Taller de SciLab

INTRODUCCIÓN A SCILAB

UNJU – Facultad de Ingeniería

CÁLCULO NUMÉRICO 2018

Introducción

Scilab es un paquete de software libre de código abierto para computación científica, orientado al cálculo numérico, a las operaciones matriciales y especialmente a las aplicaciones científicas y de ingeniería.

Puede ser utilizado como simple calculadora matricial, pero su interés principal radica en los cientos de funciones tanto de propósito general como especializadas que posee así como en sus posibilidades para la visualización gráfica.

Scilab posee además un lenguaje de programación propio, muy próximo a los habituales en cálculo numérico (Fortran, C, …) que permite al usuario escribir sus propios scripts (conjunto de comandos escritos en un fichero que se pueden ejecutar con una única orden) para resolver un problema concreto y también escribir nuevas funciones, por ejemplo, sus propios algoritmos. Scilab dispone, además, de numerosas Toolboxes, que le añaden funcionalidades especializadas.

Scilab dispone de un manual de usuario que se puede consultar en una ventana de ayuda (Help Browser). Esta ayuda se puede invocar desde la barra de herramientas (? ---> Scilab Help en Windows, Help ---> Help Browser en Linux) o escribiendo mediante el comando help(). Se puede acceder fácilmente a la descripción de todas las funciones que, en muchos casos, se acompaña de ejemplos de uso ilustrativos.

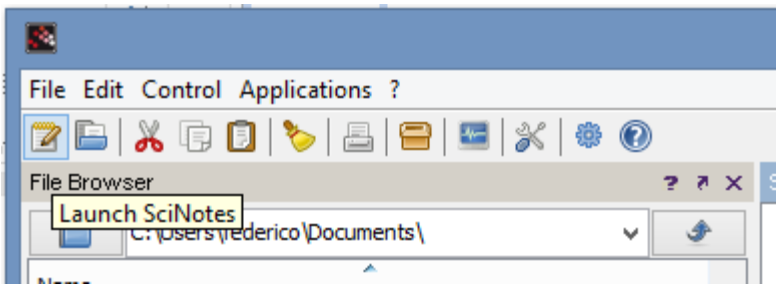
Al ejecutar SCILAB aparece el prompt - - > que indica que se pueden ejecutar los comandos de programación y aparece la siguiente ventana:

Scripts y Funciones

Scripts

Un script es un conjunto de instrucciones (de cualquier lenguaje) guardadas en un fichero (usualmente de texto) que son ejecutadas normalmente mediante un intérprete. Son útiles para automatizar pequeñas tareas. También puede hacer las veces de un "programa principal" para ejecutar una aplicación.

Así, para llevar a cabo una tarea, en vez de escribir las instrucciones una por una en la línea de comandos de Scilab, se pueden escribir una detrás de otra en un fichero. Para ello se puede utilizar el Editor integrado de Scilab: botón "Editor" de la barra de menús



Por convenio, los scripts de Scilab tienen el sufijo (extensión) **.sce**

Para ejecutar un script se usa la orden

→ exec(nombre_fichero) // repite todas las instrucciones en pantalla

→ exec(nombre_fichero, -1) // para que no repita las instrucciones

Lógicamente, en el nombre del fichero hay que incluir el **path**, caso de que el fichero no esté en el directorio actual. El nombre del fichero debe ir entre apóstrofes o comillas dobles, puesto que es una cadena de caracteres

Ejemplo:

```
x = rand(100,1);
plot(x)
```

Lo guardamos en la raíz del disco "D:", y para ejecutarlo:

→ exec("D:\test.sce")

→ exec("D:\test.sce", -1)

Funciones

Es posible definir nuevas **funciones** Scilab. La diferencia entre un script y una función es que esta última tiene una "interface" de comunicación con el exterior mediante argumentos de entrada y de salida.

Las funciones Scilab responden al siguiente formato de escritura:

function [argumentos de salida] = nombre(argumentos de entrada)

// comentarios

//

. . .

instrucciones (normalmente terminadas por ; para evitar eco en pantalla)

```
...
endfunction
```

Las funciones se pueden definir on-line o bien escribiéndolas en un fichero (ASCII). A los ficheros que contienen funciones Scilab, por convenio, se les pone el sufijo .sci . Las funciones definidas on-line están disponibles de modo inmediato. Las funciones guardadas en un fichero hay que cargarlas en el espacio de trabajo de una sesión de Scilab mediante el comando exec. Se pueden incluir varias funciones en un mismo fichero, una a continuación de la otra.

También es posible definir funciones on-line mediante el comando **deff**:

```
-->deff('[arg_out]=nombre(arg_in)','instrucciones')
```

EJEMPLO

```
-->function [y]=fun1(x), y=2*x+1, endfunction
-->sqrt(fun1(4))
ans  =
    3.

-->deff('[s]=fun2(x,y)','s=sqrt(x^2+y^2)')
-->fun2(3,4)
ans  =
    5.

-->exec('misfunciones.sci',-1)
-->t=5;
-->max(fun3(t),0)
ans  =
    0.
```

FICHERO misfunciones.sci

```
function [z]=fun3(x)
z=sin(x);
endfunction
```

Ejemplo:

```
function [r]=suma(x, y)
  r = x + y;
endfunction
```

Guardado en un archivo **funciones.sci** en la razi del disco D:

Para cargarlo:

```
--> exec("D:/funciones.sci")

--> suma(2,3)

 ans  =
```

5.

Objetos y Sintaxis Básicos

En Scilab, por defecto, los números son codificados como números reales en coma flotante en doble precisión. La precisión, esto es, el número de bits dedicados a representar la mantisa y el exponente, depende de cada (tipo de) máquina.

El objeto básico de trabajo de Scilab es una matriz bidimensional cuyos elementos son números reales o complejos. Escalares y vectores son casos particulares de matrices. También se pueden manipular matrices de cadenas de caracteres, booleanas, enteras y de polinomios. También es posible construir otro tipo de estructuras de datos definidos por el usuario.

Algunas constantes numéricas están predefinidas. Sus nombres comienzan por el símbolo %. En particular **%pi** es el número π, **%e** es el número e, **%i** es la unidad imaginaria, **%eps** es la precisión de la máquina (mayor número real doble precisión para el que 1+%eps/2 es indistinguible de 1), %inf es el infinitomáquina (overflow: cualquier número que supere al mayor número real representable en doble precisión), **%nan** es el símbolo NaN (Not a Number) para una operación inválida (por ejemplo, 0/0 es %nan).

El lenguaje de Scilab es interpretado, esto es, las instrucciones se traducen a lenguaje máquina una a una y se ejecutan antes de pasar a la siguiente. Es posible escribir varias instrucciones en la misma línea, separándolas por una coma o por punto y coma.

Scilab distingue entre mayúsculas y minúsculas: %nan NO ES LO MISMO QUE %Nan

Se pueden recuperar comandos anteriores, usando las teclas de flechas arriba y abajo. Con las flechas izquierda y derecha nos podemos desplazar sobre la línea de comando y modificarlo.

Constantes y operadores aritméticos

Reales: 8.01 -5.2 .056 1.4e+5 0.23E-2 -.567d-21 8.003D-12
Complejos: 1+2*%i
Booleanos: %t %f
Caracteres (entre apóstrofes o comillas): ‘esto es una cadena de caracteres’ “string“
Operadores aritméticos: + - * / ^
Operadores de comparación: == ~= (ó <>) < > <= >=
Operadores lógicos: & | ~

Funciones elementales

Los nombres de las funciones elementales son los habituales. Algunas de ellas:

sqrt(x)	raiz cuadrada	sin(x)	seno (radianes)
abs(x)	módulo	cos(x)	coseno (radianes)
conj(z)	complejo conjugado	tan(z)	tangente (radianes)
real(z)	parte real	cotg(x)	cotangente (radianes)
imag(z)	parte imaginaria	asin(x)	arcoseno
exp(x)	exponencial	acos(x)	arcocoseno
log(x)	logaritmo natural	atan(x)	arcotangente
log10(x)	logaritmo decimal	cosh(x)	cos. hiperbólico
rat(x)	aprox. racional	sinh(x)	seno hiperbólico
modulo(x,y)	resto de dividir x por y	tanh(x)	tangente hiperbólica
floor(x)	n tal que n<=x<(n+1)	acosh(x)	arcocoseno hiperb.
ceil(x)	n tal que (n-1)<x<=n	asinh(x)	arcoseno hiperb.
int(x)	parte entera inglesa: floor(x) si x>=0 ceil(x) si x<0	atanh(x)	arcotangente hiperb.

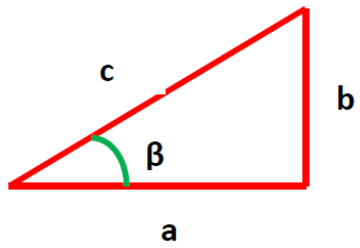
Scilab puede calcular las funciones trigonométricas seno, coseno, tangente, cotangente. El ángulo debe darse en radianes, por lo tanto si el ángulo se da en grados debe convertirse en radianes usando la fórmula

$$radianes = \frac{grados * pi}{180}$$

$$grados = \frac{radianes * 180}{pi}$$

Ejercicio

Para el triángulo rectángulo de la figura su hipotenusa es igual a c = 8.3 y el cateto b = 4.5. Encontrar el valor del otro cateto y el ángulo que forman.



Analíticamente:

$$a = \sqrt{c^2 - b^2}$$

$$\beta = arcsen(b / c)$$

Programando en Scilab,

```
--> // valores conocidos
--> c=8.3;
--> b=4.5;
```

```
--> // cálculo del cateto a
-->a=sqrt(c^2-b^2)
a =
6.9742383
--> // cálculo del ángulo
-->beta=asin(b/c)
beta =
0.5730159
--> // pasar valor a grados
-->ang=beta*180/%pi
ang =
32.831392
```

La respuesta al problema es que el otro cateto vale 6.97 y el ángulo es de 32.8°

Uso como calculadora

Se puede utilizar Scilab como simple calculadora, escribiendo expresiones aritméticas y terminando por **RETURN** (**<R>**). Se obtiene el resultado inmediatamente a través de la variable del sistema ans (answer).
Si no se desea que Scilab escriba el resultado en el terminal, debe terminarse la orden por punto y coma (útil, sobre todo en programación).

EJEMPLO

```
-->sqrt(34*exp(2))/(cos(23.7)+12)
ans  =
    1.3058717

-->7*exp(5/4)+3.54
ans  =
    27.97240

-->exp(1+3*%i)
ans  =
  - 2.6910786 + 0.3836040i
```

Variables

En Scilab las variables no son nunca declaradas: su tipo y su tamaño cambian de forma dinámica de acuerdo con los valores que le son asignados. Así, una misma variable puede ser utilizada, por ejemplo, para almacenar un número complejo, a continuación una matriz 25x40 de números enteros y luego para almacenar un texto. Las variables se crean automáticamente al asignarles un contenido. Asimismo, es posible eliminar una variable de la memoria si ya no se utiliza.

EJEMPLOS

```
-->a=10
a =
     10.
--> pepito=exp(2.4/3)
pepito  =
    2.2255409
-->pepito=a+pepito*(4-0.5*%i)
pepito  =
    18.902164 - 1.1127705i
```

Para conocer en cualquier instante el valor almacenado en una variable basta con teclear su nombre (Atención: recuérdese que las variables AB ab Ab y aB SON DISTINTAS, ya que Scilab distingue entre mayúsculas y minúsculas).

`who`	lista las variables actuales
`whos`	como el anterior, pero más detallado
`clear`	elimina todas las variables que existan en ese momento
`clear a b c`	elimina las variables a, b y c (atención: sin comas!)
`browsevar()`	abre, en ventana aparte, un "ojeador" de la memoria de trabajo de Scilab: permite "ver" el contenido y características de las variables e, incluso, editar su valor.

Algunos comandos utilitarios

Están disponibles algunos comandos utilitarios, como

`ls`	Lista de ficheros del directorio actual (como Unix)
`dir`	Lista de ficheros del dirrectorio (de otra forma)
`pwd`	Devuelve el nombre y path del directorio actual
`cd`	Para cambiar de directorio
`clc`	"Limpia" la ventana de comandos
`date()`	Fecha actual

Ejercicio

Para x = 3.28, calcular el valor de la expresión:

$$y = \frac{x^3(2x^2 + 3.56x - 5.21)}{\sqrt{3x + 2.3}}$$

Con Scilab se resuelve así,

```
--> x = 3.28;
--> y = x^3*(2*x^2+3.56*x-5.21)/sqrt(3*x+2.3)
y =
        283.41039
```

Matrices

Operadores elementales y funciones de construcción de matrices

La forma más elemental de introducir matrices en Scilab es escribiendo sus elementos, **por filas**, entre corchetes rectos (**[ ]**) : elementos de una fila se separan unos de otros por comas y una fila de la siguiente por punto y coma.

EJEMPLOS (construcciones elementales de matrices)

```
-->v=[1,-1,0,sin(2.88)]                  // vector fila longitud 4
-->w=[0;1.003;2;3;4;5*%pi]                // vector columna longitud 6
-->a=[1,2,3,4;5,6,7,8;9,10,11,12]        // matriz 3x4
-->mat=['Hola','Mari';'¿Como','estas?']  // matriz 2x2 de caracteres
```

Observaciones:

- Lo que se escribe en cualquier línea detrás de // es considerado como comentario
- El hecho de que, al introducirlas, se escriban las matrices por filas no significa que internamente, en la memoria del ordenador, estén así organizadas: en la memoria las matrices se almacenancomo un vector unidimensional ordenadas por columnas, como siempre.

Otras órdenes para crear matrices son:

```
-->v1=a:h:b    // crea un vector fila de números desde a hasta un número c <= b
               // tal que c+h > b, con incrementos de h
-->v2=a:b      // como el anterior, con paso h=1
-->v3=v2'      // matriz traspuesta (conjugada si es compleja)
-->v4=v2.'     // matriz traspuesta sin conjugar
```

Se pueden también utilizar los vectores/matrices como objetos para construir otras matrices (bloques):

EJEMPLOS (construcciones elementales de matrices)

```
-->v1=1:4
-->v2=[v1,5;0.1:0.1:0.5]
-->v3=[v2',[11,12,13,14,15]']
```

$$v1 = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix} \qquad v2 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 0.1 & 0.2 & 0.3 & 0.4 & 0.5 \end{pmatrix} \qquad v3 = \begin{pmatrix} 1 & 0.1 & 11 \\ 2 & 0.2 & 12 \\ 3 & 0.3 & 13 \\ 4 & 0.4 & 14 \\ 5 & 0.5 & 15 \end{pmatrix}$$

Algunas funciones útiles para generar matrices aparecen en la tabla siguiente:

`diag(v)`	Si **v** es un vector, **diag(v)** es una matriz cuadrada de ceros con diagonal principal = **v**
`diag(A)`	Si **A** es una matriz, **diag(A)** es un vector = diagonal principal de **A**
`diag(A,k)`	Si **A** es una matriz y **k** es un entero, **diag(A,k)** es un vector = **k**-ésima sub o super diagonal de **A** (según sea **k**<0 ó **k**>0)
`zeros(n,m)`	matriz **n**x**m** con todas sus componentes iguales a cero
`ones(n,m)`	matriz **n**x**m** con todas sus componentes iguales a uno
`eye(n,m)`	matriz unidad: matriz **n**x**m** con diagonal principal =1 y el resto de las componentes =0
`matrix(A,n,m)`	Re-dimensiona una matriz: si **A** es una matriz **h**x**k**, **matrix(A,n,m)** es otra matriz con los mismos elementos que **A**, pero de dimensiones **n**x**m** (tiene que ser **h*****k**=**n*****m**)

`linspace(a,b,n)`	Si **a** y **b** son números reales y **n** un número entero, genera una partición regular del intervalo **[a,b]** con **n** nodos (**n**-1 subintervalos)
`linspace(a,b)`	Como el anterior, pero se asume **n**=100
`logspace(e,f,n)`	Vector con **n** elementos logarítmicamente espaciados desde 10^**e** hasta 10^**f**, es decir
`logspace(e,f)`	Como el anterior, pero se asume **n**=50

La mayoría de las funciones Scilab están hechas de forma que admiten matrices como argumentos. Esto se aplica en particular a las funciones matemáticas elementales y su utilización debe entenderse en el sentido de "elemento a elemento": si A es una matriz de elementos aij, exp(A) es otra matriz cuyos elementos son exp(aij). No debe confundirse con la función exponencial matricial que, a una matriz cuadrada A, asocia la suma de la serie exponencial matricial, y que en Scilab se calcula mediante la función expm.

EJEMPLO (diferencia entre exp y expm)

```
-->A=[1,0;0,2]
-->B=exp(A)
-->C=expm(A)
```

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \qquad B = \begin{pmatrix} 2.7182818 & 1 \\ 1 & 7.3890561 \end{pmatrix} \qquad C = \begin{pmatrix} 2.7182818 & 0 \\ 0 & 7.3890561 \end{pmatrix}$$

Manipulación de los elementos de una matriz. Extracción, inserción y eliminación

Para especificar los elementos de una matriz se usa la sintaxis habitual:

`v(i)`	Si `v` es un vector es v_i
`A(i,j)`	Si `A` es una matriz, es a_{ij}
`A(k)`	Si `A` es una matriz, es el **k**-ésimo elemento de A, en el orden en que está almacenada en la memoria (por columnas)

Pero Scilab posee un buen número de facilidades para designar globalmente un conjunto de elementos de una matriz o vector, consecutivos o no.

Por ejemplo, si v es un vector y h es un vector de subíndices, v(h) hace referencia al subconjunto de componentes de v correspondientes a los valores contenidos en h. Análogamente con A(h,k) si A es una matriz bidimensional y h y k son vectores de subíndices. El símbolo **:** (dos puntos) en el lugar de un índice indica que se toman todos. El símbolo $ indica el último valor del subíndice.

EJEMPLOS (referencias a subconjuntos de elementos de una matriz mediante vectores de subíndices)

```
-->A=[1.1,1.2,1.3;2.1,2.2,2.3;3.1,3.2,3.3]
-->A(2:3,1:2)
-->A(:,2)       // representa la segunda columna de A
-->A(:,2:$)     // representa las columnas desde 2 hasta la última
-->A(:)         // representa todos los elementos de A, en una sola columna
```

$$A = \begin{pmatrix} 1.1 & 1.2 & 1.3 \\ 2.1 & 2.2 & 2.3 \\ 3.1 & 3.2 & 3.3 \end{pmatrix} \qquad \begin{pmatrix} 2.1 & 2.2 \\ 3.1 & 3.2 \end{pmatrix}$$

Esta sintaxis para designar conjuntos de elementos de un matriz puede usarse tanto para recuperar los valores que contienen (para, por ejemplo, utilizarlos en una expresión), como para asignarles valores.
Cuando estas expresiones aparecen a la izquierda de un signo igual (es decir, en una instrucción de asignación) pueden tener distintos significados:

```
EJEMPLOS (asignación de valores a partes de una matriz y modificación de su dimensión) (se recomienda ejecutarlos para comprender sus efectos)

-->A=rand(4,4)    // Se almacena en A una matriz 4x4 de num. aleatorios

-->A(2,2)=0       // Se modifica el segundo elem. diagonal de A

-->A(5,2)=1       // Obsérvese que A(5,2) no existía:: de hecho, se
                  // MODIFICAN LAS DIMENSIONES de la matriz, para AÑADIR
                  // el elemento A(5,2). El resto se llena con ceros.
                  // Ahora A es una matriz 5x4

-->A(2:3,1:2)=1   // La submatriz A(2:3,1:2) se llena con unos

-->A(:,2)=[]      // El símbolo [] representa una "matriz vacía"
                  // Esta instrucción ELIMINA la segunda columna de A
                  // Ahora A tiene dimensión 5x3

-->A=[A,[1:5]']   // Se añade a A una nueva columna al final
```

REPRESENTACIONES GRÁFICAS CON SCILAB

Estas notas sólo pretenden exponer algunos de los comandos más básicos de que dispone Scilab para generar gráficos. En principio se exponen, exclusivamente, los comandos de dibujo y posteriormente se explicará cómo modificar los distintos (y numerosos) parámetros que determinan sus características.

Curvas planas

El comando básico de Scilab para dibujar curvas planas es **plot2d**.

plot2d

Dados dos vectores (VER **(*)**)

```
x=[x1,x2,. . . ,xn] e y=[y1,y2, . . . ,yn]
```

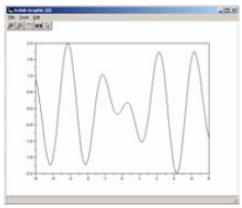
```
-->plot2d(x,y)
```

dibuja la curva que pasa por los puntos **(x1,y1). . .(xn,yn)**

Ejemplo: dibujar, en **[-5,5]**

$$y = 2\,\text{sen}\,(\frac{x}{2})\cos(3x)$$

```
x=linspace(-5,5)';
y=2*sin(x/2).*cos(3*x);    // ( VER (**) )
plot2d(x,y)
```



También se podría poner, directamente:

```
x=linspace(-5,5)';
plot2d(x, 2*sin(x/2).*cos(3*x))
```

(*) Para dibujar una única curva, es indiferente que los vectores **x** e **y** sean filas o columnas. Sin embargo, cuando se desean dibujar varias curvas juntas no lo es. Por ello, usaremos siempre vectores-columna.

(**) Obsérvese que, puesto que se calculan todas las ordenadas "de una sola vez", es preciso "vectorizar" la escritura de la fórmula, para que, al ser el argumento x un vector, la fórmula devuelva un vector de las mismas dimensiones calculado elemento e elemento.

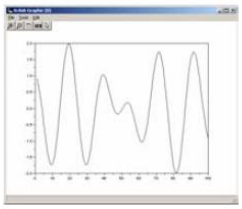
Si se hubiera escrito **y=2*sin(x/2)*cos(3*x)** se hubiera obtenido un error, ya que x es un vector columna de longitud 100 (i.e. una matriz 100x1), luego también **sin(x/2)** y **cos(3*x)** son matrices 100x1 y la multiplicación (matricial) **sin(x/2)*cos(3*x)** carece de sentido.

```
-->plot2d(y)
```

dibuja la curva que pasa por los puntos **(1,y1). . .(n,yn)**

Ejemplo:

```
x=linspace(-5,5)';
y=2*sin(x/2).*cos(3*x);
plot2d(y)
```

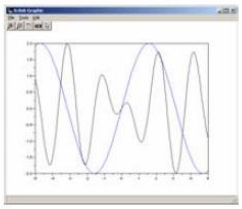


```
-->plot2d(x,y)
```

siendo **x** un vector e **y** una matriz dibuja una curva por cada columna de **y**

Ejemplo:

```
x=linspace(-5,5)';
y=2*sin(x/2).*cos(3*x);
z=2*sin(x);
w=[y,z];
plot2d(x,w)
```



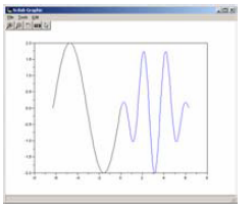
```
-->plot2d(x,y)
```

siendo **x** e **y** matrices de las mismas dimensiones, dibuja una curva por cada par (columna de **x** , columna de **y**)

Ejemplo: dibujar en **[-2 pi , 2 pi]** la gráfica de la función:

$$y = \begin{cases} 2\,\mathrm{sen}\,(x), & \text{si } x \leq 0 \\ 2\,\mathrm{sen}\,(\frac{x}{2})\cos(3x), & \text{si } x > 0 \end{cases}$$

```
x1=linspace(-2*%pi,0)';
x2=linspace(0,2*%pi)';
y1=2*sin(x1);
y2=2*sin(x2/2).*cos(3*x2);
plot2d([x1,x2],[y1,y2])
```

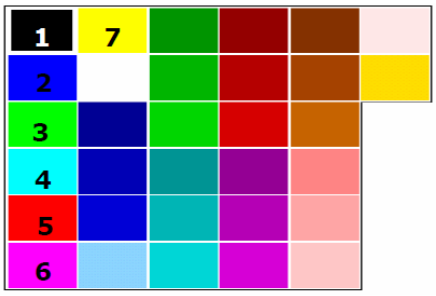


Observaciones:

• Obsérvese que, por defecto, gráficas sucesivas se superponen. Para evitarlo, hay que borrar la gráfica anterior antes de dibujar de nuevo. Ello puede hacerse ó bien cerrando la ventana gráfica ó bien borrando su contenido desde la barra de menús (**Edit --> Clear Figure**) o mediante un comando (**xbasc()**).

• Cuando se borra el contenido de la ventana gráfica, pero no se cierra, se conservan sus características. Si, por ejemplo, se ha modificado la carta de colores de esa ventana, se seguirá conservando la carta al borrarla, pero no al crerrarla.

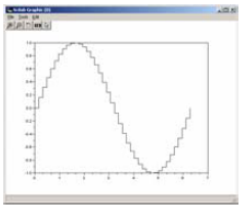
• Cuando plot2d dibuja varias curvas, les asigna distintos colores. El orden de los colores asignados viene determinado por la carta de colores activa. Por defecto, es la siguiente:



plot2d2, plot2d3, plot2d4

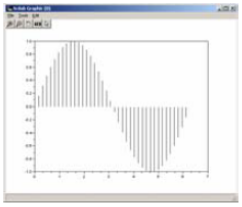
```
-->plot2d2(x,y)
```

es similar a plot2d, pero dibuja una función constante a trozos (escalonada)



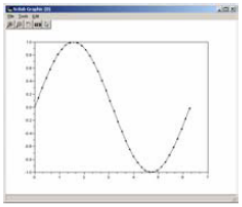
```
-->plot2d3(x,y)
```

es similar a plot2d, pero dibuja una función utilizando líneas verticales



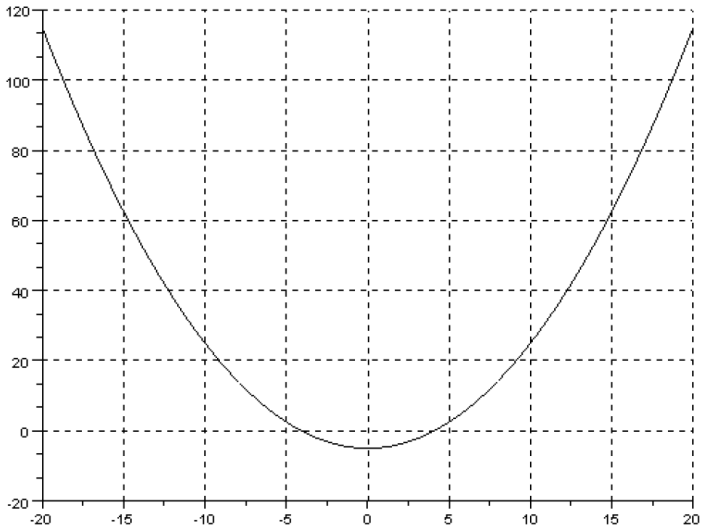
```
-->plot2d4(x,y)
```

es similar a plot2d, pero dibuja una curva utilizando puntas de flecha (por ejemplo para señalar la dirección de una trayectoria)



Ejercicio

```
-->clf
-->x=[-20:0.1:20];
-->y=0.3*x^2-5;
-->plot2d(x,y)
-->xgrid
```



-->// vector por matriz

-->clf

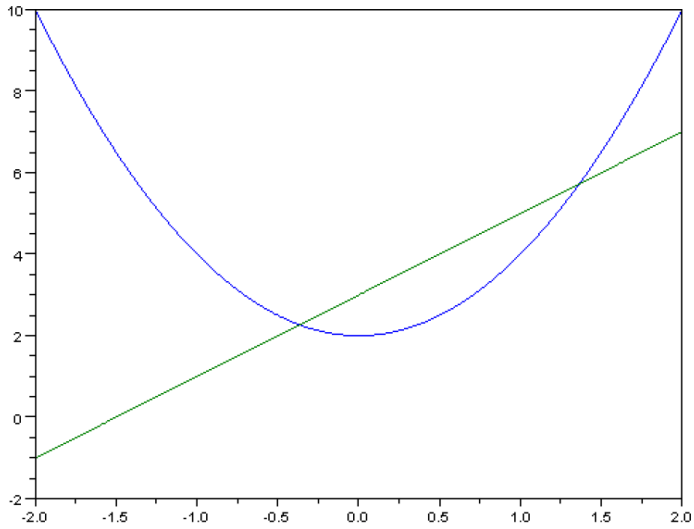
-->t=[-2:0.01:2];

-->f=2*t^2+2;

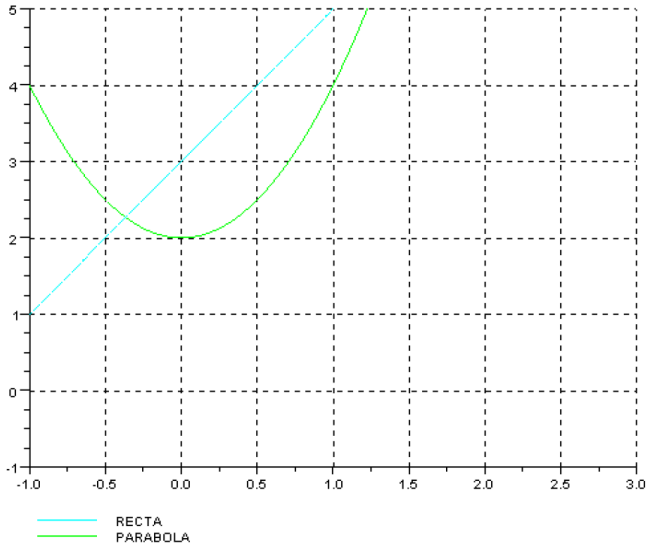
-->g=2*t+3;

-->plot2d(t,[f'g'])

La gráfica resultante es



-->// colocar colores, leyenda y límites a ejes
-->clf
-->plot2d(t,[f' g'],[2 4],leg="RECTA@PARABOLA")
-->plot2d(t,[f' g'],[3 4],leg="PARABOLA@RECTA",rect=[-1 -1 3 5])
-->xgrid



Comando plot

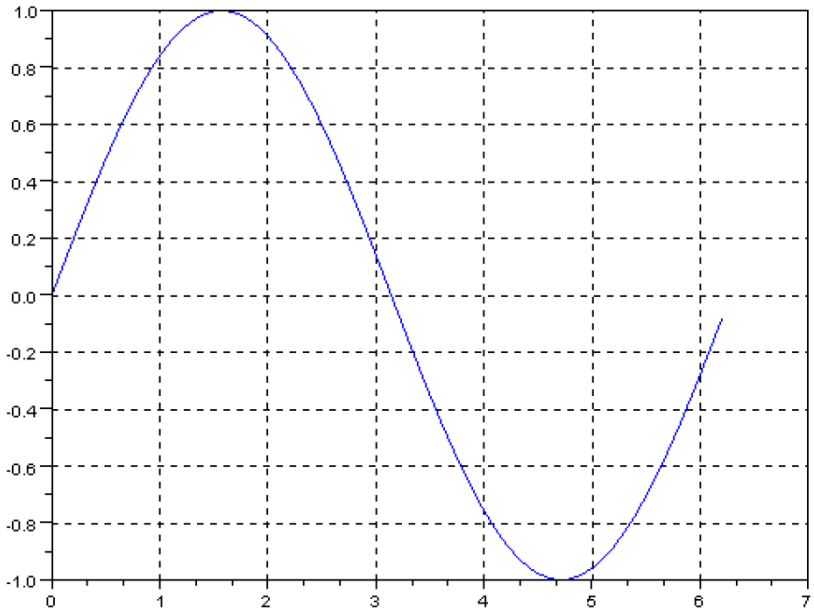
Un comando similar a plot2d es **plot(x,y)** donde x es un vector fila que contiene los valores del eje x, y es la función a graficar y = f(x)

Ejemplo:

Graficar la función del seno para valores en el eje x de 0 a 2pi

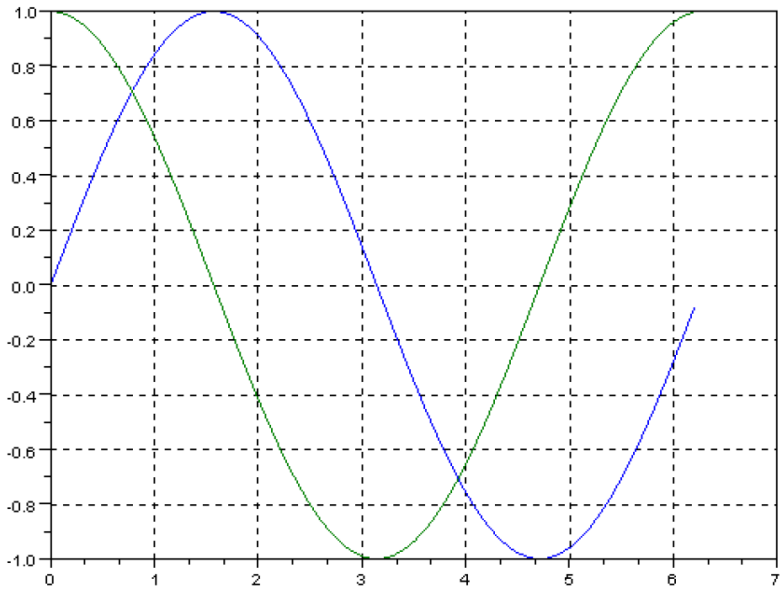
```
-->x=[0:0.1:2*%pi];
-->y=sin(x);
-->plot(x,y)
-->// Poner rejilla
-->xgrid
```

Da como resultado:



También se puede hacer directamente,

```
-->plot(x,sin(x))
-->// graficar dos figuras
-->// borrar figura anterior
-->clf
-->plot(x,sin(x),x,cos(x));
-->xgrid
```



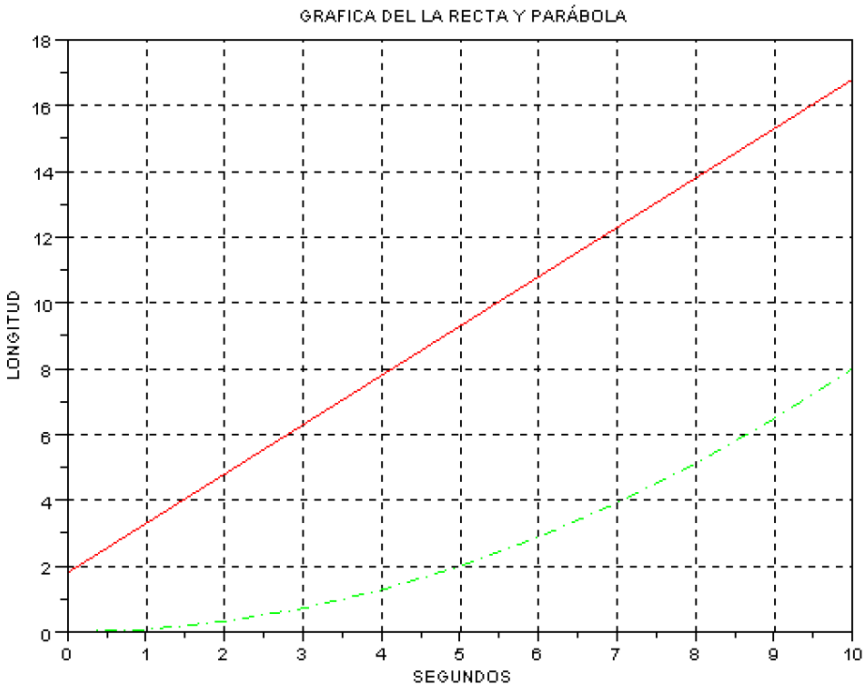
Parámetros

Se utilizan para definir el tipo de línea, su color y su forma.

Línea	Color	Marca	Rejilla
- Sólida	r rojo	+ más	0,1 negro
-. Rayada	g verde	o círculo	2 azul
: Punteada	b azul	*asterisco	3 verde
-.Raya y punto	c cyan	. punto	4 cyan
	m magenta	x por	5 rojo
	y amarillo	s cuadrado	6 violeta, etc
	k negro	d rombo	
	w blanco	^ v < > triángulos	

Ejemplos:

```
-->clf // borrar gráfica anterior
-->t=0:1:10; // valores del eje x (tiempo)
-->x=1.5*t+1.8; // ecuación de una recta
-->y=0.08*t^2; // ecuación de una parábola
-->plot(t,x,'r-',t,y,'g-.') // graficar las dos curvas
-->xgrid // poner rejilla
-->xtitle('GRAFICA DEL LA RECTA Y PARÁBOLA','SEGUNDOS','LONGITUD')
```



Subgraficas

Para dibujar varias gráficas en la misma ventana se usa el comando subplot que permite dividir la ventana Windows en varias subventanas, su sintaxis es,

subplot(mnq)

Donde m es el número de filas, n el de columnas y q el de la subgráfica.

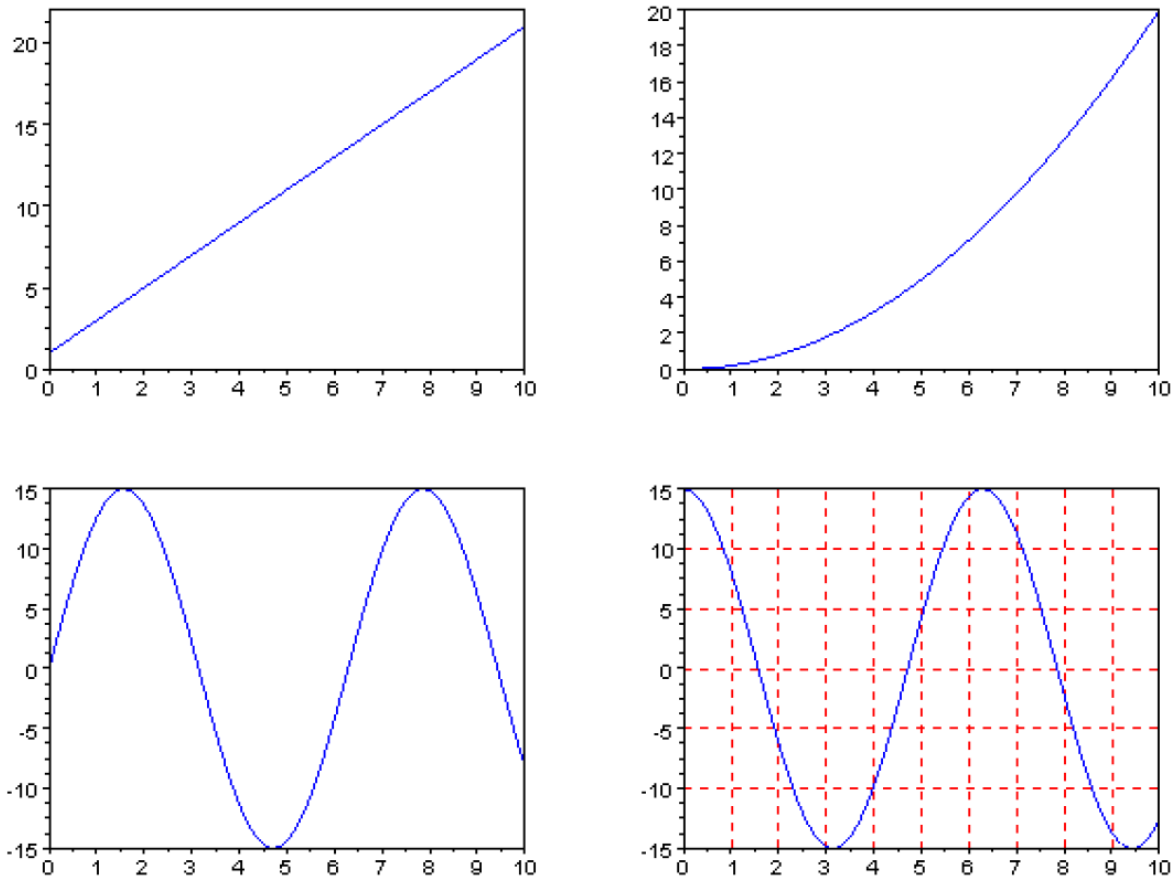
Ejemplo:

```
->clf
-->x=[0:0.1:10];
-->subplot(221) // primera gráfica
-->y1=2*x+1;
-->plot(x,y1)
-->subplot(222) // segunda gráfica
-->y2=0.2*x^2;

-->plot(x,y2)

-->subplot(223) // tercera gráfica
-->y3=15*sin(x);
-->plot(x,y3)
-->subplot(224) // cuarta gráfica
-->y4=15*cos(x);
-->plot(x,y4)
-->xgrid(4)
```

El resultado de Scilab es:



Sentencias de control

Bucle for

```
for i=1:5
    disp(i)
end
```

El resultado es:

```
-->for i=1:5
--> disp(i)
-->end

    1.

    2.

    3.

    4.

    5.
```

Otro ejemplo del bucle *for*

```
x = zeros(1,10)
x
for i = [1 2 3 4]
  x(i) = i
end
```

Ejemplo de bucle while

```
i=0;
while i<5
    disp(i)
```

```
    i=i+1;
end
```

Resultado:

```
-->while i<5
--> disp(i)
--> i=i+1;
-->end

    0.

    1.

    2.

    3.

    4.
```

Condicional

```
if expr1 then sentencias
elseif expri then sentencias
....
else sentencias
end
```

Ejemplo

```
function f1=pcondicional(a,b)
if a>b then
    f1="a es mayor b"
elseif a>.5
    f1="a es mayor a 0.5"
else
```

```
    f1="a es menor a b"
end
endfunction
```

Se muestran varias llamadas a la función con diferentes valores para a y b.

```
-->pcondicional(1,2)
 ans  =

 a es mayor a .5

-->pcondicional(5,2)
 ans  =

 a es mayor b

-->pcondicional(-1,2)
 ans  =

 a es menor a b
```

Ejercicio

Crear una función para determinar si un número es par o impar

```
function [r]=par(x)
  if modulo(x,2)==0 then
    r = "Es par"
  else
    r = "Es impar"
  end
endfunction
```