

SCILAB

SCILAB è un prodotto free sviluppato da INRIA per applicazioni di controllo di sistemi e signal processing.

E' distinto in tre parti distinte: un interprete, le librerie di funzioni (Scilab procedures) e le librerie di routines in C e in Fortran.

Le caratteristiche più importanti di SCILAB sono:

- la semplicità con la quale si può manipolare le matrici;
- l'analisi di sistemi non-lineari;
- la possibilità di manipolare facilmente polinomi e matrici polinomiali ;
- l'interfacciabilità con funzioni Fortran e C;
- la definizione grafica e la simulazione con SCICOS di sistemi particolarmente complessi;
- la possibilità di utilizzare SCILAB con sistemi operativi free (GNU/Linux) o a pagamento (windows);
- l'interfaccia di problemi LMI;
- case-sensitive;
- la quantità di DEMO e HELP completo;
- l'assoluta gratuità e completezza.

Alla partenza si ottiene il seguente schema con il cursore lampeggiante in attesa di un comando:

scilab-3.0

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

Startup execution:

loading initial environment

-->

Tipi di dati

SCILAB riconosce molti tipi di dati:

1. Costanti Speciali;
2. Matrici costanti;
3. Matrici di caratteri stringa;
4. Polinomi e matrici polinomiali;
5. Matrici Booleane;
6. Matrici intere;
7. Liste;
8. Sistemi lineari;
9. Funzioni;
10. Librerie.

1) Costanti Speciali

Scilab prevede le seguenti costanti speciali: %i (rappresenta l'immaginario), %pi (rappresenta il π), %e (rappresenta la costante $e = 2.7182818\dots$), %eps (rappresenta la precisione di una macchina; è il più grande numero per il quale $1+\%eps = 1$), %inf (rappresenta Infinity), %nan (rappresenta NotANumber). %s è il polinomio $s = \text{poly}(0,'s')$ con simbolo s.

%t e %f rappresentano rispettivamente true e false.

2) Matrici costanti

Scilab considera molti dati come matrici. Sia scalari che vettori vengono considerati delle matrici *Scalari* Sono numeri reali o complessi.

Ricordiamo che con il ; finale non viene stampato il risultato.

```
-->a = 5 - 4 * %i
```

```
a =
```

```
5. - 4.i
```

```
-->B = -2 +%i;
```

```
-->b = 4-3*%i
```

```
b =
```

```
4. - 3.i
```

```
-->a*b
```

```
ans =
```

```
8. - 31.i
```

Vettori Un vettore viene creato inserendo gli elementi (separati da uno spazio o da una virgola) tra parentesi quadre e ponendolo uguale ad una variabile:

```
-->v = [1 2 3 4 5 %i]
```

```
v =
```

```
! 1. 2. 3. 4. 5. i !
```

E' possibile inserire dei punti e virgola per indicare i vettori colonna:

```
-->w = [-3; 4; -3; +2*%i; 6]
```

```
w =
```

```
! - 3. !
```

```
! 4. !
```

```
! - 3. !
```

```
! 2.i !
```

```
! 6. !
```

```
-->v'
```

```
ans =
```

```
! 1. !
```

```
! 2. !
```

```
! 3. !
```

```
! 4. !
```

```
! 5. - i !
```

```
-->v'+w
```

```
ans =
```

```
! - 2. !
```

```
! 6. !
```

```
! 0 !
```

```
! 4. + 2.i !
```

```
! 11. - i !
```

```
-->v*w
```

```
ans =
```

```
26. + 14.i
```

```
-->w' .* v // prodotto numero per numero
```

```
ans =
```

```
! - 3. 8. - 9. - 8.i 30. + 6.i !
```

E' possibile costruire vettori di elementi che incrementano o decrementano:

```
-->v=5:-.5:3
```

```
v =
```

```
! 5. 4.5 4. 3.5 3. !
```

Quando non specificato l'incremento (o il decremento) vale 1 (o -1).

```
-->g=5:10
```

```
g =
```

```
! 5. 6. 7. 8. 9. 10. !
```

Si possono creare vettori usando **ones** e **zeros**

```
-->ones(g)
```

```
ans =
```

```
! 1. 1. 1. 1. 1. 1. !
```

```
-->zeros(g)
```

```
ans =
```

```
! 0. 0. 0. 0. 0. 0. !
```

```
-->3*ones(1:3)
```

ans =

! 3. 3. 3.!

Matrici Gli elementi riga sono separati da spazi o virgole e gli elementi colonna da punti e virgola. Generalmente per distinguere le matrici dai vettori si usano variabili maiuscole. E' possibile effettuare tutte le operazioni tra matrici e tra matrici e scalari o vettori.

```
-->A=[2 1 4; 5 -8 2] //matrice 2x3
```

A =

! 2. 1. 4.!

! 5. -8. 2.!

```
-->B=ones(2,3)
```

B =

! 1. 1. 1.!

! 1. 1. 1.!

```
-->A.*B
```

ans =

! 2. 1. 4.!

! 5. -8. 2.!

```
-->A*B' //B' è la trasposta di B (3x2) ed il risultato è una matrice (2x2)
```

ans =

! 7. 7.!

! -1. -1.!

```
-->A=[1,2;3,4]
```

A =

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

```
! 3. 4.!
```

```
-->B=[5 6;7 8];
```

```
-->C=[9 10;11,12]
```

```
C =
```

```
! 9. 10.!
```

```
! 11. 12.!
```

```
-->D=[A,B,C] //Compatta le matrici
```

```
D =
```

```
! 1. 2. 5. 6. 9. 10.!
```

```
! 3. 4. 7. 8. 11. 12.!
```

```
-->E=matrix(D,3,4) //Crea una nuova matrice partendo da D con 3 righe e 4 colonne
```

```
E =
```

```
! 1. 4. 6. 11.!
```

```
! 3. 5. 8. 10.!
```

```
! 2. 7. 9. 12.!
```

Infine, la funzione eye crea una matrice con 1 sulla diagonale principale:

```
-->F = eye(E)
```

```
F =
```

```
! 1. 0. 0. 0.!
```

```
! 0. 1. 0. 0.!
```

```
! 0. 0. 1. 0.!
```

```
-->G = eye(4,3)
```

```
G =
```

```
! 1. 0. 0.!
```

```
! 0. 1. 0.!
```

```
! 0. 0. 1.!
```

```
! 0. 0. 0.!
```

Vediamo l'estrazione della prima riga e dell'ultima colonna di una matrice A definita:

```
-->A=[1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

```
! 1. 2. 3.!
```

```
! 4. 5. 6.!
```

```
! 7. 8. 9.!
```

```
-->A(1,:)
```

```
ans =
```

```
! 1. 2. 3.!
```

```
-->A(:,9)
```

```
ans =
```

```
! 3.!
```

```
! 6.!
```

```
! 9.!
```

3) Matrici di caratteri stringa

Sono matrici ordinarie, con la capacità di manipolare e creare funzioni. La concatenazione di stringhe può essere effettuata con il simbolo +

```
A =
```

```
!x y !
```

```
!z w+v !
```

```
-->At=trianfml(A)
```

```
At =
```

```
!z w+v !
```

```
!0 z*y-x*(w+v) !
```

```
-->x=1;y=2;z=3;w=4;v=5;
```

```
-->evstr(At) //evaluation string
```

```
ans =
```

```
! 3. 9.!
```

```
! 0. -3.!
```

4) Polinomi e Matrici polinomiali

I polinomi vengono creati e manipolati in SCILAB molto semplicemente.

La primitiva **poly** può essere usata per specificare i coefficienti o le radici (roots) di un polinomio:

```
-->p=poly([1 2], 's') //polinomio definito dalle sue radici
```

```
p =
```

$$2 - 3s + s^2$$

```
-->q=poly([1 2], 's', 'c') //polinomio definito a partire dai coefficienti
```

```
q =
```

$$1 + 2s$$

```
-->p+q
```

```
ans =
```

$$3 - s + s^2$$

```
-->p*q
```

```
ans =
```

$$2 + s - 5s^2 + 2s^3$$

```
-->q/p
```

```
ans =
```

$$1 + 2s$$

$$2 - 3s + s^2$$

Per conoscere le radici di un polinomio appena dichiarato si utilizza la funzione **roots**:

```
-->x=poly(0,'x');
```

```
-->p=2+3*x+x^2
```

```
p =
```

$$2 + 3x + x^2$$

```
-->roots(p)
```

```
ans =
```

```
! - 1. !
```

```
! - 2. !
```

I polinomi possono essere usati come elementi di una matrice:

```
-->s=poly(0,'s');
```

```
-->A=[1 s; s 1+s^2]
```

```
A =
```

```
! 1   s   !
```

```
! s   1+s^2 !
```

```
-->B=[1/s 1/(1+s); 1/(1+s) 1/s^2]
```

```
B =
```

```
! 1       1   !
```

```
! -       ---- !
```

```
! s       1+s !
```

```
!         !
```

```
! 1       1   !
```

```
! ----   -   !
```

```
! 1+s    s^2 !
```

SCILAB automaticamente permette la semplificazione dei poli e zeri. E' anche possibile evitare la semplificazione automatica, settando opportunamente la funzione `simp_mode`.

5) Matrici Booleane

Le costanti booleane sono, come visto, `%t` e `%f`. Possono essere usate nelle matrici booleane con la stessa sintassi di quelle ordinarie. I simboli delle operazioni sono `==` e `~`.

```
-->%t
```

```
%t =
```

```
T
```

```
-->[1 2]==[1 3]
```

```
ans =
```

```
! T F !
```

```
-->[1 2]==1
```

```
ans =
```

```
! T F !
```

```
-->a=1:5; a(a>2)
```

```
ans =
```

```
! 3. 4. 5. !
```

```
-->A=[%t,%f,%t,%f,%f,%f]
```

```
A =
```

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

```
-->B=[%t,%f,%t,%f,%t,%t]
```

```
B =
```

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

```
-->A|B //Corrisponde a OR
```

```
ans =
```

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

```
-->A&B //Corrisponde a AND
```

```
ans =
```

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

6) Matrici intere

In SCILAB sono definiti 6 tipi di interi: a 8, 16, 32 bit con e senza segno.

```
-->x=[0 3.2 27 135];
```

```
-->int32(x)
```

```
ans =
```

```
! 0 3 27 135 !
```

```
-->int8(x)
```

```
ans =
```

```
! 0 3 27 -121 !
```

```
-->uint8(x)
```

```
ans =
```

```
! 0 3 27 135 !
```

```
-->double(ans)
```

```
ans =  
! 0. 3. 27. 135.!
```

Si possono applicare le operazioni aritmetiche e di confronto.

```
-->x=int16([1 5 12])
```

```
x =  
! 1 5 12!
```

```
-->x([1 3])
```

```
ans =  
! 1 12!
```

```
-->x+x
```

```
ans =  
! 2 10 24!
```

```
-->x*x'
```

```
ans =  
170
```

```
-->y=int16([1 7 11])
```

```
y =  
! 1 7 11!
```

```
-->x>y
```

```
ans =  
! F F T!
```

```
-->x
```

```
x =  
! 1 5 12!
```

```
-->x|int16(2)
```

```
ans =
```

```
! 3 7 14 !
```

```
-->int16(14)&int16(2)
```

```
ans =
```

```
2
```

```
-->~uint8(2)
```

```
ans =
```

```
253
```

7) Liste

La lista è una collezione di dati non necessariamente dello stesso tipo. Una lista può contenere tutti i tipi di dati visti e altre liste. Esistono due tipi di liste: ordinarie e liste tipo. Una lista è definita dalla funzione *list*.

Esempi di liste ordinarie:

```
-->L = list(1, 'w', ones(2, 2)) // L è una lista di tre entità diverse
```

```
L =
```

```
L(1)
```

```
1.
```

```
L(2)
```

```
w
```

```
L(3)
```

```
! 1. 1. !
```

```
! 1. 1. !
```

```
-->L(3) // Estrazione della terza entità dalla lista L
```

```
ans =  
! 1. 1. !  
! 1. 1. !
```

```
-->L(3) (2,2) // prende l'elemento 2x2 della matrice L(3)
```

```
ans =
```

```
1.
```

```
-->L(2)=list('w',rand(2,2)) //Adesso L(2) è una lista
```

```
L =
```

```
L(1)
```

```
1.
```

```
L(2)
```

```
L(2)(1)
```

```
w
```

```
L(2)(2)
```

```
! 0.2113249 0.0002211 !
```

```
! 0.7560439 0.3303271 !
```

```
L(3)
```

```
! 1. 1. !
```

```
! 1. 1. !
```

```
-->L(2) (2) (1,2) //Estrazione dell'elemento 1x2 del secondo elemento di L(2)
```

```
ans =
```

0.0002211

-->L(2) (2) (1,2)=5 //assegnazione di un nuovo valore

L =

L(1)

1.

L(2)

L(2)(1)

w

L(2)(2)

! 0.2113249 5. !

! 0.7560439 0.3303271 !

L(3)

! 1. 1. !

! 1. 1. !

Le liste tipo (*tlist*) hanno una prima entità che può essere un carattere stringa (il tipo) o un vettore di caratteri stringa (allora, il primo componente è il tipo, e gli elementi seguenti i nomi degli elementi della lista).

Esempi di tali liste:

-->L=tlist(['Auto';'Nome';'Dimensioni'],'ALFA',[2,3])

L =

L(1)

!Auto !
! !
!Nome !
! !
!Dimensioni !

L(2)

ALFA

L(3)

! 2. 3. !

-->L.Nome //Stessa cosa di L(2)

ans =

ALFA

-->L.Dimensioni(1,2)=2.3 //Modifica il secondo valore di Dimensioni: da 3 diventa 2.3

L =

L(1)

!Auto !
!Nome !
!Dimensioni !

L(2)

ALFA

L(3)

! 2. 2.3 !

-->L(3) (1,2)

ans =

2.3

-->L(1)(1)

ans =

Auto

8) Sistemi lineari

I sistemi lineari vengono trattati come *tlist*. La funzione di base utilizzata per definire i sistemi lineari è *syslin*. Questa funzione riceve come parametri le matrici costanti che definiscono un sistema lineare come variabili di stato o, nel caso di sistemi in forma di trasformata, i suoi ingressi possono essere una matrice razionale.

Per chiarire, la sequenza è **S1=syslin('dom',A,B,C,D,x0)** oppure **S1=syslin('dom',trmat)**.

dom rappresenta il carattere 'c' o 'd' per sistemi a tempo continuo o sistemi a tempo discreto.

E' noto che D può essere una matrice polinomiale (sistemi impropri); D e x0 sono argomenti opzionali.

Trmat è una matrice razionale.

syslin converte gli argomenti in una *tlist* S1. Per lo spazio di stato la S1 è la

tlist(['lss', 'A', 'B', 'C', 'D'], A, B, C, D, 'dom').

Le funzioni che permettono la conversione da una rappresentazione all'altra sono *ss2tf* o *tf2ss*.

Esempi:

-->A=[0 -1;1 -3]; B=[0;1]; C=[-1 0];

-->SL = syslin('c',A,B,C)

SL =

SL(1) (state-space system:)

!lss A B C D X0 dt !

SL(2) = A matrix =

! 0. - 1. !

! 1. - 3. !

SL(3) = B matrix =

! 0. !

! 1. !

SL(4) = C matrix =

! - 1. 0. !

SL(5) = D matrix =

0.

SL(6) = X0 (initial state) =

! 0. !

! 0. !

SL(7) = Time domain =

c

-->HS=ss2tf(SL) //converte nella f.d.T.

HS =

$$\frac{1}{1 + 3s + s^2}$$

-->HS.num //numeratore della f.d.T.

ans =

1

-->HS.den //denominatore della f.d.T.

ans =

$$1 + 3s + s^2$$

-->typeof(HS) //tipo di funzione

ans =

rational

-->//Inversione della matrice di trasferimento

-->inv(HS)

ans =

$$\frac{1 + 3s + s^2}{1}$$

-->// Inversione della forma dello spazio di stato

-->inv(SL)

ans =

ans(1) (state-space system:)

!lss A B C D X0 dt !

ans(2) = A matrix =

[]

ans(3) = B matrix =

[]

ans(4) = C matrix =

[]

ans(5) = D matrix =

$1 + 3s + s^2$

ans(6) = X0 (initial state) =

[]

ans(7) = Time domain =

c

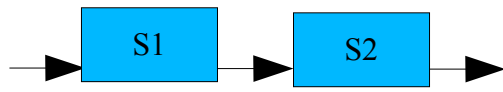
-->//Convertiamo l'inversa nella f.d.t.

-->ss2tf(ans)

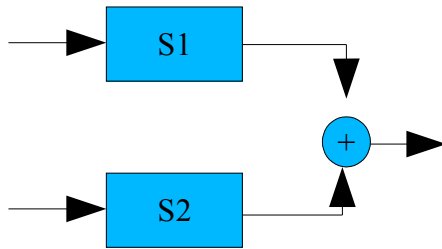
ans =

$$1 + 3s + s^2$$

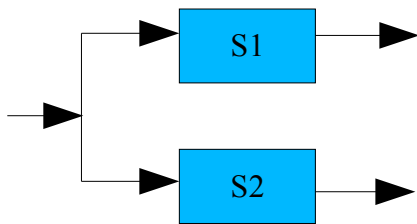
I sistemi lineari possono essere interconnessi come mostrato:



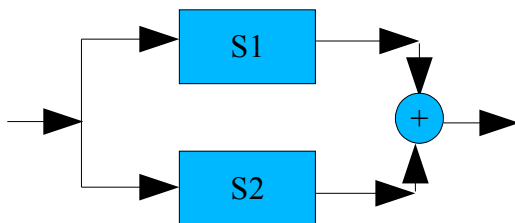
$S2 * S1$



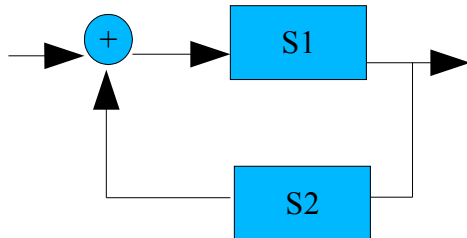
$[S1, S2]$



$[S1; S2]$



$S1 + S2$



$S1 / S2$

Esempio completo:

```
-->s=poly(0,'s');
```

```
-->S1=1/(s-1)
```

S1 =

1

- 1 + s

```
-->S2=1/(s-2)
```

S2 =

1

- 2 + s

```
-->SL1=syslin('c',S1);
```

```
-->SL2=syslin('c',S2);
```

```
-->Gls=tf2ss(S2);
```

```
-->ssprint(Gls)
```

•

$$\dot{x} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} x + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u$$
$$y = \begin{bmatrix} 1 & 1 \end{bmatrix} x$$

```
-->hls=Gls*S1;
```

-->ssprint(hls)

$$\bullet \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$x = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} x$$

-->ht=ss2tf(hls)

ht =

$$\frac{1}{2 - 3s + s^2}$$

-->S2*S1

ans =

$$\frac{1}{2 - 3s + s^2}$$

-->S1+S2

ans =

$$\frac{-3 + 2s}{2 - 3s + s^2}$$

-->[S1,S2]

ans =

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

```
! ----- !
! - 1 + s - 2 + s !
```

```
-->[S1;S2]
```

```
ans =
```

```
! 1 !
! ----- !
! - 1 + s !
!      !
! 1 !
! ----- !
! - 2 + s !
```

```
-->S1/S2
```

```
ans =
```

```
- 2 + s
-----
3 - 3s + s2
```

```
-->S1./(2*S2) // Semplice divisione di polinomi
```

```
ans =
```

```
- 2 + s
-----
2 + 2s
```

9) Funzioni

Le funzioni sono formate da un insieme di comandi che vengono eseguiti in un nuovo ambiente isolando così le variabili delle funzioni da quelle iniziali. Possono essere create ed eseguite in vari modi. Esse possono essere argomenti di altre funzioni e possono essere elementi in una lista.

Possono essere create mediante un text editor o direttamente in SCILAB usando la sintassi **function** o la primitiva **deff**.

Editare la funzione per poi richiamarla:

```
-->function [x]=foo(y)
-->  if y>0 then, x=1; else, x=-1; end
-->endfunction
```

Dal prompt di SCILAB:

```
-->deff('[x]=foo(y)','if y>0 then, x=1; else, x=-1;end')
```

```
-->foo(5)
```

```
ans =
```

```
1.
```

10) Librerie

Le librerie sono insiemi di funzioni che possono essere caricate automaticamente all'avvio di SCILAB o successivamente quando necessario. Sono create dal comando **lib**.

Operazioni sulle matrici

La tabella seguente mostra le possibili operazioni sulle matrici.

| <i>SIMBOLI</i> | <i>OPERAZIONE</i> |
|-----------------------|-------------------------------------|
| [] | Definizione matrice, concatenazione |
| ; | Separatore righe |
| () | Estrazione $m=a(k)$ |
| () | Inserzione: $a(k) = m$ |
| ' | trasposta |
| + | addizione |
| - | sottrazione |
| * | moltiplicazione |
| \ | Divisione sinistra |
| / | Divisione destra |

| <i>SIMBOLI</i> | <i>OPERAZIONE</i> |
|----------------|----------------------------|
| ^ | Esponente |
| .* | Elementwise multiplication |
| .\ | Elementwise left division |
| / | Elementwise right division |
| .^ | Elementwise exponent |
| .*. | Kronecker product |
| .\. | Kronecker left division |
| ./. | Kronecker right division |

Programmazione

Una delle più importanti caratteristiche di SCILAB è quella di poter creare e usare funzioni. E' possibile utilizzare programmi che si possono integrare con SCILAB in modo semplice e modulare, attraverso, per esempio, l'uso di librerie.

SCILAB supporta molti strumenti di programmazione: loops, conditionals, case selection.

LOOPS:

Esistono due tipi di cicli: *for* e *while*.

-->x=1; for k=1:4,x=x*k, end //Esempio di for

x =

1.

x =

2.

x =

6.

x =

24.

-->x=1; for k=[-1 3 0],x=x+k, end

x =

0.

x =

3.

x =

3.

-->x=1; while x<14, x=2*x, end //Esempio di while

x =

2.

x =

4.

x =

8.

x =

16.

Sia il for sia il while possono essere terminati con il comando break:

-->a=0; for i=1:5:100, a=a+1; if i>10 then break,end;end

-->a

a =

3.

(Notare l'if interno tra i due ';').

CONDITIONALS:

Esistono due tipi di comandi condizionali: *if-then-else* e *select-case*.

-->x=1;

```
-->if x>0 then, y=-x,else, y=x,end
```

```
y =
```

```
-1.
```

```
-->x=-1;
```

```
-->if x>0 then, y=-x,else, y=x,end
```

```
y =
```

```
-1.
```

```
-->x=-1
```

```
x =
```

```
- 1.
```

```
-->select x, case 1,y=x+5,case -1,y=sqrt(x),end
```

```
y =
```

```
i
```

Tornando alle funzioni, è possibile definirle direttamente nell'ambiente SCILAB. E' però conveniente crearle in un file. Vediamo la struttura di una funzione e molti comandi che possono essere usati.

Una generica funzione deve avere il seguente formato:

```
function [y1,.....yn] = foo(x1,.....,xm)
```

```
....
```

```
....
```

```
endfunction
```

dove foo è il nome della funzione, gli xi sono gli ingressi alla funzione e gli yj sono le uscite dalla funzione. Per esempio, supponiamo di voler calcolare il fattoriale di k (k!). Scegliamo l'editor interno e scriviamo la funzione:

```
function [x]=fact(k)
```

```
    k=int(k)
```

```

    if (k<1 or k=0) then k=1, end
    x=1;
    for j=1:k,x=x*j;end

```

endfunction

Questa funzione la salviamo come fact.sci; poi, la chiamiamo da SCILAB con il comando exec o getf e solo adesso può essere utilizzata:

```
-->exists('fact') //Inizialmente non esiste (da come risultato 0)
```

ans =

0.

```
-->exec('.../Desktop/Scilab/fact.sci',-1); //Viene caricata
```

```
-->exists('fact') // Adesso esiste
```

ans =

1.

```
-->x=fact(5) //Voglio il fattoriale di 5
```

x =

120.

Avremmo potuto usare anche il comando getf(filename). Un file può contenere anche più funzioni.

Variabili locali e globali

Se una variabile in una funzione non è definita (e non è tra i parametri di ingresso), allora si prende il valore di una variabile che ha lo stesso nome nell'ambiente chiamante. Comunque, questa variabile rimane locale nel senso che modificandola con la funzione, non viene alterata a meno che è usato il comando *resume*. Le funzioni possono essere invocate con meno ingressi o uscite. Esempi:

```
function [y1,y2]=f(x1,x2)
```

```
    y1=x1+x2
```

```
    y2=x1-x2
```

endfunction

```
-->exec('c:/Docume~1/nicola/Desktop/Scilab/fact.sci',-1);
```

```
-->[y1,y2]=f(1,1)
```

y2 =

```

0.
y1 =
2.
-->f(1,1)
ans =
2.
-->f(1)
!--error 4
undefined variable : x2
at line 2 of function f
-->x2=1;

```

```

-->[y1,y2]=f(1)
y2 =
0.
y1 =
2.
-->f(1)
ans =
2.

```

Le variabili globali sono definiti dal comando `global`. Queste possono essere lette e modificate dentro le funzioni.

Comandi speciali

SCILAB ha molti comandi usati quasi esclusivamente nelle funzioni:

- ***argn***: ritorna il numero di parametri di ingresso e di uscita;
- ***error*** : usato per sospendere l'operazione di una funzion. Scrive un messaggio di errore e ritorna al livello precedente;
- ***warning***;
- ***pause***: sospende temporaneamente le operazioni di una funzione;
- ***break***: forza la fine di un ciclo;
- ***return* o *resume***: usati per ritornare all'ambiente chiamante e per passare le variabili locali.

Esempio di utilizzo di questi comandi.

Definiamo la funzione:

```
function [z]=foo(x,y)
[out,in]= argn(0)
if x==0 then,
    error('divisione per zero');
end,
scope = y/x;
pause,
z=sqrt(scope);
s=resume(scope);
endfunction
```

Dopo aver chiamato la funzione, la usiamo:

```
-->z=foo(0,1)
!--error 9999
divisione per zero
at line    4 of function foo          called by :
z=foo(0,1)
-->z=foo(2,1)
-1->resume
z =
    0.7071068
-->s
s =
    0.5
```

In questo esempio, la prima chiamata alla funzione passa un argomento che non può essere usato nel calcolo della funzione. La funzione indica la natura dell'errore. La seconda chiamata alla funzione sospende l'operazione dopo il calcolo di scope. Il prompt -1-> indica che l'ambiente corrente creato dal comando pause, è l'ambiente della funzione e non quello chiamante. Il controllo ritorna alla funzione con il comando *resume*.

Le operazioni di una funzione possono essere stoppati con i comandi *quit* o *abort*. Finalmente termina i suoi calcoli ritornando il valore di z. Inoltre è disponibile la variabile locale s.

Primitive base

Adesso, descriviamo in breve i più importanti aspetti dell'ambiente di SCILAB.

Quando SCILAB parte, si caricano un elevato numero di variabili e primitive. Il comando *who* mostra una lista di variabili accessibili. Invece, *who('local')* o *who('global')* mostrano una lista di variabili accessibili con memoria in doppia precisione.

Le variabili possono essere salvate in un file binario esterno con il comando *save*. Con *load* vengono caricate.

Il comando *clear* permette di cancellare tutto ciò che è presente nell'ambiente. Altrimenti, si devono indicare le variabili da cancellare.

save e *load* senza alcun argomento salvano e caricano tutte le variabili, librerie e funzioni nell'ambiente.

Per caricare le librerie di funzioni, si utilizza *lib*.

Input e Output

Altre funzioni importanti per i dati sono *read* e *write*. Funzionano in modo simile ai corrispondenti comandi presenti in Fortran. La sintassi di questi due comandi è la seguente:

```
-->x=[1 2 %pi; %e 3 4]
x =
! 1.      2.  3.1415927 !
! 2.7182818 3.  4.    !
-->write('x.dat',x)
-->clear x
-->xnew=read('x.dat',2,3)
xnew =
! 1.      2.  3.1415927 !
! 2.7182818 3.  4.    !
```

Da notare che *read* specifica il numero di righe e di colonne della matrice x.

Possono essere anche usate le funzioni C-like *mfscanf* e *mfprintf*:

```
-->x=[1 2 %pi; %e 3 4]
x =
! 1.      2.  3.1415927 !
! 2.7182818 3.  4.    !
-->fd=mopen('x_c.dat','w')
fd =
```

```

1.
-->mfprintf(fd,'%f %f %f\n',x)
-->fclose(fd)
ans =
    0.
-->clear x
-->fd=mopen('x_c.dat','r')
fd =
    1.
-->xnew(1,1:3)=mfscanf(fd,'%f %f %f\n')
xnew =
! 1.      2.  3.141593 !
! 2.7182818  3.  4.    !
-->xnew(2,1:3)=mfscanf(fd,'%f %f %f\n')
xnew =
! 1.      2.  3.141593 !
! 2.718282  3.  4.    !
-->fclose(fd)

```

Help

E' presente un help in linea molto ben fatto che sostituisce totalmente il manuale.

Funzioni più usate

Mostriamo le funzioni e le parole chiavi più usate:

- funzioni elementari: sum, prod, sqrt, diag, cos, max, round, sign, fft
- sorting: sort, gsort, find
- matrici particolari: zeros, eye, ones, matrix, empty
- algebra lineare: det, inv, gr, svd, bdiag, spec, schur
- polinomiali: poly, roots, coeff, horner, clean, freq
- sistemi lineari: syslin
- numeri casuali: rand
- esecuzione di un file: exec
- simboli di confronto: ==, >=, >, =, <, <=, &, |
- bottoni: x_choose, x_dialog, x_mdialog, addmenu

- programmazione: function, for, if, end, while, select, error, return, break, deff, warning
- debug: pause, return, abort
- caratteri stringa: string, part, evstr, execstr
- interpolazione e funzioni spline: splin, interp, interpln
- grafiche: plot, xset, driver, plot2d, plot3d, locate, xgrid, Graphics
- Ode: ode, dassl, dassrt, odedc
- ottimizzazione: optim, linpro, quapro, lmitool
- sistema dinamico: scicos
- routine C e Fortran: link, fort, addinter, intersci

Calcolo non lineare

SCILAB prevede primitive non lineari molto potenti per simulare e ottimizzare.

Simulazioni numeriche di sistemi di equazioni differenziali sono effettuate con la primitiva *ode*. Sistemi impliciti possono essere risolte con *dassl*. Questi comandi hanno altre possibilità e un numero opzionale di argomenti utili.

La funzione *optim* permette l'ottimizzazione delle funzioni non lineari.

Le funzioni di SCILAB o le routine di C e di Fortran possono essere usate come argomenti di qualche primitiva di alto livello come *ode*, *optim*, *dassl*,...Queste funzioni vengono chiamate funzioni argomento o funzioni esterne. Per esempio la funzione *costfunc* è un argomento di *optim*.

La sequenza che la chiama deve essere: [f, g, ind]=costfunc(x,ind) come imposto da *optim*.

Per problemi dove è importante il computo del tempo, conviene utilizzare subroutine di C o di Fortran.

Una volta che le subroutine sono scritte devono essere linkate in SCILAB, utilizzando il comando *link*.

Xwindow Dialog

E' conveniente aprire una finestra specifica per entrare interattivamente parametri nelle funzioni o per i demo. Ciò è possibile grazie alle funzioni *x_dialog*, *x_choose*, *x_mdialog*, *x_matrix* e *x_message*.

Grafica

E' possibile avere più finestre grafiche aperte contemporaneamente (ScilabGraphicx x), ma una sola sarà attiva. x rappresenta il numero della finestra attiva. Ovviamente l'esecuzione di un comando di

plottaggio automaticamente crea una finestra quando è necessario.

Ogni finestra grafica permette la rotazione con il mouse se il grafico è tridimensionale. Permette lo zoom se il grafico è bidimensionale. Il menu File permette tra l'altro di stampare o esportare la figura in vari formati (gif, bmp, Xfig, meta-file, postscript,...) o di copiarla su clipboard.

I comandi di grafica sono:

- driver: selezione un driver grafico (X11, Pos, Fig, GIF)

I prossimi tre comandi sono specifici per screen driver:

- xclear: libera una o più finestre grafiche, ma non ha influenza sul loro contesto;
- xbasr: libera una finestra e cancella il grafico memorizzato associato;
- xpause: una pausa in millisecondi;
- xselect: seleziona la finestra corrente;
- xclick: aspetta un click del mouse;
- xbasr: ridisegna il grafico;
- xdel: chiude una finestra;

I seguenti due comandi sono specifici per driver Postscript, Xfig, GIF:

- xinit: inizia una sessione;
- xend: termina la sessione grafica.

Parametri di plottaggio

Alcuni parametri grafici sono controllati con il comando *xset()*, che apre un pannello dove è possibile modificare i vari parametri direttamente con il mouse.

Esistono alcuni comandi che servono a manipolare i grafici:

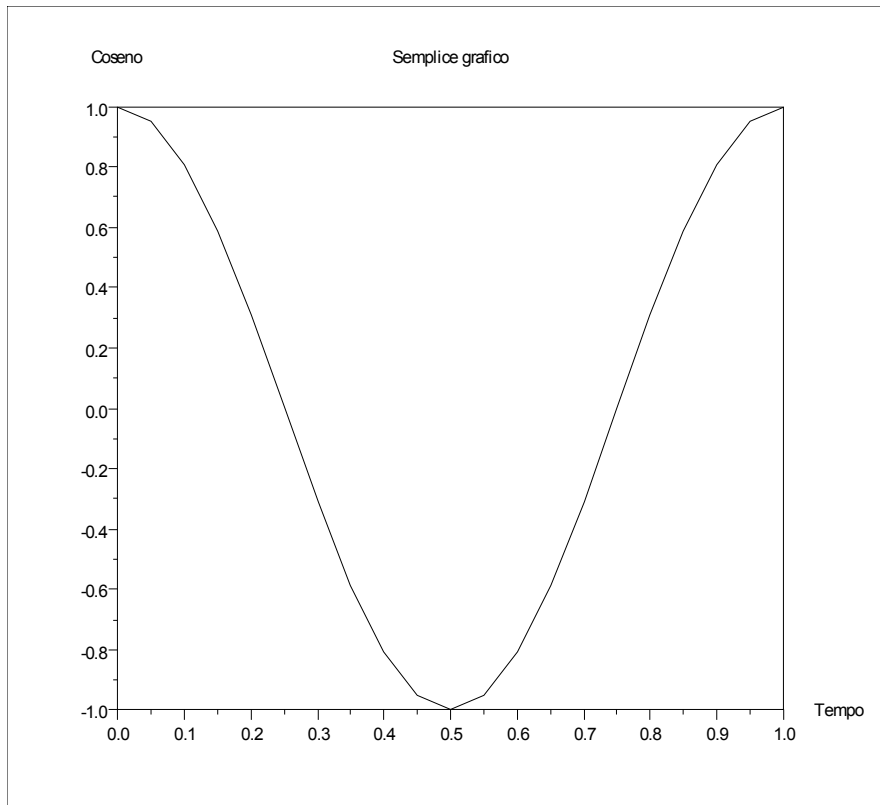
- isoview: scala isometrica senza cambiare le dimensioni della finestra;
- square: scala isometrica cambiando le dimensioni della finestra;
- scaling: trasformazione di un insieme di punti;
- rotate: rotazione; scaling e rotate eseguono rispettivamente una trasformazione e una rotazione geometrica di una matrice di due linee corrispondenti ai valori (x,y) di un insieme di punti;
- xgetech, xsetech: cambio di scala in una finestra.

2D Plotting

La più semplice funzione per plottare un grafico in bidimensionale è *plot(x,y,[xcap,ycap,caption])*, con il plottaggio della funzione $y(x)$ dove y e x sono due vettori. Se x manca, viene sostituito dal vettore $(1, \text{size}(y, '*'))$. Se y è una matrice vengono plottate tutte le righe. Sono presenti anche delle

opzioni. Le opzioni sono caratteri stringa o matrice stringa. Vediamo un esempio:

```
-->t=(0:0.05:1)';  
-->yt=cos(2*%pi*t);  
-->plot(t,yt,'Tempo','Coseno', 'Semplice grafico');
```



La generica funzione bidimensionale multipla è ***plot2d(x,y,[opzioni])***. Le opzioni possono essere:

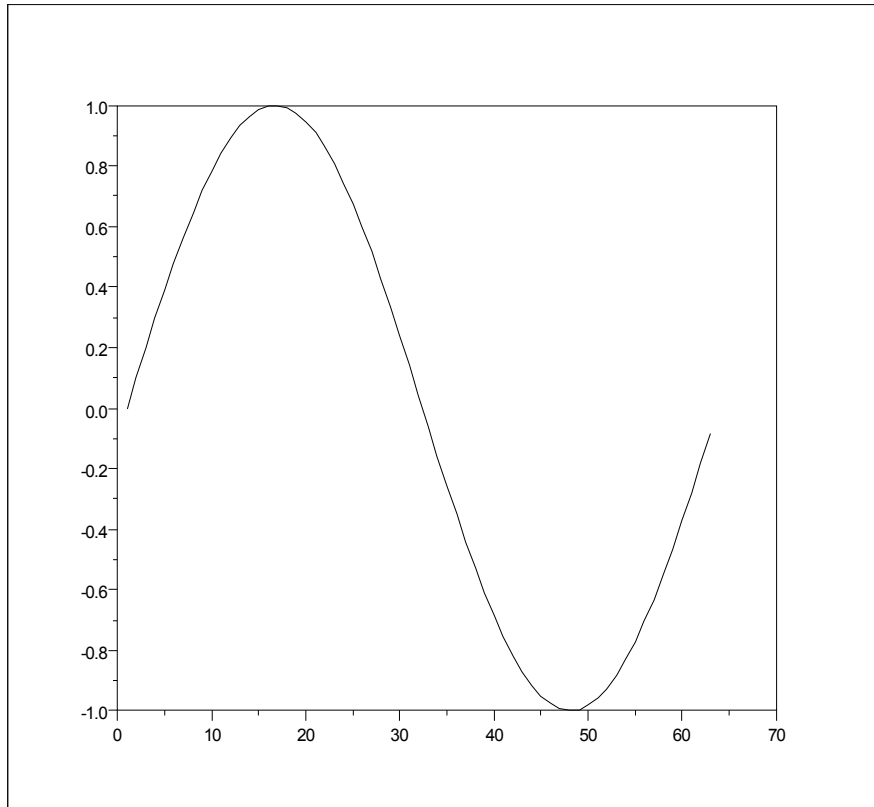
- **style**: setta lo stile di ogni curva;
- **rect**: setta il rettangolo dove disegnare la curva con questi valori:[xmin,ymin,xmax,ymax];
- **logflag**: setta la scala degli assi: lineare('n') e/o logaritmica('l');
- **frameflag**: controlla il calcolo del range delle coordinate attuali per i minimi valori richiesti. Va da 0 a 8;
- **axesflag**: specifica come gli assi sono disegnati. Va da 0 a 5
- **nax**: setta le etichette degli assi. Può essere usata solo quando axesflag=1;
- **leg**: setta la legenda delle curve.

Per default, i grafici successivi si sovrappongono. Per cancellare i grafici precedenti si usa **clf()**.

Esempio:

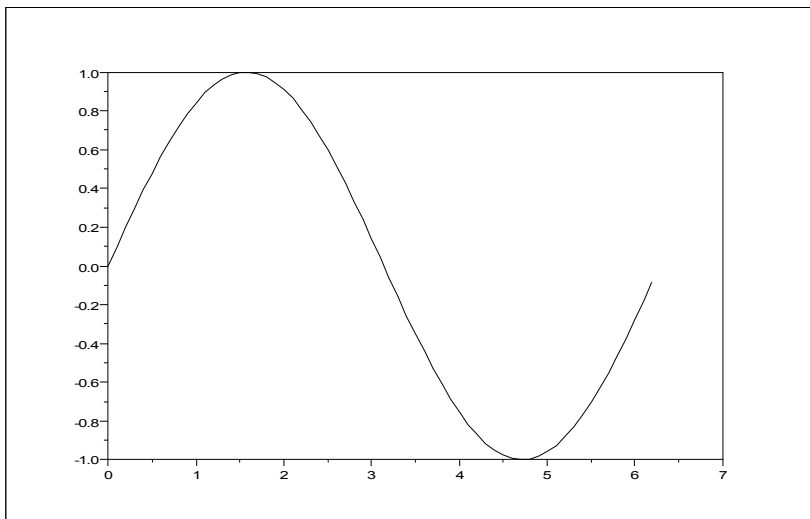
```
-->// x initialisation  
-->x=[0:0.1:2*%pi]';  
-->//simple plot
```

```
-->plot2d(sin(x))
```



```
-->clf() //cancella
```

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



```
-->clf() //cancella
```

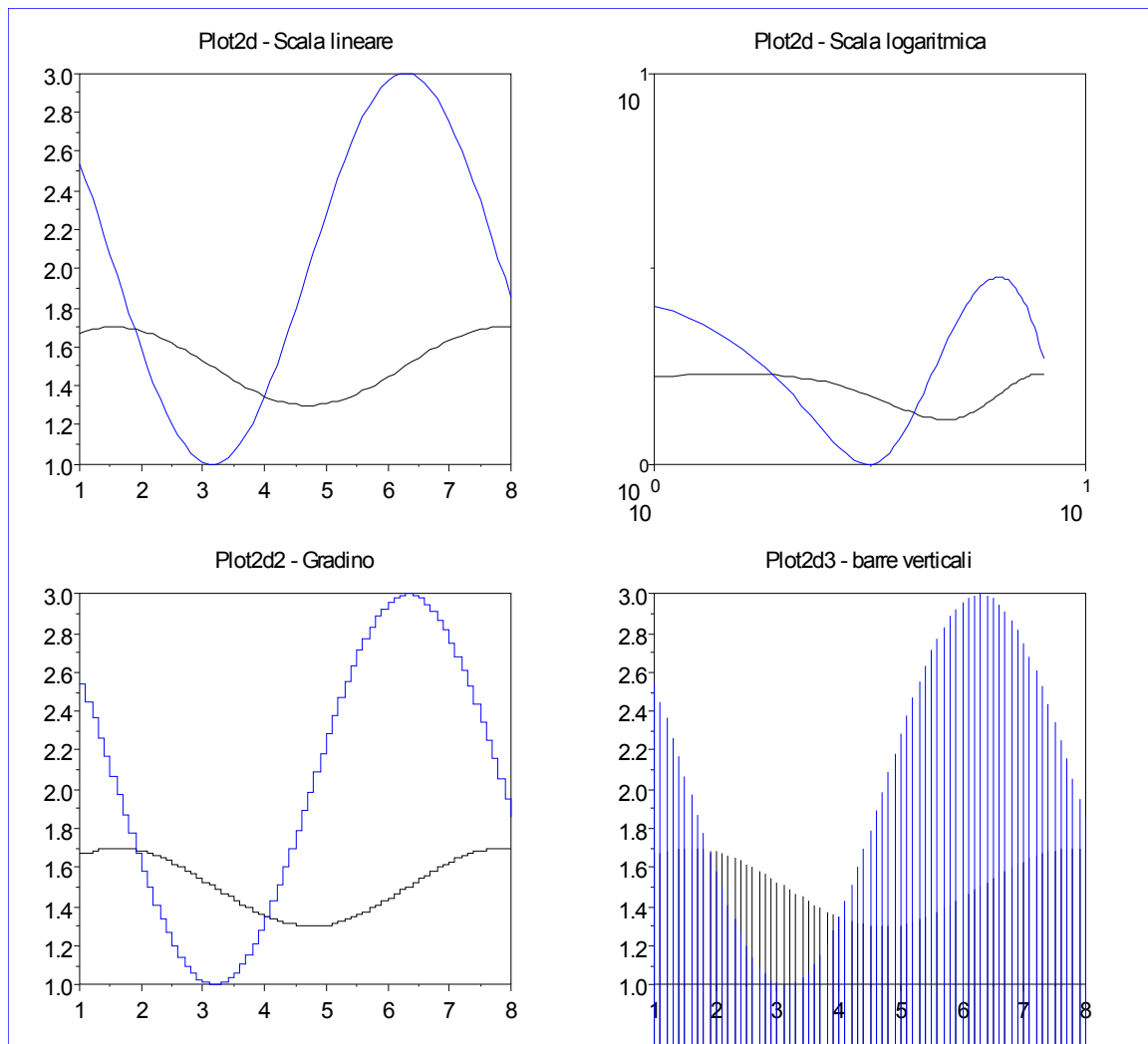
Esistono altre tre funzioni dello stello livello:

- plot2d2: le funzioni si suppongono essere costanti;

- plot2d3: le curve vengono plottate usando barre verticali;
- plot2d4: le curve vengono plottate usando frecce. Ciò è possibile quando si plottano soluzioni di un ODE nella fase.

Vediamo degli esempi semplici su quanto detto. Si utilizza *subplot* per avere più immagini con un solo plottaggio.

```
-->t=(1:0.1:8)';xset("font",2,3);
-->subplot(2,2,1)
-->plot2d([t t], [1.5+0.2*sin(t) 2+cos(t)]);
-->//
-->subplot(2,2,2)
-->plot2d(t, [1.5+0.2*sin(t) 2+cos(t)], logflag='ll');
-->xtitle('Plot2d - Scala logaritmica');
-->//
-->subplot(2,2,3)
-->plot2d2(t, [1.5+0.2*sin(t) 2+cos(t)]);
-->xtitle('Plot2d2 - Gradino');
-->//
-->subplot(2,2,4)
-->plot2d3(t, [1.5+0.2*sin(t) 2+cos(t)]);
-->xtitle('Plot2d3 - barre verticali');
```



Guardare l'help in linea per esempi vari sulle opzioni.

Presentazione

Vediamo insieme alcune opzioni interessanti sulla presentazione di una immagine.

- **xgrid:** aggiunge una griglia su un grafico bidimensionale. Il parametro chiamato è il numero del colore;
- **xtitle:** aggiunge un titolo sopra il grafico e i nomi degli assi su un grafico bidimensionale;
- **titlepage:** titolo in mezzo al grafico;

//Presentazione

```
-->x=-%pi:0.1:%pi;
```

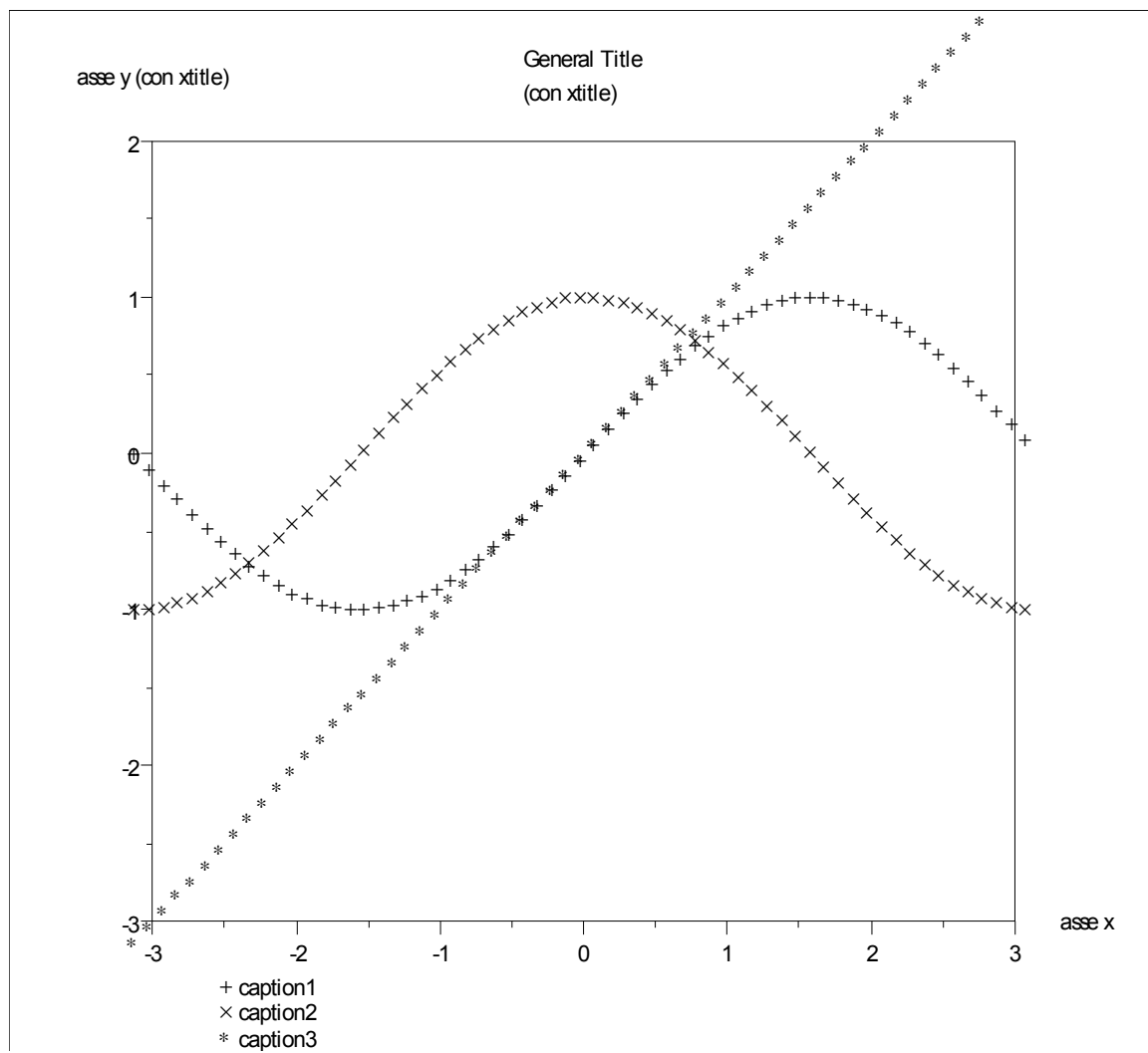
```
-->y1=sin(x); y2=cos(x); y3=x;
```

```
-->X=[x;x;x]; Y=[y1;y2;y3];
```

```

-->plot2d(X',Y',style=[-1 -2 -3],leg="caption1@caption2@caption3",rect=[-3, -3, 3, 2],nax=
[2,10,2,5]);
-->xtitle(["General Title";"(con xtitle)"],"asse x","asse y (con xtitle)");

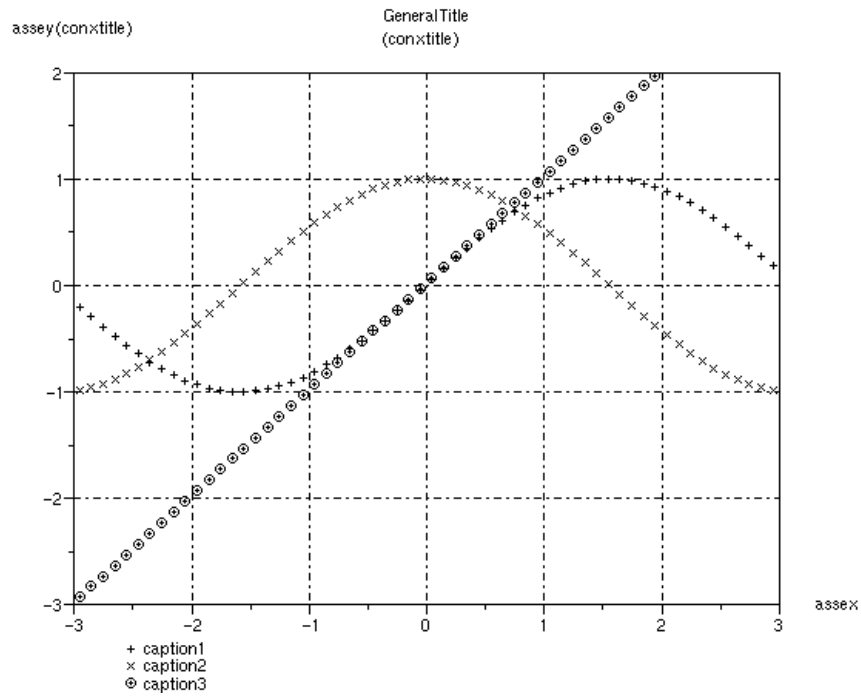
```



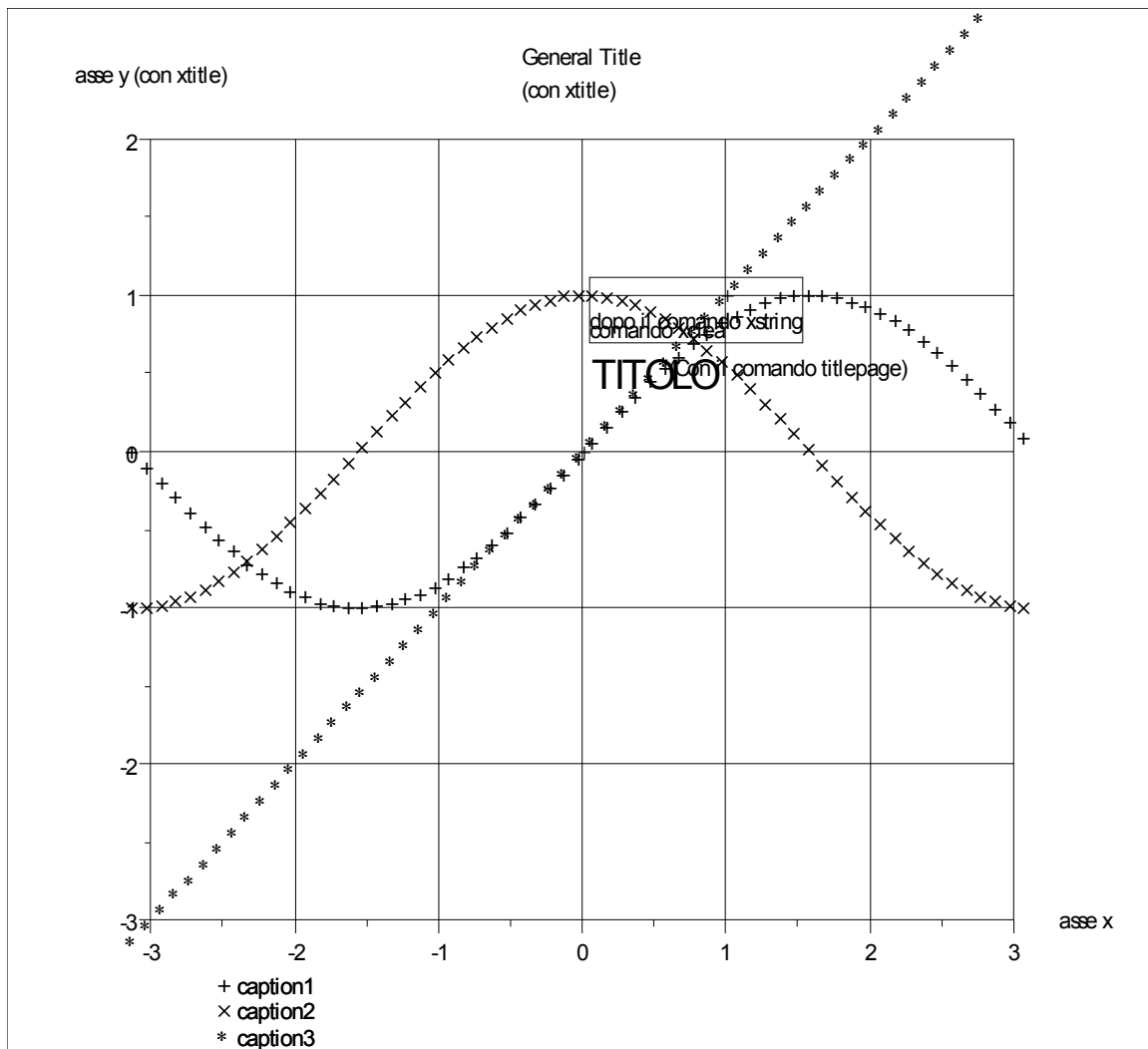
```

-->xgrid();

```



```
-->xclea(-2.7,1.5,1.5,1.5);
-->titlepage("TITOLO");
-->xstring(0.6,.45,"(Con il comando titlepage)");
-->xstring(0.05,.7,["dopo il comando xstring";"e il comando xclea"],0,1);
-->plot2d(X',Y',style=[-1 -2 -3],leg="caption1@caption2@caption3",rect=[-3, -3, 3, 2],nax=
[2,10,2,5]);
```



plotframe: frame grafico con scala e griglia.

Si è visto che è possibile controllare gli assi, scegliere la dimensione del rettangolo per plottare aggiungendo una griglia. Questa operazione può essere preparata in anticipo e poi usata in sequenza.

```
--> rect=[-%pi, -1, %pi, 1];
--> tics=[2,10,4,10];
--> plotframe(rect,tics,['t,%t'],['Plot con griglia','angolo']);
```

Altri comandi possono essere:

- **fplot2d**: disegno a due dimensioni di una curva descritta da una funzione;
- **grayplot**: plottaggio di una superficie con livelli di grigio;
- **fgrayplot**: lo stesso di prima per una superficie definita da una funzione.
- **errbar**: crea un plot con barre di errore.

Plottaggio di figure geometriche

- **xsegs**: disegna un insieme di segmenti non connessi;
- **xrect**: disegna un singolo rettangolo;
- **xfrect**: riempie un singolo rettangolo;
- **xrects**: riempie o disegna un insieme di rettangoli;
- **xpoly**: disegna una polilinea;
- **xpolys**: disegna un insieme di polilinee;
- **xfpoly**: riempie un poligono;
- **xfpolys**: riempie un insieme di poligoni;
- **xclea**: cancella un rettangolo su una finestra;
- **xarc**: disegna una ellisse;
- **xfarc**: riempie una ellisse;
- **xstring**: disegna una stringa o una matrice di stringhe.

Comandi per Controlli Automatici

- **bode**: disegna i diagrammi di Bode in fase ed in ampiezza;
- **nyquist**: disegna sul piano di Gauss una risposta in frequenza;
- **chart**: disegna la carta di Nichols;
- **plzr**: plotta poli-zero del sistema lineare;
- **black**: plotta il diagramma di Black per un sistema lineare;
- **evans**: plotta il luogo dei poli di un sistema lineare;
- **gainplot**: stesso di Bode ma plotta solo il guadagno.

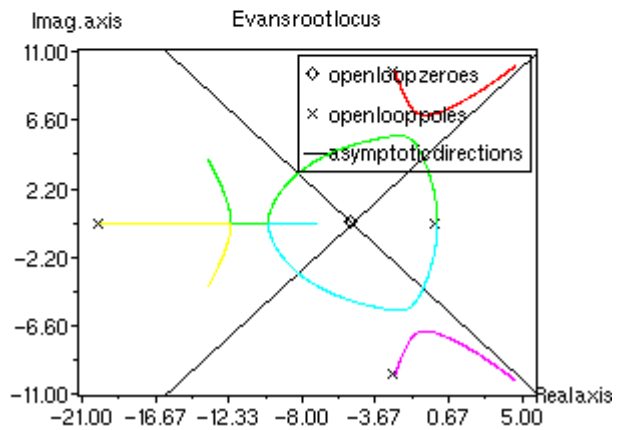
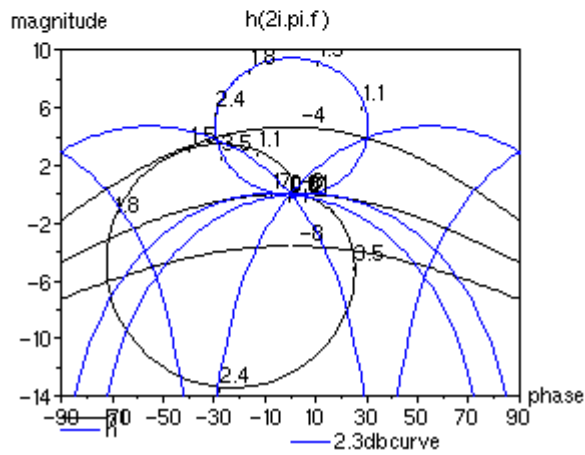
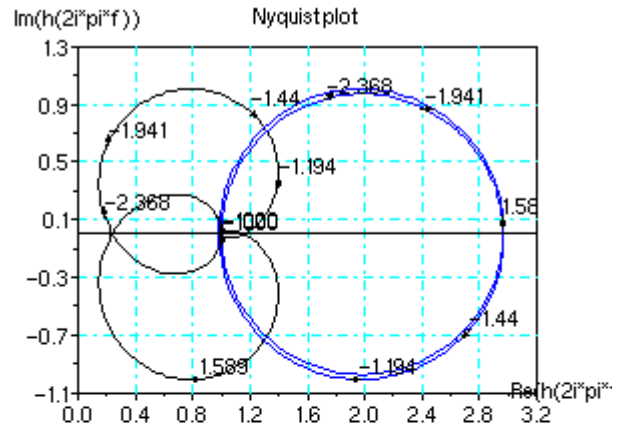
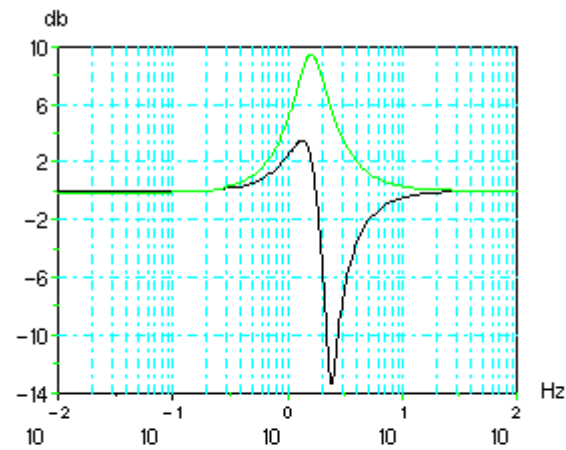
Esempio:

```
-->s=poly(0,'s');
-->h=syslin('c',(s^2+2*0.9*10*s+100)/(s^2+2*0.3*10.1*s+102.01));
-->h1=h*syslin('c',(s^2+2*0.1*15.1*s+228.01)/(s^2+2*0.9*15*s+225));
-->//bode
-->subplot(2,2,1)
-->gainplot([h1;h],0.01,100);
-->//nyquist
-->subplot(2,2,2)
-->nyquist([h1;h])
```

```

-->//chart e black
-->subplot(2,2,3)
-->black([h1;h],0.01,100,['h1';'h']);
-->chart ([-8 -6 -4],[80 120],list(1,0));

```



I grafici tridimensionali non servono per il nostro studio, pertanto vengono tralasciati.