.1), y el problema de la integración de la ecuación diferencial se reduce a hallar las llamadas *curvas integrales*, para las cuales la dirección de las tangentes a éstas coincide en cada punto con la dirección del campo.

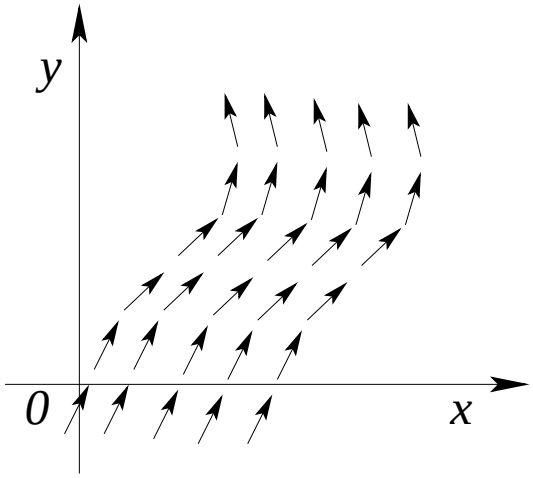


Figura 6.1: Curvas integrales.

Ejemplo Consideremos la siguiente ecuación

$$\frac{dy}{dx} = \frac{y}{x} \ .$$

En cada punto, diferente del punto (0,0), el coeficiente angular de la tangente a la curva integral buscada es igual a la razón $\frac{y}{x}$, o sea, coincide con el coeficiente angular de la recta dirigida desde el origen de coordenadas al mismo punto (x, y). En la figura (6.2) está representado con flechas el campo de direcciones determinado por la ecuación estudiada. Evidentemente, en este caso las curvas integrales serán las rectas $y = cx$, ya que las direcciones de estas rectas coinciden en todas partes con la dirección del campo.

En muchos problemas, en particular en casi todos los problemas de carácter geométricos, las variables x e y son equivalentes. Por ello, en dichos problemas, si éstos se reducen a la resolución de la ecuación diferencial

$$\frac{dy}{dx} = f(x, y) \ , \tag{6.8}$$

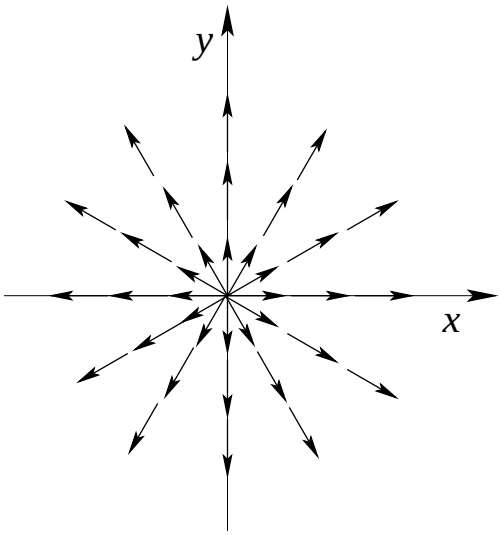


Figura 6.2: Curvas integrales $y = cx$.

es natural considerar, conjuntamente con (6.8), también

$$\frac{dx}{dy} = \frac{1}{f(x,y)} \ . \tag{6.9}$$

Si ambas ecuaciones tienen sentido, entonces son equivalentes, ya que si la función $y = y(x)$ es solución de la ecuación (6.8), la función inversa $x = x(y)$ es solución de (6.9) y, por lo tanto, ambas ecuaciones poseen curvas integrales comunes.

Si, en cambio, en algunos puntos una de las ecuaciones (6.8) o (6.9) pierde sentido, entonces en esos puntos es natural sustituirla por la otra ecuación.

6.3 Ecuaciones con variables separables.

Las ecuaciones diferenciales del tipo

$$f_2(y)\,dy = f_1(x)\,dx \ , \tag{6.10}$$

se llaman *ecuaciones con variables separadas*. Consideremos que las funciones f_1 y f_2 son continuas.

Supongamos que $y(x)$ es solución de esta ecuación; entonces al sustituir $y(x)$ en la ecuación (6.10), obtenemos una identidad que, al ser integrada, da

$$\int f_2(y)\,dy = \int f_1(x)\,dx + c \ , \tag{6.11}$$

donde c es una constante arbitraria.

Hemos obtenido una ecuación en que no figuran ni derivadas ni diferenciales, ecuaciones así se denominan *ecuaciones finitas*, satisfechas por todas las soluciones de la ecuación (6.10).

Además, cada solución de la ecuación (6.11) es solución de (6.10), ya que si una función $y(x)$, al ser sustituida en la ecuación (6.11), la transforma en una identidad, entonces derivando dicha identidad obtenemos que $y(x)$ satisface (6.10).

La ecuación finita $\Phi(x, y) = 0$ que determina la solución $y(x)$ de la ecuación diferencial como función implícita de x, se llama *integral* de la ecuación diferencial considerada.

Si esta ecuación finita determina sin excepción todas las soluciones de la ecuación diferencial dada, entonces se llama *integral general* de dicha ecuación diferencial. Por consiguiente, la ecuación (6.11) es integral general de la ecuación (6.10). Para que (6.11) determine y como función implícita de x, es suficiente exigir que $f_2(y) \neq 0$.

Es posible que en algunos problemas no sea posible expresar las integrales indefinidas $\int f_1(x)\,dx$ y $\int f_2(y)\,dy$ en funciones elementales; pero, a pesar de esto, consideramos resuelto también en este caso el problema de la integración de la ecuación diferencial (6.10), en el sentido de que lo hemos reducido a un problema simple del cálculo de integrales indefinidas. Como el término integral en teoría de ecuaciones diferenciales se utiliza frecuentemente en el sentido de integral de la ecuación diferencial, entonces para referirnos a integrales de funciones, $\int f(x)\,dx$, generalmente se utiliza el término "cuadratura".

Si hay que obtener la solución particular que satisface la condición $y(x_0) = y_0$, ésta evidentemente se determina por la ecuación

$$\int_{y_0}^{y} f_2(y)\,dy = \int_{x_0}^{x} f_1(x)\,dx \ ,$$

la cual se obtiene de

$$\int_{y_0}^{y} f_2(y)\,dy = \int_{x_0}^{x} f_1(x)\,dx + c \ ,$$

utilizando las condiciones iniciales $y(x_0) = y_0$.

Ejemplo Considere la siguiente ecuación

$$xdx + ydy = 0 \ .$$

Las variables están separadas, ya que el coeficinete de dx es función sólo de x, y el coeficiente de dy, sólo de y. Integrando, obtenemos

$$\int x\,dx + \int y\,dy = c \quad \text{o bien } x^2 + y^2 = c_1^2 \ ,$$

que es una familia de circunferencias con centro en el origen de coordenadas.

Las ecuaciones del tipo

$$\phi_1(x)\psi_1(y)\,dx = \phi_2(x)\psi_2(y)\,dy \ ,$$

en las cuales los coeficientes de las diferenciales se descomponen en factores dependientes sólo de x o de y, se llaman *ecuaciones diferenciales con variables separables*, ya que dividiendo entre $\psi_1(y)\phi_2(x)$, éstas se reducen a una ecuación de variables separadas:

$$\frac{\phi_1(x)}{\phi_2(x)}\,dx = \frac{\psi_2(y)}{\psi_1(y)}\,dy \ ,$$

Obsérvese que la división entre $\psi_1(y)\phi_2(x)$ puede conducir a la pérdida de soluciones particulares, que reducen a cero el producto $\psi_1(y)\phi_2(x)$; si las funciones $\psi_1(y)$ y $\phi_2(x)$ pueden ser discontinuas, es posible la aparición de soluciones superfluas, que reducen a cero el factor

$$\frac{1}{\psi_1(y)\phi_2(x)} \ .$$

Ejemplo Como ya fue mencionado, se ha establecido que la velocidad de desintegración radioactiva es proporcional a la cantidad x de sustancia aún no desintegrada. Hallar la dependencia de x respecto al tiempo t, si en el momento inicial para $t = t_0$, era $x = x_0$.

Supondremos conocido el coeficiente de proporcionalidad k, llamado constante de desintegración. La ecuación diferencial que gobierna el proceso tendrá la forma

$$\frac{dx}{dt} = -kx \ ,$$

El signo menos en el término derecho de la ecuación indica que x decrece cuando t aumenta, $k > 0$. Separando variables e integrando, se obtiene

$$\frac{dx}{x} = -k\,dt \ ; \quad \ln(|\,x\,|) - \ln(|\,x_0\,|) = k\,(t - t_0) \ ,$$

de donde

$$x = x_0\, e^{-k(t-t_0)} \ .$$

Determinemos también el período de desintegración τ (o sea, el tiempo durante el cual se desintegra $x_0/2$). Haciendo $t - t_0 = \tau$, obtenemos $x_0/2 = x_0 e^{-k\tau}$, de donde $\tau = \ln 2/k$.

La ecuación

$$\frac{dx}{dt} = kx \ , \quad k > 0 \ ,$$

se diferencia sólo en el signo del segundo miembro de la ecuación anterior, sin embargo, describe procesos de "reproducción" completamente diferentes, por ejemplo: la variación de la cantidad de neutrones en las reacciones en nucleares en cadena, o la reproducción de una colonia de bacterias en condiciones ideales. La solución de esta ecuación que satisface la condición inicial $x(t_0) = x_0$ tiene la forma

$$x = x_0\, e^{k(t-t_0)} \ ,$$

y, a diferencia de las soluciones anteriores, $x(t)$ no disminuye, sino que crece exponencialmente con el incremento de t

6.4 Ecuaciones que se reducen a ecuaciones de variables separables

Muchas ecuaciones diferenciales pueden ser reducidas a ecuaciones con variables separables mediante una sustitución de variables. A dicho grupo pertenecen, por ejemplo, las ecuaciones

de la forma

$$\frac{dy}{dx} = f(ax + by) \ ,$$

donde a y b son magnitudes constantes, las cuales se transforman en ecuaciones con variables separables por medio de la sustitución $z = ax + by$. Efectivamente, pasando a las nuevas variables x y z, tendremos

$$\frac{dz}{dx} = a + b\frac{dy}{dx} \ , \quad \frac{dz}{dx} = a + bf(z) \ ,$$

o bien

$$\frac{dz}{a + bf(z)} = dx \ ,$$

con lo que hemos separado las variables. Integrando, obtenemos

$$x = \int \frac{dz}{a + bf(z)} + c \ .$$

Ejemplo Consideremos la siguiente ecuación diferencial

$$\frac{dy}{dx} = \frac{1}{x - y} + 1 \ .$$

Haciendo $x - y = z$, obtenemos

$$\frac{dy}{dx} = 1 - \frac{dz}{dx} \quad 1 - \frac{dz}{dx} = \frac{1}{z} + 1 \ ;$$

$$\frac{dz}{dx} = -\frac{1}{z} \ , \quad zdz = -dx \ , \quad z^2 = -2x + c \ , \quad (x - y)^2 = -2x + c \ .$$

A ecuaciones con variables separables se reducen también las *ecuaciones diferenciales homogéneas de primer orden*, que tienen la forma

$$\frac{dy}{dx} = f\left(\frac{y}{x}\right) \ .$$

En efecto, después de la sustitución $z = \dfrac{y}{x}$, o bien $y = xz$, obtenemos

$$\frac{dy}{dx} = x\frac{dz}{dx} + z \ , \quad x\frac{dz}{dx} + z = f(z) \ , \quad \frac{dz}{f(z) - z} = \frac{dx}{x} \ ,$$

$$\int \frac{dz}{f(z) - z} = \ln |\, x \,| + \ln c \ , \quad x = c\, e^{\int \frac{dz}{f(z) - z}} \ .$$

Obsérvese que el segundo miembro de la ecuación homogénea es un función homogénea de variables x e y de grado nulo[2] de homogeneidad; por eso la ecuación del tipo

$$M(x,y)\,dx + N(x,y)\,dy = 0\ ,$$

será homogénea si $M(x,y)$ y $N(x,y)$ son funciones homegéneas de x e y, del mismo grado de homogeneidad, puesto que en este caso

$$\frac{dy}{dx} = -\frac{M(x,y)}{N(x,y)} = f\left(\frac{y}{x}\right)\ .$$

Ejemplo Considere la siguiente ecuación

$$\frac{dy}{dx} = \frac{y}{x} + \tan\frac{y}{x}\ .$$

Haciendo $y = xz$, $\dfrac{dy}{dx} = x\dfrac{dz}{dx} + z$ y sustituyendo en la ecuación inicial, obtenemos

$$x\frac{dz}{dx} + z = z + \tan z\ , \quad \frac{\cos z\,dz}{\operatorname{sen} z} = \frac{dx}{x}\ ,$$

$$\ln|\operatorname{sen} z| = \ln|x| + \ln c\ , \quad \operatorname{sen} z = cx \quad \operatorname{sen}\frac{y}{x} = cx\ .$$

Las ecuaciones del tipo

$$\frac{dy}{dx} = f\left(\frac{a_1x + b_1y + c_1}{a_2x + b_2y + c_2}\right)\ , \tag{6.12}$$

pueden reducirse a ecuaciones homogéneas, si trasladamos el origen de coordenadas al punto de intersección (x_1, y_1) de las rectas

$$a_1x + b_1y + c_1 = 0\ , \qquad a_2x + b_2y + c_2 = 0\ .$$

Efectivamente, el miembro independiente en las ecuaciones de estas rectas en las nuevas coordenadas $X = x - x_1$, $Y = y - y_1$, será igual a cero; los coeficientes de las coordenadas permanecen invariables; $\dfrac{dy}{dx} = \dfrac{dY}{dX}$, y la ecuación (6.12) toma la forma

$$\frac{dY}{dX} = f\left(\frac{a_1X + b_1Y}{a_2X + b_2Y}\right)\ ,$$

o bien

$$\frac{dY}{dX} = f\left(\frac{a_1 + b_1\frac{Y}{X}}{a_2 + b_2\frac{Y}{X}}\right) = \varphi\left(\frac{Y}{X}\right)\ ,$$

[2]Una función $f(u,v)$ es homogénea de grado n si al multiplicar las variables por λ la función resultante es λ^n veces la función original.

que ya es una ecuación homogénea.

Este método no se puede aplicar sólo en el caso en que haya paralelismo entre las rectas $a_1x + b_1y + c_1 = 0$ y $a_2x + b_2y + c_2 = 0$. Pero en este caso los coeficientes de las coordenadas son proporcionales: $\dfrac{a_2}{a_1} = \dfrac{b_2}{b_1} = k$, y la ecuación (6.12) se puede escribir en la forma

$$\frac{dy}{dx} = f\left(\frac{a_1x + b_1y + c_1}{k(a_1x + b_1y) + c_2}\right) = F(a_1x + b_1y) \ ,$$

como ya sabemos el cambio de variable $z = a_1x + b_1y$ transforma la ecuación considerada en una ecuación con variables separables.

Ejemplo Considere la siguiente ecuación

$$\frac{dy}{dx} = \frac{x - y + 1}{x + y - 3} \ .$$

Resolviendo el sistema de ecuaciones $x - y + 1 = 0$, $x + y - 3 = 0$, obtenemos $x_1 = 1$, $y_1 = 2$. Haciendo $x = X + 1$, $y = Y + 2$, tendremos

$$\frac{dY}{dX} = f\left(\frac{X - Y}{X + Y}\right) \ .$$

El cambio de variable $Y = zX$ conduce a una ecuación de variables separables:

$$z + X\frac{dz}{dX} = \frac{1 - z}{1 + z} \ ,$$

$$\frac{(1 + z)\, dz}{1 - 2z + z^2} = \frac{dX}{X} \ ,$$

$$-\frac{1}{2}\ln\left|\, 1 - 2z - z^2 \,\right| = \ln|\, X \,| - \frac{1}{2}\ln c \ ,$$

$$(1 - 2z - z^2)X^2 = c \ ,$$

$$X^2 - 2XY - Y^2 = c \ ,$$

$$x^2 - 2xy - y^2 + 2x + 6y = c_1 \ .$$

6.5 Ecuaciones lineales de primer orden.

Se llama *ecuación diferencial de primer orden* a una ecuación lineal con respecto a la función desconocida y a su derivada. La ecuación lineal tiene la forma

$$\frac{dy}{dx} + p(x)y = f(x) \ , \tag{6.13}$$

donde $p(x)$ y $f(x)$ se considerarán en lo sucesivo funciones continuas de x en la región en que se exige integrar la ecuación (6.13).

Si $f(x) \equiv 0$, la ecuación (6.13) se llama *lineal homogénea.* En la ecuación lineal homogénea las variables se separan

$$\frac{dy}{dx} + p(x)y = 0 \ , \quad \text{de donde } \frac{dy}{y} = -p(x)\,dx \ ,$$

e integrando, obtenemos

$$\ln|\,y\,| = -\int p(x)\,dx + \ln c_1 \ , \quad c_1 > 0 \ ,$$

$$y = c\,e^{-\int p(x)\,dx} \ , \quad c \neq 0 \ . \tag{6.14}$$

Al dividir entre y se perdió la solución $y = 0$; sin embargo, ésta puede ser incluida en la familia de soluciones halladas (6.14), si se considera que c puede tomar el valor 0.

Para integrar la ecuación lineal no homogénea

$$\frac{dy}{dx} + p(x)y = f(x) \ . \tag{6.13}$$

puede ser aplicado el así llamado *método de variación de la constante.* Al aplicar dicho método, primeramente se integra la ecuación homogénea correspondiente (o sea, la que tiene el mismo primer miembro):

$$\frac{dy}{dx} + p(x)y = 0 \ ,$$

cuya solución general, como fue indicado anteriormente, tiene la forma

$$y = c\,e^{-\int p(x')\,dx'} \ .$$

Cuando c es constante, la función $c\,e^{-\int p(x')\,dx'}$ es la solución de la ecuación homogénea. Probemos ahora satisafacer la ecuación no homegénea considerando c como función de x, o sea, realizando en esencia la sustitución de variables

$$y = c(x)\,e^{-\int p(x')\,dx'} \ ,$$

donde $c(x)$ es una función desconocida de x.

Calculando la derivada

$$\frac{dy}{dx} = \frac{dc}{dx}\, e^{-\int p(x')\,dx'} - c(x)p(x)\, e^{-\int p(x')\,dx'} \ ,$$

y sustituyéndola en la ecuación no homogénea inicial (6.13), se obtiene

$$\frac{dc}{dx}\, e^{-\int p(x')\,dx'} - c(x)p(x)\, e^{-\int p(x')\,dx'} + p(x)c(x)\, e^{-\int p(x')\,dx'} = f(x) \ ,$$

o bien

$$\frac{dc}{dx} = f(x)\, e^{\int p(x')\,dx'} \ ,$$

de donde, integrando, se halla

$$c(x) = \int f(x'')e^{\int p(x')\,dx'}\,dx'' + c_1 \ ,$$

y, por consiguiente,

$$y = c(x)\, e^{-\int p(x')\,dx'} = c_1\, e^{-\int p(x')\,dx'} + e^{-\int p(x')\,dx'} \int f(x'')e^{\int p(x')\,dx'}\,dx'' \ . \tag{6.15}$$

De este modo, la solución general de la ecuación lineal no homogénea es igual a la suma de la solución general de la ecuación homogénea correspondiente

$$c_1\, e^{-\int p(x')\,dx'} \ ,$$

y de la solución particular de la ecuación no homogénea

$$e^{-\int p(x')\,dx'} \int f(x'')e^{\int p(x')\,dx'}\,dx'' \ ,$$

que se obtiene de (6.15) si $c_1 = 0$.

Obsérvese que en ejemplos concretos no es conveniente utilizar la fórmula (6.15) , compleja y dificil de recordar; es más sencillo repetir cada vez todas las operaciones expuestas más arriba.

Ejemplo En un circuito eléctrico con autoinducción tiene lugar el paso de corriente alterna. La tensión U es un función dada del tiempo, $U = U(t)$, la resistencia R y la autoinducción L son constantes; la corriente inicial es dada, $I(0) = I_0$. Hallar la dependencia de la intensidad de la corriente $I = I(t)$ respecto al tiempo. Aplicando la ley de Ohm para el cirrcuito obtenemos

$$U - L\frac{dI}{dt} = RI \ .$$

La solución de esta ecuación lineal que satisface la condición inicial $I(0) = I_0$, de acuardo con (6.15), tiene la forma

$$I = e^{-\frac{R}{L}t}\left[I_0 + \frac{1}{L}\int_0^t U(t')e^{\frac{R}{L}t'}\,dt'\right] \ . \tag{6.16}$$

Para una tensión constante $U = U_0$, obtenemos

$$I = \frac{U_0}{R} + \left(I_0 - \frac{U_0}{R} \right) e^{-\frac{R}{L}t} \ .$$

Es interesante el caso de tensión alterna sinusoidal: $U = A \operatorname{sen} \omega t$. En este caso, según (6.16), obtenemos

$$I = e^{-\frac{R}{L}t} \left[I_0 + \frac{A}{L} \int_0^t \operatorname{sen} \omega t' e^{\frac{R}{L}t'} \right] \ .$$

La integral del segundo miembro se toma fácilmente.

Muchas ecuaciones diferenciales pueden ser reducidas a ecuaciones lineales mediante un cambio de variables. Por ejemplo, la *ecuación de Bernouilli*, que tiene la forma

$$\frac{dy}{dx} + p(x)y = f(x)y^n \ , \quad n \neq 1 \ ,$$

o bien

$$y^{-n}\frac{dy}{dx} + p(x)y^{1-n} = f(x) \ , \tag{6.17}$$

con el cambio de variable $y^{1-n} = z$, se reduce a una ecuación lineal. Efectivamente, derivando $y^{1-n} = z$, hallamos

$$(1-n)y^{-n}\frac{dy}{dx} = \frac{dz}{dx} \ ,$$

y sustituyendo en (6.17), obtenemos la ecuación lineal

$$\frac{1}{1-n}\,\frac{dz}{dx} + p(x)\,z = f(x) \ .$$

Ejemplo Consideremos la ecuación

$$\frac{dy}{dx} = \frac{y}{2x} + \frac{x^2}{2y} \ ,$$

o bien

$$2y\frac{dy}{dx} = \frac{y^2}{x} + x^2 \ ,$$

con el cambio de variable $y^2 = z$ tenemos $2y\dfrac{dy}{dx} = \dfrac{dz}{dx}$ con lo que la ecuación nos queda

$$\frac{dz}{dx} = \frac{z}{x} + x^2 \ ,$$

Que es una ecuación diferencial lineal de primer orden no homogénea.

La ecuación

$$\frac{dy}{dx} + p(x)y + q(x)y^2 = f(x) \ ,$$

llamada *ecuación de Riccati*, en general no se integra en cuadraturas, pero por sustitución de variables puede ser transformada en una ecuación de Bernoulli, si se conoce una solución particular $y_1(x)$ de esta ecuación. Efectivamente, haciendo $y = y_1 + z$, se obtiene

$$y_1' + z' + p(x)(y_1 + z) + q(x)(y_1 + z)^2 = f(x) \ ,$$

o, como $y_1' + p(x)y_1 + q(x)y_1^2 \equiv f(x)$, tendremos la ecuación de Bernoulli

$$z' + [p(x) + 2q(x)y_1]z + q(x)z^2 = 0 \ .$$

Ejemplo Consideremos la ecuación

$$\frac{dy}{dx} = y^2 - \frac{2}{x^2} \ .$$

Aquí no es dificil hallar la solución particular $y_1 = \frac{1}{x}$. Haciendo $y = z + frac1x$, obtenemos

$$y' = z' + \frac{1}{x^2} \ , \quad z' - \frac{1}{x^2} = \left(z - \frac{1}{x2}\right)^2 - \frac{2}{x^2} \ ,$$

o bien

$$z' = z^2 + 2\frac{z}{x} \ ,$$

que es una ecuación de Bernoulli.

$$\frac{z'}{z^2} = \frac{2}{xz} + 1 \ , \quad u = \frac{1}{z} \ , \quad \frac{du}{dx} = -\frac{z'}{z^2} \ ,$$

$$\frac{du}{dx} = -\frac{2u}{x} - 1 \ , \quad \frac{du}{u} = -\frac{2dx}{x} \ ,$$

$$\ln|\,u\,| = -2\ln|\,x\,| + \ln c \ , \quad u = \frac{c}{x^2} \ , \quad u = \frac{c(x)}{x^2} \ ,$$

$$\frac{c'(x)}{x^2} = -1 \ , \quad c(x) = -\frac{x^3}{3} + c_1 \ ,$$

$$u = \frac{c_1}{x^2} - \frac{x}{3} \ , \quad \frac{1}{z} = \frac{c_1}{x^2} - \frac{x}{3} \ , \quad \frac{1}{y - \frac{1}{x}} = \frac{c_1}{x^2} - \frac{x}{3} \ ,$$

$$y = \frac{1}{x} + \frac{3x^2}{c_2 - x^3} \ .$$

6.6 Ecuaciones en diferenciales totales.

Puede suceder que el primer miembro de la ecuación diferencial

$$M(x,y)\,dx + N(x,y)\,dy = 0\ , \tag{6.18}$$

sea la diferencial total de cierta función $u(x,y)$:

$$du(x,y) = M(x,y)\,dx + N(x,y)\,dy\ ,$$

y que por consiguiente, la ecuación (6.18) tome la forma

$$du(x,y) = 0\ .$$

Si la función $y(x)$ es solución de la ecuación (6.18), entonces

$$u(x,y(x)) = c\ , \tag{6.19}$$

donde c es una constante. recíprocamente, si cierta función $y(x)$ convierte en identidad la ecuación finita (6.19), entonces, derivando la identidad obtenida, tendremos $du(x,y(x)) = 0$ y, por consiguiente, $u(x,y) = c$, donde c es una constante arbitraria, es integral general de la ecuación inicial.

Si las condiciones iniciales $y(x_0) = y_0$ están dadas, la constante c se determina de (6.19): $c = u(x_0,y_0)$ y

$$u(x,y(x)) = u(x_0,y_0)\ , \tag{6.20}$$

es la integral particular buscada. Si $\dfrac{\partial u}{\partial y} = N(x,y) \neq 0$ en el punto (x_0,y_0), entonces la ecuación (6.20) determina y como función implícita de x.

para que el primer miembro de la ecuación (6.18)

$$M(x,y)\,dx + N(x,y)\,dy\ ,$$

sea un diferencial total de cierta función $u(x,y)$, como se sabe, es necesario y suficiente que

$$\frac{\partial M(x,y)}{\partial y} \equiv \frac{\partial N(x,y)}{\partial x}\ . \tag{6.21}$$

Si esta condición, señalada por primera vez por Euler, se cumple, entonces (6.18) se integra fácilmente. En efecto

$$du = M\,dx + N\,dy\ .$$

Por otra parte,

$$du = \frac{\partial u}{\partial x}\,dx + \frac{\partial u}{\partial y}\,dy\ .$$

Por consiguiente,

$$\frac{\partial u}{\partial x} = M(x,y)\ ; \qquad \frac{\partial u}{\partial y} = N(x,y)\ ,$$

de donde

$$u(x,y) = \int M(x,y)\,dx + c(y)\ .$$

Al calcular la integral de la expresión anterior, la magnitud y se considera constante; por eso, $c(y)$ es una función arbitraria de y.

PAradeterminar la función $c(y)$, derivamos la función hallada $u(x,y)$ respecto a y y, como $\dfrac{\partial u}{\partial y} = N(x,y)$, obtenemos

$$\frac{\partial}{\partial y}\left(\int M(x,y)\,dx\right) + c'(y) = N(x,y)\ .$$

De esta ecuación se determina $c'(y)$, e integrando se halla $c(y)$.

Se puede determinar aún más fácilmente la función $u(x,y)$ por su diferencial total $du = M(x,y)\,dx + N(x,y)\,dy$, tomando la integral curvilínea de $M(x,y)\,dx + N(x,y)\,dy$ desde cierto punto fijo (x_0, y_0) hasta un punto con coordenadas variables (x,y), por cualquier camino:

$$u(x,y) = \int_{(x_0,y_0)}^{(x,y)} M(x,y)\,dx + N(x,y)\,dy\ .$$

Con frecuencia, es cómodo tomar una línea quebrada, compuesta por dos segmentos paralelos

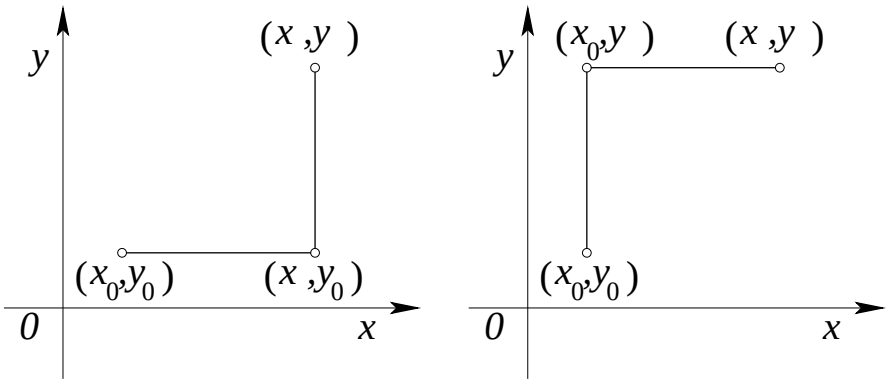


Figura 6.3: Caminos de integración posibles.

alos ejes de coordenadas (figura 6.3); en este caso

$$\int_{(x_0,y_0)}^{(x,y)} M\,dx + N\,dy = \int_{(x_0,y_0)}^{(x,y_0)} M\,dx + \int_{(x,y_0)}^{(x,y)} N\,dy\ ,$$

o bien

$$\int_{(x_0,y_0)}^{(x,y)} M\,dx + N\,dy = \int_{(x_0,y_0)}^{(x_0,y)} N\,dy + \int_{(x_0,y_0)}^{(x,y)} N\,dy\ .$$

Ejemplo Consideremos la siguiente ecuación

$$(x+y+1)\,dx + (x-y^2+3)\,dy = 0\ .$$

El primer miembro de la ecuación es la diferencial total de cierta función $u(x,y)$, puesto que

$$\frac{\partial(x+y+1)}{\partial y} \equiv \frac{\partial(x-y^2+3)}{\partial x}\ ,$$

$$\frac{\partial u}{\partial x} = x+y+1 \quad u = \frac{x^2}{2} + xy + x + c(y)\ ,$$

$$\frac{\partial u}{\partial y} = x + c'(y)\ , \quad x + c'(y) = x - y^2 + 3\ ,$$

$$c'(y) = -y^2+3\ , \quad c(y) = -\frac{y^3}{3} + 3y + c_1\ .$$

Por lo tanto, la integral general tiene la forma

$$3x^2 + 6xy + 6x - 2y^3 + 18y = c_2 \tag{6.22}$$

Se puede utilizar también el otro método de determinación de la función $u(x,y)$:

$$u(x,y) = \int_{(x_0,y_0)}^{(x,y)} (x+y+1)\,dx + (x-y^2+3)\,dy\ .$$

Como punto inicial (x_0,y_0) escogemos por ejemplo el origen de coordenadas, y como camino de integración el mostrado en la figura 6.4). Entonces

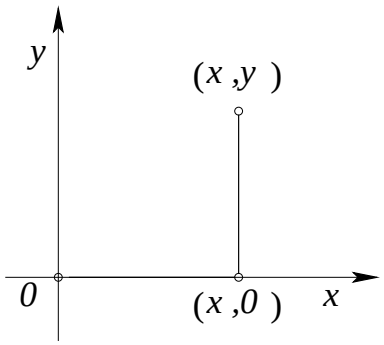


Figura 6.4: Camino de integración.

$$u(x,y) = \int_{(0,0)}^{(x,0)} (x+1)\,dx + \int_{(x,0)}^{(x,y)} (x-y^2+3)\,dy\ ,$$

$$u(x,y) = \frac{x^2}{2} + x + xy - \frac{y^3}{3} + 3y\ ,$$

y la integral general tiene la forma

$$\frac{x^2}{2} + x + xy - \frac{y^3}{3} + 3y = c \ ,$$

o bien como en (6.22).

En algunos casos, si el primer miembro de la ecuación

$$M(x,y)\,dx + N(x,y)\,dy = 0 \ , \tag{6.18}$$

no es una diferencial total, resulta fácil escoger una función $\mu(x,y)$,luego del producto por la cual el primer miembro de (6.18) se transforma en una diferencial total:

$$du = \mu M\,dx + \mu N\,dy \ .$$

Esta función μ se llama *factor integrante*. Observese que la multiplicación por el factor integrante $\mu(x,y)$ puede conducir a que aparezcan soluciones particulares superfluas, que reducen este factor a cero.

Ejemplo Consideremos la siguiente ecuación

$$x\,dx + y\,dy + (x^2+y^2)x^2\,dx = 0 \ .$$

es fácil comprobar que después de multiplicar por el factor $\mu = 1/(x^2+y^2)$, el primer miembro se transforma en diferencial total. Efectivamente, luego del producto por $\mu = 1/(x^2+y^2)$, obtenemos

$$\frac{x\,dx + y\,dy}{x^2+y^2} + x^2\,dx = 0 \ ,$$

o integrando:

$$\frac{1}{2}\ln(x^2+y^2) + \frac{x^3}{3} = \ln c_1 \ .$$

multiplicando por 2 y potenciando, tendremos

$$(x^2+y^2)\,e^{2x^3/3} = c \ .$$

Es claro que no siempre el factor integrante se escoge tan facilmente. En general, para hallar dicho factor es necesario escoger por lo menos una solución particular no idénticamente nula de la ecuacion en derivadas parciales

$$\frac{\partial \mu M}{\partial y} = \frac{\partial \mu N}{\partial x} \ ,$$

o, en forma desarrollada,

$$\frac{\partial \mu}{\partial y}M + \mu\frac{\partial M}{\partial y} = \frac{\partial \mu}{\partial x}N + \mu\frac{\partial N}{\partial x} \ ,$$

la cual, después de dividir entre μ y de cambiar de miembro algunos términos, se reduce a

$$\frac{\partial \ln \mu}{\partial y} M - \frac{\partial \ln \mu}{\partial x} N = \frac{\partial N}{\partial x} - \frac{\partial M}{\partial y} \ . \tag{6.23}$$

En general, la integración de esta ecuación en derivadas parciales no es un problema más simple que la integración de la ecuación inicial. Sin embargo, a veces la elección de la solución particular de (6.23) no presenta dificultades.

Aparte de ello, considerando que el factor integrante es función de un solo argumento (por ejemplo, es función sólo de $x+y$ o de x^2+y^2, o función sólo de x o de y, etc.), se puede integrar ya sin dificultad la ecuación (6.23) e indicar las condiciones bajo las cuales existe un factor integrante del tipo considerado. Con esto se obtienen clases de ecuaciones para las cuales el factor integrante puede ser hallado fácilmente.

Por ejemplo, encontremos las condiciones bajo las cuales la ecuación $M\,dx + N\,dy = 0$ tiene factor integrante que depende sólo de x, $\mu = \mu(x)$. En este caso, la ecuación (6.23) se simplifica y toma la forma

$$-\frac{d \ln \mu}{dx} N = \frac{\partial N}{\partial x} - \frac{\partial M}{\partial y} \ ,$$

de donde, considerando $\dfrac{\frac{\partial M}{\partial y} - \frac{\partial N}{\partial x}}{N}$ función continua de x, obtenemos

$$\ln \mu = \int \frac{\frac{\partial M}{\partial y} - \frac{\partial N}{\partial x}}{N}\, dx + \ln c \ ,$$

$$\mu = c \exp\left(\int \frac{\frac{\partial M}{\partial y} - \frac{\partial N}{\partial x}}{N}\, dx \right) \ . \tag{6.24}$$

Se puede considerar $c = 1$, ya que es suficoiente tener sólo un factor integrante.

Si $\dfrac{\frac{\partial M}{\partial y} - \frac{\partial N}{\partial x}}{N}$ es función sólo de x, entonces existe un factor integrante que sólo depende de x, y es igual a (6.24). En caso contrario , no existe ningún factor de la forma $\mu(x)$.

La condición de existencia de un factor integrante que depende sólo de x se cumple, por ejemplo, para la ecuación lineal

$$\frac{dy}{dx} + p(x)y = f(x) \ , \qquad \text{o bien } \ [p(x)y - f(x)]\, dx + dy = 0 \ .$$

Efectivamente, $\dfrac{\frac{\partial M}{\partial y} - \frac{\partial N}{\partial x}}{N} = p(x)$ y, por lo tanto, $\mu = e^{\int p(x)\,dx}$. De manera análoga se pueden hallar las condiciones de existencia de factores integrantes de la forma

$$\mu(y) \ , \ \mu(x \pm y) \ , \ \mu(x^2 \pm y^2) \ , \ \mu(xy) \ , \ \mu\left(\frac{x}{y}\right) \text{ etc.}$$

Ejemplo ¿Tiene la ecuación

$$x\,dx + y\,dy + x\,dy - y\,dx = 0\ , \tag{6.25}$$

un factor integrante de la forma $\mu = \mu(x^2 + y^2)$?

Designemos $x^2 + y^2 = z$. La ecuación (6.23) para $\mu = \mu(x^2 + y^2) = \mu(z)$ toma la forma

$$2(My - Nx)\frac{\ln \mu}{dz} = \frac{\partial N}{\partial x} - \frac{\partial M}{\partial y}\ ,$$

de donde

$$\ln |\,\mu\,| = \frac{1}{2}\int \varphi(z)\,dz + \ln c\ ,$$

o bien

$$\mu = c\,\exp\left(\frac{1}{2}\int \varphi(z)\,dz\right)\ , \tag{6.26}$$

donde

$$\varphi(z) = \frac{\dfrac{\partial N}{\partial x} - \dfrac{\partial M}{\partial y}}{My - Nx}\ .$$

Para la existencia de un factor integrante del tipo dado, es necesario (y en caso que $\varphi(z)$ sea continua, suficiente) que φ sea función sólo de $x^2 + y^2$. En nuestro caso,

$$\frac{\dfrac{\partial N}{\partial x} - \dfrac{\partial M}{\partial y}}{My - Nx} = -\frac{2}{x^2 + y^2}\ ;$$

por lo tanto, el factor integrante $\mu = \mu(x^2 + y^2)$ existe y es igual a (6.26). Para $c = 1$, obtenemos:

$$\mu = \exp\left(-\int \frac{dz}{z}\right) = \frac{1}{z} = \frac{1}{x^2 + y^2}\ .$$

Multiplicando la ecuación (6.25) por el μ que determinamos, la reducimos a la forma

$$\frac{x\,dx + y\,dy}{x^2 + y^2} + \frac{x\,dy - y\,dx}{x^2 + y^2} = 0\ ,$$

o bien

$$\frac{\dfrac{1}{2}d(x^2 + y^2)}{x^2 + y^2} + \frac{\left(d\dfrac{y}{x}\right)}{1 + \left(\dfrac{y}{x}\right)^2} = 0\ , \quad \frac{1}{2}d\ln(x^2 + y^2) + d\arctan\frac{y}{x} = 0\ .$$

Integrando, obtenemos

$$\ln\sqrt{x^2 + y^2} = -\arctan\frac{y}{x} + \ln c\ , \quad \sqrt{x^2 + y^2} = c\,\exp\left(-\arctan\frac{y}{x}\right)\ ,$$

o bien en coordenadas polares $\rho = c\,e^{-\varphi}$, que es una familia de espirales logarítmicas.

6.7 Teoremas.

Teorema 6.1 Existencia y unicidad de la solución.
Si en la ecuación

$$\frac{dy}{dx} = f(x, y) \ , \tag{6.27}$$

la función $f(x, y)$ es continua en el rectángulo D:

$$x_0 - a \leq x \leq x_0 + a \ , \quad y_0 - b \leq y \leq y_0 + b \ , \tag{6.28}$$

y satisface en D la condición de Lipschitz:

$$| \, f(x, y_1) - f(x, y_2) \, | \leq N \, | \, y_1 - y_2 \, | \ ,$$

donde N es una constante, entonces existe una solución única $y = y(x)$, $x_0 - H \leq x \leq x_0 + H$ de la ecuación (6.27), que satisface la condición $y(x_0) = y_0$, donde

$$H < \min \left(a \ , \ \frac{b}{M} \ , \ \frac{1}{N} \right)$$
$$M = \max | \, f(x, y) \, | \ \text{ en } D.$$

Teorema 6.2 Sobre la dependencia continua de la solución con respecto al parámetro y a los valores iniciales.
Si el segundo mienbro de la ecuación diferencial

$$\frac{dy}{dx} = f(x, y, \mu) \ , \tag{6.29}$$

es continuo en μ para $\mu_0 \leq \mu \leq \mu_1$ y satisface la condiciones del teorema de existencia y unicidad, y la constante de Lipschitz N no depende de μ, entonces la solución $y(x, y)$ de la ecuación considerada que satisface la condición $y(x_0) = y_0$ depende en forma continua de μ.

Teorema 6.3 Sobre la derivabilidad de las soluciones.
Si en un entorno del punto (x_0, y_0) la función $f(x, y)$ tiene derivadas continuas hasta k-ésimo orden inclusive, la solución $y(x)$ de la ecuación

$$\frac{dy}{dx} = f(x, y) \ ,$$

que satisface la condición inicial $y(x_0) = y_0$ tendrá derivadas continuas hasta $k + 1$-ésimo orden inclusive en cierto entorno del punto (x_0, y_0).

Capítulo 7

Ecuaciones diferenciales de orden mayor que uno.

versión final 1.2b-011015[1]

7.1 Teorema de existencia y unicidad para la ecuación diferencial de n-ésimo orden.

Las ecuaciones diferenciales de n-ésimo orden tienen la forma

$$y^{(n)} = f(x, y, y', \dots, y^{(n-1)}) \ , \tag{7.1}$$

o bien, si no están resueltas con respecto a la derivada de orden mayor:

$$F(x, y, y', \dots, y^{(n)}) = 0 \ .$$

El teorema de existencia y unicidad para ecuación de n-ésimo orden se puede obtener fácilmente, llevándola a un sistema de ecuaciones.

Teorema 7.1 Si en un entorno de las condiciones iniciales $(x_0, y_0, y_0', \dots, y_0^{(n-1)})$ la función f es continua en todos sus argumentos y satisface la condición de Lipschitz respecto a todos los argumentos a partir del segundo, existe una solución única de la ecuación diferencial de n-ésimo orden $y^{(n)} = f(x, y, y', ..., y^{(n-1)})$ que satisface las condiciones

$$y(x_0) = y_0 \ , \quad y'(x_0) = y_0' \ , \quad y''(x_0) = y_0'' \ , \quad \dots \ , \quad y^{(n-1)}(x_0) = y_0^{(n-1)} \ .$$

Se llama solución general de la ecuación diferencial de n-ésimo orden al conjunto de soluciones formado por todas las soluciones particulares, sin excepción. Si el segundo miembro de la ecuación

$$y^{(n)} = f(x, y, y', \dots, y^{(n-1)}) \ , \tag{7.1}$$

satisface, en cierta región de variación de sus argumentos, las condiciones del teorema de existencia y unicidad, entonces la solución general de la ecuación (7.1) depende de n parámetros,

[1]Este capítulo está basado en el segundo capítulo del libro: *Ecuaciones diferenciales y cálculo variacional* de L. Elsgoltz, editorial MIR

en calidad de los cuales se pueden tomar, por ejemplo, las condiciones iniciales de la función buscada y de sus derivadas y_0, y_0', y_0'', ... , $y_0^{(n-1)}$.

En particular, la solución general de la ecuación de segundo grado $y'' = f(x, y, y')$ depende de dos parámetros, por ejemplo, de y_0 y de y_0'. Si fijamos y_0 e y_0' o sea, damos el punto (x_0, y_0), y la dirección de la tangente a la curva integral buscada en dicho punto, entonces, si se cumplen las condiciones del teorema de existencia y unicidad, mediante estas condiciones se determinará una sola curva integral.

Por ejemplo, la ecuación del movimiento rectilíneo de un punto material de masa m bajo la acción de la fuerza $f(t, x, \dot{x})$:

$$m\ddot{x} = f(t, x, \dot{x}) \ ,$$

la posición inicial del punto $x(t_0) = x_0$ y la velocidad inicial $\dot{x}(t_0) = \dot{x_0}$ determinan una solución única, una trayectoria única $x = x(t)$ si, por supuesto, la función f satisface las condiciones del teorema de existencia y unicidad.

7.2 Casos simples de reducción del orden.

En ciertos casos el orden de la ecuación diferencial puede ser reducido, lo que a menudo facilita su integración.

Señalemos algunas clases de ecuaciones que se encuentran con mayor frecuencia y que pueden reducir su orden.

1. La ecuación no contiene la función buscada y sus derivadas hasta el orden $k-1$ inclusive:

$$F(x, y^{(k)}, y^{(k+1)}, \dots , y^{(n)}) = 0 \ . \tag{7.2}$$

En este caso el orden de la ecuación puede ser reducido a $n - k$ mediante el cambio de variables $y^{(k)} = p$.

En efecto, luego del cambio de variables, la ecuación (7.2) toma la forma

$$F(x, p, p', \dots , p^{(n-k)}) = 0 \ . \tag{7.3}$$

De esta ecuación se determina $p = p(x, c_1, c_2, \dots , c_{n-k})$, e y se halla de la ecuación $y^{(k)} = p(x, c_1, c_2, \dots , c_{n-k})$ integrando k veces. En particular, si la ecuación de segundo orden no contiene a y, entonces la sustitución de variables $y' = p$ conduce a una ecuación de primer orden.

Ejemplo Consideremos la siguiente ecuación

$$\frac{d^5y}{dx^5} - \frac{1}{x}\frac{d^4y}{dx^4} = 0 \ .$$

Haciendo $\dfrac{d^4y}{dx^4} = p$ obtenemos $\dfrac{dp}{dx} - \dfrac{1}{x}p = 0$; separando variables e integrando, tendremos: $\ln|p| = \ln|x| + \ln c$, o bien $p = cx$, $\dfrac{d^4y}{dx^4} = cx$, de donde

$$y = c_1x^5 + c_2x^3 + c_3x^2 + c_4x + c_5 \ .$$

Ejemplo Hallar la trayectoria de un cuerpo que cae sin velocidad inicial en la atmósfera, considerando la resistencia del aire proporcional al cuadrado de la velocidad.

La ecuación de movimiento tiene la forma

$$m\frac{d^2s}{dt^2} = mg - k\left(\frac{ds}{dt}\right)^2 \ ,$$

donde s es el espacio recorrido por el cuerpo; m, la masa del mismo; t, el tiempo. Para $t = 0$, se tiene $s = 0$ y $\dfrac{ds}{dt} = 0$.

La ecuación no contiene explícitamente a la función incógnita s; por lo tanto, se puede reducir el orden de la misma considerando $\dfrac{ds}{dt} = v$. Entonces la ecuación de movimiento toma la forma

$$m\frac{dv}{dt} = mg - kv^2 \ .$$

Separando variables e integrando, se obtiene

$$\frac{m\,dv}{mg - kv^2} = dt \ ; \quad t = m\int_0^v \frac{dv}{mg - kv^2} = \frac{1}{k\sqrt{g}}\text{Arctanh}\frac{kv}{\sqrt{g}} \ ,$$

de donde $v = \dfrac{\sqrt{g}}{k}\tanh(k\sqrt{g}\,t)$; multiplicando por dt e integrando nuevamente, hallamos la trayectoria del movimiento:

$$s = \frac{1}{k^2}\ln\cosh(k\sqrt{g}\,t) \ .$$

2. La ecuación no contiene a la variable independiente:

$$F(y, y', y'', \ldots, y^{(n)}) = 0 \ .$$

En este caso el orden de la ecuación se puede reducir en una unidad por medio de la sustitución $y' = p$; además, p se considera nueva función desconocida de y, $p = p(y)$ y, por lo tanto, todas las derivadas $\dfrac{d^k}{dx^k}$ deben expresarse por medio de las derivadas de la nueva función desconocida $p(y)$ respecto a y:

$$\begin{aligned}
&\frac{dy}{dx} = p \ , \\
&\frac{d^2y}{dx^2} = \frac{dp}{dx} = \frac{dp}{dy}\frac{dy}{dx} = \frac{dp}{dy}\,p \ , \\
&\frac{d^3y}{dx^3} = \frac{d}{dx}\left(\frac{dp}{dy}\,p\right) = \frac{d}{dy}\left(\frac{dp}{dy}\,p\right)\frac{dy}{dx} = \frac{d^2p}{dy^2}\,p^2 + \left(\frac{dp}{dy}\right)^2 p \ .
\end{aligned}$$

y análogamente para las derivadas de orden superior. Además, es evidente que la derivada $\dfrac{d^k y}{dx^k}$ se expresa mediante las derivadas de p respecto a y de orden no superior a $k-1$, lo cual precisamente conduce a la disminución del orden en una unidad.

En particular, si la ecuación de segundo orden no contiene a la variable independiente, entonces la sustitución de variables señalada conduce a una ecuación de primer orden.

Ejemplo Integrar la ecuación del péndulo matemático $\ddot{x} + a^2 \operatorname{sen} x = 0$ con condiciones iniciales $x(0) = x_0$, $\dot{x}(0) = \dot{x}_0$.

Reducimos el orden, haciendo

$$\dot{x} = v \ , \quad \ddot{x} = v\frac{dv}{dx} \ , \quad v\,dv = -a^2 \operatorname{sen} x\,dx \ ,$$

$$\frac{v^2}{2} = a^2\,(\cos x - \cos x_0) \ , \quad v = \pm a\sqrt{2(\cos x - \cos x_0)} \ ,$$

$$\frac{dx}{dt} = \pm a\sqrt{2(\cos x - \cos x_0)} \ , \quad t = \pm\frac{1}{a\sqrt{2}} \int_{x_0}^{x} \frac{dx'}{\sqrt{\cos x' - \cos x_0}} \ .$$

La integral del segundo miembro no se resuelve en funciones elementales, pero se reduce fácilmente a funciones elípticas.

3. El primer miembro de la ecuación

$$F(x, y, y', \ldots, y^{(n)}) = 0 \ . \tag{7.4}$$

es la derivada de cierta expresión diferencial $\Phi(x, y, y', \ldots, y^{(n-1)})$ de orden $n-1$.

En este caso se halla fácilmente la llamada *primera integral*, o sea, una ecuación diferencial de orden $n-1$, que contiene una constante arbitraria, y que es equivalente a la ecuación dada de n-ésimo orden, con lo cual reducimos el orden de la ecuación en una unidad. Efectivamente, la ecuación (7.4) puede escribirse en la forma

$$\frac{d}{dx}\Phi(x, y, y', \ldots, y^{(n-1)}) = 0 \tag{7.5}$$

Si $y(x)$ es solución de la ecuación (7.5), entonces la derivada de $\Phi(x, y, y', \ldots, y^{(n-1)})$ es idénticamente nula. Por lo tanto, la función $\Phi(x, y, y', \ldots, y^{(n-1)})$ es igual a una constante, con lo que se obtiene la primera integral

$$\Phi(x, y, y', \ldots, y^{(n-1)}) = c$$

Ejemplo Consideremos la siguiente ecuación

$$yy'' + (y')^2 = 0 \ .$$

Esta ecuación se puede escribir en la forma $d(yy') = 0$, de donde $yy' = c_1$, o bien $y\,dy = c_1\,dx$. Por lo tanto, la integral general será $y^2 = c_1x + c_2$.

A veces el primer miembro de la ecuación $F(x, y, y', \dots, y^{(n)}) = 0$ se convierte en derivada de la diferencial $\Phi(x, y, y', \dots, y^{(n-1)})$ de orden $n-1$ sólo después de multiplicarlo por un factor, $\mu(x, y, y', \dots, y^{(n-1)})$.

Ejemplo Consideremos la siguiente ecuación:

$$yy'' - (y')^2 = 0\ . \tag{7.6}$$

Multiplicando por el factor $\mu = 1/y^2$, se obtiene $[yy'' - (y')^2]/y^2 = 0$, o bien $\dfrac{d}{dx}\left(\dfrac{y'}{y}\right) = 0$, de donde $\dfrac{y'}{y} = c_1$, ó $\dfrac{d}{dx}\ln|\,y\,| = c_1$. Por lo tanto, $\ln|\,y\,| = c_1x + \ln c_2$, $c_2 > 0$, de donde $y = c_2e^{c_1x}$, $c_2 \neq 0$.

Observación: Al multiplicar por el factor $\mu(x, y, y', \dots, y^{(n-1)})$ se pueden introducir soluciones superfluas, que reducen dicho factor a cero. Si μ, es discontinuo, pueden también perderse soluciones. En el ejemplo anterior, al multiplicar por $\mu = 1/y^2$ se perdió la solución $y = 0$; sin embargo, puede incluirse en la solución obtenida, si se considera que c_2 puede tomar el valor 0.

4. La ecuación $F(x, y, y', \dots, y^{(n)}) = 0$ es homogénea con respecto a los argumentos $y, y', \dots, y^{(n)}$.

   El orden de la ecuación homogénea respecto a $y, y', \dots, y^{(n)}$

$$F(x, y, y', \dots, y^{(n)}) = 0 \tag{7.7}$$

   es decir, de la ecuación para la cual se cumple la identidad

$$F(x, ky, ky', \dots, ky^{(n)}) = k^p\,F(x, y, y', \dots, y^{(n)})\ ,$$

   puede ser reducido en una unidad por medio de la sustitución $y = e^{\int z\,dx}$, donde z es una nueva función desconocida. En efecto, derivando, se obtiene

$$\begin{aligned}
y' &= e^{\int z\,dx}\,z\ ,\\
y'' &= e^{\int z\,dx}\,(z^2 + z')\ ,\\
y''' &= e^{\int z\,dx}\,(z^3 + 3zz' + z'')\ ,\\
&\vdots\\
y^{(k)} &= e^{\int z\,dx}\,\Phi(z, z', z'', \dots, z^{(k-1)})\ ,
\end{aligned}$$

   se puede comprobar la veracidad de esta igualdad mediante el método de inducción completa.

   Sustituyendo en (7.7) y observando que en base a la homogeneidad el factor $e^{p\int z\,dx}$ se puede sacar de la función F, reordenamos en función de las derivadas de z obteniendo

$$e^{p\int z\,dx}\,f(x, z, z', \dots, z^{(n-1)}) = 0\ ,$$

o bien, dividiendo entre $e^{p\int z\,dx}$, tendremos para la nueva función f,

$$f(x, z, z', \dots, z^{(n-1)}) = 0 \ .$$

Ejemplo Consideremos la ecuación:

$$yy'' - (y')^2 = 6xy^2 \ .$$

Haciendo $y = e^{\int z\,dx}$, obtenemos $z' = 6x$, $z = 3x^2 + c_1$, $y = e^{\int (3x^2+c_1)\,dx}$, o bien $y = c_2 e^{(x^3+c_1 x)}$.

En las aplicaciones se encuentran con particular frecuencia ecuaciones diferenciales de segundo orden que pueden reducir su orden.

1.

$$F(x, y'') = 0 \ . \tag{7.8}$$

En esta ecuación se puede disminuir el orden mediante la sustitución $y' = p$, y reducirla a la ecuación $F\left(x, \frac{dp}{dx}\right) = 0$.

La ecuación (7.8) se puede resolver con respecto al segundo argumento, $y'' = f(x)$, e integrar dos veces, o introducir un parámetro y sustituir la ecuación (7.8) por su representación paramétrica

$$\frac{d^2y}{dx^2} = \varphi(t) \ , \quad x = \psi(t) \ ,$$

de donde

$$dy' = y''\,dx = \varphi(t)\psi'(t)\,dt \ , \quad y' = \int \varphi(t)\psi'(t)\,dt + c_1 \ ,$$

$$dy = y'\,dx \ , \quad y = \int \left[\int \varphi(t)\psi'(t)\,dt \ + c_1\right] \psi'(t')\,dt' + c_2 \ .$$

2.

$$F(y', y'') = 0 \ , \tag{7.9}$$

la ecuación (7.9) en forma paramétrica:

$$y'_x = \varphi(t) \ , \quad y''_{xx} = \psi(t) \ ,$$

de donde

$$dx = \frac{dy'}{y''} = \frac{\varphi'(t)\,dt}{\psi(t)} \ , \quad x = \int \frac{\varphi'(t)\,dt}{\psi(t)} + c_1 \ ,$$

luego de lo cual y se determina por cuadratura:

$$dy = y'\,dx = \varphi(t)\frac{\varphi'(t)}{\psi(t)}\,dt \ , \quad y = \int \frac{\varphi(t)\varphi'(t)}{\psi(t)}\,dt + c_2 \ .$$

3.

$$F(y, y'') = 0 \ . \tag{7.10}$$

Se puede reducir el orden haciendo

$$\frac{dy}{dx} = p \quad \frac{d^2y}{dx^2} = \frac{dp}{dy}\frac{dy}{dx} = p\frac{dp}{dy} \ .$$

Si la ecuación (7.10) es posible resolver fácilmente con respecto al segundo argumento, $y'' = f(y)$, entonces, multiplicando esta ecuación por la igualdad $2y'\,dx = 2\,dy$, obtenemos $d(y')^2 = 2f(y)\,dy$, de donde

$$\frac{dy}{dx} = \pm\sqrt{2\int f(y)\,dy + c_1} \ , \qquad \pm\frac{dy}{\sqrt{2\int f(y)\,dy + c_1}} = dx \ ,$$

$$x + c_2 = \pm\int \frac{dy}{\sqrt{2\int f(y)\,dy + c_1}}$$

La ecuación (7.10) se puede sustituir por su representación paramétrica $y = \varphi(t)$, $y'' = \psi(t)$; entonces de $dy' = y''dx$ y de $dy = y'dx$ se obtiene $y'dy' = y''dy$, o bien

$$\frac{1}{2}d(y')^2 = \psi(t)\varphi'(t)\,dt \ ,$$

$$(y')^2 = 2\int \psi(t)\varphi'(t)\,dt + c_1 \ ,$$

$$y' = \pm\sqrt{2\int \psi(t)\varphi'(t)\,dt + c_1} \ ,$$

luego de lo cual, de $dy = y'dx$ se halla dx, y después x:

$$dx = \frac{dy}{y'} = \pm\frac{\varphi'(t)\,dt}{\sqrt{2\int \psi(t)\varphi'(t)\,dt + c_1}} \ ,$$

$$x = \pm\int \frac{\varphi'(t)\,dt}{\sqrt{2\int \psi(t)\varphi'(t)\,dt + c_1}} + c_2 \ .$$

Las ecuaciones anteriores e $y = \varphi(t)$ determinan en forma paramétrica a la familia de curvas integrales.

Ejemplo Consideremos la siguiente ecuación

$$y'' = 2y^3 \ , \quad y(0) = 1 \ , \quad y'(0) = 1 \ .$$

Multiplicando ambos miembros de esta ecuación por $2y'dx$, se obtiene $d(y')^2 = 4y^3\,dy$, de donde $(y')^2 = y^4 + c_1$. Teniendo en cuenta las condiciones iniciales, se halla que $c_1 = 0$ e $y' = y^2$. Por lo tanto, $\frac{dy}{y^2} = dx$, $-\frac{1}{y} = x + c_2$, $c_2 = -1$, $y = \frac{1}{1-x}$.

7.3 Ecuaciones diferenciales lineales de n-ésimo orden.

Se llama *ecuación diferencial lineal de n-ésimo orden* una ecuación lineal con respecto a la función desconocida y a sus derivadas, y que, por lo tanto, tiene la forma

$$a_0(x)y^{(n)} + a_1(x)y^{(n-1)} + \ldots + a_{n-1}(x)y' + a_n(x)y = \varphi(x) \ . \tag{7.11}$$

Si el segundo miembro $\varphi(x) = 0$, entonces la ecuación se llama lineal homogénea, puesto que es homogénea con respecto a la función desconocida y y a sus derivadas.

Si el coeficiente $a_0(x)$ es diferente de cero en todos los puntos de cierto intervalo $a \leq x \leq b$, entonces, dividiendo entre $a_0(x)$, reducimos la ecuación lineal homogénea (si x varía en dicho intervalo) a la forma:

$$y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_{n-1}(x)y' + p_n(x)y = 0 \ , \tag{7.12}$$

o bien

$$y^{(n)} = -\sum_{i=1}^{n} p_i(x)y^{(n-i)} \ . \tag{7.13}$$

Si los coeficientes $p(x)$ son contínuos en el intervalo $a \leq x \leq b$, entonces en un entorno de cualesquiera condiciones iniciales

$$y(x_0) = y_0 \ , \quad y'(x_0) = y_0' \ , \quad y''(x_0) = y_0'' \ , \quad \ldots \quad y^{(n-1)}(x_0) = y_0^{(n-1)} \ ,$$

donde x_0 es cualquier punto del intervalo $a \leq x \leq b$, se satisfacen las condiciones del teorema de existencia y unicidad.

Obsérvese que la linealidad y la homogeneidad de la ecuación se conservan en cualquier transformación de la variable independiente $x = \varphi(t)$, donde $\varphi(t)$ es una función arbitraria derivable n veces, cuya derivada $\varphi(t) \neq 0$ en el segmento de variación de t considerado.

En efecto,

$$\begin{aligned} \frac{dy}{dx} &= \frac{dy}{dt}\frac{1}{\varphi'(t)} \ , \\ \frac{d^2y}{dx^2} &= \frac{d^2y}{dt^2}\frac{1}{[\prime(t)]^2} - \frac{dy}{dt}\frac{\varphi''}{[\varphi'(t)]^3} \ , \\ &\vdots \end{aligned}$$

La derivada de cualquier orden $\dfrac{d^ky}{dx^k}$ es función lineal homogénea de $\dfrac{dy}{dt}$, $\dfrac{d^2y}{dt^2}$, $\ldots$, $\dfrac{d^ky}{dt^k}$ y, por lo tanto, al sustituir en la ecuación (7.12) su linealidad y su homogeneidad se conservan.

La linealidad y la homogeneidad se conservan también al efectuarse una transformación lineal homogénea de la función desconocida: $y(x) = \alpha(x)z(x)$. En efecto, por la fórmula de derivada de un producto,

$$y^{(k)} = \alpha(x)z^{(k)} + k\alpha'(x)z^{(k-1)} + \frac{k(k-1)}{2!}\alpha''(x)z^{(k-2)} + \ldots + \alpha^{(k)}(x)z \ ,$$

es decir, la derivada $y^{(k)}$ es función lineal homogénea de z, z', z'', ... , $z^{(k)}$. En consecuencia, el primer miembro de la ecuación lineal homogénea

$$a_0(x)y^{(n)} + a_1(x)y^{(n-1)} + \ldots + a_{n-1}(x)y' + a_n(x)y = 0 \ .$$

luego de sustituir las variables, será función lineal homogénea de z, z', z'', ... , $z^{(n)}$.

Escribamos la ecuación lineal homogénea

$$y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_{n-1}(x)y' + p_n(x)y = 0 \ ,$$

en forma compacta:

$$L[y] = 0 \ ,$$

donde

$$L[y] = y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_{n-1}(x)y' + p_n(x)y \ .$$

Llamaremos a $L[y]$ *operador diferencial lineal.*

El operador diferencial lineal posee las dos propiedades fundamentales siguientes:

1. Un factor constante puede sacarse del símbolo del operador:

$$L[cy] \equiv cL[y] \ .$$

En efecto,

$$\begin{aligned}(cy)^{(n)} + p_1(x)(cy)^{(n-1)} + \ldots + p_{n-1}(x)(cy)' + p_n(x)(cy) \\ = c[y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_{n-1}(x)y' + p_n(x)y] \ .\end{aligned}$$

2. El operador diferencial lineal, aplicado a la suma de dos funciones es igual a la suma de los resultados de la aplicación del mismo a cada función por separado:

$$L[y_1 + y_2] \equiv L[y_1] + L[y_2] \ .$$

en efecto,

$$\begin{aligned}(y_1 + y_2)^{(n)} + p_1(x)(y_1 + y_2)^{(n-1)} + \ldots + p_{n-1}(x)(y_1 + y_2)' + p_n(x)(y_1 + y_2) \equiv \\ [y_1^{(n)} + p_1(x)y_1^{(n-1)} + \ldots + p_{n-1}(x)y_1' + p_n(x)y_1] + \\ [y_2^{(n)} + p_1(x)y_2^{(n-1)} + \ldots + p_{n-1}(x)y_2' + p_n(x)y_2].\end{aligned}$$

Como consecuencia de las propiedades anteriores, resulta

$$L\left[\sum_{i=1}^{m} c_i y_i\right] \equiv \sum_{i=1}^{m} c_i L[y_i] \ ,$$

donde los c_i son constantes.

Basándonos en las propiedades del operador lineal L, se puede demostrar una serie de teoremas sobre las soluciones de la ecuación lineal homogénea

Teorema 7.2 Si y_1 es solución de la ecuación de lineal homogénea $L[y] = 0$, entonces cy_1, donde c es una constante arbitraria, también es solución de ésta.

Teorema 7.3 La suma $y_1 + y_2$ de dos soluciones y_1 e y_2 de la ecuación lineal homogénea $L[y] = 0$ es solución de dicha ecuación.

Corolario de los dos teoremas anteriores. La combinación lineal con coeficientes arbitrarios constantes $\sum_{i=1}^{m} c_i y_i$ de las soluciones y_1, y_2, ... , y_m de la ecuación lineal homogénea $L[y] = 0$ es solución de dicha ecuación.

Teorema 7.4 Si la ecuación lineal homogénea $L[y] = 0$ con coeficientes reales $p_i(x)$ tiene solución compleja $y(x) = u(x) + iv(x)$, entonces la parte real $u(x)$ de esta solución y su parte imaginaria $v(x)$ son, por separado, soluciones de dicha ecuación homogénea.

Las funciones y_1, y_2, ... , y_n se llaman *linealmente dependientes* en cierto intervalo de x, $a \leq x \leq b$, si existen constantes α_1, α_2, ... , α_n, que en dicho intervalo

$$\alpha_1 y_1 + \alpha_2 y_2 + \ldots + \alpha_n y_n \equiv 0 \ , \tag{7.14}$$

y además por lo menos un $\alpha_i \neq 0$. Si la identidad (7.14) se verifica sólo para el caso en que $\alpha_1 = \alpha_2 =, \ldots , = \alpha_n = 0$, las funciones y_1, y_2, ... , y_n se llaman *linealmente independientes* en el intervalo $a \leq x \leq b$.

Teorema 7.5 Si las funciones y_1, y_2, ... , y_n son linealmente dependientes en el intervalo $a \leq x \leq b$, entonces en dicho intervalo el determinante

$$W(x) = W[y_1, y_2, \ldots , y_n] = \begin{vmatrix} y_1 & y_2 & \cdots & y_n \\ y_1' & y_2' & \cdots & y_n' \\ y_1'' & y_2'' & \cdots & y_n'' \\ \vdots & \vdots & & \vdots \\ y_1^{(n-1)} & y_2^{(n-1)} & \cdots & y_n^{(n-1)} \end{vmatrix}$$

llamado *wroskiano*, es idénticamente nulo.

Teorema 7.6 Si las funciones linealmente independientes y_1, y_2, ... , y_n son soluciones de la ecuación lineal homogénea.

$$y^{(n)} + p_1(x) y^{(n-1)} + \ldots + p_{n-1}(x) y' + p_n(x) y = 0 \ , \tag{7.15}$$

con coeficientes contínuos $p_i(x)$ en el intervalo $a \leq x \leq b$, entonces el wroskiano

$$W(x) = \begin{vmatrix} y_1 & y_2 & \cdots & y_n \\ y_1' & y_2' & \cdots & y_n' \\ y_1'' & y_2'' & \cdots & y_n'' \\ \vdots & \vdots & & \vdots \\ y_1^{(n-1)} & y_2^{(n-1)} & \cdots & y_n^{(n-1)} \end{vmatrix}$$

es diferente de cero en todos los puntos del intervalo $a \leq x \leq b$.

Teorema 7.7 La combinación lineal $\sum_{i=1}^{n} c_i y_i$ con coeficientes constantes arbitrarios de n soluciones particulares linealmente independientes y_i $(i = 1, 2, \ldots, n)$ en el intervalo $a \leq x \leq b$ es solución general, para $a \leq x \leq b$, de la ecuación lineal homogénea

$$y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_{n-1}(x)y' + p_n(x)y = 0 \ , \tag{7.15}$$

con coeficientes $p_i(x)$ contínuos en dicho intervalo $(i = 1, 2, \ldots, n)$.

Corolario del teorema anterior. El número máximo de soluciones linealmente independientes de una ecuación lineal homogénea es igual a su orden.

Observación: Se llama *sistema fundamental de soluciones* de una ecuación lineal homogénea de n-ésimo orden al conjunto de cualquiera n soluciones particulares linealmente independiente. Para cada ecuación lineal homogénea (7.15) existe un sistema fundamental de soluciones. Para la construcción de un sistema fundamental de soluciones, se dan arbitrariamente n^2 cifras

$$y_i^{(k)}(x_0) \quad \{i = 1, 2, \ldots, n;\ k = 0, 1, \ldots, n-1\} \ ,$$

sometiendo su elección exclusivamente a la condición

$$\begin{vmatrix} y_1(x_0) & y_2(x_0) & \ldots & y_n(x_0) \\ y_1'(x_0) & y_2'(x_0) & \ldots & y_n'(x_0) \\ y_1''(x_0) & y_2''(x_0) & \ldots & y_n''(x_0) \\ \vdots & \vdots & & \vdots \\ y_1^{(n-1)}(x_0) & y_2^{(n-1)}(x_0) & \ldots & y_n^{(n-1)}(x_0) \end{vmatrix} \neq 0 \ ,$$

donde x_0 es un punto cualquiera del intervalo $a \leq x \leq b$. Entonces las soluciones $y_i(x)$, determinadas por los valores iniciales $y_i^{(k)}(x_0)$ con $\{i = 1, 2, \ldots, n;\ k = 0, 1, \ldots, n-1\}$, forman un sistema fundamental, puesto que su wroskiano $W(x)$ en el punto $x = x_0$ es diferente de cero y, por lo tanto, en virtud del teorema 7.5 y del teorema 7.6, las soluciones y_1, y_2, $\ldots$, y_n son linealmente independientes.

Conociendo una solución particular no trivial y_1 de la ecuación lineal homogénea

$$y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_{n-1}(x)y' + p_n(x)y = 0 \ , \tag{7.15}$$

se puede, por medio de la sustitución $y = y_1 \int u\, dx$, reducir su orden manteniendo la linealidad y la homogeneidad.

En efecto, la sustitución $y = y_1 \int u\, dx$ se puede reemplazar por dos sustituciones $y = y_1 z$ y $z' = u$. La trasformación lineal homogénea

$$y = y_1 z \tag{7.16}$$

conserva la linealidad y la homogeneidad de la ecuación; por lo tanto, la ecuación (7.15) se reduce en este caso a la forma

$$a_0(x)\, z^{(n)} + a_1(x)\, z^{(n-1)} + \cdots + a_n(x)\, z = 0 \ . \tag{7.17}$$

Además, a la solución $y = y_1$ de la ecuación (7.15) le corresponde, en virtud de (7.16), la solución $z \equiv 1$ de la ecuación (7.17). Sustituyendo $z \equiv 1$ en la ecuación (7.17), se obtiene $a_n = 0$. Por consiguiente, la ecuación (7.17) tiene la forma

$$a_0(x)\, z^{(n)} + a_1(x)\, z^{(n-1)} + \cdots + a_{n-1}(x)\, z' = 0 \ ,$$

y la sustitución $z' = u$ reduce su orden en una unidad:

$$a_0(x)\, u^{(n-1)} + a_1(x)\, u^{(n-2)} + \cdots + a_{n-1}(x)\, u = 0 \ .$$

Obsérvese que la misma sustitución $y = y_1 \int u\, dx$, donde y_1 es solución de la ecuación $L[y] = 0$, reduce en una unidad el orden de la ecuación lineal no homogénea $L[y] = f(x)$, puesto que dicha sustitución no altera el segundo miembro de la ecuación.

Conociendo k soluciones linealmente independientes $y_1, \ldots, y_k$ en el intervalo $a \le x \le b$, de la ecuación lineal homogénea, se puede reducir su orden hasta $n-k$, en el mismo intervalo $a \le x \le b$.

Lema. Dos ecuaciones de la forma

$$y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_{n-1}(x)y' + p_n(x)y = 0 \ , \tag{7.18}$$

$$y^{(n)} + q_1(x)y^{(n-1)} + \ldots + q_{n-1}(x)y' + q_n(x)y = 0 \ , \tag{7.19}$$

donde las funciones $p_i(x)$ y $q_i(x)$ $(i = 1, 2, \cdots, n)$ son continuas en el intervalo $a \le x \le b$, las cuales tienen un sistema fundamental común de soluciones $y_1, y_2, \ldots, y_n$, coinciden, es decir, que $p_i(x) \equiv q_i(x)$ $(i = 1, 2, \cdots, n)$ en el intervalo $a \le x \le b$.

De este modo, el sistema fundamental de soluciones $y_1, y_2, \ldots, y_n$ determina por completo la ecuación lineal homogénea

$$y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_{n-1}(x)y' + p_n(x)y = 0 \tag{7.18}$$

y, por consiguiente, se puede plantear el problema de hallar la ecuación (7.18) que posea el sistema fundamental de soluciones

$$y_1, \ y_2, \ \ldots, \ y_n \ .$$

Como cualquier solución y de la ecuación buscada (7.18) debe ser linealmente dependiente de las soluciones $y_1, y_2, \ldots, y_n$, entonces el Wronskiano $W[y_1, y_2, \ldots, y_n] = 0$. Escribamos esta ecuación en forma desarrollada

$$\begin{vmatrix} y_1 & y_2 & \cdots & y_n & y \\ y_1' & y_2' & \cdots & y_n' & y' \\ y_1'' & y_2'' & \cdots & y_n'' & y'' \\ \vdots & \vdots & & \vdots & \vdots \\ y_1^{(n)} & y_2^{(n)} & \cdots & y_n^{(n)} & y^{(n)} \end{vmatrix} = 0 \ ,$$

o bien, descomponiéndola por los elementos de la última columna,

$$W[y_1, y_2, \ldots, y_n]\, y^{(n)} - \begin{vmatrix} y_1 & y_2 & \cdots & y_n \\ y_1' & y_2' & \cdots & y_n' \\ \vdots & \vdots & & \vdots \\ y_1^{(n-2)} & y_2^{(n-2)} & \cdots & y_n^{(n-2)} \\ y_1^{(n)} & y_2^{(n)} & \cdots & y_n^{(n)} \end{vmatrix} y^{(n-1)} + \ldots = 0 \ . \tag{7.20}$$

La ecuación obtenida (7.20) es la ecuación lineal homogénea buscada, que posee el sistema dado de soluciones $y_1,\ y_2,\ \ldots\ ,y_n$. Dividiendo ambos miembros de la ecuación entre el Wronskiano de la derivada de mayor grado, diferente de cero, la reducimos a la forma (7.18).

De aquí se deduce que, en particular,

$$p_1(x) = -\frac{\begin{vmatrix} y_1 & y_2 & \ldots & y_n \\ y_1' & y_2' & \ldots & y_n' \\ \vdots & \vdots & & \vdots \\ y_1^{(n-2)} & y_2^{(n-2)} & \ldots & y_n^{(n-2)} \\ y_1^{(n)} & y_2^{(n)} & \ldots & y_n^{(n)} \end{vmatrix}}{W[y_1, y_2, \ldots, y_n]} \ .$$

Obsérvese que el determinante

$$\begin{vmatrix} y_1 & y_2 & \ldots & y_n \\ y_1' & y_2' & \ldots & y_n' \\ \vdots & \vdots & & \vdots \\ y_1^{(n-2)} & y_2^{(n-2)} & \ldots & y_n^{(n-2)} \\ y_1^{(n)} & y_2^{(n)} & \ldots & y_n^{(n)} \end{vmatrix} , \tag{7.21}$$

es igual a la derivada del wroskiano $W[y_1, y_2, \ldots, y_n]$. En efecto, según la regla de derivación de un determinante, la derivada

$$\frac{d}{dx}\begin{vmatrix} y_1 & y_2 & \ldots & y_n \\ y_1' & y_2' & \ldots & y_n' \\ \vdots & \vdots & & \vdots \\ y_1^{(n-2)} & y_2^{(n-2)} & \ldots & y_n^{(n-2)} \\ y_1^{(n-1)} & y_2^{(n-1)} & \ldots & y_n^{(n-1)} \end{vmatrix} ,$$

es igual a la suma sobre i desde 1 hasta n de determinantes que se diferencian del wronskiano en que en ello se han derivado los elementos de la i-ésima fila, y las filas restantes se dejan sin variación. En esta suma solamente el último determinante, para $i = n$, que coincide con el determinante (7.21) puede ser diferente de cero. Los restantes son iguales a cero, ya que sus filas i e $i + 1$ coinciden.

Por lo tanto, $p_1(x) = \dfrac{W'}{W}$, de donde, multiplicando por dx e integrando, se obtiene

$$\log |\, W \,| = -\int p_1(x)\, dx + \log c \ , \quad W = c\, e^{-\int p_1(x)\, dx} \ ,$$

o bien

$$W = c\, e^{-\int_{x_o}^{x} p_1(x')\, dx'} \ . \tag{7.22}$$

Para $x = x_o$ se obtiene $c = W(x_o)$, de donde

$$W = W(x_o)\, e^{-\int_{x_o}^{x} p_1(x')\, dx'} \ . \tag{7.23}$$

La fórmulas anteriores son conocidas como *fórmulas de Ostrogradski-Liouville*. Estas fórmulas pueden aplicarse a la integración de la ecuación lineal homogénea de segundo orden

$$y'' + p_1(x)\, y' + p_2(x)\, y = 0 \ , \tag{7.24}$$

si es conocida una solución no trivial y_1 de la misma. Según la fórmula (7.23), cualquier solución de la ecuación (7.24) debe ser también solución de la ecuación

$$\begin{vmatrix} y_1 & y \\ y_1' & y' \end{vmatrix} = c_1\, e^{-\int p_1(x)\, dx} \ ,$$

o bien

$$y_1 y' - y y_1' = c_1\, e^{-\int p_1(x)\, dx} \ .$$

Para integrar esta ecuación lineal de primer orden lo más fácil es aplicar el método del factor integrante.

Multiplicando por $\mu = \dfrac{1}{y_1^2}$, se obtiene

$$\frac{d}{dx}\left(\frac{y}{y_1}\right) = \frac{c_1}{y_1^2}\, e^{-\int p_1(x)\, dx} \ ,$$

de donde

$$\frac{y}{y_1} = \int \frac{c_1 e^{-\int p_1(x')\, dx'}}{y_1^2}\, dx + c_2 \ ,$$

o bien

$$y = c_2 y_1 + c_1 y_1 \int \frac{e^{-\int p_1(x')\, dx'}}{y_1^2}\, dx \ .$$

7.4 Ecuaciones lineales homogéneas con coeficientes constantes y ecuación de Euler.

7.4.1 Ecuaciones lineales homogeneas con coeficientes constantes.

Si en la ecuación lineal homogénea

$$a_0\, y^{(n)} + a_1\, y^{(n-1)} + \ldots + a_n y = 0 \ , \tag{7.25}$$

todos los coeficientes a_i son constantes, entonces sus soluciones particulares pueden ser halladas en la forma $y = e^{kx}$, donde k es una constante. En efecto sustituyendo en (7.25) $y = e^{kx}$ e $y^{(p)} = k^p e^{kx}$ con $p = 1, 2, \cdots, n$, tendremos:

$$a_0 k^n e^{kx} + a_1 k^{n-1} e^{kx} + \ldots + a_n e^{kx} = 0 \ .$$

Dividiendo entre el factor e^{kx}, diferente de cero, se obtiene la llamada ecuación característica

$$a_0k^n + a_1k^{n-1} + \ldots + a_n = 0 \ . \tag{7.26}$$

Esta ecuación de n-ésimo grado determina los valores de k para los cuales $y = e^{kx}$ es solución de la ecuación lineal homogénea inicial con coeficientes constantes (7.25). Si todas las raíces k_1, k_2, ... , k_n de la ecuación característica son diferentes, entonces de esta forma se hallan n soluciones linealmente independientes e^{k_1x}, e^{k_2x}, ... , e^{k_nx} de la ecuación (7.25). Por consiguiente,

$$y = c_1e^{k_1x} + c_2e^{k_2x} + \ldots + c_ne^{k_nx} \ ,$$

donde c_i son constantes arbitrarias, es solución general de la ecuación inicial (7.25). Este método de integración de las ecuaciones lineales con coeficientes constantes fue aplicado por primera vez por Euler.

Ejemplo Consideremos la ecuación

$$y'' - 3y' + 2y = 0 \ .$$

la ecuación característica tiene la forma $k^2 - 3k + 2 = 0$; sus raíces son $k_1 = 1$, $k_2 = 2$. Por lo tanto, la solución general de la ecuación inicial tiene la forma $y = c_1e^x + c_2e^{2x}$.

Puesto que los coeficientes de la ecuación (7.25) se presuponen reales, las raíces complejas de las ecuación característica pueden aparecer sólo en pares conjugados. Las soluciones complejas $e^{(\alpha+i\beta)x}$ y $e^{(\alpha-i\beta)x}$, correspondientes al par de raíces complejas conjugadas

$$k_1 = \alpha + i\beta \quad \text{y} \quad k_2 = \alpha - i\beta \ ,$$

pueden ser sustituídas por dos soluciones reales: por las partes reales e imaginarias de una de las soluciones.

$$e^{(\alpha\pm i\beta)x} = e^{\alpha x}\left(\cos\beta x \pm i \operatorname{sen}\beta x\right) \ ,$$

De esta manera, al par de raíces complejas conjugadas $k_{1,2} = \alpha \pm i\beta$ le corresponden dos soluciones reales: $e^{\alpha x}\cos\beta x$ y $e^{\alpha x}\operatorname{sen}\beta x$.

Si entre las raíces de la ecuación característica hay raíces múltiples, entonces la cantidad de soluciones distintas del tipo e^{kx} es menor que n y, por lo tanto, las soluciones linealmente independientes que faltan deben ser buscadas en otra forma.

Se puede demostrar que si la ecuación característica tiene una raíz k_i de multiplicidad α_i, entonces no sólo e^{k_jx} será solución de la ecuación inicial, sino también xe^{k_jx}, $x^2e^{k_jx}$, ... , $x^{\alpha_i-1}e^{k_jx}$. Por lo tanto, la solución general de la ecuación (7.25) tiene la forma

$$y = \sum_{i=1}^{m}\left(c_{0i} + c_{1i}x + c_{2i}x^2 + \ldots + c_{\alpha_i-1,i}x^{\alpha_i-1}\right)e^{k_ix} \ ,$$

donde c_{si} son constantes arbitrarias.

Ejemplo Consideremos la ecuación

$$y''' - 3y'' + 3y' - y = 0 \ .$$

La ecuación característica $k^3 - 3k^2 + 3k - 1 = 0$, o bien $(k-1)^3 = 0$, posee la raíz triple $k_{1,2,3} = 1$. Por consiguiente, la solución general tiene la forma

$$y = (c_1 + c_2 x + c_3 x^2) e^x \ .$$

Si la ecuación característica tiene una raíz múltiple compleja $p + iq$ de multiplicidad α, entonces sus soluciones correspondientes

$$e^{(p+iq)x} = e^{px} \left(\cos qx + i \operatorname{sen} qx\right)$$

y, separando las partes real e imaginaria, obtener 2α soluciones reales:

$$\begin{aligned} &e^{px}\cos qx \ , \ xe^{px}\cos qx \ , \ x^2 e^{px}\cos qx \ , \ \ldots \ , \ x^{\alpha-1}e^{px}\cos qx \ , \\ &e^{px}\operatorname{sen} qx \ , \ xe^{px}\operatorname{sen} qx \ , \ x^2 e^{px}\operatorname{sen} qx \ , \ \ldots \ , \ x^{\alpha-1}e^{px}\operatorname{sen} qx \ . \end{aligned} \tag{7.27}$$

Tomando las partes reales e imaginarias de las soluciones correspondientes a la raíz conjugada $p - iq$ de la ecuación característica, no se obtienen nuevas soluciones linealmente independientes. De esta manera, al par de raíces complejas conjugadas $p \pm iq$ de multiplicidad α le corresponde 2α soluciones reales linealmente independientes (7.27).

7.4.2 Ecuaciones de Euler.

Las ecuaciones de la forma

$$a_0 x^n \, y^{(n)} + a_1 x^{n-1} \, y^{(n-1)} + \ldots + a_{n-1} x y' + a_n y = 0 \ , \tag{7.28}$$

donde todas las a_i son constantes, se llaman *ecuaciones de Euler*. La ecuación de Euler se reduce, mediante la sustitución de la variable independiente $x = e^t$, a una ecuación lineal homogénea con coeficientes constantes.

En efecto, como fue señalado anteriormente, la linealidad y la homogeneidad de la ecuación se conservan en la transformación de la variable independiente, y los coeficientes se vuelven constantes, puesto que

$$\begin{aligned} \frac{dy}{dx} &= \frac{dy}{dt} e^{-t} \ , \\ \frac{d^2y}{dx^2} &= e^{-2t}\left(\frac{d^2y}{dt^2} - \frac{dy}{dt}\right) \ , \\ &\vdots \\ \frac{d^k y}{dx^k} &= e^{-kt}\left(\beta_1 \frac{dy}{dt} + \beta_2 \frac{d^2y}{dt^2} + \ldots + \beta_k \frac{d^k y}{dt^k}\right) \ , \end{aligned} \tag{7.29}$$

donde todas las β_i son constantes, y al sustituir en la ecuación (7.28) los factores e^{-kt} se simplifican con los factores $x^k = e^{kt}$.

La validez de la igualdad (7.29) puede ser demostrada por el método de inducción. Por lo tanto, los productos

$$x^k \frac{d^k y}{dx^k} = \beta_1 \frac{dy}{dt} + \beta_2 \frac{d^2 y}{dt^2} + \ldots + \beta_k \frac{d^k y}{dt^k} \ ,$$

que entran en la forma lineal con coeficientes constantes en la ecuación de Euler

$$\sum_{k=0}^{n} a_{n-k} x^k \frac{d^k y}{dx^k} = 0 \ , \tag{7.30}$$

se expresan en forma lineal (y con coeficientes constantes) mediante las derivadas de la función y respecto a la nueva variable t. De aquí se deduce que la ecuación transformada será una ecuación lineal homogénea con coeficientes constantes:

$$b_0 \frac{d^n y}{dt^n} + b_1 \frac{d^{n-1} y}{dt^{n-1}} + \ldots + b_{n-1} \frac{dy}{dt} + b_n y = 0 \ . \tag{7.31}$$

En lugar de transformar la ecuación de Euler en una ecuación lineal con coeficientes constantes, cuyas soluciones particulares tienen la forma $y = e^{kt}$, se puede buscar directamente la solución de la ecuación inicial en la forma $y = x^k$, ya que

$$e^{kt} = x^k \ .$$

La ecuación obtenida después de simplificar entre x^k

$$a_0 k(k-1)\ldots(k-n+1) + a_1 k(k-1)\ldots(k-n+2) + \ldots + a_n = 0 \ , \tag{7.32}$$

para la determinación de k, debe coincidir con la ecuación característica para la ecuación transformada (7.31). En consecuencia, a las raíces k_i de la ecuación (7.32), de multiplicidad α_i, le corresponden las soluciones

$$e^{k_i t} \ , \ t e^{k_i t} \ , \ t^2 e^{k_i t} \ , \ \ldots \ , \ t^{\alpha_i - 1} e^{k_i t} \ .$$

de la ecuación transformada, o bien las

$$x^{k_i} \ , \ x^{k_i} \log(n) \ , \ x^{k_i} \log^2(n) \ , \ \ldots \ , \ x^{k_i} \log^{\alpha_i - 1}(n) \ ,$$

de la ecuación inicial. A las raíces complejas conjugadas $p \pm iq$ de la ecuación (7.32) de multiplicidad α le corresponden las soluciones

$$\begin{aligned} &e^{pt}\cos qt \ , \ t e^{pt}\cos qt \ , \ \ldots \ , \ t^{\alpha-1} e^{pt}\cos qt \ , \\ &e^{pt}\operatorname{sen} qt \ , \ t e^{pt}\operatorname{sen} qt \ , \ \ldots \ , \ t^{\alpha-1} e^{pt}\operatorname{sen} qt \ , \end{aligned}$$

de la ecuación trasformada, o las

$$\begin{aligned} &x^p \cos(q\log x) \ , \ x^p \log x \cos(q\log x) \ , \ \ldots \ , x^p \log^{\alpha_i - 1} x \cos(q\log x) \ , \\ &x^p \operatorname{sen}(q\log x) \ , \ x^p \log x \operatorname{sen}(q\log x) \ , \ \ldots \ , x^p \log^{\alpha_i - 1} x \operatorname{sen}(q\log x) \ , \end{aligned}$$

de la ecuación inicial de Euler.

Ejemplo Consideremos la ecuación

$$x^2y'' - xy' + y = 0 \ .$$

Buscamos la solución en la forma $y = x^k$; $k(k-1) + k + 1 = 0$, o bien $(k-1)^2 = 0$, $k_{1,2} = 1$. Por consiguiente, la solución general para $x > 0$ será

$$y = (c_1 + c_2 \log x)x \ .$$

Las ecuaciones de la forma

$$a_0(ax+b)^n \, y^{(n)} + a_1(ax+b)^{n-1} \, y^{(n-1)} + \ldots + a_{n-1}(ax+b)y' + a_n y = 0 \ , \tag{7.33}$$

se donominan tambien *ecuaciones de Euler*, y se reducen a la ecuación (7.28) por medio de la sustitución de la variable independiente $(ax+b) = x_1$. Por lo tanto, las soluciones particulares de esta ecuación se pueden buscar en la forma $y = (ax+b)^k$, o transformar la ecuación (7.33) a una ecuación lineal homogénea con coeficientes constantes, mediante la sustitución de las variables $ax + b = e^t$.

7.5 Ecuaciones lineales no homogéneas.

La ecuación *lineal no homogénea* tiene la forma

$$a_0(x) \, y^{(n)} + a_1(x) \, y^{(n-1)} + \ldots + a_{n-1}(x)y' + a_n(x)y = \varphi(x) \ ,$$

Si $a_0(x) \neq 0$ en el intervalo considerado de variación de x, entonces dividiendo entre $a_0(x)$ se obtiene

$$y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_{n-1}(x)y' + p_n(x)y = f(x) \ . \tag{7.34}$$

Esta ecuación, conservando las notaciones anteriores, la escribimos en forma compacta:

$$L[y] = f(x) \ .$$

Si para $a \leq x \leq b$, en la ecuación (7.34) todos los coeficientes p_i y el segundo miembro $f(x)$ son continuos, entonces ella posee una solución única que satisface las condiciones

$$y^{(k)}(x_0) = y_0^{(k)} \ , \quad \text{con } k = 0, 1, \ldots, n-1 \ ,$$

donde $y_0^{(k)}$ son números reales cualesquiera, y x_0 un punto arbitrario del intervalo $a \leq x \leq b$.

De las dos propiedades fundamentales del operador lineal

$$L[cy] \equiv cL[y] \ ,$$
$$L[y_1 + y_2] \equiv L[y_1] + L[y_2] \ ,$$

donde c es una constante, se deduce directamente que:

(i) La suma $\tilde{y} + y_1$ de una solución $\tilde{y}$ de la ecuación no homogénea

$$L[y] = f(x) \tag{7.35}$$

y de una solución y_1 de la ecuación homogénea correspondiente $L[y] = 0$, es solución de la ecuación no homogénea (7.35).

(ii) Si y_i es solución de la ecuación $L[y] = f_i(x)$ con $i = 1, 2, \dots, m$ entonces $y = \sum_{i=1}^{m} \alpha_i y_i$ es solución de la ecuación

$$L[y] = \sum_{i=1}^{m} \alpha_i f_i(x) \ ,$$

donde las α_i son constantes.

Esta propiedad, denominada *principio de superposición*, conserva evidentemente su validez también para $m \to \infty$, si la serie $\sum_{i=1}^{\infty} \alpha_i y_i$ converge y puede ser derivada término a término n veces.

(iii) Si la ecuación $L[y] = U(x) + iV(x)$, donde todos los coeficientes $p_i(x)$ y las funciones $U(x)$ y $V(x)$ son reales, tiene la solución $y = u(x) + iv(x)$, entonces la parte real $u(x)$ y la parte imaginaria $v(x)$ son respectivamente soluciones de las ecuaciones

$$L[y] = U(x) \quad \text{y} \quad L[y] = V(x) \ .$$

Teorema 7.8 La solución general en el intervalo $a \leq x \leq b$ de la ecuación $L[y] = f(x)$ con coeficientes p_i y la función del miembro derecho $f(x)$ continuos en dicho intervalo, es igual a la suma de la solución general $\sum_{i=1}^{n} c_i y_i$ de la ecuación homogénea correspondiente y de cualquier solución particular $\tilde{y}$ de la ecuación no homogénea.

Ejemplo Consideremos la ecuación

$$y'' + y = x \ ,$$

Una solución particular de esta ecuación $y = x$ es inmediata; la solución general de la ecuación homogénea correspondiente tiene la forma

$$y = c_1 \cos x + c_2 \operatorname{sen} x \ .$$

Por lo tanto, la solución general de la ecuación no homogénea inicial es

$$y = c_1 \cos x + c_2 \operatorname{sen} x + x \ .$$

Si la elección de una solución particular de la ecuación no homogénea es difícil, pero la solución general de la ecuación homogénea correspondiente $y = \sum_{i=1}^{n} c_i y_i$ ya fue hallada, entonces se puede integrar la ecuación lineal no homogénea por el método de variación de las constantes.

Al aplicar este método, la solución de la ecuación no homogénea se busca en la forma $y = \sum_{i=1}^{n} c_i(x) y_i$, o sea que en esencia, en lugar de la función incógnita y introducimos n funciones desconocidas $c_i(x)$. Puesto que escogiendo las funciones $c_i(x)$ con $i = 1, 2, \ldots, n$ hay que satisfacer solamente una ecuación

$$y^{(n)} + p_1(x) y^{(n-1)} + \ldots + p_{n-1}(x) y' + p_n(x) y = f(x) \ , \tag{7.34}$$

se puede exigir que estas n funciones $c_i(x)$ satisfagan otras $n-1$ ecuaciones, las cuales se escogen de manera que las derivadas de la función $y = \sum_{i=1}^{n} c_i(x) y_i$ tengan en lo posible la misma forma que tiene cuando las c_i son constantes. Escojamos $c_i(x)$ de manera tal que la segunda suma de

$$y' = \sum_{i=1}^{n} c_i(x) y_i'(x) + \sum_{i=1}^{n} c_i'(x) y_i(x) \ ,$$

sea igual a cero,

$$\sum_{i=1}^{n} c_i'(x) y_i(x) = 0$$

y, por lo tanto,

$$y' = \sum_{i=1}^{n} c_i(x) y_i'(x) \ ,$$

es decir, que y' tiene la misma forma que cuando las c_i son constantes. De la misma manera, en la derivada segunda

$$y'' = \sum_{i=1}^{n} c_i(x) y_i''(x) + \sum_{i=1}^{n} c_i'(x) y_i'(x) \ ,$$

exigimos que la segunda suma sea igual a cero, con lo cual se somete $c_i(x)$ a la segunda condición

$$\sum_{i=1}^{n} c_i'(x) y_i'(x) = 0 \ .$$

Continuamos calculando las derivadas de la función $y = \sum_{i=1}^{n} c_i(x) y_i$ hasta el orden $n-1$ inclusive, e igualando cada vez a cero la suma $\sum_{i=1}^{n} c_i'(x) y_i^{(k)}(x)$:

$$\sum_{i=1}^{n} c_i'(x) y_i^{(k)}(x) = 0 \quad \text{para } k = 0, 1, 2, \ldots, n-2 \ , \tag{7.36}$$

obtenemos

$$\begin{aligned} y &= \sum_{i=1}^{n} c_i(x) y_i \ , \\ y' &= \sum_{i=1}^{n} c_i(x) y_i'(x) \ , \\ y'' &= \sum_{i=1}^{n} c_i(x) y_i''(x) \ , \\ &\vdots \\ y^{(n-1)} &= \sum_{i=1}^{n} c_i(x) y_i^{(n-1)}(x) \ , \\ y^{(n)} &= \sum_{i=1}^{n} c_i(x) y_i^{(n)}(x) + \sum_{i=1}^{n} c_i'(x) y_i^{(n-1)}(x) \ . \end{aligned} \tag{7.37}$$

En la última igualdad no podemos exigir que $\sum_{i=1}^{n} c_i'(x) y_i^{(n-1)}(x) = 0$, puesto que las funciones $c_i(x)$ ya están sometidas a las $n-1$ condiciones (7.36), y hay aún que satisfacer la ecuación inicial (7.34). Sustituyendo y, y', $\ldots$,$y^{(n)}$ de (7.37) en la ecuación

$$y^{(n)} + p_1(x) y^{(n-1)} + \ldots + p_{n-1}(x) y' + p_n(x) y = f(x) \ , \tag{7.34}$$

se obtiene la ecuación que falta para la determinación de $c_i(x)$ con $i = 1, 2, \ldots , n$. Es evidente que en el primer miembro de (7.34) queda sólo la suma $\sum_{i=1}^{n} c_i'(x) y_i^{(n-1)}(x)$, ya que todos los términos restantes tienen la misma forma que cuando las c_i son constantes, y cuando éstas son constantes, la función $y = \sum_{i=1}^{n} c_i(x) y_i$ satisface la ecuación homogénea correspondiente.

De esta manera, las funciones $c_i(x)$ con $i = 1, 2, \ldots , n$ se determinan del sistema de n ecuaciones lineales

$$\begin{aligned} \sum_{i=1}^{n} c_i'(x) y_i &= 0 \ , \\ \sum_{i=1}^{n} c_i'(x) y_i' &= 0 \ , \\ \sum_{i=1}^{n} c_i'(x) y_i'' &= 0 \ , \\ &\vdots \\ \sum_{i=1}^{n} c_i'(x) y_i^{(n-2)} &= 0 \ , \\ \sum_{i=1}^{n} c_i'(x) y_i^{(n-1)} &= f(x) \ . \end{aligned} \tag{7.38}$$

cuyo determinante es diferente de cero, debido a que éste

$$\begin{vmatrix} y_1 & y_2 & \dots & y_n \\ y_1' & y_2' & \dots & y_n' \\ \vdots & \vdots & & \vdots \\ y_1^{(n-2)} & y_2^{(n-2)} & \dots & y_n^{(n-2)} \\ y_1^{(n-1)} & y_2^{(n-1)} & \dots & y_n^{(n-1)} \end{vmatrix} ,$$

es el wronskiano de las soluciones linealmente independientes de la ecuación homogénea correspondiente. Al determinar de (7.38) todas las $c_i(x) = \varphi_i(x)$ por cuadraturas, hallamos

$$c_i(x) = \int \varphi_i(x')\, dx' + \tilde{c}_i \ .$$

Ejemplo Consideremos la ecuación

$$y'' + y = \frac{1}{\cos x} \ .$$

La solución general de la ecuación homogénea correspondiente es $y = c_1 \cos(x) + c_2 \operatorname{sen} x$. Variemos c_1 y c_2:

$$y = c_1(x) \cos x + c_2(x) \operatorname{sen} x \ .$$

$c_1(x)$ y $c_2(x)$ se determinan del sistema (7.38):

$$\begin{aligned} c_1'(x) \cos x + c_2'(x) \operatorname{sen} x &= 0 \ , \\ -c_1'(x) \operatorname{sen} x + c_2'(x) \cos x &= \frac{1}{\cos x} \ , \end{aligned}$$

de donde

$$c_1'(x) = -\frac{\operatorname{sen} x}{\cos x} \ , \quad c_1(x) = \log|\cos x| + \tilde{c}_1 \ ;$$

$$c_2(x) = 1 \ , \quad c_2(x) = x + \tilde{c}_2 \ .$$

La solución general de la ecuación inicial es:

$$y = \tilde{c}_1 \cos x + \tilde{c}_2 \operatorname{sen} x + \cos x \log|\cos x| + x \operatorname{sen} x \ .$$

De este modo, si se conocen n soluciones particulares linealmente independientes de la ecuación homogénea correspondiente, se puede, por el método de variación de constantes, integrar la ecuación no homogénea

$$L[y] = f(x) \ .$$

Si se conoce, en cambio, solamente k (donde $k < n$) soluciones linealmente independientes $y_1,\ y_2,\ \dots\ ,y_k$ de la ecuación homogénea correspondiente, entoces, como ya fue señalado,

el cambio de variables permite reducir su orden hasta $n-k$, conservando su linealidad. Obsérvese que si $k=n-1$, el orden de la ecuación se reduce a 1, y la ecuación lineal de primer orden siempre se puede integrar en cuadraturas.

Análogamente se pueden utilizar k soluciones de la ecuación no homogénea $\tilde{y}_1$, $\tilde{y}_2$, ... ,$\tilde{y}_k$, puesto que sus diferencias son ya soluciones de la ecuación homogénea correspondiente. En efecto,

$$L[\tilde{y}_j] \equiv f(x) \ , \qquad L[\tilde{y}_p] \equiv f(x) \ ;$$

por lo tanto,

$$L[\tilde{y}_j - \tilde{y}_p] \equiv L[\tilde{y}_j] - L[\tilde{y}_p] \equiv f(x) - f(x) \equiv 0 \ .$$

Si las soluciones particulares de la ecuación homogénea correspondiente

$$(\tilde{y}_1 - \tilde{y}_k) \ , (\tilde{y}_2 - \tilde{y}_k) \ , \ldots \ , (\tilde{y}_{k-1} - \tilde{y}_k) \ , \tag{7.39}$$

son linealmente independientes, entonces el orden de la ecuación $L[y] = f(x)$ puede ser reducido hasta $n-(k-1)$. Es evidente que las otras diferencias $\tilde{y}_j - \tilde{y}_k$ son combinaciones lineales de las soluciones (7.39):

$$\tilde{y}_j - \tilde{y}_p = (\tilde{y}_j - \tilde{y}_k) - (\tilde{y}_p - \tilde{y}_k) \ ,$$

y, por consiguiente, no pueden ser utilizadas para la reducción ulterior del orden.

Señalemos otro método, el *método de Cauchy*, para hallar la solución particular de la ecuación lineal no homogénea

$$L[y(x)] = f(x) \ . \tag{7.40}$$

En este método se supone conocida la solución $K(x,s)$, que depende de un parámetro, de la ecuación homogénea correspondiente $L[y(x)]=0$, y que satisface las condiciones

$$K(s,s) = K'(s,s) = \ldots = K^{(n-2)}(s,s) = 0 \tag{7.41}$$

$$K^{(n-1)}(s,s) = 1 \ . \tag{7.42}$$

No es difícil comprobar que en este caso

$$y(x) = \int_{x_0}^{x} K(x,s) f(s) \, ds \ , \tag{7.43}$$

será solución particular de la ecuación (7.40), que satisface las condiciones iniciales nulas

$$y(x_0) = y'(x_0) = \ldots = y^{(n-1)}(x_0) = 0 \ .$$

En efecto, derivando[2] (7.43) y teniendo en cuenta las condiciones (7.41) y (7.42), se obtiene

$$\begin{aligned} y'(x) &= \int_{x_0}^{x} K'_x(x,s) f(s)\, ds \ , \\ y''(x) &= \int_{x_0}^{x} K''_x(x,s) f(s)\, ds \ , \\ &\vdots \\ y^{(n-1)}(x) &= \int_{x_0}^{x} K_x^{(n-1)}(x,s) f(s)\, ds \ , \\ y^{(n)}(x) &= \int_{x_0}^{x} K_x^{(n)}(x,s) f(s)\, ds + f(x) \ . \end{aligned} \tag{7.44}$$

El subíndice x en la función K indican que las derivadas son tomadas respecto a esa variable. Sustituyendo (7.43) y (7.44) en la ecuación (7.40), obtenemos

$$\int_{x_0}^{x} L[K(x,s)] f(s)\, ds + f(x) \equiv f(x) \ ,$$

y puesto que $K(x,s)$ es solución de la ecuación homogénea correspondiente tenemos que $L[K(x,s)] \equiv 0$ lo cual demuestra que la función $y(x) = \int_{x_0}^{x} K(x,s) f(s)\, ds$ es solución de $L[y(x)] = f(x)$.

La solución $K(x,s)$ puede ser tomada de la solución general $y = \sum_{i=1}^{n} c_i y_i$ de la ecuación homogénea, si se escogen las constantes arbitrarias c_i de manera que se cumplan las condiciones (7.41) y (7.42).

Ejemplo Para la ecuación

$$y'' + \alpha^2 y = f(x) \ , \tag{7.45}$$

la solución general es $y = c_1 \cos ax + c_2 \operatorname{sen} ax$. Las condiciones (7.41) y (7.42) conducen a las siguientes ecuaciones:

$$\begin{aligned} c_1 \cos as + c_2 \operatorname{sen} as &= 0 \ , \\ -a c_1 \operatorname{sen} as + a c_2 \cos as &= 1 \ . \end{aligned}$$

Por lo tanto,

$$c_1 = -\frac{\operatorname{sen} as}{a} \ , \qquad c_2 = -\frac{\cos as}{a} \ ,$$

y la solución buscada $K(x,s)$ tiene la forma

$$K(x,s) = \frac{1}{a} \operatorname{sen} a(x-s) \ .$$

[2]Debemos de considerar la regla de diferenciación de integrales:

$$\frac{d}{dx}\left[\int_{\phi_1(x)}^{\phi_2(x)} F(x,s)\, ds\right] = \int_{\phi_1(x)}^{\phi_2(x)} \frac{\partial F(x,s)}{\partial x}\, ds + F(\phi_1(x),x)\frac{d\phi_1}{dx} - F(\phi_2(x),x)\frac{d\phi_2}{dx} \ .$$

La solución de la ecuación (7.45) que satisface las condiciones iniciales nulas, según (7.43) se puede representar en la forma

$$y(x) = \frac{1}{a}\int_{x_0}^{x} \operatorname{sen} a(x'-s)f(s)\,ds\ .$$

Se puede dar una interpretación física a la función $K(x,s)$ y a la solución de la ecuación lineal con segundo miembro en la forma (7.43). Aquí será más cómodo designar la variable independiente por la letra t.

En muchos problemas, la solución $y(t)$ de la ecuación

$$y^{(n)} + p_1(t)y^{(n-1)} + \ldots + p_n(t)y = f(t)\ , \tag{7.46}$$

describe el desplazamiento de cierto sistema, y la función $f(t)$ es la fuerza que actúa en este sistema; t es el tiempo.

Supongamos primeramente que, para $t < s$, el sistema se encuentra en estado de reposo, y que su desplazamiento se efectúa debido a la fuerza $f_\varepsilon(t)$, diferente de cero sólo en el intervalo $s < t < s+\varepsilon$, y cuyo impulso es igual a 1:

$$\int_s^{s+\varepsilon} f_\varepsilon(\tau)\,d\tau = 1\ . \tag{7.47}$$

Designemos por $y_\varepsilon(t)$ la solución de la ecuación

$$y^{(n)} + p_1(t)y^{(n-1)} + \ldots + p_n(t)y = f_\varepsilon(t)\ .$$

Se comprueba fácilmente la existencia del límite $y_\varepsilon(t)$ cuando $\varepsilon \to 0$, el cual no depende de la función $f_\varepsilon(t)$, si suponemos que ésta no cambia su signo. En efecto,

$$y_\varepsilon(x) = \int_{t_0}^{t} K(t,s)f_\varepsilon(s)\,ds\ .$$

Aplicando el teorema del valor medio para $t > s+\varepsilon$, obtenemos

$$y_\varepsilon(x) = K(t, s+\varepsilon^*)\int_s^{s+\varepsilon} f_\varepsilon(\tau)\,d\tau = K(t,s+\varepsilon)\ ,$$

donde $0 < \varepsilon^* - \varepsilon$; por lo tanto,

$$\lim_{\varepsilon\to 0} y_\varepsilon(t) = K(t,s)\ .$$

Por ello, es natural llamar a la función $K(t,s)$ *funcion de influencia* del impulso instantáneo en el momento $t = s$. También se le conoce como la *función de Green* del sistema.

Dividiendo el intervalo (t_0, t) mediante los puntos s_i con $i = 0, 1, 2, \ldots, m$ en m partes iguales de longitud $\Delta s = (t-t_0)/m$, representamos la función $f(t)$ en (7.46) como una suma de las funciones $f_i(t)$, donde $f_i(t)$ es diferente de cero sólo en el i-ésimo intervalo $s_{i-1} < t < s_i$. En éste, $f_i(t)$ coincide con la función $f(t)$:

$$f(t) = \sum_{i=1}^{m} f_i(t)\ .$$

Debido al principio de superposición, la solución de la ecuación (7.46) tiene la forma

$$y(t) = \sum_{i=1}^{m} y_i(t) \ ,$$

donde y_i son soluciones de la ecuación

$$y^{(n)} + p_1(t)y^{(n-1)} + \ldots + p_n(t)y = f_i(t) \ ,$$

con condiciones iniciales nulas. Si m es suficientemente grande, la solución $y_i(t)$ se puede considerar como función de influencia del impulso instantáneo de intensidad $f_i(s_i)\Delta s$. Por consiguiente,

$$y(t) \approx \sum_{i=1}^{m} K(t, s_i) f(s_i)\Delta s \ .$$

Pasando al límite cuando $m \to \infty$, se obtiene la solución de la ecuación (7.46) con condiciones iniciales nulas, en la forma

$$y(t) = \int_{t_0}^{t} K(t, s) f(s)\, ds \ ,$$

la cual demuestra que la influencia de la fuerza de acción continua se puede considerar como superposición de las influencias de impulsos separados.

7.6 Ecuaciones lineales no homogéneas con coeficientes constantes y ecuación de Euler.

Al resolver ecuaciones lineales no homogéneas con coeficientes constantes, en muchos casos es posible escoger una solución particular sin dificultad, reduciendo así el problema a la integración de la ecuación homogénea correspondiente.

Supongamos, por ejemplo, que el segundo miembro es un polinomio de grado s y que , por lo tanto, la ecuación tiene la forma

$$a_0\, y^{(n)} + a_1\, y^{(n-1)} + \ldots + a_{n-1}y' + a_n y = A_0\, x^s + A_1\, x^{s-1} + \ldots + A_s \ , \tag{7.48}$$

donde todas las a_j y las A_i son constantes.

Si $a_n \neq 0$, entonces existe una solución particular de la ecuación (7.48) que tiene también la forma de polinomio de grado s. En efecto, sustituyendo

$$y = B_0\, x^s + B_1\, x^{s-1} + \ldots + B_s \ ,$$

en la ecuación (7.48) y comparando coeficientes de iguales potencias de x en ambos miembros, se obtiene un sistema de ecuaciones lineales para la determinación de los coeficientes B_i, que es siempre resoluble si $a_n \neq 0$:

$$a_n B_o = A_0 \ , \quad B_0 = \frac{A_0}{a_n} \ ,$$

$$a_n B_1 + s a_{n-1} B_0 = A_1 \ ,$$

de donde se determina B_1

$$a_n B_2 + (s-1) a_{n-1} B_1 + s(s-1) a_{n-2} B_0 = A_2 \ ,$$

de donde se determina B_2, y seguimos así hasta

$$a_n B_s + \ldots = A_s \ ,$$

de donde se determina B_s.

De esta manera, si $a_n \neq 0$ existe una solución particular que tiene la forma de polinomio cuyo grado es igual al grado del polinomio del segundo miembro.

Supongamos ahora que el coeficiente $a_n = 0$ y, para mayor generalidad, esciogemos que también $a_{n-1} = a_{n-2} = \ldots = a_{n-\alpha+1} = 0$, pero $a_{n-\alpha} \neq 0$, o sea, que $k = 0$ es raíz de multiplicidad α de la ecuación característica; además, el caso $\alpha = 1$ no se excluye. Entonces, la ecuación (7.48) toma la forma

$$a_0\, y^{(n)} + a_1\, y^{(n-1)} + \ldots + a_{n-\alpha} y^{(\alpha)} = A_0\, x^s + A_1\, x^{s-1} + \ldots + A_s \ . \tag{7.49}$$

Haciendo $y^{(\alpha)} = z$, llegamos al caso anterior y, en consecuencia, hay una solución particular de la ecuación (7.49), para la cual

$$y^{(\alpha)} = B_0\, x^s + B_1\, x^{s-1} + \ldots + B_s \ .$$

Esto significa que y es un polinomio de grado $s + \alpha$; además , los términos de grado menor o igual a $\alpha - 1$ de dicho polinomio tendrán coeficientes constantes arbitrarios, que pueden ser, en particular, escogidos iguales a cero. Entonces, la solución particular de la forma:

$$y = x^\alpha \left(B_0\, x^s + B_1\, x^{s-1} + \ldots + B_s \right) \ .$$

Ejemplo Consideremos la ecuación

$$y'' + y = x^2 + x \ . \tag{7.50}$$

La solución particular tiene la forma

$$y = B_0 x^2 + B_1 x + B_2 \ .$$

Sustituyendo en la ecuación (7.50) e igualando los coeficientes de los términos de igual grado respecto a x, obtenemos

$$B_0 = 1 \ , \quad B_1 = 1 \ , \quad B_2 = -2 \ , \quad \tilde{y} = x^2 + x - 2 \ .$$

La solución general es

$$y = c_1 \cos x + c_2 \operatorname{sen} x + x^2 + x - 2 \ .$$

Consideremos ahora la ecuación lineal no homogénea de la forma

$$a_0\, y^{(n)} + a_1\, y^{(n-1)} + \ldots + a_n y = e^{px}\left(A_0\, x^s + A_1\, x^{s-1} + \ldots + A_s\right)\ , \tag{7.51}$$

donde todas las a_j las A_i son constantes. Como fue indicado anteriormente, el cambio de variables $y = e^{px}z$ reduce la ecuación (7.51) a la forma

$$e^{px}\left[b_0\, z^n + b_1\, z^{n-1} + \ldots + b_n z\right] = e^{px}\left(A_0\, x^s + A_1\, x^{s-1} + \ldots + A_s\right)\ ,$$

o bien

$$b_0\, z^n + b_1\, z^{n-1} + \ldots + b_n z = A_0\, x^s + A_1\, x^{s-1} + \ldots + A_s\ , \tag{7.52}$$

donde todas las b_i son constantes.

La solución particular de la ecuación (7.52), si $b_n \neq 0$ tiene la forma

$$\tilde{z} = \left(B_0\, x^s + B_1\, x^{s-1} + \ldots + B_s\right)\ ;$$

por lo tanto, la solución particular de la ecuación (7.51) será

$$\tilde{y} = e^{px}\left(B_0\, x^s + B_1\, x^{s-1} + \ldots + B_s\right)\ .$$

La condición $b_n \neq 0$ significa que $\tilde{k} = 0$ no es raíz de la ecuación característica

$$b_0\, \tilde{k}^n + b_1\, \tilde{k}^{n-1} + \ldots + b_n = 0\ . \tag{7.53}$$

Por consiguiente, $k = p$ no es raíz de la ecuación característica

$$a_0\, k^n + a_1\, k^{n-1} + \ldots + a_n = 0\ , \tag{7.54}$$

puesto que las raíces de estas ecuaciones están ligadas por la dependencia $k = \tilde{k} + p$.

Si $\tilde{k}$, en cambio, es raíz de multiplicidad α de la ecuación característica (7.53) o, en otras palabras, $k = p$ es raíz de la misma multiplicidad α de la ecuación característica (7.54), entonces las soluciones particulares de las ecuaciones (7.52) y (7.51) tienen respectivamente las formas

$$\begin{aligned}\tilde{z} &= x^{\alpha}\left(B_0\, x^s + B_1\, x^{s-1} + \ldots + B_s\right)\ ,\\ \tilde{y} &= x^{\alpha}e^{px}\left(B_0\, x^s + B_1\, x^{s-1} + \ldots + B_s\right)\ .\end{aligned}$$

De esta manera, si el segundo miembro de la ecuación diferencial lineal con coeficientes constantes tiene la forma

$$e^{px}\left(A_0\, x^s + A_1\, x^{s-1} + \ldots + A_s\right)\ ,$$

y si p no es raíz de la ecuación característica, la solución particular debe buscarse en la misma forma

$$\tilde{y} = e^{px}\left(B_0\, x^s + B_1\, x^{s-1} + \ldots + B_s\right)\ .$$

Si, en cambio, p es raíz de multiplicidad α de la ecuación característica (este caso se denomina *singular* o *resonante*), la solución particular debe ser buscada en la forma

$$\tilde{y} = x^{\alpha} e^{px} \left(B_0\, x^s + B_1\, x^{s-1} + \ldots + B_s \right) \ .$$

Ejemplo Consideremos la ecuación

$$y'' - y = e^{3x}(x-2) \ .$$

La solución particular debe ser buscada en la forma

$$\tilde{y} = x e^{3x}(B_0\, x^{+} B_1) \ .$$

Obsérvese que nuestros razonamientos son válidos también si las p son complejas; por eso, si el segundo miembro de la ecuación diferencial lineal tiene la forma

$$e^{px} \left[P_s(x) \cos qx + Q_s(x) \operatorname{sen} qx \right] \ , \tag{7.55}$$

donde uno de los dos polinomios $P_s(x)$ o $Q_s(x)$ tiene grado s y el otro, no mayor que s, entonces reduciendo según las fórmulas de Euler las funciones trigonométricas a la forma exponencial, obtenemos en el segundo miembro

$$e^{(p+iq)x} R_s(x) + e^{(p-iq)x} T_s(x) \ , \tag{7.56}$$

donde R_s y T_s son polinomios de grado s.

A cada sumando del segundo miembro se le puede aplicar la regla anteriormente indicada, es decir, si $p \pm iq$ no son raíces de la ecuación característica, la solución particular se puede buscar en la misma forma que el segundo miembro (7.56); si, en cambio, $p \pm iq$ son raíces de multiplicidad α de la ecuación característica, la solución particular debe multiplicarse además por x^{α}.

Si volvemos a las funciones trigonométricas, esta regla se puede formular así:

a) Si $p \pm iq$ no son raíces de la ecuación característica, la solución particular debe buscarse en la forma

$$\tilde{y} = e^{px} \left[\tilde{P}_s(x) \cos(qx) + \tilde{Q}_s(x) \operatorname{sen}(qx) \right] \ ,$$

donde $\tilde{P}_s(x)$ y $\tilde{Q}_s(x)$ son polinomios de grado s con coeficientes indeterminados.

Obsérvese que si uno de los polinomios de grado $P_s(x)$ ó $Q_s(x)$ tienen un grado menor que s, e incluso, en particular, es identicamente nulo, de todos modos ambos polinomios $\tilde{P}_s(x)$ y $\tilde{Q}_s(x)$ tendrán, en general, grado s.

b) Si $p \pm iq$ son raices de multiplicidad α de la ecuación característica, la solución particular debe ser buscada en la forma

$$\tilde{y} = x^{\alpha} e^{px} \left[\tilde{P}_s(x) \cos(qx) + \tilde{Q}_s(x) \operatorname{sen}(qx) \right] \ ,$$

Ejemplo Consideremos la ecuación

$$y'' + 4y' + 4y = \cos(2x) \ .$$

Como los números $\pm 2i$ no son raíces de la ecuación característica, buscamos la solución particular en la forma

$$\tilde{y} = A\cos 2x + B\sin 2x \ .$$

Ejemplo Consideremos la ecuación

$$y'' + 4y = \cos(2x) \ .$$

Como los números $\pm 2i$ son raíces simples de la ecuación característica, buscamos la solución particular en la forma

$$\tilde{y} = x\left[A\cos 2x + B\,\mathrm{sen}\,2x\right] \ .$$

Ejemplo Consideremos la ecuación

$$y'''' + 2y'' + y = \mathrm{sen}(x) \ .$$

Puesto que los números $\pm i$ son raíces dobles de al ecuación característica, la solución particular se busca en la forma

$$\tilde{y} = x^2\left[A\cos x + B\,\mathrm{sen}\,x\right] \ .$$

Ejemplo Consideremos la ecuación

$$y'' + 2y'' + 2y = e^{-x}\left(x\cos x + 3\,\mathrm{sen}\,x\right) \ .$$

Debido a que los números $-1 \pm i$ son raíces simples de la ecuación característica, la solución particular debe buscarse en la forma

$$\tilde{y} = xe^{-x}\left[(A_0x + A_1)\cos x + (B_0x + B_1)\,\mathrm{sen}\,x\right] \ .$$

En muchos casos, al buscar soluciones particulares de ecuaciones lineales con coeficientes constantes con segundos miembros de la forma (7.55), es conveniente pasar a las funciones exponenciales.

Por ejemplo, en la ecuación

$$y'' - 2y' + y = \cos x \ ,$$

se puede trasformar $\cos x$ por la fórmula de Euler, o aún de modo más sencillo, considerar la ecuación

$$y'' - 2y' + y = e^{ix} \ , \tag{7.57}$$

la parte real de cuya solución debe satisfacer la ecuación original.

La solución particular de la ecuación (7.57) se puede buscar en la forma

$$y = Ae^{ix} \ .$$

Entonces

$$A = \frac{i}{2} \ , y = \frac{i}{2}\left(\cos x + i \operatorname{sen} x\right) \ .$$

La solución particular de la ecuación original es

$$\tilde{y_1} = \operatorname{Re} y = -\frac{1}{2} \operatorname{sen} x \ .$$

7.7 Integración de las ecuaciones diferenciales por medio de series.

El problema de la integración de ecuaciones lineales homogéneas de n-ésimo orden

$$p_0(x)y^{(n)} + p_1(x)y^{(n-1)} + \ldots + p_n(x)y = 0 \ , \tag{7.58}$$

se reduce a elegir n, o por lo menos $n-1$ soluciones linealmente independientes. Sin embargo, las soluciones particulares se escogen con facilidad sólo en casos excepcionales. En casos más complejos las soluciones particulares son buscadas en forma de suma de una serie $\sum_{i=1}^{\infty} a_i\varphi_i(x)$, sobre todo en forma de suma de una serie de potencias o de una serie generalizada de potencias.

Las condiciones bajo las cuales existen soluciones en forma de suma de una serie de potencia o de una serie generalizada de potencias, se establecen comúnmente por métodos de la teoría de funciones de variables complejas, la cual suponemos desconocida por el lector. Los teoremas fundamentales se darán sin demostración y aplicados a las ecuaciones de segundo orden, las cuales se encuentran con mayor frecuencia en la práctica.

Teorema 7.9 Sobre la propiedad analítica de la solución
Si p_0, $p_1(x)$ y $p_2(x)$ son funciones analíticas de x en un entorno del punto $x = x_0$ y $p_0(x_0) \neq 0$, entonces las soluciones de la ecuación

$$p_0(x)y'' + p_1(x)y' + p_2(x)y = 0 \tag{7.59}$$

son también funciones analíticas en cierto entorno del mismo punto; por lo tanto, la solución de la ecuación (7.59) se puede buscar de la forma

$$y = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \ldots + a_n(x - x_n)^n + \ldots \ .$$

Teorema 7.10 Sobre el desarrollo de la solución en una serie generalizada de potencias.
Si la ecuación (7.59) satisface las condiciones del teorema anterior, pero $x = x_0$ es un cero de orden finito s de la función $p_0(x)$, cero de orden $s-1$ o superior de la función $p_1(x)$ $(s > 1)$ y cero de orden no inferior $s-2$ del coeficinete $p_2(x)$ (si $s > 2$), entonces existe por lo menos una solución no trivial de la ecuación (7.59) en forma de suma de una serie generalizada de potencias

$$y = a_0(x - x_0)^k + a_1(x - x_0)^{k+1} + a_2(x - x_0)^{k+2} + \ldots + a_n(x - x_n)^{k+n} + \ldots \ . \qquad (7.60)$$

donde k es un número real que puede ser entero y fraccionario, positivo o negativo.

La segunda solución linealmente independiente de (7.60) por regla general, tiene también la forma de suma de una serie generalizada de potencias, pero a veces puede contener además el producto de una serie generalizada de potencia por $\log(x - x_0)$.

En problemas concretos se puede proceder sin los dos teoremas formulados más arriba, sobre todo porque en el enunciado de éstos no se establecen las regiones de convergencia de las series consideradas. Com mayor frecuencia en problemas concretos se escoge una serie de potencias o una serie generalizada de potencias que satisfaga formalmente la ecuación diferencial, o sea, que al sustituírla en la ecuación considerada de orden n (7.58) la transforme en una identidad, si suponemos la convergencia de la serie y la posibilidad de su derivación término a término n veces. Al obtener formalmente la solución en forma de serie, se investiga su convergencia y la posibilidad de su derivación término a término n veces. En la región donde la serie converge y permite su derivación término a término n veces, la misma no solamente satisface formalmente la ecuación, sino que su suma es en realidad la solución buscada.

Ejemplo Consideremos la ecuación

$$y'' - xy = 0 \ . \qquad (7.61)$$

Busquemos la solución en forma de una serie de potencias

$$y = \sum_{n=0}^{\infty} a_n x^n \ .$$

Basándonos en los teorema anteriores, o derivando esta serie formalmente término a término dos veces y sustituyendo en la ecuación (7.61), obtenemos

$$\sum_{n=0}^{\infty} a_n n(n-1)x^{n-2} - x\sum_{n=0}^{\infty} a_n x^n \equiv 0 \ .$$

Igualando los coeficientes de iguales potencias de x en ambos miembros de la identidad, obtenemos $a_2 = 0$, $3 \cdot 2a_3 - a_0 = 0$, de donde $a_3 = a_0/(2 \cdot 3)$; $4 \cdot 3a_4 - a_1 = 0$, de donde $a_4 = a_1/(3 \cdot 4)$; $5 \cdot 4a_5 - a_2 = 0$, de donde $a_5 = a_2/(4 \cdot 5)$,$\ldots$, $n(n-1)a_n - a_{n-3} = 0$, de donde $a_n = a_{n-3}/(n-1)n$, $\ldots$. Por consiguiente,

$$a_{3n-1} = 0 \ , \qquad a_{3n} = \frac{a_0}{2 \cdot 3 \cdot 5 \cdot 6 \ldots (3n-1)3n} \ ,$$

$$a_{3n+1} = \frac{a_1}{3 \cdot 4 \cdot 6 \cdot 7 \ldots 3n(3n+1)} \ , \qquad \text{con } n = 1, 2, 3, \ldots ,$$

a_0 y a_1 permanecen arbitrarios. De esta manera,

$$\begin{aligned} y = a_0 \left[1 + \frac{x^3}{2 \cdot 3} + \frac{x^6}{2 \cdot 3 \cdot 5 \cdot 6} + \ldots + \frac{x^{3n}}{2 \cdot 3 \cdot 5 \cdot 6 \ldots (3n-1)3n} + \ldots \right] + \\ + a_1 \left[x + \frac{x^4}{3 \cdot 4} + \frac{x^7}{3 \cdot 4 \cdot 6 \cdot 7} + \ldots + \frac{x^{3n+1}}{3 \cdot 4 \cdot 6 \cdot 7 \ldots 3n(3n+1)} + \ldots \right] \ . \end{aligned} \tag{7.62}$$

El radio de convergencia de esta serie de potencias es infinito. Por consiguiente, la suma de la serie (7.62) para valores cualesquiera de x es solución de la ecuación considerada.

Capítulo 8

Sistemas de ecuaciones diferenciales.

versión 1.1-011002[1]

8.1 Conceptos generales.

La ecuación de movimiento de una partícula de masa m bajo la acción de la fuerza $\vec{F}(t, \vec{r}, \dot{\vec{r}})$, es

$$m\frac{d^2\vec{r}}{dt^2} = \vec{F}(t, \vec{r}, \dot{\vec{r}}) \ ;$$

proyectando sobre los ejes de coordenadas, ésta puede ser sustituida por un sistema de tres ecuaciones escalares de segundo orden:

$$\begin{aligned} m\frac{d^2x}{dt^2} &= F_x(t, x, y, z, \dot{x}, \dot{y}, \dot{z}) \ , \\ m\frac{d^2y}{dt^2} &= F_y(t, x, y, z, \dot{x}, \dot{y}, \dot{z}) \ , \\ m\frac{d^2z}{dt^2} &= F_z(t, x, y, z, \dot{x}, \dot{y}, \dot{z}) \ , \end{aligned}$$

o por un sistema de seis ecuaciones de primer orden, si consideramos como funciones desconocida no sólo las coordenadas x, y, z de la partícula, sino también las proyecciones $\dot{x}$, $\dot{y}$, $\dot{z}$ de su velocidad

$$\begin{aligned} \dot{x} &= u \ , \\ \dot{y} &= v \ , \\ \dot{z} &= w \ ; \\ m\dot{u} &= F_x(t, x, y, z, u, v, w) \ , \\ m\dot{v} &= F_y(t, x, y, z, u, v, w) \ , \\ m\dot{w} &= F_z(t, x, y, z, u, v, w) \ . \end{aligned}$$

En este caso, por lo general, se dan la posición inicial del punto $x(t_0) = x_0$, $y(t_0) = y_0$, $z(t_0) = z_0$, y la velocidad inicial $u(t_0) = u_0$, $v(t_0) = v_0$, $w(t_0) = w_0$.

[1]Este capítulo está basado en el tercer capítulo del libro: *Ecuaciones diferenciales y cálculo variacional* de L. Elsgoltz, editorial MIR

Se puede demostrar un teorema sobre existencia y unicidad de la solución del sistema de ecuaciones diferenciales

$$\begin{aligned} \frac{dx_1}{dt} &= f_1(t, x_1, x_2, \dots, x_n) \ , \\ \frac{dx_2}{dt} &= f_2(t, x_1, x_2, \dots, x_n) \ , \\ &\vdots \\ \frac{dx_n}{dt} &= f_n(t, x_1, x_2, \dots, x_n) \ , \end{aligned} \tag{8.1}$$

que satisfacen las condiciones iniciales

$$x_i(t_0) = x_{i0} \ , \quad (i = 1, 2, \dots, n) \ . \tag{8.2}$$

Enumeremos las condiciones suficientes para la existencia y unicidad de la solución del sistema (8.1) con las condiciones iniciales (8.2):

i) Continuidad de todas las funciones f_i en un entorno de las condiciones iniciales.

ii) Cumplimiento de la condición de Lipschitz para todas las funciones f_i en todos sus argumentos, a partir del segundo, en dicho entorno.

La segunda condición se puede cambiar por una más grosera: la existencia de las derivadas parciales

$$\frac{\partial f_i}{\partial x_j} \ , \quad (i = 1, 2, \dots, n) \ ,$$

acotadas en valor absoluto.

La solución $x_1(t)$, $x_2(t)$, ... , $x_n(t)$ del sistema de ecuaciones diferenciales es una función vectorial n-dimensional, que denotaremos abreviadamente por $X(t)$. Con esta notación, el sistema (8.1) se puede escribir en la forma

$$\frac{dX}{dt} = F(t, X) \ ,$$

donde F es una función vectorial con coordenadas $(f_1, f_2, \dots, f_n)$ y las condiciones iniciales, en la forma $X(t_0) = X_0$, donde X_0 es un vector de n-dimensiones, con coordenadas $(x_{10}, x_{20}, \dots, x_{n0})$. La solución

$$x_1 = x_1(t) \ , \quad x_2 = x_2(t) \ , \quad \dots \quad x_n = x_n(t) \ ,$$

o, más compactamente, $X = X(t)$, del sistema de ecuaciones, determina en el espacio euclidiano de coordenadas t, x_1, x_2, ... ,x_n, cierta curva llamada *curva integral*. Cuando se cumplen las condiciones i) y ii) del teorema de existencia y unicidad, por cada punto de dicho espacio pasa una sola curva integral, y el conjunto de éstas forma una familia dependiente de n parámetros. Como parámetros de esta familia se pueden tomar, por ejemplo, los valores iniciales x_{10}, x_{20}, ... ,x_{n0}.

Se puede dar otra interpretación de las soluciones

$$x_1 = x_1(t) \ , \quad x_2 = x_2(t) \ , \quad \dots \ , x_n = x_n(t) \ ,$$

o, en la forma más compacta, $X = X(t)$, que es particularmente cómoda si los segundos miembros del sistema (8.1) no dependen explícitamente del tiempo.

En el espacio euclidiano con coordenadas x_1, x_2, ... ,x_n, la solución $x_1 = x_1(t)$, $x_2 = x_2(t)$, ... , $x_n = x_n(t)$ determina la ley de movimiento por cierta trayectoria según la variación del parámetro t, el cual en esta interpretación se considera como el tiempo. Así, la derivada dX/dt será la velocidad del movimiento de la partícula, y dx_1/dt, dx_2/dt, ... , dx_n/dt, las coordenadas de la velocidad en ese punto. En esta interpretación, muy natural y cómoda en ciertos problemas físicos y mecánicos, el sistema

$$\frac{dx_i}{dt} = f_i(t, x_1, x_2, \dots , x_n) \ , \qquad (i = 1, 2, \dots , n) \ , \tag{8.1}$$

o bien

$$\frac{dX}{dt} = F(t, X) \ ,$$

se llama generalmente *dinámicos*; el espacio de coordenadas x_1, x_2, ... ,x_n, *espacio de fase*, y la curva $X = X(t)$, *trayectoria de fases*.

El sistema dinámico (8.1) determina en un momento dado t en el espacio x_1, x_2, ... ,x_n un campo de velocidades. Si la función vectorial F depende explícitamente de t, entonces el campo de velocidades cambia con el tiempo, y las trayectorias de fases pueden intersectarse. Si la función vectorial F o, lo que es lo mismo, todas las funciones f_i no dependen explícitamente de t, el campo de velocidades es estacionario, es decir, no cambia en el tiempo, y el movimiento será permanente.

En este último caso, si las condiciones del teorema de existencia y unicidad se cumplen, por cada punto del espacio de fase $(x_1, x_2, \dots , x_n)$ pasará una sola trayectoria. En efecto, en este caso por cada trayectoria $X = X(t)$ se realizan infinitos movimiento diferentes entre $X = X(t + c)$, donde c es una constante arbitraria. Es fácil comprobar esto realizando la sustitución de variables $t_1 = t + c$, con lo cual el sistema dinámico no cambia su forma:

$$\frac{dX}{dt_1} = F(X) \ ,$$

Por consiguiente, $X = X(t)$ será su solución que, expresada en las variables anteriores, será $X = X(t + c)$.

Si por un punto X_0 del espacio de fase, en el caso considerado, pasarán dos trayectorias

$$X = X_1(t) \quad \text{y} \quad X = X_2(t) \ , \quad X_1(\tilde{t}_0) = X_2(\tilde{\tilde{t}}_0) = X_0 \ ,$$

entonces, tomando en cada una de ellas el movimiento para el cual el punto X_0 se alcanza en el momento $t = t_0$, o sea, considerando las soluciones

$$X = X_1(t - t_0 + \tilde{t}_0) \quad \text{y} \quad X = X_2(t - t_0 + \tilde{\tilde{t}}_0) \ ,$$

se obtendría una contradicción con el teorema de existencia y unicidad, puesto que las dos soluciones diferentes $X_1(t - t_0 + \tilde{t}_0)$ y $X_2(t - t_0 + \tilde{\tilde{t}}_0)$ satisfarían a la misma condición inicial $X(t_0) = X_0$.

8.2 Integración de un sistema de ecuaciones diferenciales por reducción a una sola ecuación de mayor orden.

Uno de los métodos fundamentales de integración de sistemas de ecuaciones diferenciales consiste en lo siguiente: de las ecuaciones del sistema (8.1) y de las ecuaciones obtenidas derivando éstas, se excluyen todas las funciones desconocidas, excepto una, para cuya determinación se obtiene una ecuación diferencial de orden mayor. Integrando dicha ecuación, se halla una de las funciones desconocidas. Las funciones desconocidas restantes se determinan, en lo posible sin integración partiendo de las ecuaciones originales y de las obtenidas por derivación. Veamos algunos ejemplos:

Ejemplo Consideremos el sistema de ecuaciones

$$\frac{dx}{dt} = y \ , \qquad \frac{dy}{dt} = x \ .$$

Derivemos una de las ecuaciones, por ejemplo, la primera, $\dfrac{d^2x}{dt^2} = \dfrac{dy}{dt}$; eliminando $\dfrac{dy}{dt}$ mediante la segunda ecuación, se obtiene $\dfrac{d^2x}{dt^2} - x = 0$, de donde $x = c_1e^t + c_2e^{-t}$. Utilizando la primera ecuación, obtenemos $y = \dfrac{dx}{dt} = c_1e^t - c_2e^{-t}$.

Hemos determinado y sin integrar, mediante la primera ecuación. Si hubiéramos determinado y de la segunda ecuación,

$$\frac{dy}{dt} = x = c_1e^t + c_2e^{-t} \ , \quad y = c_1e^t - c_2e^{-t} + c_3 \ ,$$

entonces habríamos introducido soluciones superfluas, puesto que la sustitución directa en el sistema original muestra que las funciones $x = c_1e^t + c_2e^{-t}$ e $y = c_1e^t - c_2e^{-t} + c_3$ satisfacen al sistema no para c_3 cualesquiera sino para $c_3 = 0$.

Ejemplo Consideremos el sistema de ecuaciones

$$\frac{dx}{dt} = 3x - 2y \ , \tag{8.3a}$$

$$\frac{dy}{dt} = 2x - y \ . \tag{8.3b}$$

Derivemos la segunda ecuación:

$$\frac{d^2y}{dt^2} = 2\frac{dx}{dt} - dydt \ . \tag{8.4}$$

De las ecuaciones (8.3b) y (8.4) se determina x y $\dfrac{dx}{dt}$:

$$x = \frac{1}{2}\left(\frac{dy}{dt} + y\right) \ , \tag{8.5}$$

$$\frac{dx}{dt} = \frac{1}{2}\left(\frac{d^2y}{dt^2} + \frac{dy}{dt}\right) .$$

Sustituyendo en (8.3a), obtenemos

$$\frac{d^2y}{dt^2} - 2\frac{dy}{dt} + y = 0 .$$

Integrando la ecuación lineal homogénea con coeficientes constantes $y = e^t(c_1 + c_2t)$ y sustituyendo en (8.5) se halla $x(t)$:

$$x = \frac{1}{2}e^t(2c_1 + c_2 + 2c_2t) .$$

Si aplicamos el proceso de eliminación de funciones desconocidas al sistema

$$\frac{dx_i}{dt} = \sum_{j=1}^{n} a_{ij}(t)x_j , \quad (i = 1, 2, \dots , n) ,$$

llamado lineal homogéneo, entonces, como es fácil comprobar, la ecuación de n-ésimo orden

$$\frac{d^nx_1}{dt^n} = \phi\left(t, x_1, \frac{dx_1}{dt}, \dots , \frac{d^{n-1}x_1}{dt^{n-1}}\right) , \tag{8.6}$$

también será lineal homogénea. Además, si todos los coeficientes a_{ij} son constantes, la ecuación (8.6) será también lineal homogénea con coeficientes constantes. Una observación análoga se cumple también para el sistema lineal no homogéneo

$$\frac{dx_i}{dt} = \sum_{j=1}^{n} a_{ij}(t)x_j + f_i(t) , \quad (i = 1, 2, \dots , n) ,$$

para el cual la ecuación (8.6) será una ecuación lineal no homogénea de n-ésimo orden.

8.3 Sistema de ecuaciones diferenciales lineales.

Un sistema de ecuaciones diferenciales se llama *lineal*, si es lineal respecto a todas las funciones desconocidad y a sus derivadas. El sistema de n ecuaciones lineales de primer orden, escrito en la forma normal, es

$$\frac{dx_i}{dt} = \sum_{j=1}^{n} a_{ij}(t)x_j + f_i(t) , \quad (i = 1, 2, \dots , n) , \tag{8.7}$$

o, en forma vectorial,

$$\frac{dX}{dt} = AX + F , \tag{8.8}$$

donde X es un vector n-dimensional de coordenadas $x_1(t)$, $x_2(t)$, ... , $x_n(t)$; F es un vector n-dimensional de coordenadas $f_1(t)$, $f_2(t)$, ... ,$f_n(t)$. Es conveniente en lo sucesivo representar dichos vectores como matrices de una columna:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad , \quad F = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \quad ,$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \quad , \quad \frac{dX}{dt} = \begin{bmatrix} \dfrac{dx_1}{dt} \\ \dfrac{dx_2}{dt} \\ \vdots \\ \dfrac{dx_n}{dt} \end{bmatrix} \quad .$$

Según la regla del producto de matrices, las filas del primer factor deben multiplicarse por la columna del segundo: por tanto,

$$AX = \begin{bmatrix} \sum_{j=1}^{n} a_{1j}x_j \\ \sum_{j=1}^{n} a_{2j}x_j \\ \vdots \\ \sum_{j=1}^{n} a_{nj}x_j \end{bmatrix} \quad , \quad AX + F = \begin{bmatrix} \sum_{j=1}^{n} a_{1j}x_j + f_1 \\ \sum_{j=1}^{n} a_{2j}x_j + f_2 \\ \vdots \\ \sum_{j=1}^{n} a_{nj}x_j + f_n \end{bmatrix} \quad .$$

La igualdad de matrices significa la igualdad de todos sus elementos, por lo cual una sola ecuación matricial (8.8). o bien

$$\begin{bmatrix} \dfrac{dx_1}{dt} \\ \dfrac{dx_2}{dt} \\ \vdots \\ \dfrac{dx_n}{dt} \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} a_{1j}x_j + f_1 \\ \sum_{j=1}^{n} a_{2j}x_j + f_2 \\ \vdots \\ \sum_{j=1}^{n} a_{nj}x_j + f_n \end{bmatrix} \quad .$$

es equivalente al sistema (8.7).

Si todas las funciones $a_{ij}(t)$ y $f_i(t)$ en (8.7) son continuas en el intervalo $a \leq t \leq b$, entoces en un entorno suficientemente pequeño de cada punto $(t_0, x_{10}, x_{20}, \dots , x_{n0})$, donde $a \leq t_0 \leq b$

se cumplen las conmdiciones del teorema de existencia y unicidad y, en consecuencia, por cada punto con estas propiedades pasa una sola curva integral del sistema (8.7).

En efecto, en el caso considerado los segundos miembros del sistema (8.7) son continuos, y sus derivadas parciales con respecto a cualquier x_j son acotadas, puesto que dichas derivadas son iguales a los coeficientes $a_{ij}(t)$, continuos en el intervalo $a \leq t \leq b$.

Definamos el *operador lineal* L por la igualdad

$$L[X] = \frac{dX}{dt} - AX \ ;$$

entonces la ecuación (8.8) puede escribirse en la forma aún más compacta

$$L[X] = F \ . \tag{8.9}$$

Si todas las $f_i(t) \equiv 0$ con $i = 1, 2, \ldots, n$ o, lo que es lo mismo, la matriz $F = 0$, el sistema (8.7) se llama *lineal homogéneo*. En forma compacta, el sistema lineal homogéneo tiene la forma

$$L[X] = 0 \ . \tag{8.10}$$

El operador L posee las dos propiedades siguientes:

(i) $L[cX] \equiv cL[X]$, donde c es una constante arbitraria.

(ii) $L[X_1 + X_2] \equiv L[X_1] + L[X_2]$.

Un corolario de (i) y (ii) es

$$L\left[\sum_{i=1}^{m} c_i X_i\right] \equiv \sum_{i=1}^{m} c_i L[X_i] \ ,$$

donde las c_i son constantes arbitrarias.

Teorema 8.1 Si X es solución del sistema lineal homogéneo $L[X] = 0$, entonces cX, donde c es una constante arbitraria, es también solución de dicho sistema.

Teorema 8.2 La suma $X_1 + X_2$ de dos soluciones X_1 y X_2 del sistema de ecuaciones lineal homogéneo es solución de dicho sistema.

Corolario de los teoremas anteriores. La combinación lineal $\sum_{i=1}^{m} c_i X_i$ de las soluciones X_1, X_2, ... ,X_m del sistema $L[X] \equiv 0$ con coefientes constantes arbitrarios es solución de dicho sistema.

Teorema 8.3 Si el sistema lineal homogéneo (8.10) con coeficientes resles a_{ij} tiene una solución compleja $X = U + iV$, las partes real e imaginaria

$$U = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \ , \quad V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \ ,$$

son por separado soluciones de dicho sistema.

Los vectores $X_1, X_2, \ldots, X_n$, donde

$$X_i = \begin{bmatrix} x_{1i}(t) \\ x_{2i}(t) \\ \vdots \\ x_{ni}(t) \end{bmatrix} ,$$

se llaman *linealmente dependientes* en el intervalo $a \leq t \leq b$, si existen $\alpha_1, \alpha_2, \ldots, \alpha_n$ constantes tales que

$$\alpha_1 X_1 + \alpha_2 X_2 + \ldots + \alpha_n X_n \equiv 0 \tag{8.11}$$

cuando $a \leq t \leq b$, y al menos un $\alpha_i \neq 0$. Si, en cambio, la identidad (8.11) se cumple sólo cuando $\alpha_1 = \alpha_2 = \ldots = \alpha_n = 0$, entonces los vectores $X_1, X_2, \ldots, X_n$ se llaman *linealmente independientes*.

Obsérvese que la identidad vectorial (8.11) es equivalente a las n identidades:

$$\begin{aligned} \sum_{i=1}^{n} \alpha_i x_{1i}(t) &\equiv 0 \ , \\ \sum_{i=1}^{n} \alpha_i x_{2i}(t) &\equiv 0 \ , \\ &\vdots \\ \sum_{i=1}^{n} \alpha_i x_{ni}(t) &\equiv 0 \ . \end{aligned} \tag{8.12}$$

Si los vectores X_i $(i = 1, 2, \ldots, n)$ son linealmente dependientes y, por lo tanto, existe un sistema no trivial de α_i (es decir, no todas las α_i son iguales a cero) que satisface al sistema (8.12) de n ecuaciones lineales homogéneas con respecto a α_i, entonces el determinante del sistema (8.12)

$$W = \begin{vmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ x_{21} & x_{21} & \ldots & x_{2n} \\ \vdots & \vdots & \ldots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nn} \end{vmatrix} ,$$

debe ser igual a cero para todos los valores de t del intervalo $a \leq t \leq b$. Este determinante se llama *wronskiano* del sistema de vectores $X_1, X_2, \ldots, X_n$.

Teorema 8.4 Si el wronskiano W de las soluciones $X_1, X_2, \ldots, X_n$ del sistema de ecuaciones lineales homogéneo (8.10) con coeficientes continuos $a_{ij}(t)$ en el intervalo $a \leq t \leq b$, es igual a cero por lo menos en un punto $t = t_0$ de dicho intervalo, entonces el conjunto de soluciones $X_1, X_2, \ldots, X_n$ son linealmente dependientes en el intervalo mencionado y, por consiguiente, $W = 0$ en dicho intervalo.

Observación. Este teorema no se extiende a vectores arbitrarios X_1, X_2, ... , X_n, que no son soluciones de un sistema (8.10) con coeficientes continuos.

Teorema 8.5 La combinación lineal $\sum_{i=1}^{n} c_i X_i$ de n soluciones linealmente independientes X_1, X_2, ... , X_n del sistema lineal homogéneo (8.10) con coeficientes continuos $a_{ij}(t)$ en el intervalo $a \leq t \leq b$, es solución general de este sistema en dicho intervalo.

Teorema 8.6 Si $\tilde{X}$ es solución del sistema lineal no homogéneo

$$L[X] = F \ , \tag{8.9}$$

y X_1 es solución del sistema homogéneo correspondiente $L[X] = 0$, entonces la suma $X_1 + \tilde{X}$ es también solución del sistema no homogéneo $L[X] = F$.

Teorema 8.7 La solución general en el intervalo $a \leq t \leq b$ del sistema no homogéneo (8.9) con coeficientes $a_{ij}(t)$ y segundo miembro $f_i(t)$ continuos en dicho intervalo, es igual a la suma de la solución general $\sum_{i=1}^{n} c_i X_i$ del sistema homogéneo correspondiente y de una solución particular $\tilde{X}$ del sistema no homogéneo considerado.

Teorema 8.8 principio de superposición.
La solución del sistema de ecuaciones lineales

$$L[X] = \sum_{i=1}^{m} F_i \ , \quad F_i = \begin{bmatrix} f_{1i}(t) \\ f_{2i}(t) \\ \vdots \\ f_{ni}(t) \end{bmatrix} \ , \tag{8.13}$$

es la suma $\sum_{i=1}^{m} X_i$ de las soluciones X_i de las ecuaciones

$$L[X_i] = F_i \ , (i = 1, 2, \dots , m) \ . \tag{8.14}$$

Observación. El teorema anterior puede extenderse también al caso cuando $m \to \infty$, si la serie $\sum_{i=1}^{\infty} X_i$ converge y puede ser derivada término a término.

Teorema 8.9 Si el sistema de ecuaciones lineales

$$L[X] = U + iV \ ,$$

donde

$$U = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \ , \quad V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \ ,$$

con funciones reales $a_{ij}(t)$, $u_i(t)$, $v_i(t)$ $(i, \ j = 1, 2, \dots , n)$, tiene la solución

$$X = \tilde{U} + i\tilde{V} \ , \ quad \tilde{U} = \begin{bmatrix} \tilde{u}_1 \\ \tilde{u}_2 \\ \vdots \\ \tilde{u}_n \end{bmatrix} \ , \quad \tilde{V} = \begin{bmatrix} \tilde{v}_1 \\ \tilde{v}_2 \\ \vdots \\ \tilde{v}_n \end{bmatrix} \ ,$$

entonces la parte real de $\tilde{U}$ de la solución y su parte imaginaria $\tilde{V}$ son, respectivamente, soluciones de las ecuaciones

$$L[X] = U \quad \text{y} \quad L[X] = V \ .$$

Capítulo 9

Preliminares.

versión 2.0-19 agosto 2002[1].

9.1 Programas y funciones.

Programa de ortogonalidad en Octave

En esta sección nosotros escribiremos algunos programas simples usando Octave y C++. En nuestro primer ejemplo, llamado `orthog`, probaremos si dos vectores son ortogonales calculando su producto punto. Este simple programa lo bosquejamos a continuación

- Inicializamos los vectores $\vec{a}$ y $\vec{b}$.
- Evaluamos el producto punto como $\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$.
- Imprimir el producto punto y establecer si los vectores son ortogonales.

Primero consideremos la versión en Octave del programa que llamaremos `orthog.m`. Las primeras líneas de `orthog` son:

```
% orthog - Programa para probar si un par de vectores es ortogonal.
% Supondremos vectores en 3D.
clear all; % Borra la memoria
```

Las primeras dos líneas son comentarios; si tipeamos `help orthog` desde al línea de comandos, Octave desplegará estas líneas. El comando `clear all` en la tercera línea borra la memoria. Las próximas líneas del programa

```
%* Inicializa los vectores a y b
a= input('Entre el primer vector: ');
b= input('Entre el segundo vector: ');
```

Los vectores son entrados usando el comando `input` en estas líneas. Los comentarios que comienzan con `%*` son aquellos que corresponden al bosquejo del programa que hicimos. En las líneas siguientes se evalúa el producto punto.

[1]Este capítulo está basado en el primer capítulo del libro: *Numerical Methods for Physics, second edition* de Alejandro L. Garcia, editorial Prentice Hall

```
%* Evalua el producto punto como la suma sobre el producto de los elementos
adotb=0;
for i=1:3
  adotb=adotb+a(i)*b(i);
end
```

El ciclo `for`, usando el índice `i`, recorre las componentes de los vectores. Una manera hábil de hacer lo mismo podría ser usar

```
%* Evalua el producto punto como la suma sobre el producto de los elementos
adotb=a*b' ;
```

En este caso, hemos usado la multiplicación de matrices de Octave para calcular el producto punto como el producto vetorial de el vector fila `a` y el columna `b'` (donde hemos usado el operador Hermítico conjugado `'`). Las últimas líneas del programa

```
%* Imprime el producto punto y si los vetores son ortogonales
if(adotb==0)
  disp('Los vectores son ortogonales');
else
  disp('Los vectores no son ortogonales');
  printf('Producto punto = %g \n', adotb);
end
```

De acuerdo al valor de `adotb` el program despliega una de las dos posibles respuestas. A continuación la salida al ejecutar el `help` del programa

```
octave> help orthog
orthog is the file: /home/jrogan/orthog.m

 orthog - Programa para probar si un par de vectores es ortogonal.
 Supondremos vectores en 3D.


Additional help for builtin functions, operators, and variables
is available in the on-line version of the manual.  Use the command
‘help -i <topic>’ to search the manual index.

Help and information about Octave is also available on the WWW
at http://www.che.wisc.edu/octave/octave.html and via the
help-octave@bevo.che.wisc.edu mailing list.
```

Ahora ejecutamos el programa con diferentes vectores.

```
octave> orthog
Entre el primer vector: [1 1 1]
Entre el segundo vector: [1 -2 1]
Los vectores son ortogonales
```

```
octave> orthog
Entre el primer vector: [1 1 1]
Entre el segundo vector: [2 2 2]
Los vectores no son ortogonales
Producto punto = 5
```

Programa de ortogonalidad en C++.

Ahora consideremos la versión en C++ del programa `orthog`, el cual prueba si dos vectores son ortogonales mediante el cálculo de su producto punto. Las primeras líneas son

```
//  orthog - Programa para probar si un par de vectores es ortogonal.
// Supondremos vectores en 3D.
#include <iostream.h>
```

Las primeras líneas son comentarios que nos recuerdan lo que el programa hace. La tercera línea incluye las definiciones del `iostream.h` el cual nos permite la entrada y salida. En la proxima línea comienza el programa

```
main()
{
```

El paréntesis se cierra con uno al final del programa. Las primeras líneas del cuerpo del programa son

```
  //* Inicializa los vectores a y b
  double a[3+1], b[3+1] ;
  cout << "Entre el primer vector"<< endl;
  for(int i=1; i<=3; i++) {
    cout << " a["<<i<<"] = ";
    cin >> a[i] ;
  }
  cout << "Entre el segundo vector"<< endl;
  for(int i=1; i<=3; i++) {
    cout << " b["<<i<<"] = ";
    cin >> b[i] ;
  }
```

Los comentarios que empiezan con `//*` corresponden a un punto en el bosquejo inicial que hicmos del programa. Los vectores `a` y `b` son declarados arreglos de punto flotante con cuatro elementos, tres componentes más una componente de índice cero no usada. La instrucción de salida despliega sobre la pantalla:

```
Entre el primer vector
 a[1] =
```

La instrucción de entrada lee los valores desde el teclado dentro del arreglo `a[i]`. Las próximas líneas

```
  //* Evalua el producto punto como la suma sobre el producto de los elementos
  double adotb=0.0 ;
  for(int i=1; i<=3; i++) {
    adotb += a[i]*b[i];
  }
```

Finalmente las líneas

```
  //* Imprime el producto punto y si los vetores son ortogonales
  if(adotb==0) {
    cout << "Los vectores son ortogonales"<< endl ;
  } else {
    cout << "Los vectores no son ortogonales"<< endl ;
    cout << "El producto punto = " << adotb << endl ;
  }
```

Despliega los resultados. Aquí la salida típica del programa.

```
jrogan@pucon:~$ orthog
Entre el primer vector
 a[1] = 1
 a[2] = 1
 a[3] = 1
Entre el segundo vector
 b[1] = 1
 b[2] = -2
 b[3] = 1
Los vectores son ortogonales
```

Programa de interpolación en Octave.

Es bien sabido que dado tres pares (x, y), se puede encontrar una cuadrática que pasa por los puntos deseados. Hay varias maneras de encontrar el polinomio y varias maneras de escribirlo. La forma de Lagrange del polinomio es

$$p(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3 \ , \qquad (9.1)$$

donde (x_1, y_1), (x_2, y_2), (x_3, y_3), son los tres puntos por los que queremos pasar. Comúnmente tales polinomios son usados para interpolar entre los puntos dados. A continuación el bosquejo de un programa simple de interpolación `interp`

- Inicializa los puntos (x_1, y_1), (x_2, y_2) y (x_3, y_3) para ser ajustados por el polinomio.
- Establece el intervalo de la interpolación (desde $x_{\text{mín}}$ hasta $x_{\text{máx}}$)
- Encontrar y^* para los valores deseados de x^*, usando la función `intrpf`.
- Graficar al curva dada por (x^*, y^*), y marcar los puntos originales.

Las primeras líneas del programa

```
% interp - Programa para interpolar datos usando
% el polinomio de Lagrange cuadratico para tres puntos dados.
clear all;
%* Inicializa los puntos a ser ajustados con una cuadratica
disp('Entre los puntos como pares x,y (e.g., [1 2])');
for i=1:3
  temp =input('Ingrese el punto: ');
  x(i)=temp(1);
  y(i)=temp(2) ;
end
%* Establece el intervalo de interpolacion (desde x_min a x_max)
xr = input ('Ingrese el intervalo de valores de x como [x_min x_man]: ');
```

Aquí el programa lee los tres pares (x, y) y el intervalo de valores entre los cuales será interpolado.

Los valores interpolados $y^* = p(x^*)$ son calculados por la función `intrpf` desde $x^* = x_{\text{mín}}$ a $x^* = x_{\text{máx}}$. Estos valores de y^* (`yi`) son calculados en el ciclo.

```
%* Encontrar yi para los valores deseados de interpolacion xi
% usando la funcion intrpf
nplot= 100; % Numero de puntos para la curva interpolada
for i=1:nplot
  xi(i) = xr(1)+(xr(2)-xr(1))*(i-1)/(nplot-1);
  yi(i) = intrpf(xi(i), x, y); % Usando intrpf para interpolar
end
```

Finalmente, los resultados son graficados usando las funciones gráfica de Octave.

```
%* Grafica la curva dada por (xi,yi) y marca los puntos originales
xlabel('x');
ylabel('y');
title('Interpolacion de tres puntos');
gset nokey;
plot(x,y, '*',xi,yi,'-');
```

Los puntos de la interpolación (x^*, y^*) son graficados con línea segmentada y los datos originales con círculos, ver figura (9.1) El trabajo real es hecho por la función `intrpf`. Un bosquejo de lo que queremos que haga esta función a continuación.

- Entrada: $\vec{x} = [x_1\ x_2\ x_3]$, $\vec{y} = [y_1\ y_2\ y_3]$, y x^*.
- Salida: y^*.
- Cálculo de $y^* = p(x^*)$ usando el polinomio de Lagrange (9.1).

Las funciones en Octave están implementadas como en la mayoría de los lenguajes, excepto que aquí cada función tiene que ir en un archivo separado. El nombre del archivo debe coincidir con el nombre de la función (la función `intrpf` está en el archivo `intrpf.m`).

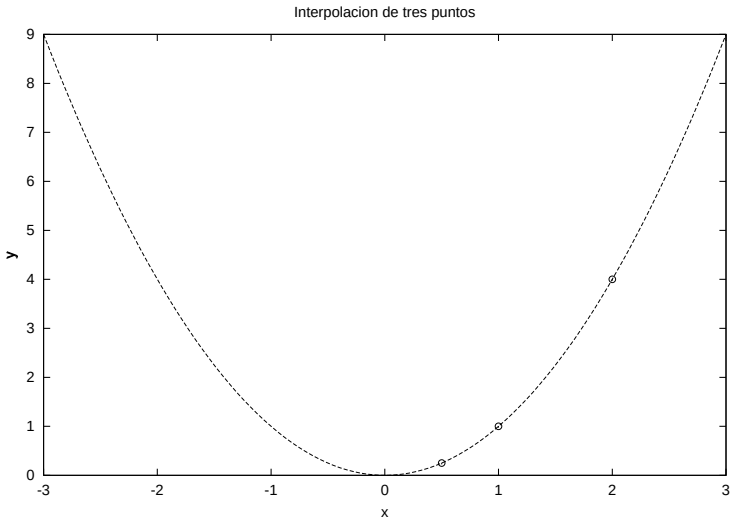


Figura 9.1: Salida grafica del programa `interp`.

```
function yi=intrpf(xi,x,y)
% Funcion para interpolar entre puntos
% usando polinomio de Lagrange (cuadratico)
% Entradas
%   x   Vector de las coordenadas x de los puntos dados (3 valores)
%   y   Vector de las coordenadas y de los puntos dados (3 valores)
% Salida
%   yi  El polinomio de interpolacion evaluado en xi
```

La función `intrpf` tiene tres argumentos de entrada y uno de salida. El resto de la función es directa, sólo evalúa el polinomio definido en (9.1). El cuerpo de la función a continuación

```
yi = (xi-x(2))*(xi-x(3))/((x(1)-x(2))*(x(1)-x(3)))*y(1) ...
   + (xi-x(1))*(xi-x(3))/((x(2)-x(1))*(x(2)-x(3)))*y(2) ...
   + (xi-x(1))*(xi-x(2))/((x(3)-x(1))*(x(3)-x(2)))*y(3);
return;
```

Programa de interpolación en C++.

Las primeras líneas de la versión en C++ del programa `interp` son

```
// interp - Programa para interpolar datos usando
// el polinomio de Lagrange cuadratico para tres puntos dados.
#include "NumMeth.h"

double intrpf(double xi, double x[], double y[]);
main()
{
```

Comentarios más una instrucción para incluir el archivo de encabezamiento `NumMeth.h`, listado a continuación

```
#include <iostream.h>
#include <fstream.h>
#include <math.h>
#include <stdlib.h>
```

La declaración `double intrpf(..)` afirma que el programa pretende llamar una función `intrpf` la cual tiene tres argumentos de tipo `double` y devuelve un `double`. Las próximas líneas del programa

```
  //* Inicializa los puntos a ser ajustados con una cuadratica
  double x[3+1], y[3+1] ;
  cout << "Entre los puntos como pares x,y (e.g., [1 2])" << endl ;
  for(int i=1; i<=3; i++) {
    cout << "x["<<i<<"] = ";
    cin>>  x[i];
    cout << "y["<<i<<"] = ";
    cin >> y[i];
  }


  //* Establece el intervalo de interpolacion (desde x_min a x_max)
 double xmin, xmax;
 cout <<"Entre el valor minimo de x: "; cin >> xmin ;
 cout <<"Entre el valor maximo de x: "; cin >> xmax ;
```

El programa pregunta por los puntos para ajustar el polinomio de Lagrange (9.1) y por el intervalo de interpolación. Lo siguiente, los arreglos `xi` y `yi` son declarados:

```
 //* Encontrar yi para los valores deseados de interpolacion xi
 // usando la funcion intrpf
 int nplot= 100; // Numero de puntos para la curva interpolada
 double * xi = new double[nplot+1] ; // Reserva memoria para
 double * yi = new double[nplot+1] ; // estos arreglos.
```

Estas líneas también podrían reemplazarse por

```
 const int nplot =100; // Numero de puntos para la curva interpolada
 double xi[nplot+1], yi[nplot+1] ;
```

En el primer caso hay asignamiento dinámico de memoria, `nplot` podría ser una entrada del programa. En el segundo caso `nplot` debe ser constante y para modificar el número de puntos debemos recompilar el programa, asignación estática.

Los valores interpolados son calculados en un `for`

```
 for(int i=1; i<=nplot;i++) {
  xi[i] = xmin+(xmax-xmin)*double(i-1)/double(nplot-1);
  yi[i] = intrpf(xi[i], x, y); // Usando intrpf para interpolar
 }
```

Notemos que `xi[1]`=$x_{\text{mín}}$, `xi[nplot]`=$x_{\text{máx}}$, con valores equiespaciados entre ellos. Los valores de `yi` ($y^* = p(x^*)$) son evaluados usando la ecuación (9.1) en la función `intrpf`. La salida del programa

```
//* Imprime las variables para graficar: x, y, xi, yi
ofstream xOut("x.txt"), yOut("y.txt"), xiOut("xi.txt"), yiOut("yi.txt");
for(int i =1; i <=3; i++) {
  xOut << x[i] << endl;
  yOut << y[i] << endl;
}
for(int i =1; i <=nplot; i++) {
  xiOut << xi[i] << endl;
  yiOut << yi[i] << endl;
}
```

Esto cuatro archivos de datos (`x.txt`, `y.txt`, etc) son creados.

Desgraciadamente , C++ carece de una biblioteca gráfica estandar así que necesitamos una aplicación gráfica adicional para graficar la salida. También podemos usar un pequeño *script* en Octave:

```
#!/usr/bin/octave
load x.txt; load y.txt; load xi.txt; load yi.txt;
%* Grafica la curva dada por (xi,yi) y marca los puntos originales
xlabel('x');
ylabel('y');
title('Interpolacion de tres puntos');
gset nokey;
plot(x,y, '*',xi,yi,'-');
pause
```

Al cual incluso podemos llamar desde el mismo programa mediante

```
system( "grafica.m" ) ;
```

La última línea del programa

```
delete [] xi, yi ; // Libera la memoria pedida con "new"
```

Esta línea no es absolutamente necesaria, por que al salir el programa liberara la memoria de todas maneras. Sin embargo se considera de buen estilo de programación limpiar uno la memoria que requirio durante la ejecución del programa.

La función `intrpf` la cual evalúa el polinomio de Lagrange comienza por las siguientes líneas

```
double intrpf( double xi, double x[], double y[])
{
  //  Funcion para interpolar entre puntos
  // usando polinomio de Lagrange (cuadratico)
```

```
// Entradas
//   x   Vector de las coordenadas x de los puntos dados (3 valores)
//   y   Vector de las coordenadas y de los puntos dados (3 valores)
// Salida
//   yi  El polinomio de interpolacion evaluado en xi
```

Especifica los argumentos de llamada y lo que devuelve. Todas las variables dentro de la función son *locales*. El C++ pasa las variables por defecto por valor, la función recibe una copia que se destruye cuando termina la función, si se desea pasar una variable `double a` por referencia debemos anteponerle el signo &, es decir, pasarla como `double &a`. De esta manera la función puede modificar el valor que tenía la variable en el programa principal.

El resto de la función

```
//* Calcula yi=p(xi) usando Polinomio de Lagrange

double yi = (xi-x[2])*(xi-x[3])/((x[1]-x[2])*(x[1]-x[3]))*y[1]
  + (xi-x[1])*(xi-x[3])/((x[2]-x[1])*(x[2]-x[3]))*y[2]
  + (xi-x[1])*(xi-x[2])/((x[3]-x[1])*(x[3]-x[2]))*y[3];
return yi ;
```

Estas líneas evalúan el polinomio. Inicialmente pondremos esta función en el mismo archivo, luego la podemos separar en otro archivo y escribir un `Makefile` que compile y linkee todo junto.

9.2 Errores numéricos.

9.2.1 Errores de escala.

Un computador almacena números de punto flotante usando sólo una pequeña cantidad de memoria. Típicamente, a una variable de precición simple (un `float` en C++) se le asigna 4 bytes (32 bits) para la representación del número, mientras que a una variable de doble precisión (`double` en C++, por defecto en Octave) usa 8 bytes. Un número de punto flotante es representado por su mantisa y su exponente (por ejemplo, para 6.625×10^{-27} la mantisa decimal es 6.625 y el exponente es -27). El formato IEEE para doble precisión usa 53 bits para almacenar la mantisa (incluyendo un bit para el signo) y lo que resta, 11 bit para el exponente. La manera exacta en que el computador maneja la representación de los números no es tan importante como saber el intervalo máximo de valores y el número de cifras significativas.

El intervalo máximo es el límite sobre la magnitud de los números de punto flotante impuesta por el número de bit usados para el exponente. Para precisión simple un valor típico es $2^{\pm 127} \approx 10^{\pm 38}$; para precisión double es típicamente $2^{\pm 1024} \approx 10^{\pm 308}$. Exceder el intervalo de la precisión simple no es difícil.. Consideremos, por ejemplo, la evaluación del radio de Bohr en unidades SI,

$$a_0 = \frac{4\pi\varepsilon_0\hbar^2}{m_e e^2} \approx 5.3 \times 10^{-11} \text{ [m]} . \tag{9.2}$$

Mientrás sus valores caen dentro del intervalo de un numero de precisión simple, el eintervalo es excedido en el cálculo del numerador ($4\pi\varepsilon_0\hbar^2 \approx 1.24\times10^{-78}$ [kg C^2 m]) y del denominador ($m_e e^2 \approx 2.34\times10^{-68}$ [kg C^2]). La mejor solución para lidiar con este tipo de dificultades de intervalo es trabajar en un conjunto de unidades naturales al problema (e.g. para problemas atómicos se trabaja en las distancias en angstroms, la carga en unidades de la carga del electrón).

Algunas veces los problemas de intervalo no son causados por la elección de las unidades sino porque los números en el problema son inherentemente grandes. Consideremos un importante ejemplo, la función factorial. Usando la definición

$$n! = n\times(n-1)\times(n-2)\times\ldots\times3\times2\times1\ ,$$

es fácil evaluar $n!$ en C++ como

```
double nFactorial=1;
for(int i=1; i <=n; i++) nFactorial *=i ;
```

donde `n` es un número dado.

En Octave, usando el operador dos puntos este cálculo puede ser realizado como

```
nFactorial = prod(1:n);
```

donde `prod(x)` es el producto de los elementos del vector `x` y `:n=[` 2— ... `n]`. Infortunadamente, debido a problemas de intervalo, no podemos calcular $n!$ para $n > 170$ usando estos métodos directos de evaluación (9.2.1).

Una solución común para trabajar con números grandes es usar su logaritmo. Para el factorial

$$\log(n!) = \log(n) + \log(n-1) + \ldots + \log(3) + \log(2) + \log(1)\ . \tag{9.3}$$

En Octave, esto puede ser evaluado como

```
log_nFactorial = sum( log(1:n) ) ;
```

donde `sum(x)` es la suma de los elementos del vector `x`. Sin embargo, este esquema es computacionalmente pesado si n es grando. Una mejor estrategia es combinar el uso de logaritmos con la fórmula de Stirling[2]

$$n! = \sqrt{2n\pi}n^n e^{-n}\left(1+\frac{1}{12n}+\frac{1}{288n^2}+\cdots\right) \tag{9.4}$$

o

$$\log(n!) = \frac{1}{2}\log(2n\pi) + nlog(n) - n + \log\left(1+\frac{1}{12n}+\frac{1}{288n^2}+\cdots\right)\ . \tag{9.5}$$

Esta aproximación puede ser usada cuando n es grande ($n > 30$), de otra manera es preferible la definición original.

Finalmente, si el valor de $n!$ necesita ser impreso, podemos expresarlo como

$$n! = (\text{mantisa})\times10^{(\text{exponente})}\ , \tag{9.6}$$

donde el exponente es la parte entera de $\log_{10}(n!)$, y la mantisa es 10^a donde a es la parte fracionaria de $\log_{10}(n!)$. Recordemos que la conversión entre logaritmo natural y logaritmo en base 10 es $\log_{10}(x) = log_{10}(e)\log(x)$.

[2]M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions* (New York: Dover 1972).

9.2.2 Errores de redondeo.

Supongamos que deseamos calcular numéricamente $f'(x)$, la derivada de una función conocida $f(x)$. En cálculo se aprendio que la fórmula para la derivada es

$$f'(x) = \frac{f(x+h) - f(x)}{h} \ , \tag{9.7}$$

en el límite en que $h \to 0$. ¿Qué sucede si evaluamos el lado derecho de esta expresión, poniendo $h = 0$?. Como el computador no entiende que la expresión es válida sólo como un límite, la división por cero tiene varios posibles salidas. El computador puede asignar el valor, `Inf`, el cual es un número de punto flotante especial reservado para representar el infinito. Ya que el numerador es también cero el computador podría evaluar el cuociente siendo indefinido (Not-a-Number), `NaN`, otro valor reservado. O el cálculo podría parar con un mensaje de error.

Claramente, poniendo $h = 0$ para evaluar (9.7) no nos dará nada útil, pero ¿ si le ponemos a h un valor muy pequeño, digamos $h = 10^{-300}$, usamos doble precisión? La respuesta aún será incorrecta debido a la segunda limitación sobre la representación de números con punto flotante: el número de dígitos en la mantisa. Para precisión simple, el número de dígitos significantes es típicamente 6 o 7 dígitos decimales; para doble precisión es sobre 16 dígitos. Así, en doble precisión, la operación $3 + 10^{-20}$ retorna una respuesta 3 por el redondeo; usando $h = 10^{-300}$ en la ecuación (9.7) casi con seguridad regresará 0 cuando evaluemos el númerador.

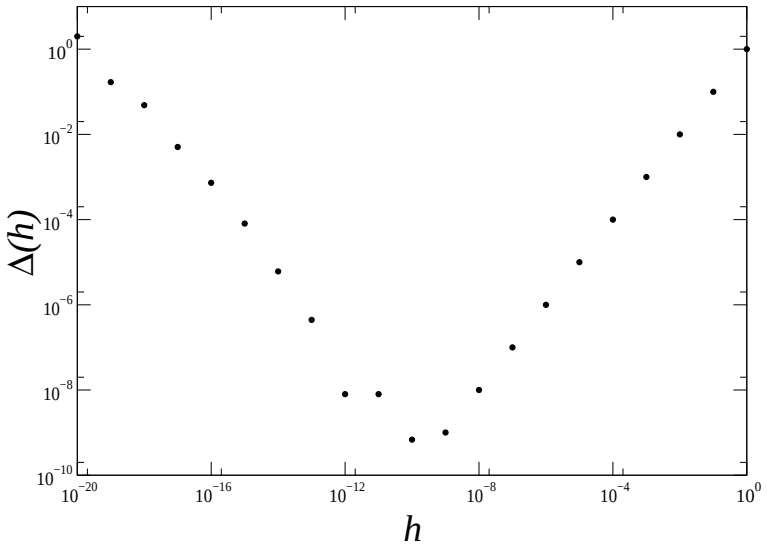


Figura 9.2: Error absoluto $\Delta(h)$, ecuación (9.8), versus h para $f(x) = x^2$ y $x = 1$.

La figura 9.2 ilustra la magnitud del error de redondeo en un cálculo típico de derivada. Definimos el error absoluto

$$\Delta(h) = \left| f'(x) - \frac{f(x+h) - f(x)}{h} \right| \ . \tag{9.8}$$

Notemos que $\Delta(h)$ decrece cuando h se hace más pequeño, lo cual es esperado dada que la ecuación (9.7) es exacta cuando $h \to 0$. Por debajo de $h = 10^{-10}$, el error comienza a incrementarse debido a efectos de redondeo. En valores menores de h el error es tan grande que la respuesta carece de sentido. Volveremos en el proximo capítulo a la pregunta como mejorar el cálculo de la derivada numericamente.

Para testear la tolerancia del redondeo, definimos ε_Γ como el más pequeño número que, cuando es sumado a uno, regresa un valor distinto de 1. En Octave, la función integrada `eps` devuelve $\varepsilon_\Gamma \approx 2.22 \times 10^{-16}$. En C++, el archivo de encabezamiento `<float.h>` define `DBL_EPSILON` como ε_Γ para doble precisión.

Debido a los errores de redondeo la mayoría de los cálculos científicos usan doble precisión. Las desventajas de la doble precisión son que requiere más memoria y que algunas veces (no siempre) es más costosa computacionalmente. Los procesadores modernos están construídos para trabajar en doble precisión, tanto que puede ser más lento trabajar en precisión simple. Usar doble precisión algunas veces sólo desplaza las dificultades de redondeo. Por ejemplo, el cálculo de la inversa de una matriz trabaja bien en simple precisión para matrices pequeñas de 50×50 elementos, pero falla por errores de redondeo para matrices mas grandes. La double precisión nos permite trbajar con matrices de 100×100, pero si necesitamos resolver sistemas aún más grandes debemos usar un algoritmo diferente. La mejor manera de trabajar es usar algoritmos robustos contra el error de redondeo.

Capítulo 10

Ecuaciones diferenciales ordinarias: Métodos básicos.

versión final 2.4-19 agosto 2002[1]

En este capítulo resolveremos uno de los primeros problemas considerados por un estudiante de física: el vuelo de un proyectil y, en particular, el de una pelota de *baseball*. Sin la resistencia del aire el problema es fácil de resolver. Sin embargo, incluyendo un arrastre realista, nosotros necesitamos calcular la solución numéricamente. Para analizar este problema definiremos primero la diferenciación numérica. De hecho antes de aprender Física uno aprende cálculo así que no debemos sorprendernos si este es nuestro punto de partida. En la segunda mitad del capítulo nos acuparemos de otro viejo conocido, el péndulo simple, pero sin la aproximación a ángulos pequeños. Interesantemente, problemas oscilatorios, tales como el péndulo, revelan una falla fatal en algunos de los métodos numéricos para resolver ecuaciones diferenciales ordinarias.

10.1 Movimiento de un proyectil.

10.1.1 Ecuaciones básicas.

Consideremos el simple movimiento de un proyectil, digamos un pelota de *baseball*. Para describir el movimiento nosotros debemos calcular el vector posición $\vec{r}(t)$ y el vector velocidad $\vec{v}(t)$ del proyectil. Las ecuaciones básicas de movimiento son

$$\frac{d\vec{v}}{dt} = \frac{1}{m}\vec{F}_a(\vec{v}\,) - g\hat{y}\ , \quad \frac{d\vec{r}}{dt} = \vec{v}\ , \tag{10.1}$$

donde m es la masa del proyectil. La fuerza debido a la resistencia del aire es $\vec{F}_a(\vec{v}\,)$, la aceleración gravitacional es g, e $\hat{y}$ es un vector unitario en la dirección y. El movimiento es bidimensional, tal que podemos ignorar la componente z y trabajar en el plano xy.

La resistencia del aire se incrementa con la velocidad del objeto, y la forma precisa para $\vec{F}_a(\vec{v}\,)$ depende del flujo alrededor del proyectil. Comúnmente, esta fuerza es aproximada por

$$\vec{F}_a(\vec{v}) = -\frac{1}{2}C_d\rho A\,|\,v\,|\,\vec{v}\ , \tag{10.2}$$

[1]Este capítulo está basado en el segundo capítulo del libro: *Numerical Methods for Physics, second edition* de Alejandro L. Garcia, editorial Prentice Hall.

donde C_d es el coeficiente de arrastre, ρ es la densidad del aire, y A es el área de la sección transversal del proyectil. El coeficiente de arrastre es un parámetro adimensional que depende de la geometría del proyectil —Mientras más aerodinámico el objeto, el coeficiente es menor.

Para una esfera suave de radio R moviéndose lentamente a través del fluido, el coeficiente de arrastre es dado por la Ley de Stokes,

$$C_d = \frac{12\nu}{Rv} = \frac{24}{\mathrm{Re}} \ , \tag{10.3}$$

donde ν es la viscosidad del fluido ($\nu \approx 1.5 \times 10^{-5}$ [m^2/s] para el aire) y $\mathrm{Re} = 2Rv/\nu$ es el adimensional *número de Reynolds*. Para un objeto del tamaño de un pelota de baseball moviéndose a través del aire, la ley de Stokes es válida sólo si la velocidad es menor que 0.2 [mm/s] ($\mathrm{Re} \approx 1$).

A velocidades altas (sobre 20 [cm/s], $\mathrm{Re} > 10^3$), la estela detrás de la esfera desarrolla vórtices y el coeficiente de arrastre es aproximadamente constante ($C_d \approx 0.5$) para un amplio intervalo de velocidades. Cuando el número de Reynolds excede un valor crítico, el flujo en la estela llega a ser turbulento y el coeficiente de arrastre cae dramáticamente. Esta reducción ocurre porque la turbulencia rompe la región de bajas presiones en la estela detrás de la esfera[2]. Para una esfera suave este número crítico de Reynolds es aproximadamente 3×10^5. Para una pelota de *baseball*, el coeficiente de arrastre es usualmente más pequeño que el de una esfera suave, porque las costuras rompen el flujo laminar precipitando el inicio de la turbulencia. Nosotros podemos tomar $C_d = 0.35$ como un valor promedio para el intervalo de velocidades típicas de una pelota de *baseball*.

Notemos que la fuerza de arrastre, ecuación (10.2), varía como el cuadrado de la magnitud de la velocidad ($\vec{F}_a \propto v^2$) y, por supuesto, actúa en la dirección opuesta a la velocidad. La masa y el diámetro de una pelota de *baseball* son 0.145 [kg] y 7.4 [cm]. Para una pelota de *baseball*, el arrastre y la fuerza gravitacional son iguales en magnitud cuando $v \approx 40$ [m/s].

Nosotros sabemos cómo resolver las ecuaciones de movimiento si la resistencia del aire es despreciable. La trayectoria es

$$\vec{r}(t) = \vec{r}_1 + \vec{v}_1 t - \frac{1}{2} g t^2 \hat{y} \ , \tag{10.4}$$

donde $\vec{r}_1 \equiv \vec{r}(0)$ y $\vec{v}_1 \equiv \vec{v}(0)$ son la posición y la velocidad inicial. Si el proyectil parte del origen y la velocidad inicial forma un ángulo θ con la horizontal, entonces

$$x_{\mathrm{máx}} = \frac{2v_1^2}{g} \operatorname{sen}\theta \cos\theta \ , \quad y_{\mathrm{máx}} = \frac{v_1^2}{2g} \operatorname{sen}^2\theta \ , \tag{10.5}$$

son el alcance horizontal y la altura máxima. El tiempo de vuelo es

$$t_{fl} = \frac{2v_1}{g} \operatorname{sen}\theta \ . \tag{10.6}$$

Nuevamente estas expresiones son válidas sólo cuando no hay resistencia con el aire. Es fácil demostrar que el máximo alcance horizontal se obtiene cuando la velocidad forma un ángulo de 45° con la horizontal. Deseamos mantener esta información en mente cuando construyamos nuestra simulación. Si se sabe la solución exacta para un caso especial, se debe comparar constantemente que el programa trabaje bien para este caso.

[2] D.J. Tritton, Physical Fluid Dynamics, 2d ed. (Oxford: Clarendon Press, 1988).

10.1.2 Derivada avanzada.

Para resolver las ecuaciones de movimiento (10.1) necesitamos un método numérico para evaluar la primera derivada. La definición formal de la derivada es

$$f'(t) \equiv \lim_{\tau \to 0} \frac{f(t+\tau) - f(t)}{\tau} \ , \tag{10.7}$$

donde τ es el incremento temporal o paso en el tiempo. Como ya vimos en el capítulo pasado esta ecuación debe ser tratada con cuidado. La figura 9.2 ilustra que el uso de un valor extremadamente pequeño para τ causa un gran error en el cálculo de $(f(t+\tau) - f(t))/\tau$. Específicamente, los errores de redondeo ocurren en el cálculo de $t+\tau$, en la evaluación de la función f y en la sustracción del numerador. Dado que τ no puede ser elegido arbitrariamente pequeño, nosotros necesitamos estimar la diferencia entre $f'(t)$ y $(f(t+\tau) - f(t))/\tau$ para un τ finito.

Para encontrar esta diferencia usaremos una expansión de Taylor. Como físicos usualmente vemos las series de Taylor expresadas como

$$f(t+\tau) = f(t) + \tau f'(t) + \frac{\tau^2}{2} f''(t) + \cdots \tag{10.8}$$

donde los puntos suspensivos indican términos de más alto orden que son usualmente despreciados. Una alternativa, forma equivalente de la serie de Taylor usada en análisis numérico es

$$f(t+\tau) = f(t) + \tau f'(t) + \frac{\tau^2}{2} f''(\zeta) \ , \tag{10.9}$$

donde ζ es un valor entre t y $t+\tau$. No hemos botado ningún término, esta expansión tiene un número finito de términos. El teorema de Taylor garantiza que existe *algún* valor ζ para el cual (10.9) es cierto, pero no sabemos cuál valor es éste.

La ecuación previa puede ser reescrita

$$f'(t) = \frac{f(t+\tau) - f(t)}{\tau} - \frac{1}{2}\tau f''(\zeta) \ , \tag{10.10}$$

donde $t \leq \zeta \leq t+\tau$. Esta ecuación es conocida como la fórmula de la *derivada derecha* o *derivada adelantada*. El último término de la mano derecha es el *error de truncamiento*; este error es introducido por cortar la serie de Taylor.

En otras palabras, si mantenemos el último término en (10.10), nuestra expresión para $f'(t)$ es exacta. Pero no podemos evaluar este término porque no conocemos ζ, todo lo que conocemos es que ζ yace en algún lugar entre t y $t+\tau$. Así despreciamos el término $f''(\zeta)$ (truncamos) y decimos que el error que cometemos por despreciar este término es el error de truncamiento. No hay que confundir éste con el error de redondeo discutido anteriormente. El error de redondeo depende del *hardware*, el error de truncamiento depende de la aproximación usada en el algoritmo. Algunas veces veremos la ecuación (10.10) escrita como

$$f'(t) = \frac{f(t+\tau) - f(t)}{\tau} + \mathcal{O}(\tau) \ , \tag{10.11}$$

donde el error de truncamiento es ahora especificado por su orden en τ, en este caso el error de truncamiento es lineal en τ. En la figura 9.2 la fuente de error predominante en estimar $f'(x)$ como $[f(x+h) - f(x)]/h$ es el error de redondeo cuando $h < 10^{-10}$ y es el error de truncamiento cuando $h > 10^{-10}$.

10.1.3 Método de Euler.

Las ecuaciones de movimiento que nosotros deseamos resolver numéricamente pueden ser escritas como:

$$\frac{d\vec{v}}{dt} = \vec{a}(\vec{r}, \vec{v}) \ , \quad \frac{d\vec{r}}{dt} = \vec{v} \ , \tag{10.12}$$

donde $\vec{a}$ es la aceleración. Notemos que ésta es la forma más general de las ecuaciones. En el movimiento de proyectiles la aceleración es sólo función de $\vec{v}$ (debido al arrastre), en otros problemas (e.g., órbitas de cometas) la aceleración dependerá de la posición.

Usando la derivada adelantada (10.11), nuestras ecuaciones de movimiento son

$$\frac{\vec{v}(t+\tau) - \vec{v}(t)}{\tau} + \mathcal{O}(\tau) = \vec{a}(\vec{r}(t), \vec{v}(t)) \ , \tag{10.13}$$

$$\frac{\vec{r}(t+\tau) - \vec{r}(t)}{\tau} + \mathcal{O}(\tau) = \vec{v}(t) \ , \tag{10.14}$$

o bien

$$\vec{v}(t+\tau) = \vec{v}(t) + \tau\vec{a}(\vec{r}(t), \vec{v}(t)) + \mathcal{O}(\tau^2) \ , \tag{10.15}$$

$$\vec{r}(t+\tau) = \vec{r}(t) + \tau\vec{v}(t) + \mathcal{O}(\tau^2) \ . \tag{10.16}$$

Notemos que $\tau\mathcal{O}(\tau) = \mathcal{O}(\tau^2)$. Este esquema numérico es llamado el *método de Euler*. Antes de discutir los méritos relativos de este acercamiento, veamos cómo sería usado en la práctica.

Primero, introducimos la notación

$$f_n = f(t_n) \ , \quad t_n = (n-1)\tau \ , \quad n = 1, 2, \ldots \tag{10.17}$$

tal que $f_1 = f(t = 0)$. Nuestras ecuaciones para el método de Euler (despreciando el término del error) ahora toman la forma

$$\vec{v}_{n+1} = \vec{v}_n + \tau\vec{a}_n \ , \tag{10.18}$$

$$\vec{r}_{n+1} = \vec{r}_n + \tau\vec{v}_n \ , \tag{10.19}$$

donde $\vec{a}_n = \vec{a}(\vec{r}_n, \vec{v}_n)$. El cálculo de la trayectoria podría proceder así:

1. Especifique las condiciones iniciales, $\vec{r}_1$ y $\vec{v}_1$.
2. Elija un paso de tiempo τ.
3. Calcule la aceleración dados los actuales $\vec{r}$ y $\vec{v}$.
4. Use las ecuaciones (10.18) y (10.19) para calcular los nuevos $\vec{r}$ y $\vec{v}$.
5. Vaya al paso 3 hasta que suficientes puntos de trayectoria hayan sido calculados.

El método calcula un conjunto de valores para $\vec{r}_n$ y $\vec{v}_n$ que nos da la trayectoria, al menos en un conjunto discreto de valores. La figura 10.1 ilustra el cálculo de la trayectoria para un único paso de tiempo.

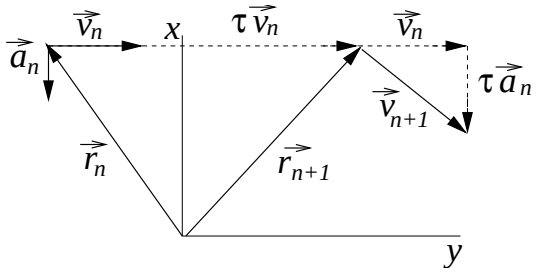


Figura 10.1: Trayectoria de una partícula después de un único paso de tiempo con el método de Euler. Sólo para efectos ilustrativos τ es grande.

10.1.4 Métodos de Euler-Cromer y de Punto Medio.

Una simple (y por ahora injustificada) modificación del método de Euler es usar las siguientes ecuaciones:

$$\vec{v}_{n+1} = \vec{v}_n + \tau\vec{a}_n \ , \tag{10.20}$$
$$\vec{r}_{n+1} = \vec{r}_n + \tau\vec{v}_{n+1} \ . \tag{10.21}$$

Notemos el cambio sutil: La velocidad actualizada es usada en la segunda ecuación. Esta fórmula es llamada *método de Euler-Cromer*[3]. El error de truncamiento es aún del orden de $\mathcal{O}(\tau^2)$, no parece que hemos ganado mucho. Interesantemente, veremos que esta forma es marcadamente superior al método de Euler en algunos casos.

En el *método del punto medio* usamos

$$\vec{v}_{n+1} = \vec{v}_n + \tau\vec{a}_n \ , \tag{10.22}$$
$$\vec{r}_{n+1} = \vec{r}_n + \tau\frac{\vec{v}_{n+1} + \vec{v}_n}{2} \ . \tag{10.23}$$

Notemos que hemos promediado las dos velocidades. Usando la ecuación para la velocidad en la ecuación de la posición, vemos que

$$\vec{r}_{n+1} = \vec{r}_n + \tau\vec{v}_n + \frac{1}{2}\vec{a}_n\tau^2 \ , \tag{10.24}$$

lo cual realmente hace esto lucir atractivo. El error de truncamiento es aún del orden de τ^2 en la ecuación velocidad, pero para la posición el error de truncamiento es ahora τ^3. Realmente, para el movimiento de proyectiles este método trabaja mejor que los otros dos. Infortunadamente, en otros sistemas físicos este método da resultados pobres.

10.1.5 Errores locales, errores globales y elección del paso de tiempo.

Para juzgar la precisión de estos métodos necesitamos distinguir entre errores de truncamiento locales y globales. Hasta ahora, el error de truncamiento que hemos discutido ha sido el error

[3]A. Cromer, "Stable solutions using the Euler approximation", *Am. J. Phys.*, **49** 455-9 (1981).

local, el error cometido en un único paso de tiempo. En un problema típico nosotros deseamos evaluar la trayectoria desde $t = 0$ a $t = T$. El número de pasos de tiempo es $N_T = T/\tau$; notemos que si reducimos τ, debemos tomar más pasos. Si el error local es $\mathcal{O}(\tau^n)$, entonces estimamos el error global como

$$\begin{aligned} \text{error global} &\propto N_T \times (\text{error local}) \\ &= N_T \mathcal{O}(\tau^n) = \frac{T}{\tau}\mathcal{O}(\tau^n) = T\mathcal{O}(\tau^{n-1}) \ . \end{aligned} \tag{10.25}$$

Por ejemplo, el método de Euler tiene un error local de truncamiento de $\mathcal{O}(\tau^2)$, pero un error global de truncamiento de $\mathcal{O}(\tau)$. Por supuesto, este análisis nos da sólo una estimación ya que no sabemos si los errores locales se acumularán o se cancelarán (*i.e.* interferencia constructiva o destructiva). El verdadero error global para un esquema numérico es altamente dependiente del problema que se está estudiando.

Una pregunta que siempre aparece es ¿cómo elegir el τ? Tratemos de responderla. Primero, supongamos que los errores de redondeo son despreciables tal que sólo debemos preocuparnos por los errores de truncamiento. Desde (10.10) y (10.16), el error local de truncamiento en el cálculo de la posición usando el método de Euler es aproximadamente $\tau^2 r'' = \tau^2 a$. Usando sólo estimaciones del orden de magnitud, tomamos $a \approx 10$ [m/s^2], el error en un solo paso en la posición es de 10^{-1} [m], cuando $\tau = 10^{-1}$ [s]. Si el tiempo de vuelo $T \approx 10^0$ [s], entonces el error global es del orden de metros. Si un error de esta magnitud es inaceptable entonces debemos disminuir el paso en el tiempo. Finalmente usando un paso de tiempo 10^{-1} [s] no introduciríamos ningún error significativo de redondeo dada la magnitud de los otros parámetros del problema.

En el mundo real, a menudo no podemos hacer un análisis tan elegante por una variedad de razones (ecuaciones complicadas, problemas con el redondeo, flojera, etc.). Sin embargo, a menudo podemos usar la intuición física. Respóndase usted mismo "¿en qué escala de tiempo el movimiento es casi lineal?". Por ejemplo, para la trayectoria completa de una pelota de *baseball*, que es aproximadamente parabólica, el tiempo en el aire son unos pocos segundos, entonces el movimiento es aproximadamente lineal sobre una escala de tiempo de unos pocos centésimos de segundo. Para revisar nuestra intuición, nosotros podemos comparar los resultados obtenidos usando $\tau = 10^{-1}$ [s] y $\tau = 10^{-2}$ [s] y, si ellos son suficientemente cercanos, suponemos que todo está bien. A veces automatizamos la prueba de varios valores de τ; el programa es entonces llamado *adaptativo* (construiremos un programa de este tipo más adelante). Como con cualquier método numérico, la aplicación ciega de esta técnica es poco recomendada, aunque con sólo un poco de cuidado ésta puede ser usada exitosamente.

10.1.6 Programa de la pelota de *baseball*.

La tabla 10.1 bosqueja un simple programa, llamado `balle`, que usa el método de Euler para calcular la trayectoria de una pelota de *baseball*. Antes de correr el programa, establezcamos algunos valores razonables para tomar como entradas. Una velocidad inicial de $|\vec{v}_1| = 15$ [m/s] nos da una pelota que le han pegado débilmente. Partiendo del origen y despreciando la resistencia del aire, el tiempo de vuelo es de 2.2 [s], y el alcance horizontal es sobre los 23 [m] cuando el ángulo inicial $\theta = 45°$. A continuación, mostramos la salida a pantalla del programa `balle` en C++ cuando es corrido bajo estas condiciones.

- Fijar la posición inicial $\vec{r}_1$ y la velocidad inicial $\vec{v}_1$ de la pelota de *baseball*.
- Fijar los parámetros físicos (m, C_d, etc.).
- Iterar hasta que la bola golpée en el piso o el máximo número de pasos sea completado.
  - Grabar posición (calculada y teórica) para graficar.
  - Calcular la aceleración de la pelota de *baseball*.
  - Calcular la nueva posición y velocidad, $\vec{r}_{n+1}$ y $\vec{v}_{n+1}$, Usando el método de Euler, (10.18) y (10.19).
  - Si la pelota alcanza el suelo ($y < 0$) para la iteración.
- Imprimir el alcance máximo y el tiempo de vuelo.
- Graficar la trayectoria de la pelota de *baseball*.

Tabla 10.1: Bosquejo del programa `balle`, el cual calcula la trayectoria de una pelota de *baseball* usando el método de Euler.

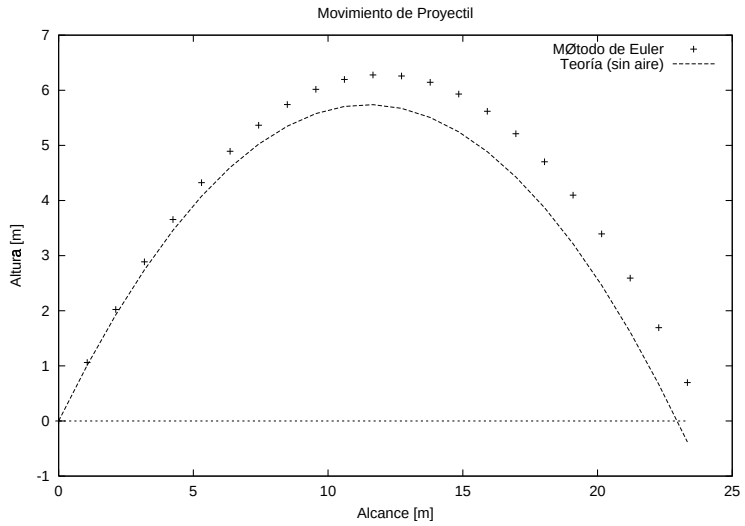


Figura 10.2: Salida del programa `balle` para una altura inicial de 0 [m], una velocidad inicial de 15 [m/s], y un paso de tiempo τ =0.1 [s]. No hay resistencia del aire. La línea continua es la teórica y los puntos son los calculados, la diferencia se debe a errores de truncamiento.

```
jrogan@huelen:~/programas$ balle
Ingrese la altura inicial [m] : 0
Ingrese la velocidad inicial [m/s]: 15
Ingrese angulo inicial (grados): 45
Ingrese el paso en el tiempo, tau en [s]: 0.1
```

```
Tiempo de vuelo: 2.2
Alcance: 24.3952
```

La salida en Octave debiera ser muy similar.

La trayectoria calculada por el programa es mostrada en la figura 10.2. Usando un paso de $\tau = 0.1$ [s], el error en el alcance horizontal es sobre un metro, como esperábamos del error de truncamiento. A velocidades bajas los resultados no son muy diferentes si incluimos la resistencia con el aire, ya que $|\vec{F}_a(\vec{v}_1)|/m \approx g/7$.

Ahora tratemos de batear un cuadrangular. Consideremos una velocidad inicial grande $|\, v_1 \,| = 50$ [m/s]. Debido a la resistencia, encontramos que el alcance es reducido a alrededor de 125 [m], menos de la mitad de su maximo teórico. La trayectoria es mostrada figura 10.3, notemos cómo se aparta de la forma parabólica.

En nuestras ecuaciones para el vuelo de una pelota de *baseball* no hemos incluido todos los factores en el problema. El coeficiente de arrastre no es constante sino más bien una complicada función de la velocidad. Además, la rotación de la pelota ayuda a levantar la pelota (efecto Magnus).

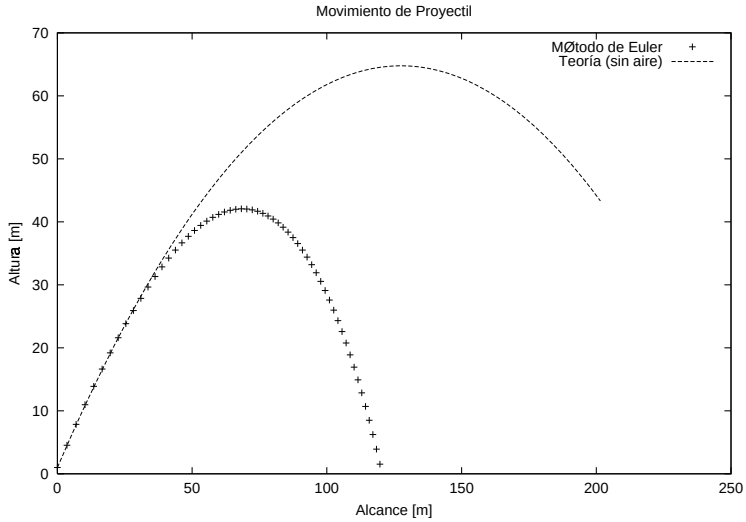


Figura 10.3: Salida del programa `balle` para una altura inicial de 1 [m], una velocidad inicial de 50 [m/s], y un paso de tiempo τ =0.1 [s]. Con resistencia del aire.

10.2 Péndulo simple.

10.2.1 Ecuaciones básicas.

El movimiento de los péndulos ha fascinado a físicos desde que Galileo fue hipnotizado por la lámpara en la Catedral de Pisa. El problema es tratado en los textos de mecánica básica pero antes de apresurarnos a calcular con el computador, revisemos algunos resultados básicos.

Para un péndulo simple es más conveniente describir la posición en términos del desplazamiento angular, $\theta(t)$. La ecuación de movimiento es

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\,\mathrm{sen}\,\theta \ , \tag{10.26}$$

donde L es la longitud del brazo y g es la aceleración de gravedad. En la aproximación para ángulo pequeño, $\mathrm{sen}\,\theta \approx \theta$, la ecuación (10.26) se simplifica a

$$\frac{d^2\theta}{dt^2} = -\frac{g}{L}\theta \ . \tag{10.27}$$

Esta ecuación diferencial ordinaria es fácilmente resuelta para obtener

$$\theta(t) = C_1 \cos\left(\frac{2\pi t}{T_s} + C_2\right) \ , \tag{10.28}$$

donde las constantes C_1 y C_2 están determinadas por los valores iniciales de θ y $\omega = d\theta/dt$. El período para ángulos pequeños, T_s es

$$T_s = 2\pi\sqrt{\frac{L}{g}} \ . \tag{10.29}$$

Esta aproximación es razonablemente buena para oscilaciones con amplitudes menores o iguales a 20°.

Sin la aproximación para ángulos pequeños, la ecuación de movimiento es más difícil de resolver. Sin embargo, sabemos de la experiencia que el movimiento es todavía periódico. En efecto, es posible obtener una expresión para el período sin resolver explícitamente $\theta(t)$. La energía total es

$$E = \frac{1}{2}mL^2\omega^2 - mgL\cos\theta \ , \tag{10.30}$$

donde m es la masa de la lenteja. La energía total es conservada e igual a $E = -mgL\cos\theta_m$, donde θ_m es el ángulo máximo. De lo anterior, tenemos

$$\frac{1}{2}mL^2\omega^2 - mgL\cos\theta = -mgL\cos\theta_m \ , \tag{10.31}$$

o

$$\omega^2 = \frac{2g}{L}\left(\cos\theta - \cos\theta_m\right) \ . \tag{10.32}$$

Ya que $\omega = d\theta/dt$,

$$dt = \frac{d\theta}{\sqrt{\dfrac{2g}{L}\left(\cos\theta - \cos\theta_m\right)}} \ . \tag{10.33}$$

En un período el péndulo se balancea de $\theta = \theta_m$ a $\theta = -\theta_m$ y regresa a $\theta = \theta_m$. Así, en medio período el péndulo se balancea desde $\theta = \theta_m$ a $\theta = -\theta_m$. Por último, por el mismo argumento,

en un cuarto de período el péndulo se balancea desde $\theta = \theta_m$ a $\theta = 0$, así integrando ambos lados de la ecuación (10.33)

$$\frac{T}{4} = \sqrt{\frac{L}{2g}} \int_0^{\theta_m} \frac{d\theta}{\sqrt{(\cos\theta - \cos\theta_m)}} \ . \tag{10.34}$$

Esta integral podría ser reescrita en términos de funciones especiales usando la identidad $\cos 2\theta = 1 - 2\,\mathrm{sen}^2\,\theta$, tal que

$$T = 2\sqrt{\frac{L}{g}} \int_0^{\theta_m} \frac{d\theta}{\sqrt{(\mathrm{sen}^2\,\theta_m/2 - \mathrm{sen}^2\,\theta/2)}} \ . \tag{10.35}$$

Introduciendo $K(x)$, la integral elíptica completa de primera especie,[4]

$$K(x) \equiv \int_0^{\pi/2} \frac{dz}{\sqrt{1 - x^2\,\mathrm{sen}^2\,z}} \ , \tag{10.36}$$

podríamos escribir el período como

$$T = 4\sqrt{\frac{L}{g}} K(\mathrm{sen}\,\theta_m/2) \ , \tag{10.37}$$

usando el cambio de variable $\mathrm{sen}\,z = \mathrm{sen}(\theta/2)/\,\mathrm{sen}(\theta_m/2)$. Para valores pequeños de θ_m, podríamos expandir $K(x)$ para obtener

$$T = 2\pi\sqrt{\frac{L}{g}} \left(1 + \frac{1}{16}\,\theta_m^2 + \ldots\right) \ . \tag{10.38}$$

Note que el primer término es la aproximación para ángulo pequeño (10.29).

10.2.2 Fórmulas para la derivada centrada.

Antes de programar el problema del péndulo miremos un par de otros esquemas para calcular el movimiento de un objeto. El método de Euler está basado en la formulación de la derivada derecha para df/dt dado por (10.7). Una definición equivalente para la derivada es

$$f'(t) = \lim_{\tau\to 0} \frac{f(t+\tau) - f(t-\tau)}{2\tau} \ . \tag{10.39}$$

Esta fórmula se dice centrada en t. Mientras esta fórmula parece muy similar a la ecuación (10.7), hay una gran diferencia cuando τ es finito. Nuevamente, usando la expansión de Taylor,

$$f(t+\tau) = f(t) + \tau f'(t) + \frac{1}{2}\tau^2 f''(t) + \frac{1}{6}\tau^3 f^{(3)}(\zeta_+) \ , \tag{10.40}$$

$$f(t-\tau) = f(t) - \tau f'(t) + \frac{1}{2}\tau^2 f''(t) - \frac{1}{6}\tau^3 f^{(3)}(\zeta_-) \ , \tag{10.41}$$

[4] I.S. Gradshteyn and I.M. Ryzhik, *Table of Integral, Series and Products* (New York: Academic Press, 1965)

donde $f^{(3)}$ es la tercera derivada de $f(t)$ y $\zeta_\pm$ es un valor ente t y $t \pm \tau$. Restando las dos ecuaciones anteriores y reordenando tenemos,

$$f'(t) = \frac{f(t+\tau) - f(t-\tau)}{2\tau} - \frac{1}{6}\tau^2 f^{(3)}(\zeta) \ , \tag{10.42}$$

donde $t - \tau \leq \zeta \leq t + \tau$. Esta es la *aproximación en la primera derivada centrada*. El punto clave es que el error de truncamiento es ahora cuadrático en τ, lo cual es un gran progreso sobre la aproximación de las derivadas adelantadas que tiene un error de truncamiento $\mathcal{O}(\tau)$.

Usando las expansiones de Taylor para $f(t+\tau)$ y $f(t-\tau)$ podemos construir una fórmula centrada para la segunda derivada. La que tiene la forma

$$f''(t) = \frac{f(t+\tau) + f(t-\tau) - 2f(t)}{\tau^2} - \frac{1}{12}\tau^2 f^{(4)}(\zeta) \ , \tag{10.43}$$

donde $t - \tau \leq \zeta \leq t + \tau$. De nuevo, el error de truncamiento es cuadrático en τ. La mejor manera de entender esta fórmula es pensar que la segunda derivada está compuesta de una derivada derecha y de una derivada izquierda, cada una con incrementos de $\tau/2$.

Usted podría pensar que el próximo paso sería preparar fórmulas más complicadas que tengan errores de truncamiento aún más pequeños, quizás usando ambas $f(t \pm \tau)$ y $f(t \pm 2\tau)$. Aunque tales fórmulas existen y son ocasionalmente usadas, las ecuaciones (10.10), (10.42) y (10.43) sirven como el "caballo de trabajo" para calcular las derivadas primera y segunda.

10.2.3 Métodos del "salto de la rana" y de Verlet.

Para el péndulo, las posiciones y velocidades generalizadas son θ y ω, pero para mantener la misma notación anterior trabajaremos con $\vec{r}$ y $\vec{v}$. Comenzaremos de las ecuaciones de movimiento escritas como

$$\frac{d\vec{v}}{dt} = \vec{a}(\vec{r}(t)) \ , \tag{10.44}$$

$$\frac{d\vec{r}}{dt} = \vec{v}(t) \ . \tag{10.45}$$

Note que explícitamente escribimos la aceleración dependiente solamente de la posición. Discretizando la derivada temporal usando la aproximación de derivada centrada da,

$$\frac{\vec{v}(t+\tau) - \vec{v}(t-\tau)}{2\tau} + \mathcal{O}(\tau^2) = \vec{a}(\vec{r}(t)) \ , \tag{10.46}$$

para la ecuación de la velocidad. Note que aunque los valores de velocidad son evaluados en $t + \tau$ y $t - \tau$, la aceleración es evaluada en el tiempo t.

Por razones que pronto serán claras, la discretización de la ecuación de posición estará centrada entre $t + 2\tau$ y t,

$$\frac{\vec{r}(t+2\tau) - \vec{r}(t)}{2\tau} + \mathcal{O}(\tau^2) = \vec{v}(t+\tau) \ . \tag{10.47}$$

De nuevo usamos la notación $f_n \equiv f(t = (n-1)\tau)$, en la cual la ecuación (10.47) y (10.46) son escritas como,

$$\frac{\vec{v}_{n+1} - \vec{v}_{n-1}}{2\tau} + \mathcal{O}(\tau^2) = \vec{a}(\vec{r}_n) \ , \tag{10.48}$$

$$\frac{\vec{r}_{n+2} - \vec{r}_n}{2\tau} + \mathcal{O}(\tau^2) = \vec{v}_{n+1} \ . \tag{10.49}$$

Reordenando los términos para obtener los valores futuros a la izquierda,

$$\vec{v}_{n+1} = \vec{v}_{n-1} + 2\tau\vec{a}(\vec{r}_n) + \mathcal{O}(\tau^3) \ , \tag{10.50}$$

$$\vec{r}_{n+2} = \vec{r}_n + 2\tau\vec{v}_{n+1} + \mathcal{O}(\tau^3) \ , \tag{10.51}$$

el cual es el *método del "salto de la rana" (leap frog)*. Naturalmente, cuando el método es usado en un programa, el término $\mathcal{O}(\tau^3)$ no va y por lo tanto constituye el error de truncamiento para el método.

El nombre "salto de la rana" es usado ya que la solución avanza en pasos de 2τ, con la posición evaluada en valores impares ($\vec{r}_1$, $\vec{r}_3$, $\vec{r}_5$, ...), mientras que la velocidad está calculada en los valores pares ($\vec{v}_2$, $\vec{v}_4$, $\vec{v}_6$, ...). Este entrelazamiento es necesario ya que la aceleración, la cual es una función de la posición, necesita ser evaluada en a tiempo, esto es centrada entre la nueva velocidad y la antigua. Algunas veces el esquema del "salto de la rana" es formulado como

$$\vec{v}_{n+1/2} = \vec{v}_{n-1/2} + \tau\vec{a}(\vec{r}_n) \ , \tag{10.52}$$

$$\vec{r}_{n+1} = \vec{r}_n + \tau\vec{v}_{n+1/2} \ , \tag{10.53}$$

con $\vec{v}_{n\pm1/2} \equiv \vec{v}(t = (n-1\pm1/2)\tau)$. En esta forma, el esquema es funcionalmente equivalente al método de Euler-Cromer.

Para el último esquema numérico de este capítulo tomaremos una aproximación diferente y empezaremos con,

$$\frac{d\vec{r}}{dt} = \vec{v}(t) \ , \tag{10.54}$$

$$\frac{d^2\vec{r}}{dt^2} = \vec{a}(\vec{r}) \ . \tag{10.55}$$

Usando las fórmulas diferenciales centradas para la primera y segunda derivada, tenemos

$$\frac{\vec{r}_{n+1} - \vec{r}_{n-1}}{2\tau} + \mathcal{O}(\tau^2) = \vec{v}_n \ , \tag{10.56}$$

$$\frac{\vec{r}_{n+1} + \vec{r}_{n-1} - 2\vec{r}_n}{\tau^2} + \mathcal{O}(\tau^2) = \vec{a}_n \ , \tag{10.57}$$

donde $\vec{a}_n \equiv \vec{a}(\vec{r}_n)$. Reordenando términos,

$$\vec{v}_n = \frac{\vec{r}_{n+1} - \vec{r}_{n-1}}{2\tau} + \mathcal{O}(\tau^2) \ , \tag{10.58}$$

$$\vec{r}_{n+1} = 2\vec{r}_n - \vec{r}_{n-1} + \tau^2\vec{a}_n + \mathcal{O}(\tau^4) \ . \tag{10.59}$$

Estas ecuaciones, conocidas como el *método de Verlet*[5], podrían parecer extrañas a primera vista, pero ellas son fáciles de usar. Suponga que conocemos $\vec{r}_0$ y $\vec{r}_1$; usando la ecuación (10.59), obtenemos $\vec{r}_2$. Conociendo $\vec{r}_1$ y $\vec{r}_2$ podríamos ahora calcular $\vec{r}_3$, luego usando la ecuación (10.58) obtenemos $\vec{v}_2$, y así sucesivamente.

Los métodos del "salto de la rana" y de Verlet tienen la desventaja que no son "autoiniciados". Usualmente tenemos las condiciones iniciales $\vec{r}_1 = \vec{r}(t=0)$ y $\vec{v}_1 = \vec{v}(t=0)$, pero no $\vec{v}_0 = \vec{v}(t=-\tau)$ [necesitado por el "salto de la rana" en la ecuación (10.50)] o $\vec{r}_0 = \vec{r}(t=-\tau)$ [necesitado por Verlet en la ecuación (10.59)]. Este es el precio que hay que pagar para los esquemas centrados en el tiempo.

Para lograr que estos métodos partan, tenemos una variedad de opciones. El método de Euler-Cromer, usando la ecuación (10.53), toma $\vec{v}_{1/2} = \vec{v}_1$, lo cual es simple pero no muy precisa. Una alternativa es usar otro esquema para lograr que las cosas partan, por ejemplo, en el "salto de la rana" uno podría tomar un paso tipo Euler para atrás, $\vec{v}_0 = \vec{v}_1 - \tau\vec{a}_1$. Algunas precauciones deberían ser tomadas en este primer paso para preservar la precisión del método; usando

$$\vec{r}_0 = \vec{r}_1 - \tau\vec{v}_1 + \frac{\tau^2}{2}\,\vec{a}(\vec{r}_1)\ , \tag{10.60}$$

es una buena manera de comenzar el método de Verlet.

Además de su simplicidad, el método del "salto de la rana" a menudo tiene propiedades favorables (*e.g.* conservación de la energía) cuando resuelve ciertos problemas. El método de Verlet tiene muchas ventajas. Primero, la ecuación de posición tiene un error de truncamiento menor que otros métodos. Segundo, si la fuerza es solamente una función de la posición y si nos preocuparnos sólo de la trayectoria de la partícula y no de su velocidad (como en muchos problemas de mecánica celeste), podemos saltarnos completamente el cálculo de velocidad. El método es popular para el cálculo de las trayectorias en sistemas con muchas partículas, por ejemplo, el estudio de fluidos a nivel microscópico.

10.2.4 Programa de péndulo simple.

Las ecuaciones de movimiento para un péndulo simple son

$$\frac{d\omega}{dt} = \alpha(\theta) \quad \frac{d\theta}{dt} = \omega\ , \tag{10.61}$$

donde la aceleración angular $\alpha(\theta) = -g\,\mathrm{sen}\,\theta/L$. El método de Euler para resolver estas ecuaciones diferenciales ordinarias es iterar las ecuaciones:

$$\theta_{n+1} = \theta_n + \tau\omega_n\ , \tag{10.62}$$

$$\omega_{n+1} = \omega_n + \tau\alpha_n\ . \tag{10.63}$$

Si estamos interesados solamente en el ángulo y no la velocidad, el método de Verlet sólo usa la ecuación

$$\theta_{n+1} = 2\theta_n - \theta_{n-1} + \tau^2\alpha_n\ . \tag{10.64}$$

[5]L.Verlet, "Computer experiments on classical fluid I. Thermodynamical properties of Lennard-Jones molecules", *Phys. Rev.* **159**, 98-103 (1967).

- Seleccionar el método a usar: Euler o Verlet.
- Fijar la posición inicial θ_1 y la velocidad $\omega_1 = 0$ del péndulo.
- Fijar los parámetros físicos y otras variables.
- Tomar un paso para atrás para partir Verlet; ver ecuación (10.60).
- Iterar sobre el número deseado de pasos con el paso de tiempo y método numérico dado.
  - Grabar ángulo y tiempo para graficar.
  - Calcular la nueva posición y velocidad usando el método de Euler o de Verlet.
  - Comprobar si el péndulo a pasado a través de $\theta = 0$; Si es así usar el tiempo transcurrido para estimar el período.
- Estima el período de oscilación, incluyendo barra de error.
- Graficar las oscilaciones como θ versus t.

Tabla 10.2: Bosquejo del programa `pendulo`, el cual calcula el tiempo de evolución de un péndulo simple usando el método de Euler o Verlet.

En vez de usar las unidades SI, usaremos las unidades adimensionales naturales del problema. Hay solamente dos parámetros en el problema, g y L y ellos siempre aparecen en la razón g/L. Fijando esta razón a la unidad, el período para pequeñas amplitudes $T_s = 2\pi$. En otras palabras, necesitamos solamente una unidad en el problema: una escala de tiempo. Ajustamos nuestra unidad de tiempo tal que el período de pequeñas amplitudes sea 2π.

La tabla 10.2 presenta un bosquejo del programa `pendulo`, el cual calcula el movimiento de un péndulo simple usando o el método de Euler o el de Verlet. El programa estima el período a registrar cuando el ángulo cambia de signo; esto es verificar si θ_n y θ_{n+1} tienen signos opuestos probando si $\theta_n * \theta_{n+1} < 0$. Cada cambio de signo da una estimación para el período, $\tilde{T}_k = 2\tau(n_{k+1} - n_k)$, donde n_k es el paso de tiempo en el cual el k-ésimo cambio de signo ocurre. El período estimado de cada inversión de signo es registrado, y el valor medio calculado como

$$\left\langle \tilde{T} \right\rangle = \frac{1}{M} \sum_{k=1}^{M} \tilde{T}_k \ , \tag{10.65}$$

donde M es el número de veces que $\tilde{T}$ es evaluado. La barra de error para esta medición del período es estimada como $\sigma = s/M$, donde

$$s = \sqrt{\frac{1}{M-1} \sum_{k=1}^{M} \left(\tilde{T}_k - \left\langle \tilde{T} \right\rangle \right)^2} \ , \tag{10.66}$$

es la desviación estándar de la muestra $\tilde{T}$. Note que cuando el número de medidas se incrementa, la desviación estándar de la muestra tiende a una constante, mientras que la barra de error estimado decrece.

Para comprobar el programa `pendulo`, primero tratamos con ángulos iniciales pequeños, θ_m, ya que conocemos el período $T \approx 2\pi$. Tomando $\tau = 0.1$ tendremos sobre 60 puntos por oscilación; tomando 300 pasos deberíamos tener como cinco oscilaciones. Para $\theta_m = 10°$, El método de Euler calcula un período estimado de $\langle \tilde{T} \rangle = 6.375 \pm 0.025$ sobre un 1.5% mayor que el esperado $T = 2\pi(1.002)$ dado por la ecuación (10.38). Nuestro error estimado para el período es del orden de $\pm\tau$ en cada medida. Cinco oscilaciones son 9 medidas de $\tilde{T}$, así que nuestro error estimado para el período debería ser $(\tau/2)/\sqrt{9} \approx 0.02$. Notemos que la estimación está en buen acuerdo con los resultados obtenidos usando la desviación estándar. Hasta aquí todo parece razonable.

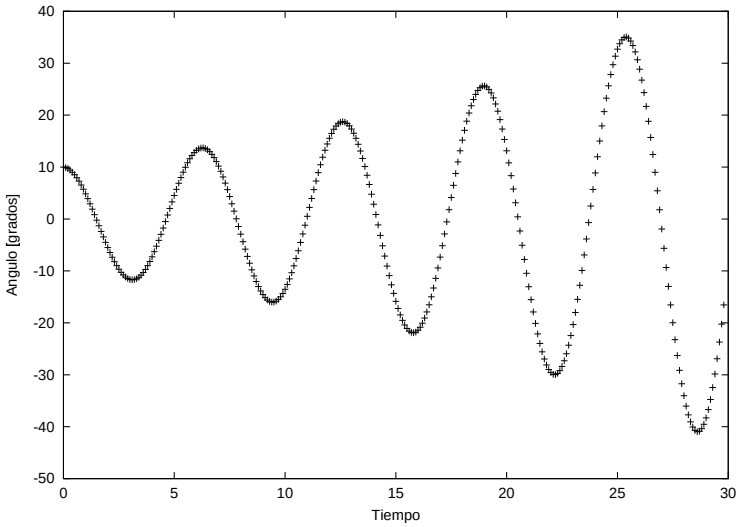


Figura 10.4: Salida del programa `pendulo` usando el método de Euler. El ángulo inicial es $\theta_m = 10°$, el paso en el tiempo es $\tau = 0.1$, y 300 iteraciones fueron calculadas.

Infortunadamente si miramos el gráfico 10.4 nos muestra los problemas del método de Euler: La amplitud de oscilación crece con el tiempo. Ya que la energía es proporcional al ángulo máximo, esto significa que la energía total se incrementa en el tiempo. El error global de truncamiento en el método de Euler se acumula en este caso. Para pasos de tiempos pequeños $\tau = 0.05$ e incrementos en el número de pasos (600) podemos mejorar los resultados, ver figura 10.5, pero no eliminamos el error. El método del punto medio tiene la misma inestabilidad numérica.

Usando el método de Verlet con $\theta_m = 10°$, el paso en el tiempo $\tau = 0.1$ y 300 iteraciones obtenemos los resultados graficados en 10.6. Estos resultados son mucho mejores; la amplitud de oscilación se mantiene cerca de los 10° y $\langle \tilde{T} \rangle = 6.275 \pm 0.037$. Afortunadamente el método de Verlet, el del "salto de rana" y el de Euler-Cromer no sufren de la inestabilidad numérica encontrada al usar el método de Euler.

Para $\theta_m = 90°$, la primera corrección para la aproximación de ángulo pequeño, ecuación (10.38), da $T = 7.252$. Usando el método de Verlet, el programa da un período estimado de

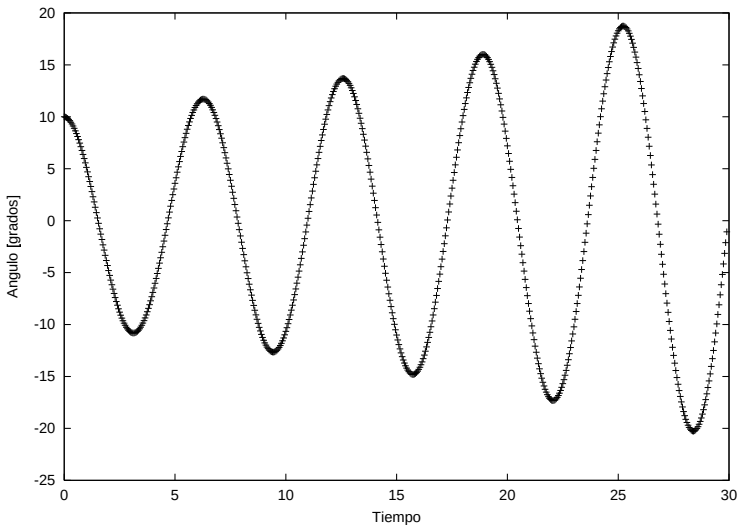


Figura 10.5: Salida del programa `pendulo` usando el método de Euler. El ángulo inicial es $\theta_m = 10°$, el paso en el tiempo es $\tau = 0.05$ y 600 iteraciones fueron calculadas.

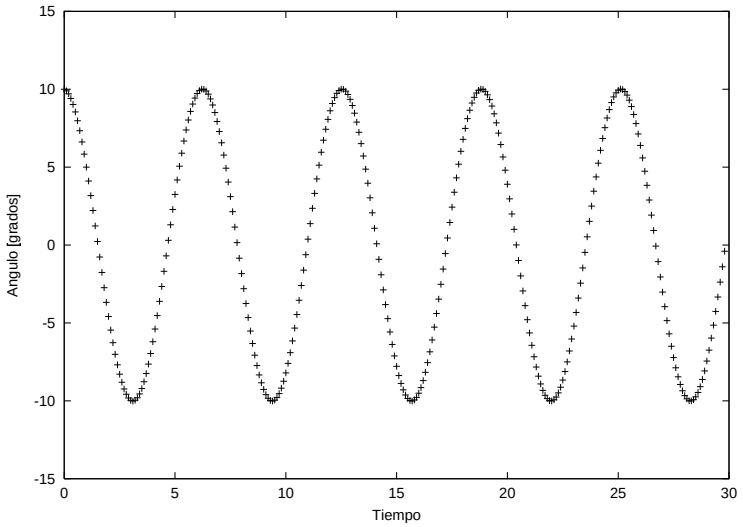


Figura 10.6: Salida del programa `pendulo` usando el método de Verlet. El ángulo inicial es $\theta_m = 10°$, el paso en el tiempo es $\tau = 0.1$ y 300 iteraciones fueron calculadas.

$\langle \tilde{T} \rangle = 7.414 \pm 0.014$, lo cual indica que (10.38) es una buena aproximación (alrededor de un 2% de error), aún para ángulos grandes. Para el ángulo muy grande de $\theta_m = 170°$, vemos la trayectoria en la figura 10.7. Notemos cómo la curva tiende a aplanarse en los puntos de retorno. En este caso el período estimado es $\langle \tilde{T} \rangle = 15.3333 \pm 0.0667$, mientras que (10.38) da $T = 9.740$, indicando que esta aproximación para (10.37) deja de ser válida para este ángulo tan grande.

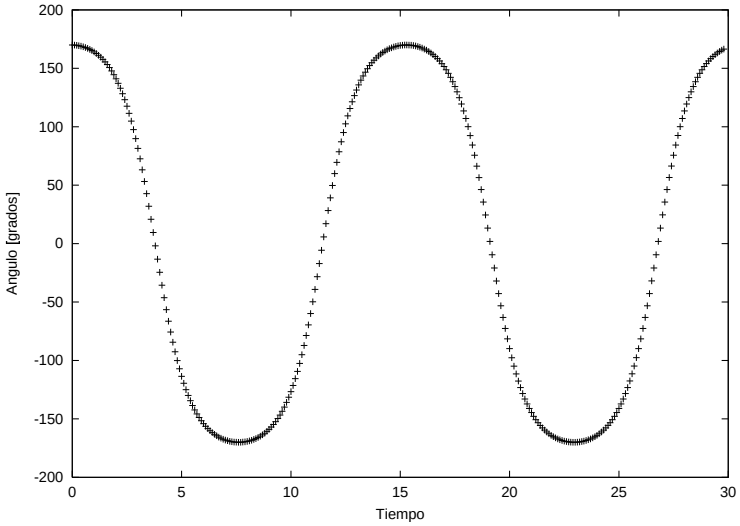


Figura 10.7: Salida del programa `pendulo` usando el método de Verlet. El ángulo inicial es $\theta_m = 170°$, el paso en el tiempo es $\tau = 0.1$ y 300 iteraciones fueron calculadas.

10.3 Listado de los programas.

10.3.1 balle.cc

```
#include "NumMeth.h"

main()
{
  const double Cd=0.35;
  const double rho=1.293;                // [kg/m^3]
  const double radio=0.037;                      // [m]
  double A= M_PI*radio*radio ;
  double m=0.145;                                // [kg]
  double g=9.8;                  // [m/s^2]
  double a = -Cd*rho*A/(2.0e0*m) ;

  double v0, theta0, tau;
  ofstream salida ("salida.txt") ;
  ofstream salidaT ("salidaT.txt") ;

  double x0, y0;
  x0=0.0e0 ;
  cout << "Ingrese la altura inicial [m] : ";
  cin >> y0;
  cout << "Ingrese la velocidad inicial [m/s]: ";
  cin >> v0;
  cout <<"Ingrese angulo inicial (grados): ";
```

```
  cin >> theta0;

  int flagRA = 2 ;
  while (flagRA!=0 && flagRA !=1) {
    cout <<"Con resistencia del aire, Si= 1, No= 0: ";
    cin >> flagRA;
  }
  cout <<"Ingrese el paso en el tiempo, tau en [s]: ";
  cin >> tau ;
  double vxn=v0*cos(M_PI*theta0/180.0) ;
  double vyn=v0*sin(M_PI*theta0/180.0) ;
  double xn=x0 ;
  double yn=y0 ;
  double tiempo = -tau;
  while( yn >= y0) {
    tiempo +=tau ;
    salidaT << x0+v0*cos(M_PI*theta0/180.0) *tiempo <<" " ;
    salidaT << y0+v0*sin(M_PI*theta0/180.0) *tiempo -g*tiempo*tiempo/2.0e0<< endl;
    salida << xn << " " << yn << endl;
    if(flagRA==0) a=0.0e0 ;
    double v=sqrt(vxn*vxn+vyn*vyn) ;
    double axn= a*v*vxn ;
    double ayn= a*v*vyn -g ;
    double xnp1 = xn + tau*vxn ;
    double ynp1 = yn + tau*vyn ;
    double vxnp1 = vxn + tau*axn;
    double vynp1 = vyn + tau*ayn;
    vxn=vxnp1;
    vyn=vynp1;
    xn=xnp1 ;
    yn=ynp1 ;
  }
  cout << "Tiempo de vuelo: " << tiempo<< endl;
  cout << "Alcance: " << xn<<endl;
  salida.close();
}
```

10.3.2 pendulo.cc

```
#include "NumMeth.h"

main()
{
  int respuesta=2 ;
  while(respuesta != 0 && respuesta !=1 ) {
```

```
  cout << "Eliga el metodo: Euler=0 y Verlet=1: " ;
  cin >> respuesta ;
}
double theta1 ;
double omega1 = 0.0e0;
cout << "Ingrese el angulo inicial (grados): ";
cin >> theta1 ;
theta1*=M_PI/180.0e0 ;
double tau ;
cout << "Ingrese el paso de tiempo: ";
cin >> tau ;
int pasos ;
cout << "Ingrese el numero de pasos: ";
cin >> pasos ;

double * periodo = new double[pasos] ;

ofstream salida ("salidaPendulo.txt");

double theta0= theta1-tau*omega1-tau*tau*sin(theta1) ;

double thetaNm1=theta0 ;
double thetaN=theta1 ;
double omegaN=omega1;

double thetaNp1, omegaNp1 ;

int nK=1;
int M=0 ;

for(int i=1; i< pasos; i++) {
  double alphaN=-sin(thetaN);
  if(respuesta==0) {          // Euler
    thetaNp1=thetaN+tau*omegaN ;
    omegaNp1=omegaN+tau*alphaN ;
  } else {
    thetaNp1=2.0e0*thetaN-thetaNm1+tau*tau*alphaN ;
  }
  salida << (i-1)*tau<<" " <<thetaNp1*180/M_PI<< endl ;
  if (thetaNp1*thetaN<0) {
    if(M==0) {
      periodo[M++]=0.0e0;
      nK=i ;
    } else {
      periodo[M++] = 2.0e0*tau*double(i-nK);
```

```
        nK=i ;
      }
    }

    thetaNm1=thetaN ;
    thetaN=thetaNp1 ;
    omegaN=omegaNp1 ;
  }
  double Tprom=0.0e0;
  for (int i=1; i < M; i++) Tprom+=periodo[i] ;
  Tprom/=double(M-1) ;
  double ssr=0.0 ;
  for (int i=1; i < M; i++) ssr+=(periodo[i]-Tprom)* (periodo[i]-Tprom);
  ssr/=double(M-2);
  double sigma =sqrt(ssr/double(M-1)) ;
  cout <<" Periodo = "<< Tprom << "+/-"<< sigma << endl ;
  salida.close() ;
  delete [] periodo;
}
```

Capítulo 11

Ecuaciones Diferenciales Ordinarias II: Métodos Avanzados.

versión final 2.2-19 agosto 2002[1]

En el capítulo anterior aprendimos cómo resolver ecuaciones diferenciales ordinarias usando algunos métodos simples. En este capítulo haremos algo de mecánica celeste básica comenzando con el problema de Kepler. Al calcular la órbita de un satélite pequeño alrededor de un cuerpo masivo (e.g. un cometa orbitando el Sol), descubriremos que métodos mucho más sofisticados son necesarios para manipular sistemas simples de dos cuerpos.

11.1 Órbitas de cometas.

11.1.1 Ecuaciones básicas.

Considere el problema de Kepler en el cual un pequeño satélite, tal como un cometa, orbita el Sol. Usamos un sistema de coordenadas Copernicano y fijamos el Sol en el origen. Por ahora, consideremos solamente la fuerza gravitacional entre el cometa y el Sol, y despreciemos todas las otras fuerzas (e.g., fuerzas debidas a los planetas, viento solar). La fuerza sobre el cometa es

$$\vec{F} = -\frac{GmM}{|\vec{r}|^3}\vec{r}\,, \tag{11.1}$$

donde $\vec{r}$ es la posición del cometa, m es su masa, $M = 1.99 \times 10^{30}$ [kg] es la masa del Sol, y $G = 6.67 \times 10^{-11}$ [m^3/kg s^2] es la constante gravitacional.

Las unidades naturales de longitud y tiempo para este problema no son metros ni segundos. Como unidad de distancia usaremos la unidad astronómica [AU], 1 AU=1.496×10^{11} [m], la cual es igual a la distancia media de la Tierra al Sol. La unidad de tiempo será el [año] AU (el período de una órbita circular de radio 1 [AU]). En estas unidades, el producto $GM = 4\pi^2$ [AU^3/año^2]. Tomaremos la masa del cometa, m, como la unidad; en unidades MKS la masa típica de un cometa es $10^{15\pm3}$ [kg].

Ahora tenemos suficiente para ensamblar nuestro programa, pero antes hagamos una rápida revisión de lo que sabemos de órbitas. Para un tratamiento completo podemos recurrir

[1]Este capítulo está basado en el tercer capítulo del libro: *Numerical Methods for Physics, second edition* de Alejandro L. Garcia, editorial Prentice Hall.

a algunos textos de mecánica estandard, tales como Symon[2] o Landau y Lifshitz[3]. La energía total del satélite es

$$E = \frac{1}{2}mv^2 - \frac{GMm}{r} \ , \tag{11.2}$$

donde $r = |\vec{r}|$ y $v = |\vec{v}|$. Esta energía total es conservada, tal como el momento angular,

$$\vec{L} = \vec{r} \times (m\vec{v}) \ . \tag{11.3}$$

Ya que este problema es bidimensional, consideraremos el movimiento en el plano x-y. El único componente distinto de cero del momento angular está en la dirección z.

Cuando la órbita es circular, la fuerza centrípeta es compensada por la fuerza gravitacional,

$$\frac{mv^2}{r} = \frac{GMm}{r^2} \ , \tag{11.4}$$

o

$$v = \sqrt{\frac{GM}{r}} \ . \tag{11.5}$$

Por colocar algunos valores, en una órbita circular en r=1 [AU] la velocidad orbital es v= 2π [AU/año] (cerca de 30.000 [km/h]). Reemplazando la ecuación (11.5) en (11.2), la energía total en una órbita circular es

$$E = -\frac{GMm}{2r} \ . \tag{11.6}$$

En una órbita elíptica, los semiejes mayores y menores, a y b, son desiguales (Figura 11.1).

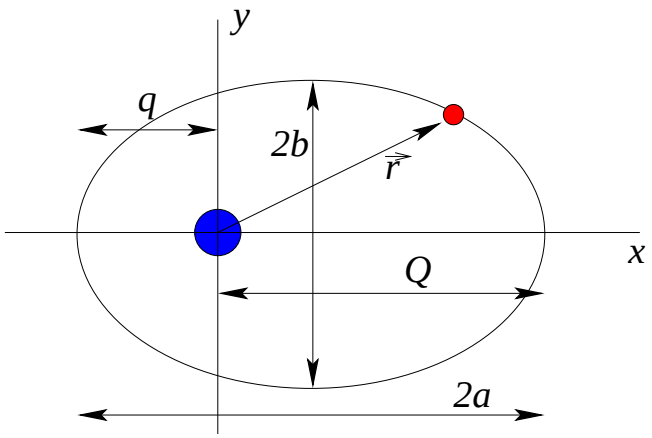


Figura 11.1: Órbita elíptica alrededor del Sol.

[2]K. Symon, *Mechanics* (Reading Mass.: Addison-Wesley, 1971).
[3]L. Landau and E. Lifshitz, *Mechanics* (Oxford: Pergamon, 1976).

Nombre del Cometa	T [años]	e	q [AU]	i	Primera pasada
Encke	3.30	0.847	0.339	12.4°	1786
Biela	6.62	0.756	0.861	12.6°	1772
Schwassmann-Wachmann 1	16.10	0.132	5.540	9.5°	1925
Halley	76.03	0.967	0.587	162.2°	239 A.C.
Grigg-Mellish	164.3	0.969	0.923	109.8°	1742
Hale-Bopp	2508.0	0.995	0.913	89.4°	1995

Tabla 11.1: Datos orbitales de algunos cometas.

La excentricidad, e, está definida como

$$e = \sqrt{1 - \frac{b^2}{a^2}} \ . \tag{11.7}$$

La excentricidad de la Tierra es $e = 0.017$, por lo tanto esta órbita está muy cercana de ser circular. La distancia del Sol al perihelio (punto de mayor aproximación) es $q = (1 - e)a$; la distancia del Sol al afelio es $Q = (1 + e)a$.

La ecuación (11.6) también se mantiene para una órbita elíptica si reemplazamos el radio con el semieje mayor; por lo tanto la energía total es

$$E = -\frac{GMm}{2a} \ . \tag{11.8}$$

Note que $E \leq 0$. De las ecuaciones (11.2) y (11.8), encontramos que la velocidad orbital como función de la distancia radial es

$$v = \sqrt{GM\left(\frac{2}{r} - \frac{1}{a}\right)} \ . \tag{11.9}$$

La velocidad es máxima en el perihelio y mínima en el afelio, la razón entre las velocidades está dada por Q/q. Finalmente, usando la conservación de momento angular, podríamos derivar la tercera ley de Kepler,

$$T^2 = \frac{4\pi^2}{GM}a^3 \ , \tag{11.10}$$

donde T es el período de la órbita.

Los datos orbitales para unos pocos cometas bien conocidos están dados en la tabla 11.1. La inclinación, i, es el ángulo entre el plano orbital del cometa y el plano eclíptico (el plano de la órbita de los planetas). Cuando la inclinación es menor que los 90°, se dice que la órbita es directa, cuando es mayor que 90°, se dice que la órbita es retrógrada (*i.e.*, orbita el Sol en la dirección opuesta a la de los planetas).

11.1.2 Programa `orbita`.

Un programa simple, llamado `orbita`, que calcula las órbitas para el problema de Kepler usando varios métodos numéricos es propuesto en la tabla 11.2. El método de Euler, descrito

- Fijar la posición y velocidad inicial del cometa.
- Fijar los parámetros físicos (m, G, etc.).
- Iterar sobre el número deseado de pasos usando el método numérico especificado.
  - Grabar posición y la energía para graficar.
  - Calcular la nueva posición y velocidad usando:
    * Método de Euler (11.11), (11.12) o;
    * Método de Euler-Cromer (11.13), (11.14) o;
    * Método Runge-Kutta de cuarto orden (11.30), (11.31) o;
    * Método de Runge-Kutta adaptativo.
- Graficar la trayectoria del cometa.
- Graficar la energía del cometa versus el tiempo.

Tabla 11.2: Bosquejo del programa `orbita`, el cual calcula la trayectoria de un cometa usando varios métodos numéricos.

en el capítulo anterior, calcula la trayectoria del cometa como

$$\vec{r}_{n+1} = \vec{r}_n + \tau \vec{v}_n \ , \tag{11.11}$$
$$\vec{v}_{n+1} = \vec{v}_n + \tau \vec{a}(\vec{r}_n) \ , \tag{11.12}$$

donde $\vec{a}$ es la aceleración gravitacional. De nuevo, discretizamos el tiempo y usamos la notación $f_n \equiv f(t = (n-1)\tau)$, donde τ es el paso tiempo.

El caso de prueba más simple es una órbita circular. Para un radio orbital de 1 [AU], la ecuación (11.5) da una velocidad tangencial de 2π [AU/año]. Unos 50 puntos por revolución orbital nos daría una suave curva, tal que $\tau = 0.02$ [años] (o cercano a una semana) es un paso de tiempo razonable. Con esos valores, el programa órbita usando el método de Euler, da los resultados mostrados en la figura 11.2. Inmediatamente vemos que la órbita no es circular, sino una espiral hacia fuera. La razón es clara observando el gráfico de energía; en vez de ser constante, la energía total aumenta continuamente. Este tipo de inestabilidad se observa, también, en el método de Euler para el péndulo simple. Afortunadamente hay una solución simple a este problema: el método Euler-Cromer para calcular la trayectoria,

$$\vec{v}_{n+1} = \vec{v}_n + \tau \vec{a}(\vec{r}_n) \ , \tag{11.13}$$
$$\vec{r}_{n+1} = \vec{r}_n + \tau \vec{v}_{n+1} \ . \tag{11.14}$$

Note que el único cambio respecto al método de Euler es que primero calculamos la nueva velocidad, $\vec{v}_{n+1}$, y luego la usamos en el cálculo de la nueva posición. Para las mismas condiciones iniciales y paso de tiempo, el método de Euler-Cromer da resultados mucho mejores, como los mostrados en la figura 11.3. La órbita es casi circular, y la energía total

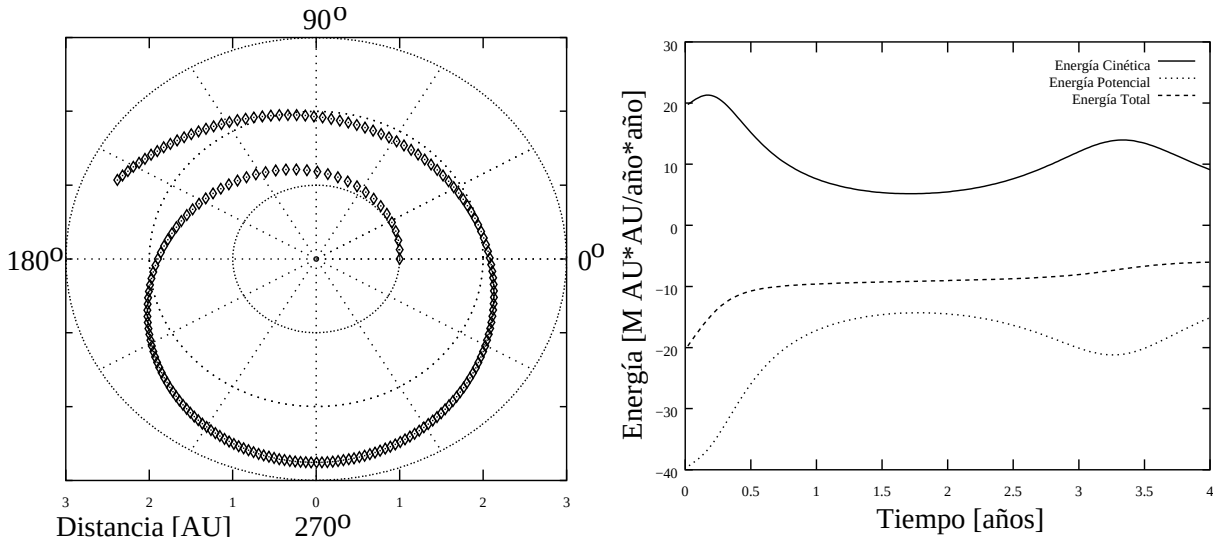


Figura 11.2: Gráfico de la trayectoria y la energía con el programa `orbita` usando el método de Euler. La distancia radial inicial es 1 [AU] y la velocidad tangencial inicial es 2π [AU/año]. El paso en el tiempo es $\tau = 0.02$ [años]; y 200 pasos son calculados. Los resultados están en desacuerdo con la predicción teórica de una órbita circular con energía total constante.

se conserva. Las energías potencial y cinética no son constantes, pero este problema podría ser mejorado usando un paso de tiempo pequeño. El programa órbita también da la opción de usar el método de Runge-Kutta, los cuales son descritos en las próximas dos secciones.

Aunque el método de Euler-Cromer hace un buen trabajo para bajas excentricidades, tiene problemas con órbitas más elípticas, como se muestra en la 11.4. Note que si la energía llega a ser positiva; el satélite alcanza la velocidad de escape. Si bajamos el paso de tiempo desde $\tau = 0.02$ [años] a $\tau = 0.005$ [años] obtenemos mejores resultados, como los mostrados en la figura 11.5. Estos resultados no son del todo perfectos; la órbita puede ser una elipse cerrada, pero todavía tiene una notable deriva espúrea.

En este punto usted se podría estar preguntando, "¿Por qué estamos estudiando este problema, si la solución analítica es bien conocida?". Es verdad que hay problemas mecánicos celestes más interesantes (*e.g.*, el efecto de perturbaciones sobre la órbita, problema de tres cuerpos). Sin embargo, antes de hacer los casos complicados podríamos, siempre, chequear los algoritmos con problemas conocidos. Suponga que introducimos una pequeña fuerza de arrastre sobre el cometa. Podríamos pecar de inocentes creyendo que la precesión de la figura 11.5 fue un fenomeno físico más que un artefacto numérico.

Claramente, el método de Euler-Cromer hace un trabajo inaceptable de rastreo de las órbitas más elípticas. Los resultados mejoran si achicamos el paso de tiempo, pero entonces sólo podemos rastrear unas pocas órbitas. Suponga que deseamos rastrear cometas para posibles impactos con la Tierra. Un gran cometa impactando sobre la Tierra sería más destructivo que una guerra nuclear. Muchos cometas tienen órbitas extremadamente elípticas y períodos de cientos de años. Esta amenaza desde el espacio exterior motiva nuestro estudio de métodos más avanzados para resolver ecuaciones diferenciales ordinarias.

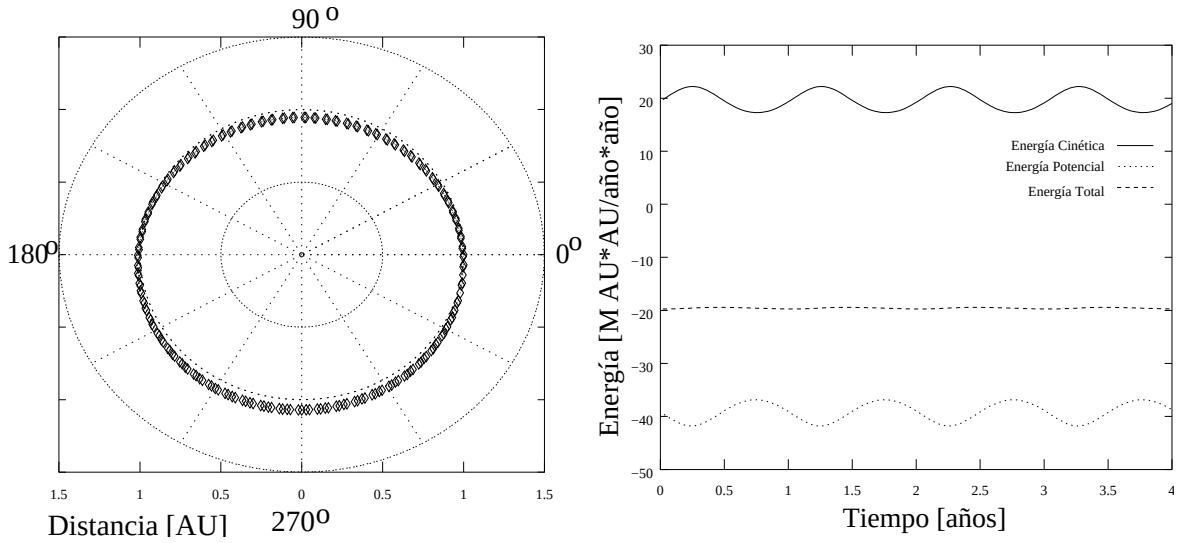


Figura 11.3: Gráfico de la trayectoria y la energía con el programa `orbita` usando el método de Euler-Cromer. Los parámetros son los mismos que en la figura 11.2. Los resultados están en un acuerdo cualitativo al menos con la predicción teórica de una órbita circular con energía total constante.

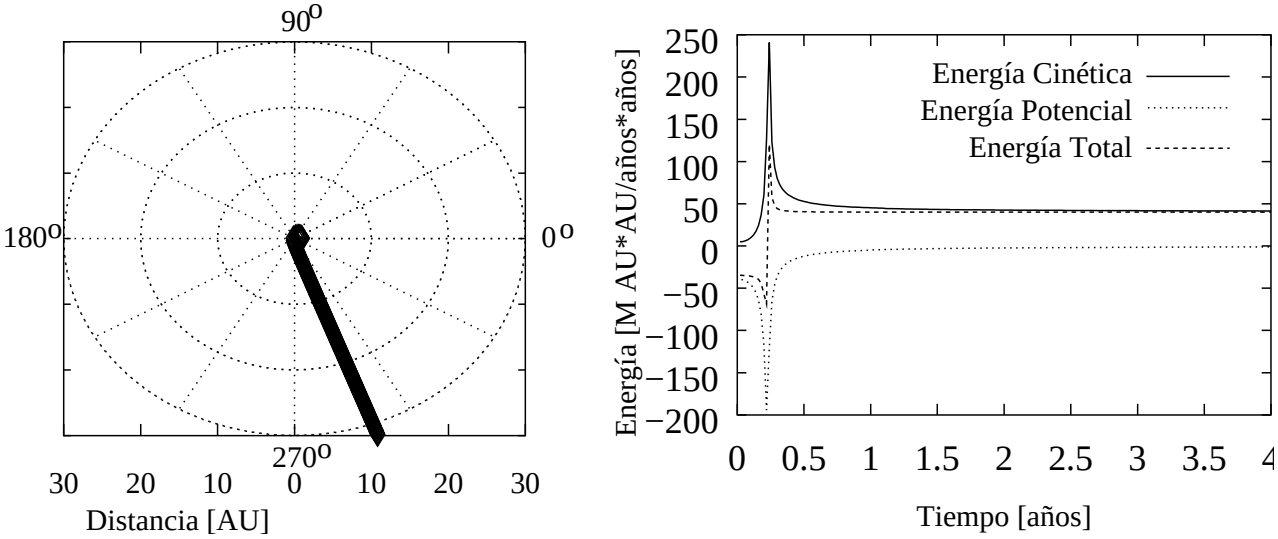


Figura 11.4: Gráfico de la trayectoria y la energía con el programa `orbita` usando el método de Euler-Cromer. La distancia radial inicial es 1 [AU] y la velocidad tangencial inicial es π [AU/año]. El paso en el tiempo es $\tau = 0.02$ [años]; y 200 pasos son calculados. Debido al error numérico el cometa alcanza la velocidad de escape, la posición final es 35 [AU] y la energía total es positiva.

11.2 Métodos de Runge-Kutta.

11.2.1 Runge-Kutta de segundo orden.

Ahora miremos uno de los métodos más populares para resolver numéricamente las ecuaciones diferenciales ordinarias: Runge-Kutta. Primero trabajaremos las fórmulas generales de

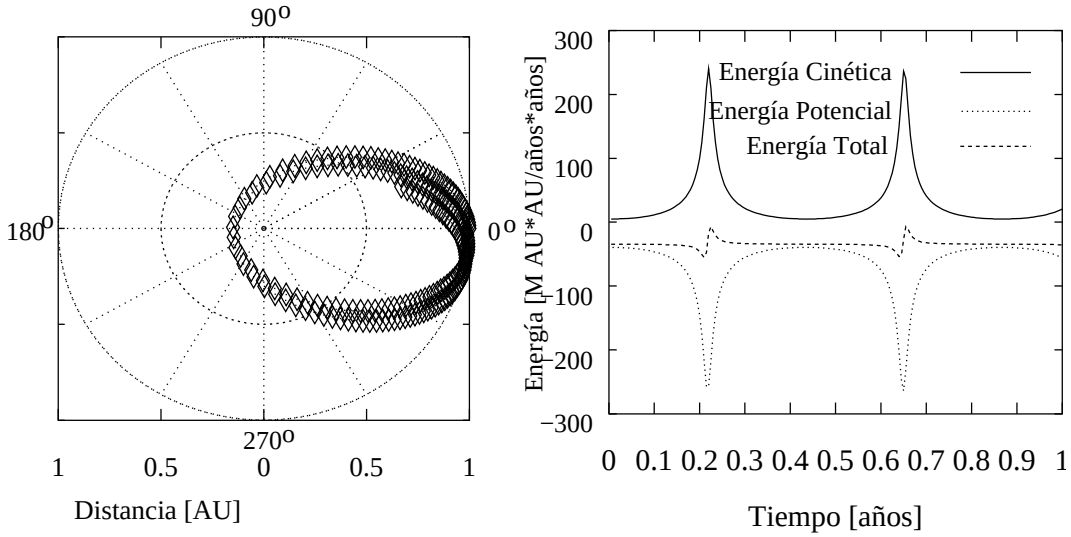


Figura 11.5: Gráfico de la trayectoria y la energía con el programa `orbita` usando el método de Euler-Cromer. Los parámetros son los mismos que en la figura 11.4 excepto que el tiempo es más pequeño $\tau = 0.005$ [años]. Los resultados son mejores, pero aún presenta una precesión espúrea.

Runge-Kutta y luego las aplicaremos específicamente a nuestro problema del cometa. De esta manera será fácil usar el método Runge-Kutta para otros sistemas físicos. Nuestra ecuación diferencial ordinaria gerneral toma la forma

$$\frac{d\vec{x}}{dt} = \vec{f}(\vec{x}(t), t) \ , \tag{11.15}$$

donde el vector de estado $x(t) = [x_1(t), x_2(t), \ldots, x_N(t)]$ es la solución deseada. En el problema de Kepler tenemos

$$\vec{x}(t) = [r_x(t)\ r_y(t)\ v_x(t)\ v_y(t)] \ , \tag{11.16}$$

y

$$\begin{aligned} \vec{f}(\vec{x}(t), t) &= \left[\frac{dr_x}{dt}\ \frac{dr_y}{dt}\ \frac{dv_x}{dt}\ \frac{v_y}{dt}\right] \ , \\ &= [v_x(t)\ v_y(t)\ F_x(t)/m\ F_y(t)/m] \ , \end{aligned} \tag{11.17}$$

donde r_x, v_x, y F_x son las componentes x de la posición, la velocidad y la fuerza respectivamente (y lo mismo para la componente y). Note que en el problema de Kepler, la función f no depende explícitamente del tiempo sino que sólo depende de $\vec{x}(t)$.

Nuestro punto de partida es el método simple de Euler; en forma vectorial podría ser escrito como

$$\vec{x}(t + \tau) = \vec{x}(t) + \tau \vec{f}(\vec{x}, t) \ . \tag{11.18}$$

Consideremos la primera fórmula de Runge-Kutta:

$$\vec{x}(t+\tau) = \vec{x}(t) + \tau \vec{f}\left(\vec{x}^{\,*}\left(t+\frac{1}{2}\tau\right), t+\frac{1}{2}\tau\right) \ , \tag{11.19}$$

donde

$$\vec{x}^{\,*}\left(t+\frac{1}{2}\tau\right) \equiv \vec{x}(t) + \frac{1}{2}\tau \vec{f}(\vec{x},t) \ . \tag{11.20}$$

Para ver de dónde viene esta fórmula, consideremos por el momento el caso de una variable. Sabemos que la expansión de Taylor

$$\begin{aligned} x(t+\tau) &= x(t) + \tau \frac{dx(\zeta)}{dt} \ , \\ &= x(t) + \tau f(x(\zeta),\zeta) \ , \end{aligned} \tag{11.21}$$

es exacta para algún valor de ζ entre t y $t+\tau$, como se vio en la ecuación (10.10). La fórmula de Euler toma $\zeta = t$; Euler-Cromer usa $\zeta = t$ en la ecuación de velocidad y $\zeta = t+\tau$ en la ecuación de posición. Runge-Kutta usa $\zeta = t+\frac{1}{2}\tau$, lo cual pareciera una mejor estimación. Sin embargo, $x\left(t+\frac{1}{2}\tau\right)$ no es conocida, podemos aproximarla de la manera simple: usamos un paso de Euler para calcular $x^*\left(t+\frac{1}{2}\tau\right)$ y usamos ésta como nuestra estimación de $x\left(t+\frac{1}{2}\tau\right)$.

Avancemos a un ejemplo simple usando la fórmula Runge-Kutta. Consideremos la ecuación

$$\frac{dx}{dt} = -x \ , \quad x(t=0) = 1 \ . \tag{11.22}$$

La solución de la ecuación (11.22) es $x(t) = e^{-t}$. Usando el método de Euler con un paso de tiempo de $\tau = 0.1$, tenemos

$$\begin{aligned} x(0.1) &= 1 + 0.1(-1) = 0.9 \ , \\ x(0.2) &= 0.9 + (0.1)(-0.9) = 0.81 \ , \\ x(0.3) &= 0.81 + 0.1(-0.81) = 0.729 \ , \\ x(0.4) &= 0.729 + 0.1(-0.729) = 0.6561 \ . \end{aligned}$$

Ahora tratemos con Runge-Kutta. Para hacer una correcta comparación usaremos un paso de tiempo mayor para Runge-Kutta $\tau = 0.2$ porque hace el doble de evaluaciones de $f(x)$. Por la fórmula de Runge-Kutta presentada arriba,

$$\begin{aligned} x^*(0.1) &= 1 + 0.1(-1) = 0.9 \ , \\ x(0.2) &= 1 + 0.2(-0.9) = 0.82 \ , \\ x^*(0.3) &= 0.82 + 0.1(-0.82) = 0.738 \\ x(0.4) &= 0.82 + 0.2(-0.738) = 0.6724 \ . \end{aligned}$$

Podemos comparar esto con la solución exacta $x(0.4) = \exp(-0.4) \approx 0.6703$. Claramente, Runge-Kutta lo hace mucho mejor que Euler; los errores porcentuales absolutos son 0.3% y 2.1%, respectivamente.

11.2.2 Fórmulas generales de Runge-Kutta.

La fórmula discutida arriba no es la única formula posible para un Runge-Kutta de segundo orden. Aquí hay una alternativa:

$$\vec{x}(t+\tau) = \vec{x}(t)\frac{1}{2}\tau[\vec{f}(\vec{x}(t),t) + \vec{f}(\vec{x}^{\,*}(t+\tau),t+\tau)] \ , \tag{11.23}$$

donde

$$\vec{x}^{\,*}(t+\tau) \equiv \vec{x}(t) + \tau\vec{f}(\vec{x}(t),t) \ . \tag{11.24}$$

Para entender este esquema, consideremos nuevamante el caso en una variable. En nuestra fórmula original, estimamos que $f(x(\zeta),\zeta)$ como $\frac{1}{2}[f(x,t) + f(x^{\,*}(t+\tau),t+\tau)]$.

Estas fórmulas no son "sacadas de la manga"; se las puede deducir usando la expansión de Taylor con dos variables,

$$f(x+h,t+\tau) = \sum_{n=0}^{\infty}\frac{1}{n!}\left(h\frac{\partial}{\partial x} + \tau\frac{\partial}{\partial t}\right)^n f(x,t) \ , \tag{11.25}$$

donde todas las derivadas son evaluadas en $(x,\, t)$. Para una fórmula general de Runge-Kutta de segundo orden queremos obtener una expresión de la siguiente forma

$$x(t+\tau) = x(t) + w_1\tau f(x(t),t) + w_2\tau f(x^*, t+\alpha\tau) \ , \tag{11.26}$$

donde

$$x^* \equiv x(t) + \beta\tau f(x(t),t) \ . \tag{11.27}$$

Hay cuatro coeficientes no especificados: a, b, w_1 y w_2. Note que cubrimos las ecuaciones (11.19) y (11.20) eligiendo los valores

$$w_1 = 0 \ , \quad w_2 = 1 \quad \alpha = \frac{1}{2} \ , \quad \beta = \frac{1}{2} \ , \tag{11.28}$$

y las ecuaciones (11.23) y (11.24) eligiendo

$$w_1 = \frac{1}{2} \ , \quad w_2 = \frac{1}{2} \ , \quad \alpha = 1 \ , \quad \beta = 1 \ . \tag{11.29}$$

Deseamos seleccionar cuatro coeficientes tales que tengamos una precisión de segundo orden; esto es deseamos calzar la serie de Taylor a través de los términos de la segunda derivada. Los detalles del cálculo se proponen como un ejercicio, pero cualquier grupo de coeficientes que satisfagan las relaciones siguientes: $w_1 + w_2 = 1$, $\alpha w_2 = 1/2$ y $\alpha = \beta$, darán un esquema Runge-Kutta de segundo orden. El error de truncamiento local es $\mathcal{O}(\tau^3)$, pero la expresión explícita no tiene una forma simple. No está claro que un esquema sea superior al otro ya que el error de truncamiento, siendo una función complicada de $f(x,t)$, variará de problema a problema.

11.2.3 Runge-Kutta de cuarto orden.

Presentamos las fórmulas de Runge-Kutta de segundo orden porque es fácil de comprender su construcción. En la práctica, sin embargo, el método más comúnmente usado es la siguiente fórmula de cuarto orden:

$$\vec{x}(t+\tau) = \vec{x}(t) + \frac{1}{6}\tau\left[\vec{F}_1 + 2\vec{F}_2 + 2\vec{F}_3 + \vec{F}_4\right] \ , \tag{11.30}$$

donde

$$\begin{aligned}
\vec{F}_1 &= \vec{f}(\vec{x}, t) \ , \\
\vec{F}_2 &= \vec{f}\left(\vec{x} + \frac{1}{2}\tau\vec{F}_1, t + \frac{1}{2}\tau\right) \ , \\
\vec{F}_3 &= \vec{f}\left(\vec{x} + \frac{1}{2}\tau\vec{F}_2, t + \frac{1}{2}\tau\right) \ , \\
\vec{F}_4 &= \vec{f}(\vec{x} + \tau\vec{F}_3, t + \tau) \ .
\end{aligned} \tag{11.31}$$

El siguiente extracto del *Numerical Recipes*[4] resume mejor el estado que las fórmulas de arriba tienen en el mundo del análisis numérico:

> Para muchos usuarios científicos, el método de Runge-Kutta de cuarto orden no es sólo la primera palabra en esquemas de integración para ecuaciones diferenciales ordinarias, sino que es la última también. De hecho, usted puede ir bastante lejos con este viejo caballito de batalla, especialmente si los combina con un algoritmo de paso adaptativo ... Bulirsch-Stoer o los métodos predictor-corrector pueden ser mucho más eficientes para problemas donde se require una alta precisión. Estos métodos son los finos caballos de carrera mientras que Runge-Kutta es el fiel caballo de tiro.

Usted se preguntará, ¿por qué fórmulas de cuarto orden y no de orden superior? Bien, los métodos de orden superior tienen un error de truncamiento mejor, pero también requieren más cálculo, esto es, más evaluaciones de $f(x,t)$. Hay dos opciones, hacer más pasos con un τ pequeño usando un método de orden inferior o hacer pocos pasos con un τ más grande usando un método de orden superior. Ya que los métodos de Runge-Kutta de órdenes superiores son muy complicados, el esquema de cuarto orden dado anteriormente es muy conveniente. Entre paréntesis, el error de truncamiento local para Runge-Kutta de cuarto orden es $\mathcal{O}(\tau^5)$.

Para implementar métodos de cuarto orden para nuestro problema de la órbita, usaremos la función `rk4` (tabla 11.3). Esta función toma como datos: el estado actual del sistema, $\vec{x}(t)$; el paso de tiempo para ser usado, τ; el tiempo actual, t; la función $\vec{f}(\vec{x}(t), t; \lambda)$; donde λ es una lista de parámetros usados por $\vec{f}$. La salida es el nuevo estado del sistema, $\vec{x}(t+\tau)$, calculado por el método de Runge-Kutta. Usando Runge-Kutta de cuarto orden sse obtienen los resultados mostrados en la figura 11.6, la cual es mucho mejor que las obtenidas usando el método de Euler-Cromer (figura 11.5).

[4] W. Press, B. Flannery, S. Tukolsky and W. Vetterling, *Numerical Recipes in* FORTRAN, 2nd ed. (Cambridge: Cambridge University Press 1992).

- *Entradas:* $\vec{x}(t)$, t, τ, $\vec{f}(\vec{x}, t; \lambda)$, y λ.
- *Salidas:* $\vec{x}(t + \tau)$.
- Evaluación $\vec{F}_1$, $\vec{F}_2$, $\vec{F}_3$ y $\vec{F}_4$ usando ecuación (11.31).
- Cálculo de $\vec{x}(t + \tau)$ usando Runge-Kutta de cuarto orden, usando ecuación (11.30).

Tabla 11.3: Bosquejo de la función `rk4`, la cual evalúa un paso simple usando el método Runge-Kutta de cuarto orden.

- *Entradas:* $\vec{x}(t)$, t (no se usa), GM.
- *Salidas:* $d\vec{x}(t)/dt$.
- Evalúa la aceleración $\vec{a} = -(GM\vec{r}/\,|\,\vec{r}\,|^3)$.
- *Retorno:* $d\vec{x}(t)/dt = [v_x, v_y, a_x, a_y]$.

Tabla 11.4: Bosquejo de la función `gravrk`, la cual es usada por la función Runge-Kutta para evaluar las ecuaciones de movimiento para el problema de Kepler.

11.2.4 Pasando funciones a funciones.

La función de Runge-Kutta `rk4` es muy simple, pero introduce un elemento de programación que no hemos usado antes. La función $\vec{f}(\vec{x}, t; \lambda)$ está introducida como un parámetro de entrada a `rk4`. Esto nos permite usar `rk4` para resolver diferentes problemas cambiando simplemente la definición de $\vec{f}$ (como lo haremos en la última sección). Para el problema de Kepler, la función `gravrk` (tabla 11.4) define la ecuación de movimiento devolviendo dx/dt, ecuación (11.17).

En C++ el puntero a la función $\vec{f}(\vec{x}, t; \lambda)$ es pasado como parámetro a `rk4`. El programa `orbita` llama a `rk4` como

```
rk4( state, nState, time, tau, gravrk, param) ;
```

donde el vector de estado es $\vec{x} = [r_x, r_y, v_x, v_y]$. En el inicio del archivo, la función `gravrk` es declarada con el prototipo

```
void gravrk( double * x, double t, double param, double * deriv );
```

La primera línea de `rk4` es

```
void rk4(double * x, int nX, double t, double tau,
    void (*derivsRK) (double *, double, double, double *) , double param)
```

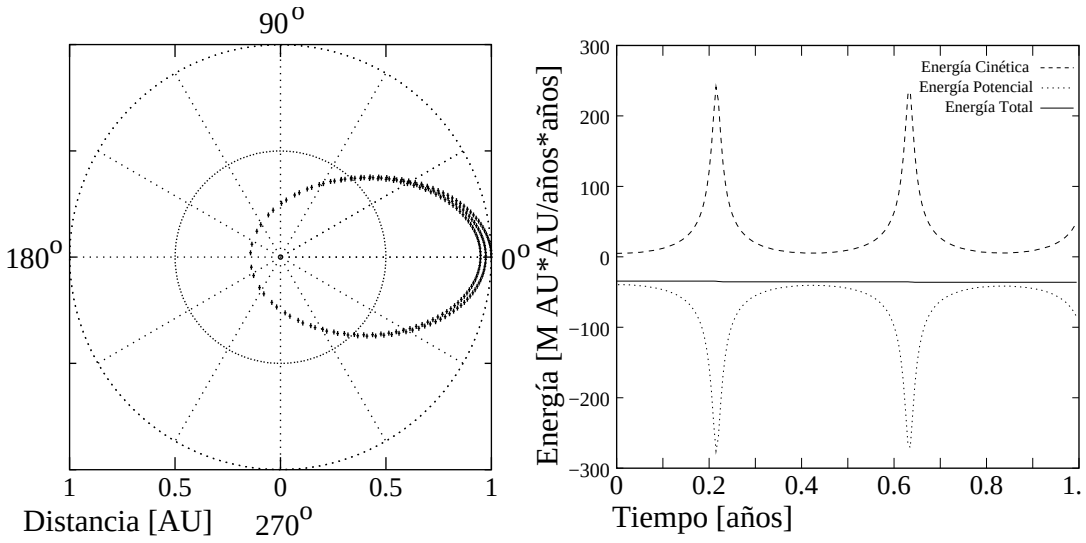


Figura 11.6: Gráfico de la trayectoria y la energía con el programa `orbita` usando el método de Runge-Kutta. La distancia radial inicial es 1 [AU] y la velocidad tangencial inicial es π [AU/año]. El paso en el tiempo es $\tau = 0.005$ [años]; y 200 pasos son calculados. Compararemos con la figura 11.5.

Cuando es llamado por `orbita`, esta función recibe un puntero a `gravrk` en la variable `derivsRK`. Dentro de `rk4`, la sentencia

```
(*derivsRK)( x, t, param, F1 ) ;
```

es equivalente a

```
gravrk( x, t, param, F1 ) ;
```

ya que `dervsRK` apunta a `gravrk`.

11.3 Métodos adaptativos

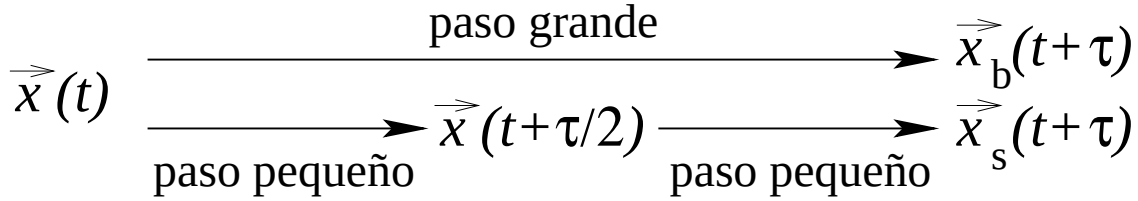
11.3.1 Programas con paso de tiempo adaptativo.

Ya que el método de Runge-Kutta de cuarto orden es más preciso (errores de truncamiento pequeños), hace un mejor trabajo dentro de una órbita altamemente elíptica. Aún para una distancia inicial al afelio de 1 [AU] y una velocidad inicial en el afelio de $\pi/2$ [AU/año], usando un paso tiempo tan pequeño como $\tau = 0.0005$ [años] ($\approx 4\frac{1}{2}$ [hrs]), la energía total varía sobre el 7% por órbita. Si pensamos la física, llegamos a que realizar una integración con un paso pequeño es sólo necesaria cuando el cometa haga su acercamiento más próximo, punto en el cual su velocidad es máxima. Cualquier error pequeño en la trayectoria cuando rodea al Sol causa una gran desviación en la energía potencial.

La idea ahora es diseñar un programa que use un paso de tiempo pequeño cuando el cometa está cerca del Sol y pasos de tiempo grande cuando está lejos. Tal como está,

normalmente tenemos sólo una idea aproximada de lo que τ pudiera ser; ahora tenemos que seleccionar un $\tau_{\text{mín}}$ y un $\tau_{\text{máx}}$ y una manera de intercambiar entre ellos. Si tenemos que hacer esto por prueba y error manual, podría ser peor que haciéndolo por la fuerza bruta calculando con un paso de tiempo pequeño toda la trayectoria. Idealmente, deseamos estar completamente liberados de tener que especificar un paso de tiempo. Deseamos tener una trayectoria calculada de la misma posición inicial hasta algún tiempo final con la seguridad de que la solución es correcta a una precisión especificada

Los programas adaptativos continuamente monitorean la solución y modifican el paso de tiempo para asegurar que se mantenga la precisión especificada por el usuario. Esos programas pueden hacer algunos cálculos extras para optimizar la elección de τ, en muchos casos este trabajo extra vale la pena. Aquí está una manera para implementar esta idea: dado el estado actual $\vec{x}(t)$, el programa calcula $\vec{x}(t+\tau)$ como siempre, y luego repite el cálculo haciendolo en dos pasos, cada uno con paso de tiempo $\frac{\tau}{2}$. Visualmente, esto es



La diferencia entre las dos respuestas, $\vec{x}_b(t+\tau)$ y $\vec{x}_s(t+\tau)$, estima el error de truncamiento local. Si el error es tolerable, el valor calculado es aceptado y un valor mayor de τ es usado en la próxima iteración. Por otra parte, si el error es muy grande, la respuesta es rebotada, el paso de tiempo es reducido y el procedimiento es repetido hasta que se obtenga una respuesta aceptable. El error de truncamiento estimado para el actual paso de tiempo puede guiarnos en seleccionar un nuevo paso de tiempo para la próxima iteración.

11.3.2 Función adaptativa de Runge-Kutta.

Aquí mostramos cómo una iteración adaptativa puede ser implementada para un esquema de Runge-Kutta de cuarto orden: llamemos Δ al error de truncamiento; sabemos que $\Delta \propto \tau^5$ para un esquema Runge-Kutta de cuarto orden. Supongamos que el paso de tiempo actual τ_{ant} da un error de $\Delta_c = |\,\vec{x}_b - \vec{x}_s\,|$; ésta es nuestra estimación para el error de truncamiento. Dado que deseamos que el error sea menor o igual que el error ideal especificado por el usuario, le llamamos Δ_i; luego, el nuevo paso de tiempo estimado es

$$\tau_{\text{est}} = \tau \left| \frac{\Delta_i}{\Delta_c} \right|^{1/5} . \tag{11.32}$$

Ya que esto es sólo una estimación, el nuevo paso de tiempo es $\tau_{\text{nuevo}} = S_1 \tau_{\text{est}}$, donde $S_1 < 1$. Esto nos hace sobreestimar el cambio cuando disminuimos τ y subestimar el cambio cuando lo aumentamos. Malogramos los esfuerzos computacionales cada vez que rebotamos una respuesta y necesitamos reducir el paso de tiempo, por lo tanto es mejor ajustar $\tau_{\text{nuevo}} < \tau_{\text{est}}$.

Podríamos poner un segundo factor de seguridad, $S_2 < 1$, para asegurarse que el programa no sea demasiado entusiasta en aumentar o disminuir precipitadamente el paso de tiempo.

Con ambas precauciones, el nuevo paso de tiempo es

$$\tau_{\text{nuevo}} = \begin{cases} S_2\tau_{\text{ant}} & \text{si } S_1\tau_{\text{est}} > S_2\tau_{\text{ant}} \\ \tau/S_2 & \text{si } S_1\tau_{\text{est}} < \tau_{\text{ant}}/S_2 \\ S_1\tau_{\text{est}} & \text{en otro caso} \end{cases} . \tag{11.33}$$

- *Entradas:* $\vec{x}(t)$, t, τ, Δ_i, $\vec{f}(\vec{x}, t; \lambda)$, y λ.
- *Salidas:* $x(t')$, t', τ_{nuevo}.
- Fijar las variables iniciales.
- Iterar sobre el número deseado de intentos para satisfacer el error límite.
  - Tomar dos pequeños pasos de tiempo.
  - Tomar un único paso grande de tiempo.
  - Calcule el error de truncamiento estimado.
  - Estime el nuevo valor de τ (incluyendo factores de seguridad).
  - Si el error es aceptable, regresar los valores calculados.

  Mostrar un mensaje de error si el error límite nunca es satisfecho.

Tabla 11.5: Bosquejo de la función `rka`, la cual evalúa un único paso usando un método adaptativo de Runge-Kutta de cuarto orden.

Esto obliga a asegurar que nuestro nueva estimación para τ nunca aumente o decrezca por más que un factor S_2. Por supuesto, este nuevo τ podría ser insuficientemente pequeño, y tendríamos que continuar reduciendo el paso de tiempo; pero al menos sabríamos que no ocurrirá de un modo incontrolado.

Este procedimiento no es a prueba de balas. Los errores de redondeo llegan a ser significativos en pasos de tiempos muy pequeños. Por esta razón la iteración adaptativa podría fallar para encontrar un paso de tiempo que dé la precisión deseada. Debemos mantener esta limitación en mente cuando especifiquemos el error aceptable. Una función de Runge-Kutta adaptativa, llamada `rka`, es esbozada en la tabla 11.5. Note que los datos de entrada en la secuencia de llamada son los mismos que para `rk4`, excepto por la suma de Δ_i, el error ideal especificado. Las salidas del `rka` son el nuevo estado del sistema, $\vec{x}(t')$; el tiempo nuevo, t' y el nuevo paso de tiempo, τ nuevo, el cual podría ser usado la próxima vez que sea llamada la `rka`.

Usando el método de Runge-Kutta adaptativo, el programa `orbita` da los resultados en la figura 11.7 para una órbita altamente elíptica. Notemos que el programa toma muchos más pasos en el perihelio que en el afelio. Podemos comparar con los resultados usando el método de Runge-Kutta no adaptativo (figura 11.6) en el cual los pasos en el perihelio son

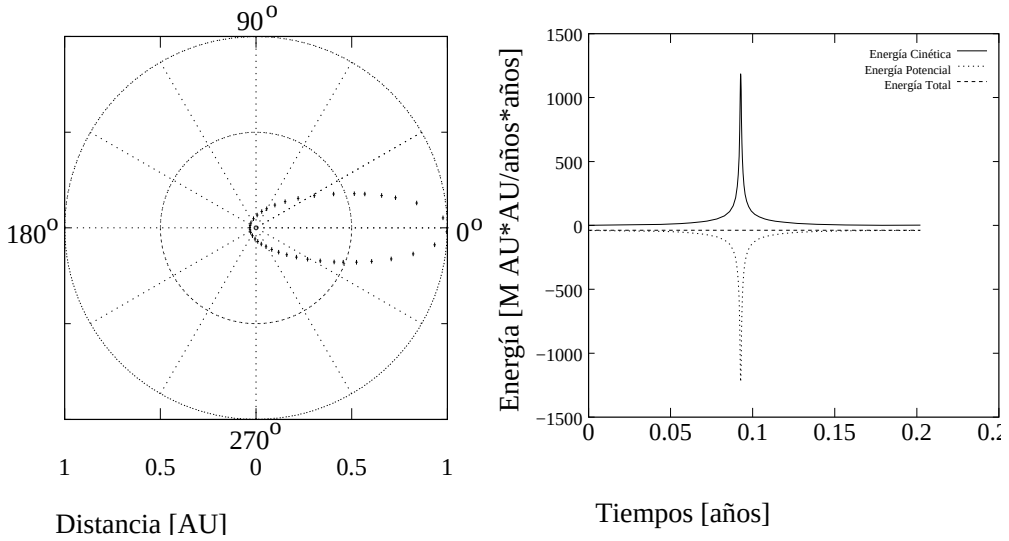


Figura 11.7: Gráfico de la trayectoria y la energía con el programa `orbita` usando el método de Runge-Kutta adaptativo. La distancia radial inicial es 1 [AU] y la velocidad tangencial inicial es $\pi/2$ [AU/año]. El paso inicial en el tiempo es $\tau = 0.1$ [años]; y 40 pasos son calculados.

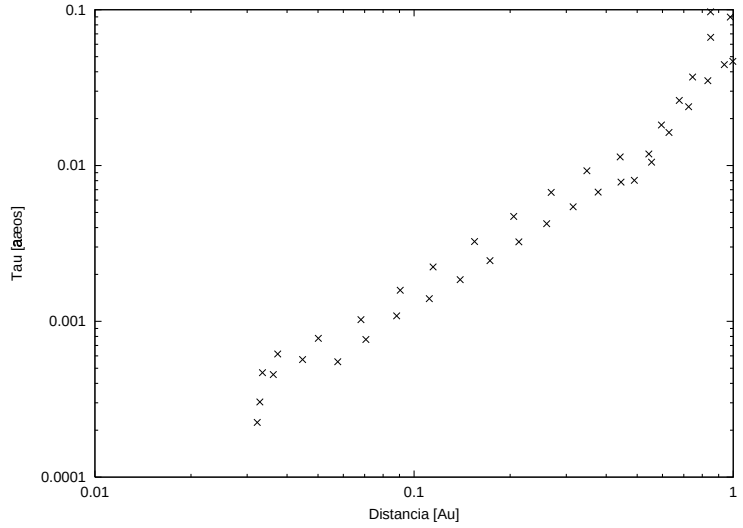


Figura 11.8: Paso de tiempo como función de la distancia radial con el programa `orbita` usando el método de Runge-Kutta adaptativo. Los paramétros son los mismos de la figura 11.7.

ampliamente espaciados. Una gráfica de los pasos de tiempo versus la distancia radial (figura 11.8) muestra que τ varía casi tres órdenes de magnitud. Interesantemente esta gráfica revela una relación exponencial aproximada de la forma $\tau \propto \sqrt{r^3}$. Por supuesto esta dependencia nos recuerda la tercera ley de Kepler, ecuación (11.10). Esperamos alguna dispersión en los puntos, ya que nuestra rutina adaptada solamente estima el paso de tiempo óptimo.

11.4 Listados del programa.

11.4.1 orbita.cc

```
#include "NumMeth.h"

const double GM=4.0e0*M_PI*M_PI ;
const double masaCometa = 1.0e0 ;
const double adaptErr = 1.0e-3;

void rk4(double * x, int nX, double t, double tau,
         void(*derivsRK)(double *, double, double, double *),
         double param)
{
  double * F1=new double [nX] ;
  double * F2=new double [nX] ;
  double * F3=new double [nX] ;
  double * F4=new double [nX] ;
  double * xtemp=new double [nX] ;

  // Evaluemos F1=f(x,t)
  (*derivsRK) (x, t, param, F1) ;

  double half_tau = tau/2.0e0;
  double t_half = t+half_tau ;
  double t_full = t+tau ;

  // Evaluamos F2=f(x+tau*F1/2, t+tau/2)
  for(int i=0; i<nX; i++) xtemp[i]=x[i]+half_tau*F1[i] ;
  (*derivsRK) (xtemp, t_half, param, F2) ;

  // Evaluamos F3=f(x+tau*F2/2, t+tau/2)
  for(int i=0; i<nX; i++) xtemp[i]=x[i]+half_tau*F2[i] ;
  (*derivsRK) (xtemp, t_half, param, F3) ;

  // Evaluamos F4=f(x+tau*F3, t+tau)
  for(int i=0; i<nX; i++) xtemp[i]=x[i]+tau*F3[i] ;
  (*derivsRK) (xtemp, t_full, param, F4) ;

  // Retornamos x(t+tau)

  for(int i=0; i<nX; i++) x[i] += tau*(F1[i]+F4[i]+2.0e0*(F2[i]+F3[i]))/6.0e0 ;

  delete [] F1, F2, F3, F4, xtemp ;
}
```

```
void rka ( double * x, int nX, double  & t, double & tau, double erro,
           void(*derivsRK)(double *, double, double, double *),
           double param)
{
  double tSave = t ;
  double safe1 = 0.9, safe2 = 4.0 ; //factores de seguridad

  double * xSmall = new double[nX] ;
  double * xBig = new double[nX] ;
  int maxTray=100 ;
  for (int iTray=0; iTray<maxTray; iTray++) {
    // Tomemos dos peque{\~n}os pasos en el tiempo
    double half_tau = 0.5*tau ;
    for (int i =0; i < nX; i++) xSmall[i]=x[i] ;
    rk4( xSmall, nX, tSave, half_tau, derivsRK, param) ;
    t= tSave + half_tau ;
    rk4( xSmall, nX, t, half_tau, derivsRK, param) ;
   // Tomemos un solo tiempo grande
    for (int i =0; i < nX; i++) xBig[i]=x[i] ;
    rk4( xBig, nX, tSave, tau, derivsRK, param) ;

    // Calculemos el error de truncamiento estimado
    double erroRatio = 0.0e0  ;
    double eps = 1.0e-16 ;
    for (int i = 0 ; i < nX; i++) {
      double scale = erro * (fabs(xSmall[i]) + fabs(xBig[i]))/2.0e0 ;
      double xDiff = xSmall[i] - xBig[i] ;
      double ratio = fabs(xDiff)/(scale+eps) ;
      erroRatio = (erroRatio > ratio ) ? erroRatio:ratio ;
    }
    // Estimamos el nuevo valor de tau (incluyendo factores de seguridad)
    double tau_old= tau ;
    tau = safe1*tau_old*pow(erroRatio, -0.20) ;
    tau = (tau > tau_old/safe2) ? tau:tau_old/safe2 ;
    tau = (tau < safe2*tau_old) ? tau:safe2*tau_old ;

    // Si el error es aceptable regrese los valores computados
    if ( erroRatio < 1 ) {
      for (int i =0 ; i < nX; i++) x[i]=xSmall[i] ;
      return ;
    }
  }
  cout << "Error: Runge-Kutta adaptativo fallo" << endl ;
  exit(-1) ;
```

```
}


void gravrk( double * x, double t, double param, double * deriv)
{
  double gm=param ;
  double rX=x[0], rY=x[1];
  double vX=x[2], vY=x[3] ;
  double mod_r= sqrt(rX*rX+rY*rY) ;
  double aX= -gm*rX/(mod_r*mod_r*mod_r) ;
  double aY= -gm*rY/(mod_r*mod_r*mod_r) ;

  // Retorna la derivada

  deriv[0] = vX;
  deriv[1] = vY;
  deriv[2] = aX;
  deriv[3] = aY;
}

main()
{
  ofstream salida0 ("Orbita.txt") ;
  ofstream salidaE ("Energia.txt") ;
  ofstream salidaT ("Tau.txt") ;

  double r0 ;
  cout << "Ingrese la distancia radial inicial [AU]: " ;
  cin >> r0 ;
  double vT ;
  cout << "Ingrese la velocidad tangencial inicial [AU/a{\~n}os]: " ;
  cin >> vT ;
  double x0=r0 ;
  double y0=0.0e0;
  double vx0=0.0e0 ;
  double vy0=vT;
  //
  // Suponemos angulo inicial nulo
  //
  int metodo = 0 ;
  while( metodo < 1|| metodo > 4 ) {
    cout << "Ingrese el m{\'e}todo num{\'e}rico a usar :" << endl ;
    cout << "\t M{\'e}todo de Euler \t\t\t[1]" << endl;
    cout << "\t M{\'e}todo de Euler-Cromer \t\t[2]" << endl;
    cout << "\t M{\'e}todo de Runge-Kutta 4 orden \t\t[3]" << endl;
```

```
    cout << "\t M{\'e}todo de Runge-Kutta adaptativo \t[4]" << endl;
    cout << "elija: " ;
    cin >> metodo ;
  }
  double tau ;
  cout << "Ingrese paso en el tiempo: " ;
  cin >> tau ;
  int numPasos ;
  cout << "Ingrese el n{\'u}mero de pasos: " ;
  cin >> numPasos ;

  double param=GM ;
  const int dimX= 4;
  double * x = new double[dimX] ;

  double xN= x0;
  double yN= y0;
  double vxN=vx0;
  double vyN=vy0;
  double vxNp1, vyNp1, xNp1, yNp1;
  double tiempo = 0.0e0 ;

  for(int pasos=0; pasos < numPasos; pasos++) {
    double r =sqrt(xN*xN+yN*yN) ;
    double v2 =vxN*vxN+vyN*vyN ;
    double theta= atan2(yN, xN) ;
    salida0 << theta << " " << r << endl ;

    double Ep = -GM*masaCometa/r ;
    double Ek = masaCometa*v2/2.0e0 ;
    double ET= Ep+Ek ;

    salidaE<< tiempo << " " << Ek<< " " << Ep<<" " << ET << endl ;

    double modr3=pow(xN*xN+yN*yN, 3.0e0/2.0e0) ;
    double axN= -GM*xN/modr3 ;
    double ayN= -GM*yN/modr3 ;
    switch( metodo ) {
    case 1: {                    // Euler
      vxNp1=vxN+tau*axN ;
      vyNp1=vyN+tau*ayN ;
      xNp1= xN+tau* vxN ;
      yNp1= yN+tau* vyN ;
      tiempo += tau ;
    }
```

```
    break ;
    case 2: {                    // Euler-Cromer
      vxNp1=vxN+tau*axN ;
      vyNp1=vyN+tau*ayN ;
      xNp1= xN+tau* vxNp1 ;
      yNp1= yN+tau* vyNp1 ;
      tiempo += tau ;
    }
    break ;
    case 3: {                    // Runge-Kutta 4to Orden
      x[0] = xN;
      x[1] = yN;
      x[2] = vxN;
      x[3] = vyN;
      rk4( x, dimX, tiempo, tau, gravrk, param);
      xNp1=x[0] ;
      yNp1=x[1];
      vxNp1=x[2];
      vyNp1=x[3];
      tiempo += tau ;
    }
    break ;
    case 4: {
      x[0] = xN;
      x[1] = yN;
      x[2] = vxN;
      x[3] = vyN;
      rka( x, dimX, tiempo, tau, adaptErr, gravrk, param);
      double distancia = sqrt( x[0]*x[0]+x[1]*x[1]) ;
      salidaT<< distancia << " " <<tau << endl ;
      xNp1=x[0] ;
      yNp1=x[1];
      vxNp1=x[2];
      vyNp1=x[3];
    }
    }
    xN=xNp1 ;
    yN=yNp1 ;
    vxN=vxNp1 ;
    vyN=vyNp1 ;
  }
  salidaO.close() ;
  salidaE.close() ;
  salidaT.close() ;
  delete [] x;
```

```
}
```

Capítulo 12

Resolviendo sistemas de ecuaciones.

versión final 2.0-19 agosto 2002[1]

En este capítulo aprenderemos a cómo resolver sistemas de ecuaciones de ambos tipos, lineal y no lineal. Ya conocemos los algoritmos básicos para los sistemas lineales: eliminar variables hasta que tengamos una sola ecuación con una sola incógnita. Para sistemas no-lineales desarrollamos un esquema iterativo que en cada paso resuelve una versión linealizada de estas ecuaciones. Para mantener la continuidad, la discución será motivada por el cálculo del estado estacionario, un importante tema en ecuaciones diferenciales ordinarias.

12.1 Sistemas de ecuaciones lineales.

12.1.1 Estado estacionario de EDO.

En el capítulo pasado, nosotros vimos cómo resolver ecuaciones diferenciales ordinarias de la forma

$$\frac{d\vec{x}}{dt} = \vec{f}(\vec{x}, t) \ , \tag{12.1}$$

donde $\vec{x} = [x_1, x_2, \ldots, x_N]$. Dada una condición inicial para las N variables $x_i(t) = 0$, nosotros podemos calcular la serie de tiempo $x_i(t)$ por una variedad de métodos (*e.g.* Runge-Kutta).

Los ejemplos que hemos estudiado hasta aquí han sido sistemas autónomos donde $\vec{f}(\vec{x}, t) = \vec{f}(\vec{x})$, esto es, $\vec{f}$ no depende explícitamente del tiempo. Para sistemas autónomos, a menudo existe una importante clase de condiciones iniciales para las cuales $x_i(t) = x_i(0)$ para todo i y t. Estos puntos en el espacio N-dimensional de nuestras variables son llamados *estado estacionario*. Si nosotros partimos de un estado estacionario nos quedamos ahí para siempre. Localizar los estados estacionarios para ecuaciones diferenciales ordinarias es importante, ya que ellos son usados en el análisis de bifurcaciones.[2]

Es fácil ver que $\vec{x}^* = [x_1^*, x_2^*, \ldots, x_N^*]$ es un estado estacionario si y sólo si

$$\vec{f}(\vec{x}^*) = 0 \ , \tag{12.2}$$

[1]Este capítulo está basado en el cuarto capítulo del libro: *Numerical Methods for Physics, second edition* de Alejandro L. Garcia, editorial Prentice Hall.

[2]R. Seydel, *From Equilibrium to Chaos, Practical Bifurcation and Stability Analysis* (New York: Elsevier, 1988).

o

$$f_i(x_1^*, x_2^*, \ldots, x_N^*) = 0 \ , \quad \text{para todo } i, \tag{12.3}$$

ya que esto implica que $d\vec{x}^*/dt = 0$. Localizar los estados estacionarios se reduce al problema de resolver N ecuaciones con N incognitas ${x_i}^*$. Este problema es también llamado "encontrar las raices de $f(x)$".

Como un ejemplo, consideremos el péndulo simple; el estado está descrito por el ángulo θ y la velocidad angular ω. Los estados estacionarios se encuentran al resolver las ecuaciones no lineales

$$-\frac{g}{L}\,\mathrm{sen}\,\theta^* = 0 \ , \quad \omega^* = 0 \ . \tag{12.4}$$

Las raices son $\theta^* = 0, \pm\pi, \pm 2\pi, \ldots$ y $\omega^* = 0$. Por supuesto que no todos los sistemas son tan fáciles de resolver.

Debería ser claro que encontrar raices tiene muchas más aplicaciones que calcular los estados estacionario. En este capítulo consideraremos una variedad de maneras para resolver ambos sistemas lineales y sistemas no lineales. En esta sección y en la próxima consideraremos sistemas lineales, dejando los sistemas no lineales para la parte final del capítulo.

12.1.2 Eliminación Gaussiana.

El problema de resolver $f_i(\{x_j\}) = 0$ se divide en dos importantes clases. En esta sección consideramos el caso fácil cuando $f_i(\{x_j\})$ es una función lineal. El problema en ese caso se reduce a resolver un sistema de N ecuaciones con N incógnitas.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N - b_1 &= 0 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2N}x_N - b_2 &= 0 \\ \vdots \qquad\quad \vdots \qquad\quad \vdots \qquad\quad &\vdots \\ a_{N1}x_1 + a_{N2}x_2 + \ldots + a_{NN}x_N - b_N &= 0 \ , \end{aligned} \tag{12.5}$$

o en forma matricial

$$\check{A}\,\vec{x} - \vec{b} = 0 \ , \tag{12.6}$$

donde

$$\check{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots \\ a_{21} & a_{22} & \ldots \\ \vdots & \vdots & \ddots \end{bmatrix} \ , \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix} \ , \quad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \end{bmatrix} \ . \tag{12.7}$$

Sabemos cómo resolver este conjunto de ecuaciones. Combinando las ecuaciones para eliminar variables hasta que tengamos una sola ecuación con una sola incógnita. Hagamos un ejemplo simple para revisar cómo el procedimiento trabaja. Tomemos las ecuacion es

$$\begin{aligned} x_1 \ + \ x_2 \ + \ x_3 \ &= \ 6 \ , \\ -x_1 \ + \ 2x_2 \ \qquad &= \ 3 \ , \\ 2x_1 \ \qquad + \ x_3 \ &= \ 5 \ . \end{aligned} \tag{12.8}$$

Deseamos eliminar x_1 de la segunda y de la tercera ecuación. Para llevar a cabo esto, primero sumamos la primera y la segunda y después restamos dos veces la primera a la tercera. Esto nos da

$$\begin{aligned} x_1 + x_2 + x_3 &= 6\,, \\ 3x_2 + x_3 &= 9\,, \\ -2x_2 - x_3 &= -7\,. \end{aligned} \tag{12.9}$$

Ahora, eliminamos x_2 de la última ecuación multiplicando la segunda ecuación por $-\frac{2}{3}$ y la restamos de la tercera, dándonos

$$\begin{aligned} x_1 + x_2 + x_3 &= 6\,, \\ 3x_2 + x_3 &= 9\,, \\ -\tfrac{1}{3}x_3 &= -1\,. \end{aligned} \tag{12.10}$$

Este procedimiento es llamado *eliminación hacia adelante*. Si se tienen N ecuaciones, eliminamos x_1 de las ecuaciones 2 hasta la N, luego eliminamos x_1 y x_2 de las ecuaciones 3 hasta la N, y así sucesivamente. La última ecuación sólo contendrá la variable x_N.

Volviendo al ejemplo, es ahora trivial resolver la tercera ecuación resultando $x_3 = 3$. Podemos ahora sustituir este valor en la segunda ecuación obteniendo $3x_2 + 3 = 9$ tal que $x_2 = 2$. Finalmente, introduciendo los valores de x_2 y x_3 en la primera ecuación obtenemos $x_1 = 1$. Este segundo procedimiento es llamado *sustitución hacia atrás*. Debería ser claro cómo esto trabaja con sistemas grandes de ecuaciones. Usando la última ecuación obtenemos x_N, éste es usado en la penúltima ecuación para obtener x_{N-1} y así seguimos.

Este método de resolver sistemas de ecuaciones lineales por eliminación hacia adelante y luego sustitución hacia atrás es llamado *eliminación Gaussiana*.[3] Esta es una secuencia rutinaria de pasos que es simple para un computador realizar sistemáticamente. Para N ecuaciones con N incógnitas, el tiempo de computación para eliminación Gaussiana va como N^3. Afortunadamente, si el sistema es esparcido[4] (la mayoría de los coeficientes son cero), El tiempo de cálculo puede ser considerablemente reducido.

12.1.3 Pivoteando.

La eliminación Gaussiana es un procedimiento simple, sin embargo, se debe estar conciente de sus riesgos. Para ilustrar la primera fuente de problemas, consideremos el conjunto de ecuaciones

$$\begin{aligned} \epsilon x_1 + x_2 + x_3 &= 5\,, \\ x_1 + x_2 &= 3\,, \\ x_1 + x_3 &= 4\,. \end{aligned} \tag{12.11}$$

En el límite $\epsilon \to 0$ la solución es $x_1 = 1$, $x_2 = 2$, $x_3 = 3$. Para estas ecuaciones, el primer paso de la eliminación hacia adelante podría partir por multiplicar la primera ecuación por

[3] G.E. Forsythe and C.B. Moler, *Computer Solution of Linear Algebraic System* (Upper Saddle River, N.J.: Prentice-Hall, 1967).

[4] *sparse*

$(1/\epsilon)$ y sustrayéndola de la segunda y tercera ecuaciones, lo que da

$$\begin{aligned} \epsilon x_1 \;+\; & x_2 \;+\; & x_3 \;&=\; 5\,, \\ (1-1/\epsilon)x_2 \;-\; & (1/\epsilon)x_3 \;&=\; 3-5/\epsilon\,, \\ -(1/\epsilon)x_2 \;+\; & (1-1/\epsilon)x_3 \;&=\; 4-5/\epsilon\,. \end{aligned} \tag{12.12}$$

Por supuesto, si $\epsilon = 0$ tenemos grandes problemas, ya que el factor $1/\epsilon$ estalla. Aún si $\epsilon \neq 0$, pero es pequeño, vamos a tener serios problemas de redondeo. Supongamos que $1/\epsilon$ es tan grande que $(C - 1/\epsilon) \to -1/\epsilon$, donde C es del orden de la unidad. Nuestras ecuaciones, después del redondeo, llegan a ser

$$\begin{aligned} \epsilon x_1 \;+\; & x_2 \;+\; & x_3 \;&=\; 5\,, \\ -(1/\epsilon)x_2 \;-\; & (1/\epsilon)x_3 \;&=\; -5/\epsilon\,, \\ -(1/\epsilon)x_2 \;-\; & (1/\epsilon)x_3 \;&=\; -5/\epsilon\,. \end{aligned} \tag{12.13}$$

En este punto está claro que no podemos proceder ya que la segunda y la tercera ecuación en (12.13) son ahora idénticas; no tenemos más que tres ecuaciones independientes. El próximo paso de eliminación sería transformar la tercera ecuación en (4.13) en la tautología $0 = 0$.

Afortunadamente, hay una solución simple: intercambiar el orden de las ecuaciones antes de realizar la eliminación. Cambiando las ecuaciones primera y segunda en (12.11),

$$\begin{aligned} x_1 \;+\; & x_2 & &\;=\; 3\,, \\ \epsilon x_1 \;+\; & x_2 \;+\; & x_3 \;&=\; 5\,, \\ x_1 \;+\; & & x_3 \;&=\; 4\,. \end{aligned} \tag{12.14}$$

El primer paso de la eliminación hacia adelante nos da las ecuaciones

$$\begin{aligned} x_1 \;+\; & x_2 & &\;=\; 3\,, \\ (1-\epsilon)x_2 \;+\; & x_3 \;&=\; 5-3\epsilon\,, \\ -x_2 \;+\; & x_3 \;&=\; 4-3\,. \end{aligned} \tag{12.15}$$

El redondeo elimina los términos proporcionales a ϵ, dando

$$\begin{aligned} x_1 \;+\; & x_2 & &\;=\; 3\,, \\ & x_2 \;+\; & x_3 \;&=\; 5\,, \\ & -x_2 \;+\; & x_3 \;&=\; 1\,. \end{aligned} \tag{12.16}$$

El segundo paso de eliminación hacia adelante elimina x_2 de la tercera ecuación en (12.16) usando la segunda ecuación,

$$\begin{aligned} x_1 \;+\; & x_2 & &\;=\; 3\,, \\ & x_2 \;+\; & x_3 \;&=\; 5\,, \\ & & 2x_3 \;&=\; 6\,. \end{aligned} \tag{12.17}$$

Se puede comprobar fácilmente que la sustitución hacia atrás da $x_1 = 1$, $x_2 = 2$, y $x_3 = 3$, la cual es la respuesta correcta en el límite $\epsilon \to 0$.

Los algoritmos que reordenan las ecuaciones cuando ellas sitúan elementos pequeños en la diagonal son llamados *pivotes*. A menudo si todos los elementos de una matriz son inicialmente de una magnitud comparable, el procedimiento de eliminación hacia adelante podría producir pequeños elementos sobre la diagonal principal. El precio de pivotear es sólo un par de líneas extras en el programa, pero es esencial usar pivoteo para todas, no sólo para las matrices más pequeñas. Aún con el pivoteo, no podemos garantizar estar a salvo de problemas de redondeo si se trata de matrices muy grandes.

12.1.4 Determinantes.

Es fácil obtener el determinante de una matriz usando la eliminación Gaussiana. Después de completar la eliminación hacia adelante, uno simplemente calcula el producto de los coeficientes de los elementos diagonales. Tomando nuestro ejemplo original, la ecuación (12.8); la matriz es

$$\check{A} = \begin{bmatrix} 1 & 1 & 1 \\ -1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix} . \tag{12.18}$$

Completada la eliminación hacia adelante, ecuación (12.10), el producto de los coeficientes de los elementos diagonales es $(1)(3)(-\frac{1}{3}) = -1$, el cual, usted puede comprobar, es el determinante de $\check{A}$. Este método es sutílmente más complicado cuando se usa el pivoteo. Si el número de pivoteos es impar, el determinante es el producto negativo de los coeficientes de los elementos de la diagonal. Ahora debería ser obvio que la regla de Cramer es computacionalmente una manera ineficiente para resolver el conjunto de ecuaciones lineales.

12.1.5 Eliminación Gaussiana en Octave.

No necesitamos escribir un programa en Octave para realizar la eliminación Gaussiana con pivoteo. En vez de esto nosotros podemos usar las capacidades de manipulación de matrices que viene con Octave. En Octave, la eliminación Gaussiana es una rutina primitiva, tal como las funciones seno o raíz cuadrada. Como con cualquier rutina "enlatada", usted debería entender, en general, como trabaja y reconocer posibles problemas, especialmente los computacionales (*e.g.*, ¿poner `sqrt(-1)` retorna un número imaginario o un error?).

Octave implementa la eliminación de Gauss usando los operadores *slash* `/` y *backslash* `\`. El sistema de ecuaciones lineales $\vec{x}\,\check{A} = \vec{b}$ donde $\vec{x}$ y $\vec{b}$ son vectores fila, se resuelve usando el operador *slash* como $\vec{x} = \vec{b}/\check{A}$. El sistema de ecuaciones lineales $\check{A}\,\vec{x} = \vec{b}$ donde $\vec{x}$ y $\vec{b}$ son vectores columna, se resuelve usando el operador *backslash* como $\vec{x} = \vec{b}\backslash\check{A}$. Como un ejemplo, tomemos la ecuación (12.11) con $\epsilon = 0$, escrito en forma matricial

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 4 \end{bmatrix} \tag{12.19}$$

Usando Octave en forma interactiva la solución es ilustrada a continuación:

```
octave> A=[0 1 1; 1 1 0; 1 0 1];
octave> b=[5;3;4];
octave> x=A\b;
octave> disp(x);
  1
  2
  3
```

Claramente, Octave usa pivoteo en este caso. Los operadores *slash* y *backslash* pueden ser usados de la misma manera en programas. El comando en Octave para el determinante de una matriz es `det(A)`.

12.1.6 Eliminacion Gaussiana con C++ de objetos matriciales.

El uso de arreglos multidimensionales en C++ no es del todo comodo. Las maneras usuales de declarar un arreglo de $M \times N$ de números con punto flotante en doble precisión son:

```
const int M=3, N=3;
double A[M][N] ;
```

para asignación estática de memoria y

```
int M, N
. . .  // Se fijan valores a M y N
double * * A = new double * [M] ; // asignacion de un vector de punteros
for(int i=0;i<M;i++) {
     A[i] = new double [N] ; // asignacion de memoria para cada fila
}
```

para un reserva dinámica (no olvide desasignar cada fila con un `delete[]`). Una asignación estática es simple pero rígida, ya que las dimensiones son constantes fijas. Cuando el arreglo estático es pasado a una función, esta función debería declarar algún arreglo con el mismo número de columnas de acuerdo al arreglo a ser indexado apropiadamente. Una asignación dinámica es flexible pero es difícil de manejar, aunque los detalles pueden ser escondidos dentro de funciones separadas. Accesar elementos fuera de los contornos del arreglo es un error de programación común (*bug*) el cual es difícil de rastrear con arreglos dinámicos.

C++ nos permite corregir estas deficiencias creando nuestros propios tipos de variables. Estas variables definidas por el usuario son llamadas *clases de objetos* . De aquí en adelante usaremos la clase `Matrix` para declarar arreglos de una o dos dimensiones de números de punto flotante. Esta clase está enteramente declarada dentro del archivo de cabecera `Matrix.h` e implementada dentro de `Matrix.cc`. Algunos ejemplos de declaración de objetos de `Matrix` son

```
int M=3, N=2 ;
Matrix A(M,N), b(N), x(3) ;
```

Aunque esto se parece a una asignación de arreglo estático note que las dimensiones no son constantes fijas. Tanto vectores unidimensionales como matrices bidimensionales pueden ser declarados; las anteriores son tratadas como matrices de una sola columna. Los valores en estas variables pueden ser fijados por la declaración de asignamiento

```
A(0,0)=0;          A(0,1)=1;          A(0,2)=1;
A(1,0)=1;          A(1,1)=1;          A(1,2)=0;
A(2,0)=1;          A(2,1)=0;          A(2,2)=1;
b(1)=5;            b(2)=3;            b(3)=4.
```

El formato para el objeto `Matrix` es $A(i,j)$ en vez de $A[i][j]$, para distinguirlos de los arreglos C++ convencionales.

Un objeto `Matrix` conoce sus dimensiones, y usted puede obtenerlas usando las funciones miembros `nRow()` y `nCol()`. Por ejemplo

- *Entradas:* $\check{A}$, $\vec{b}$.
- *Salidas:* $\vec{x}$, $\det \check{A}$.
- Fija el factor de escala, $s_i = \max_j(|\, A_{i,j}\,|)$ para cada fila.
- Itera sobre las filas $k = 1, \ldots, (N-1)$.
  - Selecciona la columna a pivotear a partir de $\max_j(|\, A_{i,j}\,|)/s_i$.
  - Realiza pivoteos usando la lista de índice de columnas.
  - Realiza la eliminación hacia adelante.
- Calcula el determinante $\det \check{A}$, como producto de los elementos de la diagonal.
- Realiza la sustitución hacia atrás.

Tabla 12.1: Bosquejo de la función `ge`, la cual resuelve un sistema de ecuaciones lineales $\check{A}\,\vec{x} = \vec{b}$ por eliminación Gaussiana. La función también devuelve el determinante de $\check{A}$.

```
int m = A.nRow(), n = A.nCol() ;
```

asigna `m` y `n` a las dimensiones de `A`. La verificación de los contornos se realiza cada vez que un objeto `Matrix` es indexado. Las líneas

```
Matrix newA(3, 7) ;  // Una matriz no cuadrada de 3 por 7
newA(4,5) = 0;        // Fuera de los limites
```

produce el mensage de error: `Indice de fila fuera de los limites` cuando el programa es corrido. Esto significa que al menos uno de los índices i no satisface la condición $0 < i \leq N$. Los objetos `Matrix` automáticamente liberan la memoria que les fue asignada cuando ellos exceden su alcance (*scope*), por lo tanto no es necesario invocar `delete`.

Para asignar todos los elementos de un objeto `Matrix` a un valor dado usamos la función miembro `set(double x )`, por ejemplo `A.set(1.0)`. Una matriz entera puede ser asignada a otra del mismo tamaño (*e.g.*, `Matrix C(3,3); C=A;`). Sin embargo, a diferencia de las matrices de Octave, las operaciones aritméticas (`A+C`, `2*A`, etc.) no son aún permitidas. Estas operaciones deberían ser llevadas a cabo elemento por elemento, típicamente usando un ciclo `for`.

La rutina de eliminación Gaussiana `ge`, la cual usa los objetos tipo `Matrix`, es descrita en la tabla 12.1. Declarando y asignando la matriz `A` y los vectores `b` y `x` como arriba, un programa puede llamar esta rutina como

```
double determ = ge(A, b, x) ; // Eliminacion Gaussiana
cout << x(1) << ", " << x(2) << ", " << x(3) << endl ;
cout   << "Determinante = " << determ << endl ;
```

para resolver el sistema lineal $\check{A}\,\vec{x} = \vec{b}$ y calcular el determinante de $\check{A}$.

12.2 Matriz inversa.

12.2.1 Matriz inversa y eliminación Gaussiana.

En la sección previa hemos revisado cómo resolver un conjunto de ecuaciones simultáneas por eliminación de Gauss. Sin embargo, si usted está familiarizado con el álgebra lineal, probablemente podría escribir la solución de

$$\check{A}\,\vec{x} = \vec{b}\ , \tag{12.20}$$

como

$$\vec{x} = \check{A}^{-1}\,\vec{b}\ , \tag{12.21}$$

donde $\check{A}^{-1}$ es la matriz inversa de $\check{A}$. No nos debería sorprender que el cálculo de la matriz inversa esté relacionada al algoritmo para resolver un conjunto de ecuaciones lineales. Usualmente la inversa de una matriz es calculada por aplicaciones repetidas de eliminaciones de Gauss (o una variante de descomposición llamada LU).

La inversa de una matriz está definida por la ecuación

$$\check{A}\,\check{A}^{-1} = \check{I}\ , \tag{12.22}$$

donde $\check{I}$ es la matriz identidad

$$\check{I} = \begin{bmatrix} 1 & 0 & 0 & \dots \\ 0 & 1 & 0 & \dots \\ 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{12.23}$$

Definiendo los vectores columna

$$\hat{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}\ , \quad \hat{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}\ , \quad \dots\ , \quad \hat{e}_N = \begin{bmatrix} \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}\ , \tag{12.24}$$

podríamos escribir la matriz identidad como una vector fila de los vectores columna

$$\check{I} = \begin{bmatrix} \hat{e}_1\ \hat{e}_2\ \dots\ \hat{e}_N \end{bmatrix}\ . \tag{12.25}$$

Si resolvemos el conjunto de ecuaciones lineales,

$$\check{A}\,\vec{x}_1 = \hat{e}_1\ . \tag{12.26}$$

El vector solución $\vec{x}_1$ es la primera columna de la inversa $\check{A}^{-1}$. Si procedemos de esta manera con los otros $\hat{e}$ calcularemos todas las columnas de $\check{A}^{-1}$. En otras palabras, nuestra ecuación para la matriz inversa $\check{A}\check{A}^{-1} = \check{I}$ es resuelta escribiéndola como

$$\check{A}\ \begin{bmatrix} \hat{x}_1\ \hat{x}_2\ \dots\ \hat{x}_N \end{bmatrix} = \begin{bmatrix} \hat{e}_1\ \hat{e}_2\ \dots\ \hat{e}_N \end{bmatrix}\ . \tag{12.27}$$

- *Entradas:* $\check{A}$.
- *Salidas:* $\check{A}^{-1}$, $\det \check{A}$.
- Matriz $\check{b}$ es inicializada a la matriz identidad.
- Fija el factor de escala, $s_i = \max_j(|\, A_{i,j}\,|)$ para cada fila.
- Itera sobre las filas $k = 1, \ldots, (N-1)$.
  - Selecciona la columna a pivotear a partir de $\max_j(|\, A_{i,j}\,|)/s_i$.
  - Realiza pivoteos usando la lista de índice de columnas.
  - Realiza la eliminación hacia adelante.
- Calcula el determinante $\det\check{A}$, como producto de los elementos de la diagonal.
- Realiza la sustitución hacia atrás.

Tabla 12.2: Bosquejo de la función `inv`, la cual calcula el inverso de una matriz y su determinante.

Después de calcular los $\vec{x}$, construimos $\check{A}^{-1}$ como

$$\check{A}^{-1}\,\left[\hat{x}_1\;\hat{x}_2\;\ldots\;\hat{x}_N\right] = \begin{bmatrix} (\vec{x}_1)_1 & (\vec{x}_2)_1 & \ldots & (\vec{x}_N)_1 \\ (\vec{x}_1)_2 & (\vec{x}_2)_2 & \ldots & (\vec{x}_N)_2 \\ \vdots & \vdots & \ddots & \vdots \\ (\vec{x}_1)_N & (\vec{x}_2)_N & \ldots & (\vec{x}_N)_N \end{bmatrix} \tag{12.28}$$

donde$(\vec{x}_i)_j$ es el elemento j de $\vec{x}_i$.

La tabla 12.2 muestra la función `inv` para calcular la inversa de una matriz. En Octave, `inv(A)` es una función internamente construida que regresa a la matriz inversa de `A` . Es posible resolver un sistema de ecuaciones lineales usando la matriz inversa, pero hacer esto es usualmente sobrepasarse. Una excepción podría ser el caso donde deseamos resolver un número de problemas similares en el cual la matriz $\check{A}$ es fija pero el vector $\vec{b}$ toma muchos valores diferentes. Finalmente, aquí hay una fórmula manual para tener presente: la inversa de una matriz 2×2 es

$$\check{A}^{-1} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix} . \tag{12.29}$$

Para matrices mayores las fórmulas rápidamente llegan a ser muy desordenadas.

12.2.2 Matrices singulares y patológicas.

Antes que regresemos a la física, discutamos otro posible riesgo en resolver sistemas de ecuaciones lineales. Consideremos las ecuaciones

$$\begin{array}{rcrcl} x_1 & + & x_2 & = & 1 \\ 2x_1 & + & 2x_2 & = & 2 \end{array} . \tag{12.30}$$

Note que realmente no tenemos dos ecuaciones independientes, ya que la segunda es sólo el doble de la primera. Estas ecuaciones no tienen una solución única. Si tratamos de hacer una eliminación hacia adelante, obtenemos

$$\begin{array}{rcrcl} x_1 & + & x_2 & = & 1 \\ & & 0 & = & 0 \end{array} , \tag{12.31}$$

y no se puede hacer nada más.

Otra manera de mirar este problema es ver que la matriz

$$\check{A} = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \tag{12.32}$$

no tiene inversa. Una matriz sin inversa se dice que es *singular*. Una matriz singular también tiene un determinante nulo. Las matrices singulares no siempre son evidentes. ¿Podría adivinar si esta matriz

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

es singular?

Aquí está lo que pasa en Octave cuando tratamos de resolver (12.30)

```
octave:1> A=[1 1; 2 2];
octave:2> b=[1; 2];
octave:3> x=A\b;
warning: matrix singular to machine precision, rcond = 0
```

Dependiendo de su compilador, la rutina `inv` de C++ probablemente retorne infinito o arroje un mensage de error. En algunos casos la rutina calcula una inversa para una matriz singular, pero naturalmente la respuesta es espuria.

Algunas veces la matriz no es singular pero está tan cerca de serlo que los errores de redondeo podrían "empujarla al abismo". Un ejemplo trivial sería

$$\begin{bmatrix} 1+\epsilon & 1 \\ 2 & 2 \end{bmatrix} , \tag{12.33}$$

donde ϵ es un número muy pequeño. A una matriz se le dice *patológica* cuando está muy cerca de ser singular.

Si usted sospecha que está tratrando con una matriz patológica cuando resuelve $\check{A}\vec{x} = \vec{b}$, entonces calcule el error absoluto, $\left| \check{A}\vec{x} - \vec{b} \right| / \left| \vec{b} \right|$ para comprobar si $\vec{x}$ es una solución precisa.

Formalmente, la condición numérica está definida como la "distancia" normalizada entre una matriz y la matriz singular más cercana[5]. Hay una variedad de maneras para definir la distancia, dependiendo del tipo de norma usada. La función de Octave `cond(A)` regresa la condición numérica de una matriz `A`. Una matriz con una gran condición numérica es patológica. Un pequeño determinante puede algunas veces advertirnos que la matriz podría ser patológica, pero la condición numérica es el criterio real.[6]

12.2.3 Osciladores armónicos acoplados.

En el comienzo de este capítulo, discutimos el problema de encontrar los estados estacionarios de ecuaciones diferenciales ordinarias. Un ejemplo canónico de un sistema con interacciones lineales es el caso de osciladores armónicos acoplados. Consideremos el sistem mostrado en la figura 12.1; las constantes de resorte son $k_1, \ldots, k_4$. La posición de los bloques, relativas a la pared izquierda, son x_1, x_2 y x_3. La distancia entre las paredes es L_P, y las longitudes naturales de los resortes son $L_1, \ldots, L_4$. Los bloques son de ancho despreciable.

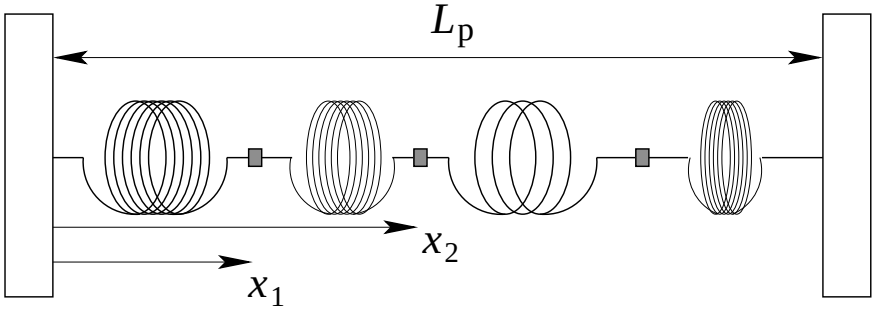


Figura 12.1: Sistema de bloques acoplados por resortes anclados entre paredes.

La ecuación de movimiento para el bloque i es

$$\frac{dx_i}{dt} = v_i \ , \qquad \frac{dv_i}{dt} = \frac{1}{m_i} F_i \ , \tag{12.34}$$

donde F_i es la fuerza neta sobre el bloque i. En el estado estacionario, las velocidades v_i, son nulas y las fuerzas netas, F_i son cero. Esto es justo el caso de equilibrio estático. Nuestro trabajo ahora es encontrar las posiciones en reposo de las masas. Explícitamente las fuerzas netas son

$$\begin{aligned} F_1 &= -k_1(x_1 - L_1) + k_2(x_2 - x_1 - L_2) \\ F_2 &= -k_2(x_2 - x_1 - L_2) + k_3(x_3 - x_2 - L_3) \\ F_3 &= -k_3(x_3 - x_2 - L_3) + k_4(L_p - x_3 - L_4) \end{aligned} \tag{12.35}$$

[5]S. Conte and C. de Boor, *Elementary Numerical Analysis* (New York: McGraw-Hill, 1980).

[6]G.H. Golub and C.F. van Loan, *Matrix computation* 2d ed. (Baltimore Johns Hopkins University Press, 1989).

o en forma matricial,

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} = \begin{bmatrix} -k_1 - k_2 & k_2 & 0 \\ k_2 & -k_2 - k_3 & k_3 \\ 0 & k_3 & k_3 - k_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} - \begin{bmatrix} -k_1 L_1 + k_2 L_2 \\ -k_2 L_2 + k_3 L_3 \\ -k_3 L_3 + k_4 (L_4 - L_p) \end{bmatrix} . \tag{12.36}$$

Por conveniencia, abreviaremos la ecuación de arriba como $\vec{F} = \check{K}\vec{x} - \vec{b}$. En forma matricial las simetrías son claras, y vemos que no sería difícil extender el problema a un sistema más grande de osciladores acoplados. En el estado de equilibrio estático las fuerzas netas son iguales a cero, así obtenemos el resto de las posiciones de las masas resolviendo $\check{K}\vec{x} - \vec{b}$.

12.3 Sistemas de ecuaciones no lineales.

12.3.1 Método de Newton en una variable.

Ahora que sabemos cómo resolver sistemas de ecuaciones lineales, procederemos al caso más general (y más desafiante) de resolver sistemas de ecuaciones no lineales. Este problema es difícil, tanto que consideraremos primero el caso de una variable. Deseamos resolver para x^* tal que

$$f(x^*) = 0 . \tag{12.37}$$

donde $f(x^*)$ es ahora una función general. Hay un número de métodos disponibles para encontrar raices en una variable. Quizás usted conozca algunos métodos como el de bisección o de la secante u otros algoritmos. También hay algoritmos especializados para cuando $f(x)$ es un polinomio (*e.g.*, función de `roots` en Octave). En vez de utilizar todos estos esquemas, nos concentraremos en el más simple y útil de los métodos para el caso general de N variables.

El *método de Newton*[7] está basado en la expansión de Taylor de $f(x)$ entorno a la raíz x^*. Supongamos que hacemos una estimación acerca de la localización de la raíz; llamamos a esta estimación x_1. Nuestro error podría ser escrito como $\delta x = x_1 - x^*$ o $x^* = x_1 - \delta x$. Escribiendo la expansión de Taylor de $f(x^*)$,

$$f(x^*) = f(x_1 - \delta x) = f(x_1) - \delta x \frac{df(x_1)}{dx} + \mathcal{O}(\delta x^2) . \tag{12.38}$$

Note que si x^* es una raíz, $f(x^*) = 0$, así podemos resolver para δx

$$\delta x = \frac{f(x_1)}{f'(x_1)} + \mathcal{O}(\delta x^2) . \tag{12.39}$$

Despreciamos el término $\mathcal{O}(\delta x^2)$ (este será nuestro error de truncamiento) y usamos la expresión resultante de δx para corregir nuestra estimación inicial. La nueva estimación es

$$x_2 = x_1 - \delta x = x_1 - \frac{f(x_1)}{f'(x_1)} . \tag{12.40}$$

Este procedimiento podría ser iterado como

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} , \tag{12.41}$$

para mejorar nuestra estimación hasta obtener la precisión deseada.

[7] F.S. Acton, *Numerical Methods that Work* (New York: Harper&Row, 1970).

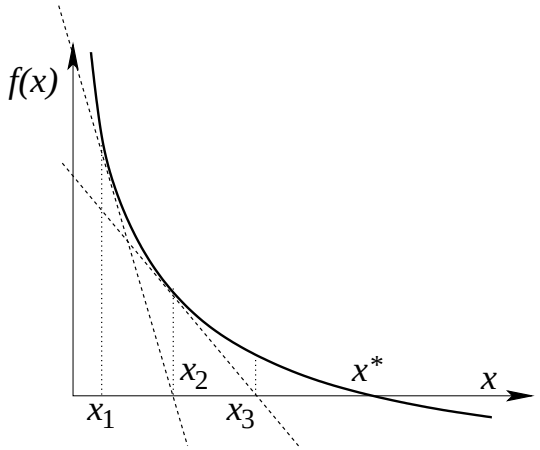


Figura 12.2: Representación gráfica del método de Newton.

El procedimiento iterativo descrito más arriba puede entenderse mejor gráficamente (figura 12.2). Note que en cada paso usamos la derivada de $f(x)$ para dibujar la línea tangente a la función. Donde esta línea tangente intersecta el eje x es nuestra siguiente estimación de la raíz. Efectivamente, estamos linealizando $f(x)$ y resolviendo el problema lineal. Si la función es bien comportada, pronto será aproximadamente lineal en alguna vecindad cerca de la raíz x^*.

Una pocas notas acerca del método de Newton: primero, cuando converge, encuentra una raíz muy rápidamente; pero, desafortunadamente, algunas veces diverge (*e.g.*, $f'(x_n) \approx 0$) y falla. Para funciones bien comportadas, este método es garantizado que converger si estamos suficientemente cerca de la raíz. Por esta razón algunas veces está combinada con un algoritmo más lento pero que asegura encontrar la raíz, tal como la bisección. Segundo, si hay varias raices, la raíz a la cual el método converge depende de la estimación inicial (que podría no ser la raíz deseada). Hay procedimientos (*e.g.*, "*deflation*") para encontrar múltiples raices usando el método de Newton. Finalmente, el método es más lento cuando se encuentran raices tangentes, tales como para $f(x) = x^2$.

12.3.2 Método de Newton multivariable.

No es difícil generalizar el método de Newton a problemas de N variables. Ahora nuestra incógnita $\vec{x} = [x_1\ ,\ x_2\ ,\ \ldots\ ,\ x_N]$ es un vector fila, y deseamos encontrar los ceros (raices) de la función vector fila

$$\vec{f}(\vec{x}) = [f_1(\vec{x})\ ,\ f_2(\vec{x})\ ,\ \ldots\ ,\ f_N(\vec{x})]\ . \tag{12.42}$$

De nuevo, hacemos una estimación inicial para ubicar la raíz, llamamos a esta estimación $\vec{x}_1$. Nuestro error podría ser esrito como $\delta\vec{x} = \vec{x}_1 - \vec{x}^*$ o $\vec{x}^* = \vec{x}_1 - \delta\vec{x}$. Usando la expansión de Taylor de $\vec{f}(\vec{x}_1)$,

$$\vec{f}(\vec{x}^*) = \vec{f}(\vec{x}_1 - \delta\vec{x}) = \vec{f}(\vec{x}_1) - \delta\vec{x}\,\check{D}(\vec{x}_1) + \mathcal{O}(\delta\vec{x}^2)\ , \tag{12.43}$$

donde $\check{D}$ es el Jacobiano, definido como

$$D_{ij}(\vec{x}) = \frac{\partial f_j(\vec{x})}{\partial x_i} \ . \tag{12.44}$$

Ya que $\vec{x}^*$ es la raíz, $\vec{f}(\vec{x}^*) = 0$, como antes podríamos resolverla para $\delta\vec{x}$. Despreciando el término de error, podemos escribir la ecuación (12.43) como

$$\vec{f}(\vec{x}_1) = \delta\vec{x}\,\check{D}(\vec{x}_1) \ , \tag{12.45}$$

o

$$\delta\vec{x} = \vec{f}(\vec{x}_1)\,\check{D}^{-1}(\vec{x}_1) \ . \tag{12.46}$$

Nuestra nueva estimación es

$$\vec{x}_2 = \vec{x}_1 - \delta\vec{x} = \vec{x}_1 - \vec{f}(\vec{x}_1)\,\check{D}^{-1}(\vec{x}_1) \ . \tag{12.47}$$

Este procedimiento puede ser iterado para mejorar nuestra estimación hasta que sea obtenida la precisión deseada. Ya que $\check{D}$ cambia en cada iteración, podría ser desgastador calcular su inversa. En cambio, usamos la eliminación Gaussiana sobre la ecuación (12.45) para resolver para $\delta\vec{x}$, y calcular la nueva estimación como $\vec{x}_2 = \vec{x}_1 - \delta\vec{x}$.

12.3.3 Programa del método de Newton.

Para demostrar el método de Newton, lo usaremos para calcular estados estacionarios del modelo de Lorenz.

A principios de la década de 1960 el meteorólogo del MIT, Ed Lorenz, encontró que el clima es intrínsecamente impredecible, no por su complejidad, sino por la naturaleza no lineal de las ecuaciones que lo gobiernan. Lorenz formuló un modelo simple del clima global, reduciendo el problema a un sistema de ecuaciones diferenciales ordinarias de 12 variables. Él observó que este sistema tenía un comportamiento aperiódico y que era extremadamente sensible a las condiciones iniciales. Estudiemos un modelo de Lorenz simplificado a tres variables

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x) \ , \\ \frac{dy}{dt} &= rx - y - xz \ , \\ \frac{dz}{dt} &= xy - bz \ , \end{aligned} \tag{12.48}$$

donde σ, r y b son contantes positivas. Estas ecuaciones simples fueron originalmente desarrolladas como un modelo para un fluido con convección. La variable x mide la razón de convección, y y z miden los gradientes de temperaturas horizontales y verticales. Los parámetros σ y b dependen de las propiedades del fluido y de la geometría del contenedor; comúnmente, los valores $\sigma = 10$ y $b = 8/3$ son usados. El parámetro r es proporcional al gradiente de temperatura aplicado.

- Conjunto inicial de estimaciones $\vec{x}_1$ y parámetros λ_k.
- Itera sobre el número de pasos deseados.
  - Evalúa la función $\vec{f}(\vec{x}_n; \lambda_k)$ y su Jacobiano $\check{D}$.
  - Encuentra $\delta\vec{x}$ por la eliminación Gaussiana [ver (12.45)].
  - Actualiza la estimación para la raíz usando $\vec{x}_{n+1} = \vec{x}_n - \delta\vec{x}$.
- Imprime la estimación final de la raíz.

Tabla 12.3: Descripción del programa `newtn`, el cual encuentra una raíz para un conjunto de ecuaciones.

Un programa que encuentra las raices de este sistema de ecuaciones usando el método de Newton, al que llamadomos `newtn`, es esbozado en la tabla 12.3. Este programa llama la función `fnewt` (tabla 12.4), la cual dado un $\vec{x} = [x\ ,\ y\ ,\ z]$ regresa

$$\vec{f} = \begin{bmatrix} \sigma(y-x) \\ rx - y - xz \\ xy - bz \end{bmatrix} \quad , \quad \check{D} = \begin{bmatrix} -\sigma & \sigma & 0 \\ r-z & -1 & -x \\ y & x & -b \end{bmatrix} \quad . \tag{12.49}$$

Para $r = 28$, $\sigma = 10$ y $b = 8/3$, las tres raíces son $[x, y, z] = [0, 0, 0]$, $[6\sqrt{2}, 6\sqrt{2}, 27]$ y $[-6\sqrt{2}, -6\sqrt{2}, 27]$.

Un ejemplo de la salida de `newtn` está dada más abajo; note que el programa obtiene la raíz $[6\sqrt{2}, 6\sqrt{2}, 27]$.

```
octave:1> newtn
newtn is the file: /home/jrogan/cursosNEW/IntroduccionMFM.apuntes/programas/newtn.m

 newtn - Programa para resolver un sistema de ecuaciones no lineales
 usando el metodo de Newton. Las ecuaciones estan definidas en fnewt

Entre la estimacion inicial (vector columna): [50; 50; 50]
Entre los parametros a: [28 10 8/3]
Despues de  10 iteraciones la raiz es
     8.4853   8.4853  27.0000
```

Otras condiciones iniciales convergen a otras raices. Por ejemplo, comenzando por $[2, 2, 2]$, convergemos a la raíz $[0, 0, 0]$; si comenzamos en $[5, 5, 5]$, converge a $[-6\sqrt{2}, -6\sqrt{2}, 27]$. Comenzando por el punto $[4, 4, 15]$, el método converge a una raíz sólo después de hacer una excursión increíblemente distante del origen.

- *Entradas:* $\vec{x} = [x,\, y,\, z]$, $\lambda = [r,\, \sigma,\, b]$.
- *Salidas:* $\vec{f}, \check{D}$.
- Evalúa $\vec{f}$ por el modelo de Lorenz [(12.49)].
- Evalúa el Jacobiano $\check{D}$ para el modelo de Lorenz.

Tabla 12.4: Descripción de la función `fnewt`, la cual es usada por `newtn` para encontrar los estados estables de la ecuación de Lorenz.

12.3.4 Continuación.

La primera dificultad con el método de Newton es la necesidad de dar una buena estimación inicial para la raíz. A menudo nuestro problema es de la forma

$$\vec{f}(\vec{x}^*; \lambda) = 0 \ , \tag{12.50}$$

donde λ es algún parámetro en el problema. Supongamos que conocemos una raíz $\vec{x}^*_0$ para el valor λ_0 pero necesitamos encontrar una raíz para un valor diferente de λ_a. Intuitivamente, si $\lambda_a \approx \lambda_0$, entonces $\vec{x}^*_0$ debería ser una buena estimación inicial para encontrar $\vec{x}^*_a$ usando el método de Newton. Pero si $\lambda_a \not\approx \lambda_0$, ¿hay alguna manera para hacer uso de nuestra raíz conocida?

La respuesta es sí y la técnica es conocida como *continuación*. La idea es acercarse poco a poco a λ_a definiendo la siguiente secuencia de λ :

$$\lambda_i = \lambda_0 + (\lambda_a - \lambda_0)\frac{i}{N} \ , \tag{12.51}$$

para $i = 1, \ldots\ldots., N$. Usando el método de Newton, resolvemos $f(\vec{x}^*_1; \lambda_1) = 0$ con $\vec{x}^*_0$ como la estimación inicial. Si N es suficientemente grande, entonces $\lambda_1 \approx \lambda_0$ y el método podría converger rápidamente. Luego usamos $\vec{x}^*_1$ como una estimación inicial para resolver $f(\vec{x}^*_2; \lambda_2) = 0$; la secuencia continúa hasta alcanzar nuestro valor deseado en λ_a. La técnica de continuación tiene un beneficio adicional que podemos usar cuando estamos interesados en conocer no sólo una raíz simple, sino conocer como $\vec{x}^*$ varía con λ.

12.4 Listados del programa.

12.4.1 Definición de la clase `Matrix`.

```
//
// Clase de Matrices doubles
//
#ifndef _Matrix_h
#define _Matrix_h 1

#include <iostream.h>
#include <String.h>

class Matrix {
 private:
  int nFilP ;
  int nColP ;
  int dimP ;
  double * dataP ;
  void copy(const Matrix &) ;
 public:
  Matrix() ;
  Matrix( int, int = 1, double = 0.0e0) ;
  Matrix( const Matrix &) ;
  ~Matrix() ;
  Matrix & operator = (const Matrix &) ;
  double & operator () (int, int =0) ;
  int nRow () const;
  int nCol () const ;
  void set(double) ;
  double maximoF(int) const ;
  void pivot(int,int) ;
  void multi(double, int, int ) ;
};

ostream & operator << (ostream &, Matrix );

void error ( int, String="Error") ;

Matrix operator * (Matrix, Matrix) ;
Matrix operator + (Matrix, Matrix) ;
Matrix operator - (Matrix, Matrix) ;

#endif
```

12.4.2 Implementación de la clase `Matrix`.

```
//
// Implementacion de la clase Matrix

#include "Matrix.h"
#include <assert.h>
#include <stdlib.h>

// Funcion privada de copia
void Matrix::copy(const Matrix & mat) {
  nFilP=mat.nFilP ;
  nColP=mat.nColP ;
  dimP = mat.dimP ;
  dataP = new double[dimP] ;
  for(int i =0 ; i< dimP; i++) dataP[i]=mat.dataP[i] ;
}

// Constructor default Crea una matriz de 1 por 1 fijando su valor a cero
Matrix::Matrix() {
  Matrix(1) ;
}

// Constructor standar Crea una matriz de N por M
// y fija sus valores a cero o a un valor ingresado
Matrix::Matrix(int nF, int nC, double v) {
  error(nF>0 && nC >0, "Una de las dimensiones no es positiva" ) ;
  nFilP= nF ;
  nColP = nC ;
  dimP = nFilP*nColP ;
  dataP = new double[dimP] ;
  assert(dataP != 0) ;          // Habia memoria para reservar
  for (int i=0; i < dimP; i++) dataP[i]=v ;
}

// Constructor de copia
Matrix::Matrix( const Matrix & mat) {
  this->copy(mat) ;
  // this permite acceder a la misma vartiable como un todo
  // *this representa la variable, y this un puntero a ella
  // -> nos permite elegir miembros de la clase pero cuando tenemos el puntero
  // a la variable y no su valor
}

// Destructor
```

```
Matrix::~Matrix() {
  delete [] dataP ;             // Libera la memoria
}

// Operador Asignacion, sobrecarga del = para que funcione con Matrix
Matrix & Matrix::operator= (const Matrix & mat) {
  if( this == &mat ) return *this ; // Si ambos lados son iguales no hace nada
  delete [] dataP ;             //  borra la data del lado izquierdo
  this-> copy(mat) ;
  return * this ;
}

// Funciones simples que retornan el numero de filas y de columnas
// el const al final indica que no modifican ningun miembro de la clasa
int Matrix::nRow() const {return nFilP; }
int Matrix::nCol() const {return nColP;}

// operador Parentesis
// Permite accesar los valores de una matriz via el par (i,j)
// Ejemplo a(1,1)=2*b(2,3)
// Si no se especifica la columna la toma igual a 0
double & Matrix::operator() (int indF, int indC) {
  error(indF>-1 && indF< nFilP, "Indice de fila fuera de los limites") ;
  error(indC>-1 && indC<nColP, "Indice de columna fuera de los limites" ) ;
  return dataP[indC+nColP*indF] ;
}

void Matrix::set(double value) {
  for(int i=0; i<dimP; i++) dataP[i]=value ;
}

double Matrix::maximoF(int indF) const {
  error(indF>-1 && indF< nFilP, "Indice de fila fuera  en maximoF") ;
  double max= dataP[nColP*indF];
  for(int i=nColP*indF; i<nColP*(indF+1); i++ ) {
    max = (max > fabs(dataP[i]) ) ? max : fabs(dataP[i]) ;
  }
  return max ;
}

void Matrix::pivot(int indF1, int indF2) {
  error(indF1>-1 && indF1< nFilP, "Indice de fila fuera en Pivot") ;
  error(indF1>-1 && indF1< nFilP, "Indice de columna fuera en Pivot") ;
```

```
  if(indF1==indF2) return ;
  double paso ;
  int ind1, ind2 ;
  for (int i = 0; i < nColP; i++ ) {
    ind1 = nColP*indF1+i ;
    ind2 = nColP*indF2+i ;
    paso = dataP[ind1] ;
    dataP[ind1] = dataP[ind2] ;
    dataP[ind2] = paso ;
  }
}

void Matrix::multi(double fact, int indF1, int indF2 ) {
  int ind1, ind2 ;
  for (int i = 0; i < nColP; i++ ) {
    ind1 = nColP*indF1+i ;
    ind2 = nColP*indF2+i ;
    dataP[ind2] += fact*dataP[ind1] ;
  }
}

//
// IOSTREAM <<
//
ostream & operator << (ostream & os, Matrix mat ) {
  for(int indF=0; indF < mat.nRow(); indF++) {
        os << "| " ;
    for(int indC=0; indC<mat.nCol(); indC++) {
      os << mat(indF,indC) << " ";
    }
    os << "|"<<endl ;
  }
  return os ;
}

void error ( int i, String s) {
  if(i) return ;
  cout << s<<"\a" <<endl ;
  exit(-1);
}

Matrix  operator * (Matrix A , Matrix B) {
  int nColA = A.nCol() ;
  int nColB = B.nCol() ;
  int nFilA = A.nRow() ;
```

```
  int nFilB = B.nRow() ;
  error(nColA == nFilB, "No se pueden multiplicar por sus dimensiones") ;
  Matrix R(nFilA   , nColB ) ;
  for (int indF = 0; indF<nFilB; indF++){
    for (int indC=0; indC<nColA; indC++){
      double sum = 0.0 ;
      for (int k=0; k < nColA; k++ ) sum += A(indF, k) * B(k, indC) ;
      R(indF, indC) = sum  ;
    }
  }
  return R ;
}

Matrix  operator + (Matrix A , Matrix B) {
  int nColA = A.nCol() ;
  int nColB = B.nCol() ;
  int nFilA = A.nRow() ;
  int nFilB = B.nRow() ;
  error(nFilA == nFilB && nColA== nColB, "tienen dimensiones distintas") ;
  Matrix R(nFilA   , nColA ) ;
  for (int indF = 0; indF<nFilB; indF++){
    for (int indC=0; indC<nColA; indC++){
      R(indF, indC) = A( indF, indC) + B(indF, indC);
    }
  }
  return R ;
}

Matrix  operator - (Matrix A , Matrix B) {
  int nColA = A.nCol() ;
  int nColB = B.nCol() ;
  int nFilA = A.nRow() ;
  int nFilB = B.nRow() ;
  error(nFilA == nFilB && nColA== nColB, "tienen dimensiones distintas") ;
  Matrix R(nFilA   , nColA ) ;
  for (int indF = 0; indF<nFilB; indF++){
    for (int indC=0; indC<nColA; indC++){
      R(indF, indC) = A( indF, indC) - B(indF, indC);
    }
  }
  return R ;
}
```

12.4.3 Función de eliminación Gaussiana ge.

```
//
// Eliminacion Gaussiana.
//

#include "NumMeth.h"
#include "Matrix.h"

double ge( Matrix A, Matrix  b, Matrix & x)
{
  int nFil = A.nRow() ;
  int nCol = A.nCol() ;
  Matrix si (nFil) ;

  for ( int indF= 0; indF < nFil ; indF++)  si(indF)= A.maximoF(indF) ;
  double determinante=1.0e0 ;
  for (int indC= 0; indC< nCol-1; indC++) {
    // pivoteo
    double max = A(indC,indC)/si(indC) ;
    int indicePivot = indC ;
    for (int i = indC+1; i < nCol ; i++){
      if( A(i, indC)/si(i) > max ) {
        max = A(i,indC)/si(i) ;
        indicePivot = i ;
      }
    }
    if(indicePivot != indC)  {
      A.pivot(indC, indicePivot) ;
      b.pivot(indC, indicePivot) ;
      determinante *= -1.0e0 ;
    }
    double Ad= A(indC,indC) ;
    for (int indF=indC+1; indF< nFil; indF++ ) {
      double factor =  - A(indF, indC)/Ad ;
      A.multi(factor, indC, indF);
      b.multi(factor, indC, indF);
    }
  }
  for (int indF= nFil-1; indF >=0; indF--) {
    double sum =0.0e0 ;
    for (int i = indF+1; i < nCol; i++) sum += A(indF, i)*x(i) ;
    x(indF) =  b(indF)/A(indF, indF) - sum /A(indF, indF);
  }
  double determ = 1.0e0 ;
```

```
  for (int i = 0; i < nCol; i++) determ *= A(i,i) ;

  return determinante*determ  ;
}
```

12.4.4 Función para inversión de matrices inv.

```
//
// Invierte una matriz
//

#include "NumMeth.h"
#include "Matrix.h"

double inv( Matrix A, Matrix & invA) {
  int nFil = A.nRow() ;
  int nCol = A.nCol() ;
  error(nFil==nCol, "Matriz no es cuadrada");
  Matrix e(nFil) ;
  Matrix x(nFil) ;
  double deter = 0.0e0 ;
  invA.set(0.0e0) ;
  for(int indC=0; indC < nCol; indC++) {
    e.set(0.0e0) ;
    e(indC) = 1.0e0;
    deter = ge( A, e, x) ;
    error(fabs(deter)>1.0e-10, " Determinante Singular" );
    for (int indF=0; indF< nFil; indF++)   invA(indF, indC) = x(indF) ;
  }
  return deter ;
}
```

12.4.5 Programa newtn en Octave.

```
% newtn - Programa para resolver un sistema de ecuaciones no lineales
% usando el metodo de Newton. Las ecuaciones estan definidas en fnewt
suppress_verbose_help_message=1;
clear all; help  newtn;

x0=input('Entre la estimacion inicial (vector columna): ');
x=x0;
a=input('Entre los parametros a: ');

nStep =10;
```

```
for iStep=1:nStep

[f D] = fnewt(x,a) ;
dx=D\f ;
x=x-dx;
end
fprintf('Despues de  %g iteraciones la raiz es \n', nStep);
disp(x');
```

Función `fnewt` en Octave.

```
function [f,D] = fnewt(x,a)

f(1)=a(2)*(x(2)-x(1));
f(2)= a(1)*x(1)-x(2)-x(1)*x(3);
f(3)=x(1)*x(2)-a(3)*x(3) ;

D(1,1)=-a(2);
D(2,1)=a(1)-x(3);
D(3,1)=x(2);
D(1,2)=a(2);
D(2,2)=-1;
D(3,2)=x(1);
D(1,3)=0;
D(2,3)=-x(1);
D(3,3)=-a(3);

return
```

12.4.6 Programa newtn en c++.

```
#include <iostream.h>
#include "NumMeth.h"
#include "Matrix.h"

void fnewt( Matrix & f, Matrix & D, Matrix xN, Matrix lambda )
{
  double r= lambda(0) ;
  double s= lambda(1) ;
  double b=lambda(2) ;
  double x=xN(0) ;
  double y=xN(1) ;
  double z=xN(2) ;
  f(0) = s*(y-x) ;
  f(1) = r*x-y-x*z ;
  f(2) = x*y-b*z ;
  D(0,0) = -s ;
  D(1,0) = r-z ;
  D(2,0) = y ;
  D(0,1) = s ;
  D(1,1)= -1.0e0 ;
  D(2,1) = x ;
  D(0,2)=0.0e0 ;
  D(1,2)= -x ;
  D(2,2) =-b ;
}

double ge( Matrix A, Matrix  b, Matrix & x)
{
  int nFil = A.nRow() ;
  int nCol = A.nCol() ;
  Matrix si (nFil) ;

  for ( int indF= 0; indF < nFil ; indF++)  si(indF)= A.maximoF(indF) ;
  double determinante=1.0e0 ;
  for (int indC= 0; indC< nCol-1; indC++) {
    // pivoteo
    double max = A(indC,indC)/si(indC) ;
    int indicePivot = indC ;
    for (int i = indC+1; i < nCol ; i++){
      if( A(i, indC)/si(i) > max ) {
        max = A(i,indC)/si(i) ;
        indicePivot = i ;
      }
```

```
    }
    if(indicePivot != indC)  {
      A.pivot(indC, indicePivot) ;
      b.pivot(indC, indicePivot) ;
      determinante *= -1.0e0 ;
    }
    double Ad= A(indC,indC) ;
    for (int indF=indC+1; indF< nFil; indF++ ) {
      double factor =  - A(indF, indC)/Ad ;
      A.multi(factor, indC, indF);
      b.multi(factor, indC, indF);
    }
  }
  for (int indF= nFil-1; indF >=0; indF--) {
    double sum =0.0e0 ;
    for (int i = indF+1; i < nCol; i++) sum += A(indF, i)*x(i) ;
    x(indF) =  b(indF)/A(indF, indF) - sum /A(indF, indF);
  }
  double determ = 1.0e0 ;
  for (int i = 0; i < nCol; i++) determ *= A(i,i) ;

  return determinante*determ  ;
}


main()
{
  Matrix D(3,3) ;
  Matrix f(3) ;
  Matrix lambda(3), xN(3), xNp1(3), dx(3) ;

  cout <<"Ingrese r , Sigma, b : " ;
  cin >> lambda(0) >> lambda(1) >> lambda(2) ;
  cout << "Ingrese estimacion inicial : " ;
  cin>>   xN(0) >>  xN(1) >>   xN(2) ;

  int iteraciones ;
  cout << "Ingrese numero de iteraciones : " ;
  cin >> iteraciones ;

  for (int itera = 0; itera < iteraciones; itera ++ ) {
    f.set(0.0e0) ;
    D.set(0.0e0) ;
    dx.set(0.0e0) ;
    fnewt(f, D, xN, lambda ) ;
```

```
    ge(D, f, dx) ;
    xNp1=xN-dx ;
    xN=xNp1 ;
    cout << xN(0)<< " " << xN(1) << " " << xN(2) << endl ;
  }
  cout << xN <<endl ;
}
```

Capítulo 13

Análisis de datos.

versión final 2.0-19 agosto 2002[1]

A menudo se destaca que los sistemas físicos simulados sobre un computador son similares al trabajo experimental. La razón de esta analogía es hecha ya que la simulación computacional produce datos en muchas maneras similares a los experimentos de laboratorio. Sabemos que en el trabajo experimental uno a menudo necesita analizar los resultados y esto es lo mismo que con la simulación numérica. Este capítulo cubre parcialmente algunos tópicos en el análisis de datos.

13.1 Ajuste de curvas.

13.1.1 El calentamiento global.

En el presente, parece que predicciones sobre el tiempo de largo alcance y precisas nunca se llevarán a cabo. La razón es a causa de las ecuaciones gobernantes que son altamente no lineales y sus soluciones son extremadamente sensibles a las condiciones iniciales (ver el modelo de Lorenz). Por otra parte, las predicciones generales acerca del clima terrestre son posibles. Se pueden predecir si habrá o no condiciones de sequía en África en los próximos años, aunque no la cantidad de precipitaciones sobre un día en particular.

El calentamiento global es un importante tópico acaloradamente debatido en la investigación climática. El calentamiento es culpa de los gases de invernadero, tal como es el dióxido de carbono en la atmósfera. Estos gases calientan la Tierra porque son transparentes a la radiación de onda corta que llega desde el Sol, pero opacas a la radiación infrarroja desde la tierra. Científicos y legisladores todavía están debatiendo la amenaza del calentamiento global. Sin embargo, nadie se pregunta qué concentraciones de gases invernadero están en aumento. Específicamente, los niveles de dióxido de carbono han estado firmemente aumentados desde la revolución industrial. La figura 13.1 muestra el aumento en la concentración de dióxido de carbono durante los años ochenta, medidos en Mauna Loa, Hawaii.

El estudio del calentamiento global ha producido una vasta cantidad de datos, tanto mediciones prácticas, como datos de las simulaciones computacionales. En este capítulo estudiaremos algunas técnicas básicas para analizar y reducir tales conjuntos de datos. Por ejemplo, para los datos mostrados en la figura 13.1, ¿cuál es la razón estimada de crecimiento

[1]Este capítulo está basado en el quinto capítulo del libro: *Numerical Methods for Physics, second edition* de Alejandro L. Garcia, editorial Prentice Hall.

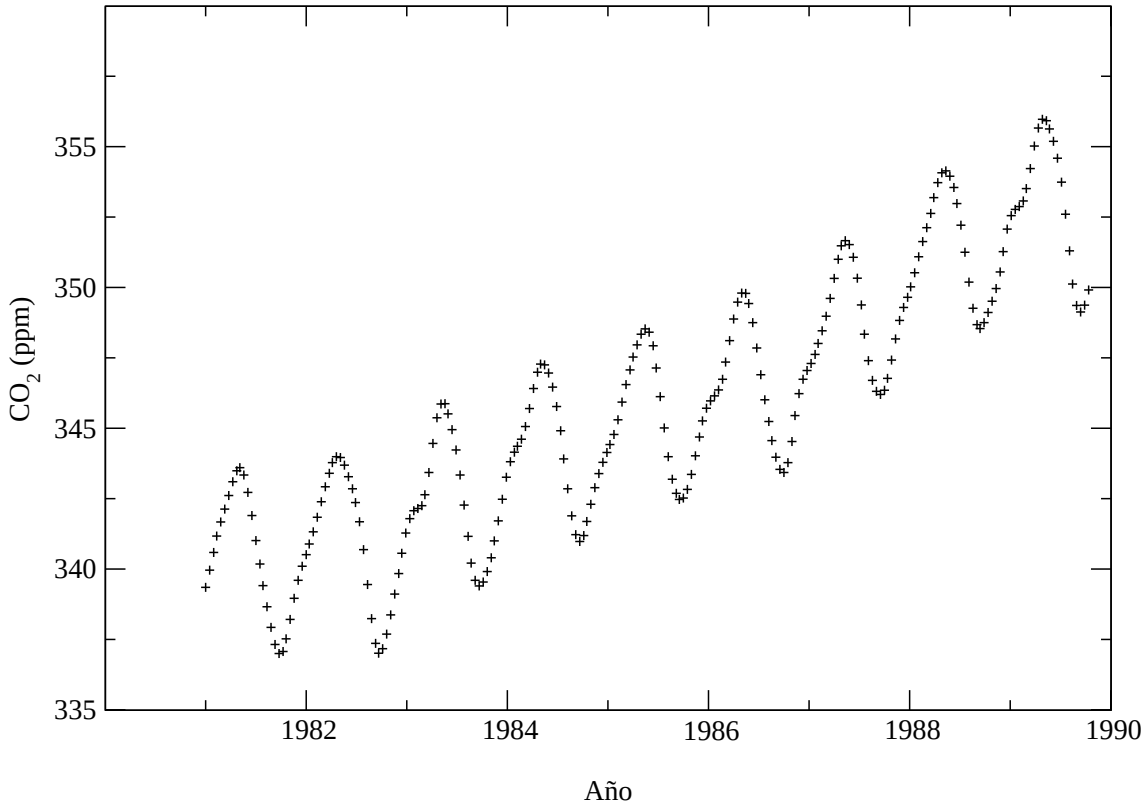


Figura 13.1: Dióxido de carbono (en partes por millón) medida en Mauna Loa, Hawai desde 1981 a 1990. Barra de error estimada es $\sigma_0 = 0.16$ [p.p.m.] .

de la concentración de CO_2 por año?. Esta es la primera pregunta que motiva nuestro estudio de ajustes de curvas.

13.1.2 Teoría general.

El tipo más simple de análisis de datos es ajustar una curva. Suponga que tenemos un conjunto de datos de N puntos (x_i, y_i). Deseamos ajustar estos datos a una función $Y(x; \{a_j\})$, donde $\{a_j\}$ es un conjunto de M parámetros ajustables. Nuestro objetivo es encontrar los valores de esos parámetros, para lo cual la función ajusta mejor los datos. Intuitivamemte, esperamos que si nuestro ajuste de la curva es bueno, un gráfico del conjunto de datos (x_i, y_i) y la función $Y(x; \{a_j\})$ mostrarán la curva pasando cerca de los puntos (ver figura 13.2). Podemos cuantificar esta sentencia midiendo la distancia entre un punto y la curva.

$$\Delta_i = Y(x_i; \{a_j\}) - y_i \ . \tag{13.1}$$

Nuestro criterio de ajuste para la curva será que la suma de los cuadrados de los errores

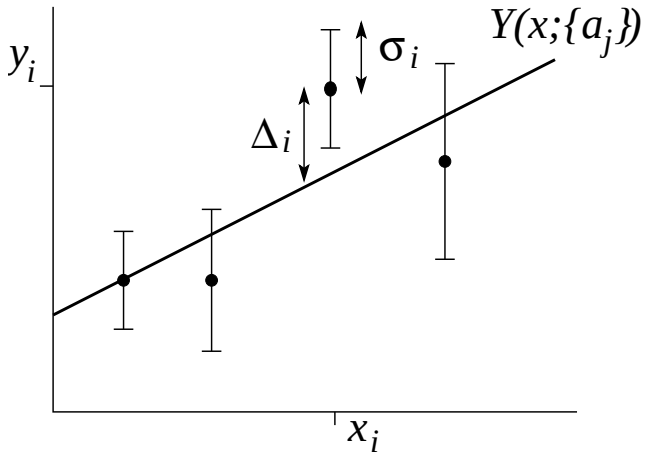


Figura 13.2: Ajuste de datos a una curva.

sea un mínimo; esto es, necesitamos encontrar $\{a_j\}$ que minimice la función

$$D(\{a_j\}) = \sum_{i=1}^{N} \Delta_i^2 = \sum_{i=1}^{N} [Y(x_i;\{aj\}) - y_i]^2 \ . \tag{13.2}$$

Esta técnica nos dará el *ajuste de los cuadrados mínimos*; esta no es la única manera de obtener un ajuste de la curva, pero es la más común. El método de los mínimos cuadrados fue el primero usado por Gauss para determinar las órbitas de los cometas a partir de los datos observados.

A menudo, nuestros datos tienen una barra de error estimada (o intervalo de seguridad), el cual escribimos como $y_i \pm \sigma_i$. En este caso podríamos modificar nuestro criterio de ajuste tanto como para dar un peso menor a los puntos con más error. En este espíritu definimos

$$\chi^2(\{a_j\}) = \sum_{i=1}^{N} \left(\frac{\Delta_i}{\sigma_i}\right)^2 = \sum_{i=1}^{N} \frac{[Y(x_i;\{a_j\}) - y_i]^2}{\sigma_i^2} \ . \tag{13.3}$$

La función chi-cuadrado es la función de ajuste más comúnmente usada porque si los errores son distribuidos gaussianamente, podemos hacer sentencias estadísticas concernientes a la virtud del ajuste.

Antes de continuar, debemos destacar que discutiremos brevemente la validación de la curva de ajuste. Usted puede ajustar cualquier curva para cualquier conjunto de datos, pero esto no significa que los resultados sean significativos. Para establecer una significancia tenemos que preguntarnos lo siguiente; ¿Cuál es la probabilidad que los datos, dado el error experimental asociado con cada uno de los puntos, sean descrito por la curva?. Desafortunadamente, las hipótesis de prueba ocupan una porción significativa de un curso estadístico y están fuera de los objetivos de esta libro.[2]

[2]W. Mendenhall, R.L. Scheaffer, and D.D. Wackerly, *Mathematical Statistics with Applications* (Boston: Duxbury Press, 1981).

13.1.3 Regresión lineal.

Primero consideremos ajustar el conjunto de datos con una línea recta,

$$Y(x;\{a_1,a_2\}) = a_1 + a_2 x \ . \tag{13.4}$$

Este tipo de ajuste de curva es también conocido como regresión lineal. Deseamos determinar a_1 y a_2 tal que

$$\chi^2(a_1,a_2) = \sum_{i=1}^{N} \frac{1}{\sigma_i^2}\left(a_1 + a_2 x_i - y_i\right)^2 \ , \tag{13.5}$$

es minimizada. El mínimo es encontrado diferenciando (13.5) y ajustando la derivada a cero:

$$\frac{\partial \chi^2}{\partial a_1} = 2\sum_{i=1}^{N} \frac{1}{\sigma_i^2}\left(a_1 + a_2 x_i - y_i\right) = 0 \ , \tag{13.6}$$

$$\frac{\partial \chi^2}{\partial a_2} = 2\sum_{i=1}^{N} \frac{1}{\sigma_i^2}\left(a_1 + a_2 x_i - y_i\right) x_i = 0 \ , \tag{13.7}$$

o

$$a_1 S + a_2 \Sigma x - \Sigma y = 0 \tag{13.8}$$

$$a_1 \Sigma x + a_2 \Sigma x^2 - \Sigma xy = 0 \ , \tag{13.9}$$

donde

$$\begin{aligned} S \equiv \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \ , \quad \Sigma x \equiv \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} \ , \quad \Sigma y \equiv \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} \ , \\ \Sigma x^2 \equiv \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} \ , \quad \Sigma xy \equiv \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2} \ . \end{aligned} \tag{13.10}$$

Puesto que las sumas pueden ser calculadas directamente de los datos, ellas son constantes conocidas. De este modo tenemos un conjunto lineal de dos ecuaciones simultáneas en las incógnitas a_1y a_2. Estas ecuaciones son fáciles de resolver:

$$a_1 = \frac{\Sigma y \Sigma x^2 - \Sigma x \Sigma xy}{S\Sigma x^2 - (\Sigma x)^2} \ , \qquad a_2 = \frac{S\Sigma xy - \Sigma x \Sigma y}{S\Sigma x^2 - (\Sigma x)^2} \ . \tag{13.11}$$

Note que si σ_i, esto es, si el error es el mismo para todos los datos, los σ se cancelan en las ecuaciones (13.11). En este caso, los parámetros, a_1 y a_2, son independientes de la barra de error. Es común que un conjunto de datos no incluya las barras de error asociadas. Todavía podríamos usar las ecuaciones (13.11) para ajustar los datos si tenemos que los σ_i son constantes (σ_i=1 siendo la elección más simple).

Lo siguiente es obtener una barra de error asociado, σ_{aj}, para el parámetro a_j de la curva de ajuste. Usando la ley de la propagación de errores,[3]

$$\sigma_{a_j}^2 = \sum_{i=1}^{N} \left(\frac{\partial a_j}{\partial y_i} \right) \sigma_i^2 \ , \tag{13.12}$$

e incertando las ecuaciones (13.11), después de un poco de álgebra obtenemos

$$\sigma_{a_1} = \sqrt{\frac{\Sigma x^2}{S\Sigma x^2 - (\Sigma x)^2}} \ , \qquad \sigma_{a_2} = \sqrt{\frac{S}{S\Sigma x^2 - (\Sigma x)^2}} \ . \tag{13.13}$$

Note que σ_{a_j} es independiente de y_i. Si las barras de error de los datos son constantes ($\sigma_i = \sigma_0$), el error en los parámetros es

$$\sigma_{a_1} = \frac{\sigma_0}{\sqrt{N}} \sqrt{\frac{\langle x^2 \rangle}{\langle x^2 \rangle - \langle x \rangle^2}} \ , \qquad \sigma_{a_2} = \frac{\sigma_0}{\sqrt{N}} \sqrt{\frac{1}{\langle x^2 \rangle - \langle x \rangle^2}} \ , \tag{13.14}$$

donde

$$\langle x \rangle = \frac{1}{N} \sum_{i=1}^{N} x_i \ , \quad \langle x^2 \rangle = \frac{1}{N} \sum_{i=1}^{N} x_i^2 \ . \tag{13.15}$$

Finalmente, si nuestro conjunto de datos no tiene un conjunto de barras de error aociadas, podríamos estimar σ_o a partir de la diferencia muestral de los datos,

$$\sigma_0^2 \approx s^2 = \frac{1}{N-2} \sum_{i=1}^{N} \left[y_i - (a_1 + a_2 x_i) \right]^2 \ , \tag{13.16}$$

donde s es al desviación estándar de la muestra. Note que esta varianza muestral está normalizada por $N-2$, puesto que hemos extraído dos parámetros a_1 y a_2, de los datos.

Muchos problemas de ajuste de curva no lineales, pueden ser transformados en problemas lineales por un simple cambio de variable. Por ejemplo,

$$Z(x; \{\alpha, \beta\}) = \alpha e^{\beta x} \ , \tag{13.17}$$

podría ser reescrita como las ecuaciones (13.4) usando el cambio de variable

$$\ln Z = Y \ , \quad \ln \alpha = a_1 \ , \quad \beta = a_2 \ . \tag{13.18}$$

Similarmente, para ajustar una potencia de la forma

$$Z(t; \{\alpha, \beta\}) = \alpha t^{\beta} \ , \tag{13.19}$$

usamos el cambio de variable

$$\ln Z = Y \ , \quad \ln t = x \ , \quad \ln \alpha = a_1 \ , \quad \beta = a_2 \ . \tag{13.20}$$

Estas transformaciones deberían ser familiares debido a que se usan para graficar datos en escala semi logarítmica o escalas log-log.

[3]P. Bevington, *Data Reduction an Error Analysis for the Physical Sciences* 2d ed. (New York: McGraw-Hill, 1992).

13.1.4 Ajuste general lineal de mínimos cuadrados.

El procedimiento de ajuste de mínimos cuadrados es fácil de generalizar para funciones de la forma

$$Y(x;\{a_j\}) = a_1 Y_1(x) + a_2 Y_2(x) + \ldots + a_M Y_M(x) = \sum_{j=1}^{M} a_j Y_j(x) \ . \tag{13.21}$$

Para encontrar los óptimos parámetros procedemos como antes encontrando el mínimo de χ^2,

$$\frac{\partial \chi^2}{\partial a_j} = \frac{\partial}{\partial a_j} \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left\{ \sum_{k=1}^{M} a_k Y_k(x) - y_i \right\}^2 = 0 \ , \tag{13.22}$$

o

$$\sum_{i=1}^{N} \frac{1}{\sigma_i^2} Y_j(x_i) \left\{ \sum_{k=1}^{M} a_k Y_k(x) - y_i \right\} = 0 \ , \tag{13.23}$$

por lo tanto

$$\sum_{i=1}^{N} \sum_{k=1}^{M} \frac{Y_j(x_i) Y_k(x_i)}{\sigma_i^2} a_k = \sum_{i=1}^{N} \frac{Y_j(x_i) y_i}{\sigma_i^2} \ , \tag{13.24}$$

para $j = 1, \ldots, M$. Este conjunto de ecuaciones es conocida como las *ecuaciones normales* del problema de los mínimos cuadrados.

Las ecuaciones normales son más fáciles de trabajar en forma matricial. Primero, definamos la matriz de diseño $\check{A}$, como

$$\check{A} = \begin{bmatrix} Y_1(x_i)/\sigma_1 & Y_2(x_i)/\sigma_1 & \ldots \\ Y_1(x_2)/\sigma_2 & Y_2(x_i)/\sigma_2 & \ldots \\ \vdots & \vdots & \ddots \end{bmatrix} \ , \qquad A_{ij} = \frac{Y_j(x_i)}{\sigma_i} \ . \tag{13.25}$$

Note que la matriz de diseño no depende de y_i, los valores de datos, pero depende de x_i, esto es, sobre el diseño del experimento (donde las mediciones son tomadas). Usando la matriz de diseño, podemos escribir (13.24) como

$$\sum_{i=1}^{N} \sum_{k=1}^{M} A_{ij} A_{ik} a_k = \sum_{i=1}^{N} A_{ij} \frac{y_i}{\sigma_i^2} \ , \tag{13.26}$$

o en forma matricial,

$$(\check{A}^T \check{A}) \vec{a} = \check{A}^T \vec{b} \ , \tag{13.27}$$

donde el vector $\vec{b}$ está definido como $b_i = y_i/\sigma_i$ y $\check{A}^T$ es la traspuesta de la matriz de diseño. Esta ecuación es fácil de resolver para el parámetro vector $\vec{a}$,

$$\vec{a} = (\check{A}^T \check{A})^{-1} \check{A}^T \vec{b} \ . \tag{13.28}$$

Usando este resultado y la ley de propagación de errores, en la ecuación (13.12), encontramos que el error estimado en el parámetro a_j es

$$\sigma_{aj} = \sqrt{C_{jj}} \tag{13.29}$$

donde $\check{C} = (\check{A}^T \check{A})^{-1}$.

Una aplición común de esta formulación general lineal de ajuste de mínimos cuadrados es el ajuste de los datos polinomiales,

$$Y(x; \{a_j\}) = a_1 + a_2 x + a_3 x^2 + \ldots + a_M x^{M-1} \ . \tag{13.30}$$

La matriz de diseño tiene elementos $A_{ij} = a_j x_I^{j-1}/\sigma_i$, el cual es un tipo de matriz de Vandermonde. Desafortunadamente estas matrices son notoriamente patológicas, por lo tanto sea cuidadoso para $M > 10$.

13.1.5 Bondades del ajuste.

Podemos fácilmente ajustar cada punto de dato si el número de parámetros, M, es igual al número de puntos de datos, N. En este caso ya sea que hemos construido una teoría de Rube Goldberg[4] o no hemos tomado suficientes datos. En vez de eso, supongamos el escenario más común en el cual $N \gg M$. Porque cada dato de punto tiene un error, no esperemos que la curva pase exactamente a través de los datos. Sin embargo, preguntemos, con las barras de error dadas, ¿cuán probable es que la curva en efecto describa los datos?. Por supuesto, si no se tienen barras de error no hay nada que se pueda decir acerca de la bondad del ajuste.

El sentido común sugiere que si el ajuste es bueno, entonces el promedio de la diferencia debería ser aproximadamente igual a la barra de error,

$$| \, y_i - Y(x_i) \, | \approx \sigma_i \ . \tag{13.31}$$

Colocando esto dentro de nuestra definición para χ^2, la ecuación (13.3), da $\chi^2 \approx N$. Sin embargo, sabemos que entre más parámetros usamos, más fácil es ajustar la curva; el ajuste puede ser perfecto si $M = N$. Esto sugiere que nuestra regla práctica para que un ajuste sea bueno sea

$$\chi^2 \approx N - M \ . \tag{13.32}$$

Por supuesto, este es sólo un crudo indicador, pero es mejor que una simple ojeada a la curva. Un análisis completo podría usar χ estadístico para asignar una probabilidad de que los datos sean ajustados por la curva.

Si encontramos que $\chi^2 \gg N - M$, entonces una de dos: no hemos usado una función apropiada, $Y(x)$, para nuestro ajuste de curva o las barras de errores, σ_I, son demasiado pequeñas. Por otra parte, si $\chi^2 \ll N - M$, el ajuste es tan espectacularmente bueno que podemos esperar que las barras de error sean realmente demasiado grandes.

[4]Como un caricaturista, Rube Golberg, quien creaba máquinas absurdamente elavoradas que realizaban tareas triviales.

Capítulo 14

Análisis vectorial.

versión final 2.0-19 agosto 2002[1]

14.1 Definiciones, una aproximación elemental.

En ciencia e ingeniería encontramos frecuentemente cantidades que tienen magnitud y sólo magnitud: la masa, el tiempo, la temperatura. Estas cantidades las etiquetamos como *escalares*. En constraste, muchas cantidades físicas interesantes tienen magnitud y, adicionalmente, una dirección asociada. Este segundo grupo incluye el desplazamiento, la velocidad, la aceleración, la fuerza, el momentum, el momentum angular. Cantidades con magnitud y dirección serán etiquetadas como cantidades *vectoriales*. Usualmente, en tratamientos elementales, un vector es definido como una cantidad que tiene magnitud y dirección. Para distinguir vectores de escalares, identificaremos las cantidades vectoriales con letra en "negrita", esto es, $\mathbf{V}$.

Un vector puede ser convenientemente representado por una flecha con largo proporcional a la magnitud. La dirección de la flecha da la dirección del vector, el sentido positivo de la dirección está siendo indicado por la punta. En esta representación la suma vectorial

$$\mathbf{C} = \mathbf{A} + \mathbf{B} \ , \tag{14.1}$$

consiste en poner en la parte posterior del vector $\mathbf{B}$ en la punta del vector $\mathbf{A}$. El vector $\mathbf{C}$ es

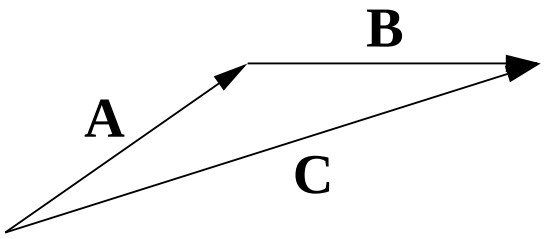


Figura 14.1: Ley del triángulo para suma de vectores.

el representado por una flecha dibujada desde la parte posterior de $\mathbf{A}$ y la punta de $\mathbf{B}$. Este

[1]Este capítulo está basado en el primer capítulo del libro: *Mathematical Methods for Physicists, fourth edition* de George B. Arfken & Hans J. Weber, editorial Academic Press.

procedimiento, la ley del triángulo de la suma, le asigna un significado a la ecuación (14.1) y es ilustrada en la figura 14.1. Al completar el paralelógramo, vemos que

$$\mathbf{C} = \mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A} \ , \tag{14.2}$$

como muestra la figura 14.2. En otras palabras la suma vectorial es *conmutativa*

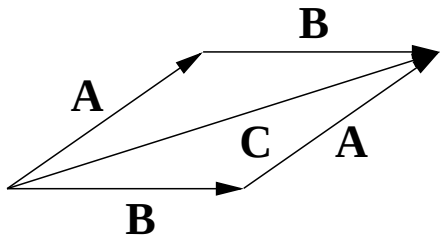


Figura 14.2: Ley del paralelógramo para suma de vectores.

Para la suma de tres vectores

$$\mathbf{D} = \mathbf{A} + \mathbf{B} + \mathbf{C} \ ,$$

figura 14.3, podemos primero sumar $\mathbf{A}$ y $\mathbf{B}$

$$\mathbf{A} + \mathbf{B} = \mathbf{E} \ .$$

Entonces a esta suma le agregamos $\mathbf{C}$

$$\mathbf{D} = \mathbf{E} + \mathbf{C} \ .$$

Similarmente, podemos primero sumar $\mathbf{B}$ y $\mathbf{C}$

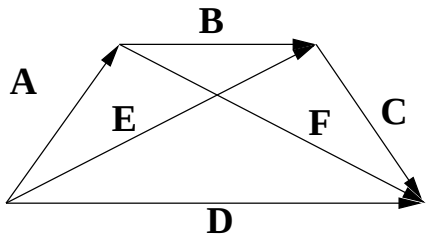


Figura 14.3: La suma de vectores es asociativa.

$$\mathbf{B} + \mathbf{C} = \mathbf{F} \ .$$

Entonces

$$\mathbf{D} = \mathbf{A} + \mathbf{F} \ .$$

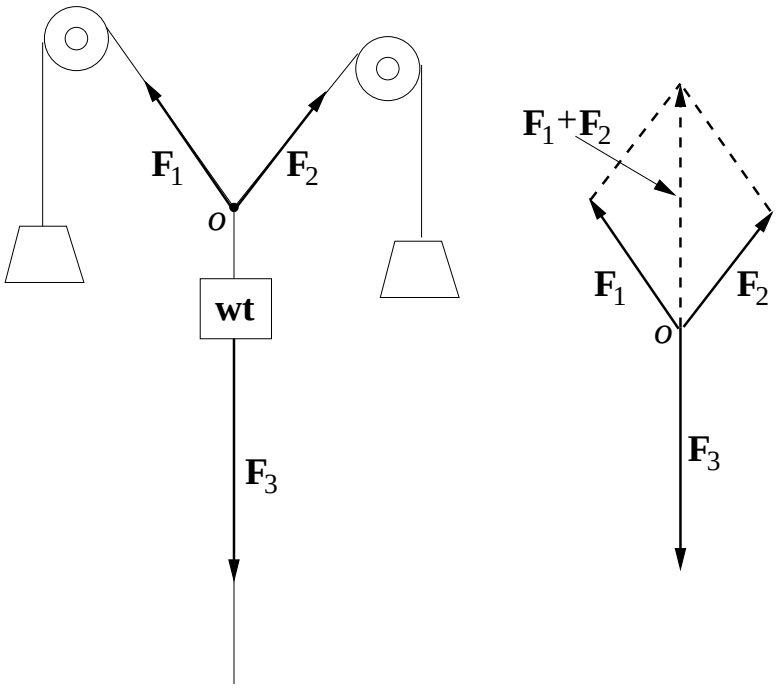


Figura 14.4: Equilibrio de fuerzas. $\mathbf{F}_1 + \mathbf{F}_2 = -\mathbf{F}_3$.

En términos de la expresión original,

$$(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C}) \ .$$

La suma de vectores es *asociativa*.

Un ejemplo físico directo de la ley de suma del paralelógramo es provisto por un peso suspendido de dos cuerdas. Si el punto de juntura (o en la figura 14.4) está en equilibrio, el vector suma de las dos fuerzas $\mathbf{F}_1$ y $\mathbf{F}_2$ deben justo cancelarse con la fuerza hacia abajo $\mathbf{F}_3$. Aquí la ley de suma del paralelógramo es sujeta a una verificación experimental inmediata.[2]

La resta puede ser manejada definiendo el negativo de un vector como un vector de la misma magnitud pero con la dirección contraria. Entonces

$$\mathbf{A} - \mathbf{B} = \mathbf{A} + (-\mathbf{B}) \ .$$

En la figura 14.3

$$\mathbf{A} = \mathbf{E} - \mathbf{B}$$

Notemos que los vectores son tratados como objetos geométricos que son independientes de cualquier sistema de coordenadas. Realmente, no hemos presentado aún un sistema de

[2]Estrictamente hablando la ley de suma del paralelógramo fue introducida como definición. Los experimentos muestran que si suponemos que las fuerzas son cantidades vectoriales y las combinamos usando la ley de suma del paralelógramo, la condición de una fuerza resultante nula como condición de equilibrio es satisfecha.

coordenadas. Este concepto de independencia de un particular sistema de coordenadas es desarrollado en detalle en la próxima sección.

La representación del vector **A** por una flecha sugiere una segunda posibilidad. La flecha **A** (figura 14.5), partiendo desde el origen,[3] y terminando en el punto (A_x, A_y, A_z). Así, acordamos que los vectores partan en el origen y su final puede ser especificado dando las coordenadas cartesianas (A_x, A_y, A_z) de la punta de la flecha.

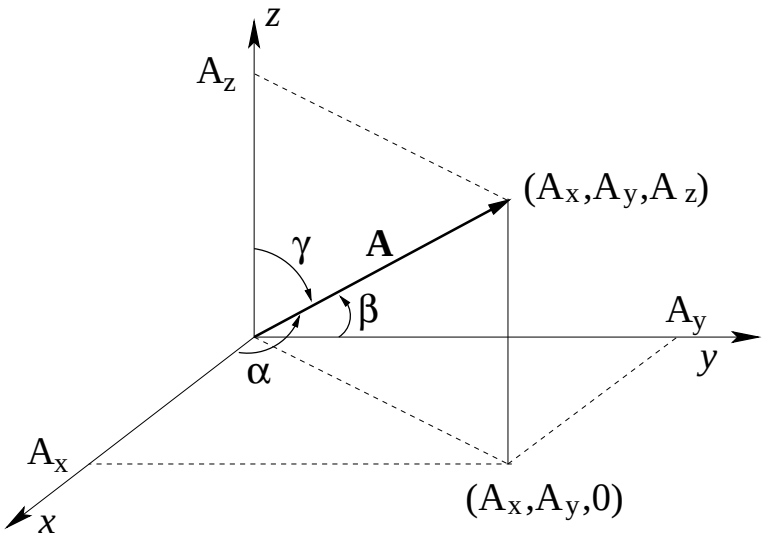


Figura 14.5: Componentes cartesianas y cosenos directores de **A**.

Aunque **A** podría haber representado cualquier cantidad vectorial (momentum, campo eléctrico, etc.) exite una cantidad vectorial importante, el desplazamiento desde el origen al punto (x, y, z), que es denotado por el símbolo especial **r**. Entonces tenemos la elección de referirnos al desplazamiento como el vector **r** o por la colección (x, y, z), que son las coordenadas del punto final:

$$\mathbf{r} \leftrightarrow (x, y, z) \ . \tag{14.3}$$

Usando r para la magnitud del vector **r** encontramos que la figura 14.5 muestra que las coordenadas del punto final y la magnitud están relacionadas por

$$x = r\cos\alpha \ , \quad y = r\cos\beta \ , \quad z = r\cos\gamma \ . \tag{14.4}$$

Los $\cos\alpha$, $\cos\beta$ y $\cos\gamma$ son llamados los *cosenos directores*, donde α es el ángulo entre el vector dado y el eje-x positivo, y así sucesivamente. Un poco de vocabulario: las cantidades A_x, A_y y A_z son conocidas como las *componentes* (cartesianas) de **A** o las *proyecciones* de **A**.

Así, cualquier vector **A** puede ser resuelto en sus componentes (o proyecciones sobre los ejes coordenados) produciendo $A_x = A\cos\alpha$, etc., como en la ecuación (14.4). Podemos elegir referirnos al vector como un cantidad simple **A** o por sus componentes (A_x, A_y, A_z). Notemos

[3]Se verá que se puede partir desde cualquier punto en el sistema de referencias cartesiano, hemos elegido el origen por simplicidad. Esta libertad de desplazar el origen del sistema de coordenadas sin afectar la geometría es llamada **invariancia translacional**.

que el subíndice x en A_x denota la componente x y no la dependencia sobre la variable x. La componente A_x puede ser función de x, y y z tal que $A_x(x, y, z)$. La elección entre usar $\mathbf{A}$ o sus componentes (A_x, A_y, A_z) es esencialmente una elección entre una representación geométrica y una algebraica. En el lenguaje de teoría de grupos (Capítulo 18), las dos representaciones son *isomórficas*.

Usaremos la representación que nos convenga más en cada caso. La representación geométrica "flecha en el espacio" puede ayudar en la visualización. El conjunto algebraico de componentes es usualmente mucho más conveniente para cálculos numéricos o algebraicos.

Los vectores entran en la Física de dos maneras distintas : (1) Un vector $\mathbf{A}$ puede representar una sola fuerza actuando sobre un único punto. La fuerza de gravedad actuando sobre el centro de gravedad ilustra esta forma. (2) Un vector $\mathbf{A}$ puede estar definido sobre una región extendida; esto es, $\mathbf{A}$ y sus componentes son funciones de la posición: $A_x = A_x(x, y, z)$, y así sucesivamente. Ejemplo de este tipo incluyen el campo de velocidades de un fluido. Algunos autores distinguen estos dos casos refiriéndose a los vectores definidos sobre una región como un campo vectorial. Los conceptos de campos vectoriales como funciones de la posición serán extremadamente importantes más adelante.

En este estado es conveniente presentar los vectores unitarios a lo largo de cada eje coordenado. Sea $\hat{\mathbf{x}}$ un vector de magnitud uno apuntando en la dirección x-positiva, $\hat{\mathbf{y}}$, un vector de magnitud uno apuntando en la dirección y-positiva, y $\hat{\mathbf{z}}$, un vector de magnitud unitaria en la dirección z-positiva. Entonces $\hat{\mathbf{x}}A_x$, es un vector con magnitud igual a A_x, en la dirección de x-positiva. Por suma de vectores

$$\mathbf{A} = \hat{\mathbf{x}}A_x + \hat{\mathbf{y}}A_y + \hat{\mathbf{z}}A_z \ , \tag{14.5}$$

lo cual establece que un vector es igual a la suma vectorial de sus componentes. Notemos que si $\mathbf{A}$ se anula, todas sus componentes deben anularse individualmente; esto es, si

$$\mathbf{A} = 0 \ , \qquad \text{entonces } A_x = A_y = A_z = 0 \ .$$

Finalmente, por el Teorema de Pitágoras, la magnitud del vector $\mathbf{A}$ es

$$A = (A_x^2 + A_y^2 + A_z^2)^{1/2} \ . \tag{14.6}$$

Esta resolución de un vector en sus componentes puede ser llevada a cabo en una variedad de sistemas de coordenadas, como mostramos en el próximo capítulo. Aquí nos restringiremos a coordenadas cartesianas.

La ecuación (14.5) es realmente la afirmación de que los tres vectores unitarios $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ y $\hat{\mathbf{z}}$ "abarcan" nuestro espacio real tridimensional. Cualquier vector constante puede ser escrito como una combinación lineal de $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ y $\hat{\mathbf{z}}$. Ya que $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ y $\hat{\mathbf{z}}$ son linealmente independientes (ninguno es una combinación lineal de los otros dos), ellos forman una *base* para el espacio real tridimensional.

Como un reemplazo a la técnica gráfica, la suma y resta de vectores puede ser llevada a cabo en términos de sus componentes. Para $\mathbf{A} = \hat{\mathbf{x}}A_x + \hat{\mathbf{y}}A_y + \hat{\mathbf{z}}A_z$ y $\mathbf{B} = \hat{\mathbf{x}}B_x + \hat{\mathbf{y}}B_y + \hat{\mathbf{z}}B_z$,

$$\mathbf{A} \pm \mathbf{B} = \hat{\mathbf{x}}(A_x \pm B_x) + \hat{\mathbf{y}}(A_y \pm B_y) + \hat{\mathbf{z}}(A_z \pm B_z) \ . \tag{14.7}$$

Deberíamos enfatizar aquí que los vectores unitarios $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, y $\hat{\mathbf{z}}$ son usados por conveniencia. Ellos no son esxenciales; podemos describir vectores y usarlos enteramente en términos de

sus componentes: $\mathbf{A} \leftrightarrow (A_x, A_y, A_z)$. Esta aproximación es la más poderosa, definiciones más sofisticadas de vectores serán discutidas en las próximas secciones.

Hasta aquí hemos definido las operaciones de suma y resta de vectores. Tres variedades de multiplicaciones son definidas sobre las bases de sus aplicaciones: un producto interno o escalar, un producto propio del espacio tridimensional, y un producto externo o directo que produce un tensor de rango dos. La división por un vector no está definida.

14.2 Rotación de los ejes de coordenadas.

En la sección anterior los vectores fueron definidos o representados en dos maneras equivalentes: (1) geométricamente especificando la magnitud y la dirección, con una flecha y (2) algebraicamente especificando las componentes relativas a ejes de coordenadas cartesianos. Esta segunda definición es adecuada para el análisis vectorial del resto del capítulo. En esta sección propondremos dos definiciones ambas más sofisticadas y poderosas. Primero, el campo vectorial está definido en términos del comportamiento de las componentes de los vectores bajo la rotación de los ejes de coordenadas. Este acercamiento teórico de transformación nos llevara al análisis tensorial en el capítulo 15 y a la teoría de grupo de las transformaciones en el capítulo 17. Segundo, la definición de componentes de la sección anterior será refinada y generalizada de acuerdo a los conceptos matemáticos de espacio vectorial. Este acercamiento nos llevará a los espacios vectoriales de funciones incluyendo o el espacio de Hilbert.

La definición de vector como una cantidad con magnitud y dirección se quiebra en trabajos avanzados. Por otra parte, encontramos cantidades, tales como constantes elásticas e índices de refracción en cristales anisotrópicos, que tienen magnitud y dirección *pero* no son vectores. Por otro lado, nuestra aproximación ingenua es inconveniente para generalizar y extenderla a cantidades más complejas. Tratamos de obtener una nueva definición de campo vectorial, usando nuestro vector desplazamiento $\mathbf{r}$ como un prototipo.

Hay una importante base física para el desarrollo de una nueva definición. Describimos nuestro mundo físico por matemáticas, pero él y cualquier predicción física que podamos hacer debe ser independiente de nuestro análisis matemático. Algunos comparan el sistema físico a un edificio y el análisis matemático al andamiaje usado para construir el edificio. Al final el andamiaje es sacado y el edificio se levanta.

En nuestro caso específico suponemos que el espacio es isotrópico; esto es, no hay direcciones privilegiadas o, dicho de otra manera, todas las direcciones son equivalentes. Luego cualquier sistema físico que está sienddo analizado o las bien las leyes físicas que son enunciadas no pueden ni debe depender de nuestra elección u *orientación* de los ejes coordenados.

Ahora, volveremos al concepto del vector $\mathbf{r}$ como un objeto geométrico independiente del sistema de coordenadas. Veamos un $\mathbf{r}$ en dos sistemas diferentes, uno rotado respecto al otro.

Por simplicidad consideremos primero el caso en dos dimensiones. Si las coordenadas x e y son rotadas en el sentido contrario a los punteros del reloj en un ángulo φ, *manteniendo* $\mathbf{r}$ *fijo* (figura 14.6) obtenemos las siguientes relaciones entre las componentes resueltas en el sistema original (sin primas) y aquellas resueltas en el nuevo sistema rotado (con primas):

$$\begin{aligned} x' &= \quad x\cos\varphi + y\,\mathrm{sen}\,\varphi\ , \\ y' &= -x\,\mathrm{sen}\,\varphi + y\cos\varphi\ . \end{aligned} \tag{14.8}$$

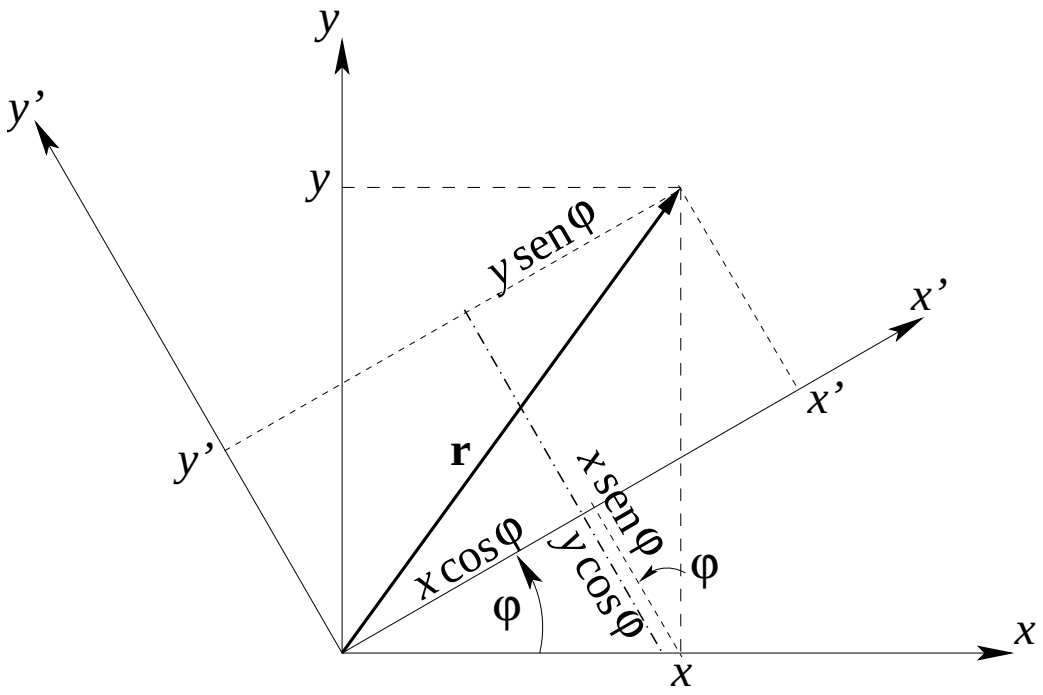


Figura 14.6: Rotación de los ejes coordenados cartesianos respecto del eje z.

Vimos en la sección anterior que un vector podría ser representado por las coordenadas de un punto; esto es, las coordenadas eran proporcionales a los componentes del vector. En consecuencia las componentes de un vector deberían transformar bajo rotaciones como las coordenadas de un punto (tal como $\mathbf{r}$). Por lo tanto cualquier par de cantidades $A_x(x, y)$ y $A_y(x, y)$ en el sistema de coordenadas xy es transformada en (A'_X, A'_y) por esta rotación del sistema de coordenadas con

$$\begin{aligned} A'_x &= \quad A_x \cos\varphi + A_y \operatorname{sen}\varphi \ , \\ A'_y &= -A_x \operatorname{sen}\varphi + A_y \cos\varphi \ , \end{aligned} \tag{14.9}$$

definimos[4] A_x y A_y como las componentes de un vector $\mathbf{A}$. Nuestro vector está definido ahora en términos de la transformación de sus componentes bajo rotación del sistema de coordenadas. Si A_x y A_y transforman de la misma manera como x e y, las componentes del vector de desplazamiento bidimensional, ellas son las componentes del vector $\mathbf{A}$. Si A_x y A_y no muestran esta forma invariante cuando las coordenadas son rotadas, ellas no forman un vector.

Las componentes del campo vectorial A_x y A_y que satisfacen las ecuaciones de definición (14.9), están asociadas a una magnitud A y a una dirección en el espacio. La magnitud es una cantidad escalar, invariante a la rotación del sistema de coordenadas. La dirección (relativa al sistema sin primas) es asimismo invariante ante la rotación del sistema de coordenadas. El resultado de todo esto es que las componentes de un vector pueden variar de acuerdo a la rotación del sistema de coordenadas con primas. Esto es lo que dicen las ecuaciones (14.9).

[4]La definición correspondiente de una cantidad escalar es $S' = S$, esto es, invariante ante rotaciones de los ejes de coordenadas.

Pero la variación con el ángulo es tal que las componentes en el sistema de coordenadas rotado A'_x y A'_y definen un vector con la misma magnitud y la misma dirección que el vector definido por las componentes A_x y A_y relativas al eje de coordenadas xy. Las componentes de $\mathbf{A}$ en un sistema de coordenadas particular constituye la *representación* de $\mathbf{A}$ en el sistema de coordenadas. Las ecuaciones (14.9), que corresponden a las relaciones de transformación, son una garantía de que la identidad $\mathbf{A}$ es preservada de la rotación del sistema de coordenadas. Para ir a tres y, luego, a cuatro dimensiones, encontramos conveniente usar una notación más compacta. Sea

$$\begin{aligned} x &\to x_1 \\ y &\to x_2 \end{aligned} \tag{14.10}$$

$$\begin{aligned} a_{11} &= \cos\varphi\ , \qquad a_{12} = \operatorname{sen}\varphi\ , \\ a_{21} &= -\operatorname{sen}\varphi\ , \qquad a_{22} = \cos\varphi\ . \end{aligned} \tag{14.11}$$

Las ecuaciones (14.8) transforman como

$$\begin{aligned} x'_1 &= a_{11}x_1 + a_{12}x_2\ , \\ x'_2 &= a_{21}x_1 + a_{22}x_2\ . \end{aligned} \tag{14.12}$$

Los coeficientes a_{ij} pueden ser interpretados como los cosenos directores, es decir, el coseno del ángulo entre x'_i y x_j; esto es,

$$\begin{aligned} a_{12} &= \cos(x'_1, x_2) = \operatorname{sen}\varphi, \\ a_{21} &= \cos(x'_2, x_1) = \cos\left(\varphi + \frac{\pi}{2}\right) = -\operatorname{sen}\varphi\ . \end{aligned} \tag{14.13}$$

La ventaja de la nueva notación[5]es que nos permite usar el símbolo suma $\sum$ y reescribir las ecuaciones (14.12) como

$$x'_i = \sum_{j=1}^{2} a_{ij}x_j\ , \qquad i = 1, 2\ . \tag{14.14}$$

Note que i permanece como un parámetro que da origen a una ecuación cuando este conjunto es igual a 1 y a una segunda ecuación cuando este conjunto es igual a 2. El índice j, por supuesto, es un índice de suma, es un índice mudo, como una variable de integración, j puede ser reemplazada por cualquier otro símbolo.

[5]Podemos extrañarnos por qué hemos sustituído un parámentro φ por cuatro parámetros a_{ij}. Claramente, los a_{ij} no constituyen un conjunto mínimo de parámetros. Para dos dimensiones los cuatro a_{ij} están sujetos a tres restricciones dadas en la ecuación (14.19). La justificación para el conjunto redundante de cosenos directores es la conveniencia que provee. Se tiene la esperanza que esta conveniencia sea más aparente en los próximos capítulos. Para rotaciones en tres dimensiones son nueve los a_{ij}, pero solamente tres de ellos son independientes, existen además descripciones alternativas como: los ángulos de Euler, los cuaterniones o los parámetros de Cayley-Klein. Estas alternativas tienen sus respectivas ventajas y sus desventajas.

La generalización para tres, cuatro, o N dimensiones es ahora muy simple. El conjunto de N cantidades, V_j, se dice que son las componentes de un vector de N dimensiones, $\mathbf{V}$, si y sólo si sus valores relativos a los ejes coordenados rotados están dados por

$$V_i' = \sum_{j=1}^{N} a_{ij} V_j \ , \qquad i = 1, 2, \ldots , N \ . \tag{14.15}$$

Como antes, a_{ij} es el coseno del ángulo entre x_i' y x_j. A menudo el límite superior N y el correspondiente intervalo de i no será indicado. Se da por descontado que se sabe en qué dimensión se está trabajando.

A partir de la definición de a_{ij} como el coseno del ángulo entre la dirección positiva de x_i' y la dirección positiva de x_j podemos escribir (coordenadas cartesianas)[6]

$$a_{ij} = \frac{\partial x_i'}{\partial x_j} \ . \tag{14.16}$$

Usando la rotación inversa ($\varphi \to -\varphi$) produce

$$x_j = \sum_{j=1}^{2} a_{ij} x_i' \ , \quad \text{o} \quad \frac{\partial x_j}{\partial x_i'} = a_{ij} \ . \tag{14.17}$$

Notemos que estas son derivadas parciales. Por uso de las ecuaciones (14.16), (14.17) y (14.15) se convierten en

$$V_i' = \sum_{j=1}^{N} \frac{\partial x_i'}{\partial x_j} V_j = \sum_{j=1}^{N} \frac{\partial x_j}{\partial x_i'} V_j \ . \tag{14.18}$$

Los cosenos directores a_{ij} satisface una *condición de ortogonalidad*

$$\sum_i a_{ij} a_{ik} = \delta_{jk} \ , \tag{14.19}$$

o, equivalentemente,

$$\sum_i a_{ji} a_{ki} = \delta_{jk} \ . \tag{14.20}$$

El símbolo δ_{ij} es la delta de Kronecker definida por

$$\delta_{ij} = \begin{cases} 1 & \text{para } i = j \ , \\ 0 & \text{para } i \neq j \ . \end{cases} \tag{14.21}$$

Podemos fácilmente verificar las ecuaciones (14.19) y (14.20) mantienendose en el caso de dos dimensiones y sustituyendo los específicos a_{ij} desde la ecuación (14.11). El resultado es la bien conocida identidad $\text{sen}^2\,\varphi + \cos^2\varphi = 1$ para el caso no nulo. Para verificar la ecuación

[6] Diferenciando $x_i' = \sum a_{ik} x_k$ con respecto a x_j.

(14.19) en forma general, podemos usar la forma en derivadas parciales de las ecuaciones (14.16) y (14.17) para obtener

$$\sum_i \frac{\partial x_j}{\partial x_i'}\frac{\partial x_k}{\partial x_i'} = \sum_i \frac{\partial x_j}{\partial x_i'}\frac{\partial x_i'}{\partial x_k} = \frac{\partial x_j}{\partial x_k} \ . \tag{14.22}$$

El último paso sigue las reglas usuales para las derivadas parciales, suponiendo que x_j es una función de x_1', x_2', x_3', etc. El resultado final, $\partial x_j/\partial x_k$, es igual a δ_{ij}, ya que x_j y x_k son coordenadas lineales ($j \neq k$) que se suponen perpendiculares (en dos o tres dimensiones) u ortogonales (para cualquier número de dimensiones). Equivalentemente, podemos suponer que x_j y x_k con ($j \neq k$) son variables totalmente independientes. Si $j = k$, la derivada parcial es claramente igual a 1. Al redefinir un vector en términos de cómo sus componentes transforman bajo rotaciones del sistema de coordenadas, deberíamos enfatizar dos puntos:

1.- Esta definición se desarrolla porque es útil y apropiada en describir nuestro mundo físico. Nuestras ecuaciones vectoriales serán independientes de cualquier sistema particular de coordenadas. (El sistema de coordenadas no necesita ser cartesiano.) La ecuación vectorial siempre puede ser expresada en algún sistema particular de coordenadas y, para obtener resultados numéricos, deberíamos finalmente expresarla en algún sistema de coordenadas específico.

2.- Esta definición se puede generalizar abriendo la rama de la matemática conocida como análisis tensorial, (próximo capítulo).

El comportamiento de las componentes vectoriales bajo rotaciones de las coordenadas es usado en al sección 14.3 para probar que un producto escalar es un escalar, en la sección 14.4 para probar que un producto vectorial es un vector, y en la sección 14.6 para mostrar que la gradiente de un escalar, $\bigtriangledown\psi$, es un vector. Lo que resta de este capítulo deriva de las definiciones menos restrictivas de vector que las dadas en la sección 14.1.

14.2.1 Vectores y espacio vectoriales.

Es habitual en matemáticas etiquetar un vector en tres dimensiones $\mathbf{x}$ por un triplete ordenado de números reales (x_1, x_2, x_3). El número x_n es llamado la componente n del vector $\mathbf{x}$. El conjunto de todos los vectores (que obedecen las propiedades que siguen) forman un espacio vectorial sobre los reales en este caso de tres dimensiones. Atribuimos cinco propiedades a nuestros vectores: si $\mathbf{x}$ =(x_1, x_2, x_3) e $\mathbf{y} = (y_1, y_2, y_3)$,

1. Igualdad entre vectores : $\mathbf{x} = \mathbf{y}$ significa $x_i = y_i$, para i =1, 2, 3.

2. Suma de vectores: $\mathbf{x} + \mathbf{y} = \mathbf{z}$ significa $x_i + y_i = z_i$, para i =1,2,3.

3. Multiplicación por un escalar: $a\mathbf{x} \leftrightarrow (ax_1, ax_2, ax_3)$ con $a \in \mathbb{R}$.

4. El negativo de un vector o inverso aditivo: $-\mathbf{x} = (-1)\mathbf{x} \leftrightarrow (-x_1, -x_2, -x_3)$.

5. Vector nulo: aquí existe un vector nulo $\mathbf{0} \leftrightarrow (0, 0, 0)$.

Ya que nuestras componentes vector son números reales, las siguientes propiedades también se mantienen:

1. La suma de vectores es conmutativa: $\mathbf{x}+\mathbf{y}=\mathbf{y}+\mathbf{x}$.
2. La suma de vectores es asociativa: $(\mathbf{x}+\mathbf{y})+\mathbf{z}=\mathbf{x}+(\mathbf{y}+\mathbf{z})$.
3. La multiplicación por escalar es distributiva: $a(\mathbf{x}+\mathbf{y})=a\mathbf{x}+a\mathbf{y}$, y también se cumple $(a+b)\mathbf{x}=a\mathbf{x}+b\mathbf{x}$.
4. La multiplicación por escalar es asociativa: $(ab)\mathbf{x}=a(b\mathbf{x})$.

Además, el vector nulo $\mathbf{0}$ es único, tal como lo es el inverso de un vector $\mathbf{x}$ dado.

Esta formulación nos permite formalizar la discusion de componentes de la sección 14.1. Su importancia radica en las extensiones, las cuales serán consideradas en los capítulos posteriores. En el capítulo 17, mostraremos que los vectores forman tanto un grupo Abeliano bajo la suma como un espacio lineal con las transformaciones en el espacio lineal descrito por matrices. Finalmente, y quizá lo más importante, para la física avanzada el concepto de vectores presentado aquí puede ser generalizado a números complejos, funciones, y a un número infinito de componentes. Esto tiende a espacios de funciones de dimensión infinita, espacios de Hilbert, los cuales son importantes en la teoría cuántica moderna.

14.3 Producto escalar o producto punto.

Habiendo definido vectores, debemos proceder a combinarlos. Las leyes para combinar vectores deben ser matemáticamente consistentes. A partir de las posibilidades que existen seleccionamos dos que son tanto matemática como físicamente interesantes. Una tercera posibilidad es introducida en el capítulo 15, en la cual formaremos tensores.

La proyección de un vector $\mathbf{A}$ sobre los ejes de coordenados, la cual define su componente cartesiana en la ecuación (14.4), es un caso especial del producto escalar de $\mathbf{A}$ y los vectores unitarios coordenados,

$$A_x = A\cos\alpha \equiv \mathbf{A}\cdot\hat{\mathbf{x}}\ , \quad A_y = A\cos\beta \equiv \mathbf{A}\cdot\hat{\mathbf{y}}\ , \quad A_x = A\cos\gamma \equiv \mathbf{A}\cdot\hat{\mathbf{z}}\ . \tag{14.23}$$

Al igual que la proyección es lineal en $\mathbf{A}$, deseamos que el producto escalar de dos vectores sea lineal en $\mathbf{A}$ y $\mathbf{B}$, es decir, obedezca las leyes de distributividad y asociatividad

$$\mathbf{A}\cdot(\mathbf{B}+\mathbf{C}) = \mathbf{A}\cdot\mathbf{B}+\mathbf{A}\cdot\mathbf{C}\ , \tag{14.24}$$

$$\mathbf{A}\cdot y\mathbf{B} = (y\mathbf{A})\cdot\mathbf{B} = y\mathbf{A}\cdot\mathbf{B}\ , \tag{14.25}$$

donde y es un número. Ahora podemos usar la descomposición de $\mathbf{B}$ en sus componentes cartesianas de acuerdo a la ecuación (14.5), $\mathbf{B}=B_x\hat{\mathbf{x}}+B_y\hat{\mathbf{y}}+B_z\hat{\mathbf{z}}$, para construir el producto escalar general o producto punto de los vectores $\mathbf{A}$ y $\mathbf{B}$ como

$$\begin{aligned}\mathbf{A}\cdot\mathbf{B} &= \mathbf{A}\cdot(B_x\hat{\mathbf{x}}+B_y\hat{\mathbf{y}}+B_z\hat{\mathbf{z}})\\ &= B_x\mathbf{A}\cdot\hat{\mathbf{x}}+B_y\mathbf{A}\cdot\hat{\mathbf{y}}+B_z\mathbf{A}\cdot\hat{\mathbf{z}}\\ &= B_xA_x+B_yA_y+B_zA_z\ ,\end{aligned}$$

por lo tanto

$$\mathbf{A}\cdot\mathbf{B}\equiv\sum_i B_iA_i=\sum_i A_iB_i=\mathbf{B}\cdot\mathbf{A}\ . \tag{14.26}$$

Si $\mathbf{A}=\mathbf{B}$ en la ecuación (14.26), recuperamos la magnitud $A=(\sum A_i^2)^{1/2}$ de $\mathbf{A}$ en la ecuación (14.6) a partir de la ecuación (14.26).

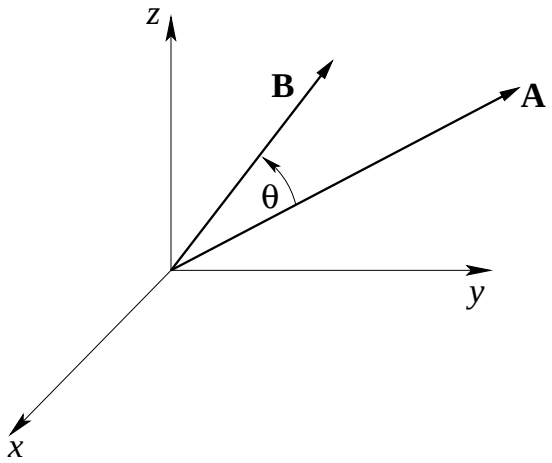


Figura 14.7: Producto escalar $\mathbf{A}\cdot\mathbf{B}=AB\cos\theta$.

Es obvio, a partir de la ecuación (14.26), que el producto escalar trata a $\mathbf{A}$ y $\mathbf{B}$ de la misma manera, o es simétrica en $\mathbf{A}$ y $\mathbf{B}$, y es conmutativa. Así, alternativamente y equivalentemente, podemos primero generalizar la ecuación (14.23) a la proyección A_B de un vector $\mathbf{A}$ sobre la dirección de un vector $\mathbf{B}\neq 0$ como $A_B=A\cos\theta\equiv\mathbf{A}\cdot\hat{\mathbf{B}}$, donde $\hat{\mathbf{B}}=\mathbf{B}/B$ es el vector unitario en la direción de $\mathbf{B}$ y θ es el ángulo entre $\mathbf{A}$ y $\mathbf{B}$ como muestra la figura 14.7. Similarmente, proyectamos $\mathbf{B}$ sobre $\mathbf{A}$ como $B_A=B\cos\theta\equiv\mathbf{B}\cdot\hat{\mathbf{A}}$. Segundo, hacemos esta proyección simétrica en $\mathbf{A}$ y $\mathbf{B}$, la cual produce la definición

$$\mathbf{A}\cdot\mathbf{B}\equiv A_BB=AB_A=AB\cos\theta\ . \tag{14.27}$$

La ley distributiva en la ecuación (14.24) es ilustrada en la figura 14.8, en la cual se muestra que la suma de las proyecciones de $\mathbf{B}$ y $\mathbf{C}$, B_A+C_A, es igual a la proyección de $\mathbf{B}+\mathbf{C}$ sobre $\mathbf{A}$, $(\mathbf{B}+\mathbf{C})_A$.

Se sigue a partir de las ecuaciones (14.23), (14.26) y (14.27) que los vectores unitarios coordenados satisfacen la relación

$$\hat{\mathbf{x}}\cdot\hat{\mathbf{x}}=\hat{\mathbf{y}}\cdot\hat{\mathbf{y}}=\hat{\mathbf{z}}\cdot\hat{\mathbf{z}}=1\ , \tag{14.28}$$

mientras

$$\hat{\mathbf{x}}\cdot\hat{\mathbf{y}}=\hat{\mathbf{x}}\cdot\hat{\mathbf{z}}=\hat{\mathbf{y}}\cdot\hat{\mathbf{z}}=0\ . \tag{14.29}$$

Si la definición de las componentes, ecuación (14.26), es etiquetada como una definición algebraica, entonces la ecuación (14.27) es una definición geométrica. Una de las más comunes

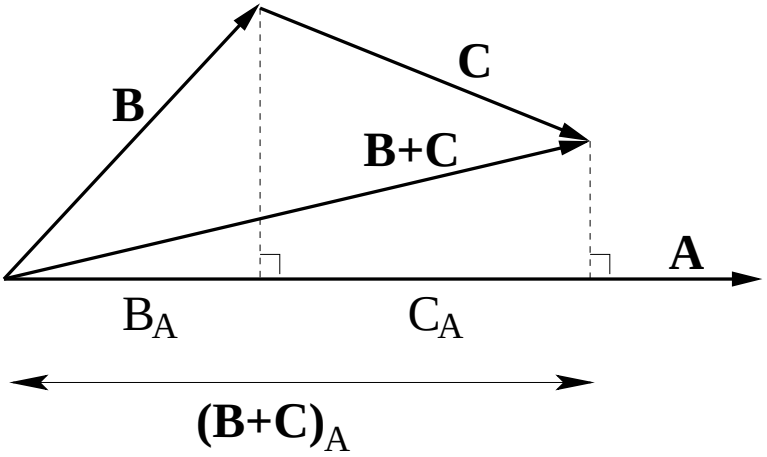


Figura 14.8: La ley distributiva $\mathbf{A}\cdot(\mathbf{B}+\mathbf{C}) = AB_A + AC_A = A(\mathbf{B}+\mathbf{C})_A$, ecuación (14.24).

aplicaciones del producto escalar en Física es el cálculo del trabajo ejercido por una fuerza constante = fuerza × desplazamiento × $\cos\theta$, lo cual es interpretado como el desplazamiento por la proyección de la fuerza en la dirección del desplazamiento, *i.e.*, el producto escalar de la fuerza y el desplazamiento, $W = \mathbf{F}\cdot\mathbf{S}$.

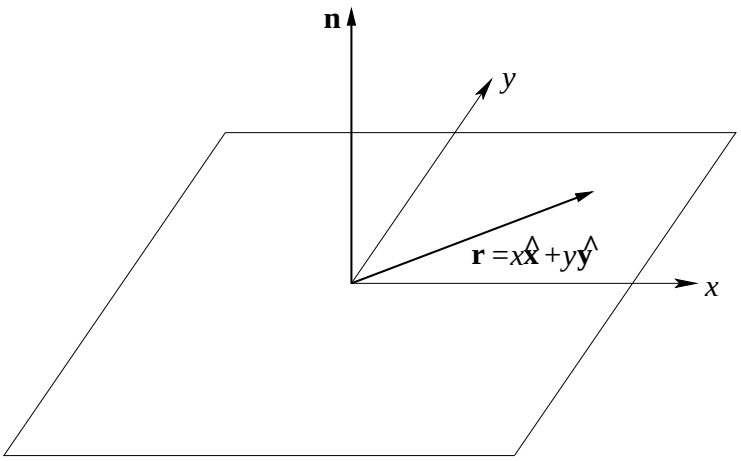


Figura 14.9: Un vector normal.

Si $\mathbf{A}\cdot\mathbf{B} = 0$ y sabemos que $\mathbf{A} \neq 0$ y $\mathbf{B} \neq 0$, entonces desde la ecuación (14.27), $\cos\theta = 0$ o θ =90°, 270°, y así sucesivamente. Los vectores $\mathbf{A}$ y $\mathbf{B}$ deben ser perpendiculares. Alternativamente, podemos decir que son ortogonales. Los vectores unitarios $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ y $\hat{\mathbf{z}}$ son mutuamente ortogonales. Para desarrollar esta noción de ortogonalidad demos un paso más, supongamos que $\mathbf{n}$ es un vector unitario y $\mathbf{r}$ es un vector distinto de cero en el plano-xy; esto es $\mathbf{r} = \hat{\mathbf{x}}x + \hat{\mathbf{y}}y$ (figura 14.9). Si

$$\mathbf{n}\cdot\mathbf{r} = 0\ ,$$

para todas las elecciones posibles de $\mathbf{r}$, entonces $\mathbf{n}$ debe ser perpendicular (ortogonal) al plano-xy.

A menudo es conveniente reemplazar $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ y $\hat{\mathbf{z}}$ por vectores unitarios con subíndice $\mathbf{e}_m$ con $m = 1, 2, 3$, con $\hat{\mathbf{x}} = \mathbf{e}_1$ y así sucesivamente. Entonces las ecuaciones (14.28) y (14.29) llegan a ser

$$\mathbf{e}_m \cdot \mathbf{e}_n = \delta_{mn} \ . \tag{14.30}$$

Para $m \neq n$ los vectores unitarios $\mathbf{e}_m$ y $\mathbf{e}_n$ son ortogonales. Para $m = n$ cada vector es normalizado a la unidad, esto es, tiene magnitud uno. El conjunto de vectores $\mathbf{e}_m$ se dice que es *ortonormal*. La mayor ventaja de la ecuación (14.30) sobre las ecuaciones (14.28) y (14.29) es que la ecuación (14.30) puede ser fácilmente genrealizada a un espacio de N dimensiones; $m, n = 1, 2, \ldots, N$. Finalmente, escogeremos un conjunto de vectores unitarios $\mathbf{e}_m$ que sean ortonormales por conveniencia.

14.3.1 Invariancia del producto escalar bajo rotaciones.

No hemos mostrado aún que la palabra escalar este justificada o que el producto escalar sea realmente un a cantidad escalar. Para hacer esto, investiguemos el comportamiento de $\mathbf{A} \cdot \mathbf{B}$ bajo una rotación del sistema de coordenadas. Usando la ecuación (14.15)

$$A'_x B'_x + A'_y B'_y + A'_z B'_z = \sum_i a_{xi} A_i \sum_j a_{xj} B_j + \sum_i a_{yi} A_i \sum_j a_{yj} B_j + \sum_i a_{zi} A_i \sum_j a_{zj} B_j \ . \tag{14.31}$$

Usando los índices k y l para sumar sobre x, y y z, obtenemos

$$\sum_k A'_k B'_k = \sum_l \sum_i \sum_j a_{li} A_i a_{lj} B_j \ , \tag{14.32}$$

y, rearreglando términos en el lado derecho, tenemos

$$\sum_k A'_k B'_k = \sum_l \sum_i \sum_j (a_{li} a_{lj}) A_i B_j = \sum_i \sum_j \delta_{ij} A_i B_j = \sum_i A_i B_i \ . \tag{14.33}$$

Los dos últimos pasos se siguen de la ecuación (14.19), la condición de ortogonalidad de los cosenos directores, y de la ecuación (14.21) la cual define la delta de Kronecker. El efecto de la delta de Kronecker es cancelar todos los términos en la suma sobre uno de sus dos índices excepto el término en el cual son iguales. En la ecuación (14.33) su efecto es fijar $j = i$ y eliminar la suma sobre j. Por supuesto, podríamos fijar $i = j$ y eliminar la suma sobre i. la ecuación (14.33) nos da

$$\sum_k A'_k B'_k = \sum_i A_i B_i \ , \tag{14.34}$$

la cual es justo nuestra definición de una cantidad escalar, una que permanece *invariante* ante rotaciones de los sistemas de coordenadas.

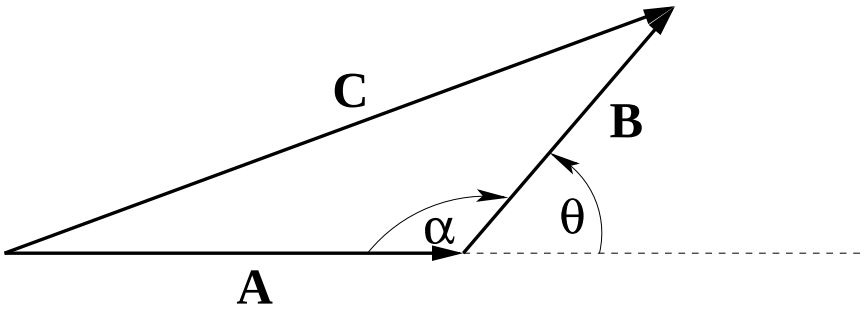


Figura 14.10: La ley de los cosenos.

En un acercamiento similar el cual se aprovecha de este concepto de invarianza, tomamos $\mathbf{C} = \mathbf{A} + \mathbf{B}$ y hacemos producto punto consigo mismo.

$$\begin{aligned} \mathbf{C} \cdot \mathbf{C} &= (\mathbf{A} + \mathbf{B}) \cdot (\mathbf{A} + \mathbf{B}) \\ &= \mathbf{A} \cdot \mathbf{A} + \mathbf{B} \cdot \mathbf{B} + 2\mathbf{A} \cdot \mathbf{B} \ . \end{aligned} \tag{14.35}$$

Ya que

$$\mathbf{C} \cdot \mathbf{C} = C^2 \ , \tag{14.36}$$

el cuadrado de la magnitud del vector $\mathbf{C}$ y por esto una cantidad invariante, veamos que

$$\mathbf{A} \cdot \mathbf{B} = \frac{C^2 - A^2 - B^2}{2} \ , \quad \text{invariante.} \tag{14.37}$$

Ya que la mano derecha de la ecuación (14.37) es invariante –esto es, una cantidad escalar– el lado izquierdo, $\mathbf{A} \cdot \mathbf{B}$ también debiera ser invariante bajo una rotación del sistema de coordenadas. En consecuencia $\mathbf{A} \cdot \mathbf{B}$ es un escalar.

La ecuación (14.35) es realmente otra forma de escribir el teorema del coseno, el cual es

$$\begin{aligned} C^2 &= A^2 + B^2 + 2AB\cos\theta \ , \\ &= A^2 + B^2 + 2AB\cos(\pi - \alpha) \ , \\ &= A^2 + B^2 - 2AB\cos\alpha \ , \end{aligned} \tag{14.38}$$

Comparando las ecuaciones (14.35) y (14.38), tenemos otra verificación de la ecuación (14.27), o, si se prefiere, una derivación vectorial del teorema del coseno (figura 14.10).

El producto punto, dado por la ecuación (14.26), podría ser generalizado en dos formas. El espacio no necesita ser restringido a tres dimensiones. En un espacio n-dimensional, la ecuación se aplica con la suma corriendo desde 1 a n. Donde n puede ser infinito, con la suma como una serie convergente infinita. La otra generalización extiende el concepto de vector para abarcar las funciones. La función análoga a un producto punto o interno aparece más adelante.

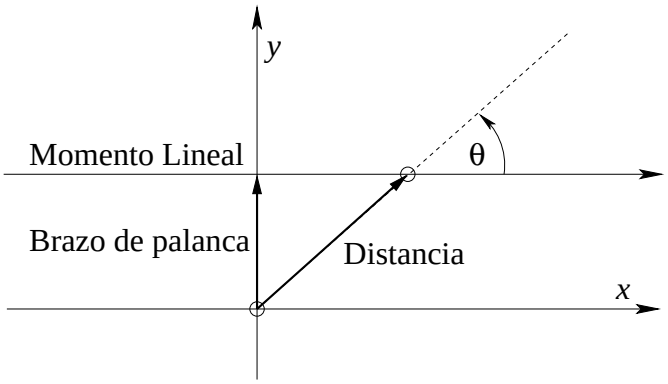


Figura 14.11: Momento angular.

14.4 Producto vectorial o producto cruz.

Una segunda forma de multiplicar vectores emplea el seno del ángulo sustentado en vez del coseno. Por ejemplo, el momento angular (figura 1.11) de un cuerpo está definido como

$$\begin{aligned}\text{momento angular} &= \text{brazo de palanca} \times \text{momento lineal} \\ &= \text{distancia} \times \text{momento lineal} \times \operatorname{sen}\theta \ .\end{aligned}$$

Por conveniencia en el tratamiento de problemas relacionados con cantidades tales como momento angular, torque y velocidad angular, definiremos el producto vectorial o producto cruz como

$$\mathbf{C} = \mathbf{A} \times \mathbf{B} \ ,$$

con

$$C = AB \operatorname{sen}\theta \ . \tag{14.39}$$

A diferencia del caso anterior del producto escalar, $\mathbf{C}$ ahora es un vector, y le asignamos una dirección perpendicular al plano que contiene a $\mathbf{A}$ y $\mathbf{B}$ tal que $\mathbf{A}$, $\mathbf{B}$ y $\mathbf{C}$ forman un sistema diestro. Con esta elección de dirección tenemos

$$\mathbf{A} \times \mathbf{B} = -\mathbf{B} \times \mathbf{A} \ , \quad \text{anticonmutación.} \tag{14.40}$$

A partir de esta definición de producto cruz tenemos

$$\hat{\mathbf{x}} \times \hat{\mathbf{x}} = \hat{\mathbf{y}} \times \hat{\mathbf{y}} = \hat{\mathbf{z}} \times \hat{\mathbf{z}} = 0 \ , \tag{14.41}$$

mientras

$$\begin{aligned} \hat{\mathbf{x}} \times \hat{\mathbf{y}} &= \ \ \hat{\mathbf{z}} \ , \quad \hat{\mathbf{y}} \times \hat{\mathbf{z}} = \ \ \hat{\mathbf{x}} \ , \quad \hat{\mathbf{z}} \times \hat{\mathbf{x}} = \ \ \hat{\mathbf{y}} \ , \\ \hat{\mathbf{y}} \times \hat{\mathbf{x}} &= -\hat{\mathbf{z}} \ , \quad \hat{\mathbf{z}} \times \hat{\mathbf{y}} = -\hat{\mathbf{x}} \ , \quad \hat{\mathbf{x}} \times \hat{\mathbf{z}} = -\hat{\mathbf{y}} \ . \end{aligned} \tag{14.42}$$

Entre los ejemplos del producto cruz en Física Matemática están la relación entre el momento lineal $\mathbf{p}$ y el momento angular $\mathbf{L}$ (que definine al momento angular),

$$\mathbf{L} = \mathbf{r} \times \mathbf{p} \ ,$$

y la relación entre velocidad lineal $\mathbf{v}$ y velocidad angular $\boldsymbol{\omega}$,

$$\mathbf{v} = \boldsymbol{\omega} \times \mathbf{r} \ .$$

Los vectores $\mathbf{v}$ y $\mathbf{p}$ describen propiedades de las partículas o un sistema físico. Sin embargo, la posición del vector $\mathbf{r}$ está determinado por la elección del origen de las coordenadas. Esto significa que $\boldsymbol{\omega}$ y $\mathbf{L}$ dependen de la elección del origen.

La campo magnético $\mathbf{B}$ usualmente está definido por el producto vectorial de la ecuación de fuerza de Lorentz[7]

$$\mathbf{F} = \frac{q}{c}\,\mathbf{v} \times \mathbf{B} \ , \quad (\text{CGS}).$$

Donde $\mathbf{v}$ es la velocidad de la carga eléctrica q, c es la velocidad de la luz en el vacío y $\mathbf{F}$ es la fuerza resultante sobre la carga en movimiento.

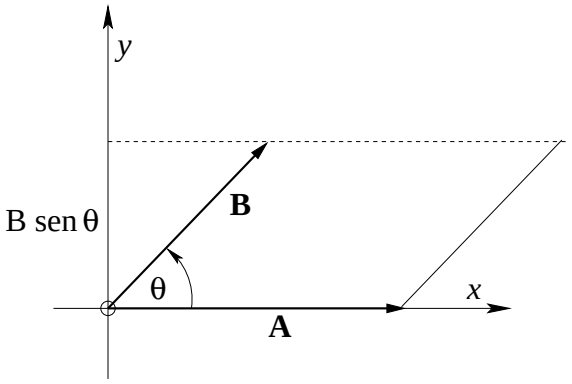


Figura 14.12: Paralelógramo que representa el producto cruz.

El producto cruz tiene una importante interpretación geométrica la cual se usará en secciones futuras. En el paralelógramo definido por $\mathbf{A}$ y $\mathbf{B}$ (figura 14.12), $B \operatorname{sen}\theta$ es la altura si A es tomada como la longitud de la base. Luego $|\,(\mathbf{A} \times \mathbf{B})\,| = AB \operatorname{sen}\theta$ es el área del paralelógramo. Como un vector, $\mathbf{A} \times \mathbf{B}$ es el área del paralelógramo definido por $\mathbf{A}$ y $\mathbf{B}$, con el vector normal al plano del paralelógramo. Esto sugiere que el área podría ser tratada como una cantidad vectorial.

Entre paréntesis, podemos notar que la ecuación (14.42) y una ecuación modificada (14.41) forman el punto de partida para el desarrollo de los cuaterniones. La ecuación (14.41) es remplazada por $\hat{\mathbf{x}} \times \hat{\mathbf{x}} = \hat{\mathbf{y}} \times \hat{\mathbf{y}} = \hat{\mathbf{z}} \times \hat{\mathbf{z}} = -1$.

Una definición alternativa del producto vectorial puede ser derivada del caso especial de los vectores unitarios coordenados en las ecuaciones (14.40-14.42), en conjunto con la

[7]El campo eléctrico $\mathbf{E}$ lo hemos supuesto nulo.

linealidad del producto cruz en ambos argumentos vectoriales, en analogía con la ecuación (14.24) y (14.25) para el producto punto,

$$\begin{aligned} \mathbf{A}\times(\mathbf{B}+\mathbf{C}) &= \mathbf{A}\times\mathbf{B}+\mathbf{A}\times\mathbf{C}\ , \\ (\mathbf{A}+\mathbf{B})\times\mathbf{C} &= \mathbf{A}\times\mathbf{C}+\mathbf{B}\times\mathbf{C}\ , \end{aligned} \tag{14.43}$$

$$\mathbf{A}\times(y\mathbf{B}) = y\mathbf{A}\times\mathbf{B} = (y\mathbf{A})\times\mathbf{B}\ , \tag{14.44}$$

donde y es un número. Usando la descomposición de $\mathbf{A}$ y $\mathbf{B}$ en sus componentes cartesianas de acuerdo a la ecuación (14.5), encontramos

$$\begin{aligned} \mathbf{A}\times\mathbf{B}\equiv\mathbf{C} = (C_x,C_y,C_z) &= (A_x\hat{\mathbf{x}}+A_y\hat{\mathbf{y}}+A_z\hat{\mathbf{z}})\times(B_x\hat{\mathbf{x}}+B_y\hat{\mathbf{y}}+B_z\hat{\mathbf{z}}) \\ &= (A_xB_y-A_yB_x)\hat{\mathbf{x}}\times\hat{\mathbf{y}}+(A_xB_z-A_zB_x)\hat{\mathbf{x}}\times\hat{\mathbf{z}}+(A_yB_z-A_zB_y)\hat{\mathbf{y}}\times\hat{\mathbf{z}}\ , \end{aligned}$$

por aplicación de las ecuaciones (14.43),(14.44) y sustituyendo en las ecuaciones (14.40-14.42), tal que las componentes cartesianas de $\mathbf{A}\times\mathbf{B}$ sean

$$C_x = A_yB_z-A_zB_y\ ,\quad C_Y = A_zB_x-A_xB_z\ ,\quad C_z = A_xB_y-A_yB_x\ , \tag{14.45}$$

o

$$C_i = A_jB_k-A_kB_j\ ,\quad i,\,j,\,k \text{ todos diferentes}, \tag{14.46}$$

y con la permutación ciclíca de los índices i, j, k. El producto vectorial $\mathbf{C}$ puede ser convenientemente representado por un determinante

$$\mathbf{C} = \begin{bmatrix} \hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{bmatrix}\ . \tag{14.47}$$

La expansión del determinante por la fila superior reproduce las tres componentes de $\mathbf{C}$ listadas en la ecuación (14.45).

La ecuación (14.39) podría ser llamada una definición geométrica del producto vectorial. Luego la ecuación (14.45) podría ser una definición algebraica.

Para mostrar la equivalencia de la ecuación (14.39) y la definición de componente, ecuación (14.45), formemos $\mathbf{A}\cdot\mathbf{C}$ y $\mathbf{B}\cdot\mathbf{C}$, usando la ecuación (14.45). Tenemos

$$\begin{aligned} \mathbf{A}\cdot\mathbf{C} &= \mathbf{A}\cdot(\mathbf{A}\times\mathbf{B})\ , \\ &= A_x(A_yB_z-A_zB_y)+A_y(A_zB_x-A_xB_z)+A_z(A_xB_y-A_yB_x)\ , \\ &= 0\ . \end{aligned} \tag{14.48}$$

Similarmente,

$$\mathbf{B}\cdot\mathbf{C} = \mathbf{B}\cdot(\mathbf{A}\times\mathbf{B}) = 0\ . \tag{14.49}$$

Las ecuaciones (14.48) y (14.49) muestran que $\mathbf{C}$ es perpendicular tanto al vector $\mathbf{A}$ como al vector $\mathbf{B}$ ($\cos\theta = 0, \theta = \pm 90^\circ$) y por lo tanto perpendicular al plano que ellos determinan.

La dirección positiva está determinada considerando el caso especial de los vectores unitarios $\hat{\mathbf{x}} \times \hat{\mathbf{y}} = \hat{\mathbf{z}}$ ($C_z = +A_x B_y$).

La magnitud es obtenida a partir de

$$\begin{aligned}(\mathbf{A} \times \mathbf{B}) \cdot (\mathbf{A} \times \mathbf{B}) &= A^2 B^2 - (\mathbf{A} \cdot \mathbf{B})^2 \ , \\ &= A^2 B^2 - A^2 B^2 \cos^2 \theta \ , \\ &= A^2 B^2 \operatorname{sen}^2 \theta \ .\end{aligned} \tag{14.50}$$

De donde

$$C = AB \operatorname{sen} \theta \ . \tag{14.51}$$

El gran primer paso en la ecuación (14.50) puede ser verificado expandiendo en componentes, usando la ecuación (14.45) para $\mathbf{A} \times \mathbf{B}$ y la ecuación (14.26) para el producto punto. A partir de las ecuaciones (14.48), (14.49) y (14.51) vemos la equivalencia de las ecuaciones (14.39) y (14.45), las dos definiciones del producto vectorial. Todavía permanece el problema de verificar que $\mathbf{C} = \mathbf{A} \times \mathbf{B}$ es por cierto un vector; esto es, que obedece la ecuación (14.15), la ley de transformación vectorial. Comencemos en un sistema rotado (sistema prima)

$$\begin{aligned}C'_i &= A'_j B'_k - A'_k B'_j \ , \quad i, \ j \text{ y } k \text{ en orden cíclico}, \\ &= \sum_l a_{jl} A_l \sum_m a_{km} B_m - \sum_l a_{kl} A_l \sum_m a_{jm} B_m \ , \\ &= \sum_{l,m} (a_{jl} a_{km} - a_{kl} a_{jm}) A_l B_m \ .\end{aligned} \tag{14.52}$$

La combinación de cosenos directores en paréntesis desaparece para $m = l$. Por lo tanto tenemos que j y k toman valores fijos, dependiendo de la elección de i, y seis combinaciones de l ym. Si $i = 3$, luego $j = 1$, $k = 2$ (en orden cíclico), y tenemos la siguiente combinación de cosenos directores[8]

$$\begin{aligned}a_{11} a_{22} - a_{21} a_{12} &= a_{33} \ , \\ a_{13} a_{21} - a_{23} a_{11} &= a_{32} \ , \\ a_{12} a_{23} - a_{22} a_{13} &= a_{31} \ ,\end{aligned} \tag{14.53}$$

y sus negativos. Las ecuaciones (14.53) son identidades que satisfacen los cosenos directores. Ellas pueden ser verificadas con el uso de determinantes y matrices. Sustituyendo hacia atrás en la ecuación (14.52),

$$\begin{aligned}C'_3 &= a_{33} A_1 B_2 + a_{32} A_3 B_1 + a_{31} A_2 B_3 - a_{33} A_2 B_1 - a_{32} A_1 B_3 + a_{31} A_3 B_2 \ , \\ &= a_{31} C_1 + a_{32} C_2 + a_{33} C_3 \ , \\ &= \sum_n a_{3n} C_n \ .\end{aligned} \tag{14.54}$$

[8]La ecuación (14.53) se mantiene ante rotaciones porque ellas mantienen el volumen.

Permutando los índices para levantar C_1' y C_2', vemos que la ecuación (14.15) se satisface y $\mathbf{C}$ es por cierto un vector. Podría mencionarse aquí que esta naturaleza vectorial del producto cruz es un accidente asociado con la naturaleza tridimensional del espacio ordinario[9]. Veremos en el capítulo siguiente que el producto cruz también puede ser tratado como un tensor asimétrico de rango dos.

Definimos un vetor como un triplete ordenado de números (o funciones) como en la última parte de la sección 14.2, luego no hay problemas en identificar el producto cruz como un vector. La operación de producto cruz mapea los dos tripletes $\mathbf{A}$ y $\mathbf{B}$ en un tercer triplete $\mathbf{C}$ el cual por definición es un vector.

Ahora tenemos dos maneras de multipicar vectores; una tercera forma aparece en el próximo capítulo . Pero ¿qué hay de la división por un vector? Esto resulta ser que la razón $\mathbf{B}/\mathbf{A}$ no está unívocamente especificada a menos que $\mathbf{A}$ y $\mathbf{B}$ sean paralelos. Por lo tanto la división de un vector por otro no está bien definida.

14.5 Productos escalar triple y vectorial triple.

14.5.1 Producto escalar triple.

Las secciones 14.3 y 14.4 cubren los dos tipos de multiplicación de interés aquí. Sin embargo, hay combinaciones de tres vectores, $\mathbf{A}\cdot(\mathbf{B}\times\mathbf{C})$ y $\mathbf{A}\times(\mathbf{B}\times\mathbf{C})$, las cuales ocurren con la suficiente frecuencia para merecer una atención más amplia. La combinación

$$\mathbf{A}\cdot(\mathbf{B}\times\mathbf{C})\ ,$$

es conocida como el producto escalar triple. El producto $\mathbf{B}\times\mathbf{C}$ produce un vector, el cual producto punto con $\mathbf{A}$, da un escalar. Notemos que $(\mathbf{A}\cdot\mathbf{B})\times\mathbf{C}$ representa un escalar producto cruz con un vector, una operación que no está definida. Por lo tanto, si estamos de acuerdo en excluír estas interpretaciones no definidas, los paréntesis puede ser omitidos y el producto de escalar triple puede ser escrito como $\mathbf{A}\cdot\mathbf{B}\times\mathbf{C}$.

Usando la ecuación (14.45) para el producto cruz y la ecuación (14.26) para el producto punto, obtenemos

$$\begin{aligned}\mathbf{A}\cdot\mathbf{B}\times\mathbf{C} &= A_x(B_yC_z-B_zC_y)+A_y(B_zC_x-B_xC_z)+A_z(B_xC_y-B_yC_x)\ ,\\ &=\mathbf{B}\cdot\mathbf{C}\times\mathbf{A}=\mathbf{C}\cdot\mathbf{A}\times\mathbf{B}\ ,\\ &=-\mathbf{A}\cdot\mathbf{C}\times\mathbf{B}=-\mathbf{C}\cdot\mathbf{B}\times\mathbf{A}=-\mathbf{B}\cdot\mathbf{A}\times\mathbf{C}\ ,\quad \text{y así sucesivamente.}\end{aligned}\tag{14.55}$$

Notemos el alto grado de simetría presente en la expansión en componentes. Cada término contiene los factores A_i, B_j y C_k. Si i, j, k están en un orden cíclico (x,y,z), el signo es positivo. Si el orden es anticíclico, el signo es negativo. Por lo tanto, el punto y la cruz pueden ser intercambiados,

$$\mathbf{A}\cdot\mathbf{B}\times\mathbf{C}=\mathbf{A}\times\mathbf{B}\cdot\mathbf{C}\ .\tag{14.56}$$

[9]Específicamente la ecuación (14.53) se mantiene sólo para un espacio tridimensional. Técnicamente, es posible definir un producto cruz en $\mathbb{R}^7$, el espacio de dimensión siete, pero el producto cruz resulta tener propiedades inaceptables (patológicas).

Una representación conveniente de la componente de expansión de la ecuación (14.55) está dada por el determinante

$$\mathbf{A}\cdot\mathbf{B}\times\mathbf{C}=\begin{vmatrix}A_x & A_y & A_z\\ B_x & B_y & B_z\\ C_x & C_y & C_z\end{vmatrix}\ . \tag{14.57}$$

La regla para intercambiar filas por columnas de un determinante provee una inmediata verificación de la permutación listada en la ecuación (14.55), mientras la simetría de **A**, **B** y **C** en la forma determinante sugiere la relación dada en la ecuación (14.56).

El producto triple encontrado en la sección 14.4, en la cual se muestra que $\mathbf{A}\times\mathbf{B}$ era perpendicular a ambos, **A** y **B**, son casos especiales del resultado general (ecuación (14.55)).

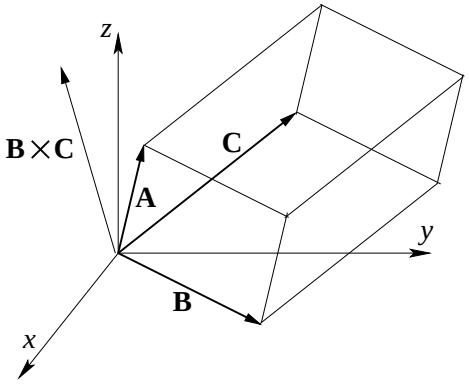


Figura 14.13: Paralelepípedo que representa el producto escalar triple.

El producto escalar triple tiene una interpretación geométrica directa. Los tres vectores **A**, **B**, y **C** pueden ser interpretados como la definición de un paralelepípedo (figura 14.13).

$$\begin{aligned}|\,\mathbf{B}\times\mathbf{C}\,| &= BC\,\mathrm{sen}\,\theta\ ,\\ &= \text{área de la base del paralelógramo.}\end{aligned} \tag{14.58}$$

La dirección, por supuesto, es normal a la base. Haciendo el producto punto con **A**, esto significa multiplicar el área de la base, por la proyección de A sobre la normal, o la base tantas veces por la altura. Por lo tanto

$$\mathbf{A}\cdot\mathbf{B}\times\mathbf{C}=\text{volumen del paralelepípedo definido por }\mathbf{A},\ \mathbf{B}\text{ y }\mathbf{C}.$$

Este es el volumen del paralelepípedo definido por **A**, **B** y **C**. Debemos notar que $\mathbf{A}\cdot\mathbf{B}\times\mathbf{C}$ puede algunas veces volverse negativo. Este problema y su interpretación los consideraremos más adelante.

El producto escalar triple encuentra una interesante e importante aplicación en la construcción de una red recíproca cristalina. Sea **a**, **b** y **c** (no necesariamente perpendiculares entre ellos) vectores que definen una red cristalina. La distancia desde un punto de la red a otro puede ser escrita

$$\mathbf{r}=n_a\mathbf{a}+n_b\mathbf{b}+n_c\mathbf{c}\ , \tag{14.59}$$

con n_a, n_b y n_c tomando los valores sobre los enteros. Con estos vectores podemos formar

$$\mathbf{a}' = \frac{\mathbf{b}\times\mathbf{c}}{\mathbf{a}\cdot\mathbf{b}\times\mathbf{c}} , \quad \mathbf{b}' = \frac{\mathbf{c}\times\mathbf{a}}{\mathbf{a}\cdot\mathbf{b}\times\mathbf{c}} , \quad \mathbf{c}' = \frac{\mathbf{a}\times\mathbf{b}}{\mathbf{a}\cdot\mathbf{b}\times\mathbf{c}} . \tag{14.60}$$

Vemos que $\mathbf{a}'$ es perpendicular al plano que contiene a $\mathbf{b}$ y a $\mathbf{c}$ y tiene una magnitud proporcional a a^{-1}. En efecto, podemos rápidamente mostrar que

$$\mathbf{a}'\cdot\mathbf{a} = \mathbf{b}'\cdot\mathbf{b} = \mathbf{c}'\cdot\mathbf{c} = 1 , \tag{14.61}$$

mientras

$$\mathbf{a}'\cdot\mathbf{b} = \mathbf{a}'\cdot\mathbf{c} = \mathbf{b}'\cdot\mathbf{a} = \mathbf{b}'\cdot\mathbf{c} = \mathbf{c}'\cdot\mathbf{a} = \mathbf{c}'\cdot\mathbf{b} = 0 . \tag{14.62}$$

Esto es a partir de las ecuaciones (14.61) y (14.62) derivamos el nombre de red recíproca. El espacio matemático en el cual esta red recíproca existe es llamado a veces espacio de Fourier. Esta red recíproca es útil en problemas que intervienen el *scattering* de ondas a partir de varios planos en un cristal. Más detalles pueden encontrarse en R.B. Leigthon's *Principles of Modern Physics*, pp. 440-448 [New York: McGraw-Hill (1995)].

14.5.2 Producto vectorial triple.

El segundo producto triple de interés es $\mathbf{A}\times(\mathbf{B}\times\mathbf{C})$, el cual es un vector. Aquí los paréntesis deben mantenerse, como puede verse del caso especial $(\hat{\mathbf{x}}\times\hat{\mathbf{x}})\times\hat{\mathbf{y}} = 0$, mientras que si evaluamos $\hat{\mathbf{x}}\times(\hat{\mathbf{x}}\times\hat{\mathbf{y}}) = \hat{\mathbf{x}}\times\hat{\mathbf{z}} = -\hat{\mathbf{y}}$. El producto vectorial triple es perpendicular a $\mathbf{A}$ y $\mathbf{B}\times\mathbf{C}$. El plano definido por $\mathbf{B}$ y $\mathbf{C}$ es perpendicular a $\mathbf{B}\times\mathbf{C}$ y así el producto triple yace en este plano (ver figura 14.14)

$$\mathbf{A}\times(\mathbf{B}\times\mathbf{C}) = x\mathbf{B} + y\mathbf{C} . \tag{14.63}$$

Multiplicando (14.63) por $\mathbf{A}$, lo que da cero para el lado izquierdo, tal que $x\mathbf{A}\cdot\mathbf{B}+y\mathbf{A}\cdot\mathbf{C} = 0$. De aquí que $x = z\mathbf{A}\cdot\mathbf{C}$ e $y = -z\mathbf{A}\cdot\mathbf{B}$ para un z apropiado. Sustituyendo estos valores en la ecuación (14.63) da

$$\mathbf{A}\times(\mathbf{B}\times\mathbf{C}) = z(\mathbf{B}\mathbf{A}\cdot\mathbf{C} - \mathbf{C}\mathbf{A}\cdot\mathbf{B}); \tag{14.64}$$

deseamos mostrar que $z = 1$ en la ecuación (14.64), una importante relación algunas veces conocida como la regla $BAC - CAB$. Ya que la ecuación (14.64) es lineal en $\mathbf{A}$, $\mathbf{B}$ y $\mathbf{C}$, z es independiente de esas magnitudes. Esto es, solamente necesitamos mostrar que $z = 1$ para vectores unitarios $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$, $\hat{\mathbf{C}}$. Denotemos $\hat{\mathbf{B}}\cdot\hat{\mathbf{C}} = \cos\alpha$, $\hat{\mathbf{C}}\cdot\hat{\mathbf{A}} = \cos\beta$, $\hat{\mathbf{A}}\cdot\hat{\mathbf{B}} = \cos\gamma$, y el cuadrado de la ecuación (14.64) para obtener

$$\begin{aligned}[\hat{\mathbf{A}}\times(\hat{\mathbf{B}}\times\hat{\mathbf{C}})]^2 &= \hat{\mathbf{A}}^2(\hat{\mathbf{B}}\times\hat{\mathbf{C}})^2 - [\hat{\mathbf{A}}\cdot(\hat{\mathbf{B}}\times\hat{\mathbf{C}})]^2 = 1 - \cos^2\alpha - [\hat{\mathbf{A}}\cdot(\hat{\mathbf{B}}\times\hat{\mathbf{C}})]^2 , \\ &= z^2[(\hat{\mathbf{A}}\cdot\hat{\mathbf{C}})^2 + (\hat{\mathbf{A}}\cdot\hat{\mathbf{B}})^2 - 2\hat{\mathbf{A}}\cdot\hat{\mathbf{B}}\hat{\mathbf{A}}\cdot\hat{\mathbf{C}}\hat{\mathbf{B}}\cdot\hat{\mathbf{C}}] , \\ &= z^2(\cos^2\beta + \cos^2\gamma - 2\cos\alpha\cos\beta\cos\gamma) ,\end{aligned} \tag{14.65}$$

usando $(\hat{\mathbf{v}}\times\hat{\mathbf{w}})^2 = \hat{\mathbf{v}}^2\hat{\mathbf{w}}^2 - (\hat{\mathbf{v}}\cdot\hat{\mathbf{w}})^2$ repetidas veces. Consecuentemente, el volumen abarcado por $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$ y $\hat{\mathbf{C}}$ que aparece en la ecuación (14.65) puede ser escrito como

$$[\hat{\mathbf{A}}\cdot(\hat{\mathbf{B}}\times\hat{\mathbf{C}})]^2 = 1 - \cos^2\alpha - z^2(\cos^2\beta + \cos^2\gamma - 2\cos\alpha\cos\beta\cos\gamma) .$$

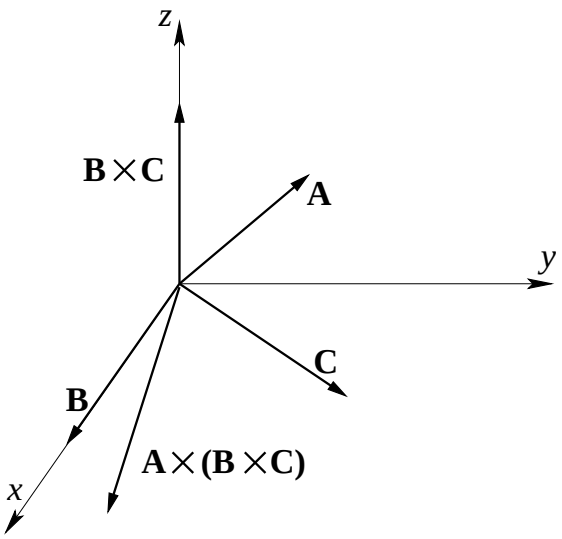


Figura 14.14: Los vectores **B** y **C** están en el plano xy. $\mathbf{B}\times\mathbf{C}$ es perpendicular al plano xy y es mostrado aquí a lo largo del eje z. Entonces $\mathbf{A}\times(\mathbf{B}\times\mathbf{C})$ es perpendicular al eje z y por lo tanto está de regreso en el plano xy.

De aquí $z^2 = 1$ ya que este volumen es simétrico en α, β y γ. Esto es $z = \pm 1$ e independiente de $\hat{\mathbf{A}}$, $\hat{\mathbf{B}}$ y $\hat{\mathbf{C}}$. Usando el caso especial $\hat{\mathbf{x}}\times(\hat{\mathbf{x}}\times\hat{\mathbf{y}}) = -\hat{\mathbf{y}}$ en la ecuación (14.64) finalmente da $z = 1$.

Una derivación alternativa usando el tensor de Levi-Civita ϵ_{ijk},

$$\epsilon_{ijk} = \begin{cases} 1 & \text{para } i,j,k{=}x,y,z \text{ o } y,z,x \text{ y } z,x,y, \\ -1 & \text{para } i,j,k{=}y,x,z \text{ o } x,z,y \text{ y } z,y,x, \\ 0 & \text{en otros casos.} \end{cases}$$

La veremos más adelante.

Podemos notar aquí que los vectores son independientes de las coordenadas tal que una ecuación vectorial es independiente de un particular sistema de coordenadas. El sistema de coordenadas sólo determina las componentes. Si la ecuación vectorial puede ser establecida en coordenadas cartesianas, ella se puede establecer y es válida en cualquier sistema de coordenadas que será introducido en el próximo capítulo.

14.6 Gradiente, ∇

Suponga que $\varphi(x,y,z)$ es una función de punto escalar, esto es, una función cuyo valor depende de los valores de las coordenadas (x,y,z). Como un escalar, debería tener el mismo valor en un punto fijo dado en el espacio, independiente de la rotación de nuestro sistema de coordenadas, o

$$\varphi'(x_1',x_2',x_3') = \varphi(x_1,x_2,x_3)\ . \tag{14.66}$$

Diferenciando con respecto a x_i' obtenemos

$$\frac{\partial\varphi'(x_1',x_2',x_3')}{\partial x_i'}=\frac{\partial\varphi(x_1,x_2,x_3)}{\partial x_i'}=\sum_j\frac{\partial\varphi}{\partial x_j}\frac{\partial x_j}{\partial x_i'}=\sum_j a_{ij}\frac{\partial\varphi}{\partial x_j}\ , \tag{14.67}$$

por las reglas de diferenciación parcial y las ecuaciones (14.16) y (14.17). Pero la comparación con la ecuación (14.18), la ley de transformación vectorial, ahora nos muestra que hemos construido un vector con componentes $\partial\varphi/\partial x_j$. Este vector lo etiquetamos como la gradiente de φ. Un simbolismo conveniente es

$$\boldsymbol{\nabla}\varphi=\hat{\mathbf{x}}\frac{\partial\varphi}{\partial x}+\hat{\mathbf{y}}\frac{\partial\varphi}{\partial y}+\hat{\mathbf{z}}\frac{\partial\varphi}{\partial z} \tag{14.68}$$

o

$$\boldsymbol{\nabla}=\hat{\mathbf{x}}\frac{\partial}{\partial x}+\hat{\mathbf{y}}\frac{\partial}{\partial y}+\hat{\mathbf{z}}\frac{\partial}{\partial z}\ . \tag{14.69}$$

$\boldsymbol{\nabla}\varphi$ (o delφ o gradφ) es nuestro gradiente del campo escalar φ, mientras $\boldsymbol{\nabla}$ es por sí mismo un operador diferencial vectorial (disponible para operar sobre o para diferenciar un campo escalar φ). Todas las relaciones para $\boldsymbol{\nabla}$ pueden ser derivadas a partir de la naturaleza híbrida del gradiente, por un lado la expresión en términos de derivadas parciales y por otra su naturaleza vectorial.

Ejemplo: el gradiente de una función de r.

Calculemos el gradiente de $f(r)=f(\sqrt{x^2+y^2+z^2})$.

$$\boldsymbol{\nabla}f(r)=\hat{\mathbf{x}}\frac{\partial f(r)}{\partial x}+\hat{\mathbf{y}}\frac{\partial f(r)}{\partial y}+\hat{\mathbf{z}}\frac{\partial f(r)}{\partial z}\ .$$

La dependencia de $f(r)$ sobre x es a través de la dependencia de r sobre x. Por lo tanto[10]

$$\frac{\partial f(r)}{\partial x}=\frac{df(r)}{dr}\frac{\partial r}{\partial x}\ .$$

De r como función de x, y, z

$$\frac{\partial r}{\partial x}=\frac{\partial\sqrt{x^2+y^2+z^2}}{\partial x}=\frac{x}{\sqrt{x^2+y^2+z^2}}=\frac{x}{r}\ .$$

Por lo tanto

$$\frac{\partial f(r)}{\partial x}=\frac{df(r)}{dr}\frac{x}{r}\ .$$

[10]Este es un caso especial de la regla de la cadena para derivadas parciales:

$$\frac{\partial f(r,\theta,\varphi)}{\partial x}=\frac{\partial f}{\partial r}\frac{\partial r}{\partial x}+\frac{\partial f}{\partial\theta}\frac{\partial\theta}{\partial x}+\frac{\partial f}{\partial\varphi}\frac{\partial\varphi}{\partial x}\ .$$

Ya que $\partial f/\partial\theta=\partial f/\partial\varphi=0$, $\partial f/\partial r\to df/dr$.

Permutando las cordenadas obtenemos las otras derivadas, para que finalmente podamos escribir

$$\boldsymbol{\nabla} f(r) = (\hat{\mathbf{x}}x + \hat{\mathbf{y}}y + \hat{\mathbf{z}}z)\,\frac{1}{r}\frac{df}{dr} = \frac{\mathbf{r}}{r}\frac{df}{dr} = \hat{\mathbf{r}}\frac{df}{dr}\ .$$

Aquí $\hat{\mathbf{r}}$ es un vector unitario $\mathbf{r}/r$ en la dirección radial positiva. El gradiente de una función de r es un vector en la dirección radial.

14.6.1 Una interpretación geométrica

Una aplicación inmediata de $\boldsymbol{\nabla}\varphi$ es hacer el producto punto contra un diferencial de camino o diferencial de longitud

$$d\mathbf{r} = \hat{\mathbf{x}}dx + \hat{\mathbf{y}}dy + \hat{\mathbf{z}}dz\ . \tag{14.70}$$

Así obtenemos

$$(\boldsymbol{\nabla}\varphi)\cdot d\mathbf{r} = \frac{\partial\varphi}{\partial x}dx + \frac{\partial\varphi}{\partial y}dy + \frac{\partial\varphi}{\partial z}dz = d\varphi\ , \tag{14.71}$$

el cambio en la función escalar φ corresponde al cambio de posición dr. Ahora consideremos P y Q para ser dos puntos sobre una superficie $\varphi(x,y,z) = C$, una constante. Esos puntos son escogidos tal que Q está a un distancia $d\mathbf{r}$ de P. Entonces moviéndose de P a Q, el cambio en $\varphi(x,y,z) = C$ está dado por

$$d\varphi = (\boldsymbol{\nabla}\varphi)\cdot d\mathbf{r} = 0\ , \tag{14.72}$$

ya que permanecemos sobre la superficie $\varphi(x,y,z) = C$. Esto muestra que $\boldsymbol{\nabla}\varphi$ es perpendicular a $d\mathbf{r}$. Ya que $d\mathbf{r}$ puede tener cualquier dirección desde P mientras permanezca en la superficie φ, el punto Q está restringido a la superficie, pero teniendo una dirección arbitraria, $\boldsymbol{\nabla}\varphi$ es normal a la superficie φ=constante (figura 14.15).

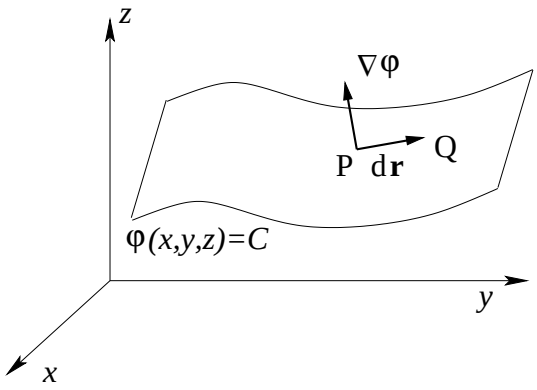


Figura 14.15: Requerimos que el diferencial de longitud $d\mathbf{r}$ permanezca sobre la superficie $\varphi = C$.

Si ahora permitimos que $d\mathbf{r}$ vaya desde la superficie $\varphi = C_1$ a una superficie adyacente $\varphi = C_2$ (figura 14.16),

$$d\varphi = C_2 - C_1 = \Delta C = (\boldsymbol{\nabla}\varphi) \cdot d\mathbf{r} \ . \tag{14.73}$$

Para un $d\varphi$ dado, $|\, d\mathbf{r}\,|$ es un mínimo cuando se escoge paralelo a $\boldsymbol{\nabla}\varphi\,(\cos\theta = 1)$; o, para un $|\, d\mathbf{r}\,|$ dado, el cambio en la función escalar φ es maximizado escogiendo $d\mathbf{r}$ paralelo a $\boldsymbol{\nabla}\varphi$. *Esto identifica $\boldsymbol{\nabla}\varphi$ como un vector que tiene la dirección de la máxima razón espacial de cambio de φ*, una identificación que será útil en el próximo capítulo cuando consideremos sistemas de coordenadas no cartesianos.

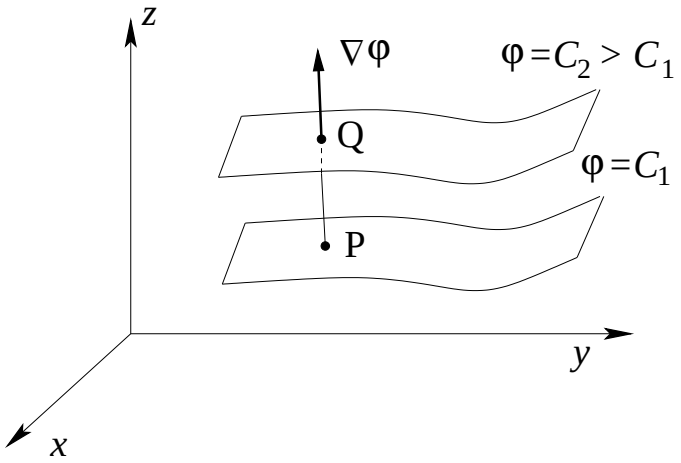


Figura 14.16: Gradiente.

Esta identificación de $\boldsymbol{\nabla}\varphi$ también puede ser desarrollada usando el cálculo de variaciones sujeto a constriciones.

El gradiente de un campo escalar es de extrema importancia en Física al expresar la relación entre un campo de fuerza y un campo potencial.

$$\text{fuerza} = -\nabla(\text{potencial}) \ . \tag{14.74}$$

Esto es ilustrado para ambos campos, gravitacional y electroestático, entre otros. Debemos notar que el signo menos en la ecuación (14.74) viene del hecho que el agua fluye cerro abajo y no cerro arriba.

14.7 Divergencia, $\boldsymbol{\nabla}\cdot$.

Diferenciar una función vectorial es una extensión simple de diferenciación de cantidades escalares. Supongamos que $\mathbf{r}(t)$ describe la posición de un satélite en algún tiempo t. Luego, por diferenciación con respecto al tiempo,

$$\begin{aligned} \frac{d\mathbf{r}(t)}{dt} &= \lim_{\Delta t \to 0} \frac{\mathbf{r}(t+\Delta t) - \mathbf{r}(t)}{\Delta t} \ , \\ &= \mathbf{v} \ , \quad \text{la velocidad lineal.} \end{aligned}$$

Gráficamente, nuevamente tenemos que la pendiente de la curva, órbita, o trayectoria, como se muestra en la figura 14.17.

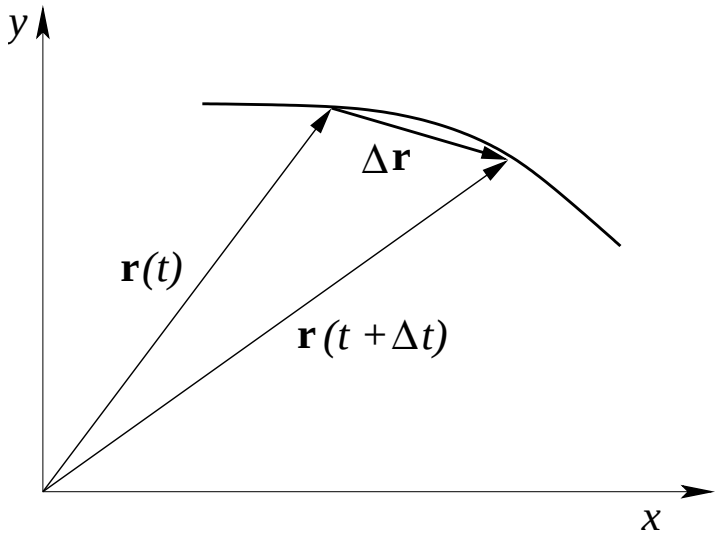


Figura 14.17: Diferenciación de un vector.

Si resolvemos $r(t)$ dentro de las componentes cartesianas, dr/dt siempre reduca directamente a una suma vectorial de no más que tres (para el espacio tridimensional) derivadas escalares. En otro sistema de coordenadas la situación es un poco más complicada, los vectores unitarios no son más constante en dirección. La diferenciación con respecto al eapacio de coordenadas es maneja de la misma manera como la diferenciación con respecto al tiempo, como se veremos en los siguientes párrafos.

En la sección 14.6, ∇ fue definida como un operador vectorial. Ahora, poniendo cuidadosamnente atención a ambas sus propiedades vectoriales y a sus propiedades diferenciales, operemoslo sobre un vector. Primero, como un vector le hacemos producto punto con un segundo vector para obtener

$$\nabla \cdot \mathbf{V} = \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z} \ , \tag{14.75}$$

lo que se conoce como divergencia de $\mathbf{V}$. Este es un escalar, como se discutió en la sección 14.3.

Ejemplo Calculemos $\nabla \cdot \mathbf{r}$

$$\nabla \cdot \mathbf{r} = \left(\hat{\mathbf{x}} \frac{\partial}{\partial x} + \hat{\mathbf{y}} \frac{\partial}{\partial y} + \hat{\mathbf{z}} \frac{\partial}{\partial z} \right) \cdot \left(\hat{\mathbf{x}} x + \hat{\mathbf{y}} y + \hat{\mathbf{z}} z = \frac{\partial x}{\partial x} + \frac{\partial y}{\partial y} + \frac{\partial z}{\partial z} \right) \ ,$$

$$\nabla \cdot \mathbf{r} = 3 \ .$$

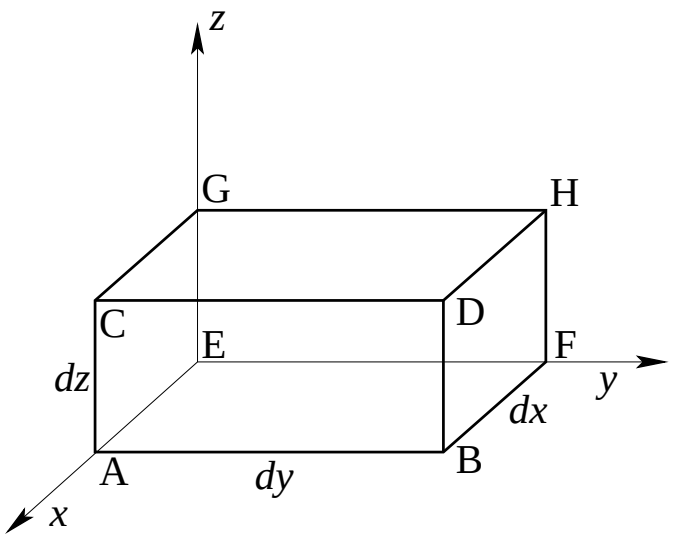


Figura 14.18: Diferencial paralelepípedo rectangular (en el primer octante).

14.7.1 Una interpretación física.

Para desarrollar una intuición del sentido físico de la divergencia, consideramos $\boldsymbol{\nabla}\cdot(\rho\mathbf{v})$ con $\mathbf{v}(x,y,z)$, la velocidad de un fluido compresible $\rho(x,y,z)$, su densidad en el punto (x,y,z). Si consideramos un pequeño volumen dx,dy,dz (figura 14.18), el fluido fluye dentro de su volumen per unidad de tiempo (dirección positiva de x) a través de la cara $EFGH$ es (masa de fuido que sale por unidad de tiempo)$_{EFGH} = \rho v_x|_{x=0}\,dydz$. Los componentes del flujo ρv_y y ρv_z tangencial a esta cara no contribuye en nada al flujo en esa cara. La masa de fluido por unidad de tiempo que pasa (todavía en la dirección positiva de x) a través de la cara $ABCD$ es $\rho v_x|_{x=dx}dydz$. Para comparar estos flujos y para encontrar el flujo neto, expandimos este último resultado en una serie de Maclaurin. Este produce

$$\begin{aligned}\text{(La masa de fluido que sale por unidad de tiempo)}_{ABCD} &= \rho v_x|_{x=dx}\,dydz\ ,\\ &= \left[\rho v_x + \frac{\partial}{\partial x}(\rho v_x)dx\right]_{x=0} dydz\ .\end{aligned}$$

Aquí el término de la derivada es un primer término de corrección permitiendo la posibilidad de no uniformidad de la densidad o velocidad o ambas[11]. El término de orden cero $\rho v_x|_{x=0}$ (correspondiente al flujo uniforme) se cancela.

$$\text{(La masa neta de fluido que sale por unidad de tiempo)}_x = \frac{\partial}{\partial x}(\rho v_x)dxdydz\ .$$

[11]Estríctamente hablando, ρv_x es promediado sobre la cara $EFGH$ y la expresión $\rho v_x + \partial/\partial x(\rho v_x)dx$ es también promediada sobre la cara $ABCD$. Usando un diferencial de volumen arbitrariamente pequeño, encontramos que los promedios se reducen a los valores empleados aquí.

Equivalentemente, podemos llegar a este resultado por

$$\lim_{\Delta x\to 0}\frac{\rho v_x(\Delta x,0,0)-\rho v_x(0,0,0)}{\Delta x}\equiv\left.\frac{\partial \rho v_x(x,y,z)}{\partial x}\right|_{0,0,0}$$

Ahora el eje x no requiere ningún tratamiento preferencial. El resultado precedente para las dos caras perpendiculares al eje x debe mantenerse para las dos caras perpendiculares al eje y, reemplazando x por y y los correspondientes cambios para y y z: $y\to z$, $z\to x$. Esta es una permutación cíclica de las coordenadas. Una ulterior permutación cíclica produce los resultados para las restantes dos caras de nuestro paralelepípedo. Sumando la velocidad de flujo neto para los tres pares de superficies de nuestro elemento de volumen, tenemos

$$\begin{aligned}\text{la masa neta de fluido que sale} &= \left[\frac{\partial}{\partial x}(\rho v_x)+\frac{\partial}{\partial y}(\rho v_y)+\frac{\partial}{\partial z}(\rho v_z)\right]dx\,dy\,dz\\ \text{(por unidad de tiempo)}\\ &=\boldsymbol{\nabla}\cdot(\rho\mathbf{v})\,dx\,dy\,dz\ .\end{aligned}\tag{14.76}$$

Por lo tanto, el flujo neto de nuestro fluido compresible del elemento volumen $dx\,dy\,dz$ per unidad de volumen per unidad de tiempo es $\boldsymbol{\nabla}\cdot(\rho\mathbf{v})$. De aquí el nombre de *divergencia*. Una aplicación directa es en la ecuación de continuidad

$$\frac{\partial\rho}{\partial t}+\boldsymbol{\nabla}\cdot(\rho\mathbf{v})=0\ ,\tag{14.77}$$

la cual simplemente establece que el flujo neto resulta en una disminución de la densidad dentro del volumen. Note que en la ecuación (14.77), ρ es considerada como una función posible del tiempo tanto como del espacio: $\rho(x,y,z,t)$. La divergencia aparece en una amplia variedad de problemas físicos, pasando desde una densidad de corriente de probabilidad en mecánica cuántica, a la pérdida de neutrones en un reactor nuclear.

La combinación $\boldsymbol{\nabla}\cdot(f\mathbf{V})$, en la cual f es una función escalar y $\mathbf{V}$ una función vectorial, puede ser escrita

$$\begin{aligned}\boldsymbol{\nabla}\cdot(f\mathbf{V}) &= \frac{\partial}{\partial x}(fV_x)+\frac{\partial}{\partial y}(fV_y)+\frac{\partial}{\partial z}(fV_z)\\ &=\frac{\partial f}{\partial x}V_x+f\frac{\partial V_x}{\partial x}+\frac{\partial f}{\partial y}V_y+f\frac{\partial V_y}{\partial y}+\frac{\partial f}{\partial z}V_z+f\frac{\partial V_z}{\partial z}\\ &=(\boldsymbol{\nabla}f)\cdot\mathbf{V}+f\boldsymbol{\nabla}\cdot\mathbf{V}\ ,\end{aligned}\tag{14.78}$$

la cual es justo lo que uno debería esperar para la derivada de un producto. Notemos que $\boldsymbol{\nabla}$ como un operador diferencial, diferencia a ambos, f y $\mathbf{V}$; como un vector se le hace producto punto con $\mathbf{V}$ (en cada término).

Si tenemos el caso especial de la divergencia de un campo vectorial igual a cero,

$$\boldsymbol{\nabla}\cdot\mathbf{B}=0\ ,\tag{14.79}$$

el vector $\mathbf{B}$ se dice que es *solenoidal*, el término viene del ejemplo en cual $\mathbf{B}$ es la campo o inducción magnética y la ecuación (14.79) aparece como una de las ecuaciones de Maxwell. Cuando un vector es solenoidal puede ser escrito como el rotor de otro vector conocido como el vector potencial. Más adelante calcularemos tal vector potencial.

14.8 Rotor, $\boldsymbol{\nabla}\times$

Otra posible operación con el operador vectorial $\boldsymbol{\nabla}$ es hacerlo producto cruz contra otro vector. Obtenemos

$$\begin{aligned}\boldsymbol{\nabla}\times\mathbf{V} &= \hat{\mathbf{x}}\left(\frac{\partial}{\partial y}V_z-\frac{\partial}{\partial z}V_y\right)+\hat{\mathbf{y}}\left(\frac{\partial}{\partial z}V_x-\frac{\partial}{\partial x}V_z\right)+\hat{\mathbf{z}}\left(\frac{\partial}{\partial x}V_y-\frac{\partial}{\partial y}V_x\right)\\ &= \begin{vmatrix}\hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}}\\ \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z}\\ V_x & V_y & V_z\end{vmatrix},\end{aligned} \tag{14.80}$$

el cual es llamado el rotor de $\mathbf{V}$. En la expansión de este determinante debemos considerar la naturaleza diferencial de $\boldsymbol{\nabla}$. Específicamente, $\mathbf{V}\times\boldsymbol{\nabla}$ está definido sólo como un operador, otro operador diferencial. Ciertamente no es igual, en general, a $-\boldsymbol{\nabla}\times\mathbf{V}$[12]. En el caso de la ecuación (14.80) el determinante debe ser expandido desde arriba hacia abajo, tal que obtengamos las derivadas correctas. Si hacemos producto cruz contra el producto de un escalar por un vector, podemos ver que

$$\begin{aligned}\boldsymbol{\nabla}\times(f\mathbf{V})\big|_x &= \left[\frac{\partial fV_z}{\partial y}-\frac{\partial fV_y}{\partial z}\right]\\ &= \left(f\frac{\partial V_z}{\partial y}+\frac{\partial f}{\partial y}V_z-f\frac{\partial V_y}{\partial z}+\frac{\partial f}{\partial z}V_y\right)\\ &= f\boldsymbol{\nabla}\times(\mathbf{V})\big|_x+(\boldsymbol{\nabla}f)\times\mathbf{V}\big|_x\,.\end{aligned} \tag{14.81}$$

Si permutamos las coordenadas tenemos

$$\boldsymbol{\nabla}\times(f\mathbf{V})=f\boldsymbol{\nabla}\times(\mathbf{V})+(\boldsymbol{\nabla}f)\times\mathbf{V}\,, \tag{14.82}$$

la cual es el producto vectorial análogo a (14.78). De nuevo, como operador diferencial diferencia a ambos f y $\mathbf{V}$. Como vector hace producto cruz contra $\mathbf{V}$ en cada término.

Ejemplo

Calculemos $\boldsymbol{\nabla}\times\mathbf{r}f(r)$. Por la ecuación (14.82),

$$\boldsymbol{\nabla}\times\mathbf{r}f(r)=f(r)\boldsymbol{\nabla}\times\mathbf{r}+[\boldsymbol{\nabla}f(r)]\times\mathbf{r}\,. \tag{14.83}$$

Primero,

$$\boldsymbol{\nabla}\times\mathbf{r}=\begin{vmatrix}\hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}}\\ \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z}\\ x & y & z\end{vmatrix}=0\,. \tag{14.84}$$

Segundo, usando $\boldsymbol{\nabla}f(r)=\hat{\mathbf{r}}(df/dr)$, obtenemos

$$\boldsymbol{\nabla}\times\mathbf{r}f(r)=\frac{df}{dr}\hat{\mathbf{r}}\times\mathbf{r}\,. \tag{14.85}$$

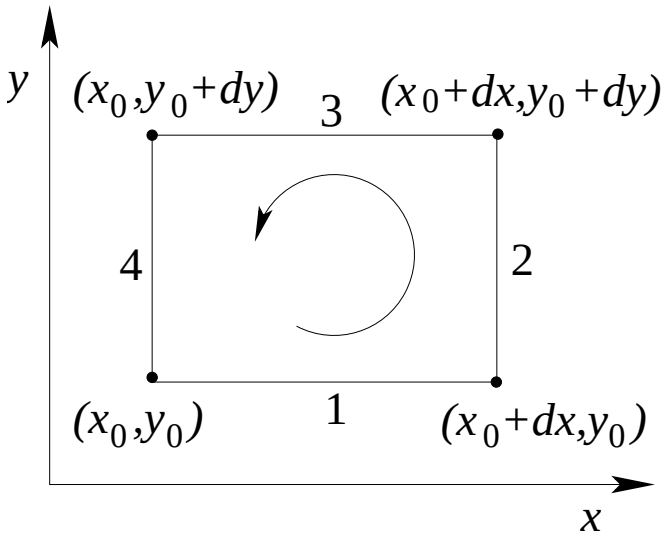


Figura 14.19: Circulación alrededor de un *loop* diferencial.

El producto vectorial se anula, ya que $\mathbf{r} = \hat{\mathbf{r}}\, r$ y $\hat{\mathbf{r}} \times \hat{\mathbf{r}} = 0$.

Para desarrollar un mejor sentido del significado físico del rotor, consideremos la circulación de un fluído alrededor de un *loop* diferencial en el plano xy, figura 14.19.

Aunque la circulación técnicamente está dada por la integral lineal de un vector $\int \mathbf{V} \cdot d\lambda$ (sección 14.10), podemos establecer las integrales escalares equivalentes. Tomemos la circulación como

$$\text{circulación}_{1234} = \int_1 V_x(x,y)d\lambda_x + \int_2 V_y(x,y)d\lambda_y + \int_3 V_x(x,y)d\lambda_x + \int_4 V_y(x,y)d\lambda_y \ . \quad (14.86)$$

Los números 1, 2, 3 y 4 se refieren al segmento lineal enumerado en la figura 14.19. En la primera integral $d\lambda_x = +dx$, pero en la tercera integral $d\lambda_x = -dx$, ya que el tercer segmento de línea es recorrido en la dirección negativa de x. Similarmente, $d\lambda_y = +dy$, para la segunda integral y $-dy$ para la cuarta. Luego, los integrandos están referidos al punto (x_0, y_0) con una expansión de Taylor[13] tomando en cuenta el desplazamiento del segmento de línea 3 desde 1

[12]En este mismo espíritu, si $\mathbf{A}$ es un operador diferencial, no necesariamente es cierto que $\mathbf{A} \times \mathbf{A} = 0$. Específicamente, en mecánica cuántica el *operador* de momento angular, $\mathbf{L} = -i(\mathbf{r} \times \boldsymbol{\nabla})$, encontramos que $\mathbf{L} \times \mathbf{L} = i\mathbf{L}$.

[13]$V_y(x_0 + dx, y_0) = V_y(x_0, y_0) + \left(\frac{\partial V_y}{\partial x}\right)_{x_0 y_0} dx + \cdots$ Los términos de órdenes más altos se van a cero en el límite $dx \to 0$. Un término de corrección por la variación de V_y con y es cancelado por el correspondiente término en la cuarta integral.

y 2 desde 4. Para nuestro segmento de línea diferencial esto tiende a

$$\begin{aligned}\text{ciculación}_{1234} &= V_x(x_0,y_0)dx + \left[V_y(x_0,y_0) + \frac{\partial V_y}{\partial x}dx\right]dy + \\ &+ \left[V_x(x_0,y_0) + \frac{\partial V_x}{\partial y}dy\right](-dx) + V_y(x_0,y_0)(-dy) \\ &= \left(\frac{\partial V_y}{\partial x} - \frac{\partial V_x}{\partial y}\right)dxdy\ .\end{aligned} \tag{14.87}$$

Dividiendo por $dxdy$, tenemos

$$\text{circulación por unidad de área} = \boldsymbol{\nabla}\times\mathbf{V}\big|_z. \tag{14.88}$$

La circulación[14] alrededor de nuestra área diferencial en el plano xy está dada por la componente z de $\boldsymbol{\nabla}\times\mathbf{V}$. En principio, el rotor, $\boldsymbol{\nabla}\times\mathbf{V}$ en (x_0,y_0), podría ser determinado insertando una (diferencial) rueda con paleta dentro del fluido en movimiento en el punto (x_0,y_0). La rotación de la pequeña rueda de paleta podría ser una medida del rotor, y su eje a lo largo de la dirección de $\boldsymbol{\nabla}\times\mathbf{V}$, la cual es perpendicular al plano de circulación.

Usaremos el resultado, ecuación (14.87), en la sección 14.13 para derivar el teorema de Stockes. Cada vez que el rotor de un vector $\mathbf{V}$ se anula,

$$\boldsymbol{\nabla}\times\mathbf{V} = 0\ , \tag{14.89}$$

$\mathbf{V}$ es llamado irrotacional. Los ejemplos físicos más importantes de vectores irrotacionales son las fuerzas gravitacional y electroestática. En cada caso

$$\mathbf{V} = C\frac{\hat{\mathbf{r}}}{r^2} = C\frac{\mathbf{r}}{r^3}\ , \tag{14.90}$$

donde C es una constante y $\hat{\mathbf{r}}$ es un vector unitario hacia afuera en la dirección radial. Para el caso gravitacional tenemos $C \equiv Gm_1m_2$, dado por la ley de Newton de la gravitación universal. Si $C = q_1q_2$, tenemos la ley de Coulomb de la electroestática (unidades cgs). La fuerza $\mathbf{V}$ dada en la ecuación (14.90) puede ser mostrada como irrotacional por expansión directa en las componentes cartesiana como lo hicimos en el ejemplo. Otro acercamiento será desarrollado en el próximo capítulo, en el cual expresamos $\boldsymbol{\nabla}\times$, el rotor, en término de coordenadas polares esféricas. En la sección 14.13 veremos que cada vez que un vector es irrotacional, el vector puede ser escrito como la gradiente (negativa) de un potencial escalar. En la sección 14.15 probaremos que un campo vectorial puede ser resuelto en una parte irrotacional y una parte solenoidal (sujetos a condiciones en infinito). En términos del campo electromagnético esto corresponde a la resolución dentro de un campo eléctrico irrotacional y un campo magnético solenoidal.

Para ondas en un medio elástico, si el desplazamiento $\mathbf{u}$ es irrotacional, $\boldsymbol{\nabla}\times\mathbf{u} = 0$, como ondas planas (u ondas esféricas en distancias grandes) estas llegan a ser longitudinales. Si $\mathbf{u}$ es solenoidal, $\boldsymbol{\nabla}\cdot\mathbf{u} = 0$, luego las ondas llegan a ser transversales. Una perturbación sísmica producirá un desplazamiento que puede ser resuelto en una parte solenoidal y una

[14]En dinámica de fluido $\boldsymbol{\nabla}\times\mathbf{V}$ es llamada la vorticidad.

parte irrotacional (compare la sección 14.15). La parte irrotacional produce la longitudinal P (primaria) de las ondas de terremotos. La parte solenoidal da origen a las ondas tranversales más lentas S (secundarias).

Usando la gradiente, divergencia y rotor y por su puesto la regla $BAC - CAB$, podemos construir o verificar un gran número de útiles identidades vectoriales. Para la verificación, una expansión completa en componentes cartesianas es siempre una posibilidad. Algunas veces si usamos agudeza de ingenio en vez de la pesada rutina de las componentes cartesianas, el proceso de verificación puede ser drásticamente acortado.

Recuerde que $\boldsymbol{\nabla}$ es un operador vectorial, un objeto híbrido que satisface dos conjuntos de reglas:

1. reglas vectoriales, y

2. reglas de diferenciación parcial incluyendo la de diferenciación de un producto.

Ejemplo Verifiquemos que

$$\boldsymbol{\nabla}(\mathbf{A}\cdot\mathbf{B}) = (\mathbf{B}\cdot\boldsymbol{\nabla})\mathbf{A} + (\mathbf{A}\cdot\boldsymbol{\nabla})\mathbf{B} + \mathbf{B}\times(\boldsymbol{\nabla}\times\mathbf{A}) + \mathbf{A}\times(\boldsymbol{\nabla}\times\mathbf{B})\ . \tag{14.91}$$

En este particular ejemplo el factor más importante es reconocer que $\boldsymbol{\nabla}(\mathbf{A}\cdot\mathbf{B})$ es el tipo de términos que aparecen en la expansión $BAC-CAB$ de un producto triple vectorial, ecuación (14.64). Por ejemplo,

$$\mathbf{A}\times(\boldsymbol{\nabla}\times\mathbf{B}) = \boldsymbol{\nabla}(\mathbf{A}\cdot\mathbf{B}) - (\mathbf{A}\cdot\boldsymbol{\nabla})\mathbf{B}\ ,$$

con $\boldsymbol{\nabla}$ diferenciando sólo a $\mathbf{B}$, no a $\mathbf{A}$. De la conmutatividad de los factores en el producto escalar nosotros podemos intercambiar $\mathbf{A}$ con $\mathbf{B}$ y escribimos

$$\mathbf{B}\times(\boldsymbol{\nabla}\times\mathbf{A}) = \boldsymbol{\nabla}(\mathbf{A}\cdot\mathbf{B}) - (\mathbf{B}\cdot\boldsymbol{\nabla})\mathbf{A}\ ,$$

ahora con $\boldsymbol{\nabla}$ diferenciando sólo a $\mathbf{A}$, no a $\mathbf{B}$. Sumando estas ecuaciones, obtenemos $\boldsymbol{\nabla}$ diferenciando al producto $\mathbf{A}\cdot\mathbf{B}$ y la identidad (14.91).

Esta identidad es usada frecuentemente en teoría electromagnética avanzada.

14.9 Aplicaciones sucesivas de ∇.

Hemos definido gradiente, divergencia y rotor para obtener un vector, un escalar y una cantidad vectorial, respectivamente. Usando el operador $\boldsymbol{\nabla}$ sobre cada una de estas cantidades, obtenemos

$$\begin{array}{ccc} \text{(a) } \boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\varphi & \text{(b) } \boldsymbol{\nabla}\times\boldsymbol{\nabla}\varphi & \text{(c)}\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\mathbf{V} \\ \text{(d) } \boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\times\mathbf{V} & \text{(e) } \boldsymbol{\nabla}\times(\boldsymbol{\nabla}\times\mathbf{V})\ , & \end{array}$$

las cinco expresiones involucran una segunda derivada y las cinco aparecen en las ecuaciones diferenciales de segundo orden de la Física Matemática, particularmente en la teoría elctromagnética.

La primera expresión, $\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\varphi$, la divergencia del gradiente, es llamada el Laplaciano de φ. Tenemos

$$\begin{aligned}\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\varphi &= \left(\hat{\mathbf{x}}\frac{\partial}{\partial x}+\hat{\mathbf{y}}\frac{\partial}{\partial y}+\hat{\mathbf{z}}\frac{\partial}{\partial z}\right)\cdot\left(\hat{\mathbf{x}}\frac{\partial\varphi}{\partial x}+\hat{\mathbf{y}}\frac{\partial\varphi}{\partial y}+\hat{\mathbf{z}}\frac{\partial}{\partial z}\right)\\ &= \frac{\partial^2\varphi}{\partial x^2}+\frac{\partial^2\varphi}{\partial y^2}+\frac{\partial^2\varphi}{\partial z^2}\ .\end{aligned} \tag{14.92}$$

Cuando φ es el potencial electroestático, tenemos

$$\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\varphi=\nabla^2\varphi=0\ . \tag{14.93}$$

la cual es la ecuación de Laplace de la electroestática. A menudo la combinación $\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}$ es escrita como $\boldsymbol{\nabla}^2$.

la expresión (b) puede ser escrita

$$\boldsymbol{\nabla}\times\boldsymbol{\nabla}\varphi=\begin{vmatrix}\hat{\mathbf{x}} & \hat{\mathbf{y}} & \hat{\mathbf{z}}\\ \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z}\\ \dfrac{\partial\varphi}{\partial x} & \dfrac{\partial\varphi}{\partial y} & \dfrac{\partial\varphi}{\partial z}\end{vmatrix}\ .$$

Expandiendo el determinante, obtenemos

$$\boldsymbol{\nabla}\times\boldsymbol{\nabla}\varphi=\hat{\mathbf{x}}\left(\frac{\partial^2\varphi}{\partial y\partial z}-\frac{\partial^2\varphi}{\partial z\partial y}\right)+\hat{\mathbf{y}}\left(\frac{\partial^2\varphi}{\partial z\partial x}-\frac{\partial^2\varphi}{\partial x\partial z}\right)+\hat{\mathbf{z}}\left(\frac{\partial^2\varphi}{\partial x\partial y}-\frac{\partial^2\varphi}{\partial y\partial x}\right)=0\ , \tag{14.94}$$

suponiendo que el orden de las derivadas parciales puede ser intercambiado. Esto es cierto ya que estas segundas derivadas parciales de φ son funciones continuas.

Luego, de la ecuación (14.94), el rotor de un gradiente es idénticamente nulo. Todos los gradientes, por lo tanto, son irrotacionales. Note cuidadosamente que el cero en la ecuación (14.94) se vuelve una identidad matemática, independiente de cualquier Física. El cero en la ecuación (14.93) es una consecuencia de la Física.

La expresión (d) es un producto escalar triple el cual puede ser escrito

$$\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\times\mathbf{V}=\begin{vmatrix}\dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z}\\ \dfrac{\partial}{\partial x} & \dfrac{\partial}{\partial y} & \dfrac{\partial}{\partial z}\\ V_x & V_y & V_z\end{vmatrix}\ . \tag{14.95}$$

De nuevo, suponiendo continuidad tal que el orden de la diferenciación es irrelevante, obtenemos

$$\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\times\mathbf{V}=0\ . \tag{14.96}$$

La divergencia de un rotor se anula para todos los rotores que sean solenoidales. En la sección 14.15 veremos que los vectores pueden ser resueltos dentro de una parte solenoidal y otra irrotacional por el teorema de Helmholtz.

Las dos expresiones remanentes satisfacen una relación

$$\boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \mathbf{V}) = \boldsymbol{\nabla}\boldsymbol{\nabla} \cdot \mathbf{V} - \boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\mathbf{V} \ . \tag{14.97}$$

Esto se deduce inmediatamente a partir de la ecuación (14.64), la regla $BAC - CAB$, la cual reescribimos tal que C aparece en el extremo derecho de cada término. El término $\boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\mathbf{V}$ no fue incluido en nuestra lista, pero puede ser definido por la ecuación (14.97).

Ejemplo Una importante aplicación de estas relaciones vectoriales es la derivación de la ecuación de ondas electromagnética. En el vacío las ecuaciones de Maxwell llegan a ser

$$\boldsymbol{\nabla} \cdot \mathbf{B} = 0 \ , \tag{14.98}$$

$$\boldsymbol{\nabla} \cdot \mathbf{E} = 0 \ , \tag{14.99}$$

$$\boldsymbol{\nabla} \times \mathbf{B} = \ \frac{1}{c}\frac{\partial \mathbf{E}}{\partial t} \ , \tag{14.100}$$

$$\boldsymbol{\nabla} \times \mathbf{E} = -\frac{1}{c}\frac{\partial \mathbf{B}}{\partial t} \ , \tag{14.101}$$

Aquí $\mathbf{E}$ es el campo eléctrico, $\mathbf{B}$ es el campo magnético c la velocidad de la luz en el vacío (unidades cgs). Supongamos que eliminamos $\mathbf{B}$ de las ecuaciones (14.100) y (14.101). Podemos hacer esto tomando rotor a ambos lados de (14.101) y la derivada temporal a ambos lados de (14.100). Ya que las derivadas espaciales y temporales conmutan

$$\frac{\partial}{\partial t}\boldsymbol{\nabla} \times \mathbf{B} = \boldsymbol{\nabla} \times \frac{\partial \mathbf{B}}{\partial t} \ , \tag{14.102}$$

y obtenemos

$$\boldsymbol{\nabla} \times (\boldsymbol{\nabla} \times \mathbf{E}) = -\frac{1}{c^2}\frac{\partial^2 \mathbf{E}}{\partial t^2} \ . \tag{14.103}$$

Aplicando las ecuaciones (14.97) y (14.99) se produce

$$\nabla^2 \mathbf{E} = \frac{1}{c^2}\frac{\partial^2 \mathbf{E}}{\partial t^2} \ , \tag{14.104}$$

que es la ecuación de onda para el vector electromagnético. De nuevo, si $\mathbf{E}$ es expresado en coordenadas cartesianas, la ecuación (14.104) se separa en tres ecuaciones escalares.

14.10 Integración vectorial.

El siguiente paso después de la derivación vectorial es integrarlos. Comencemos con las integrales de línea y luego procederemos con las integrales de superficie y de volumen. En cada caso el método de ataque será reducir la integral vectorial a un o varias integrales escalares con las cuales supuestamente estamos más familiarizado.

14.10.1 Integrales lineales.

Usando un incremento de longitud $d\mathbf{r} = \hat{\mathbf{x}}dx + \hat{\mathbf{y}}dy + \hat{\mathbf{z}}dz$, podemos encontrar las integrales de línea

$$\int_c \varphi d\mathbf{r} \ , \tag{14.105}$$

$$\int_c \mathbf{V} \cdot d\mathbf{r} \ , \tag{14.106}$$

$$\int_c \mathbf{V} \times d\mathbf{r} \ , \tag{14.107}$$

en cada una de las cuales la integral está sobre algún contorno C que puede ser abierto (con un punto inicial y un punto final separados) o cerrado (formando un *loop*). Dada la interpretación física que sigue, la segunda forma, ecuación (14.106), es lejos la más importante de las tres.

Con φ, un escalar, la primera integral se reduce a

$$\int_c \varphi \, d\mathbf{r} = \hat{\mathbf{x}} \int_c \varphi(x,y,z)\, dx + \hat{\mathbf{y}} \int_c \varphi(x,y,z)\, dy + \hat{\mathbf{z}} \int_c \varphi(x,y,z)\, dz \ . \tag{14.108}$$

Esta separación ha empleado la relación

$$\int \hat{\mathbf{x}}\, \varphi \, dx = \hat{\mathbf{x}} \int \varphi dx \ , \tag{14.109}$$

la cual es permisible por que los vectores unitarios cartesianos $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, $\hat{\mathbf{z}}$ son constantes en magnitud y dirección. Quizás esta relación es obvia aquí, pero no será verdadera en los sistemas no cartesianos encontrados en el siguiente capítulo.

Las tres integrales sobre el lado derecho de la ecuación (14.108) son integrales escalares ordinarias. Note, sin embargo, que la integral con respecto a x no puede ser evaluada a menos que y y z sean conocidos en términos de x y similarmente para las integrales con respecto a y y z. Esto simplemente significa que el camino de integración C debe ser especificado. A menos que el integrando tenga propiedades especiales que lleve a la integral a depender solamente de los valores de los puntos extremos, el valor dependerá de la elección particular del contorno de C. Por ejemplo, si escogemos el caso muy especial de $\varphi = 1$, la ecuación (14.108) es sólo la distancia vectorial desde el comienzo del contorno C al punto final, en este caso es independiente de la elección del camino que conecta los puntos extremos. Con $d\mathbf{r} = \hat{\mathbf{x}}dx + \hat{\mathbf{y}}dy + \hat{\mathbf{z}}dz$, las segunda y tercera formas también se reducen a una integral escalar y, como la ecuación (14.105), son dependientes, en general, de la elección del camino. La forma (ecuación (14.106)) es exactamente la misma, como la encontrada cuando calculamos el trabajo hecho por una fuerza que varía a lo largo del camino,

$$\begin{aligned} W &= \int \mathbf{F} \cdot d\mathbf{r} \\ &= \int F_x(x,y,z)dx + \int F_y(x,y,z)dy + \int F_z(x,y,z)dz \ . \end{aligned} \tag{14.110}$$

En esta expresión $\mathbf{F}$ es la fuerza ejercida sobre una partícula.

14.10.2 Integrales de superficie.

Las integrales de superficie aparecen en la misma forma que las integrales de línea, el elemento de área también es un vector, $d\mathbf{a}$. A menudo este elemento de área es escrito $\hat{\mathbf{n}}dA$ en el cual $\hat{\mathbf{n}}$ es un vector unitario (normal) para indicar la dirección positiva. Hay dos convenciones para escoger la dirección positiva. Primero, si la superficie es una superficie cerrada, concordamos en tomar hacia afuera como positivo. Segundo, si la superficie es abierta, la normal positiva depende de la dirección en la cual el perímetro de la superficie abierta es recorrido. Si los dedos de la mano derecha están colocados en la dirección de viaje alrededor del perímetro, la normal positiva está indicada por el pulgar de la mano derecha. Como una ilustración, un círculo en el plano xy (figura 14.23) recorrido de x a y a, $-x$ a $-y$ y de regreso a x tendrá su normal positiva paralela al eje z positivo (por el sistema de coordenadas de la mano derecha). Si encontraran superficies de un lado, tal como las bandas de Moebius, se sugiere que se corten las bandas y formen superficies bien comportadas o las etiqueten de patológicas y las envíen al departamento de Matemáticas más cercano.

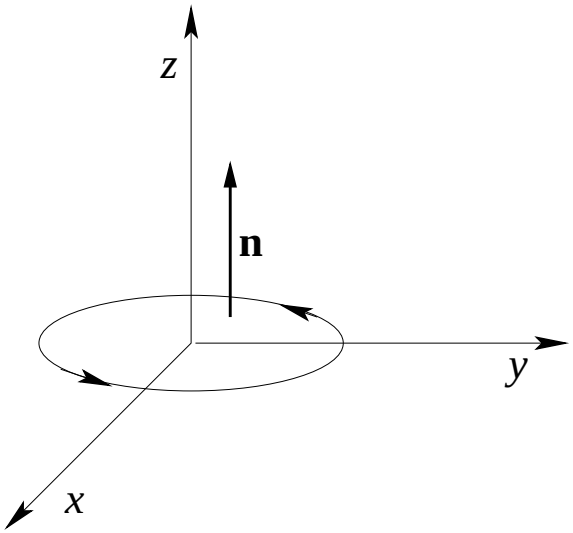


Figura 14.20: Regla de la mano derecha para la normal positiva.

Análogo a las integrales de línea, ecuaciones (14.105-14.107), las integrales de superficie pueden aparecer en las formas

$$\int \varphi d\sigma \ , \qquad \int \mathbf{V} \cdot d\sigma \ , \qquad \int \mathbf{V} \times d\sigma \ .$$

Nuevamente, el producto punto es lejos la forma más comunmente encontrada.

La integral de superficie $\int \mathbf{V} \cdot d\sigma$ puede ser interpretada como un flujo a través de la superficie dada. Esto es realmente lo que hacemos en la sección 14.7 para obtener el significado del término divergencia. Esta identificación reaparece en la sección 14.11 como el teorema de Gauss. Note que en ambos, físicamente y desde el punto de vista del producto punto, la componente tangencial de la velocidad no contribuye en nada al flujo a través de la superficie.

14.10.3 Integrales de volumen.

Las integrales de volumen son algo muy simple, porque el elemento de volumen $d\tau$ es una cantidad escalar[15]. Tenemos

$$\int_V \mathbf{V} d\tau = \hat{\mathbf{x}} \int_V V_x d\tau + \hat{\mathbf{y}} \int_V V_y d\tau + \hat{\mathbf{z}} \int_V V_z d\tau \ , \tag{14.111}$$

de nuevo reduciendo la integral vectorial a una suma vectorial de integrales escalares.

14.10.4 Definiciones integrales de gradiente, divergencia y rotor.

Una interesante y significativa aplicación de nuestras integrales de volumen y de superficie es su uso en el desarrollo de definiciones alternativas de nuestras relaciones diferenciales. Encontramos

$$\boldsymbol{\nabla}\varphi = \lim_{\int d\tau \to 0} \frac{\int \varphi d\sigma}{\int d\tau} \ , \tag{14.112}$$

$$\boldsymbol{\nabla} \cdot \mathbf{V} = \lim_{\int d\tau \to 0} \frac{\int \mathbf{V} \cdot d\sigma}{\int d\tau} \ , \tag{14.113}$$

$$\boldsymbol{\nabla} \times \mathbf{V} = \lim_{\int d\tau \to 0} \frac{\int d\sigma \times \mathbf{V}}{\int d\tau} \ . \tag{14.114}$$

En estas tres ecuaciones $\int d\tau$ es el volumen de una pequeña región del espacio y $d\sigma$ es el elemento de área vectorial de ese volumen. La identificación de la ecuación (14.113) como la divergencia de $\mathbf{V}$ fue realizada en la sección 14.7. Aquí mostramos que la ecuación (14.112) es consistente con nuestra definición inicial de $\boldsymbol{\nabla}\varphi$ (ecuación (14.68)). Por simplicidad escogemos $d\tau$ como la diferencial de volumen $dxdydz$ (figura 14.21). Esta vez colocamos el origen en el centro geométrico de nuestro elemento de volumen. La integral de área tiende a seis integrales, una por cada una de las seis caras. Recordando que $d\sigma$ es hacia afuera, $d\sigma \cdot \hat{\mathbf{x}} = -\,|\,d\sigma\,|$ para la superficie $EFHG$, y $+\,|\,d\sigma\,|$ para la superficie $ABCD$, tenemos

$$\begin{aligned}\int \varphi d\sigma = &-\hat{\mathbf{x}} \int_{EFGH} \left(\varphi - \frac{\partial\varphi}{\partial x}\frac{dx}{2}\right) dydz + \hat{\mathbf{x}} \int_{ABDC} \left(\varphi + \frac{\partial\varphi}{\partial x}\frac{dx}{2}\right) dydz - \\ &- \hat{\mathbf{y}} \int_{AEGC} \left(\varphi - \frac{\partial\varphi}{\partial y}\frac{dy}{2}\right) dxdz + +\hat{\mathbf{y}} \int_{BFHD} \left(\varphi - \frac{\partial\varphi}{\partial y}\frac{dy}{2}\right) dxdz - \\ &- \hat{\mathbf{z}} \int_{ABFE} \left(\varphi - \frac{\partial\varphi}{\partial z}\frac{dz}{2}\right) dxdy + \hat{\mathbf{z}} \int_{CDHG} \left(\varphi - \frac{\partial\varphi}{\partial z}\frac{dz}{2}\right) dxdy \ .\end{aligned}$$

Usando los dos primeros términos de la expansión de Maclaurin, evaluamos cada integrando en el origen con una corrección incluída para corregir por el desplazamiento ($\pm dx/2$, etc.)

[15] Frecuentemente se denota d^3r o d^3x al elemento de volumen.

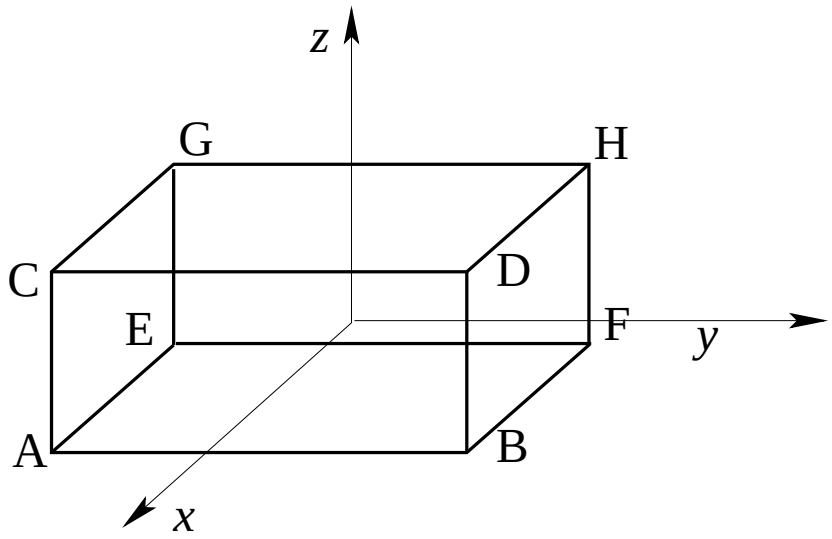


Figura 14.21: Paralelepípedo rectangular diferencial con el origen en el centro.

del centro de la cara desde el origen. Habiendo escogido el volumen total como el tamaño diferencial ($\int d\tau = dxdydz$), obtenemos

$$\int \varphi d\sigma = \left(\hat{\mathbf{x}} \frac{\partial \varphi}{\partial x} + \hat{\mathbf{y}} \frac{\partial \varphi}{\partial y} + \hat{\mathbf{z}} \frac{\partial \varphi}{\partial z} \right) \ . \tag{14.115}$$

Dividiendo por

$$\int d\tau = dxdydz \ ,$$

verificamos la ecuación (14.112).

Esta verificación ha sido sobre simplificada al ignorar otros términos más allá de las primeras derivadas. Estos términos adicionales, los cuales son introducidos más adelante cuando la expansión de Taylor es desarrollada, se anulan en el límite

$$\int d\tau \to 0 \quad (dx \to 0 \ , \ dy \to 0 \ , \ dz \to 0) \ .$$

Esto, por supuesto, es la razón especificada en las ecuaciones (14.112), (14.113) y (14.114), que esos límites son tomados.

La verificación de la ecuación (14.114) sigue esta misma línea exactamente, usando un volumen diferencial $dxdydz$.

14.11 Teorema de Gauss.

Aquí derivamos una útil relación entre una integral de superficie de un vector y la integral de volumen de la divergencia de ese vector. Supongamos que el vector $\mathbf{V}$ y su primera derivada

son continuas en la región de interés. Luego el teorema de Gauss establece que

$$\int_S \mathbf{V} \cdot d\sigma = \int_V \boldsymbol{\nabla} \cdot \mathbf{V}\, d\tau \ . \tag{14.116}$$

En otras palabras, la integral de superficie de un vector sobre una superficie cerrada es igual a la integral de volumen de la divergencia del vector integrada sobre el volumen encerrado definido por la superficie. Imaginemos que el volumen V lo subdividimos en un gran número

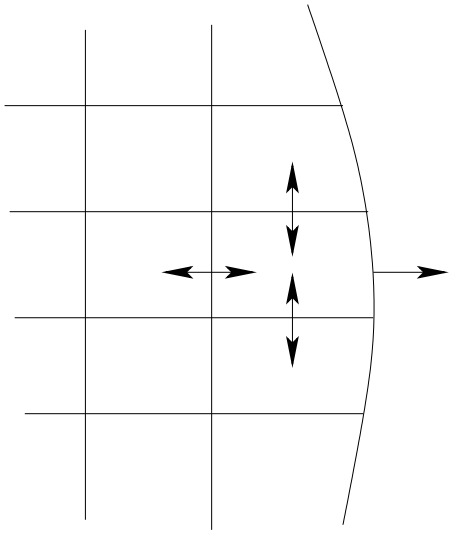


Figura 14.22: Cancelación exacta de los $d\sigma$ sobre las superficies interiores. Sobre la superficie externa no hay cancelación.

de pequeños paralelepípedos (diferenciales). Para cada uno de los paralelepípedos

$$\sum_{\text{seis superficies}} \mathbf{V} \cdot d\sigma = \boldsymbol{\nabla} \cdot \mathbf{V}\, d\tau \ , \tag{14.117}$$

del análisis de la sección 14.7, la ecuación (14.76), con $\rho\mathbf{v}$ reemplazado por $\mathbf{V}$. La suma es sobre las seis caras del paralelepípedo. Sumando sobre todos los paralelepípedos, encontramos que el término $\mathbf{V}\cdot d\sigma$ se cancela para todas las caras internas; solamente las contribuciones de las superficies externas sobreviven (figura 14.22). Análoga a la definición de una integral de Riemann como el límite de una suma, tomamos el límite sobre el número de paralelepípedos tendiendo a infinito ($\to \infty$) y las dimensiones de cada uno tendiendo a cero ($\to 0$).

$$\begin{array}{ccc} \displaystyle\sum_{\text{superficies externas}} \mathbf{V} \cdot d\sigma = \displaystyle\sum_{\text{volúmenes}} \boldsymbol{\nabla} \cdot \mathbf{V}\, d\tau \\ \downarrow \qquad\qquad \downarrow \\ \displaystyle\int_S \mathbf{V} \cdot d\sigma = \int_V \boldsymbol{\nabla} \cdot \mathbf{V}\, d\tau \end{array}$$

El resultado es la ecuación (14.116), el teorema de Gauss.

A partir de un punto de vista físico la ecuación (14.76) ha establecido $\boldsymbol{\nabla}\cdot\mathbf{V}$ como el flujo neto de fluido por unidad de volumen. Luego la integral de volumen da el flujo neto total. Pero la integral de superficie $\int \mathbf{V}\cdot d\sigma$ es solo otra manera de expresar esta misma cantidad, igualando ambas obtenemos el teorema de Gauss.

14.11.1 Teorema de Green.

Un corolario útil del teorema de Gauss es una relación conocida como teorema de Green. Si u y v son dos funciones escalares, tenemos las identidades

$$\boldsymbol{\nabla}\cdot(u\boldsymbol{\nabla}v) = u\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}v + (\boldsymbol{\nabla}u)\cdot(\boldsymbol{\nabla}v)\ , \tag{14.118}$$

$$\boldsymbol{\nabla}\cdot(v\boldsymbol{\nabla}u) = v\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}u + (\boldsymbol{\nabla}v)\cdot(\boldsymbol{\nabla}u)\ . \tag{14.119}$$

Sustrayendo la ecuación (14.119) de la (14.118) e integrando sobre un volumen (suponiendo las derivadas de u y v continuas), y aplicando la ecuación (14.116) (teorema de Gauss), obtenemos

$$\int_V (u\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}v - v\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}u)\,d\tau = \int_S (u\boldsymbol{\nabla}v - v\boldsymbol{\nabla}u)\cdot d\sigma\ . \tag{14.120}$$

Este es el teorema de Green. Una forma alternativa del teorema de Green derivada de la ecuación (14.118) es

$$\int_S u\boldsymbol{\nabla}v\cdot d\sigma = \int_V u\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}v\,d\tau + \int_V \boldsymbol{\nabla}u\cdot\boldsymbol{\nabla}v\,d\tau\ . \tag{14.121}$$

Esta es otra forma del teorema de Green.

14.11.2 Forma alternativa del Teorema de Gauss.

Aunque la ecuación (14.116) involucra la divergencia es con mucho la forma más usada del teorema de Gauss, las integrales de volumen también podrían involucrar el gradiente o el rotor. Suponga

$$\mathbf{V}(x,y,z) = V(x,y,z)\,\mathbf{a}, \tag{14.122}$$

en el cual $\mathbf{a}$ es un vector con una magnitud constante y de dirección constante pero arbitraria. (se puede elegir la dirección, pero una vez que se escoge hay que mantenerla fija). La ecuación (14.116) se convierte en

$$\begin{aligned}\mathbf{a}\cdot\int_S V d\sigma &= \int_V \boldsymbol{\nabla}\cdot\mathbf{a}V\,d\tau \\ &= \mathbf{a}\cdot\int_V \boldsymbol{\nabla}V\,d\tau\end{aligned} \tag{14.123}$$

usando la ecuación (14.78). Esta puede ser reescrita

$$\mathbf{a}\cdot\left[\int_S V d\sigma - \int_V \boldsymbol{\nabla}V\,d\tau\right] = 0\ . \tag{14.124}$$

Ya que $|\mathbf{a}| \neq 0$ y su dirección es arbitraria, significa que el coseno del ángulo sustentado entre ambos vectores no siempre se anula, lo cual implica que el término en paréntesis debe ser nulo. El resultado es

$$\int_S V d\boldsymbol{\sigma} = \int_V \boldsymbol{\nabla} V \, d\tau \ . \tag{14.125}$$

En una manera similar, usando $\mathbf{V} = \mathbf{a} \times \mathbf{P}$ en el cual $\mathbf{a}$ es un vector constante, podemos mostrar

$$\int_S d\sigma \times \mathbf{P} = \int_V \boldsymbol{\nabla} \times P \, d\tau \ . \tag{14.126}$$

Estas dos últimas formas del teorema de Gauss son usadas en la forma vectorial de la teoría de difracción de Kirchoff. También pueden ser usadas para verificar las ecuaciones (14.112) y (14.114).

14.12 El Teorema de Stokes.

El teorema de Gauss relaciona la interal de volumen de una derivada de una función con una integral de una función sobre la superficie cerrada que rodea el volumen. Aquí consideramos una relación análoga entre la integral de superficie de una derivada de una función y la integral de línea de la función, el camino de integración es el perímetro que rodea la superficie. Tomemos la superficie y subdividámosla en una red de rectángulos arbitrariamente pequeños. En la sección 14.8 mostramos que la circulación alrededor de tales rectángulos diferenciales (en el plano xy) es $\boldsymbol{\nabla} \times \mathbf{V}|_2 dxdy$. De la ecuación (14.87) aplicada a un rectángulo diferencial

$$\sum \mathbf{V} \cdot d\boldsymbol{\lambda} = \boldsymbol{\nabla} \times \mathbf{V} \cdot d\boldsymbol{\sigma} \ . \tag{14.127}$$

Sumamos sobre todos los pequeños rectángulos como en la definición de una integral de Riemann. Las constribuciones superficiales (lado derecho de la ecuación (14.127)) son sumadas juntas. Las integrales de línea (lado izquierdo de la ecuación (14.127)) sobre todos los segmentos internos se cancelan. Solamente la integral de línea alrededor del perímetro sobreviven (figura 14.23). Tomando el límite usual con el número de rectángulos tendiendo a infinito mientras $dx \to 0$, $dy \to 0$, tenemos

$$\begin{array}{ccc} \displaystyle\sum_{\text{segmentos externos}} \mathbf{V} \cdot d\boldsymbol{\lambda} = & \displaystyle\sum_{\text{rectángulos}} \boldsymbol{\nabla} \times \mathbf{V} \cdot d\boldsymbol{\sigma} \\ \downarrow \quad = & \downarrow \\ \oint \mathbf{V} \cdot d\boldsymbol{\lambda} = & \displaystyle\int_S \boldsymbol{\nabla} \times \mathbf{V} \cdot d\boldsymbol{\sigma} \ . \end{array} \tag{14.128}$$

Este es el teorema de Stokes. La integral de superficie de la derecha es sobre la superficie encerrada por el perímetro o contorno de la integral de línea de la izquierda. La dirección del vector que representa el área que es hacia afuera del plano del papel si la dirección en que se recorre el contorno para la integral de línea es en el sentido matemático positivo como se muestra en la figura 1.25.

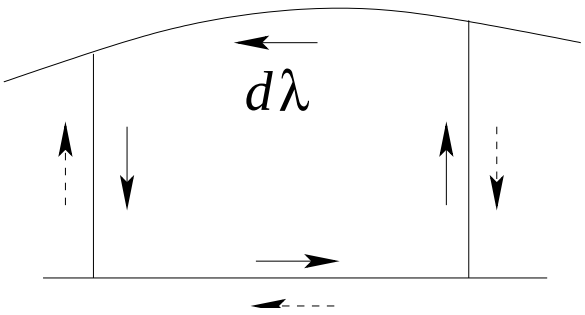


Figura 14.23: Cancelación exacta de los caminos interiores. Sobre el perímetro externo no hay cancelación.

Esta demostración del teorema de Stokes está limitada por el hecho de que usamos una expansión de Maclaurin de $\mathbf{V}(x, y, z)$ para establecer la ecuación (14.87) en la sección 14.8. Realmente sólo necesitamos pedir que el rotor de $\mathbf{V}(x, y, z)$ exista y que sea integrable sobre la superficie.

El teorema de Stokes obviamente se aplica a una superficie abierta. Es posible considerar una superficie cerrada como un caso límite de una superficie abierta con la abertura (y por lo tanto el perímetro) contraída a cero.

14.12.1 Forma alternativa del Teorema de Stokes.

Como con el teorema de Gauss, otras relaciones entre superficies e integrales de línea son posibles. Encontramos

$$\int_S d\boldsymbol{\sigma} \times \boldsymbol{\nabla}\varphi = \oint \varphi d\boldsymbol{\lambda} \tag{14.129}$$

y

$$\int_S (d\boldsymbol{\sigma} \times \boldsymbol{\nabla}) \times \mathbf{P} = \oint d\boldsymbol{\lambda} \times \mathbf{P} \ . \tag{14.130}$$

La ecuación (14.129) puede ser verificada rápidamente sustituyendo $\mathbf{V} = \mathbf{a}\,\varphi$ en la cual $\mathbf{a}$ es un vector de magnitud constante y de dirección constante, como en la sección 14.11. Sustituyendo dentro del teorema de Stokes, la ecuación (14.128)

$$\begin{aligned} \int_S (\boldsymbol{\nabla} \times \mathbf{a}\varphi) \cdot d\boldsymbol{\sigma} &= -\int_S \mathbf{a} \times \boldsymbol{\nabla}\varphi \cdot d\boldsymbol{\sigma} \\ &= -\mathbf{a} \cdot \int_S \boldsymbol{\nabla}\varphi \times d\boldsymbol{\sigma} \ . \end{aligned} \tag{14.131}$$

Para la integral de línea

$$\oint \mathbf{a}\varphi \cdot d\boldsymbol{\lambda} = \mathbf{a} \cdot \oint \varphi \, d\boldsymbol{\lambda} \ , \tag{14.132}$$

obtenemos

$$\mathbf{a}\cdot\left(\oint \mathbf{a}\varphi\cdot d\boldsymbol{\lambda}+\int_S \boldsymbol{\nabla}\varphi\times d\boldsymbol{\sigma}\right)=0\ . \tag{14.133}$$

Ya que la elección de la dirección de $\mathbf{a}$ es arbitraria, la expresión entre paréntesis debe anularse, esto verifica la ecuación (14.129). La ecuación (14.130) puede ser derivada similarmente usando $\mathbf{V}=\mathbf{a}\times\mathbf{P}$, en el cual $\mathbf{a}$ nuevamente es un vector constante.

Ambos teoremas el de Stokes y de Gauss son de tremenda importancia en una variedad de problemas que involucran cálculo vectorial.

14.13 Teoría potencial.

14.13.1 Potencial escalar.

Si una fuerza sobre una región dada del espacio S puede ser expresada como el gradiente negativo de una función escalar φ,

$$\mathbf{F}=-\boldsymbol{\nabla}\varphi\ , \tag{14.134}$$

llamamos φ a un potencial escalar el cual describe la fuerza por una función en vez de tres. Un potencial escalar está determinado salvo una constante aditiva la cual puede ser usada para ajustar su origen. La fuerza $\mathbf{F}$ presentada como el gradiente negativo de un potencial escalar, mono valuado, es etiquetada como una fuerza conservativa. Deseamos saber cuándo existe una función de potencial escalar. Para contestar esta pregunta establezcamos otras dos relaciones equivalentes a la ecuación (14.134). Estas son

$$\boldsymbol{\nabla}\times\mathbf{F}=0 \tag{14.135}$$

y

$$\oint \mathbf{F}\cdot d\mathbf{r}=0\ , \tag{14.136}$$

para todo camino cerrado en nuestra región S. Procedemos a mostrar que cada una de esas ecuaciones implica las otras dos.

Comencemos con

$$\mathbf{F}=-\boldsymbol{\nabla}\varphi\ . \tag{14.137}$$

Entonces

$$\boldsymbol{\nabla}\times\mathbf{F}=-\boldsymbol{\nabla}\times\boldsymbol{\nabla}\varphi=0 \tag{14.138}$$

por las ecuaciones (14.94) o (14.18) implica la ecuación (14.135). Volviendo a la integral de línea, tenemos

$$\oint \mathbf{F}\cdot d\mathbf{r}=-\oint \boldsymbol{\nabla}\varphi\cdot d\mathbf{r}=-\oint d\varphi\ , \tag{14.139}$$

usando la ecuación (14.71). Al integrar $d\varphi$ nos da φ. Ya que hemos especificado un *loop* cerrado, los puntos extremos coinciden y obtenemos cero para cada camino cerrado en nuestra región S por lo cual la ecuación (14.134) se mantiene. Es importante notar aquí la restricción de que el potencial sea univaluado y que la ecuación (14.134) se mantenga para todos los puntos sobre S. Este problema puede presentarse al usar un potencial escalar magnético, un procedimiento perfectamente válido mientras el camino no rodee corriente neta. Tan pronto como escojamos un camino que rodee una corriente neta, el potencial magnético escalar cesa de ser mono valuado y nuestro análisis no tiene mayor aplicación.

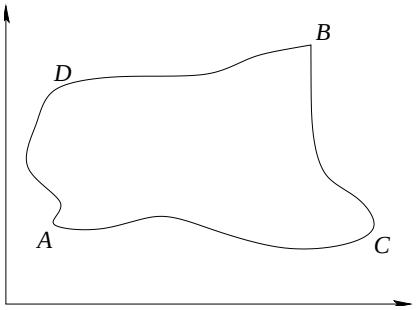


Figura 14.24: Posible camino para hacer el trabajo.

Continuando esta demostración de equivalencia, supongamos que la ecuación (14.136) se mantiene. Si $\oint \mathbf{F} \cdot d\mathbf{r} = 0$ para todos los caminos en S, vemos que el valor de la integral cerrada para dos puntos distintos A y B es independiente del camino (figura 14.24). Nuestra premisa es que

$$\oint_{ACBDA} \mathbf{F} \cdot d\mathbf{r} = 0 \ . \tag{14.140}$$

Por lo tanto

$$\int_{ACB} \mathbf{F} \cdot d\mathbf{r} = -\int_{BDA} \mathbf{F} \cdot d\mathbf{r} = \int_{ADB} \mathbf{F} \cdot d\mathbf{r} \ , \tag{14.141}$$

cambiando el signo al revertir la dirección de integración. Físicamente, esto significa que el trabajo hecho al ir de A a B es independiente del camino y que el trabajo hecho sobre un camino cerrado es cero. Esta es la razón para etiquetar como una fuerza conservativa: la energía es conservada.

Con el resultado mostrado en la ecuación (14.141), tenemos que el trabajo hecho depende solamente de los puntos extremos, A y B. Esto es,

$$\text{trabajo hecho por la fuerza} = \int_A^B \mathbf{F} \cdot d\mathbf{r} = \varphi(A) - \varphi(B) \ . \tag{14.142}$$

La ecuación (14.142) define un potencial escalar (estrictamente hablando, la diferencia de potencial entre los puntos A y B) y provee una manera de calcular el potencial. Si el punto B es tomado como una variable, digamos, (x, y, z), luego la diferenciación con respecto a x, y y z cubrirá la ecuación (14.134). La elección del signo de la mano derecha es arbitraria. La

elección aquí es hecha para obtener concordancia con la ecuación (14.134) y asegurar que el agua correrá río abajo antes que río arriba. Para los puntos A y B separados por una longitud $d\mathbf{r}$, la ecuación (14.142) se convierte

$$\begin{aligned} \mathbf{F} \cdot d\mathbf{r} &= -d\varphi \\ &= -\boldsymbol{\nabla}\varphi \cdot d\mathbf{r} \ . \end{aligned} \tag{14.143}$$

Esto puede ser reescrito

$$(\mathbf{F} + \boldsymbol{\nabla}\varphi) \cdot d\mathbf{r} = 0 \ , \tag{14.144}$$

y ya que $d\mathbf{r}$ es arbitrario se concluye la ecuación (14.134).

Si

$$\oint \mathbf{F} \cdot d\mathbf{r} = 0 \ , \tag{14.145}$$

podemos obtener la ecuación (14.135) usando el teorema de Stokes (ecuación (14.132)).

$$\oint \mathbf{F} \cdot d\mathbf{r} = \int \boldsymbol{\nabla} \times \mathbf{F} \cdot d\boldsymbol{\sigma} \ . \tag{14.146}$$

Si tomamos el camino de integración como el perímetro de una diferencial de área $d\sigma$ arbitrario, el integrando en la integral de superficie debe ser nula. De ahí que la ecuación (14.136) implica la ecuación (14.135).

Finalmente, si $\boldsymbol{\nabla} \times \mathbf{F} = 0$, necesitamos solamente revertir nuestra afirmación del teorema de Stokes (ecuación (14.146)) para derivar la ecuación (14.136). Luego, por las ecuaciones (14.142-14.144) la afirmación inicial $\mathbf{F} = -\boldsymbol{\nabla}\varphi$ está derivada. La triple equivalencia está demostrada en la figura 14.25.

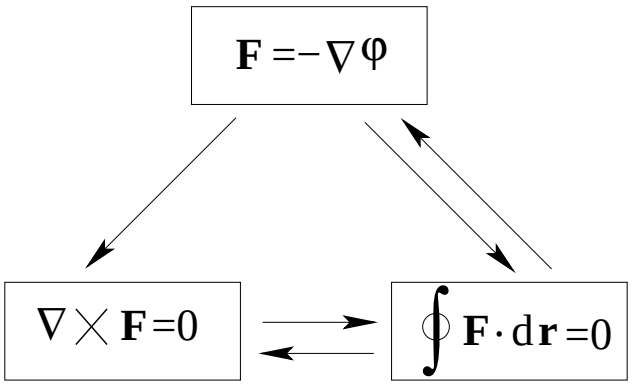


Figura 14.25: Formulaciones equivalentes para una fuerza conservativa.

Para resumir, una función potencial escalar univaluada φ existe si y sólo si $\mathbf{F}$ es irrotacional o el trabajo hecho en torno a *loops* cerrados es cero. Los campos de fuerza gravitacional y electroestáticos dados por la ecuación (14.91) son irrotacionales y por lo tanto, conservativos. Un potencial escalar gravitacional y electroestático existen.

14.13.2 Termodinámica, diferenciales exactas.

En termodinámica encontramos ecuaciones de la forma

$$df = P(x,y)dx + Q(x,y)dy \ . \tag{14.147}$$

El problema usual es determinar si $\int(P(x,y)dx+Q(x,y)dy)$ depende solamente de los puntos extremos, esto es, si df es realmente una diferencial exacta. La condición necesaria y suficiente es que

$$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy \tag{14.148}$$

o que

$$\begin{aligned} P(x,y) &= \frac{\partial f}{\partial x} \ , \\ Q(x,y) &= \frac{\partial f}{\partial y} \ . \end{aligned} \tag{14.149}$$

Las ecuaciones (14.149) dependen de que la relación

$$\frac{\partial P(x,y)}{\partial y} = \frac{\partial Q(x,y)}{\partial x} \tag{14.150}$$

se satisfaga. Esto, sin embargo, es exactamente análogo a la ecuación (14.135), con el requerimiento de que $\mathbf{F}$ sea irrotacional. Por cierto, la componente z de la ecuación (14.135) es

$$\frac{\partial F_x}{\partial y} = \frac{\partial F_y}{\partial x} \ , \tag{14.151}$$

con

$$F_x = \frac{\partial f}{\partial x} \ , \quad F_y = \frac{\partial f}{\partial y} \ .$$

Potencial vectorial.

En algunas ramas de la física, especialmente en la teoría electromagnética, es conveniente introducir un potencial vectorial $\mathbf{A}$, tal que un (fuerza) campo $\mathbf{B}$ está dado por

$$\mathbf{B} = \boldsymbol{\nabla} \times \mathbf{A} \ . \tag{14.152}$$

Claramente, si la ecuación (14.152) se mantiene, $\boldsymbol{\nabla} \cdot \mathbf{B} = 0$ por la ecuación (14.96) y $\mathbf{B}$ es solenoidal. Aquí queremos desarrollar el converso, mostrar que cuando $\mathbf{B}$ es solenoidal un potencial vectorial $\mathbf{A}$ existe. Demostramos la existencia de $\mathbf{A}$ calculándolo. Suponga que $\mathbf{B} = \hat{\mathbf{x}}b_1 + \hat{\mathbf{y}}b_2 + \hat{\mathbf{z}}b_3$ y nuestra incógnita $\mathbf{A} = \hat{\mathbf{x}}a_1 + \hat{\mathbf{y}}a_2 + \hat{\mathbf{z}}ba_3$. Por la ecuación (14.152)

$$\frac{\partial a_3}{\partial y} - \frac{\partial a_2}{\partial z} = b_1 \ , \tag{14.153}$$

$$\frac{\partial a_1}{\partial z} - \frac{\partial a_3}{\partial x} = b_2 \ , \tag{14.154}$$

$$\frac{\partial a_2}{\partial x} - \frac{\partial a_1}{\partial y} = b_3 \ . \tag{14.155}$$

Supongamos que las coordenadas han sido escogidas tal que $\mathbf{A}$ es paralelo al plano yz; esto es, $a_1 = 0$[16]. Entonces

$$\begin{aligned} b_2 &= -\frac{\partial a_3}{\partial x} \\ b_3 &= \frac{\partial a_2}{\partial x} \ . \end{aligned} \tag{14.156}$$

Integrando, obtenemos

$$\begin{aligned} a_2 &= \int_{x_0}^{x} b_3 \, dx + f_2(y,z) \ , \\ a_3 &= -\int_{x_0}^{x} b_2 \, dx + f_3(y,z) \ , \end{aligned} \tag{14.157}$$

donde f_2 y f_3 son funciones arbitrarias de y y z pero no son funciones de x. Estas dos ecuaciones pueden ser comprobadas diferenciando y recobrando la ecuación (14.156). La ecuación (14.153) se convierte en

$$\begin{aligned} \frac{\partial a_3}{\partial y} - \frac{\partial a_2}{\partial z} &= -\int_{x_0}^{x} \left(\frac{\partial b_2}{\partial y} + \frac{\partial b_3}{\partial z} \right) dx + \frac{\partial f_3}{\partial y} - \frac{\partial f_2}{\partial z} \\ &= \int_{x_0}^{x} \frac{\partial b_1}{\partial x} dx + \frac{\partial f_3}{\partial y} - \frac{\partial f_2}{\partial z} \ , \end{aligned} \tag{14.158}$$

usando $\boldsymbol{\nabla} \cdot \mathbf{B} = 0$. Integrando con respecto a x, tenemos

$$\frac{\partial a_3}{\partial y} - \frac{\partial a_2}{\partial z} = b_1(x,y,z) - b_1(x_0,y,z) + \frac{\partial f_3}{\partial y} - \frac{\partial f_2}{\partial z} \ . \tag{14.159}$$

Recordando que f_3 y f_2 son funciones arbitrarias de y y z, escogemos

$$\begin{aligned} f_2 &= 0 \ , \\ f_3 &= \int_{y_0}^{y} b_1(x_0,y,z) dy \ , \end{aligned} \tag{14.160}$$

tal que el lado derecho de la ecuación (14.159) se reduce a $b_1(x,y,z)$ en acuerdo con la ecuación (14.153). Con f_2 y f_3 dado por la ecuación (14.160), podemos construir $\mathbf{A}$.

$$\mathbf{A} = \hat{\mathbf{y}} \int_{x_0}^{x} b_3(x,y,z) dx + \hat{\mathbf{z}} \left[\int_{y_0}^{y} b_1(x_0,y,z) dy - \int_{x_0}^{x} b_2(x,y,z) dx \right] \ . \tag{14.161}$$

Esto no es muy completo. Podemos añadir una constante, ya que $\mathbf{B}$ es una derivada de $\mathbf{A}$. Y mucho más importante, podemos añadir un gradiente de una función escalar $\boldsymbol{\nabla}\varphi$ sin afectar a $\mathbf{B}$. Finalmente, las funciones f_2 y f_3 no son únicas. Otras elecciones podrían haber sido hechas. En vez de ajustar $a_1 = 0$ para obtener las ecuaciones (14.153)-(14.155) cualquier permutación cíclica de 1, 2, 3, x, y, z, x_0, y_0, z_0 tambien podría funcionar.

Agregar un gradiente de una función escalar, digamos Λ, al vector potencial $\mathbf{A}$ no afecta a $\mathbf{B}$, por la ecuación (14.94), esta tranformación es conocida como transformación de *gauge*.

$$\mathbf{A} \rightarrow \mathbf{A}' = \mathbf{A} + \boldsymbol{\nabla}\Lambda \ . \tag{14.162}$$

[16]Claramente esto puede hacerse en cualquier punto.

14.14 Ley de Gauss y ecuación de Poisson.

Consideremos una carga eléctrica puntual q en el origen de nuestro sistema de coordenadas. Esta produce un campo electrico $\mathbf{E}$ dado por

$$\mathbf{E} = \frac{q}{r^2}\,\hat{\mathbf{r}}\ , \qquad (\text{CGS}). \tag{14.163}$$

Ahora derivemos la ley de Gauss, la cual establece que la integral de superficie en la figura 14.26 es $4\pi q$ si la superficie S incluye el origen (donde q está localizada) y cero si la superficie no incluye el origen. La superficie S es cualquier superficie cerrada; no necesariamente esférica.

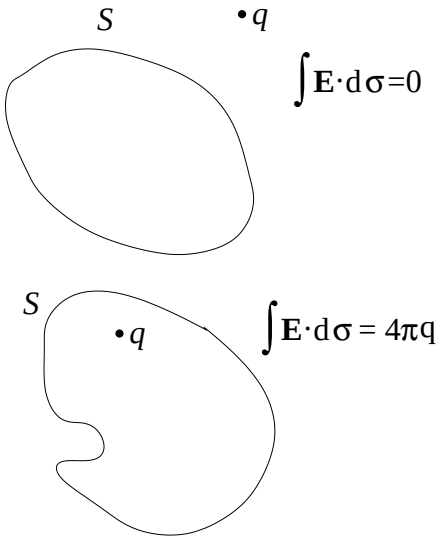


Figura 14.26: Ley de Gauss.

Usando el teorema de Gauss, la ecuación (14.116), obtenemos

$$\int_S \frac{q\,\hat{\mathbf{r}}\cdot d\sigma}{r^2} = q\int_v \boldsymbol{\nabla}\cdot\frac{\hat{\mathbf{r}}}{r^2}d\tau\ , \tag{14.164}$$

siempre que la superficie S no incluya el origen, donde el integrando no están bien definidos. Esto prueba la segunda parte de la ley de Gauss.

La primera parte, en la cual la superficie S debería incluir el origen, puede ser tomada por los alrededores del origen con una pequeña esfera S' de radio δ (figura 14.27). De esa manera no habrá duda de que si está afuera o si está adentro, imaginemos que el volumen externo a la superficie más externa y el volumen del lado interno de la superficie $S'(r > \delta)$ se conectan por un pequeño espacio. Estas superficies unidas S y S', las combinamos en una superficie cerrada. Ya que el radio del espacio interior puede hacerse infinitamente pequeño, allí no hay contribución adicional de la integral de superficie. La superficie interna está deliberadamente escogida para que sea esférica tal que seamos capaces de integrar sobre ella. El teorema de Gauss ahora se aplica sobre el volumen S y S' sin ninguna dificultad. Tenemos

$$\int_S \frac{q\,\hat{\mathbf{r}}\cdot d\sigma}{r^2} + \int_{S'} \frac{q\,\hat{\mathbf{r}}\cdot d\sigma'}{\delta^2} = 0\ , \tag{14.165}$$

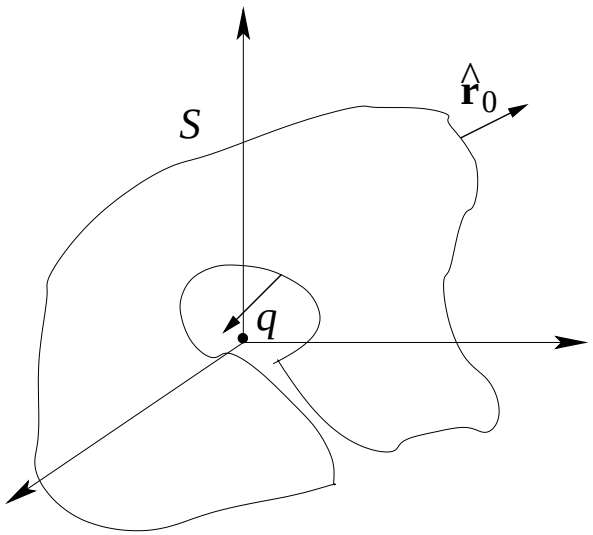


Figura 14.27: Exclusión del origen.

Podemos evaluar la segunda integral, para $d\boldsymbol{\sigma}' = -\hat{\mathbf{r}}\delta^2 d\Omega$, donde $d\Omega$ es el diferencial de ángulo sólido. El signo menos aparece porque concordamos en la sección 14.10 para tener la normal positiva $\hat{\mathbf{r}}'$ fuera del volumen. En este caso $\hat{\mathbf{r}}'$ externo está en la dirección radial, $\hat{\mathbf{r}}' = -\hat{\mathbf{r}}$. Integrando sobre todos los ángulos, tenemos

$$\int_{S'} \frac{q\,\hat{\mathbf{r}} \cdot d\boldsymbol{\sigma}'}{\delta^2} = -\int_{S'} \frac{q\,\hat{\mathbf{r}} \cdot \hat{\mathbf{r}}\delta^2 d\Omega}{\delta^2} = -4\pi q \ , \tag{14.166}$$

independiente de radio δ. Luego tenemos

$$\int_s \mathbf{E} \cdot d\boldsymbol{\sigma} = 4\pi q \ , \tag{14.167}$$

completando la prueba de la ley de Gauss. Note cuidadosamente que aunque la superficie S puede ser esférica, no necesariamente es esférica.

Avanzando un poco más, consideremos una carga distribuida tal que

$$q = \int_V \rho d\tau \ . \tag{14.168}$$

La ecuación (14.167) todavía se aplica, con q interpretada como la distribución de carga total encerrada por la superficie S.

$$\int_s \mathbf{E} \cdot d\sigma = 4\pi \int_V \rho d\tau \ . \tag{14.169}$$

Usando el teorema de Gauss, tenemos

$$\int_V \boldsymbol{\nabla} \cdot \mathbf{E} d\tau = \int_V 4\pi\rho d\tau \ . \tag{14.170}$$

Ya que nuestro volumen es completamente arbitrario, los integrandos deben ser iguales o

$$\boldsymbol{\nabla}\cdot\mathbf{E}=4\pi\rho\ , \tag{14.171}$$

una de las ecuaciones de Maxwell. Si revertimos el argumento, la ley de Gauss se deduce inmediatamente a partir de la ecuación de Maxwell.

14.14.1 Ecuación de Poisson.

Reemplazando $\mathbf{E}$ por $-\boldsymbol{\nabla}\varphi$, la ecuación (14.171) se convierte en

$$\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\varphi=\nabla^2\varphi=-4\pi\rho\ , \tag{14.172}$$

la cual es la ecuación de Poisson. Para la condición $\rho=0$ ésta se reduce a la famosa ecuación,

$$\nabla^2\varphi=0\ , \tag{14.173}$$

de Laplace. Encontramos frecuentemente la ecuación de Laplace distintos sistemas de coordenadas y las funciones especiales de la Física Matemática aparecerán como sus soluciones. La ecuación de Poisson será invaluable en el desarrollo de la teoría de las funciones de Green.

A partir de la directa comparación de la fuerza electroestática de Coulomb y al ley de Newton de la gravitación universal

$$\mathbf{F}_E=\frac{q_1q_2}{r^2}\hat{\mathbf{r}}\ ,\quad \mathbf{F}_G=-G\frac{m_1m_2}{r^2}\hat{\mathbf{r}}\ .$$

Todas las teorías de potencial de esta sección se aplican igualmente bien a los potenciales gravitacionales. Por ejemplo, la ecuación gravitacional de Poisson es

$$\nabla^2\varphi=+4\pi G\rho \tag{14.174}$$

con ρ ahora una densidad de masa.

14.15 La delta de Dirac.

Del desarrollo de la ley de Gauss en la sección 14.14

$$\int\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\left(\frac{1}{r}\right)d\tau=-\int\boldsymbol{\nabla}\cdot\left(\frac{\hat{\mathbf{r}}}{r^2}\right)d\tau=\begin{cases}-4\pi\\0\end{cases}\ , \tag{14.175}$$

dependiendo de si la integración incluye el origen $\mathbf{r}=0$ o no. Este resultado puede ser convenientemente expresado introduciendo la delta de Dirac,

$$\nabla^2\left(\frac{1}{r}\right)=-4\pi\delta(r)=-4\pi\delta(x)\delta(y)\delta(z)\ . \tag{14.176}$$

Esta "función" delta de Dirac está definida por sus propiedades

$$\delta(x)=0\ ,\quad x\neq0 \tag{14.177}$$

$$\int f(x)\delta(x)dx = f(0) \ , \tag{14.178}$$

donde $f(x)$ es cualquiera función bien comportada y la integración incluye el origen. Un caso especial de la ecuación (14.178),

$$\int \delta(x)dx = 1 \ . \tag{14.179}$$

De la ecuación (14.178), $\delta(x)$ debe ser un pico infinitamente alto y delgado en $x = 0$, como en la descripción de un impulso de fuerza o la densidad de carga para una carga puntual. El problema es que tales funciones no existen en el sentido usual de una función. Sin embargo la propiedad crucial en la ecuación (14.178) puede ser desarrollada rigurosamente como el límite de una secuencia de funciones, una distribución. Por ejemplo, la función delta puede ser aproximada por la secuencias de funciones, ecuaciones (14.180) a (14.182) y las figuras 14.28 a 14.30 :

$$\delta_n(x) = \begin{cases} 0 \ , & \text{si } x < -1/n \\ n/2 \ , & \text{si } -1/n < x < 1/n \\ 0 \ , & \text{si } x > 1/n \end{cases} \tag{14.180}$$

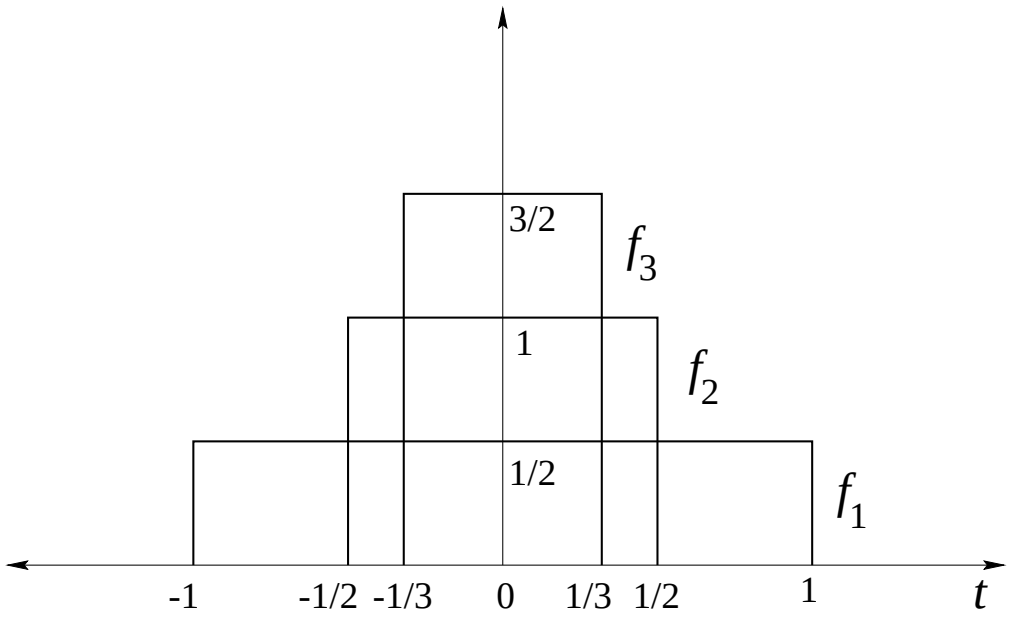


Figura 14.28: Secuencia a la delta.

$$\delta_n(x) = \frac{n}{\sqrt{\pi}} e^{n^2x^2} \tag{14.181}$$

$$\delta_n(x) = \frac{\text{sen}(nx)}{\pi x} = \frac{1}{2\pi} \int_{-n}^{n} e^{ixt}\, dt \ . \tag{14.182}$$

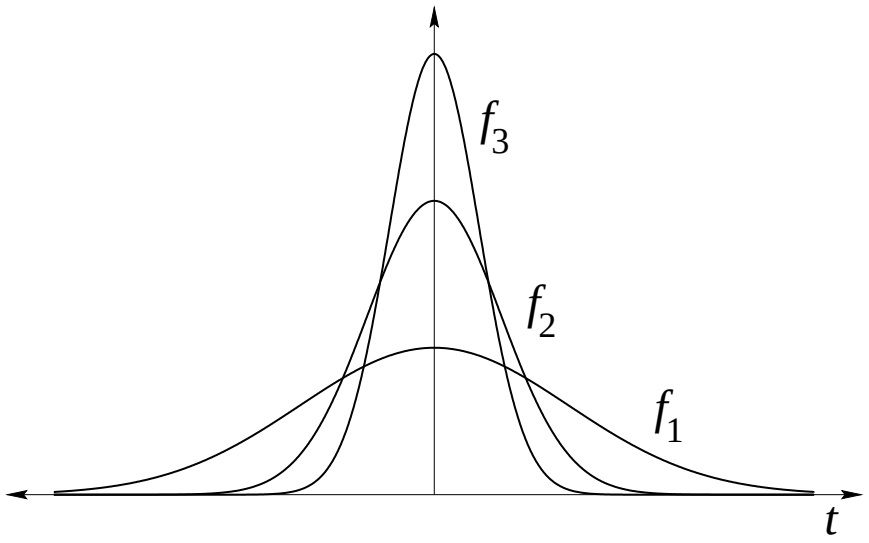


Figura 14.29: Secuencia a la delta.

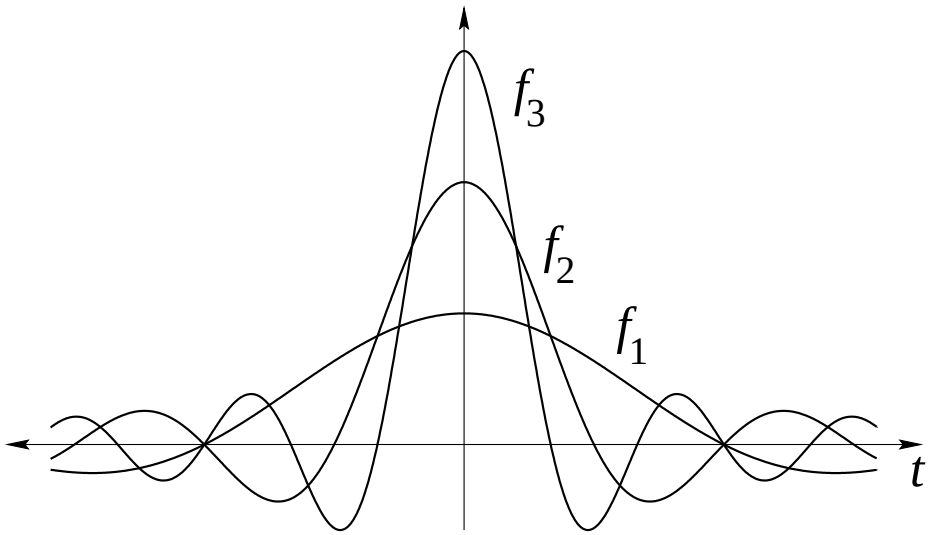


Figura 14.30: Secuencia a la delta.

Estas aproximaciones tienen grados de utilidad variable. La ecuación (14.180) es útil en dar una derivación simple de la propiedad integral, la ecuación (14.178). La ecuación (14.181) es conveniente para diferenciar. Sus derivadas tienden a los polinomios de Hermite. La ecuación (14.182) es particularmente útil en el análisis de Fourier y en sus aplicaciones a la mecánica cuantica. En la teoría de las series de Fourier, la ecuación (14.182) a menudo aparece (modificada) como el kernel de Dirichlet:

$$\delta_n(x) = \frac{1}{2\pi} \frac{\operatorname{sen}\left[(n+\frac{1}{2})x\right]}{\frac{1}{2}x} \ . \tag{14.183}$$

Usando esta aproximación en la ecuación (14.178) y después, suponemos que $f(x)$ es bien comportada no ofrece problemas para x grandes. Para muchos propósitos físicos tales aproximaciones son muy adecuadas. Desde un punto de vista matemático la situación todavía es insatisfactoria: Los límites

$$\lim_{n\to\infty} \delta_n(x)$$

no existen.

Una salida para esta dificultad está dada por la teoría de las distribuciones. Reconociendo que la ecuación (14.178) es la propiedad fundamental, enfocaremos nuestra atención sobre algo más que $\delta(x)$. Las ecuaciones (14.180)-(14.182) con $n = 1, 2, 3, \ldots$ puede ser interpretada como una secuencia de funciones normalizadas:

$$\int_{-\infty}^{\infty} \delta_n(x)\, dx = 1 \ . \tag{14.184}$$

La secuencia de integrales tiene el límite

$$\lim_{n\to\infty} \int_{-\infty}^{\infty} \delta_n(x) f(x)\, dx = f(0) \ . \tag{14.185}$$

Note que la ecuación (14.185) es el límite de una secuencias de integrales. Nuevamente , el límite $\delta_n(x)$, $n \to \infty$, no existe. (El límite para todas las formas de δ_n diverge en $x = 0$.)

Podemos tratar $\delta(x)$ consistentemente en la forma

$$\int_{-\infty}^{\infty} \delta(x) f(x)\, dx = \lim_{n\to\infty} \int_{-\infty}^{\infty} \delta_n(x) f(x)\, dx \ . \tag{14.186}$$

$\delta(x)$ es etiquetado como una distribución (no una función) definida por la secuencia $\delta_n(x)$ como está indicado en la ecuación (14.186). Podríamos enfatizar que la integral sobre el lado izquierdo de la ecuación (14.186) no es una integral de Riemann. Es un límite.

Esta distribución $\delta(x)$ es solamente una de una infinidad de posibles distribuciones, pero es la única en que estamos interesados en la ecuación (14.178). A partir de estas secuencias de funciones vemos que la "función" delta de Dirac debe ser par en x, $\delta(-x) = \delta(x)$. La propiedad integral, la ecuación (14.178), es útil en casos, donde el argumento de la función delta es una función $g(x)$ con ceros simples sobre el eje real, el cual tiende a las reglas

$$\delta(ax) = \frac{1}{a}\delta(x) \ , \quad a > 0 \ , \tag{14.187}$$

$$\delta(g(x)) = \sum_{a, g(a)=0, g'(a)\neq 0} \frac{\delta(x-a)}{\mid g'(a) \mid} \ . \tag{14.188}$$

Para obtener la ecuación (14.187) cambiamos la variable de integración en

$$\int_{-\infty}^{\infty} \delta(ax) f(x)\, dx = \frac{1}{a} \int_{-\infty}^{\infty} \delta(y) f(y/a)\, dy = \frac{1}{a} f(0) \ .$$

y aplicamos la ecuación (14.178). Para probar la ecuación (14.188) descomponemos la integral

$$\int_{-\infty}^{\infty} \delta(g(x)) f(x)\, dx = \sum_a \int_{a-\epsilon}^{a+\epsilon} f(x)\delta(x-a)g'(a)\, dx \tag{14.189}$$

en una suma de integrales sobre pequeños intervalos que contiene los ceros de $g(x)$. En esos intervalos, $g(x) \approx g(a) + (x-a)g'(a) = (x-a)g'(a)$. Usando la euación (14.187) sobre el lado derecho de la ecuación (14.189) obtenemos la integral de la ecuación (14.188.

Usando integración por partes tambien podemos definir la derivada $\delta'(x)$ de la delta de Dirac por la relación

$$\int_{-\infty}^{\infty} \delta'(x-x_0) f(x)\, dx = -\int_{-\infty}^{\infty} f'(x)\delta(x-x_0)\, dx = -f'(x_0)\ . \tag{14.190}$$

Usamos $\delta(x)$ frecuentemente y la llamamos la delta de Dirac[17] por razones históricas. Recuerde que no es realmente una función. Es escencialmente una notación más taquigráfica, definida implícitamente como el límite de integrales en una secuencia, $\delta_n(x)$, de acuerdo a la ecuación (14.186). Podría entenderse que nuestra delta de Dirac es a menudo mirada como un operador, un operador lineal: $\delta(x-x_0)$ opera sobre $f(x)$ y tiende a $f(x_0)$.

$$\mathcal{L}(x_0) f(x) \equiv \int_{-\infty}^{\infty} f(x)\delta(x-x_0)\, dx = f(x_0)\ . \tag{14.191}$$

Tambien podría ser clasificada como un mapeo lineal o simplemente como una función generalizada. Desplazando nuestra singularidad al punto $x = x_0$, escribimos la delta de Dirac como $\delta(x-x_0)$. La ecuación (14.178) se convierte en

$$\int_{-\infty}^{\infty} f(x)\delta(x-x_0)\, dx = f(x_0) \tag{14.192}$$

Como una descripción de una singularidad en $x = x_0$, la delta de Dirac puede ser escrita como $\delta(x-x_0)$ o como $\delta(x_0-x)$. En tres dimensiones y usando coordenadas polares esféricas, obtenemos

$$\int_0^{2\pi}\int_0^{\pi}\int_0^{\infty} \delta(\mathbf{r})\, r^2 dr\, \operatorname{sen}\theta\, d\theta\, d\varphi = \int -\infty^{\infty} \int -\infty^{\infty} \int_{-\infty}^{\infty} \delta(x)\delta(y)\delta(z)\, dx\, dy\, dz = 1\ . \tag{14.193}$$

Esto corresponde a una singularidad (o fuente) en el origen. De nuevo, si la fuente está en $\mathbf{r} = \mathbf{r}_1$, la ecuación (14.193) se convierte en

$$\int_0^{2\pi}\int_0^{\pi}\int_0^{\infty} \delta(\mathbf{r}_2 - \mathbf{r}_1)\, r_2^2 dr_2\, \operatorname{sen}\theta_2 d\theta_2\, d\varphi_2 = 1\ . \tag{14.194}$$

[17] Dirac introdujo la delta para Mecánica Cuántica.

14.15.1 Representación de la delta por funciones ortogonales.

La función delta de Dirac puede ser expandida en términos de alguna base de funciones ortogonales reales $\varphi_n(x)$ con $n = 0, 1, 2, \ldots$. Tales funciones aparecerán más adelante como soluciones de ecuaciones diferenciales ordinarias de la forma Sturm-Liouville. Ellas satisfacen las relaciones de ortogonalidad

$$\int_a^b \varphi_m(x)\varphi_n(x)\,dx = \delta_{mn} \ , \tag{14.195}$$

donde el interevalo (a, b) puede ser infinito en uno de los lados o en ambos.[Por conveniencia suponemos que $\varphi_n(x)$ ha sido definida para incluir $(w(x))^{1/2}$ si las relaciones de ortogonalidad contienen una función de peso positivo adicional $w(x)$.] Usamos $\varphi_n(x)$ para expandir la delta como

$$\delta(x-t) = \sum_{n=1}^{\infty} a_n(t)\varphi_n(x) \ , \tag{14.196}$$

donde los coeficientes a_n son funciones de la variable t. Multiplicando por $\varphi_m(x)$ e integrando sobre el intervalo ortogonal (ecuación (14.195)), tenemos

$$a_m(t) = \int_a^b \delta(x-t)\varphi_m(x)\,dx = \varphi_m(t) \tag{14.197}$$

o

$$\delta(x-t) = \sum_{n=1}^{\infty} \varphi_m(t)\varphi_n(x) = \delta(t-x) \ . \tag{14.198}$$

Esta serie ciertamente no es uniformemente convergente, pero puede ser usada como parte de un integrando en el cual la integración resultante la hará convergente.

Suponga que forma la integral $\int F(t)\delta(t-x)dx$, donde se supone que $F(t)$ puede ser expandida en una serie de funciones ortogonales $\varphi_p(t)$, una propiedad llamada completitud. Luego obtenemos

$$\begin{aligned} \int F(t)\delta(t-x)dt &= \int \sum_{p=0}^{\infty} a_p\varphi_p(t) \sum_{n=0}^{\infty} \varphi_n(x)\varphi_n(t)\,dt \\ &= \sum_{p=0}^{\infty} a_p\varphi_p(x) = F(x) \ , \end{aligned} \tag{14.199}$$

los productos cruzado $\int \varphi_p\varphi_n dt$ con $(n \neq p)$ se anulan por ortogonalidad. (ecuación (14.195)). Refiriéndonos a la anterior definición de la función delta de Dirac, ecuación (14.178), vemos que nuestra representación en serie, ecuación (14.198), sastiface la propiedad de definición de la delta de Dirac que es llamada clausura. La suposición de completitud de un conjunto de funciones para la expansión de $\delta(x-t)$ produce la relación de clausura. Lo converso, que clausura implica completitud.

14.15.2 Representación integral para la delta.

Las transformaciones integrales, tal como la integral de Fourier

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \exp(i\omega t)\, dt$$

conducen a representaciones integrales de la delta de Dirac. Por ejemplo,

$$\delta_n(t-x) = \frac{\operatorname{sen} n(t-x)}{\pi(t-x)} = \frac{1}{2\pi} \int_{-n}^{n} \exp(i\omega(t-x))\, d\omega \ , \tag{14.200}$$

usando la ecuación (14.182). Tenemos

$$f(x) = \lim_{n\to\infty} \int_{-\infty}^{\infty} f(t)\delta_n(t-x)\, dt \ , \tag{14.201}$$

donde $\delta_n(t-x)$ es la secuencia en la ecuación (14.200) definiendo la distribución $\delta(t-x)$. Note que la ecuación (14.201) supone que $f(t)$ es continua en $t = x$. Si sustituimos la ecuación (14.200) en la ecuación (14.201) obtenemos

$$f(x) = \lim_{n\to\infty} \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) \int_{-n}^{n} \exp(i\omega(t-x))\, d\omega \ . \tag{14.202}$$

Intercambiando el orden de integración y luego tomando el límite $n \to \infty$, tenemos el teorema de integral de Fourier.

Con el entendimiento de que pertenece bajo el signo de la integral como en la ecuación (14.201), la identificación

$$\delta(t-x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \exp(i\omega(t-x))\, d\omega \tag{14.203}$$

proporciona una muy útil representación integral de la delta.

Cuando la transformada de Laplace

$$L_\delta(s) = \int_0^{\infty} \exp(-st)\delta(t-t_0) = \exp(-st_0) \ , \quad t_0 > 0 \tag{14.204}$$

es invertida, obtenemos la representación compleja

$$\delta(t-t_0) = \frac{1}{2\pi i} \int_{\gamma-i\infty}^{\gamma+i\infty} \exp(s(t-t_0))\, ds \ , \tag{14.205}$$

la cual es esencialmente equivalente a la representación prevía de Fourier de la delta de Dirac.

14.16 Teorema de Helmholtz.

En la sección 14.13 enfatizamos que la elección de un potencial vectorial magnético $\mathbf{A}$ no era único. La divergencia de $\mathbf{A}$ estaba aún indeterminada. En esta sección dos teoremas acerca de la divergencia y el rotor de un vector son desarrollados. El primer teorema es el siguiente.

Un vector está únicamente especificado dando su divergencia y su rotor dentro de una región y su componente normal sobre el contorno.

Tomemos

$$\begin{aligned} \boldsymbol{\nabla} \cdot \mathbf{V}_1 &= s \ , \\ \boldsymbol{\nabla} \times \mathbf{V}_1 &= \mathbf{c} \ , \end{aligned} \tag{14.206}$$

donde s puede ser interpretado como una fuente (carga) densidad y $\mathbf{c}$ como una circulación (corriente) densidad. Suponiendo también que la componente normal V_{1n} sobre el límite está dada, deseamos mostrar que $\mathbf{V}_1$ es único. Hacemos esto suponiendo la existencia de un segundo vector $\mathbf{V}_2$, el cual satisface la ecuación (14.206) y tiene la misma componente normal sobre el borde, y luego mostrando que $\mathbf{V}_1 - \mathbf{V}_2 = 0$. Sea

$$\mathbf{W} = \mathbf{V}_1 - \mathbf{V}_2 \ .$$

Luego

$$\boldsymbol{\nabla} \cdot \mathbf{W} = 0 \tag{14.207}$$

y

$$\boldsymbol{\nabla} \times \mathbf{W} = 0 \ . \tag{14.208}$$

Ya que $\mathbf{W}$ es irrotacional podemos escribir (por la sección 14.13)

$$\mathbf{W} = -\boldsymbol{\nabla}\varphi \ . \tag{14.209}$$

Sustituyendo esto en la ecuación (14.207), obtenemos

$$\boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\varphi = 0, \tag{14.210}$$

la ecuación de la Laplace.

Ahora hagamos uso del teorema de Green en la forma dada en la ecuación (14.121, siendo u y v cada uno igual a φ. Ya que

$$W_n = V_{1n} - V_{2n} = 0 \tag{14.211}$$

sobre el borde, el teorema de Green se reduce a

$$\int_V (\boldsymbol{\nabla}\varphi) \cdot (\boldsymbol{\nabla}\varphi)\, d\tau = \int_v \mathbf{W} \cdot \mathbf{W}\, d\tau = 0 \ . \tag{14.212}$$

La cantidad $\mathbf{W} \cdot \mathbf{W} = W^2$ es no negativa y por eso tenemos

$$\mathbf{W} = \mathbf{V}_1 - \mathbf{V}_2 = 0 \ , \tag{14.213}$$

en todo lugar. Por lo tanto $\mathbf{V}_1$ es único, probando el teorema.

Para nuestro potencial vectorial magnético $\mathbf{A}$ la relación $\mathbf{B} = \boldsymbol{\nabla} \times \mathbf{A}$ especifica el rotor de $\mathbf{A}$. A menudo por conveniencia ajustamos $\boldsymbol{\nabla} \cdot \mathbf{A} = \mathbf{0}$. Luego (con las condiciones de borde) $\mathbf{A}$ está fijo.

Esta teorema puede ser escrito como un teorema de unicidad para las soluciones de ecuaciones de Laplace. En esta forma, este teorema de unicidad es de gran importancia en en la solución de los problemas de contorno de electroestáticas y otras de ecuaciones tipo Laplace con condiciones de borde. Si podemos encontrar una solución de las ecuaciones de Laplace que satisfagan las condiciones de borde necesarias, luego nuestra solución es la solución completa.

14.16.1 Teorema de Helmholtz.

El segundo teorema que probaremos es el teorema de Helmholtz.

Un vector $\mathbf{V}$ *que satisface la ecuacion (14.206) con ambas densidades de fuente y circulación que se anulan en infinito pueden ser escritas como la suma de dos partes, una de la cual es la irrotacional, la otra la solenoidal.*

El teorema de Helmholtz claramente será satisfecho si podemos escribir $\mathbf{V}$ como

$$\mathbf{V} = -\boldsymbol{\nabla}\varphi + \boldsymbol{\nabla} \times \mathbf{A} \ , \tag{14.214}$$

$-\boldsymbol{\nabla}\varphi$ siendo irrotacional y $\boldsymbol{\nabla} \times \mathbf{A}$ siendo solenoidal. Procederemos a justificar la ecuación (14.214).

Sea $\mathbf{V}$ un vector conocido. Tomemosle la divergencia y el rotor

$$\boldsymbol{\nabla} \cdot \mathbf{V} = s(\mathbf{r}) \tag{14.215}$$

$$\boldsymbol{\nabla} \times \mathbf{V} = \mathbf{c}(\mathbf{r}) \ . \tag{14.216}$$

con $s(\mathbf{r})$ y $c(\mathbf{r})$ funciones de conocidas de la posición. A partir de estas dos funciones construímos un potencial escalar $\varphi(\mathbf{r}_1)$,

$$\varphi(\mathbf{r}_1) = \frac{1}{4\pi} \int \frac{s(\mathbf{r}_2)}{r_{12}} \, d\tau_2 \ , \tag{14.217}$$

y un potencial vector $\mathbf{A}(\mathbf{r}_1)$,

$$\mathbf{A}(\mathbf{r}_1) = \frac{1}{4\pi} \int \frac{\mathbf{c}(\mathbf{r}_2)}{r_{12}} \, d\tau_2 \ . \tag{14.218}$$

Si $s = 0$ luego $\mathbf{V}$ es solenoidal y la ecuación (14.217) implica $\varphi = 0$. De la ecuación (14.214), $\mathbf{V} = \boldsymbol{\nabla} \times \mathbf{A}$ con $\mathbf{A}$ como dado en la ecuación (14.161 es consistente con la sección 14.13. Además, si $\mathbf{c} = 0$ luego $\mathbf{V}$ es irrotacional y la ecuación (14.218) implica $\mathbf{A} = 0$, y la ecuación (14.214) implica $\mathbf{V} = -\boldsymbol{\nabla}\varphi$, consistente con la teoría de potencial escalar de la sección 14.13.

Aquí el argumento $\mathbf{r}_1$ indica que (x_1, y_1, z_1), el campo puntual; $\mathbf{r}_2$, las coordenadas de la fuente puntual (x_2, y_2, z_2), mientras

$$r_{12} = [(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2]^{1/2}. \tag{14.219}$$

Cuando una dirección está asociada con r_{12}, la dirección positiva es tomada para estar alejarse de la fuente. Vectorialmente, $\mathbf{r}_{12} = \mathbf{r}_1 - \mathbf{r}_2$, como se muestra en la figura 14.31. Por su puesto, s y c deben anularse suficientemente rápido en distancias grandes tal que existan las integrales. La expansión actual y la evaluación de las integrales tales como las ecuaciones (14.217) y (14.218) están tratadas más adelante A partir del teorema de unicidad en los comienzos de esta sección, $\mathbf{V}$ es único al especificar su divergencia, s, y su rotor, $\mathbf{c}$ (y su valor sobre el contorno). Volviendo a la ecuación (14.214), tenemos

$$\boldsymbol{\nabla} \cdot \mathbf{V} = -\boldsymbol{\nabla} \cdot \boldsymbol{\nabla}\varphi \ , \tag{14.220}$$

la divergencia del rotor se anula y

$$\boldsymbol{\nabla} \times \mathbf{V} = \boldsymbol{\nabla} \times \boldsymbol{\nabla} \times \mathbf{A} \ , \tag{14.221}$$

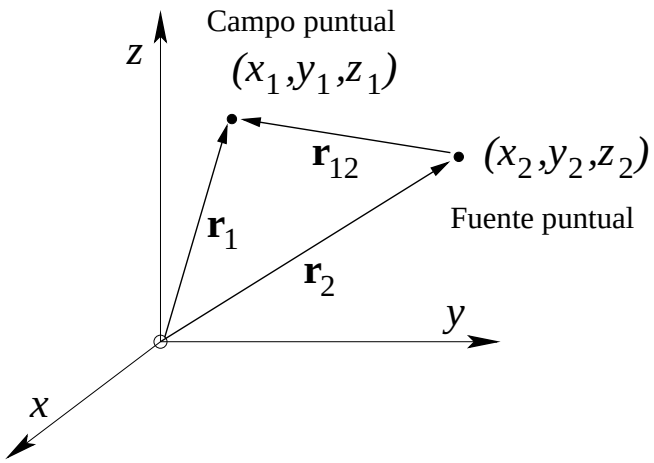


Figura 14.31: Campo y fuente puntual.

el rotor del gradiente se anula. Si podemos mostrar que

$$-\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\varphi(\mathbf{r}_1)=s(\mathbf{r}_1) \tag{14.222}$$

y

$$\boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\mathbf{A}(\mathbf{r}_1)=\mathbf{c}(\mathbf{r}_1), \tag{14.223}$$

luego $\mathbf{V}$ está dado en la ecuación (14.214) tendrá la divergencia y el rotor apropiado. Nuestra descripción será internamente consistente y la ecuación (14.214) estará justificada.

$$\boldsymbol{\nabla}\cdot\mathbf{V}=-\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\varphi=-\frac{1}{4\pi}\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\int\frac{s(\mathbf{r}_2)}{r_{12}}\,d\tau_2\ . \tag{14.224}$$

El operador Laplaciano, $\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}$ o ∇^2, opera sobre las coordenadas de campo (x_1,y_1,z_1) y por lo tanto conmuta con la integración con respecto a (x_2,y_2,z_2). Tenemos

$$\boldsymbol{\nabla}\cdot\mathbf{V}=-\frac{1}{4\pi}\int s(\mathbf{r}_2)\nabla_1^2\left(\frac{1}{r_{12}}\right)\,d\tau_2 \tag{14.225}$$

Debemos hacer dos modificaciones menores en la ecuación (14.175) antes de aplicarla.

Primero, nuestra fuente está en $\mathbf{r}_2$, no está en el origen. Esto significa que el 4π en la ley de Gauss aparece si y sólo si la superficie incluye el punto $\mathbf{r}=\mathbf{r}_2$. Para mostrar esto, reescribimos la ecuación (14.175):

$$\nabla_1^2\left(\frac{1}{r_{12}}\right)=-4\pi\delta(\mathbf{r}_1-\mathbf{r}_2)\ . \tag{14.226}$$

Este corrimiento de la fuente a $\mathbf{r}_2$ puede ser incorporado en las ecuaciones de definición (14.178) como

$$\delta(\mathbf{r}_1-\mathbf{r}_2)=0\ ,\qquad \mathbf{r}_1\neq\mathbf{r}_2\ , \tag{14.227}$$

$$\int f(\mathbf{r}_1)\delta(\mathbf{r}_1-\mathbf{r}_2)\,d\tau_1 = f(\mathbf{r}_2)\ . \tag{14.228}$$

Segundo, notando que diferenciando dos veces r_{12}^{-1} con respecto a x_2, y_2, z_2 es la misma cuando diferenciamos dos veces con respecto a x_1, y_1, z_1, tenemos

$$\begin{aligned}\nabla_1^2\left(\frac{1}{r_{12}}\right) = \nabla_2^2\left(\frac{1}{r_{12}}\right) &= -4\pi\delta(\mathbf{r}_1-\mathbf{r}_2)\\ &\quad -4\pi\delta(\mathbf{r}_2-\mathbf{r}_1)\ .\end{aligned} \tag{14.229}$$

Reescribiendo la ecuación (14.225) y usando la delta de Dirac en la ecuación (14.229), podemos integrar para obtener

$$\begin{aligned}\boldsymbol{\nabla}\cdot\mathbf{V} &= -\frac{1}{4\pi}\int s(\mathbf{r}_2)\nabla_1^2\left(\frac{1}{r_{12}}\right)\,d\tau_2\\ &= -\frac{1}{4\pi}\int s(\mathbf{r}_2)(-4\pi)\delta(\mathbf{r}_2-\mathbf{r}_1)\,d\tau_2\\ &= s(\mathbf{r}_1)\ .\end{aligned} \tag{14.230}$$

El paso final se sigue de la ecuación (14.228) con los subíndices 1 y 2 intercambiados. Nuestros resultados, ecuación (14.230), muestra que la forma supuesta de $\mathbf{V}$ y del potencial escalar φ están en concordancia con la divergencia dada (ecuación (14.215). Para completar la prueba del teorema de Helmholtz, necesitamos mostrar que nuestras suposiciones son consistentes con la ecuación (14.216), esto es, que el rotor de $\mathbf{V}$ es igual a $\mathbf{c}(\mathbf{r}_1)$. De la ecuación (14.214),

$$\boldsymbol{\nabla}\times\mathbf{V} = \boldsymbol{\nabla}\times\boldsymbol{\nabla}\times\mathbf{A} = \boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\mathbf{A} - \nabla^2\mathbf{A}\ . \tag{14.231}$$

El primer término, $\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\mathbf{A}$ tiende a

$$4\pi\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\mathbf{A} = \int \mathbf{c}(\mathbf{r}_2)\cdot\boldsymbol{\nabla_1}\boldsymbol{\nabla_1}\left(\frac{1}{r_{12}}\right)\,d\tau \tag{14.232}$$

por la ecuación (14.218). Nuevamente reemplazando la segunda derivada con respecto a x_1, y_1, z_1 por segundas derivadas con respecto a x_2, y_2, z_2, integramos cada una de las componentes de la ecuación (14.141) por partes:

$$\begin{aligned}4\pi\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\mathbf{A}\Big|_x &= \int \mathbf{c}(\mathbf{r}_2)\cdot\boldsymbol{\nabla_2}\frac{\partial}{\partial x_2}\left(\frac{1}{r_{12}}\right)\,d\tau_2\\ &= \int \boldsymbol{\nabla_2}\cdot\left[\mathbf{c}(\mathbf{r}_2)\frac{\partial}{\partial x_2}\left(\frac{1}{r_{12}}\right)\right]\,d\tau_2 - \int[\boldsymbol{\nabla_2}\cdot\mathbf{c}(\mathbf{r}_2)]\frac{\partial}{\partial x_2}\left(\frac{1}{r_{12}}\right)\,d\tau_2\ .\end{aligned} \tag{14.233}$$

La segunda integral se anula ya que la densidad de circulación $\mathbf{c}$ es solenoidal. La primera integral puede ser transformada a una integral de superficie por el teorema de Gauss. Si $\mathbf{c}$ está enlazado en el espacio o se anula más rápido que $1/r$ para r grandes, de modo que la integral en la ecuación (14.218) existe, luego escogiendo una superficie suficientemente grande la primera integral de la mano derecha de la ecuación (14.233) tambien se anula. Con $\boldsymbol{\nabla}\boldsymbol{\nabla}\cdot\mathbf{A} = 0$, la ecuación (14.231) se reduce a

$$\boldsymbol{\nabla}\times\mathbf{V} = -\nabla^2\mathbf{A} = -\frac{1}{4\pi}\int \mathbf{c}(\mathbf{r}_2)\nabla_1^2\left(\frac{1}{r_{12}}\right)\,d\tau_2\ . \tag{14.234}$$

Esto es exactamente como la ecuación (14.225) excepto que el escalar $s(\mathbf{r}_2)$ es reemplazado por la densidad de circulación vectorial $\mathbf{c}(\mathbf{r}_2)$. Introduciendo la delta de Dirac, como antes, como una manera conveniente de llevar a cabo la integración, definimos que la ecuación (14.234) se reduce a la ecuación (14.206). Vemos que nuestra forma supuesta de $\mathbf{V}$, dada por la ecuación (14.214), y del potencial vectorial $\mathbf{A}$, dado por la ecuación (14.218), están de acuerdo con la ecuación (14.206) específicamente con el rotor de $\mathbf{V}$.

Esto completa la prueba del teorema de Helmholtz, mostrando que un vector puede ser resuelto en una parte irrotacional y otra solenoidal. Aplicado al campo electromagnético, hemos resuelto nuestro campo vectorial $\mathbf{V}$ en un campo eléctrico irrotacional $\mathbf{E}$, derivado de un potencial escalar φ, y un campo de magnético solenoidal $\mathbf{B}$, derivado de un potencial vectorial $\mathbf{A}$. La densidad de fuente $s(\mathbf{r})$ puede ser interpretada como una densidad de carga eléctrica (dividida por la permitividad eléctrica ϵ), mientras que la densidad de circulación $\mathbf{c}(\mathbf{r})$ se convierte en la densidad de corriente eléctrica.