# Introducción a SCILAB

Héctor Manuel Mora Escobar
Departamento de Matemáticas
Universidad Nacional de Colombia
Bogotá

mayo del 2005

# ÍNDICE GENERAL

# Capítulo 1

# INTRODUCCIÓN

Este documento es una pequeña introducción a Scilab. Está lejos de ser exhaustivo con respecto a los temas, es decir, muchos de los temas y posibilidades de Scilab no son tratados aquí. Además, tampoco es exhaustivo con respecto al contenido de cada tema, sólo están algunos aspectos de cada tema.

El autor estará muy agradecido por los comentarios, sugerencias y correcciones enviados a:

`hmmorae@unal.edu.co` o `hectormora@yahoo.com`

Scilab fue desarrollado en el INRIA, Institut National de Recherche en Informatique et Automatique, un excelente instituto francés de investigación. Posteriormente colaboró la escuela de ingenieros ENPC, Ecole Nationale de Ponts et Chaussées.

Actualmente hay un gran consorcio con empresas como Renault, Peugeot-Citroen, CEA Commissariat à l'Energie Atomique, CNES Centre National d'Etudes spatiales, Dassault Aviation, EDF Electricité de France, Thales...

Sus principales características son:

- software para cálculo científico
- interactivo
- programable
- **de libre uso**, con la condición de siempre hacer referencia a sus autores
- disponible para diferentes plataformas: Windows, Linux, Sun, Alpha, ...

El sitio oficial de Scilab es

`www.scilab.org`

Allí se encuentra información general, manuales, FAQs (frequent asked questions), referencias sobre reportes, diferencias con Matlab, lista de errores, ... Allí se puede "bajar" la versión binaria o las fuentes para las diferentes plataformas.

Un libro bastante completo sobre el tema es:

> Gomez C. ed.,
> *Engineering and Scientific Computing with Scilab,*
> Birkhauser, Boston, 1999.

Otros libros son:

- Allaire G y Kaber S.M., *Introduction à Scilab, Exercises pratiques d'algèbre linéaire*, Ellipses, Paris, 2002.
- Guerre-Delabrière S. y Postel M., *Méthodes d'approximation : Équations différentielles - Applications Scilab, niveau L3*, Ellipses, Paris, 2004.
- Yger A., *Théorie et analyse du signal : Cours et initiation pratique via MATLAB et SCILAB* (1 décembre 1999) Ellipses, Paris, 1999.
- Urroz G., *Numerical and Statistical Methods With Scilab for Science and Engineering*, vol. 1, 2, Booksurge, 2001.
- Gomez C., Bunks C. et al., *Integrated Scientific Computing with SciLab*, Birkhauser, Boston, 1998
- Chancelier J.P. et al., *Introduction à SCILAB (Collection IRIS)* Springer, Paris, 2002

La utilización es igual para las diferentes plataformas, pero obviamente hay algunas diferencias intrínsecas al usar una u otra plataforma. Este pequeño manual se refiere principalmente a la versión 3.0 para Linux y para Windows (la última, a la fecha de hoy: 1 de abril del 2005).

Deseo agradecer a todos los que tuvieron que ver con la creación y actualización de Scilab. Realmente es una herramienta de uso libre muy poderosa. Me ha sido muy útil para mi trabajo personal y para mis clases en la Universidad.

También deseo manifestar mi agradecimiento a todas las personas que de una u otra forma me han ayudado a corregir, modificar, actualizar y mejorar este documento, en particular, a Manuel Mejía.

# Capítulo 2

# GENERALIDADES

Al activar Scilab aparece en la pantalla algo semejante a:

```
        ===========================================
                          Scilab-3.0

                   Copyright (c) 1989-2004
               Consortium Scilab (INRIA, ENPC)
        ===========================================


Startup execution:
  loading initial environment

-->
```

A partir de ese momento se puede escribir al frente de `-->` las órdenes de Scilab. Éstas deben ser acabadas oprimiendo la tecla `Enter`, denominada también `Intro` o simplemente ←┘.

## 2.1 Primeros pasos

La orden

`--> t = 3.5` ←┘

crea la variable `t` y le asigna el valor `3.5` . Además Scilab muestra el resultado de la orden desplegando en pantalla lo siguiente:

```
 t  =

   3.5

-->
```

**De ahora en adelante, en este manual no se indicará la tecla ⟵┘ ni tampoco el "prompt"** `-->` . Por lo tanto deben sobreentenderse. Generalmente tampoco se mostrará el resultado desplegado en la pantalla por Scilab ante una orden recibida.

Al dar la orden

```
T = 4.5;
```

se crea otra variable diferente con nombre `T` y se le asigna el valor `4.5`. Scilab diferencia las letras minúsculas de las mayúsculas. La presencia de punto y coma al final de la orden hace que Scilab no muestre el resultado en la pantalla. Sin embargo, la orden tuvo efecto.

Scilab no hace diferencia entre números enteros y números reales. Los números se pueden escribir usando la notación usual o la notación científica. Por ejemplo, son válidos los siguientes números.

```
3.5
-4.1234
3.14e-10
3.14E-10
0.0023e20
-12.345e+12
```

Al usar la orden

```
who
```

Scilab muestra las variables que está usando en ese momento. Su respuesta es algo semejante a:

```
your variables are...

T         t         startup   ierr      demolist
%scicos_display_mode          scicos_pal          %scicos_menu
%scicos_short       %helps    MSDOS     home      PWD       TMPDIR
percentlib          soundlib  xdesslib  utillib   tdcslib   siglib
s2flib    roblib    optlib    metalib   elemlib   commlib   polylib
autolib   armalib   alglib    intlib    mtlblib   WSCI      SCI
%F        %T        %z        %s        %nan      %inf      $
%t        %f        %eps      %io       %i        %e
```

```
using       5870 elements  out of    1000000.
         and        48 variables out of       1791
```

Ahí aparecen las variables `t` y `T` , definidas anteriormente, y otras variables propias de Scilab.

En las órdenes de Scilab los espacios en blanco antes y después del signo igual no son indispensables. Se podría simplemente escribir `t=3.5` obteniéndose el mismo resultado. Los espacios en blanco sirven simplemente para facilitar la lectura.

Los nombres en Scilab constan hasta de 24 caracteres. El primero debe ser una letra o `$` . Los otros pueden ser letras, números, `#`, `_`, `$`, `!` . Por ejemplo:

```
a12, pesoEsp, valor_ini
```

Cuando en la orden no hay ninguna asignación, sino simplemente una operación válida, Scilab crea o actualiza una variable llamada `ans` . Por ejemplo,

```
t+T
```

produce el resultado

```
   ans  =

      8.
```

Las asignaciones pueden incluir operaciones con variables ya definidas, por ejemplo

```
x = t+T
```

Si se da la orden

```
t = 2*t
```

se utiliza el antiguo valor de `t` para la operación, pero, después de la orden, `t` queda valiendo `7`.

Si se quiere conocer el valor de una variable ya definida, basta con digitar el nombre de la variable y oprimir Enter.

Es posible volver a repetir fácilmente una orden dada anteriormente a Scilab. Utilizando las teclas correspondientes a las flechas hacia arriba y hacia abajo, se obtiene una orden anterior y se activa oprimiendo $\boxed{\longleftarrow\!\rfloor}$. Por ejemplo al repetir varias veces la orden

```
x = (x+3/x)/2
```

se obtiene una aproximación muy buena de $\sqrt{3}$ .

También es posible, por medio de las flechas (hacia arriba y hacia abajo), buscar una orden anterior para editarla y enseguida activarla.

En una misma línea de Scilab puede haber varias órdenes. Éstas deben estar separadas por coma o por punto y coma. Por ejemplo,

```
t1 = 2, t2 = 3; dt = t2-t1
```

## 2.2 Operaciones y funciones

Los símbolos

```
+   -   *   /
```

sirven para las 4 operaciones aritméticas. El signo `-` también sirve para indicar el inverso aditivo. Por ejemplo

```
u = -t
```

Para elevar a una potencia se utiliza el signo `^` o también `**` . Por ejemplo

```
u = 2^8,  v = 2**8
```

Scilab utiliza para agrupar los paréntesis redondos: `(` `)`, como en la orden

```
x = (x+3/x)/2.
```

Puede haber parejas de paréntesis metidas (anidadas) dentro de otras.

En una expresión puede haber varios operadores. Las reglas de precedencia son semejantes a las de la escritura matemática usual. Los paréntesis tienen prioridad sobre todos los operadores. Entre los operadores vistos hay tres grupos de prioridad. De mayor a menor, estos son los grupos de prioridad:

```
^  **
*  /
+  -
```

Entre operadores de igual prioridad, se utiliza el orden de izquierda a derecha. Por ejemplo, `2*3+4^5-6/7` es equivalente a `((2*3)+(4^5))-(6/7)`.

Scilab tiene predefinidas muchas funciones matemáticas. A continuación está la lista de las funciones elementales más comunes.

`abs` : valor absoluto

`acos` : arcocoseno

`acosh` : arcocoseno hiperbólico

`asin` : arcoseno

`asinh` : arcoseno hiperbólico

`atan` : arcotangente

`atanh` : arcotangente hiperbólica

`ceil` : parte entera superior

`cos` : coseno

`cosh` : coseno hiperbólico

`cotg` : cotangente

`coth` : cotangente hiperbólica

`exp` : función exponencial: $e^x$

`fix` : redondeo hacia cero (igual a `int`)

`floor` : parte entera inferior

`int` : redondeo hacia cero (igual a `fix`)

`log` : logaritmo natural

`log10` : logaritmo decimal

`log2` : logaritmo en base dos

`max` : máximo

`min` : mínimo

`modulo` : residuo entero

`rand` : número aleatorio

`round` : redondeo

`sin` : seno

`sinh` : seno hiperbólico

`sqrt` : raiz cuadrada

`tan` : tangente

`tanh` : tangente hiperbólica

El significado de la mayoría de estas funciones es absolutamente claro. La siguiente tabla muestra varios ejemplos utilizando las funciones de parte entera y redondeo.

| x | ceil(x) | floor(x) | int(x) | round(x) |
|---:|---:|---:|---:|---:|
| 2.0 | 2. | 2. | 2. | 2. |
| 1.8 | 2. | 1. | 1. | 2. |
| 1.5 | 2. | 1. | 1. | 2. |
| 1.2 | 2. | 1. | 1. | 1. |
| - 3.1 | - 3. | - 4. | - 3. | - 3. |
| - 3.5 | - 3. | - 4. | - 3. | - 4. |
| - 3.8 | - 3. | - 4. | - 3. | - 4. |

Otra función matemática, ésta ya con dos parámetros de entrada, es `modulo`. Sus dos parámetros deben ser enteros. El resultado es el residuo de la división entera.

```
modulo(17,5)
```

da como resultado 2.

Para tener información más detallada sobre alguna función basta con digitar `help` y a continuación el nombre de la función o de la orden. Por ejemplo

```
help floor
```

Obviamente se requiere que la función `floor` exista. Si no se conoce el nombre de la función, pero se desea buscar sobre un tema, se debe utilizar `apropos`. Por ejemplo:

```
apropos polynomial
```

da información sobre las funciones que tienen que ver con polinomios. En cambio,

```
help polynomial
```

informa que no hay manual para `polynomial`.

Además de estas funciones elementales, Scilab tiene muchas más funciones como las funciones de Bessel, la función gama, ... Mediante la barra de menú, con la opción `Help` seguida de `Help Dialog` se obtiene un catálogo resumido de las herramientas de Scilab.

La lista de funciones elementales de Scilab es la siguiente:

```
abs, acos, acosh, acoshm, acosm, addf, adj2sp, amell, and,
asinh, asinhm, asinm, atan, atanh, atanhm, atanm, besseli,
besselj, besselk, bessely, binomial, bloc2exp, bloc2ss, calerf,
ceil, cmb_lin, conj, cos, cosh, coshm, cosm, cotg, coth, cothm,
cumprod, cumsum, delip, diag, dlgamma, double, erf, erfc,
erfcx, eval, eye, fix, floor, frexp, full, gamma, gammaln,
gsort, imag, int, int16, int32, int8, integrate, interp,
interpln, intersect, intsplin, inttrap, isdef, isinf, isnan,
isreal, kron, ldivf, lex_sort, linspace, log, log10, log2,
logm, logspace, max, maxi, mean, median, min, mini, minus,
modulo, mps2linpro, mtlb_sparse, mulf, nnz, norm, not, ones,
or, pen2ea, pertrans, pmodulo, prod, rand, rat, rdivf, real,
round, sign, signm, sin, sinh, sinhm, sinm, size, smooth,
solve, sort, sp2adj, sparse, spcompack, speye, spget, splin,
spones, sprand, spzeros, sqrt, sqrtm, squarewave, ssprint,
ssrand, st_deviation, subf, sum, sysconv, sysdiag, syslin, tan,
tanh, tanhm, tanm, toeplitz, trfmod, trianfml, tril, trisolve,
triu, typeof, uint16, uint32, uint8, union, unique, zeros.
```

## 2.3 Otros temas

Se puede modificar el formato utilizado por Scilab para mostrar los resultados, mediante `format`. Si se da la orden

```
format(16)
```

a partir de ese momento, Scilab utilizará 16 "columnas" (16 posiciones) para mostrar cada número. Estas 16 columnas incluyen el espacio para el signo la parte entera y el punto. Por defecto, Scilab usa 10 posiciones.

```
format('e',14)
```

La orden anterior sirve para utilizar notación científica con 14 posiciones. También se puede utilizar simplemente `format('e')`

Para regresar al formato inicial, el formato "variable" (el predefinido por Scilab) se usa

```
format('v')
```

o, por ejemplo,

```
format('v', 10)
```

Scilab tiene predefinidas algunas constantes especiales cuyos nombres están precedidos del signo `%` . Para los valores $e$, $\pi$, $\sqrt{-1}$, sus nombres son: `%e` , `%pi` , `%i` . Observe que dos de estas variables aparecieron al utilizar `who`. Después de la siguiente asignación, la variable `r` tendrá el valor `-1`.

```
r = log(1/%e)
```

## 2.4 Complejos

Scilab maneja de manera sencilla los números complejos. Estos pueden ser definidos de varias maneras. Por ejemplo, suponiendo que `r` vale $-5$, las dos órdenes siguientes

```
a = 3 + 4*r*%i
b = sqrt(-4)
```

definen dos variables, `a` $= 3 - 20i$, `b` $= 2i$.

Para las operaciones con números complejos (suma, resta, multiplicación, ...) se utilizan exactamente los mismos símbolos `+ - * / ** ^`.

Las funciones `real`, `imag` y `conj` permiten obtener la parte real, la parte imaginaria y el conjugado de un complejo. Si se utiliza la función `abs` con un complejo, se obtiene la magnitud o módulo de él.

Las funciones de Scilab usadas para funciones reales elementales que tienen generalizaciones en complejos, se pueden usar también para los complejos, por ejemplo, `sin`, `cos`, `log`, ... Así, es completamente lícito

```
z = 3 + 4*%i;  r = sin(z)
```

El resultado mostrado por Scilab en pantalla es

```
r  =

   3.853738 - 27.016813i
```

## 2.5 Polinomios

Un polinomio se puede definir de dos maneras: por sus coeficientes o por sus raíces. Es necesario además indicar la variable simbólica para el polinomio. La orden

```
p = poly([2 3 5 7], "x", "coeff")
```

define en la variable `p` el polinomio $2 + 3x + 5x^2 + 7x^3$. La orden

```
q = poly([2 3 5], "x", "roots")
```

define en la variable `q` el polinomio $-30+31x-10x^2+x^3$ cuyas raíces son exactamente 2, 3 y 5. Escribir `q = poly([2 3 5], "x")` produce exactamente el mismo resultado, o sea, `"roots"` es el tipo de definición por defecto.

La doble comilla `"` puede ser reemplazada por la comilla sencilla `'`. Más aún, se puede reemplazar `'coeff'` por `'c'` y `'roots'` por `'r'` . Es lícito escribir

```
r = poly([6 7 8], 'y', 'c').
```

La función `roots` calcula las raíces de un polinomio, sean éstas reales o complejas. Por ejemplo

```
roots(p)
```

Con polinomios se pueden hacer sumas, multiplicaciones, restas, multiplicación por un número. Deben ser polinomios en la misma variable. Por ejemplo:

```
v = p + q + p*q - 3.1*q
```

También se puede elevar un polinomio a una potencia, por ejemplo,

```
r = p^3
```

La función `coeff` tiene dos parámetros, el primero es el polinomio y el segundo la potencia. La siguiente orden asigna a la variable `k` el valor $-10$, el coeficiente de $x^2$ en el polinomio `q`.

```
k = coeff(q, 2)
```

Si se utiliza simplemente

```
c = coeff(q)
```

se obtendrán todos los coeficientes. La variable `c` será un vector (ver capítulo siguiente sobre matrices y vectores).

Si se utiliza `p = poly(a,'x')`, donde `a` es una matriz cuadrada (ver capítulo siguiente sobre matrices y vectores), se obtiene el polinomio característico de de la matriz `a`.

Para evaluar un polinomio `p` en un valor `t` se usa

```
horner(p, t)
```

Por ejemplo `horner(q, 1)` dará como resultado $-8$. Si `q` es un polinomio, es lícito utilizar la orden

```
r = horner(p, q)
```

para obtener `p(q(x))`.

## 2.6 Lista de algunas herramientas

Scilab tiene numerosas herramientas para diferentes temas. A continuación hay una lista de ellas.

- Álgebra lineal (capítulo 3)
- Gráficas (capítulo 5)
- Optimización
- METANET: grafos y redes
- Análisis y control de sistemas lineales
- Procesamiento de señales
- Modelación y simulación ARMA (autoregressive moving average)
- Simulación
- SCICOS: modelamiento y simulación de sistemas dinámicos híbridos
- Funciones de entrada y salida
- Manejo de funciones y librerías
- Manipulación de cadenas de caracteres
- Diálogos: ventanas y botones
- Cálculos con polinomios

- Funciones de distribución
- Control robusto
- PVM: parallel virtual machine
- Traducciones de lenguajes y datos
- GeCI: comunicación con otras aplicaciones
- Interfaz con Maple

# Capítulo 3

# VECTORES Y MATRICES

En Scilab no hay vectores como tales, los vectores se deben asimilar a matrices de una sola fila o vectores fila (tamaño $1 \times n$) o a matrices de una sola columna o vectores columna (tamaño $n \times 1$).

## 3.1 Creación

La matriz

$$\begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \end{bmatrix}$$

se puede definir por medio de

```
a = [ 11 12 13 14 15; 21 22 23 24 25; 31 32 33 34 35]
```

o también por medio de

```
a = [ 11,12,13,14,15; 21,22,23,24,25; 31,32,33,34,35]
```

o por

```
a = [ 11 12 13 14 15
21 22 23 24 25
31 32 33 34 35]
```

Scilab mostrará el siguiente resultado en la pantalla:

```
 a  =

!   11.     12.     13.     14     15. !
!   21.     22.     23.     24     25. !
!   31.     32.     33.     34     35. !
```

La definición de la matriz se hace por filas. Los elementos de una misma fila se separan por medio de espacios en blanco o por medio de comas. Una fila se separa de la siguiente por medio de punto y coma o por medio de cambio de línea.

Scilab permite crear rápidamente algunos tipos especiales de matrices:

`ones(4,5)` es una matriz de unos de tamaño $4 \times 5$

`zeros(4,5)` es una matriz de ceros de tamaño $4 \times 5$

`rand(20,30)` es una matriz aleatoria de tamaño $20 \times 30$

`eye(4,4)` es la matriz identidad de orden 4

Algunas veces es útil, especialmente para gráficas de funciones (tema que se vera en un capítulo posterior), crear vectores con elementos igualmente espaciados, por ejemplo

```
x = [1 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6]
```

Esto también se puede hacer más fácilmente así:

```
x = 1:0.2:2.6
```

Los valores 1.0, 0.2 y 2.6 corresponden respectivamente al límite inferior, al incremento y al límite superior. Si no se especifica incremento se supone que es uno. Escribir `y = 2:9` es equivalente a `y = 2:1:9`. Cuando se desea una columna basta con utilizar el operador de trasposición, por ejemplo, `z = (1:0.2:2.6)'`

## 3.2 Notación y operaciones

`a(2,5)` denota el elemento o entrada de `a` en la fila 2 y en la columna 5.

`a(3,:)` denota la tercera fila de la matriz `a`.

`a(:,4)` denota la cuarta columna de la matriz `a`.

`a(1:2,2:5)` denota la submatriz de tamaño $2 \times 4$, formado por los elementos que están en las filas 1, 2 y en las columnas 2, 3, 4, 5, o sea, la matriz

$$\begin{bmatrix} 12 & 13 & 14 & 15 \\ 22 & 23 & 24 & 25 \end{bmatrix}.$$

La anterior definición de submatrices se puede generalizar utilizando vectores(fila o columna). Por medio de las siguientes órdenes

```
u = [ 1 2 1],  v =[2 4], aa = a(u,v)
```

se asigna a la variable `aa` la matriz

$$\begin{bmatrix} 12 & 14 \\ 22 & 24 \\ 12 & 14 \end{bmatrix}.$$

Si `a` es una matriz $m \times n$, entonces `a(:)` es un vector columna de tamaño $mn \times 1$, obtenido colocando primero la primera columna de `a`, enseguida la segunda columna, luego la tercera y así sucesivamente.

Si `x` es un vector columna, entonces `x(:)` es el mismo vector columna. Si `x` es un vector fila, entonces `x(:)` es lo mismo que `x'`. Dicho de otra forma, si `x` es un vector (fila o columna), entonces `x(:)` es un vector columna con las mismas entradas de `x`. Así, `x = x(:)` convierte el vector `x` en un vector columna.

Si `x` es un vector fila, se puede escribir `x(3)` en lugar de `x(1,3)`. Análogamente, si `y` es un vector columna, se puede escribir `y(2)` en lugar de `y(2,1)`.

Por medio de `*` se puede hacer el producto entre un número y una matriz. Los signos `+` y `-` permiten hacer sumas y restas entre matrices del mismo tamaño. Cuando el producto de matrices es posible (número de columnas de la primera matriz igual al número de filas de la segunda), éste se indica por medio de `*`. La transposición de una matriz se indica por medio de comilla, por ejemplo,

```
c = a'*a
```

crea la matriz `c`, producto de la traspuesta de `a` y de `a`. Por construcción `c` es una matriz cuadrada.

La mayoría de las funciones de Scilab utilizadas para números reales se pueden aplicar a matrices. En este caso se obtiene una matriz del mismo tamaño, en la que se ha aplicado la función a cada uno de los elementos de la matriz. Por ejemplo, las dos órdenes siguientes son equivalentes.

```
D = sin([1 2; 3 4]), D = [sin(1) sin(2); sin(3) sin(4)]
```

Para matrices (y vectores) del mismo tamaño se puede hacer la multiplicación elemento a elemento utilizando `.*` . De manera análoga se puede hacer la división elemento a elemento. Por ejemplo:

```
P = rand(3,5),  Q = rand(3,5),  r = P.*Q
```

También es posible elevar a una potencia los elementos de una matriz, por ejemplo,

```
H = a.^3
```

Si `a` es una matriz rectangular

```
a.^(1/2)
```

es lo mismo que

```
sqrt(a)
```

es decir, una matriz del mismo tamaño, cuyas entradas son las raices cuadradas de las entradas de `a`.

En cambio, si `a` es una matriz cuadrada,

```
a1 = a^(1/2)
```

es una matriz raiz cuadrada de `a`, o sea, `a1*a1` es lo mismo que `a`.

Como se verá más adelante, para graficar se requieren dos vectores, uno con los valores de la variable $x$ y otro (del mismo tamaño) con los valores de la variable $y$. Las dos órdenes siguientes permiten crear los dos vectores para un polinomio:

```
x = (-2:0.01:3)';
```

```
y = 3*x.^4 + x.^3 - 5*x.*x + 3.14;
```

Para matrices cuadradas, es posible calcular directamente una potencia. Por ejemplo,

```
G = rand(6,6)^3
```

Si los tamaños son compatibles, dos o más matrices se pueden "pegar" para obtener una matriz de mayor tamaño, por ejemplo,

```
AA = [a rand[3,2)]
```

```
B = [rand(2,5); a; eye(5,5)]
```

Como `a` fue definida de tamaño $3 \times 5$, entonces `AA` es de tamaño $3 \times 7$ y `B` es de tamaño $10 \times 5$.

La orden `y = []` permite crear la matriz `y` de tamaño $0 \times 0$.

Si `x` es un vector, la orden `x(2) = []` suprime la segunda componente y desplaza el resto. Por ejemplo,

```
x = [2 3 5 7 11]; x(2) = []
```

produce el resultado

```
 x  =

!   2.    5.    7.    11. !
```

De manera análoga, si `a` es un matriz, la orden `a(:,2) = []` suprime la segunda columna.

## 3.3 Funciones elementales

Hay numerosas funciones de Scilab para el manejo de matrices. Algunas de las más usadas son:

- `rank(a)` calcula el rango de `a`.
- `det(c)` determinante de una matriz cuadrada `c`.
- `inv(c)` inversa de una matriz cuadrada e invertible `c`.

- `rref(a)` matriz escalonada reducida por filas equivalente a `a`.
- `expm(C)` produce la matriz $e^C$, donde `C` es una matriz cuadrada.

$$e^C = I + C + \frac{1}{2}C^2 + \frac{1}{6}C^3 + \frac{1}{24}C^4 + \cdots$$

- `diag(c)` produce un vector columna con los elementos diagonales de la matriz cuadrada `c`.
- `diag(x)` produce una matriz diagonal con los elementos del vector (fila o columna) `x`.
- `y = sort(x)` ordena el vector `x` de manera decreciente.
- `[y, k] = sort(x)`: `y` es el vector ordenado de manera decreciente, `k` es un vector que contiene los índices del ordenamiento, o sea, `y = x(k)`.
- `b = sort(a)` ordena la matriz `a` de manera decreciente, considerando cada matriz como un vector formado por la primera columna, la segunda columna, ..., la última columna.

$$\mathtt{a} = \begin{bmatrix} 3.2 & 3.1 \\ 3.4 & 3.8 \end{bmatrix}, \qquad \mathtt{b} = \begin{bmatrix} 3.8 & 3.2 \\ 3.4 & 3.1 \end{bmatrix}$$

- `b = sort(a, 'r')` ordena la matriz `a` de manera decreciente por columnas. **Atención**, aunque `'r'` tiene un significado interno de filas, el resultado externo es un ordenamiento de las columnas.

$$\mathtt{b} = \begin{bmatrix} 3.4 & 3.8 \\ 3.2 & 3.1 \end{bmatrix}$$

- `b = sort(a, 'c')` ordena la matriz `a` de manera decreciente por filas. **Atención**, aunque `'c'` tiene un significado interno de columnas, el resultado externo es un ordenamiento de las filas.

$$\mathtt{b} = \begin{bmatrix} 3.2 & 3.1 \\ 3.8 & 3.4 \end{bmatrix}$$

- `m = max(x)` calcula el máximo del vector (fila o columna) `x`.
- `[m, k] = max(x)`: `m` es el máximo del vector `x`, `k` indica la posición donde está el máximo.
- `m = max(a)` calcula el máximo de la matriz `a`.

- `[m, k] = max(a)`: `m` es el máximo de la matriz `a`, `k` es un vector $1 \times 2$ e indica la posición donde está el máximo.
- `m = max(a, 'r')` : `m` es un vector fila (row) que contiene los máximos de las columnas de `a`.
- `[m, k] = max(a, 'r')` : `m` es un vector fila que contiene los máximos de las columnas de `a`, `k` es un vector fila que contiene las posiciones de los máximos.
- `min` semejante a `max` pero para el mínimo.
- `m = mean(x)` calcula el promedio del vector (fila o columna) `x`.
- `m = mean(a)` calcula el promedio de la matriz `a`.
- `m = mean(a, 'r')` : `m` es un vector fila (row) que contiene los promedios las columnas de `a`.
- `m = mean(a, 'c')` : `m` es un vector columna que contiene los promedios las filas de `a`.
- `median` semejante a `mean` pero para la mediana.
- `st-deviation` semejante a `mean` pero para la desviación estándar.
- `sum` semejante a `mean` pero para la suma.
- `prod` semejante a `mean` pero para el producto.
- `norm(x)` calcula la norma euclidiana del vector `x` (fila o columna).
- `norm(x, p)` calcula la norma $l_p$ del vector `x`:

$$\left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}.$$

- `norm(x, 'inf')` calcula la norma "del máximo" del vector `x`:

$$\max_{1\leq i\leq n} |x_i|.$$

- `norm(a) = norm(a, 2)` calcula, para la matriz `a`, la norma matricial generada por la norma euclidiana, o sea, el mayor valor singular de `a`.
- `norm(a, 1)` calcula, para la matriz `a`, la norma matricial generada por la norma $l_1$, o sea,

$$\max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}|.$$

- `norm(a, 'inf')` calcula, para la matriz `a`, la norma matricial generada por la norma $l_\infty$, o sea,

$$\max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}|.$$

- `norm(a, 'fro')` calcula, para la matriz `a`, la norma de Frobenius, o sea,

$$\left( \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2 \right)^{1/2} .$$

- La orden `fc = size(a)` proporciona un vector fila $1 \times 2$ con el número de filas y el número de columnas de `a`. La orden `size(a,1)` proporciona el número de filas de `a`. Análogamente, `size(a,2)` proporciona el número de columnas de `a`.

- Por medio de `tril` se puede obtener la parte triangular inferior (low) de una matriz cuadrada. Por ejemplo,

```
a = [1 2 3; 4 5 6; 7 8 9]; L = tril(a)
```

  produce el resultado

```
 L  =

!   1.    0.    0. !
!   4.    5.    0. !
!   7.    8.    9. !
```

- La función `triu` produce la parte triangular superior (upper).

- Si `a` es una matriz cuadrada, por medio de

```
v = spec(a)
```

  se obtiene un vector columna con los valores propios (reales o complejos) de `a`.

  Con la orden

```
[v, s] = spec(a)
```

  se obtiene la matriz `v` cuyas columnas son vectores propios y la matriz diagonal `s` cuyos elementos diagonales son los valores propios.

- Si $A$ es una matriz simétrica y definida positiva, entonces se puede factorizar en la forma $A = U^{\mathrm{T}} U$, donde $U$ es una matriz triangular superior de diagonal positiva. Esta es la llamada factorización de Cholesky. Esta matriz se puede obtener por

```
U = chol(A)
```

- La factorización QR de una matriz $A$ es la expresión

$$A = QR$$

donde $Q$ es una matriz ortogonal y $R$ es una matriz del tamaño de $A$, "*triangular*" (no necesariamente cuadrada). Por ejemplo,

```
[q, r] = qr(a)
```

- La factorización LU de una matriz cuadrada invertible $A$ es la expresión

$$PA = LU$$

donde $P$ es una matriz de permutación, $L$ es una matriz triangular inferior con unos en la diagonal y $U$ es una matriz triangular superior. Por ejemplo,

```
[L, U, P] = lu(A)
```

- La descomposición SVD, descomposición en valores singulares, de una matriz $A$ es la expresión

$$A = UDV^{\mathrm{T}}$$

donde $U$ y $V$ son matrices ortogonales y $D$ es una matriz del tamaño de $A$, "*diagonal*" (no necesariamente cuadrada), cuyos elementos diagonales son los valores singulares de $A$. Por ejemplo,

```
[U, D, V] = svd(A)
```

- La inversa generalizada de Moore-Penrose o seudoinversa de una matriz se puede obtener por medio de

```
a1 = pinv(a)
```

## 3.4 Solución de sistemas de ecuaciones

En matemáticas (y todas las ciencias y técnicas relacionadas) uno de los problemas más frecuentes, o posiblemente el más frecuente, consiste en resolver un sistema de ecuaciones lineales $Ax = b$, donde se conocen la matriz $A$ y el vector columna $b$.

Si $A$ es una matriz cuadrada e invertible, el sistema tiene, teóricamente, una única solución y se puede resolver por una de las dos órdenes siguientes. La primera conlleva el cálculo de la inversa. La segunda usa un algoritmo eficiente de Scilab para hallar la solución.

```
x1 = inv(A)*b

x2 = A\b
```

Teóricamente, el resultado debe ser el mismo. Desde el punto de vista de precisión numérica, los resultados son semejantes. Para matrices medianamente grandes hay diferencias en tiempo. La primera forma gasta, aproximadamente, tres veces más tiempo que la segunda.

Fuera del caso de solución única hay otros dos casos: el caso de sistema inconsistente (sin solución) y el caso de muchas soluciones (número infinito de soluciones).

Si el sistema es inconsistente, entonces se desea encontrar una seudosolución, la solución por mínimos cuadrados, o sea, se busca $x$ que minimice $||Ax-b||_2^2$. En este caso la orden

```
x = A\b
```

encuentra una de estas"soluciones". Si las columnas de $A$ son linealmente independientes, esta seudosolución es única. Para mayor información, use la ayuda:

```
help backslash
```

## 3.5 Otras funciones

La lista de funciones de Scilab para Álgebra Lineal es la siguiente:

```
aff2ab, balanc, bdiag, chfact, chol, chsolve, classmarkov,
coff, colcomp, companion, cond, det, eigenmarkov, ereduc, exp,
expm, fstair, fullrf, fullrfk, genmarkov, givens, glever,
gschur, gspec, hess, householder, im_inv, inv, kernel, kroneck,
linsolve, lu, ludel, lufact, luget, lusolve, lyap, nlev, orth,
pbig, pencan, penlaur, pinv, polar, proj, projspec, psmall, qr,
quaskro, randpencil, range, rank, rcond, rowcomp, rowshuff,
rref, schur, spaninter, spanplus, spantwo, spchol, spec,
sqroot, sva, svd, sylv, trace
```

# Capítulo 4

# PROGRAMAS

En Scilab hay dos tipos de programas: los guiones o libretos (scripts) y las funciones. Un guión es simplemente una secuencia de órdenes de Scilab. No tiene parámetros ("argumentos") de entrada ni de salida. En cambio una función sí los tiene.

Por otro lado, las variables definidas en un guión son globales, es decir, después del llamado del guión estas variables siguen existiendo. En cambio en una función, las variables definidas dentro de la función dejan de existir una vez finalizada la ejecución de la función, son variables locales.

## 4.1 Guiones (scripts)

Un guión es simplemente un archivo ASCII en el que hay una sucesión de órdenes de Scilab. Generalmente tiene la extensión `.sce` pero eso no es obligatorio. Puede estar colocado en cualquier carpeta.

En el ejemplo que sigue se va a hacer lo siguiente:

crear aleatoriamente una matriz,

crear aleatoriamente la solución $x^0$,

crear los términos independientes correspondientes a $x^0$,

hallar la solución,

calcular la norma del error cometido.

Scilab viene con un editor llamado Scipad. Se activa mediante la opción `Editor` de la barra de menú de Scilab.

Ya sabiendo lo que se va a hacer, con Scipad o con un editor cualquiera, creamos el archivo

```
c:\misci\ensayo01.sce
```

cuyo contenido sea el siguiente:

```
n = 100;
A = rand(n,n);
x0 = rand(n,1);
b = A*x0;
x = A\b;
residuo = norm(x-x0)
```

Obviamente el nombre de la carpeta `misci` es simplemente un ejemplo y hubiera podido ser cualquier otra carpeta. Una vez guardado el contenido, desde el ambiente Scilab se da la orden

```
exec c:\misci\ensayo01.sce
```

Esto hace que se ejecuten todas las órdenes contenidas en el archivo. Mediante `who`, o de cualquier otra forma, se puede verificar que las variables `n, A, x0`, ... fueron creadas y todavía existen.

Dar la orden `exec c:\misci\ensayo01.sce` también se puede hacer en Windows por medio de la barra de menú con las opciones `File` y después `Exec`. Subiendo y bajando de nivel se busca la carpeta adecuada y se hace doble clic con el botón derecho del ratón en el archivo `ensayo01.sce`.

En Linux se hace de manera muy parecida: en la barra de menú `File`, enseguida `File Operations` y finalmente `Exec` después de haber escogido el archivo deseado.

Si se usa SciPad, se obtiene el mismo resultado con la barra de menú (de SciPad) mediante la opción `Execute` y después `Load into Scilab`.

Si se desea, se puede editar el archivo, hacer algunos cambios, guardarlos y de nuevo activar el guión mediante `exec ...`

En el siguiente ejemplo suponemos que se tiene una matriz $A$ de tamaño $m \times n$, $m \geq n$, $r(A) = n$, es decir, tiene más filas que columnas y sus columnas son linealmente independientes. Consideremos el siguiente sistema de ecuaciones

$$A\,x = b.$$

Probablemente este sistema de ecuaciones no tiene solución en el sentido estricto. Su seudosolución o solución por mínimos cuadrados está dada por la solución de la ecuación normal

$$A^{\mathrm{T}} A\,x = A^{\mathrm{T}} b.$$

proveniente de minimizar el cuadrado de la norma del error:

$$\varepsilon = ||A\,x - b||_2^2.$$

Sea `c:\estad\ensayo02.sce` el archivo que define los datos y lleva a cabo este proceso. Su contenido puede ser:

```
// solucion por minimos cuadrados
//
a = [ 1 2; 3 4; 5 6];
b = [ 3 7 10]';
//
x = (a'*a)\(a'*b)
e = norm(a*x-b)^2
```

Las líneas que empiezan por `//` son líneas de comentarios (como en C++).

Obviamente para activar este otro archivo se necesita dar la orden

```
exec c:\estad\ensayo02.sce
```

De manera natural aparece la pregunta: *¿Cómo hacer el mismo proceso con otro sistema de ecuaciones sin tener que cambiar el archivo?* Las funciones son la respuesta.

## 4.2 Funciones

En un archivo ASCII puede haber varias funciones. Generalmente el nombre de los archivos de funciones tienen la extensión `.sci`. El esquema general de una función es el siguiente:

```
function [res1, res2, ...] = nombrefuncion(par1, par2, ...)
    ...
    ...
endfunction
```

El significado de `res1` es resultado 1 o también parámetro de salida 1. El significado de `par2` es parámetro de entrada 2 o simplemente parámetro 2. Cuando la función tiene un único resultado, el esquema puede ser simplemente:

```
function result = nombrefuncion(par1, par2, ...)
    ...
    ...
endfunction
```

El siguiente archivo llamado `c:\misci\misfunc.sci` tiene varias funciones

```
function [x, error2] = seudoSol(A, b)
    // solucion por minimos cuadrados
    x = (A'*A)\(A'*b)
    error2 = norm(A*x-b)^2
```

```
endfunction
//----------------------------------------------------------
function [x, y] = polarCart(r, t)
    // Conversion de coordenadas polares a cartesianas.
    x = r*cos(t)
    y = r*sin(t)
endfunction
//----------------------------------------------------------
function [x, y] = polarCartGr(r, t)
    // Conversion de coordenadas polares a cartesianas,
    // el angulo esta dado en grados.
    [x, y] = polarCart(r, t*%pi/180)
endfunction
//----------------------------------------------------------
function fx = f(x)
    fx = x(1)^2 + x(2)^2
endfunction
```

Este archivo de funciones se debe cargar mediante la orden

```
getf c:\misci\misfunc.sci
```

o también, en Windows, mediante las opciones de la barra de menú `File` y después `Exec`. En Linux, en la barra de menú `File` , `File Operations` y finalmente `Getf` después de haber escogido el archivo deseado.

También con la barra de menú de SciPad: `Execute` y `Load into Scilab`.

Una vez cargado el archivo, las funciones se pueden utilizar como las otras funciones de Scilab. Por ejemplo, son válidas las siguientes órdenes:

```
[x1, y1] = polarCart(2, 0.7854)
[u, v] = polarCartGr(3, 30)
a = [1 2; 3 4; 5 6]; b = [3 7 10]'; [x,err] = seudoSol(a,b)
[x, dif] = seudoSol([1 2; 3 4; 5 6], [3 7 10]')
valor = f([3; 4])
x = [5; 6], res = f(x)
```

Cuando una función produce más de un resultado, también se puede utilizar asignando menos de los resultados previstos. Por ejemplo, la utilización completa de `polarCartGr` puede ser

```
[a, b] = polarCartGr(3,30)
```

lo cual produce el resultado

```
b  =
```

```
    1.5
a  =

    2.5980762
```

En cambio, la orden

```
c = polarCartGr(3,30)
```

produce el resultado

```
c  =

    2.5980762
```

Otra característica de las funciones es que una función puede llamar una o varias funciones. Obsérvese que la función `polarCartGr` utiliza la función `polarCart`.

En los ejemplos anteriores de funciones, hay simplemente una secuencia de órdenes que siempre se repite de la misma manera. Con mucha frecuencia esto no es así. Normalmente dentro de una función, dependiendo de los datos o de resultados intermedios, hay que tomar algunas decisiones, repetir un proceso, abortar un proceso, ... Esto se logra mediante los operadores relacionales, los operadores lógicos y las estructuras de control. Esto se verá en secciones posteriores de este mismo capítulo.

## 4.3 Carpeta actual o por defecto

Al arrancar, Scilab tiene definido un subdirectorio (o carpeta) preferencial, actual o por defecto. Para saber el nombre de esta carpeta, se da la orden

```
pwd
```

La respuesta de Scilab puede ser:

```
ans  =

C:\WINDOWS\Escritorio
```

En Linux la respuesta podría ser

```
ans  =

/home/h
```

Esto quiere decir que si un archivo de funciones o un archivo tipo guión están ubicados allí, no es necesario, al utilizar `getf` o `exec`, escribir la ubicación completa del archivo, basta dar el nombre del archivo. Por ejemplo, si el archivo `ejemplo4.sce` está en la carpeta `C:\WINDOWS\Escritorio`, no es necesario, dar la orden

```
exec \WINDOWS\Escritorio\ejemplo4.sce
```

Basta digitar

```
exec ejemplo4.sce
```

Para decir a Scilab que cambie la carpeta por defecto, se usa la orden `chdir`, por ejemplo,

```
chdir 'c:\algebra\matrices\'
```

## 4.4 Comparaciones y operadores lógicos

| | |
|---|---|
| `<` | menor |
| `<=` | menor o igual |
| `>` | mayor |
| `>=` | mayor o igual |
| `==` | igual |
| `~=` | diferente |
| `<>` | diferente |
| `&` | y |
| `\|` | o |
| `~` | no |

## 4.5 Órdenes y control de flujo

La principales estructuras de control de Scilab son:

- `if`
- `select` y `case`
- `for`
- `while`

Otras órdenes relacionadas con el control de flujo son:

- `break`

- `return` equivalente a `resume`
- `abort`

En este manual introductorio **se supone que el lector tiene algún conocimiento de un lenguaje de programación**. Aquí no se explica como es el funcionamiento de la estructura `while`. Simplemente hay algunos ejemplos sobre su escritura en Scilab.

### 4.5.1 if

Una forma sencilla de la escritura de la escritura `if` es la siguiente:

```
if condición then
    ...
    ...
end
```

La palabra `then` puede ser reemplazada por una coma o por un cambio de línea. Entonces se puede escribir

```
if condición
    ...
    ...
end
```

Obviamente también existe la posibilidad `else`:

```
if condición
    ...
else
    ...
end
```

Estas estructuras se pueden utilizar directamente dentro del ambiente interactivo de Scilab. Por ejemplo, se puede escribir directamente, sin necesidad de un guión

```
if x < 0, fx = x*x, else fx = x*x*x, end
```

Resulta ineficiente hacer un guión únicamente para las órdenes de la línea anterior. Pero, cuando hay muchas órdenes y muchos controles y es necesario depurar el proceso, es casi indispensable hacer un guión o una función.

### 4.5.2 for

La estructura `for` tiene la siguiente forma:

```
for var = lim1:incr:lim2

    ...

    ...

end
```

Esto quiere decir que la variable `var` empieza en el límite inferior `lim1`, después va incrementar el valor en el incremento `incr` (puede ser negativo). Si el incremento es 1, éste se puede suprimir. A continuación algunos ejemplos.

```
for i = 2:3:14
    ...
    ...
end
for j = 2:-3:-10
    ...
    ...
end
for k = 2:8
    ...
    ...
end
```

Una estructura `for` puede estar anidada dentro de otro `for` o dentro de un `if`.

### 4.5.3 while

La forma general es:

```
while condición

    ...

    ...

end
```

Por ejemplo

```
e = 1;
while e+1 > 1
    e = e/2;
end
```

### 4.5.4 select

La forma general es:

```
select variable
    case valor1 then
        ...
        ...
    case valor2 then
        ...
        ...
    case valor3 then
        ...
        ...
    case valor4 then
        ...
        ...
    ...
    else
        ...
        ...
end
```

La palabra `then` puede ser reemplazada por una coma o por un cambio de línea. La parte `else` es opcional. Entonces se puede escribir

```
select variable
    case valor1
        ...
        ...
    case valor2, ...
    case valor3
        ...
        ...
    case valor4
```

```
        ...
        ...
end
```

Por ejemplo

```
    select indic
        case 0, a = b;
        case 1
            a = b*b;
            b = 1;
        case 3
            a = max(a,b);
            b = a*a;
        else
            a = 0;
            b = 0;
    end
```

### 4.5.5 Otras órdenes

La orden `break` permite la salida forzada (en cualquier parte interna) de un bucle `for` o de un bucle `while`.

La orden `return` permite salir de una función antes de llegar a la última orden. **Todos los parámetros de salida o resultados deben estar definidos con anterioridad**.

Otra orden que sirve para interrumpir una función, en este caso interrumpiendo la evaluación, es `abort`.

En la siguiente función se calcula el máximo común divisor por el algoritmo de Euclides.

```
    function [maxcd, indic] = mcd(a, b)
        // Maximo comun divisor de  a, b enteros positivos.
        // indic valdra  1  si se calculo el  m.c.d
        //               0  si los datos son inadecuados.

        indic = 0
        maxcd = 0
        if round(a) ~= a | round(b) ~= b
            return
        end
        if a < 1 | b < 1
            return
```

```
    end
    if a < b
        t = a
        a = b
        b = t
    end
    indic = 1
    while 1 == 1
        r = modulo(a, b)
        if r == 0
            maxcd = b
            return
        end
        a = b
        b = r
    end
endfunction
```

En el ejemplo anterior, el último `return` está anidado en un solo bucle y después acaba la función, luego se podría cambiar por un `break`. La condición del `while` siempre es cierta, luego la salida del bucle se obtiene siempre en la mitad del cuerpo del `while`.

En una función no es necesario el punto y coma después de una orden, pues de todas maneras Scilab no muestra el resultado de la asignación (en Matlab si es necesario el punto y coma). Si definitivamente, en una función, se desea mostrar en pantalla algunos valores intermedios se debe hacer por medio de `disp` o por medio de `printf`. Por ejemplo, después del cálculo de `r` se puede utilizar la orden

```
disp(a, b, r)
```

Observe que primero escribe el valor de `r`, después el de `b` y finalmente el de `a`.

La orden `printf` es una emulación de la orden del mismo nombre del lenguaje C. Por ejemplo se podría utilizar la orden

```
printf(' a = %3d  b = %3d  r = %3d\n', a, b, r)
```

Esta emulación utiliza una comilla en lugar de la comilla doble. El símbolo `\n` se utiliza para cambio de línea.

Otras órdenes que se pueden utilizar para escritura son:

```
fprintf
```

```
print
```

```
write
```

## 4.6 Ejemplos

Los siguientes ejemplos, son simplemente casos que pretenden ilustrar la manera de programar en Scilab. En ellos se busca más la claridad que la eficiencia numérica.

### 4.6.1 Cálculo numérico del gradiente

En este ejemplo se desea aproximar numéricamente el gradiente de una función, que va de $\mathbb{R}^n$ en $\mathbb{R}$, cuyo nombre es exactamente `f`. Cuando se desea cambiar de función, es necesario editar la función `f` e introducir los cambios necesarios. La aproximación utilizada es

$$\frac{\partial f}{\partial x_i}(\bar{x}) \approx \frac{f(\bar{x} + he^i) - f(\bar{x} - he^i)}{2h},$$

donde $e^1$, $e^2$, ..., $e^n$ forman la base canónica. A continuación aparece la función `f` y la función que aproxima el gradiente.

```
//----------------------------------------------------------
function fx = f(x)
    fx = 3*x(1)^2 + 5*x(2)
endfunction
//----------------------------------------------------------
function g = gradf(x)
    // gradiente de la funcion  f  en el punto  x
    //********
    h = 0.001
    //********
    h2 = 2*h
    n = max(size(x))
    g = zeros(n,1)
    for i =1:n
        xi = x(i)
        x(i) = xi+h
        f1 = f(x)
        x(i) = xi-h
        f2 = f(x)
        x(i) = xi
        g(i) = (f1-f2)/h2
    end
endfunction
```

Habiendo cargado, mediante `getf ...`, el archivo que contiene estas dos funciones, su uso se puede hacer mediante órdenes semejantes a:

```
x = [3; 4];  gr = gradf(x)
```

Si se desea una función que evalúe el gradiente de varias funciones diferentes sin tener que editar cada vez la misma función `f`, entonces se debe pasar la función como uno de los parámetros.

```
//----------------------------------------------------------
function fx = func1(x)
    fx = 3*x(1)^2 + 5*x(2)
endfunction
//----------------------------------------------------------
function fx = func2(x)
    fx = 3*x(1) + 5*x(2)^3
endfunction
//----------------------------------------------------------
function g = grad(funcion, x)
    // gradiente de funcion en el punto  x
    //********
    h = 0.001
    //********
    h2 = 2*h
    n = max(size(x))
    g = zeros(n,1)
    for i =1:n
        xi = x(i)
        x(i) = xi+h
        f1 = funcion(x)
        x(i) = xi-h
        f2 = funcion(x)
        x(i) = xi
        g(i) = (f1-f2)/h2
    end
endfunction
//----------------------------------------------------------
```

El llamado se puede hacer así:

```
x = [3; 4],  gr = grad(func1, x), fp = grad(func2, x)
```

### 4.6.2 Matriz escalonada reducida por filas

En el siguiente ejemplo se obtiene la matriz escalonada reducida por filas de una matriz dada. La única diferencia con respecto al método tradicional que se ve en los cursos de

Álgebra Lineal es que se hace pivoteo parcial, es decir, se busca que el pivote sea lo más grande posible en valor absoluto con el fin de disminuir los errores de redondeo. Si la posición del pivote es $(i, j)$, entonces se busca el mayor valor absoluto en las posiciones $(i, j)$, $(i + 1, j)$, ..., $(m, j)$ y se intercambia la fila $i$ con la fila del mayor valor absoluto.

Esta función `merf` permite escribir en pantalla resultados parciales de los cálculos mediante la función `escrMatr` que a su vez llama la función `escrVect`.

En `merf` se utiliza `argn(2)` que indica el número de parámetros en el llamado a la función y permite definir los dos últimos por defecto. El llamado `merf(A)` es equivalente a `merf(A, 1.0e-12, 0)` . El llamado `merf(A, 1.0e-7)` es equivalente a `merf(A, 1.0e-7, 0)`. Para una matriz cualquiera, la orden `norm(rref(a)-merf(a))` debe producir un valor muy cercano a cero.

```
function escrVect(x, titulo)

  // escribe un  vector  en la pantalla

  //********************
  formato = '%10.4f'
  numNumerosPorLinea = 5
  //********************

  nnl = numNumerosPorLinea

  [n1, n2]= size(x)
  n = max(n1,n2)
  m = min(n1,n2)
  if m > 1
      printf('ESCRVECT: x  no es un vector.\n')
  end
  x = x(:)
  if argn(2) == 2
      printf('%s :\n', titulo)
  end
  for i=1:n
    printf(formato, x(i))
    if modulo(i, nnl) == 0 | i == n
      printf('\n')
    end
  end
endfunction
//---------------------------------------------------------
function escrMatr(A, titulo)
  // escribe una  matriz  en la pantalla
```

```
  if argn(2) == 2
     printf('%s :\n', titulo)
  end
  [m, n] = size(A)
  for i=1:m
    escrVect(A(i,:))
  end
endfunction
//----------------------------------------------------
function a = merf(A, eps0, IRP)

  // Matriz escalonada reducida por filas.
  // Se hace pivoteo parcial para buscar para
  // disminuir los errores numericos.
  // Si |t| <= eps0, se considera nulo.
  // Si IRP = 1 se escriben resultados parciales.
  //          0 no se escriben los resultados parciales

  if argn(2) == 1
    eps0 = 1.0e-12
    IRP = 0
  end
  if argn(2) == 2
    IRP = 0
  end

  if IRP == 1, escrMatr(a, 'matriz inicial'), end

  [m, n] = size(a)
  eps0 = eps0*max(abs(A))

  a = A
  i = 1
  j = 1

  while i <= m & j <= n
    [vmax, i0] = max(abs(a(i:m,j)))
    if vmax > eps0
      imax = i+i0-1
      if imax ~= i
        t = a(i,j:n)
        a(i,j:n) = a(imax,j:n)
        a(imax,j:n) = t
```

```
        if IRP == 1
          printf('interc. filas %d y %d\n', i, imax)
          escrMatr(a)
        end
      end
      aij0 = a(i,j)
      a(i,j+1:n) = a(i,j+1:n)/a(i,j)
      a(i,j) = 1
      if IRP == 1
        printf('dividir fila %d por el pivote: %f\n', i, aij0)
        escrMatr(a)
      end

      for k = 1:m
        if k ~= i
          a(k,j+1:n) = a(k,j+1:n) - a(k,j)*a(i,j+1:n)
          a(k,j) = 0.0
        end
      end
      if IRP == 1
        printf('buscar ceros en la columna %d\n', j)
        escrMatr(a)
      end
      i = i+1
    else
      a(i:m,j) = zeros(m+1-i,1)
    end
    j = j+1
  end
endfunction
```

### 4.6.3 Aproximación polinomial por mínimos cuadrados

Dados $m$ puntos $(x_1, y_1)$, $(x_2, y_2)$, ..., $(x_m, y_m)$, tales que $x_i \neq x_j$ si $i \neq j$ (no hay dos $x_i$ iguales), se desea encontrar un polinomio de grado $n \leq m-1$ que pase lo más cerca posible de estos puntos, en el sentido de mínimos cuadrados. Si $n = m-1$, el polinomio pasa exactamente por los $m$ puntos. Los cálculos que hay que hacer, utilizando las ecuaciones normales, son los siguientes:

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix},$$

$$\text{resolver } (A^{\mathrm{T}}A)\, c = A^{\mathrm{T}}y,$$

donde,

$$\begin{aligned} c &= \begin{bmatrix} c_0 & c_1 & c_2 & \dots & c_n \end{bmatrix}^{\mathrm{T}}, \\ p(x) &= c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n, \\ \tilde{y} &= Ac. \end{aligned}$$

Si $n = m - 1$, no hay que resolver el sistema dado por las ecuaciones normales, se resuelve directamente,

$$A\, c = y.$$

```
function [p, yy, ind] = aproxPol(x, y, n)
  // Dados los vectore  x, y, se desea
  // buscar p, el polinomio de aproximacion por
  // minimos cuadrados, tal que
  // p(x(i)) ~ y(i)
  // Si hay m puntos, se requiere que m >= n+1
  // ind valdra   0  si hubo problemas
  //              1  si el proceso funciono bien.
  // yy contendra los valores del polinomio p en los
  // diferentes puntos x(i)

  p = poly([0], 'x', 'c')
  yy = []
  ind = 0

  x = x(:)
  y = y(:)

  [m, nx ] = size(x)
  [my, ny ] = size(y)

  if nx > 1 | ny > 1
    printf('x, y deben ser vectores.\n')
    return
  end

  if m <> my
```

```
    printf('x, y  deben ser del mismo tama~no.\n')
    return
  end


  if n < 0 | m < n+1
    printf(' n = %d inadecuado.\n', n)
    return
  end

  // no debe haber xi repetidos
  eps0 = 1.0e-16*max(abs(x))
  for i=1:m-1
    for j = i+1:m
      if abs(x(i)-x(j)) <= eps0
        printf(' x(i) repetidos.\n')
        return
      end
    end
  end

  ind = 1
  A = zeros(m,n+1)
  for i=1:m
    A(i,1) = 1
    xi = x(i)
    xij = 1
    for j =1:n
      xij = xij*xi
      A(i,j+1) = xij
    end
  end
  if m == n+1
    c = A\y
  else
    c = (A'*A)\(A'*y)
  end
  p = poly(c, 'x', 'c')
  yy = A*c
endfunction
```

### 4.6.4 Factores primos

En el siguiente ejemplo, para un entero $n \geq 2$, se construye un vector fila con los factores primos de $n$. En este vector fila los factores primos aparecen repetidos según la multiplicidad. Por ejemplo, para $n = 72 = 2^3 \times 3^2$, los factores serán 2, 2, 2, 3 y 3. La función `factores` hace uso de la función `primerDiv` que calcula el divisor más pequeño de un entero superior a uno.

```
function [p, indic] = primerDiv(n)
  // calculo del primer divisor (divisor mas peque~no)
  // de un entero n >= 2.
  // indic  valdra  0  si  n  es un dato malo,
  //                1  si  el  proceso se realizo bien.
  // Si  n  es primo, entonces  p = n.

  p = 0
  indic = 0

  if n < 2
    printf('PRIMERDIV:  n < 2.\n'), return
  end

  if int(n) <> n
    printf('PRIMERDIV:  n  no es entero.\n'), return
  end

  ind = 1
  n1 = floor(sqrt(n))
  for p =2:n1
    if modulo(n, p) == 0, return, end
  end
  p = n
endfunction
//---------------------------------------------------------
function [p, indic] = factores(n)
  // Construccion del vector fila  p  con los factores
  // primos del entero  n >= 2.
  // El producto de los elementos de  p  sera  n.
  // indic  valdra  0  si  n  es un dato malo,
  //                1  si  el  proceso se realizo bien.

  p = []
  indic = 0
```

```
  if n < 2
    printf('FACTORES:  n  < 2.\n'), return
  end

  if int(n) <> n
    printf('FACTORES:  n  no es entero.\n'), return
  end

  ind = 1
  while n > 1
    pi = primerDiv(n)
    p = [p pi]
    n = n/pi
  end
endfunction
```

## 4.7 Miscelánea

En esta sección hay varias funciones relacionadas con diversos temas, que se pueden usar para hacer programas en Scilab, pero que también pueden ser usadas directamente desde el ambiente Scilab.

Algunas veces se desea saber, dentro de un guión o dentro de una función, si una variable ya ha sido creada anteriormente (en la misma sesión de Scilab) bien sea directamente en el ambiente Scilab o bien sea en otro guión. Se puede utilizar la función `exists`, que devuelve 1 ó 0, dependiendo de que la variable exista o no exista. El nombre de la variable va entre comillas simples o dobles.

```
function h = altura(v0, t)
  // calculo de la altura de un disparo hacia arriba
  // con velocidad inicial v0 en el tiempo  t.
  if exists('gravedad') == 0
    gravedad = 9.8
  end
  h = v0*t - 0.5*gravedad*t*t
endfunction
```

La función `isdef` hace el mismo trabajo, pero devuelve `%T` o `%F` (*true*, *false*).

```
function h = altura(v0, t)
  // calculo de la altura de un disparo hacia arriba
```

```
  // con velocidad inicial v0 en el tiempo  t.
  if ~isdef('gravedad')
    gravedad = 9.8
  end
  h = v0*t - 0.5*gravedad*t*t
endfunction
```

Cuado se involucran vectores de manera adecuada, las comparaciones dan vectores de variables boolenas (valor `%T` o `%F`, correspondientes a 1 ó 0). Por ejemplo,

```
x = [ -2  -1.0e-10  -1.0e-20  4]
y = [  3  3         3         3]
res = ( x <= y )
```

da como resultado el vector

```
! T T T F !
```

Las órdenes

```
x = [ -2 -1.0e-10  -1.0e-20  4 ]
res = ( abs(x) >= 1.0e-16 )
```

dan como resultado el vector

```
! T T F T !
```

La orden `find` permite obtener las posiciones de un vector booleano donde el valor es `%T`. Por ejemplo, las órdenes

```
x = [ -2 -1.0e-10  -1.0e-20 4 ]
p  = find( abs(x) >= 1.0e-16 )
```

dan como resultado el vector

```
!   1.    2.    4. !
```

o sea, un vector con las posiciones de `x` donde hay valores no despreciables (ni nulos ni casi nulos).

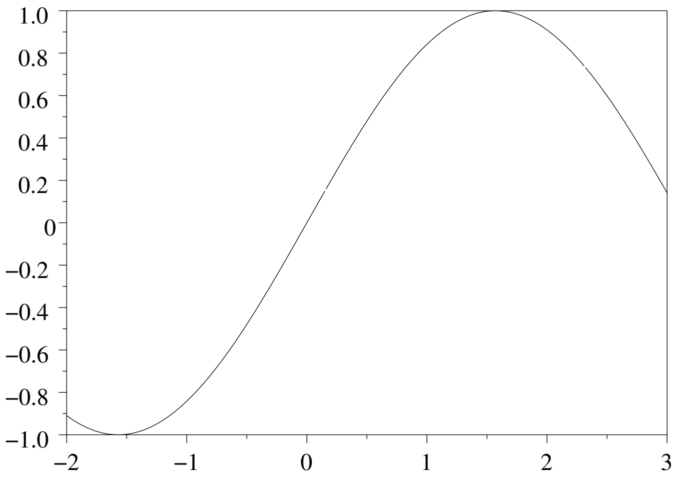
# Capítulo 5

# GRÁFICAS

## 5.1 Dos dimensiones

Para hacer la gráfica de la función $f : [a, b] \to \mathbb{R}$, basta con construir un vector con valores de $x$ en el intervalo $[a, b]$ y otro vector con los valores de $f$ en los puntos del primer vector. Por ejemplo,

```
a = -2;  b = 3;
x = a:0.01:b;
y = sin(x);
plot2d(x, y)
```




hace la gráfica de seno$(x)$ en el intervalo $[-2, 3]$.

Obviamente los vectores `x, y` tienen el mismo tamaño. Los valores de $x$ se tomaron con un espaciamiento de 0.01. Esto hace que la gráfica se vea, no sólo continua sino también suave. Si el espaciamiento es muy grande, por ejemplo,

```
a = -2; b = 3;
x = a:0.5:b;
y = sin(x);
plot2d(x, y)
```

dará una gráfica continua pero poligonal y no suave. De hecho siempre Scilab hace líneas poligonales, pero si el espaciamiento es muy pequeño la línea poligonal se confunde con una curva suave. En resumen, el espaciamiento debe ser pequeño, pero es inoficioso hacerlo muy pequeño.

Las órdenes siguientes permiten obtener la gráfica de una parábola:

```
a = -2; b = 3;
x = a:0.05:b;
y = 0.5*x.*x - 2;
plot2d(x, y)
```

En una sesión de Scilab, la primera vez que se hace una gráfica, esta aparece inmediatamente en la pantalla. Cuando se da la orden para una segunda gráfica, ésta es creada pero no aparece automáticamente en la pantalla. Es necesario, mediante un clic, activar la ventana de la gráfica.

Muy posiblemente después de la segunda orden `plot2d`, en la gráfica aparecerán las dos "curvas" superpuestas. Para limpiar la ventana gráfica se utiliza `xbasc()`. Observe la diferencia entre los dos grupos de órdenes siguientes:

```
plot2d(x,sin(x))
plot2d(x,cos(x))
```

```
plot2d(x,sin(x))
xbasc(), plot2d(x,cos(x))
```

Si se da la orden `plot(x, y, 't', 'sen(t)', 'Ejemplo 1');` en la gráfica aparecerán además 3 etiquetas (o rótulos), la primera en el eje $x$, la segunda en el eje $y$ y la tercera será la etiqueta general de la gráfica.

En la misma figura pueden aparecer varias funciones. Para esto, los datos de `plot2d` deben estar en columnas o en matrices. Si `x, y, z` son vectores columna con el mismo número de filas, entonces se puede dar la orden

```
plot2d(x, [y z] )
```

cuyo efecto es tener en la misma figura las gráficas producidas por las órdenes `plot2d(x,y)` y `plot2d(x,z)`. Por ejemplo,

```
x = (-2:0.05:3)'; y = sin(x); z = cos(x); plot2d(x, [y z sin(2*x)])
```

En este caso Scilab asigna colores diferentes a cada una de las curvas. Para conocer la correspondencia entre colores y curvas se pueden colocar letreros ("legend") para las curvas. Se utiliza la opción `leg` separando los letreros por medio de `@`.

```
plot2d(x, [y  z  sin(2*x)], leg="sen(x)@cos(x)@sen(2x)" )
```

Después de haber hecho una gráfica, se le pueden agregar letreros. Se usa `xtitle`, que tiene 3 parámetros, todos los tres deben ser cadenas. El primero para el letrero general, el segundo para el eje horizontal y el tercero para el eje vertical. Por ejemplo

```
xtitle("Funciones trigonometricas", "x", "")
```

En resumen, una de las formas de utilizar `plot2d` es `plot2d(x, a)`, donde `x` es un vector columna y `a` es una matriz con el mismo número de filas que `x`. En la figura habrá tantas gráficas como columnas tenga la matriz `a`.

Una forma más general es `plot2d(u, v)`, donde `u` y `v` son matrices del mismo tamaño, de $m$ filas y $n$ columnas. Entonces en la misma figura aparecerán la gráfica de la primera columna de `u` con la primera columna de `v`, la gráfica de la segunda columna de `u` con la segunda columna de `v`, etc. En la figura producida por el siguiente ejemplo está la gráfica de seno entre 0 y 3.14 y la gráfica de la tangente entre $-1$ y 1.

```
n = 100;
a = 0; b = 3.14;
h = (b-a)/n;
x1 = (a:h:b)';
y = sin(x1);

a = -1; b = 1;
h = (b-a)/n;
x2 = (a:h:b)';
z = tan(x2);
plot2d([x1 x2], [y z]);
```

También se puede, usando `fplot2d`, obtener la gráfica de una función $f$ definida mediante una función de Scilab. En este caso, sólo se necesita el vector de valores de $x_i$. Este vector debe ser monótono, es decir, creciente o decreciente. Supongamos que se ha definido la siguiente función

```
function fx = func4(x)
    fx = sin(x)-tan(x)
```

```
endfunction
```

y que se ha cargado el archivo donde está definida, mediante `getf ...` Entonces se puede obtener la gráfica mediante

```
u = (-1:0.01:1)';
fplot2d(u, func4)
```

Si la función es sencilla, no es necesario crear la función en un archivo. Se puede obtener la gráfica mediante:

```
t = (-1:0.01:1.2)';
deff("[y] = func4b(x)", "y = x.*x + 2.5")
fplot2d(t, func4b)
```

Algunas veces se desea, no obtener la curva, sino dibujar los **puntos**, sin que haya "continuidad", o sea, que los puntos no estén unidos por un segmento de recta. Una forma de hacerlo es mediante un argumento adicional, el estilo. Las siguientes órdenes muestran un ejemplo.

```
xbasc();
x = (-3.2:0.2:3.2)';
y = sin(x);
tipo = -1;
plot2d( x, y, tipo)
```

Si el estilo, en este ejemplo la variable `tipo`, es menor o igual a cero, en la gráfica aparecen los puntos representados por un símbolo. Los valores pueden ser 0, $-1$, $-2$, ..., $-9$

El significado de los cinco primeros es el siguiente:

0 : $\cdot$

-1 : $+$

-2 : $\times$

-3 : $\odot$

-4 : $\blacklozenge$

Obviamente en la gráfica aparecen reducciones de estos símbolos. Es posible dibujar, al mismo tiempo, dos listas de puntos (con igual cantidad de puntos), cada lista con un símbolo diferente. Por ejemplo:

```
xbasc();
x = (-3.2:0.2:3.2)';
plot2d( [x x], [sin(x) cos(x)], [-1 -2] )
```

## 5.2 Tres dimensiones

Sea $f : \mathbb{R}^2 \to \mathbb{R}$, por ejemplo, $f(x, y) = 5x^2 - y^2$. Para hacer la gráfica de la función restringida al rectángulo $[a, b] \times [c, d]$, basta con construir un vector con valores de la primera variable en el intervalo $[a, b]$, otro con valores de la segunda variable en el intervalo $[c, d]$ y una matriz con los valores $f(..., ...)$. Por ejemplo,

```
u = (-2:0.05:2)';
v = (-3:0.1:3)';
m = size(u,1);  n = size(v,1);
w = zeros(m,n);
for i=1:m
    for j = 1:n
        w(i,j) = 5*u(i)^2 - v(j)^2;
    end
end
plot3d(u, v, w);
```

De manera análoga a dos dimensiones, es posible graficar por medio de `fplot3d` una función $f$ definida por medio de una función de Scilab. Basta con dar dos vectores, uno con las coordenadas de la primera variable y otro con las coordenadas de la segunda variable.

Supongamos que se ha definido la siguiente función

```
function fst = func5(s, t)
    fst = 5*s^2 - t^2
endfunction
```

y que se ha cargado el archivo donde está definida, mediante `getf ...` Entonces se puede obtener la gráfica mediante

```
u = (-4:0.05:4)';
v = (-5:0.05:3)';
fplot3d(u, v, func5)
```

Esta función `fplot3d` construye la matriz de valores $f$`(u[i],v[j])`, de tamaño, para este ejemplo, $161 \times 221$. Esto se demora algunos segundos. Además si `u` y `v` son muy grandes, la matriz podría ser demasiado grande. En este caso Scilab muestra un aviso semejante a:

```
stack size exceeded (Use stacksize function to increase it)
```

indicando que el tamaño de la pila no es suficiente para lo solicitado. Este tamaño se puede modificar mediante la función `stacksize`, por ejemplo,

```
stacksize(2000000)
```

Se pueden obtener curvas de nivel mediante la función `contour`. Su uso es muy semejante al de `plot3d` pero con un parámetro adicional que indica el número de curvas de nivel. Por ejemplo:

```
u = (-2:0.05:2)';
v = (-3:0.1:3)';
m = size(u,1);  n = size(v,1);
w = zeros(m,n);
for i=1:m
    for j = 1:n
        w(i,j) = 5*u(i)^2 + v(j)^2;
    end
end
contour(u, v, w, 5);
```

También, mediante `fcontour`, se pueden obtener curvas de nivel de una función definida en una función de Scilab. Su uso es análogo al de `fplot3d` pero con un parámetro adicional que indica el número de curvas de nivel. Por ejemplo:

```
u = (-4:0.2:4)';
v = (-4:0.2:5)';
fcontour(u, v, func5, 7)
```

## 5.3 Otras funciones

La siguiente lista contiene los nombres de la mayoría de la funciones para gráficas.

```
addcolor alufunctions black bode champ champ1 chart colormap
contour contour2d contour2di contourf dragrect drawaxis driver
edit_curv errbar eval3d eval3dp evans fac3d fchamp fcontour
fcontour2d fec fgrayplot fplot2d fplot3d fplot3d1 gainplot
genfac3d geom3d getcolor getfont getlinestyle getmark
getsymbol gr_menu graduate graycolormap grayplot graypolarplot
hist3d histplot hotcolormap isoview legends locate m_circle
Matplot Matplot1 milk_drop nf3d nyquist param3d param3d1
paramfplot2d plot plot2d plot2d1 plot2d2 plot2d3 plot2d4 plot3d
plot3d1 plot3d2 plot3d3 plotframe plzr polarplot printing
```

```
replot rotate scaling sd2sci secto3d Sfgrayplot Sgrayplot sgrid
square subplot titlepage winsid xarc xarcs xarrows xaxis xbasc
xbasimp xbasr xchange xclea xclear xclick xclip xdel xend xfarc
xfarcs xfpoly xfpolys xfrect xget xgetech xgetmouse xgraduate
xgrid xinfo xinit xlfont xload xname xnumb xpause xpoly xpolys
xrect xrects xrpoly xs2fig xsave xsegs xselect xset xsetech
xsetm xstring xstringb xstringl xtape xtitle zgrid
```

## 5.4 Creación de un archivo Postscript

Una vez hecha la gráfica, la creación de archivos Postscript o de otros tipos de formato puede ser realizada por medio de la barra de menú de la ventana de la gráfica, con las opciones

`File Export Postscript` *nombre-del-archivo*

El nombre del archivo debe ser dado sin extensión. Observe que para exportar hay otras posibilidades:

`Postscript No Preamble`:

`Postscript-Latex`

`Xfig`

`GIF`

`PPM`

Otra forma de crear archivos postscript es mediante las órdenes: `driver xinit xend`. Utilice la ayuda de Scilab par obtener información sobre su uso.

La orden `xbasimp` permite enviar una gráfica a una impresora postscript o a un archivo.

# Capítulo 6

# MÉTODOS NUMÉRICOS

## 6.1 Sistemas de ecuaciones lineales

Como se vió anteriormente, si se tiene un matriz `A` de tamaño $n \times n$ y un vector `b` de tamaño $n \times 1$, la manera eficiente de reolver el sistema $Ax = b$, es mediante

```
x = A\b
```

Si la matriz es singular o casi singular, Scilab muestra una advertencia.

Si la matriz es definida positiva, resolver el sistema por medio de `x = A\b` funciona bien, pero es más eficiente resolverlo usando la factorización de Cholesky:

```
y = zeros(n,1);
x = zeros(n,1);
u = chol(a);
y(1) = b(1)/u(1,1);
for i=2:n
  y(i) = ( b(i) - u(1:i-1,i)'*y(1:i-1) )/u(i,i);
end

x(n) = y(n)/u(n,n);
for i=n-1:-1:1
  x(i) = ( y(i) - u(i,i+1:n)*x(i+1:n) )/u(i,i);
end
```

La diferencia en el tiempo se nota para valores grandes de $n$. Por ejemplo, si $n = 1500$, la solución `x = A\b` gasta 12.33 "segundos" (en realidadson unidades de tiempo, no exactamente segundos, en un computador específico). En cambio, obtener la solución utilizando la factorización de Cholesky dura 2.78 segundos. Para medir el tiempo se puede puede utilizar la función `timer()`.

Para economizar memoria se hubiera podido efectuar el cálculo de `y` y de `x` sobreescribiendo encima de `b`.

Una versión, con menos órdenes, pero ineficiente, podría ser:

```
u = chol(a);
y = (u'\b);
x = u\y;
```

Por ejemplo, con $n = 1500$, estas tres órdenes duran 15.12 segundos.

Una matriz dispersa (*sparse*) es aquella que tiene muchos ceros y muy pocos elementos no nulos. Si

$$A = \begin{bmatrix} -8 & 0 & 0 & 0 & 1.12 \\ -2 & 5 & 0.5 & 0 & 0 \\ 0 & 0 & 10 & 3.14 & 0 \\ 0 & 0 & 0 & 4 & -1 \\ 0 & 0 & 0 & 0 & 20 \end{bmatrix}$$

basta con almacenar, para cada entrada no nula, el índice de la fila el índice de la columna y el valor. Entonces en Scilab se requiere una matriz de dos columnas con los índices (números enteros entre 1 y $n$) y un vector con los valores:

```
ij = [1 1; 1 5; 2 1 ; 2 2; 2 3;3 3; 3 4; 4 4; 4 5; 5 5]
v = [-8 1.12 -2 5 0.5 10 3.14 4 -1 20]'
A = sparse(ij, v)

b = [-6.88 3.5 13.14 3 20]'
x = A\b
```

Construida de esa manera, se puede resolver el sistema $Ax = b$ utilizando `A\b`. Esta forma de almacenamiento es útil cuando $A$ es muy grande y muy dispersa.

## 6.2 Solución por mínimos cuadrados

Cuando se tiene el sistema

$$Ax = b,$$

donde $A$ es un matriz $m \times n$, $m \geq n$, es muy probable que el sistema no tenga solución, dado que hay más ecuaciones que incógnitas. En este caso se busca obtener un $x$ que minimice el cuadrado de la norma euclidiana del error, o sea

$$\text{minimizar} \quad ||Ax - b||_2^2$$

Por ejemplo,

```
A = [1 2; 3 4; 5 6]
b = [3 7 11.1]'
```

En este caso, la solución por mínimos cuadrados se puede obtener mediante

```
x = A\b
```

o mediante

```
x = pinv(A)*b
```

La primera forma es más eficiente.

## 6.3 Una ecuación no lineal

Para resolver

$$f(x) = 0,$$

donde $f$ es una función de variable y valor real, se utiliza `fsolve`. Por ejemplo para resolver

$$\frac{x - e^x}{1 + x^2} - \cos(x) + 0.1 = 0,$$

es necesario definir una función de Scilab donde esté $f$ y después utilizar `fsolve`.

```
function fx = func156(x)
  fx = ( x - exp(x) )/( 1 + x*x ) - cos(x) + 0.1
endfunction
```

Despues de haber cargado esta función, se utiliza `fsolve` dándole como parámetros, la aproximación inicial y la función:

```
r = fsolve(0, func156)
```

Con otra aproximación inicial podría dar otra raíz. Un parámetro opcional, que puede acelerar la obtención de la solución, es otra función de Scilab donde esté definida la derivada.

## 6.4 Sistema de ecuaciones no lineales

Dada una función $f : \mathbb{R}^n \to \mathbb{R}^n$ se desea encontrar un vector $x$ tal que $f(x) = 0$. Aquí 0 es el vector nulo. Por ejemplo,

$$
\begin{aligned}
x_1^2 + x_1 x_2 + x_3 - 3 &= 0\,, \\
2x_1 + 3x_2 x_3 - 5 &= 0\,, \\
(x_1 + x_2 + x_3)^2 - 10x_3 + 1 &= 0\,.
\end{aligned}
$$

La función $f$ debe estar definida en una función de Scilab, donde, dado un vector columna `x`, calcule el vector columna $f(\mathtt{x})$. Una vez escrita la función y cargado el archivo donde está $f$, se utiliza `fsolve`. Para el ejemplo anterior la función puede ser

```
function fx = f3(x)
    n = size(x,1)
    fx = zeros(n,1)
    fx(1) = x(1)^2 + x(1)*x(2) + x(3) - 3
    fx(2) = 2*x(1) + 3*x(2)*x(3) - 5
    fx(3) = (x(1) + x(2) + x(3))^2 - 10*x(3) + 1
endfunction
```

El llamado de `fsolve` es de la forma

```
[x, fx, info] = fsolve([3 4 5]', f3)
```

El primer parámetro de entrada es la aproximación inicial de la solución. El segundo es el nombre de la función donde está definida $f$. Hay tres parámetros de salida. Únicamente el primero es obligatorio. Es un vector con la solución, si ésta fue obtenida. El vector `fx` es el vector $f(\mathtt{x})$. El parámetro `info` puede tener 5 valores:

`0` : número inadecuado de parámetros de entrada.

`1` : se obtuvo la solución con la precisión deseada.

`2` : muchos llamados a la función.

`3` : la tolerancia permitida es muy pequeña.

`4` : el método avanza muy lentamente.

Si se dispone de una función donde está definida la matriz jacobiana, entonces `fsolve` trabajará mucho mejor. Por ejemplo,

```
function J = jacf3(x)
    t = 2*(x(1)+x(2)+x(3))
    J =  [ 2*x(1)+x(2)  x(1)    1
```

```
          2          3*x(3)  3*x(2)
          t          t       t-10  ]
endfunction
```

Entonces el llamado de `fsolve` es de la forma

```
[x, fx, info] = fsolve([3 4 5]', f3, jacf3)
```

El problema de encontrar la solución, por mínimos cuadrados, de un sistema de ecuaciones no lineales está en el capítulo siguiente.

## 6.5 Integración numérica

Para obtener una aproximación del valor de una integral definida, por ejemplo,

$$\int_{0.1}^{0.5} e^{-x^2} dx$$

se utiliza `intg`. Para eso es necesario definir en Scilab la función que se va a integrar. Puede ser, directamente en el ambiente Scilab:

```
deff('[y] = f53(x)', 'y = exp(-x*x)')
```

```
I = intg(0.1, 0.5, f53)
```

También se puede definir una función en un archivo `.sci`

```
function fx = f57(x)
  fx = exp(-x*x)
endfunction
```

y después de cargarla, dar la orden

```
I = intg(0.1, 0.5, f57)
```

Algunas veces no se conoce una expresión de la función $f$, pero se conoce una tabla de valores $(x_i, f(x_i))$, o simplemente una tabla de valores $(x_i, y_i)$. Supongamos, así lo requiere Scilab, que la lista de valores $(x_1, y_1)$, ..., $(x_n, y_n)$ está ordenada de manera creciente de acuerdo a los $x_i$, o sea, $x_1 < x_2 < \cdots < x_n$.

Para obtener el valor aproximado de la integral, entre $x_1$ y $x_n$, de la función $f$ (representada por los valores $(x_i, y_i)$), es necesario tener dos vectores con los valor $x_i$ y $y_i$, y utilizar la función `inttrap`.

```
x = [0.1 0.15 0.2 0.25 0.3 0.4 0.5]'
y = [ 0.9900 0.9778 0.9608 0.9394 0.9139 0.8521 0.7788]'
I = inttrap(x, y)
```

## 6.6 Derivación numérica

Para obtener una aproximación numérica de la derivada de una función en un punto se puede usar `derivative`.

Si `f57` es la función definida anteriormente, una vez cargada, la orden

```
d = derivative(f57, 0.3)
```

da una aproximación numérica de la derivada de la función en $x = 0.3$.

Si se tiene una función $f : \mathbb{R}^n \to \mathbb{R}$, por ejemplo,

```
function fx = func90(x)
  // funcion de Rosenbrock
  t1 = x(2)-x(1)*x(1)
  t2 = 1-x(1)
  fx = 10*t1*t1 + t2*t2 + 4
endfunction
```

entonces la orden

```
derivative(func90, [3 4]')
```

da como resultado un vector fila con el gradiente de $f$ en $x = (3, 4)$. El segundo parámetro, el punto $x$, debe ser un vector columna.

Si se tiene una función $f : \mathbb{R}^n \to \mathbb{R}^m$, entonces $f(x) = (\, f_1(x), f_2(x), ..., f_m(x)\,)$, donde $f_i : \mathbb{R}^n \to \mathbb{R}$ . Por ejemplo,

```
function Fx = func157(x)
  Fx = zeros(4,1);
  Fx(1) = x(1)^2 + x(1)*x(2) + x(3) - 3;
  Fx(2) = 2*x(1) + 3*x(2)*x(3) - 5;
  Fx(3) = ( x(1) + x(2) + x(3) )^2 - 10*x(3) + 1;
  Fx(4) = 10*( x(1) + x(2) + x(3) - 3.3 );
endfunction
```

entonces la orden

```
derivative(func157, [3 4 5]')
```

da como resultado la matriz jacobiana de la función $f$ (definida en `f157`) evaluada en el punto $x = (3, 4, 5)$. El segundo parámetro, el punto $x$, debe ser un vector columna. La matriz jacobiana es de tamaño $m \times n$ (en este ejemplo: $4 \times 3$). En la fila $i$ está el gradiente de $f_i$.

## 6.7 Interpolación

Cuando hay $m$ puntos $(x_1, y_1)$, $(x_2, y_2)$, ..., $(x_m, y_m)$ se desea obtener la función interpolante, una función que pase por esos puntos, con el objetivo de evaluarla en otros valores $x$ intermedios.

La función `interpln` permite hacer interpolación lineal. Tiene dos parámetros, el primero es una matriz de dos filas. La primera fila tiene los valores $x_i$. Deben estar en orden creciente. La segunda fila tiene los valores $y_i$. El segundo parámetro es un vector donde están los valores $x$ en los que se desea evaluar la función interpolante (afín por trozos).

```
x = [0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2]';
y = [4.8 6.2 6.8 7.2 7.8 9.2 8.8 9.2 8.8 9.2 7.8]';
nx = size(x,1);
t = (x(1):0.01:x(nx))';

ft = interpln( [x'; y'], t);
xbasc()
plot2d(t, ft)
```

También se puede hacer interpolación utilizando funciones *spline*. Para hacer esto en Scilab, se requieren dos pasos. En el primero, a partir de un lista de puntos $(x_1, y_1)$, $(x_2, y_2)$, ..., $(x_m, y_m)$ se calculan las derivadas, en los puntos $x_i$, de la función *spline* interpolante. En el segundo paso se evalúa la función interpolante en una lista de puntos $t_1$, $t_2$, ..., $t_p$

```
x = [0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2]';
y = [4.8 6.2 6.8 7.2 7.8 9.2 8.8 9.2 8.8 9.2 7.8]';
nx = size(x,1);
t = (x(1):0.01:x(nx))';

d = splin(x, y);
ft = interp(t, x, y, d);
xbasc()
plot2d(t, ft)
```

## 6.8 Aproximación por mínimos cuadrados

Cuando hay $m$ puntos $(x_1, y_1)$, $(x_2, y_2)$, ..., $(x_m, y_m)$ se desea obtener un polinomio de grado $n$, $n + 1 \leq m$, que aproxime por mínimos cuadrados. Es necesario construir una matriz de tamaño $m \times (n+1)$, resolver por mínimos cuadrados un sistema de ecuaciones. Generalmente, después se evalúa este polinomio en algunos valores (pueden ser los mismos $x_i$ u otros).

```
x = [0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2]';
y = [4.8 6.2 6.8 7.2 7.8 9.2 8.8 9.2 8.8 9.2 7.8]';
m = size(x,1);
t = (x(1):0.01:x(m))';
n = 2;

A = zeros(m, n+1);
for i=0:n
  A(:,i+1) = x.^i;
end
cf = A\y;

p = poly(cf, 'x', 'c');
ft = horner(p, t);
xbasc()
plot2d(t, ft)
```

Para la aproximación no es indispensable que los $x_i$ estén ordenados de manera creciente, pero en este ejemplo es necesario para la gráfica.

También se puede hacer, de manera análoga, aproximación por mínimos cuadrados utilizando una combinación lineal de otras funciones (trigonométricas, exponenciales, logarítmicas...). Cambia simplemente la manera de construir la matriz $A$ y la manera de evaluar la función en los valores del vector `t`.

Scilab tiene la función `lsq_splin` que permite aproximación por mínimos cuadrados con funciones *spline*. Los datos son: los valores $x_i$, ordenados de manera creciente, los valores $y_i$, y los puntos de ruptura (*breakpoint*) ordenados de manera creciente. Con ellos se determina la función *spline*. Después se evalúa en los valores deseados.

```
x = [0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2]';
y = [4.8 6.2 6.8 7.2 7.8 9.2 8.8 9.2 8.8 9.2 7.8]';
nx = size(x,1);
t = (x(1):0.01:x(nx))';
X = [0 1 2]'
[Y, d] = lsq_splin(x, y, X);
ft = interp(t, X, Y, d);
xbasc()
plot2d(t, ft)
```

## 6.9 Ecuaciones diferenciales ordinarias

Una ecuación diferencial de primer orden con condición inicial se escribe, generalmente, en la forma

$$\begin{aligned} y' &= f(x, y), \\ y(x_0) &= y_0. \end{aligned}$$

Por ejemplo,

$$\begin{aligned} y' &= \frac{x+y}{x^2+y^2} + 4\,, \\ y(2) &= 3. \end{aligned}$$

Antes de utilizar la función `ode`, es necesario crear en Scilab la función $f$ y cargarla. La función `ode` evalúa aproximaciones del valor de $y$ en valores $t_1$, $t_2$, ..., $t_p$.

Despues de definir y cargar

```
function fxy = func158(x, y)
  fxy = ( x + y )/( x*x + y*y ) + 4
endfunction
```

se obtiene la solución aproximada mediante

```
x0 = 2
y0 = 3
t = [2 2.1 2.2 2.5 2.9]'
yt = ode(y0, x0, t, func158)
```

## 6.10 Sistema de ecuaciones diferenciales

Un sistema de ecuaciones diferenciales ordinarias de primer orden con condiciones iniciales se escribe, generalmente, en la forma

$$\begin{aligned} y' &= f(x, y), \\ y(x_0) &= y_0. \end{aligned}$$

pero ahora, $y$, $y'$ y $y_0$ son vectores en $\mathbb{R}^n$; $f : \mathbb{R}^{1+n} \to \mathbb{R}^n$. Por ejemplo,

$$
\begin{aligned}
y_1' &= \frac{2y_1}{x} + x^3 y_2, \\
y_2' &= -\frac{3}{x} y_2 \\
y_1(1) &= -1 \\
y_2(1) &= \ \ 1.
\end{aligned}
$$

Despues de definir y cargar

```
function fxy = func43(x, y)
  fxy = zeros(2,1)
  fxy(1) = 2*y(1)/x + x^3*y(2)
  fxy(2) = -3*y(2)/x
endfunction
```

se utiliza la misma función `ode`, pero con los parámetros de dimensión adecuada.

```
x0 = 1
y0 = [-1 1]'
t = (1:0.2:2)'
yt = ode(y0, x0, t, func43)
```

En este caso, `yt` es un matriz de dos filas. En la fila $i$ están las aproximaciones de los valores de $y_i(t_j)$.

# Capítulo 7

# OPTIMIZACIÓN

## 7.1 Optimización lineal

Hay varias maneras o formas de presentación de problemas de OL, optimización lineal (o programación lineal). Entre las de uso más frecuente están la forma canónica y la forma estándar. La forma que más rápidamente se adapta a la solución en Scilab, mediante la función `linpro`, es la siguiente:

$$
\begin{aligned}
\min\ & z = c^{\mathrm{T}} x \\
& A_{i\cdot} x = b_i \ , \quad i = 1, ..., m_0 \\
& A_{i\cdot} x \leq b_i \ , \quad i = m_0 + 1, ..., m \\
& u \leq x \leq v.
\end{aligned}
\tag{7.1}
$$

Obviamente, siempre se puede pasar de una forma a otra sin dificultad. Para el caso de Scilab, todos los problemas deben ser de minimización y las desigualdades, si las hay, diferentes de las restricciones de caja ($u \leq x \leq v$), deben ser de la forma

$$a_{i1}x_1 + a_{i1}x_2 + \cdots + a_{i1}x_n \leq b_i.$$

La matriz $A$ es de tamaño $m \times n$; los vectores $c, u, v \in \mathbb{R}^{n\times 1}$, $b \in \mathbb{R}^{m\times 1}$ (todos son vectores columna). Las igualdades, si las hay, deben estar antes de las desigualdades.

Frecuentemente, en los problemas de OL, todas las variables deben ser no negativas. En Scilab esto se debe considerar como restricciones de caja o como restricciones usuales.

### 7.1.1 Desigualdades

El problema más sencillo para resolver por medio de `linpro` es el siguiente:

$$\min\ c^{\mathrm{T}} x$$

$$Ax \leq b.$$

La orden en Scilab es:

```
[x, lagr, z] = linpro(c, A, b)
```

El significado de los tres resultados es el siguiente : `x` es el vector solución; `lagr` es un vector con los coeficientes de Lagrange; `z` es el valor óptimo de la función objetivo, es decir, $c^{\mathrm{T}}x$.

Para resolver el siguiente problema

$$\begin{aligned} \min\ -x_1 - 1.4x_2 & \\ x_1 + x_2 &\leq 400 \\ x_1 + 2x_2 &\leq 580 \\ x_1 \quad\quad &\leq 300 \\ x &\geq 0, \end{aligned} \tag{7.2}$$

es necesario reescribir las restricciones de no negatividad:

$$\begin{aligned} \min\ -x_1 - 1.4x_2 & \\ x_1 + x_2 &\leq 400 \\ x_1 + 2x_2 &\leq 580 \\ x_1 \quad\quad &\leq 300 \\ -x_1 &\leq 0 \\ -x_2 &\leq 0. \end{aligned}$$

Su solución mediante Scilab:

```
c = [-1 -1.4 ]';
A = [ 1 1; 1 2; 1 0; -1 0 ; 0 -1];
b = [400 580 300 0 0]';
[x, lg, z ] = linpro(c, A, b)
```

### 7.1.2 Desigualdades y restricciones de caja

Se usa la orden

```
[x, lg, z ] = linpro(c, A, b, u, v)
```

para resolver

$$\min\ c^{\mathrm{T}}x$$

$$\begin{aligned} Ax &\leq b \\ u \leq x &\leq v. \end{aligned}$$

El problema (7.2) se puede resolver con las siguientes órdenes:

```
c = [-1 -1.4 ]';
A = [ 1 1; 1 2; 1 0];
b = [400 580 300]';
[x, lg, z ] = linpro(c, A, b, zeros(2,1), [])
```

Es interesante observar que como no hay restricciones superiores de caja, entonces se utiliza un vector "vacío". En el problema (7.2) la restricción $x_1 \leq 300$ se puede considerar como una restricción de caja.

$$\begin{aligned} \min\ -x_1 - 1.4x_2 & \\ x_1 + x_2 &\leq 400 \\ x_1 + 2x_2 &\leq 580 \\ \\ x_1 \qquad &\leq 300 \\ x &\geq 0. \end{aligned}$$

Así planteado, se puede resolver mediante las siguientes órdenes:

```
c = [-1 -1.4 ]';
A = [ 1 1; 1 2];
b = [400 580]';
v = [ 300 1.0e90]';
[x, lg, z ] = linpro(c, A, b, zeros(2,1), v)
```

### 7.1.3 Igualdades, desigualdades y restricciones de caja

Es el caso general (7.1) planteado al comienzo del capítulo. Este problema se resuelve mediante la orden

```
[x, lg, z ] = linpro(c, A, b, u, v, m0)
```

El siguiente problema

$$\begin{aligned} \min\ -x_1 - 1.4x_2 & \\ x_1 + x_2 &= 400 \end{aligned}$$

$$x_1 + 2x_2 \le 580$$

$$\begin{aligned} x_1 \qquad &\le 300 \\ x &\ge 0\,, \end{aligned}$$

se resuelve mediante:

```
c = [-1 -1.4 ]';
A = [ 1 1; 1 2];
b = [400 580]';
v = [ 300 1.0e90]';
[x, lg, z ] = linpro(c, A, b, zeros(2,1), v, 1)
```

## 7.2 Optimización cuadrática

El problema general de optimización cuadrática es muy parecido al caso general de OL. La única diferencia consiste en que la función objetivo tiene una parte cuadrática $\frac{1}{2}x^{\mathrm{T}}Qx$. Las restricciones siguen exactamente el mismo esquema.

$$\begin{aligned} \min\ q(x) = \frac{1}{2}&x^{\mathrm{T}}Hx + c^{\mathrm{T}}x \\ A_{i\cdot}x &= b_i\ , \quad i = 1, ..., m_0 \\ A_{i\cdot}x &\le b_i\ , \quad i = m_0 + 1, ..., m \\ u \le &\ x \le v. \end{aligned}$$

La matriz $H$ debe ser simétrica. Se usa la función `quapro`. Su llamado y uso son muy semejantes a los de la función `linpro`.

```
[x, lg, qx ] = quapro(H, c, A, b, u, v, m0)
```

Los casos de desigualdades y, desigualdades y restricciones de caja, son completamente análogos a los de OL.

Consideremos el siguiente problema:

$$\begin{aligned} \min\ (x_1 - 3)^2 + (x_2 - 4)^2 & \\ x_1 + 2x_2 &\le 3 \\ 3x_1 + x_2 &\le 4 \\ x &\ge 0. \end{aligned}$$

Entonces:

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} -6 \\ -8 \end{bmatrix}.$$

Su solución se puede obtener mediante las órdenes

```
H = [ 2 0; 0 2];
c = [-6; -8];
A = [ 1 2; 3 1];
b = [ 3 ; 4];
[x, lagr, qx ] = quapro(H, c, A, b, zeros(2,1), [])
```

## 7.3 Optimización no lineal

### 7.3.1 Optimización no restringida

El problema que se trata de resolver es simplemente minimizar una función $f : \mathbb{R}^n \to \mathbb{R}$,

$$\min f(x)$$

Para su solución en Scilab se usa la función `optim`. Se necesita haber definido previamente una función en Scilab donde se calcule $f(x)$ y el gradiente $f'(x)$. Esta función que por el momento llamaremos `f_obj` debe tener la siguiente forma

```
[fx, gr, ind2] = f_obj(x, ind1)
```

Parámetros de entrada:

- `x`: punto donde se va a evaluar $f(x)$;
- si `ind1` $= 2$, se evalúa $f(x)$;
- si `ind1` $= 3$, se evalúa $f'(x)$;
- si `ind1` $= 4$, se evalúan $f(x)$ y $f'(x)$.

Parámetros de salida:

- `fx`: valor $f(x)$;
- `gr`: gradiente $f'(x)$;
- si `ind2` $< 0$, no se puede evaluar $f(x)$;
- si `ind2` $= 0$, se interrumpe la optimización.

Consideremos dos problemas, el primero está dado por la llamada función de Rosenbrock. Las soluciones de estos problemas son $x = (1, 1)$ y $x = (3, 4)$.

$$\begin{aligned} \min \quad & f(x_1, x_2) = 10(x_2 - x_1^2)^2 + (1 - x_1)^2, \\ \min \quad & f(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 4)^2. \end{aligned}$$

Antes de resolver estos dos problemas de minimización mediante `optim` es necesario construir un archivo, por ejemplo de nombre `func.sci`

```
function [fx, gr, ind2] = rosenbr(x, ind)
    // funcion de Rosenbrock
    // f(x) = 10(x2-x1^2) + (1-x1)^2
    fx = 0.0;
    gr = 0.0*x;
    ind2 = 1;
    if ind == 2 | ind == 4
        fx = 10*(x(2)-x(1)^2)^2 + (1-x(1))^2;
    end
    if ind == 3 | ind == 4
        gr(1,1) = -40*x(1)*(x(2)-x(1)^2) - 2*(1-x(1));
        gr(2,1) = 20*(x(2)-x(1)^2);
    end
endfunction
//--------------------------------------------------
function [fx, gr, ind2] = ensayo(x, ind)
    fx = 0.0;
    gr = 0.0*x;
    ind2 = 1;
    if ind == 2 | ind == 4
        fx = (x(1)-3)^2 + (x(2)-4)^2;
    end
    if ind == 3 | ind == 4
        gr(1,1) = 2*x(1)-6;
        gr(2,1) = 2*x(2)-8;
    end
endfunction
```

La función `optim` se puede llamar de la siguiente forma:

```
[fx, x] = optim(f_obj, x0)
```

Para nuestros dos ejemplos específicos, las órdenes pueden ser

```
getf c:\...\func.sci
[fxopt, xopt] = optim(rosenbr, [5; 6])
[fxopt, xopt] = optim(ensayo, [0; 0])
```

La función `optim` admite escoger el método de minimización entre cuasi Newton, gradiente conjugado y no diferenciable. Para ello es necesario utilizar `'qn'` `'gc'` o `'nd'`. La opción por defecto es `'qn'`. Por ejemplo,

```
[fx, x]=optim(rosenbr, x0, 'qn')
[fx, x]=optim(rosenbr, x0, 'gc')
[fx, x]=optim(rosenbr, x0, 'nd')
```

### 7.3.2 Restricciones de caja

La función `optim` también admite restricciones de caja, o sea, sirve para un problema de la forma

$$\begin{aligned} \min \quad & f(x_1, x_2) = 10(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ & 3 \leq x_1 \leq 5 \\ & 2 \leq x_2 \leq 10\,. \end{aligned}$$

La órdenes pueden ser:

```
u = [3; 2];
v = [5; 10];
x0 = [0; 0];
[fx, x] = optim(rosenbr, 'b', u, v, x0)

[fx, x] = optim(rosenbr, 'b', u,  v, x0, 'qn')

[fx, x] = optim(rosenbr, 'b', u,  v, x0, 'gc')
```

El parámetro `'b'` (*bound*) indica que los dos vectores columna siguientes son las cotas de caja. También es posible escoger el criterio de parada, para ello se usa el parámetro `'ar'` . Ver la ayuda de la función `optim`.

## 7.4 Mínimos cuadrados no lineales

Dada una función $f : \mathbb{R}^n \to \mathbb{R}^m$, (usualmente $n \leq m$), se desea minimizar el cuadrado de la norma euclidiana de $f(x)$,

$$\min \quad ||f(x)||^2\,.$$

Por ejemplo,

$f(x_1, x_2, x_3) = (x_1^2 + x_1x_2 + x_3 - 3, 2x_1 + 3x_2x_3 - 5, (x_1 + x_2 + x_3)^2 - 10x_3 + 1, 10(x_1 + x_2 + x_3) - 3.3)$.

La solución se hace mediante `leastsq`. De manera análoga al caso de sistemas de ecuaciones no lineales, es necesario escribir una función de Scilab donde se define $f$ (dado un vector columna `x` calcula el vector columna $f$(`x`) ). Por ejemplo

```
function fx = fmc(x)
    m = 4;
```

```
    fx = zeros(m,1);
    fx(1) = x(1)^2+x(1)*x(2)+x(3)-3;
    fx(2) = 2*x(1)+3*x(2)*x(3)-5;
    fx(3) = (x(1)+x(2)+x(3))^2-10*x(3)+1;
    fx(4) = 10*(x(1)+x(2)+x(3)-3.3);
endfunction
```

El llamado de `leastsq` es muy parecido al de `fsolve`. Por ejemplo,

```
[fx, x] = leastsq(fmc, [3 5 7]')
```

También el método trabaja mejor si se tiene una función de Scilab donde se calcula la matriz con las derivadas parciales. Por ejemplo,

```
function D = derfmc(x)
    t = 2*(x(1)+x(2)+x(3));
    D =  [ 2*x(1)+x(2)  x(1)    1
           2            3*x(3)  3*x(2)
           t            t       t-10
           10           10      10    ];
endfunction
```

La órden pueden ser:

```
[fx, x] = leastsq(fmc, derfmc, [4 5 6]')
```

### 7.4.1 Mínimos cuadrados con restricciones de caja

En Scilab se puede resolver el problema de mínimos cuadrados no lineales con restricciones de caja:

$$\begin{aligned} &\min\ ||f(x)||^2 \\ &\quad u \le x \le v. \end{aligned}$$

Las órdenes pueden ser:

```
u = [1.2 0 1.1]';
v = [10 10 10]';
[fx, x] = leastsq(fmc, 'b', u, v, [4 5 6]')
```

o para mayor eficiencia,

```
[fx, x] = leastsq(fmc, derfmc, 'b', u, v, [4 5 6]')
```

Otras herramientas de optimización son:

• `semidef` : optimización semidefinida,

• `lmisolver, lmitool`: linear matrix inequalities,

• `karmarkar` : método de Karmarkar. para OL.

# ÍNDICE ANALÍTICO