Introduzione a SCILAB 5.3

Versione 2.3 Gennaio 2011

Gianluca Antonelli Stefano Chiaverini

Dipartimento di Automazione, Elettromagnetismo,
Ingegneria dell'Informazione e Matematica Industriale
Università degli Studi di Cassino
Via G. Di Biasio 43, 03043 Cassino (FR), Italy
antonelli@unicas.it
chiaverini@unicas.it
http://www.docente.unicas.it/gianluca_antonelli
http://webuser.unicas.it/chiaverini

Versioni

- 2.3 - Gennaio 2011
  - aggiornamento alla versione 5.3;
  - modifiche minori.
- 2.2 - Settembre 2010
  - aggiornamento alla versione 5.2;
  - aggiornamento ad Xcos 1.0;
  - aggiunta sezione calcolo simbolico;
  - aggiunti link esterni nel documento pdf;
  - modifiche minori.
- 2.1 - Settembre 2009
  - aggiunta sezione sulle stringhe;
  - aggiunta tabella funzioni scientifiche;
  - aggiunta sezione su GUI;
  - sillabazione in Italiano;
  - modifiche minori.
- 2.0 - Maggio 2009
  - aggiornamento alla versione 5.1;
  - aggiunti comandi grafica 3D;
  - aggiunta mappa colori;
  - aggiunta definizione struttura;
  - modifiche minori.
- 1.1 - Maggio 2008
  - documento ipertestuale;
  - aggiunto capitolo di confronto con Matlab;
  - aggiunti sezioni/comandi minori;
  - modifiche minori.
- 1.0 - Gennaio 2008

Indice

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Cos'è Scilab

Scilab è un programma per il calcolo numerico destinato ad applicazioni scientifiche ed ingegneristiche.

Scilab è *open source*, è disponibile dal 1994 gratuitamente nella versione binaria e con i codici sorgenti. È utilizzato in ambienti scientifici, universitari ed industriali. Può essere utilizzato per applicazioni non-commerciali e, entro certi limiti, per applicazioni commericali.

Scilab include centinaia di funzioni matematiche e la possibilità di utilizzare un proprio linguaggio di programmazione di alto livello, permette anche di utilizzare e far coesistere codice con funzioni scritte in diversi linguaggi quali il C, il C++ o Fortran. Le funzioni disponibili permettono operazioni quali:

- grafica 2-D e 3-D, animazioni
- algebra lineare, matrici sparse
- polinomi, funzioni razionali
- interpolazione, approssimazione
- simulazioni: solver ODE e DAE
- Xcos: simulatore grafico per sistemi dinamici (denominato Scicos nelle versioni precedenti)
- controllo classico e robusto, ottimizzazione LMI
- ottimizzazione
- teoria dei segnali
- Metanet: grafi e reti
- statistica
- interfaccia con Maple per generare codice Scilab
- interfaccia con Fortran, Tcl/Tk, C, C++, Java, LabVIEW

- un numero elevato di pacchetti scritti dagli utilizzatori stessi di Scilab

Scilab gira sotto la maggior parte dei sistemi Unix, GNU/Linux, MacOSX e Windows.

1.2 Ottenere informazioni ed una copia di Scilab

La pagina principale di Scilab è raggiungibile sul sito `http://www.scilab.org`.

Dalla URL `http://www.scilab.org/products/scilab/download` è possibile scaricare la più recente versione stabile del programma (eseguibile e/o codice sorgente) per i sistemi operativi GNU/Linux, Windows e MacOSX. Nella stessa pagina sono anche presenti i collegamenti alle URL che rendono disponibili le versioni in via di sviluppo ed alle versioni precedenti nonché alla documentazione. Sono inoltre presenti i link alla documentazione (sia gratuita che a pagamento), ai toolbox aggiuntivi, alla pagina dedicata a Xcos, alla pagina dedicata a LabVIEW ed al newsgroup `comp.soft-sys.math.scilab`. Un'altra pagine utile con la documentazione è `http://www.scilab.org/support/documentation`.

1.3 Un pò di storia

Il progetto Scilab nasce agli inizi del 1980 presso INRIA, in Francia, ispirandosi proprio al progetto del MIT, Matlab, allora di pubblico dominio. I primi anni, ancora con un nome differente, viene distribuito commercialmente, agli inizi degli anni 90 viene decisa la natura open source ed il nome: Scilab.

Da quel momento il software viene sviluppato con regolarità e viene sviluppato anche il simulatore grafico per sistemi dinamici: Scicos, simile al Simulink del Matlab nonchè gran parte dei comandi di ottimizzazione e di grafica.

Una prima *divisione* nello sviluppo del software avviene nel 2005, con l'introduzione della versione basata su Java, cioè dalla release 5 in poi. Viene deciso di mantenere una versione fortemente basata sulla release 4 e chiamata ScicosLab. In particolare, ScicosLab mantiene la versione di Scicos mentre Scilab 5.x sviluppa Xcos.

1.4 Scilab e gli altri

Esistono diversi programmi per il calcolo numerico quali, ad esempio,

- ScicosLab (`http://www.scicoslab.org`);
- Matlab (`http://www.mathworks.com`);
- Octave (`http://www.octave.org`);
- Freemat (`http://freemat.sourceforge.net`).

In molti newsgroup, e nello stesso sito Scilab, è possibile trovare commenti, confronti e test numerici sulle prestazioni di questi programmi. Il riferimento nell'ambito del calcolo numerico

è il programma commerciale Matlab prodotto dalla Mathworks. Scilab mette anche a disposizione delle funzioni per la traduzione automatica del codice Matlab in codice Scilab ed una corrispondenza di alcune delle funzioni principali. Ulteriori dettagli sono forniti nel Capitolo 6.

È da notare come Scilab, Octave e Freemat facciano già parte dei repository di molte distribuzioni Linux, seppure non nelle ultime versioni (ma serve avere l'ultima versione?).

Un'alternativa di più ampio respiro dal punto di vista informatico è data poi da Python (http://www.python.org).

1.5 Scilab è veramente gratis?

Scilab è gratuito. Non è distribuito con una licensa GPL (http://www.gnu.org/licenses) ma con una licenza CeCILL (http://www.cecill.info/index.en.html) che adatta la licenza GPL alla legislazione Francese. È un programma open source la cui licenza è disponibile all'indirizzo http://www.scilab.org/legal/license.html.

Capitolo 2

Programmare in Scilab

2.1 Installare, avviare e chiudere Scilab

L'istallazione di Scilab 5.3 per i sistemi operativi Linux e Windows è assolutamente banale, si rimanda per i dettagli alle istruzioni contenute nel sito ufficiale. Sotto Windows, per avviare Scilab è sufficiente selezionare l'icona corrispondente dal menu grafico o cliccare due volte sull'icona eventualmente creata sul Desktop. Sotto Linux si può anche avviare da riga di comando `./scilab` aprendo la shell/konsole nella cartella opportuna oppure mettendola nel percorso tramite la variabile `$PATH`. È ovviamente disponibile anche l'icona corrispondente dal menu grafico o il lancio tramite le applicazioni Krunner, Gnome-Do, ecc.

La figura 2.1 riporta una schermata iniziale di Scilab sotto Linux; il colore ed il font possono essere personalizzati (si veda la sezione 2.20) ed all'avvio si potrebbe richiamare uno script che, fra le altre cose, può aggiungere delle funzioni d'utente a quelle disponibili in Scilab (si veda la sezione 2.21).

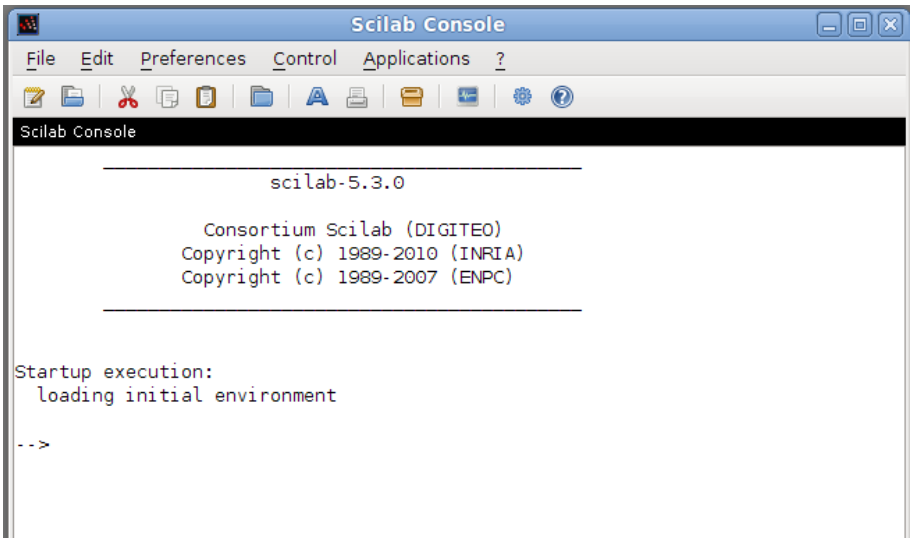


Figura 2.1: Schermata iniziale di Scilab

Per chiudere l'applicazione è possibile utilizzare il corrispondente comando dal menu `file`, o

digitare `quit` da riga di comando oppure usare i tasti `[ALT+F4]` (sia sotto Windows che sotto molte distribuzioni Linux).

Esiste anche la possibilità di lanciare dei programmi scritti in Scilab senza chiamare interfaccia grafica. È una modalità per utenti esperti che permette di sfruttare le risorse Scilab (senza ingressi nè uscite grafiche) anche all'interno di script chiamati da shell Linux. Per farlo si deve usare opportunamente il comando `scilab` i cui dettagli si possono ottenere dall'aiuto in linea.

2.1.1 La cartella di lavoro

All'avvio, Scilab lavora in una cartella predefinita che spesso è la cartella `bin` del percorso di installazione oppure la `$HOME` nelle installazioni Linux. Per verificarlo è sufficiente digitare il comando `pwd`. Tutte le operazioni che prevedono una lettura o scrittura di file d'utente sul disco rigido come, ad esempio, il salvataggio dati o l'esecuzione di script d'utente, verranno eseguite nella cartella corrente. È sconsigliabile mischiare i propri file con quelli del programma e quindi si suggerisce di modificare la cartella di lavoro con un'opportuna cartella d'utente come, ad esempio

```
-->cd /home/myname/myscilabdir
```

se il nome del percorso contiene degli spazi la sintassi da usare è:

```
-->cd('/home/myname/my␣scilab␣dir');
```

La stessa operazione può essere eseguita graficamente utilizzando l'apposita icona presente dalla versione 5.1 sia sotto Linux che Windows, nella versione precedente Linux ne era sprovvista.

2.2 Aiuto

Ci sono diversi modi per conoscere la sintassi e la semantica dei comandi Scilab oppure per verificare l'esistenza stessa di un determinato comando. Oltre all'utilizzo di manuali, è possibile avere informazioni *in linea* tramite il comando `help` che apre una nuova finestra come mostrato in figura 2.2.

Dalla finestra dell'help è possibile cercare i comandi, divisi per tipologia, oppure andare direttamente alla descrizione del singolo comando scrivendone il nome nell'apposito spazio. La stessa cosa può essere ottenuta da riga di comando scrivendo

```
-->help nomecomando
```

È utile anche il comando `apropos` che permette una ricerca di tutti i comandi in cui compaia un determinata parola chiave. Ad esempio:

```
-->apropos exponential
```

fornisce una finestra del tipo 2.3, in cui i comandi sono elencati in ordine decrescente per numero di occorrenze della parola cercata.

In rete risulta comodo anche l'help in linea che racchiude in una sola pagina tutti i comandi Scilab all'indirizzo http://www.scilab.org/product/man.

Un *dizionario* di semplice funzioni Matlab con l'equivalente Scilab è disponibile all'indirizzo

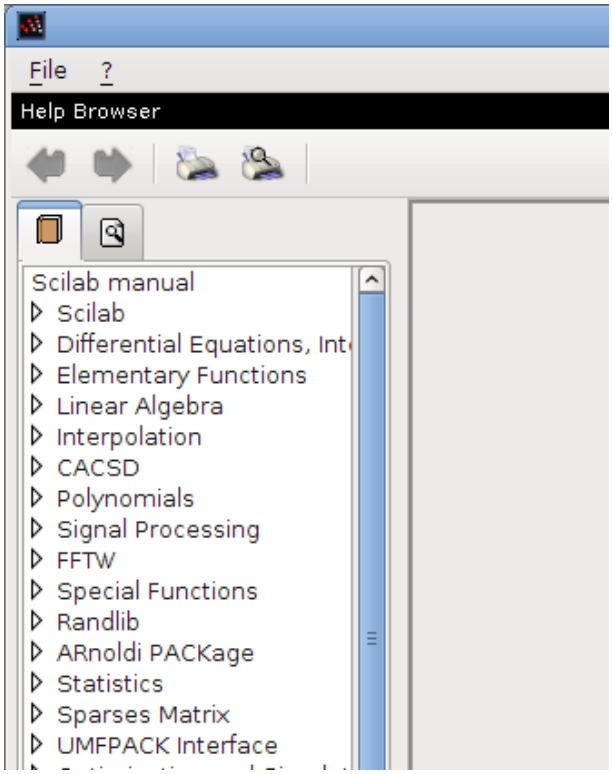


Figura 2.2: Finestra iniziale del comando `help`

http://www.scilab.org/product/dic-mat-sci/M2SCI_doc.htm

e verrà trattata nel Capitolo 6.

2.3 Variabili e spazio di lavoro

Scilab, come qualsiasi altro programma di calcolo numerico, può essere utilizzato come una calcolatrice semplicemente scrivendo sulla riga di comando le operazioni da eseguire. A parte questo utilizzo elementare sarà sempre necessario definire ed utilizzare delle variabili, delle funzioni o dei programmi (script).

In generale, ogni istruzione Scilab è una riga del tipo

```
-->nomevariabile = istruzione
```

in cui l'uscita dell'istruzione viene assegnata alla variabile `nomevariabile`. La variabile `ans` viene automaticamente utilizzata da Scilab quando si scrive un'istruzione senza fornire una variabile in uscita.

Ogni variabile va individuata con una stringa alfanumerica che non può cominciare con un numero e non può contenere caratteri speciali. Quindi `nome`, `nome01`, `nome01a` sono validi nomi di variabili mentre `01nome` non lo è. Il nome delle variabili è *case sensitive*, vale a dire maiuscole e minuscole sono lettere diverse.

A differenza di molti linguaggi come, ad esempio, il C, ed in comune con programmi quali

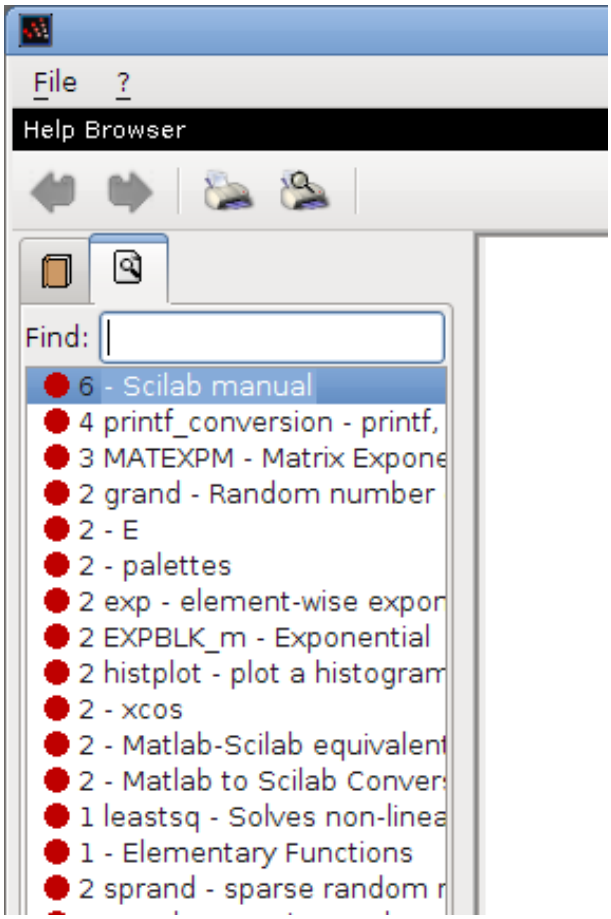


Figura 2.3: Risultato del comando `apropos exponential`

Matlab, Scilab non richiede che sia definito il tipo o la dimensione di una variabile prima del suo utilizzo.

Le variabili sono visibili nello spazio di lavoro, ottenibile con il comando `who`.

Per ottenere solo le variabili definite dall'utente è possible utilizzare il comando `who_user`.

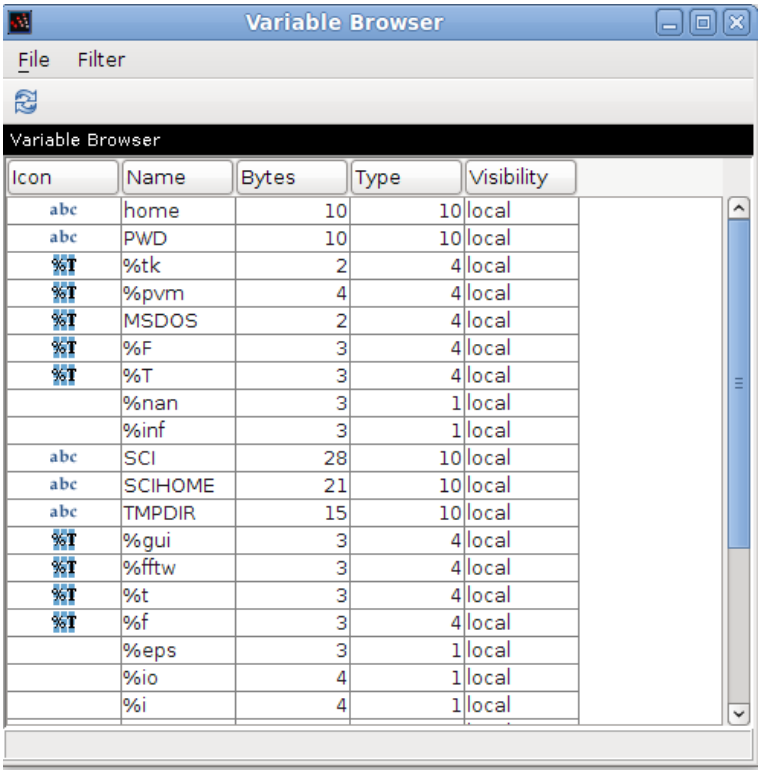
Per visualizzare ed editare le variabili è anche disponibile un'interfaccia grafica (mostrata in figura 2.4) che si apre con il comando `browsevar`. Solo sotto Windows si apre anche selezionando il menu `Applications` e `Browser Variables`.

Si noti come, in Scilab, le funzioni e le librerie di funzioni siano delle variabili e quindi compaiano con il comando `who`. Come definire od utilizzare programmi (script) e funzioni è oggetto delle Sezioni, rispettivamente 2.14 e 2.15.

Una variabile è aggiunta nello spazio di lavoro al momento del suo utilizzo, è sempre possibile cancellarla con il comando

```
-->clear nomevariabile
```

che può essere usato anche per cancellare tutte le variabili d'utente utilizzando semplicemente `clear`. Si noti, però, che l'utilizzo del comando `clear` cancella anche le funzioni (si veda Sezione 2.15) ed alcune variabili d'ambiente Scilab; è quindi un comando da utilizzare con cautela.



Icon	Name	Bytes	Type	Visibility
abc	home	10	10	local
abc	PWD	10	10	local
%T	%tk	2	4	local
%T	%pvm	4	4	local
%T	MSDOS	2	4	local
%T	%F	3	4	local
%T	%T	3	4	local
	%nan	3	1	local
	%inf	3	1	local
abc	SCI	28	10	local
abc	SCIHOME	21	10	local
abc	TMPDIR	15	10	local
%T	%gui	3	4	local
%T	%fftw	3	4	local
%T	%t	3	4	local
%T	%f	3	4	local
	%eps	3	1	local
	%io	4	1	local
	%i	4	1	local

Figura 2.4: Finestra per visualizzare lo spazio di lavoro

2.3.1 Le costanti di Scilab

Alcune costanti predefinite sono sempre disponibili nello spazio di lavoro. Il loro nome comincia con il simbolo %. Quindi, scrivendo, ad esempio:

```
-->%pi
 %pi   =

    3.1415927
```

si ottiene la costante π, il rapporto fra perimetro e diametro di un cerchio.

Le costanti predefinite sono *write protected*, non possono essere modificate nè salvate. La tabella 2.1 riporta l'elenco di alcune costanti predefinite in Scilab. Ce ne sono altre che riguardano la configurazione del programma e possono essere ignorate in questo contesto.

Il simbolo percentuale ha anche un ulteriore utilizzo come direttiva di formattazione nei comandi di lettura/scrittura su file come riportato nella sezione 2.18.

2.3.2 Tipi di variabili

In Scilab è possibile assegnare un valore ad una variabile senza averla prima definita, cosa necessaria, ad esempio, in C o in Fortran. L'interprete si occupa poi di assegnare il tipo corretto alla variabile secondo l'utilizzo contemplato nell'istruzione corrispondente. Questa caratteristica

Tabella 2.1: Costanti predefinite

`%i`	$i = \sqrt{-1}$	unità immaginaria
`%pi`	$\pi = 3.1415927\ldots$	pi greco
`%e`	$e = 2.718281\ldots$	numero di Nepero
`%eps`	$\varepsilon = 2.2 \cdot 10^{-16}$	precisione (macchina dipendente)
`%inf`		infinito
`%nan`		NotANumber
`%s`	s	variabile polinomiale
`%z`	z	variabile polinomiale
`%t %T`	true	variabile booleana
`%f %F`	false	variabile booleana

semplifica notevolmente l'operazione di scrittura del codice ma aumenta le possibilità di errore per distrazione.

Il tipo di dato più comune è una matrice (o vettore, o scalare) di numeri reali o complessi, per Scilab questo tipo di dato è un `constant` esattamente come quasi tutte le costanti predefinite.

Altri tipi di dati sono le stringhe di caratteri, per definire le quali è necessario racchiudere i caratteri fra apici o virgolette, indifferentemente:

```
-->str1='ciao'
 str1  =

 ciao

-->str2="ciao"
 str2  =

 ciao
```

le due variabili di tipo stringa, `str1` e `str2` sono uguali. Per inserire un apice(virgoletta) nella variabile è necessario ripeterla due volte. Informazioni addizionali sul tipo stringa sono riportate nella sezione 2.11.

Un tipo di dato utile nella programmazione è quello boleano, `boolean`, che in Scilab può assumere i due valori `%T` (true, vero) e `%F` (false, falso).

Il tipo di dato polinomio si definisce con il comando `poly`, un metodo semplice per definire polinomi a partire dalla conoscenza dei coefficienti è:

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

-->miopol = 3*s^2-s+2
 miopol  =

                2
    2 - s + 3s
```

Su variabili definite come tipo polinomio è possibile eseguire una serie di operazioni come precisato nella Sezione 2.9. In particolare è possibile lavorare sui rapporti di polinomi, ossia sulle funzioni razionali fratte, di tipo quindi `rational`, introdotte nella sezione 2.10, che as-

sumono notevole importanza nell'ambito dei sistemi dinamici e saranno ampiamente utilizzate nel Capitolo 4.

Per verificare il tipo di una variabile sono disponibili i comandi `type` e `typeof` che forniscono in uscita il tipo della varibile in formato numerico e stringa, rispettivamente. L'elenco seguente riporta l'elenco delle uscite che si possono ottenere con il comando `typeof`:

```
constant
polynomial
function
handle
string
boolean
list
rational
state-space
sparse
boolean sparse
hypermat
st
ce
fptr
pointer
size implicit
library
```

2.4 Formato di rappresentazione di un numero

È possibile scegliere il formato con cui visualizzare il valore dei numeri tramite il comando `format` che ammette la sintassi:

```
format([type],[long])
format()
```

in cui entrambi gli ingressi sono opzionali, l'ingresso `type` è una variabile di tipo stringa ed ammette solo due valori: `'v'`, per un formato variable e `'e'`, per il formato esponenziale. L'ingresso `long` rappresenta il numero di digit richiesti. Il valore di default è `format('v',10)`. L'utilizzo di `format()`, senza ingressi, fornisce il formato attualmente in uso. Il codice che segue mostra i possibili utilizzi del comando:

```
-->format()
 ans  =

    1.    10.

-->x=0.123456789;

-->format('v',10);x
 x  =

    0.1234568
```

```
-->format('e');x
 x  =

    1.235D-01

-->format('e',5);x
 x  =

    1.2D-01
```

2.5 Funzioni scientifiche

Scilab consente il calcolo numerico di decine di funzioni elmentari ed avanzate, un elenco completo delle funzioni elementari è ottenibile dall'aiuto in linea alla voce *Elementary Functions*; le funzioni avanzate sono ottenibili alla voce *Special Functions*. La tabella 2.2 riporta le principali funzioni elementari.

2.6 Manipolare scalari, vettori e matrici

Il modo più semplice per inserire una matrice è quello di scriverlo da riga di comando:

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

in questo modo si è inserito nello spazio di lavoro una matrice `3x3` di numeri reali. Per verificarlo è sufficiente utilizzare uno dei comandi `who` oppure digitare il nome della variable dal riga di comando:

```
-->A
 A  =

    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
```

Le seguenti convenzioni sono utilizzate per inserire una matrice:

- gli elementi della stessa riga sono separati da una virgola o da uno spazio
- ogni riga, tranne l'ultima deve termine con il simbolo punto e virgola `;`
- gli elementi della matrice devono essere racchiusi fra parentesi quadre `[]`

È quindi possibile inserire la stessa matrice $\boldsymbol{A}$ anche con la sintassi seguente:

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

Un'altra possibilità è quella di inserire la prima riga della matrice

```
A = [1,2,3
```

Tabella 2.2: Principali funzioni elementari

`abs`	absolute value, magnitude
`acos`	element wise cosine inverse
`acosh`	hyperbolic cosine inverse
`acot`	computes the element-wise inverse cotangeant of the argument
`acoth`	element wise hyperbolic cotangeant inverse
`acsc`	computes the element-wise inverse cosecant of the argument
`acsch`	computes the element-wise inverse hyperbolic cosecant of the argument
`amell`	Jacobi's am function
`asec`	computes the element-wise inverse secant of the argument
`asech`	computes the element-wise inverse hyperbolic secant of the argument
`asin`	sine inverse
`asinh`	hyperbolic sine inverse
`atan`	2-quadrant and 4-quadrant inverse tangent
`atanh`	hyperbolic tangent inverse
`cos`	cosine function
`cosh`	hyperbolic cosine
`cotd`	cotangent
`cotg`	cotangent
`coth`	hyperbolic cotangent
`exp`	element-wise exponential
`log`	natural logarithm
`log10`	logarithm
`log1p`	computes with accuracy the natural logarithm of its argument added by one
`log2`	base 2 logarithm
`sec`	Compute the element-wise secant of the argument
`sech`	Compute the element-wise hyperbolic secant of the argument
`sign`	sign function
`signm`	matrix sign function
`sin`	sine function
`sinc`	sinc function
`sinh`	hyperbolic sine
`sqrt`	square root
`tan`	tangent
`tanh`	hyperbolic tangent

e premere `ENTER` per poi inserire le altre righe. Scilab termina l'istruzione solo quando è chiusa la parentesi quadra:

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

-->4  5  6

-->7  8  9];
```

Il simbolo punto e virgola al termine di un'istruzione indica che l'utente non vuole visualizzare il risultato sullo schermo.

Un'istruzione molto lunga può essere divisa in due utilizzando tre punti alla fine della riga, scrivendo[1]

```
-->a = [1,2,3 ... [ENTER]
```

Scilab ritorna alla riga di comando ma aspetta la fine dell'istruzione come se fosse sulla riga precedente, quindi terminando con, per esempio:

```
-->4,5,6] [ENTER]
```

si ottiene un vettore $\boldsymbol{a}$ di dimensioni 1×6:

```
 a   =

    1.    2.    3.    4.    5.    6.
```

È possibile notare come vettori e scalari non siano che casi particolari di matrici. Nel caso di uno scalare è possibile evitare di inserire le parentesi quadre e la corrispondente variabile si definisce semplicemente come:

```
-->g=9.81
 g   =

    9.81
```

Per inserire un valore complesso si utilizza la costante `%i`. Quindi lo scalare complesso $1 - 3i$ si inserisce con la sintassi:

```
-->a=1-3*%i
 a   =

    1. - 3.i
```

Alcune matrici tipo

In Scilab è sono disponibili alcune matrici ricorrenti, quali la matrice Identità, la matrice di elementi nulli o tutti unitari o la matrice di numeri casuali. I comandi corrispondendi sono `eye(n,m)`, `zeros(n,m)`, `ones(n,m)` dove n rappresenta il numero di righe ed m di colonne dell'uscita. Ad esempio:

[1] Con il simbolo `[ENTER]` si indicherà la pressione del tasto corrispondente, indicato anche con `INVIO` o `CARRIAGE RETURN` (`CR`) in alcune tastiere.

Tabella 2.3: Matrici speciali

`eye`	matrice Identità
`zeros`	matrice di zeri
`ones`	matrice di tutti elementi unitari
`companion`	matrice compagna
`hank`	matrice di Hankel
`toeplitz`	matrice di Toeplitz

```
-->I=eye(3,3)
 I  =

    1.    0.    0.

    0.    1.    0.

    0.    0.    1.
```

è anche possibile utilizzare come ingresso una matrice, in questo caso Scilab fornisce in uscita la matrice desiderata con le dimensioni di quella fornita in ingresso:

```
-->O=zeros(I)
 O  =

    0.    0.    0.

    0.    0.    0.

    0.    0.    0.
```

Sulla generazione di matrici di numeri casuali, ottenibile tramite i comandi `rand(), grand()` si rimanda ad approfondimenti nel help in linea di Scilab.

Esistono anche comandi per la generazioni di altre matrici *speciali*, la tabella 2.3 ne riporta alcuni.

Definire vettori incrementali

Può essere utile definire un vettore in cui gli elementi partendo da 1 si incrementano di un certo valore. Con la sintassi:

```
-->x=1:2:10
 x  =

    1.    3.    5.    7.    9.
```

Scilab costruisce un vettore che parte da 1 ed il cui elemento successivo è pari al precedente incrementato di 2 fino all'elemento strettamente minore di 10. Per incrementi unitari è sufficiente scrivere:

```
-->x=1:4
 x  =

    1.    2.    3.    4.
```

In alternativa è possibile fornire valore iniziale, finale e numero di elementi del vettore risultante tramite il comando `linspace`:

```
-->x=linspace(0,1,6)
 x  =

    0.    0.2    0.4    0.6    0.8    1.
```

Come accedere agli elementi di una matrice

Se nello spazio di lavoro è definita una matrice $\boldsymbol{A}$ di dimensioni 3×3:

```
-->A
 A  =

    1.    2.    3.

    4.    5.    6.

    7.    8.    9.
```

può essere necessario accedere ad un solo elemento di questa matrice o ad un sottoinsieme della stessa. L'istruzione

```
-->b=A(3,2)
 b  =

    8.
```

assegna alla variabile b il valore dell'elemento di riga 3 e colonna 2 della matrice $\boldsymbol{A}$. In generale è possibile estrarre gli elementi desiderati dalla matrice $\boldsymbol{A}$, con la sintassi:

```
-->b=A(3,2:3)
 b  =

    8.    9.
```

la variabile $\boldsymbol{b}$ è ora un vettore 1×2 con gli elementi di riga 3 e quelli dalla colonna 2 alla colonna 3. Generalizzando, è sufficiente separare fra virgole gli indici degli elementi da estrarre ottenendo, nel caso più generale, un'istruzione del tipo:

```
-->B=A([1 3],2:3)
 B  =

    2.    3.

    8.    9.
```

in cui l'uscita è una matrice 2×2 in cui compaiono gli elementi delle righe 1 e 3 e quelli delle colonne 2 e 3.

Per accedere a tutti gli elementi di una riga(colonna), è sufficiente utilizzare il simbolo `:` come indice:

```
-->B=A(:,2:3)
 B  =

    2.    3.

    5.    6.

    8.    9.
```

in questo modo tutte le righe di $\boldsymbol{A}$ sono selezionate e l'uscita è una matrice 3×2.

Poichè un vettore è una matrice con una sola riga/colonna è possibile accedere ad un suo elemento indicando come 1 l'indice per l'unica riga/colonna oppure omettendo completamente tale indice.

```
-->b=[1 2 3 4]; c = b(1,2), d = b(2)
 c  =

    2.

 d  =

    2.
```

È anche possibile estrarre la sola diagonale di una matrice con il comando `diag`:

```
-->b=diag(A)
 b  =

    1.

    5.

    9.
```

Lo stesso comando può essere utilizzato per costruire una matrice diagonale partendo dal solo vettore diagonale:

```
-->C=diag(b)
 C  =

    1.    0.    0.

    0.    5.    0.

    0.    0.    9.
```

Si noti come l'indice di un vettore o di una matrice, in Scilab, parte dall'unità, quindi 1 è il primo elemento del vettore. Questa convenzione è la stessa del Matlab ma diversa dal C, in cui il primo elemento di un vettore si individua con l'indice 0.

2.6.1 Matrici multidimensionali

l'estensione al caso multidimensionale è abbastanza intuitiva, è sufficiente aggiungere degli indici opportuni per definire e gestire matrici a più di 2 dimensioni. Si definisca, ad esempio, una matrice 2×2:

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

    1.    2.
    3.    4.
```

è possibile definire la *terza* dimensione semplicemente come:

```
-->A(:,:,2) = [5 6; 7 8]
 A   =

(:,:,1)

    1.    2.
    3.    4.
(:,:,2)

    5.    6.
    7.    8.
```

e poi accedere agli elementi estendendo semplicemente la sintassi mostrata per il caso bidimensionale.

2.7 Operazioni matriciali

Scilab fornisce una libreria di funzioni per il calcolo matriciale, tutte le operazioni fra matrici sono permesse in Scilab con una sintassi abbastanza intuitiva.

Somma fra matrici

Somma di due matrici di eguali dimensioni:

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

    1.    2.    3.

    4.    5.    6.

-->A+A
```

```
 ans  =

    2.    4.    6.

    8.    10.   12.
```

Trasposta di una matrice

La trasposta di una matrice si ottiene con il simbolo apice ‘:

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

    1.    2.    3.

    4.    5.    6.

-->A‘
 ans  =

    1.    4.

    2.    5.

    3.    6.
```

Se la matrice è complessa l’apice fornisce la coniugata trasposta.

Prodotto fra matrici, vettori e scalari

Il prodotto fra matrici di dimensione opportuna, o fra una matrice ed un vettore, si ottiene in maniera intuitiva utilizzando il simbolo asterisco *

```
-->A=[1 2 3; 4 5 6]; B=[1 2; 3 4; 5 6]; C=A*B
 C  =

    22.    28.

    49.    64.
```

Il prodotto di uno scalare per un vettore o per una matrice moltiplica tutti gli elementi per lo scalare stesso:

```
-->A=ones(4,1);c=3;A*c
 ans  =

    3.

    3.

    3.
```

```
    3.
```

Elevamento a potenza di una matrice

L'elevamento a potenza si ottiene tramite il simbolo ∧

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

    37.    54.

    81.    118.
```

Si noti come la stessa sintassi riferita ad un vettore esegua l'elevamento a potenza elemento per elemento:

```
-->a=[1 2 3]; a^3
 ans  =

    1.    8.    27.
```

Esponenziale di una matrice

Data una matrice quadrata $\boldsymbol{A}$ l'operazione di esponenziale di matrice definita come:

$$\boldsymbol{A} = \sum_{i=0}^{\infty} \frac{\boldsymbol{A}^i}{i!}$$

si ottiene con la sintassi:

```
-->A=[1 0; 2 1]; expm(A)
 ans  =

    2.7182818    0.

    5.4365637    2.7182818
```

da non confondere con `exp(A)` che fornisce l'esponenziale elemento per elemento:

```
-->A=[1 0; 2 1]; exp(A)
 ans  =

    2.7182818    1.

    7.3890561    2.7182818
```

Rango, determinante, immagine e nullo di una matrice

Il rango di una matrice si ottiene tramite il comando `rank()` ed il determinante tamite il comando `det()`. Le basi per i sottospazi immagine e nullo sono ottenibili tramite i comandi `orth` e `kernel`, rispettivamente:

```
--> A   =

  - 7.    - 5.      9.   - 3.
  - 5.    - 3.    - 6.   - 14.
  - 5.      6.    - 3.   - 2.
    6.    - 4.      8.     10.
    7.    - 5.    - 6.   - 4.
    10.   - 10.     7.     7.

-->orth(A)
 ans   =

    0.0192660     0.2772380     0.8497724
  - 0.5126586   - 0.4292629     0.4286271
  - 0.2664641     0.2720885   - 0.1775415
    0.5504762     0.1003252     0.0344690
  - 0.0207670   - 0.7193396   - 0.1060547
    0.6019564   - 0.3705749     0.2240734

-->kernel(A)
 ans   =

  - 0.5
  - 0.5
  - 0.5
    0.5
```

Divisioni fra matrici

La sintassi $\boldsymbol{A}\backslash\boldsymbol{B}$ è matematicamente scorretta, poichè la moltiplicazioni fra matrici non sono commutative, infatti, è necessario indicare le *divisioni* come moltiplicazioni per la corrispondente inversa, come, ad esempio $\boldsymbol{A}\boldsymbol{B}^{-1}$. Per questo motivo il carattere di *backslash* fa un'operazione leggermente diversa, viene utilizzato per la divisione sinistra, la sintassi

```
x = A\b
```

risolve l'operazione $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ in cui le matrici devono avere dimensioni opportune e la matrice $\boldsymbol{A}$ può essere quadrata o rettangolare. Scilab implementa una inversione o una pseudoinversione a seconda della dimensione o del rango di $\boldsymbol{A}$.

In maniera analoga, la sintassi compatta

```
x = A/b
```

trova la soluzione per l'equazione algebrica $\boldsymbol{x}\boldsymbol{b} = \boldsymbol{A}$.

Se il lettore non è familiare con questi concetti è bene evitare questa sintassi compatta ed implementare le singole operazioni di cui si necessita.

Polinomio caratteristico

Uno degli usi del comando `poly` è per il calcolo del polinomio caratteristico di una matrice, definito come

$$p(x) = |x\boldsymbol{I} - \boldsymbol{A}| \tag{2.1}$$

in cui $\boldsymbol{A}$ è una matrice quadrata, $\boldsymbol{I}$ la matrice Identità delle stesse dimensioni di $\boldsymbol{A}$, x la variabile polinomiale ed il simbolo $|\cdot|$ rappresenta l'operatore determinante. Un esempio è fornito da:

```
-->A=[-2 4; 0 -1]; p=poly(A,"x"), typeof(p)
 p  =

               2
    2 + 3x + x
 ans  =

 polynomial
```

Per un altro uso del comando `poly` si rimanda alla sezione 2.9.

2.7.1 Operazioni sugli elementi di una matrice

In Scilab è possibile eseguire delle operazioni matriciali elemento per elemento, questo significa che un solo comando permette di risparmiare diverse righe di codice per ripetere la stessa operazione su tutti gli elementi di una matrice. Il comando

```
-->A=[1 %pi/2; 0 -1]; sin(A)
 ans  =

    0.8414710    1.

    0.         - 0.8414710
```

che non ha un significato matematico, calcola il seno dei singoli elementi della matrice.

L'operazione elemento per elemento ha anche un significato differente, date due matrici $\boldsymbol{A}$ e $\boldsymbol{B}$ il cui generico elemento è a_{ij} e b_{ij} la sintassi:

```
-->A=[1 %pi/2; 0 -1]; B=[1 0; 0 -2]; A.*B
 ans  =

    1.    0.

    0.    2.
```

fornisce in uscita la matrice il cui generico elemento ij è pari a $a_{ij} \cdot b_{ij}$. È sufficiente quindi anteporre il punto ad una serie di operazioni per eseguire l'operazione elemento per elemento. Oltre al prodotto, ottenibile con `.*`, è possibile eseguire la divisione con il comando `./` ed elevare a potenza con il comando `.^`.

Si faccia attenzione che la sintassi `A^n` viene interpretata come potenza di una matrice quando $\boldsymbol{A}$ è quadrata e potenza elemento per elemento quando è rettangolare. È buona norma evitare ambiguità ed utilizzare la sintassi `A.^n` anche nel caso di matrice rettangolare.

Se la variabile c è uno scalare ed $\boldsymbol{A}$ una matrice di dimensioni qualsiasi, la sintassi `c.^A` fornisce la matrice il cui generico elemento è $c^{a_{ij}}$.

2.7.2 Concatenare o distruggere matrici e vettori

Esiste una semplice sintassi per costruire matrici o vettori di cui si disponga di blocchi della matrice stessa. Separando le variabili con lo spazio, la virgola o il punto e virgola, infatti, Scilab tenta di costruire la matrice risultante come se gli elementi fossero degli scalari. Ad esempio:

```
-->a = 1; B = [2 3; 4 5]; c = [-1 -1]; D = [a c; c' B]
 D  =

    1.   - 1.   - 1.

  - 1.     2.     3.

  - 1.     4.     5.
```

È anche possibile distruggere solo alcuni elementi di una variabile utilizzando un'opportuna assegnazione:

```
-->D(2,:)=[]
 D  =

    1.   - 1.   - 1.

  - 1.     4.     5.
```

in questo modo alla matrice $\boldsymbol{D}$ è stata eliminata la seconda riga e di conseguenza cambiano le sue dimensioni.

Per eliminare completamente la variabile dallo spazio di lavoro si utilizza il comando `clear D` (si veda la sezione 2.3 sulla gestione delle variabili).

2.8 Operazioni sui numeri complessi

Scilab tratta i numeri complessi allo stesso modo dei numeri reali. Per inserire un numero complesso è sufficiente usare la costante predefinita `%i`:

```
-->a=3+2*%i
 a  =

    3. + 2.i

-->isreal(a)
 ans  =

  F
```

in cui si è usato il comando `isreal` che ritorna `%t` per numeri reali. Si faccia attenzione:

```
-->a=3+0*%i
 a   =

    3.

-->isreal(a)
 ans   =

  F
```

in questo caso il numero è riconosciuto come complesso anche se la componente immaginaria è nulla.

Il coniugato di un numero complesso si ottiene con il comando `conj`. L'operazione di coniugata trasposta di una matrice complessa si ottiene utilizzando semplicemente l'apice: `A'`.

2.8.1 Estrarre parte reale ed immaginaria

I comandi da utilizzare sono `real` e `imag` per estrarre la componente reale ed immaginaria, rispettivamente.

2.8.2 Estrarre modulo e fase

Il modulo di un numero complesso è ottenibile con il comando `abs`, la fase, espressa in gradi, con il comando `phasemag`. Per estrarre contemporaneamente modulo e fase è sufficiente usare il comando `polar`:

```
-->a=3+4*%i
 a   =

    3. + 4.i

-->abs(a)
 ans   =

    5.

-->phasemag(a)
 ans   =

    53.130102

-->[m,f]=polar(a)
 f   =

    0.9272952 - 8.882E-17i
 m   =

    5.
```

Tabella 2.4: Comandi specifici per i numeri complessi

`isreal()`	fornisce `%t` per numeri reali
`conj()`	complesso coniugato
`real()`	parte reale
`imag()`	parte immaginaria
`abs()`	modulo (numero reale)
`phasemag()`	fase (gradi)
`[m,f]=polar()`	modulo e fase (numero reale e radianti)

in cui è bene notare come il modulo sia fornito come numero reale e la fase in radianti. Si noti inoltre che la fase, per questioni di approssimazione numerica, è a sua volta un numero complesso con parte immaginaria prossima allo zero macchina. Infatti:

```
-->%eps
 %eps   =

    2.220E-16

-->isreal(f)
 ans   =

  F
```

La tabella 2.4 riassume i comandi specifici per i numeri complessi.

2.9 Operazioni sui polinomi

Diverse operazioni sono disponibili che lavorano su variabili di tipo `polinomyal`, un elenco completo si può avere consultando la guida in linea sfogliando la cartella "Polynomials" (che contiene anche i comandi definiti sulle variabili `rational`).

2.9.1 Definire un polinomio dalle radici

Il comando `poly` permette di definire un polinomio a partite dalle sue radici. Si supponga di voler definire una variabile `miopol` con radici `-1` e `2`:

```
-->miopol=poly([-1 2],'s')
 miopol   =

              2
  - 2 - s + s
```

Analogamente per definire la varibile polinomiale `s` si definisce un polinomio di grado unitario con radice nulla:

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

```
    s
```

Oppure si utilizza una variabile riservata, disponibile in Scilab per le trasformate di Laplace e la Z-trasformata:

```
-->s=%s
 s  =

    s

-->z=%z
 z  =

    z
```

2.9.2 Definire un polinomio dai suoi coefficienti

Il comando `poly` in realtà permette di definire un polinomio sia a partire dalle sue radici, opzione di default, che dai suoi coefficienti:

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

-->miopol = poly(vect,'s','c')
 miopol  =

              2
  - 2 - s + s
```

2.9.3 Definire un polinomio dalla sua espressione

Un modo semplice per definire un polinomio è utilizzare la variabile polinomiale `s` e l'espressione stessa del polinomio:

```
-->s=%s; miopol = -2-s+s^2
 miopol  =

              2
  - 2 - s + s
```

o, in alternativa:

```
-->miopol = (s+1)*(s-2)
 miopol  =

              2
  - 2 - s + s
```

2.9.4 Calcolare le radici di un polinomio

Il comando `roots` fornisce le radici del polinomio. In ingresso accetta, oltre ad una variabile di tipo polinomiale, anche un vettore, in questo caso gli elementi del vettore sono interpretati come i coefficienti del polinomio in ordine decrescente. Utilizzando la variabile polinomiale `miopol` definita prima, o i suoi coefficienti, si ottiene

```
-->roots(miopol)
 ans  =

  - 1.

    2.

-->roots([1 -1 -2])
 ans  =

  - 1.

    2.
```

2.9.5 Estrarre i coefficienti da una variabile polinomiale

Per estrarre i coefficienti da una variabile polinomiale è disponibile il comando `coeff`:

```
-->miocoeff=coeff(miopol)

 miocoeff  =

  - 2.  - 1.    1.
```

Si noti come, a differenza del Matlab, i coefficienti siano forniti in ordine crescente di grado del polinomio.

2.9.6 Calcolare il valore di un polinomio in un punto

Per calcolare il valore di un polinomio in uno specifico punto è possibile utilizzare il comando `horner`:

```
-->horner(miopol,3.2)

 ans  =

    5.04
```

2.9.7 Sostituzione simbolica di una variabile polinomiale

Il comando `horner` può essere utilizzato anche per eseguire una sostituzione simbolica della variabile polinomiale con un'altra variabile polinomiale o razionale:

```
-->s=poly(0,'s'),p1=s+1,p2=2*s^2+2
 s  =
```

```
    s
 p1  =

    1 + s
 p2  =

          2
    2 + 2s

-->p3=horner(p1,p2)
 p3  =

          2
    3 + 2s
```

Si noti come il prodotto fra due polinomi si ottenga semplicemente utilizzando il simbolo di moltiplicazione, mentre il simbolo di divisione genera una variabile di tipo razionale fratto:

```
-->out=p1*p2,typeof(out)
 out  =

                2    3
    2 + 2s + 2s + 2s
 ans  =

 polynomial

-->out=p1/p2,typeof(out)
 out  =

    1 + s
    -----
          2
    2 + 2s
 ans  =

 rational
```

Per eseguire la divisione fra polinomi si deve ricorrere al comando `pdiv`.

2.9.8 Rassegna sui comandi polinomiali

Una rassegna sui comandi polinomiali:

```
bezout    - Bezout equation for polynomials or integers
clean     - cleans matrices (round to zero small entries)
cmndred   - common denominator form
coeff     - coefficients of matrix polynomial
coffg     - inverse of polynomial matrix
colcompr  - column compression of polynomial matrix
degree    - degree of polynomial matrix
denom     - denominator
```

```
derivat    - rational matrix derivative
determ     - determinant of polynomial matrix
detr       - polynomial determinant
diophant   - diophantine (Bezout) equation
factors    - numeric real factorization
gcd        - gcd calculation
hermit     - Hermite form
horner     - polynomial/rational evaluation
hrmt       - gcd of polynomials
htrianr    - triangularization of polynomial matrix
invr       - inversion of (rational) matrix
lcm        - least common multiple
lcmdiag    - least common multiple diagonal factorization
ldiv       - polynomial matrix long division
numer      - numerator
pdiv       - polynomial division
pol2des    - polynomial matrix to descriptor form
pol2str    - polynomial to string conversion
polfact    - minimal factors
residu     - residue
roots      - roots of polynomials
routh_t    - Routh's table
rowcompr   - row compression of polynomial matrix
sfact      - discrete time spectral factorization
simp       - rational simplification
simp_mode  - toggle rational simplification
sylm       - Sylvester matrix
systmat    - system matrix
```

2.10 Operazioni sulle funzioni razionali fratte

Le funzioni razionali fratte sono di grande importanza nei sistemi dinamici, oltre ad una introduzione sul loro uso data in questa sezione si rimanda al Capitolo 4 per dettagli sulla loro applicazione in problemi di Automatica.

2.10.1 Definire una funzione razionale fratta

A partire dalla conoscenza dei polinomi del numeratore e denominatore, per definire una funzione razionale fratta è sufficiente farne il rapporto. Nell'esempio che segue il numeratore ed il denominatore sono stati definiti a partire dalla loro espressione:

```
-->s=%s; num=1+s; den = (s+2)*(s+2); mioraz=num/den
 mioraz  =

      1 + s
    ---------
              2
    4 + 4s + s

-->typeof(mioraz)
```

```
 ans  =

 rational
```

Dove possibile, Scilab effettua delle semplificazioni per cui fattori comuni ai polinomi vengono cancellati:

```
-->s=%s; num=(1+s)*(s+2); den = (s+2)*(s+2); mioraz=num/den
 mioraz  =

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

È possibile evitare le semplificazione modificando una varibile settata di default sulla modalità semplificazione:

```
-->simp_mode(%F)

-->s=%s; num=(1+s)*(s+2); den = (s+2)*(s+2); mioraz=num/den
 mioraz  =

                2
    2 + 3s + s
    ----------
                2
    4 + 4s + s
```

2.10.2 Estrarre numeratore e denominatore

Facendo riferimento alla variabile `mioraz` semplificata è possibile accedere ai polinomi al numeratore e denominatore con i comandi `numer` e `denom`:

```
-->mioraz
 mioraz  =

    1 + s
    -----
    2 + s

-->numer(mioraz)
 ans  =

    1 + s

-->denom(mioraz)
 ans  =
```

```
    2 + s
```

Si noti come sia possibile accedere al numeratore(denominatore) anche sfruttando il fatto che il tipo `rational` sia una lista:

```
-->mioraz.num
 ans  =

    1 + s
```

2.10.3 Calcolare poli e zeri di una funzione razionale fratta

Per calcolare poli e zeri di una funzione razionale fratta è necessario prima accedere ai polinomi al numeratore e denominatore e poi chiederne le radici:

```
-->s=%s;num=1+s;den=3+s^2;mioraz=num/den
 mioraz  =

    1 + s
    -----
         2
    3 + s

-->roots(mioraz.den)
 ans  =

    1.7320508i
  - 1.7320508i
```

2.10.4 Decomposizione in fratti semplici

È possibile decomporre una funzione razionale fratta in fratti semplice tramite il comando `pfss` come nell'esempio che segue in cui la funzione

$$F(s) = \frac{3(s+2)}{(s+3)(s+1)}$$

è decomposta in

$$F(s) = \frac{1.5}{s+1} + \frac{1.5}{s+3}$$

```
-->s=%s;F=syslin('c',3*(s+2),(s+3)*(s+1));out=pfss(F)
 out  =


        out(1)

     1.5
    -----
    1 + s
```

```
        out(2)

      1.5
     -----
     3 + s

-->typeof(out)
 ans  =

 list
```

Si noti come l'uscita del comando `pfss` sia una lista. È possibile utilizzare questo comando anche con un sistema lineare in spazio di stato (per dettagli sui sistemi dinamici si veda il Capitolo 4).

2.10.5 Calcolare il valore di una funziona razionale in un punto

Il comando `horner` accetta in ingresso sia polinomi che funzioni razionali:

```
-->horner(mioraz,4.33)
 ans  =

    0.2450699
```

2.10.6 Sostituzione simbolica di una variabile in una funzione razionale

Lo stesso comando `horner` permette di eseguire una sostituzione simbolica della una variabile di una funzione razionale con un'altra espressiona razionale. Si supponga di avere la funzione razionale:

$$F(s) = \frac{1+s}{5+s}$$

e di voler effettuare una sostituzione simbolica del tipo

$$s = \frac{z-1}{z+1}$$

che dovrebbe fornire come soluzione

$$F(z) = F(s)|_{s=\frac{z-1}{z+1}} = \frac{z}{2+3z}$$

In Scilab:

```
-->s=%s;raz1=(s+1)/(s+5)
 raz1  =

    1 + s
    -----
    5 + s

-->z=%z;raz2=(z-1)/(z+1)
 raz2  =
```

```
  - 1 + z
    -----
    1 + z

-->horner(raz1,raz2)
 ans  =

     0.3333333z
    ------------
    0.6666667 + z
```

2.11 Le stringhe

Il tipo stringa si ottiene definendo delle variabili utilizzando, indifferentemente, gli apici o le virgolette:

```
-->s='ciao'
 s  =

 ciao
```

È anche possibile definire una matrice di stringhe:

```
-->matstr = ['c' 'ci' 'cia' 'ciao']
 matstr  =

!c  ci  cia  ciao  !

-->typeof(matstr)
 ans  =

 string

-->size(matstr)
 ans  =

    1.    4.
```

Esistono innumerevoli comandi per la gestione delle stringhe, i principali sono

```
ascii       - string ascii conversions
blanks      - Create string of blank characters
code2str    - returns character string associated with Scilab integer
   codes
convstr     - case conversion
emptystr    - zero length string
grep        - find matches of a string in a vector of strings
isalphanum  - check that characters of a string are alphanumerics
isascii     - tests if character is a 7-bit US-ASCII character
isdigit     - check that characters of a string are digits between 0 and
   9
```

```
isletter    - check that characters of a string are alphabetics letters
isnum       - tests if a string represents a number
justify     - Justify character array
length      - length of object
part        - extraction of strings
regexp      - find a substring that matches the regular expression string
sci2exp     - converts an expression to a string
str2code    - return scilab integer codes associated with a character
   string
strcat      - concatenate character strings
strchr      - Find the first occurrence of a character in a string
strcmp      - compare character strings
strcmpi     - compare character strings (case independent)
strcspn     - Get span until character in string
strindex    - search position of a character string in an other string
string      - conversion to string
strings     - Scilab Object, character strings
stripblanks - strips leading and trailing blanks (and tabs) of strings
strncmp     - Copy characters from strings
strrchr     - Find the last occurrence of a character in a string
strrev      - returns string reversed
strsplit    - split a string into a vector of strings
strspn      - Get span of character set in string
strstr      - Locate substring
strsubst    - substitute a character string by another in a character
   string
strtod      - Convert string to double
strtok      - Split string into tokens
tokenpos    - returns the tokens positions in a character string
tokens      - returns the tokens of a character string
tree2code   - generates ascii definition of a Scilab function
```

2.12 Le strutture

In Scilab è disponibile anche la struttura, comune a molti linguaggi di programmazione, che è una collezione di oggetti; ogni oggetto deve essere un oggetto definito in Scilab. La struttura è utile quando si voglia definire un oggetto *complesso* caratterizzato da diverse variabili di diverso tipo. Si faccia l'esempio di voler scrivere una simulazione per la pianificazione del moto di robot mobili, ogni robot è caratterizzato da un nome, da una posizione e velocità, e da altre informazioni quali, ad es., l'essere acceso o spento. Queste informazioni possono essere raggruppate in una struttura opportuna tramite il comando `struct`:

```
-->robot=struct('name',[],'pos',[],'vel',[],'on',[])
 robot  =

   name: [0x0 constant]
   pos: [0x0 constant]
   vel: [0x0 constant]
   on: [0x0 constant]
```

il cui tipo è ottenuto tramite il comando `typeof`:

```
-->typeof(robot)
 ans  =

 st
```

È possibile accedere agli elementi della struttura semplicemente usando la loro etichetta, quindi, ad esempio, per definire il nome del primo robot è possibile scrivere:

```
-->robot.name='neo';
```

ed in modo analogo è possibile definire la posizione e velocità come vettori 1×2 e lo stato acceso/spento come variabile booleana.

È utile definire dei vettori di strutture, nel caso in esame, è utile definire un vettore di robot. Il secondo robot si definisce semplicemente come:

```
-->robot(2).name='morpheus';
```

La variabile robot ha ora due elementi, ognuno dei quali è una struttura[2]. Si possono richiamare i nomi dei vari robot semplicemente con il comando che segue:

```
-->robot.name
 ans  =

       ans(1)

 neo

       ans(2)

 morpheus
```

La definizione della struttura può avvenire contestualmente alla definizione degli elementi del primo elemento. Il primo robot, `neo`, può essere definito con la sua posizione, velocità e stato con il solo comando:

```
-->robot=struct('name','neo','pos',[0 0],'vel',[0 0],'on',%T)
 robot  =

   name: "neo"
   pos: [0,0]
   vel: [0,0]
   on: %t
```

È sempre possibile aggiungere un campo alla struttura, si faccia riferimento al vettore di due robot appena definiti, si vuole aggiungere il campo `modello` che definisce il modello di robot. È sufficiente il comando seguente:

```
-->robot(1).modello='beta0.9';
```

[2]Il terzo robot si chiamerebbe, ovviamente, `trinity`

Tabella 2.5: Operatori per il calcolo simbolico

`addf`	addizione
`subf`	sottrazione
`mulf`	moltiplicazione
`rdivf`	divisione
`cmb_lin`	combinazione lineare
`trianfml`	triangolarizzazione
`solve`	risoluzione di $\boldsymbol{Ax} = \boldsymbol{b}$
`eval`	valuta espressione

che aggiunge il campo anche per il secondo robot per il quale, ovviamente non è stato ancora definito un valore corrispondente:

```
-->robot(2).modello
 ans  =

     []
```

Sui campi della struttura sono poi definite tutte le operazioni esistenti per il tipo corrispondente.

Concettualmente analoghe alle strutture sono le *liste*, che si definiscono tramite i comandi `list`, `tlist` e `mlist`.

2.13 Calcolo simbolico

Scilab non è dotato di un motore di calcolo simbolico come Maple, Mathematica o lo stesso Matlab. Rende però possibili delle semplici operazioni simboliche sfruttando i comandi definiti per le stringhe. In particolare sono disponibili le funzioni riportare nella Tabella 2.5.

Si definisca una variabile simbolica `out` tramite il comando `addf`. Tale variabile è di tipo stringa:

```
-->  out = addf("a","x")
 out  =

 a+x

-->  typeof(out)
 ans  =

 string
```

Solo semplici semplificazioni sono possibili come nel caso seguente:

```
-->  addf("x+2","y-2")
 ans  =

 x+y
```

La moltiplicazione e la divisione si ottengono in maniera analoga

```
-->  mulf("x+1","y-1")
 ans  =

 (x+1)*(y-1)

-->  rdivf("x^2+x-1","x+1")
 ans  =

 (x^2+x-1)/(x+1)
```

così come è poi possibile implementare una combinazione lineare simbolica fornendo 4 ingressi di tipo stringa (si noti il segno meno):

```
-->  z = cmb_lin("c(1)","x","c(2)","y+2")
 z  =

 c(1)*x-c(2)*(y+2)
```

L'espressione risultante, memorizzata nella variabile stringa `z`, può essere valutata fornendo il valore numerico di tutte le variabili simboliche presenti:

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

-->  x = 3;

-->  y = 5;

-->  z = eval(z)
 z  =

  - 11
```

il risultato è ora un numero:

```
-->  typeof(z)
 ans  =

 constant
```

Un'altra operazione simbolica consiste nella triangolarizzazione, compiuta attraverso operazioni elementari sulle righe della matrice:

```
-->  A = ["a11","a12";"a21","a22"]
 A  =

!a11  a12  !
!          !
!a21  a22  !

-->  trianfml(A)
 ans  =

!a21  a22              !
```

```
!                          !
!0     a21*a12-a11*a22   !
```

Il comando `solve`, infine, permette di risolvere un sistema di equazioni lineari

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

in cui la matrice dei coefficienti $\boldsymbol{A}$ si assume triangolare superiore:

```
-->  A = ["a","b";"0","d"]
 A  =

!a  b  !
!      !
!0  d  !

-->  b = ["2";"4"]
 b  =

!2  !
!   !
!4  !

-->  solve(A,b)
 ans  =

!a\ (-b*(d\4)+2)  !
!                 !
!d\4              !
```

Si faccia attenzione al fatto che non viene eseguito alcun controllo sulla forma triangolare di $\boldsymbol{A}$, i cui elementi sotto la diagonale principale vengono semplicemente ignorati.

A questo punto diventa un semplice esercizio quello di usare queste funzioni anche per risolvere una sistema di equazioni con $\boldsymbol{A}$ non triangolare e ricavare l'inversa di una matrice.

In particolare, dato il problema $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ con $\boldsymbol{A}$ non triangolare si costruisce la matrice aumentata

$$\overline{\boldsymbol{A}} = [\,\boldsymbol{A} \quad \boldsymbol{b}\,]$$

su questa si esegue un'operazione di triangolarizzazione con il comando `trianfml` ottenendo una matrice $\overline{\boldsymbol{A}}_1$ partizionabile in due matrici delle stesse dimensioni delle matrici $\boldsymbol{A}$ e $\boldsymbol{b}$ del problema originario:

$$\overline{\boldsymbol{A}}_1 = [\,\boldsymbol{A}_1 \quad \boldsymbol{b}_1\,].$$

il risultato è l'uscita del comando `solve` ottenuto fornendo in ingresso $\boldsymbol{A}_1$ e $\boldsymbol{b}_1$.

Il comando `solve` può essere utilizzato per equazioni matriciali del tipo $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$, l'inversa simbolica si ottiene semplicemente utilizzando la matrice Identità di dimensioni opportune al posto del termine noto $\boldsymbol{B}$ ed eventualmente applicando la procedura appena descritta nel caso in cui la matrice dei coefficienti non sia triangolare.

Parte dei comandi utilizzati in questa sezione sono disponibili nel file `manuale_symbolic.sce`.

2.14 Script

Uno script è una collezione di istruzioni salvate in un file che possono essere eseguite con un singolo comando.

Per definire uno script in Scilab è sufficiente salvarlo in un file opportuno con suffisso `sce`; si noti come i file `.sce` siano file `ASCII`, ossia testuali, e possano essere editati con uno degli innumerevoli programmi disponibili. In Scilab è utilizzabile un semplice editor di testo scritto appositamente per la sintassi Scilab; un'applicazione con la sintassi Scilab già disponibile è anche Kile sotto sistemi Unix-like come MacOSX o Linux con librerie Qt. Per lanciare l'editor di testo è possibile utilizzare il menu `Applications` dalla finestra principale di Scilab, oppure aprire un nuovo documento con l'icona corrispondente.

Fino alla versione 5.1 l'editor si chiamava Scipad e poteva essere chiamato anche da riga di comando tramite il comando `scipad()`.

Può essere utile inserire dei commenti negli script e nelle funzioni (sezione successiva), questo si ottiene tramite il doppio slash come in C: `//`.

Per eseguire le istruzioni contenute in uno script è possibile scrivere il comando:

```
-->exec('nomefile.sce')
```

oppure lanciarlo dall'editor di testo tramite il comando da menu o i tasti rapidi.

2.15 Funzioni

Per definire una funzione in Scilab è sufficiente salvarla in un file opportuno con suffisso `sci` (per gli script si usa il suffisso `sce`); si noti come anche i file `.sci` siano file `ASCII`.

Più funzioni Scilab possono essere salvate in uno stesso file che, ad esempio, raggruppi una libreria d'utente su uno specifico tema.

Ogni funzione deve avere la struttura seguente:

```
function [out1, ..., outm] = nomefunzione(input1, ..., inputn)

   comandi

endfunction
```

dove il numero di ingressi `input1, ..., inputn` e di uscite `out1, ..., outm` dipende dalla specifica funzione e può essere eventualmente nullo. Una stessa funzione può essere chiamata con un diverso numero di ingressi, e quindi gestire, ad esempio, ingressi opzionali. In questo caso la funzione può sapere il numero di ingressi (ed uscite) richiamando il comando `argn()`.

Una volta scritta una funzione in un file opportuno è necessario richiamare in memoria la funzione (le funzioni) di un determinato file con il comando `exec('nomefile.sci')` (o anche tramite l'obsoleto `getf('nomefile.sci')`). A tal proposito è bene conoscere i comandi per determinare la cartella di lavoro in cui ci si trova: con il comando `pwd` si ottiene il percorso corrente che può essere poi modificato con il comando `chdir` o graficamente utilizzando l'apposita icona.

È quindi possibile creare una prima libreria contenente due semplici funzioni. Scriviamo il file `primalibreria.sci`:

```
// calcola la distanza fra due punti nel piano
function out = distanza(x,y)

    out = sqrt((x(1)-y(1))^2+(x(2)-y(2))^2);

endfunction

// calcola la distanza fra due punti nel piano
// e fornisce il risultato in decibel
function out = distanzadb(x,y)

    out = distanza(x,y);
    out = 20*log10(out);

endfunction
```

Per poter utilizzare questa libreria di due funzioni si deve caricare in memoria il contenuto del file con il comando `exec('primalibreria.sci');` le due funzioni sono quindi visibili tramite il comando `who` e possono essere utilizzate dall'utente.

Si noti come il caricamento in memoria di una o più librerie possa avvenire anche all'interno di un'altra funzione o script. Si noti inoltre come una funzione, o script, possa utilizzare le altre funzioni caricate in memoria.

2.15.1 Passaggio dei parametri di una funzione

Particolare cura deve essere data al passaggio dei parametri di una funzione e, in generale, alla visibilità delle variabili.

In Scilab, contrariamente al Matlab ed al C, il codice di una funzione *vede* tutto lo spazio di lavoro e può quindi utilizzarne le variabili. È buona norma di programmazione non scrivere funzioni che facciano uso di variabili che non siano sole e quelle passate alla chiamata alla funzione stessa. Si faccia attenzione che in Scilab è possibile commettere un errore di programmazione involontario quando, all'interno di una funzione, si utilizza una variabile non inizializzata (per errore) ma questo comportamento, assumendo che la variabile sia stata definita nella funzione chiamante, non viene intercettato da Scilab ed origina il classico errore di semantica. A nostro modesto parere, questo dovrebbe rientrare negli errori di sintassi (ed essere quindi intercettato) levando la visibilità completa dello spazio di lavoro alle funzioni.

A partire dalla versione 2.4 dello Scilab il passaggio dei parametri di una funzione avviene per riferimento se la funzione stessa non modifica il parametro stesso e per copia se lo modifica. Questo garantisce che, a valle della chiamata della funzione, i parametri di ingresso non vengano modificati dalla funzione stessa. Questo differente modo di gestire i parametri influenza la velocità di esecuzione del codice ed è bene tenerne conto nella scrittura di funzioni. Il seguente esempio illustra questo concetto, queste 3 funzioni sono state salvate in un file chiamato `test_passaggio.sci`:

```
// passaggio per riferimento
function out = test1(x,i)
```

```
    out = 2*x(i);

endfunction

// passaggio per copia
function out = test2(x,i)

    x(i) = 2*x(i);
    out = x(i);

endfunction


// chiamata alle due funzioni
function test

    n = 100000;
    x = rand(n,1);
    timer(); for i=1:n; out = test1(x,i); end; t1=timer()/n
    timer(); for i=1:n; out = test2(x,i); end; t2=timer()/n
    printf(' t1 = %f / n t2 = %f',t1,t2)

endfunction
```

Una volta caricate in memoria e lanciata la funzione `test` si ottiene:

```
-->test
 t1 = 0.000015

 t2 = 0.001208
```

Un'altra importante caratteristica che rappresenta un limite nella gestione delle variabili e l'assenza di una variabile di tipo *statico* o *persistente*. Sono variabili definite all'interno di una funzione che, fra una chiamata e l'altra della funzione, non perdono il proprio valore. Questa opzione è disponibile in `C` con il comando `static` o in Matlab, Freemat e Octave con `persistent`. È possibile ottenere lo stesso risultato definendo una variabile globale sia nella funzione chiamante che nella funzione di interesse, è chiaro che questa soluzione diminuisce la portabilità e la robustezza del codice.

2.16 Operatori logici e di comparazione

In Scilab si utilizzano gli seguenti operatori logici riportati in tabella 2.6 e gli operatori di comparazione riportati in tabella 2.7.

2.17 Comandi per il controllo del flusso

Così come gli altri linguaggi di programmazione, Scilab mette a disposizione dei comandi per scrivere dei cicli e controllare il flusso del programma. È bene sottolineare che, poichè Scilab,

Tabella 2.6: Operatori logici

&	e
|	o
~	non

Tabella 2.7: Operatori di comparazione

==	uguale a
<	strettamente inferiore
>	strettamente superiore
<=	minore o uguale a
>=	maggiore o uguale a
~= o <>	diverso da

come Matlab, interpreta i comandi durante l'esecuzione è bene ridurre al minimo l'utilizzo di cicli e di utilizzare, dove disponibili, le primitive Scilab per le operazioni matriciali.

2.17.1 Il ciclo `for`

Il ciclo `for` presenta la sintassi

```
for variabile=espressione

   comandi

end
```

simile in tutti i linguaggi di programmazione, nel caso più generale, se si vuole eseguire un numero noto di cicli, si utilizza la sintassi:

```
-->n=10;

-->for i=1:n, printf(' %d ',i), end

 1  2  3  4  5  6  7  8  9  10
```

oppure:

```
-->for i=n:-2:1, printf(' %d ',i), end

 10  8  6  4  2
```

2.17.2 Il ciclo `while`

Il ciclo `while` presenta la sintassi

```
while espressione

   comandi
```

```
end
```

ed anche in questo caso il suo significato è comune a molti linguaggi di programmazione. Esempio

```
-->x=2; while x<8, x=2*x, end
 x  =

    4.

 x  =

    8.
```

2.17.3 L'istruzione `if then else`

La sintassi per l'istruzione condizionale `if-then-else` è la seguente:

```
if condizione1 then

   comandi

elseif condizione2 then

   comandi

elseif condizione3 then

   comandi

else

   comandi

end
```

2.17.4 L'istruzione `select case`

La sintassi per l'istruzione condizionale `select-case` è la seguente:

```
select variabile

case espressione1

   comandi

case espressione2

   comandi

case espressione3
```

```
   comandi

else

   comandi

end
```

2.17.5 L'istruzione `break`

Permette di uscire dal ciclo più interno di istruzioni `for` o `while` e fa *saltare* il programma al comando che segue l'istruzione `end` alla fine del ciclo.

Come per qualsiasi linguaggio, anche in Scilab, l'istruzione `break` va usata con cautela perchè possibile fonte di bachi nel programma.

2.17.6 L'istruzione `error`

Per abortire un programma od una funzione è disponibile l'istruzione `error`. L'istruzione `warning` permette di avvisare l'utente di una situazione anomala ma non interrompe l'esecuzione del programma.

2.17.7 Le istruzioni `pause-return-abort`

Un modo grossolano per *debaggare* (sverminare, per i puristi della lingua...) il proprio codice può essere fornito dall'utilizzo delle istruzioni `pause-return-abort`. La prima, inserita in uno script o funzione, fornisce il cursore e da la possibilità di eseguire delle istruzioni da riga di comando come, ad es., verificare il valore di alcune variabili. L'istruzione `return` riprende l'esecuzione dello script/funzione ed infine l'istruzione `abort` esce dallo script/funzione.

2.18 Importare ed esportare dati

Ci sono diversi modi possibili per salvare le proprie variabili in un file. Una possibilità è quella di utilizzare il comando `save` che permette di creare un file binario in cui mettere tutte, o alcune, delle variabili presenti nello spazio di lavoro. Il codice seguente crea 3 variabili di diverso tipo, le salva in un file, poi le cancella dallo spazio di lavoro e le ricarica con il comando `save`:

```
-->A=[1 2 3; 4 5 6];b='ciao';c=3.12;

-->save('dati')

-->clear all

-->load('dati')
```

Si noti come il file binario creato non provochi troncamenti nelle variabili ma permetta la trasportabilità delle variabili solo da un'applicazione Scilab ad un'altra. La portabilità, quindi, è limitata alla disponibilità del programma Scilab installato sulla macchina.

Scilab mette a disposizione dei comandi che emulano le funzioni di lettura/scrittura su file del lunguaggio C. Per aprire/chiudere un file è possibile utilizzare i comandi `mopen/mclose` e per scrivere o leggere i comandi `mfprintf/mfscanf`. Analogamente, per scrivere sulla finestra Scilab, gli emulatori dei corrispondenti comandi C sono `mprintf/mscanf`. Per chi è pratico del C l'uso dei comandi Scilab non comporta difficoltà aggiuntive.

Per salvare una matrice in un file testuale, Scilab mette a disposizione una funzione `fprintfMat`. Utilizzando il formato C è possibile salvare la matrice, aggiungendo anche un eventuale commento e richiamarla tramite il comando `fscanfMat`.

Per chi è familiare con il linguaggio Fortran, Scilab mette a disposizione due comandi, `write` e `read` che permettono di salvare le variabili in file testuali utilizzando il formato Fortran. Con il comando:

```
-->v=1:10;write('dati3',v,'(10(i2,3x))')
```

si crea un file testuale la cui unica riga può essere letta da un qualsiasi editor di testo ed è:

```
1  2  3  4  5  6  7  8  9  10
```

Ulteriori dettagli sui formati possono aversi consultando un manuale Fortran.

Esiste infine la possibilità di scrivere o leggere i file di dati sia binari che testuali in formato Matlab (fino alla versione 6 di Matlab in Scilab 5.1, fino alla versione 7.3 da Scilab 5.2). Per farlo è possibile utilizzare le funzioni `savematfile/loadmatfile` con la sintassi riportata nella guida in linea dello Scilab.

2.19 Riportare dati sulla finestra di comando Scilab

Per visualizzare dei dati sulla finestra di comando Scilab è possibile utilizzare il comando `printf` con la stessa sintassi C oppure il comando `disp` che permette di visualizzare tutti i tipi di variabili definiti in Scilab. Ad esempio:

```
-->s=poly(0,'s');x=s^2+1;y=s-1;F=y/x;disp([x y F])

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

2.20 Personalizzare la finestra di comando Scilab

Dalla versione 5.1, personalizzare l'apparenza della finestra di comando Scilab è immediato sia per sistemi Windows che Linux: è sufficiente selezionare il menu `preferences` e modificare l'opzione desiderata.

In Linux, fino alla versione 4.1 era necessario modificare il file

```
$SCILABPATH/X11_defaults/Xscilab
```

dove `$SCILABPATH` è il percorso di installazione di Scilab. Le modifiche venivano applicate al riavvio di Scilab. In particolare, lo sfondo e la dimensione del font di default sono

```
Xscilab.color*background:white
Xscilab.color*foreground:black
Xscilab*vpane.Vtsci.font:9x15
```

per ottenere un font blu su sfondo grigio di domensioni ridotte vanno modificate in

```
Xscilab.color*background:gray
Xscilab.color*foreground:blue
Xscilab*vpane.Vtsci.font:8x12
```

Un'alternativa più pulita era quella di lasciare inalterato il file `Xscilab` e di aggiungere le modifiche desiderate nel file

```
$HOME/.Scilab
```

2.21 Lanciare uno script all'avvio

Può essere utile scrivere uno script che, all'avvio di Scilab, esegua una serie di operazioni di inizializzazione definite dall'utente.

Per farlo è sufficiente individuare il file `scilab.start` all'interno del proprio PC e modificarlo aggiungendo i comandi desiderati *alla fine* del file stesso. In una distribuzione Ubuntu 9.04 il file è in

```
/usr/share/scilab/etc/scilab.start
```

Un'alternativa è quella di aggiungere al file `scilab.start`, come ultima riga, un comando del tipo `exec('$PATH/fileavvio.sce');` e poi lavorare sul file `fileavvio.sce`.

Esiste una soluzione più pulita, che permette anche di personalizzare l'esecuzione di comandi in avvio su macchine gestite da più utenti. In avvio Scilab cerca il file `SCIHOME/.scilab`, se esiste, ed esegue il codice in esso contenuto. Non c'è quindi bisogno di modificare file appartenenti alle cartelle del programma. Nel caso siano entrambi presenti, il codice in `SCIHOME/.scilab` viene eseguito prima di `scilab.start`. La cartella `SCIHOME` si trova in:

```
Linux/Unix : /home/<User>/.Scilab/<Scilab-Version>
Windows    : C:/Documents and Settings/<User>/Scilab/<Scilab-Version>
Vista      : C:/<User>/AppData/Roaming/Scilab/<Scilab-Version>
```

in cui `<User>` è evidentemente il nome utente.

I due file utilizzati sulla macchina di uno degli autori, `.scilab` e `fileavvio.sce` sono forniti con il codice associato al manuale.

2.22 Lanciare comandi di sistema

Può essere necessario lanciare dei comandi di sistema. Il comando per farlo è `host()` in cui l'argomento è una stringa contenente il comando da eseguire. Si noti come `host()` non preveda uscite e quindi non è possibile lanciare comandi che richiedano un uscita testuale come, ad esempio, il comando `dir` (sotto Windows) o `ls` (sotto Linux).

Sotto Linux esistono delle varianti del comando `host()` che permettono maggiore flessibilità d'uso e superano le limitazioni di `host()` consentendo anche l'uscita sulla finestra Scilab:

`unix`	-	shell (sh) command execution
`unix_g`	-	shell (sh) command execution, output redirected to a variable
`unix_s`	-	shell (sh) command execution, no output
`unix_w`	-	shell (sh) command execution, output redirected to scilab window
`unix_x`	-	shell (sh) command execution, output redirected to a window

2.23 Esiste lo zero? alcuni commenti sull'accuratezza numerica

I programmi di calcolo numerico sono soggetti ad una serie di errori nell'esecuzione dei calcoli. L'argomento è ampio e non è il caso di trattarlo, nemmeno superficialmente, in questa sede. Si vuole solo richiamare l'attenzione sul fatto che errori di approssimazione numerica possano sorgere nelle operazioni apparentemente più semplici.

Si generi una matrice quadrata $\boldsymbol{T} \in \mathbb{R}^{3\times 3}$ con numeri casuali a funzione di densità uniforme fra 0 ed 1:

```
-->T=grand(3,3,'def')
 T  =

   0.1711867    0.7972799    0.2769230
   0.3019131    0.0318328    0.8724288
   0.7060461    0.3165504    0.0461714
```

dall'algebra è ovvio che

$$\boldsymbol{T}^{-1}\boldsymbol{T} = \boldsymbol{I}$$

in cui $\boldsymbol{I}$ è la matrice Identità. Eseguiamo questa operazione:

```
-->inv(T)*T
 ans  =

    1.            0.          - 1.388D-17
    2.776D-17     1.            2.602D-18
    0.            2.776D-17     1.
```

è evidente come questa matrice sia *praticamente* la matrice identità. Si ricordi inoltre che la costante predefinita di Scilab `%eps` contiene la precisione ed è funzione della macchina:

```
-->%eps
 %eps  =

    2.220D-16
```

Il comando `clean` permette di approssimare con zero gli elementi *piccoli* di una matrice. Con la matrice T di sopra infatti:

```
-->clean(inv(T)*T)
 ans  =

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

Un altro esempio è dato in sezione 4.1.3 in cui una *semplice* conversione di rappresentazione di un sistema lineare da origine ad un errore di arrotondamento.

2.24 Tip & tricks

- Il tasto di movimento del cursore "alto" riporta in riga di comando i comandi precedenti. Dalla versione 5.2 è possibile *filtrare* i comandi in base alle prime lettere;
- Dalla versione 5 è disponibile l'autocompletamento dei comandi, è sufficiente iniziare il comando e poi premere il tasto `[Tab]`. Questa funzione vale anche per i nomi delle cartelle e dei file;
- È possibile definire delle variabili *vuote* tramite la sintassi `x=[]`. In questo modo l'utente può creare una variabile le cui dimensioni ed i cui elementi vengono definiti nel seguito.

Capitolo 3

La grafica

Scilab mette a disposizione una buona varietà di comandi destinati alla grafica. È da evidenziare, però, come la gestione della grafica sia generalmente più laboriosa rispetto a Matlab.

3.1 Gestire le finestre grafiche

Ogni comando grafico viene eseguito su una finestra differente rispetto a quella principale che gestisce l'interprete dei comandi. Possono coesistere più finestre grafiche ciascuna identificata da un diverso numero intero.

Se si utilizza un qualsiasi comando di grafica come, ad esempio, `plot2d` (vedere la Sezione 3.2), Scilab crea una finestra grafica e la numera partendo da 0. A differenza di Matlab, i successivi comandi di grafica vengono aggiunti alla finestra corrente a meno che non si chieda esplicitamente a Scilab di aprirne una nuova.

La Tabella 3.1 riassume le operazioni basilari sulle finestre grafiche riportando alcuni possibili istruzioni Scilab che consentono di effettuarle. Si noti che ove un'operazione si riferisca ad un numero di finestra inesistente essa si traduce nella creazione della finestra corrispondente. Nel seguito si descrivono brevemente alcune di queste istruzioni, rinviando alla guida in linea di Scilab per la descrizione esaustiva delle diverse sintassi con cui possono essere utilizzate.

L'istruzione `scf` serve a rendere corrente (ovvero attiva) una finestra grafica; conseguentemente, tutti i comandi grafici successivi si applicano alla finestra così individuata. Con la sintassi `scf(num)` la finestra `num` se esistente viene resa attiva, altrimenti viene dapprima creata e poi resa attiva. In alternativa al numero `num` si può utilizzare in argomento un handle `h`. Il comando `scf()` crea una nuova finestra di indice `num` successivo al massimo valore correntemente esistente

Tabella 3.1: Operazioni basilari sulle finestre grafiche e possibili istruzioni per realizzarle.

creare una nuova finestra	`figure(num), scf(num)`
rendere corrente la finestra `num`	`scf(num)`
mostrare la finestra corrente	`xselect()`
ripulire la finestra `num`	`clf(num)`, `xclear(num)`
chiudere la finestra `num`	`xdel(num)`

e la rende attiva. In ogni caso, l'uso di un argomento di uscita (p.es., `h=scf(num)`) consente di restituire l'handle alla figura interessata dal comando.

L'istruzione `clf` serve a sbiancare una finestra grafica: `clf()` oppure `clf('clear')` elimina tutti gli oggetti contenuti nella figura corrente, mentre con la sintassi `clf('reset')` oltre all'eliminazione degli oggetti contenuti si riportano tutti le proprietà della figura corrente ai loro valori iniziali. Se si utilizza la sintassi `clf(num)`, `clf(num,'clear')` o `clf(num,'reset')` il comando si riferisce alla figura identificata dal valore dell'intero `num`; se la figura `num` non esiste viene creata. In alternativa al numero `num` si può utilizzare un handle `h`.

L'istruzione `xclear` realizza la pulizia di una o più finestre grafiche contemporaneamente; in quest'ultimo caso, l'argomento del comando è un vettore di valori interi che elenca gli indici delle finestre da sbiancare; al solito, se una finestra oggetto dell'istruzione non esiste viene creata. La sintassi `xclear()` si riferisce alla finestra corrente.

L'istruzione `xdel` chiude una o più finestre grafiche contemporaneamente; in quest ultimo caso, l'argomento del comando è un vettore di valori interi che elenca gli indici delle finestre da chiudere. La sintassi `xdel()` si riferisce alla finestra corrente.

Il comando `figure`, introdotto nella versione 5.1, può essere usato senza argomento ed aggiunge una figura a quelle correnti. In questo caso, di default, il background e' grigio, per averlo bianco si deve usare il comando `scf` (o modificarlo tramite l'handle come spiegato in questo capitolo).

Dalla versione 5.1 è poi diventato obsoleto il comando `xset`, utilizzato per gestire le proprietà grafiche delle finestre.

In Scilab non esiste un singolo comando per chiudere tutte le finestre contemporaneamente; è quindi necessario costruire una funzione d'utente. È poi utile copiarla nel file `fileavvio.sce`, lanciato all'avvio come spiegato nella sezione 2.21, per averla sempre disponibile.

Un possibile listato per la funzione `xdelall` è riportato di seguito:

```
function xdelall()

    wins = winsid();
    for w = wins
        xdel(w);
    end
endfunction
```

3.2 Grafici a 2 dimensioni

3.2.1 Il comando plot2d

L'utilizzo elementare del comando `plot2d` è il seguente:

```
x = (-1:.1:3)';
y = x.^2;
plot2d(x,y)
```

in cui si definisce dapprima un vettore delle ascisse (colonna!) e poi si costruisce un vettore delle ordinate. In questo caso si è deciso di rappresentare una parabola nell'intervallo $[-1, 3]$ con un

passo di 0.1; il risultato è in figura 3.1 (si veda la sezione 3.5 per salvare su file il contenuto delle finestre grafiche).

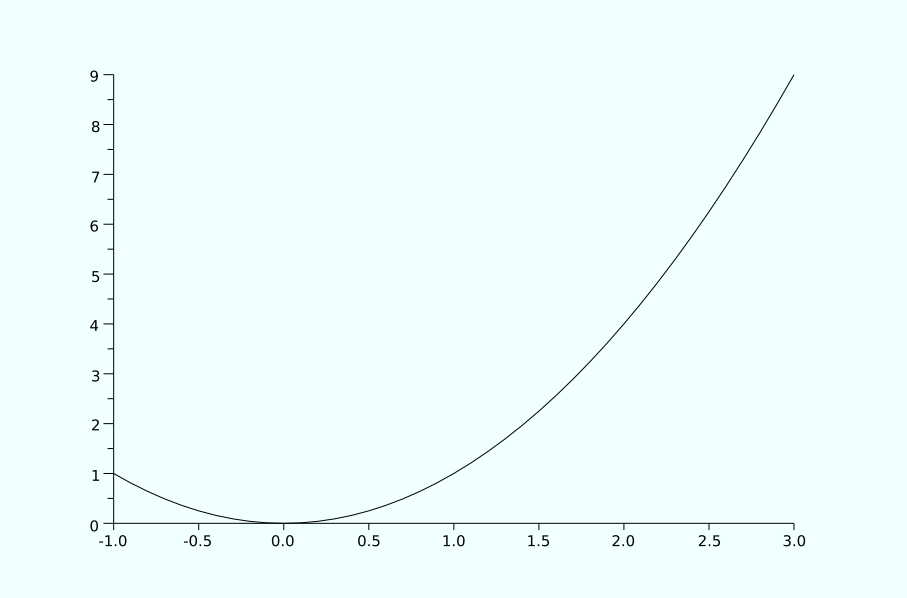


Figura 3.1: Un semplice `plot2d`

Il comando `plot2d` può ricevere come secondo ingresso una matrice, in questo caso Scilab interpreta il comando come la sovrapposizione di più comandi singoli riferiti alle singole colonne. Se, ad esempio, si volesse disegnare la funzione valore assoluto insieme alla parabola si potrebbe aggiungere al codice precedente il codice:

```
y2 = abs(x);
plot2d(x,[y y2]);
```

come riportato in figura 3.2. Si noti come Scilab abbia colorato la seconda linea, contenente i dati relativi alla seconda colonna, in blu.

3.2.2 Scegliere i colori

Utilizzando il comando con ulteriori ingressi opzionali è possibile scegliere il colore con cui disegnare la curva:

```
plot2d(x,[y y2],[opt1 opt2]);
```

disegna i dati in colonna utilizzando, rispettivamente, il colore individuato dalla variabile `opt1` e `opt2`. La tabella 3.2 riporta i codici dei 32 colori della mappa di default, la Figura 3.3 riporta la mappa così come ottenibile dal comando `getcolor()` che ne permette la scelta interattiva. La gestione dei colori è abbastanza evoluta e si rimanda alla guida in linea per dettagli, è possibile fruire di diverse mappe di colore e definire dei colori personalizzati in base al loro codice RGB (Red Green and Blue). In assenza di questo ingresso Scilab utilizza i colori corrispondenti a valori crescenti della tabella, nell'esempio precendete, infatti, i primi due colori erano il nero ed il blu.

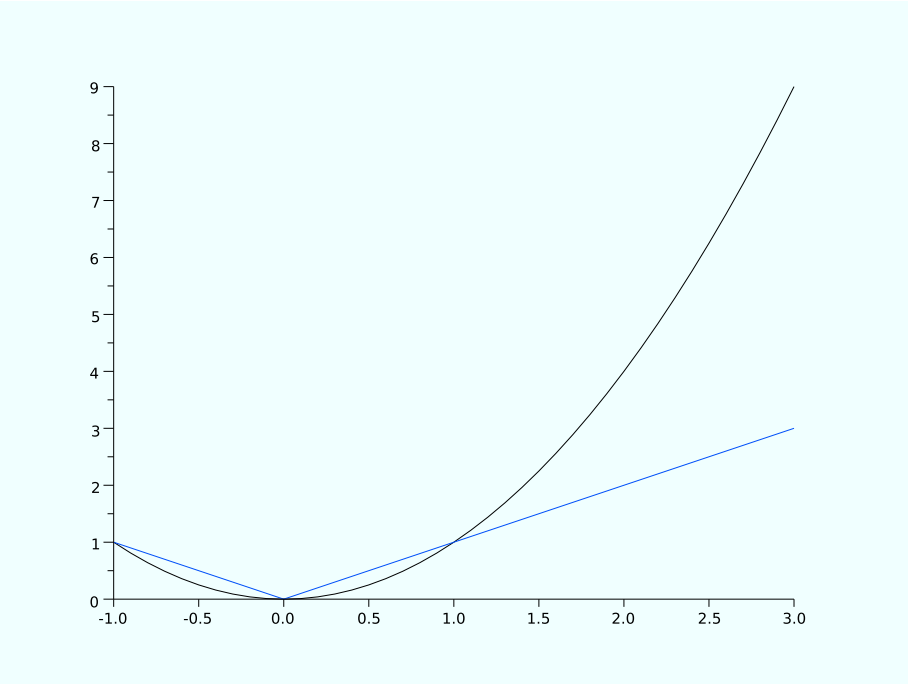


Figura 3.2: Il comando `plot2d` con una matrice in ingresso

Si noti come l'utilizzo di un ulteriore comando `plot2d` non apra una nuova finestra grafica come in Matlab ma disegni sulla finestra corrente partendo dal colore nero se non esplicitamente indicato un colore diverso.

Si faccia anche attenzione al colore n. 8, il bianco, che può provocare delle fastidiose perdite di tempo alla ricerca della curva *invisibile* per matrici con almeno 8 colonne.

3.2.3 Impostare la mappa dei colori

La mappa dei 32 colori di default ha, oltre al difetto del colore bianco, anche quello di non essere *continua*. Molti disegni richiedono che colori vicini nella mappa siano *simili* (Un esempio è dato dal disegno di superfici mostrato in Sezione 3.3.3). Per cambiare la gestione dei colori è prima necessario definire il concetto di mappa dei colori, ossia una matrice con il numero desiderato di righe e 3 colonne per le 3 componenti RGB (Red Green e Blue) con scala $[0, 1]$. Tale matrice può essere definita dall'utente o caricata usando delle mappe predefinite:

- `C = graycolormap(n)`. Scala di grigi di n livelli;
- `C = hotcolormap(n)`. Scala dal rosso al bianco di n livelli;
- `C = jetcolormap(n)`. Scala dal blue al rosso passando per verde-giallo-arancio di n livelli;

e poi caricata impostata tramite la successione di comandi

```
plot2d(x,[y y2]);
f = gcf();
f.color_map = C;
```

Tabella 3.2: Codice dei 32 colori corrispondenti alla mappa colore di default

1	nero
2	blu
3	verde chiaro
4	azzurro
5	rosso
6	magenta
7	giallo
8	bianco
9-12	tonalità di blu
13-15	tonalità di verde
16-18	tonalità di azzurro
19-21	tonalità di rosso
22-24	tonalità di magenta
25-27	tonalità di marrone
28-31	tonalità di rosa
32	*gold*

Si faccia attenzione che il comando carica la mappa dei colori per il contesto grafico *corrente* e non in maniera definitiva, è necessario quindi lanciare prima un comando di plot e poi caricare la nuova mappa dei colori ed eventualmente lanciare altri comandi di plot. Dettagli sul comando `gcf` sono forniti in sezione 3.4.

La mappa corrispondente a 16 livelli di tipo `hotcolormap` è riportata in figura 3.4.

3.2.4 Aggiungere titolo, etichette e legenda

Il grafico in figura 3.2 può senz'altro essere reso più leggibile aggiungendo un titolo, delle etichette per le coordinate e per le ascisse ed una legenda, il colore delle due linee, inoltre, viene espressamente indicato in rosso e blu:

```
plot2d(x,[y y2],[2 5],leg='parabola@valore assoluto');
xtitle('Due semplici funzioni','x','y')
```

il risultato è mostrato in figura 3.5.

Il valore di default per il testo potrebbe essere valido per la visualizzazione su schermo, come in questo caso, ma non per il savataggio su file e la successiva stampa. Per modificare la dimensione del testo è necessario aggiungere delle righe di codice:

```
nice_font = 4;
plot2d(x,[y y2],[2 5]);
legend('parabola','valore assoluto');
h = gce();
h.font_size = nice_font;
xtitle('Due semplici funzioni','x','y')
a=gca();
a.font_size = nice_font;
a.x_label.font_size = nice_font;
```

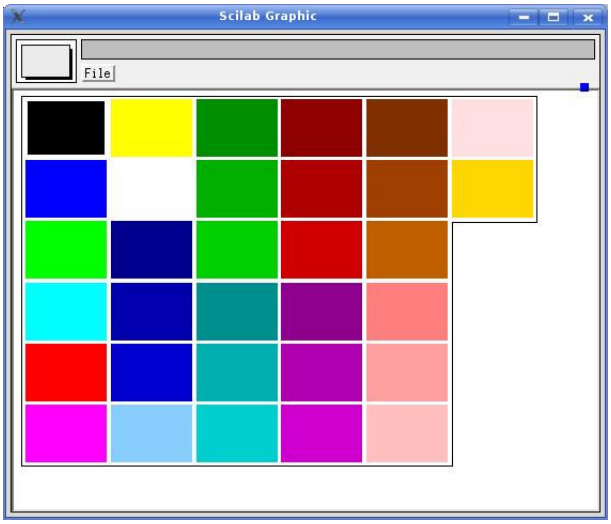


Figura 3.3: Mappa dei 32 colori di default ottenuta con il comando `out=getcolor()`. I numeri si leggono dalla casella in alto a sinistra a scendere.

Tabella 3.3: Simboli utilizzabili con `style`

style	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
simbolo	.	+	×	⊕	♦	◊	△	▽	♣	◯

```
a.y_label.font_size = nice_font;
a.title.font_size = nice_font;
```

Il risultato è mostrato in figura 3.6. I comandi `gce` e `gca` sono illustrati in dettaglio nella sezione 3.4 dove viene anche fornito un metodo interattivo per agire sugli stessi parametri.

3.2.5 Disegnare simboli

È anche possibile disegnare dei simboli al posto di linee o, per la precisione, della interpolante continua a tratti che unisce i punti del vettore in ordinata. L'utilizzo dell'opzione `style` serve allo scopo, si noti come la scelta di un valore positivo per lo stile equivalga a scegliere il colore mentre un valore negativo a scegliere il simbolo nero secondo la tabella 3.3.

L'esempio seguente, riportato in figura 3.7, mostra l'utilizzo dell'opzione `style` per disegnare un simbolo al posto dei valori di un vettore:

```
h=scf();
h.children.font_size=nice_font;
plot2d(x,y,-7);
```

In figura 3.8 si mostra un esempio di come si possano combinare gli stili per ottenere risultati compositi:

```
h=scf();
h.children.font_size=nice_font;
plot2d(x,[y y],[5 -7]);
```

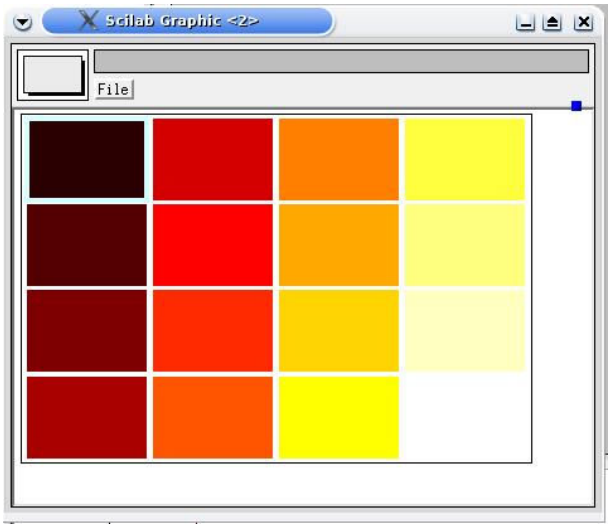


Figura 3.4: Mappa dei colori corrispondente a `hotcolormap(16)`.

3.2.6 Creare più grafici in una finestra

Il comando per creare più grafici in una sola finestra grafica è `subplot` che ammette due sintassi diverse. Si devono passare al comando gli stessi 3 parametri `mnp` dove `m` ed `n` rappresentano il numero di righe e colonne di una matrice virtuale in cui suddividere la finestra e `p` rappresenta l'elemento scelto. Il codice seguente produce la figura 3.9:

```
figure;
subplot(221),plot2d(x,y,-6);
subplot(222),plot2d(x,y,-1);
subplot(2,2,3),plot2d(x,[y y],[5 -9]);
subplot(2,2,4),plot2d(x,[y y],[-9 5]);
```

3.2.7 Scegliere la scala

Scilab si prende cura di determinare, per default, gli estremi degli assi per disegnare tutti i punti del comando `plot2d`. Per agire sulla scala è possibile utilizzare l'opzione `frameflag` come si evince dal codice seguente, dalla tabella 3.4 e dalla figura 3.10:

```
figure;
theta=0:.1:2*%pi;
x=cos(theta);
y=sin(theta);
subplot(221),plot2d(x,y,5);xgrid
subplot(222),plot2d(x,y,5,frameflag=1,rect=[-2 -2 2 2]);xgrid
subplot(2,2,3),plot2d(x,y,5,frameflag=3,rect=[-2 -2 2 2]);xgrid
subplot(2,2,4),plot2d(x,y,5,frameflag=4);xgrid
```

È anche possibile modificare la scala dopo che si sia eseguito il comando `plot2d`. Un modo è quello di agire con lo strumento grafico `zoom` e selezionare con il mouse la regione di interesse, un'alternativa più precisa è quella di modificare gli estremi alterando le proprietà dell'oggetto; si veda in proposito la sezione 3.4, la proprietà da modificare è `data_bounds`.

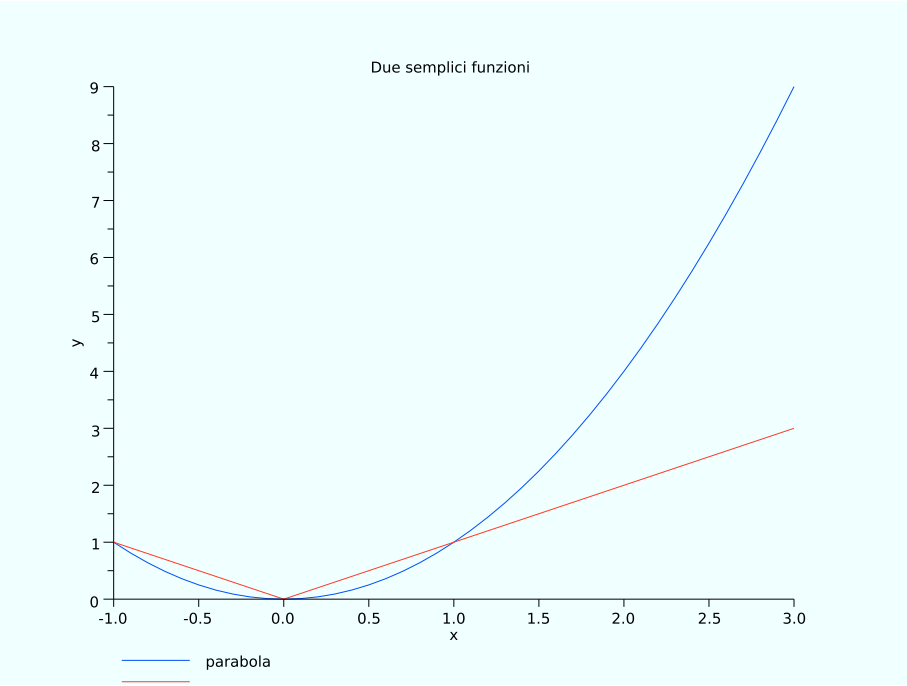


Figura 3.5: Grafico con titolo, etichette e legenda. Si noti come, in questo caso, l'etichetta per la linea blu non sia stata stampata per un baco del comando di esportazione del grafico in un file (si veda la sezione 3.5)

I grafici mostrati in questa sezione sono stati ottenuti con il codice contenuto nel file `manuale_2d.sce`.

3.2.8 Imporre una scala logaritmica

È possibile imporre una scala logaritmica per le ascisse e/o le ordinate come opzione del comando `plot2d`. Il codice che segue genera due vettori di punti, uno spaziato linearmente e l'altro logaritmicamente e poi genera due grafici nella stessa finestra, il secondo grafico è semilogaritmico e la curva rappresentata è proprio un segmento:

```
x=1:1:50;
y=logspace(1,6,50);
subplot(211);
plot2d(x,y,logflag="nn")
subplot(212);
plot2d(x,y,logflag="nl")
```

3.2.9 Selezionare gli assi

Analogamente alla scala, è possibile modificare le impostazioni relative agli assi utilizzando l'opzione riportata in tabella 3.5 del comando `plot2d`.

In alternativa, è possibile agire sul parametro `data_bounds` dell'*handle* ricavato tramite `gca()` come descritto nella sezione 3.4.

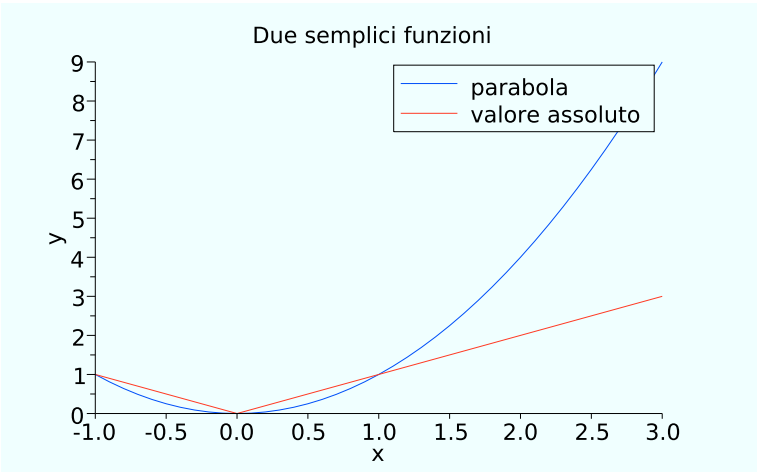


Figura 3.6: Grafico con titolo, etichette e legenda con font di dimensione 4

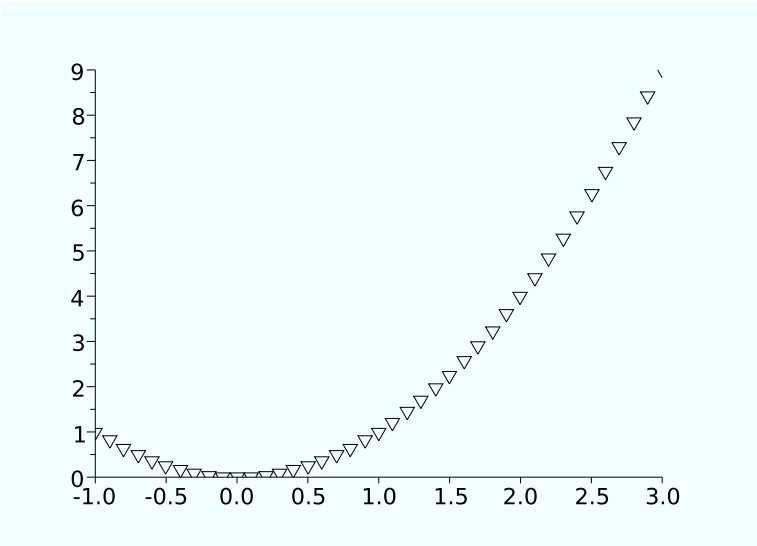


Figura 3.7: Esempio di utilizzo dell'opzione `style`

3.2.10 Aggiungere o modificare del testo

Il comando per aggiungere del testo ad un grafico è `xstring`:

```
-->xstring(x,y,'testo')
```

in cui `x`, `y` sono le coordinate dell'angolo basso a sinistra del testo.

3.2.11 Lo strumento `Datatip`

Con il nome `Datatip` si intende uno strumento, principalmente grafico, che consente di visualizzare interattivamente i punti da un grafico bidimensionale. Ogni finestra grafica permette di attivare/disattivare il modo datatip sia tramite icona che con il comando apposito sotto il menu `Edit`. A questo punto è sufficiente cliccare con il tasto sinistro per far apparire una piccola

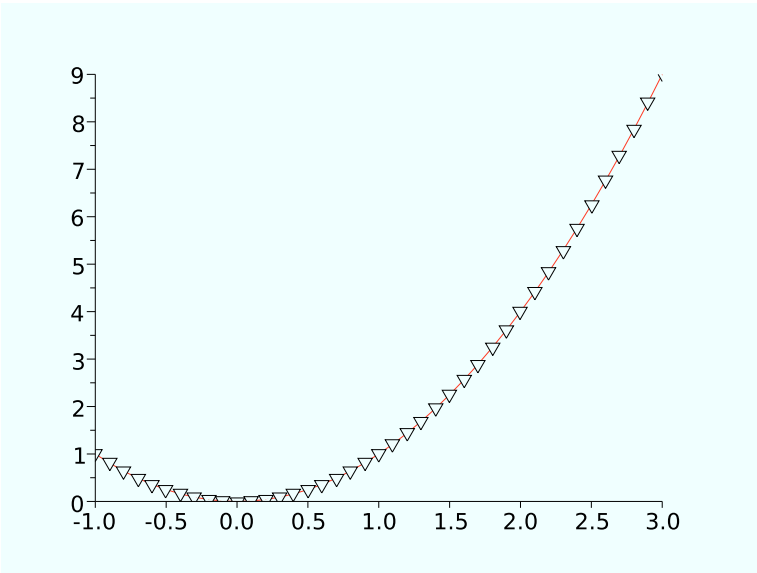


Figura 3.8: Esempio di utilizzo dell'opzione `style` per combinare stili

Tabella 3.4: Tabella delle possibili scale

`frameflag`	scala
`frameflag=0`	precedente
`frameflag=1`	fornita tramite il parametro `rect`
`frameflag=2`	calcolata tramite max e min di Mx e My
`frameflag=3`	isometrica e fornita tramite il parametro `rect`
`frameflag=4`	isometrica e calcolata tramite max e min di Mx e My
`frameflag=5`	come 1 ma con eventuale adattazione graduale
`frameflag=6`	come 2 ma con eventuale adattazione graduale
`frameflag=7`	come 1 ma le curve precedenti sono ridisegnate
`frameflag=8`	come 2 ma le curve precedenti sono ridisegnate

etichetta con le informazioni riguardanti la curva nel punto più vicino in cui si è cliccato come riportato in figura 3.11.

3.2.12 Altri tipi di grafici

Oltre al comando `plot2d` esistono altri tipi di grafici predefiniti così come riportato in sezione 3.8.

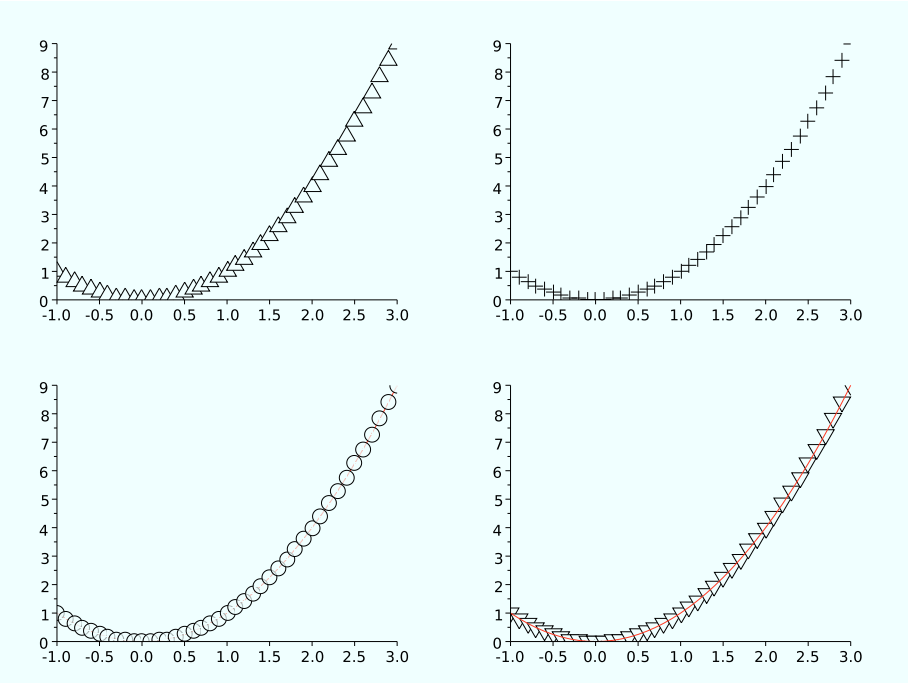


Figura 3.9: Uso del comando `subplot`

Tabella 3.5: Tabella delle opzioni sugli assi

`axesflag`	assi
`axesflag=0`	senza cornice, nè assi, nè graduazioni
`axesflag=1`	con cornice, assi, graduazioni (x in basso, y a sinistra)
`axesflag=2`	con la cornice, ma senza assi nè graduazioni
`axesflag=3`	con cornice, assi, graduazioni (x in basso, y a destra)
`axesflag=4`	senza cornice ma con assi e graduazioni (tracciati verso il centro)
`axesflag=5`	senza cornice ma con assi e graduazioni (tracciati in y = 0 e x = 0)

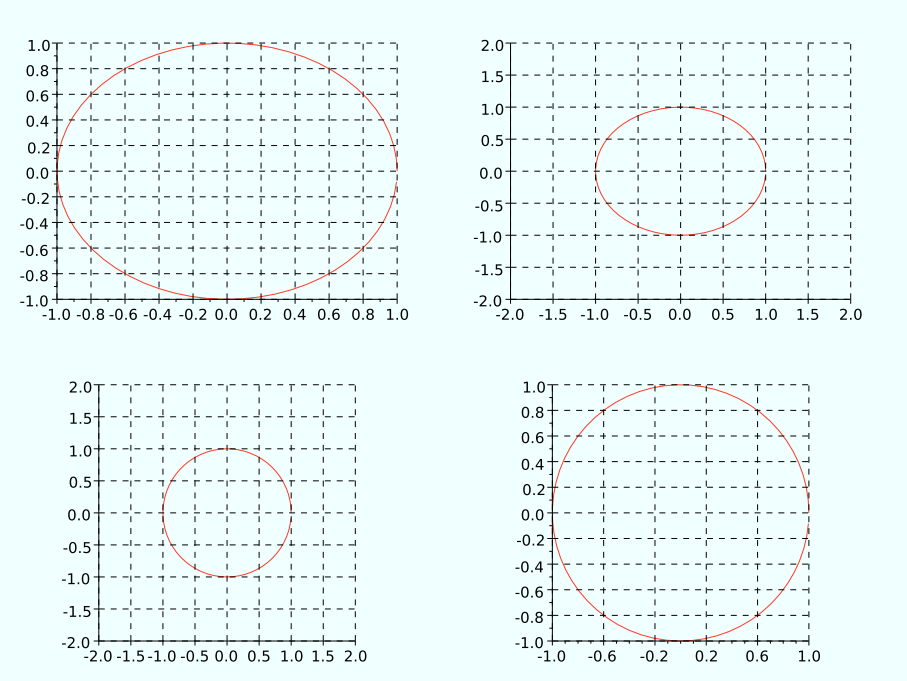


Figura 3.10: Un cerchio disegnato con diverse scale

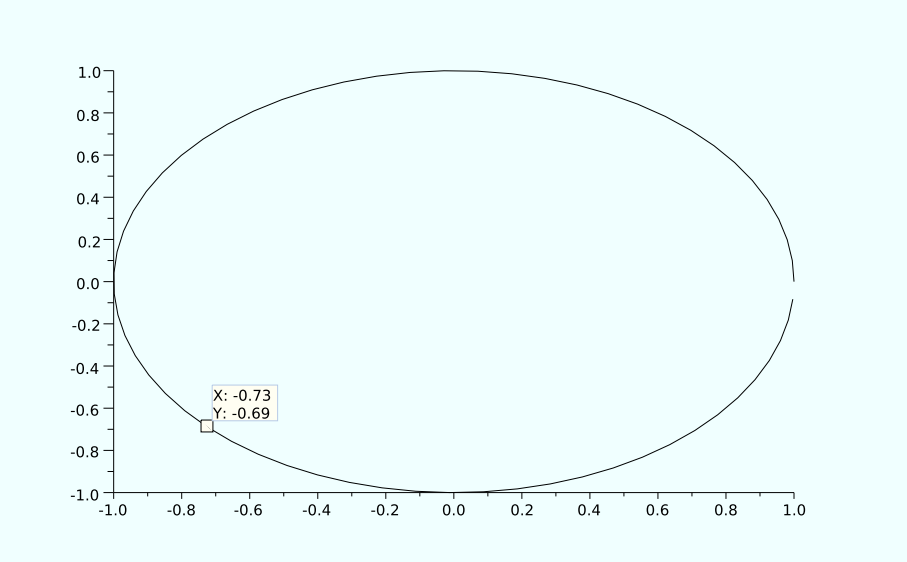


Figura 3.11: Esempio di utilizzo del `Datatip` per estrarre informazioni puntuali su uno specifico punto di una curva.

3.3 Grafici a 3 dimensioni

3.3.1 Disegnare un punto

Per un disegnare un punto in 3D non esiste un comando specifico, si deve utilizzare uno dei comandi per disegnare le curve `param3d1()` con un parametro opportuno. La sintassi per disegnare un solo punto è quindi

```
-->param3d1(x,y,list(z,0))
```

valido per variabili scalari, per disegnare più punti insieme si deve opportunamente fornire il valore 0 del parametro con la sintassi certamente poco intuitiva:

```
-->param3d1(x,y,list(z,zeros(z)))
```

La figura 3.12 rappresenta il grafico ottenuto tramite il codice seguente in cui sono stati utilizzati dei comandi per modificare le proprietà dei singoli oggetti che verranno introdotti nella Sezione 3.4:

```
subplot(211)
t=0:0.05:5*%pi;param3d1(sin(t),cos(t),list(t/10,zeros(t)));
subplot(212)
t=0:0.05:5*%pi;param3d1(sin(t),cos(t),list(t/10,zeros(t)));
h=gce();
h.mark_size_unit="point";
h.mark_size=4;
h.mark_foreground = 5;
```

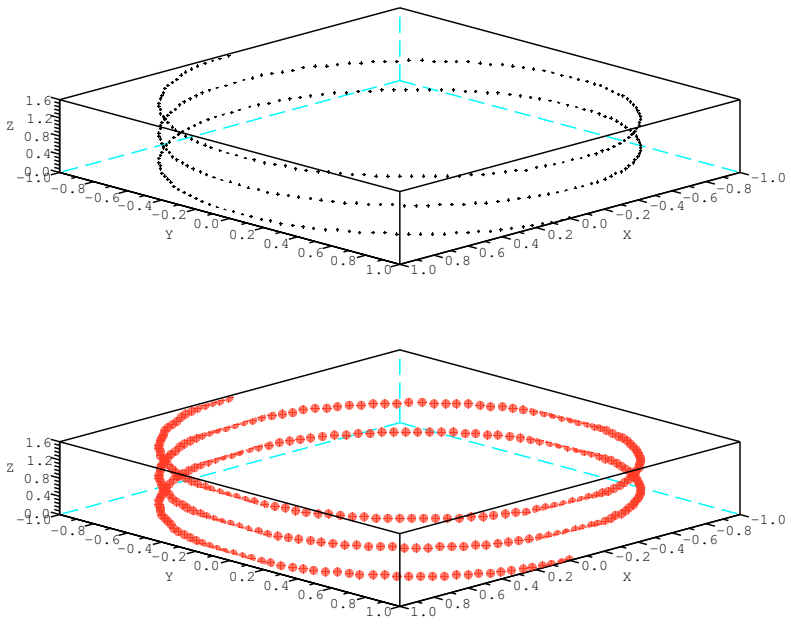


Figura 3.12: Esempio di punti in 3D

3.3.2 Disegnare una curva

Il comando utilizzato per disegnare punti è naturalmente designato al disegno di curve in 3D. Esistono due versioni dello stesso comando: `param3d()`e `param3d1()` che differiscono solo nel caso in cui si vogliano disegnare più curve contemporaneamente. La sintassi più semplice è:

```
-->param3d1(x,y,z)
```

in cui i 3 vettori, riga o colonna, contengono le coordinate dei punti da unire tramite segmenti. Si possono utilizzare anche matrici in ingresso al comando, in questo caso le colonne vengono interpretate come comandi separati. L'esempio che segue:

```
-->t=0:0.05:5*%pi;t=t';param3d1([sin(t) 3+sin(t)],[cos(t) cos(t)],[t/5 t
   ])
```

fornisce il risultato in figura 3.13

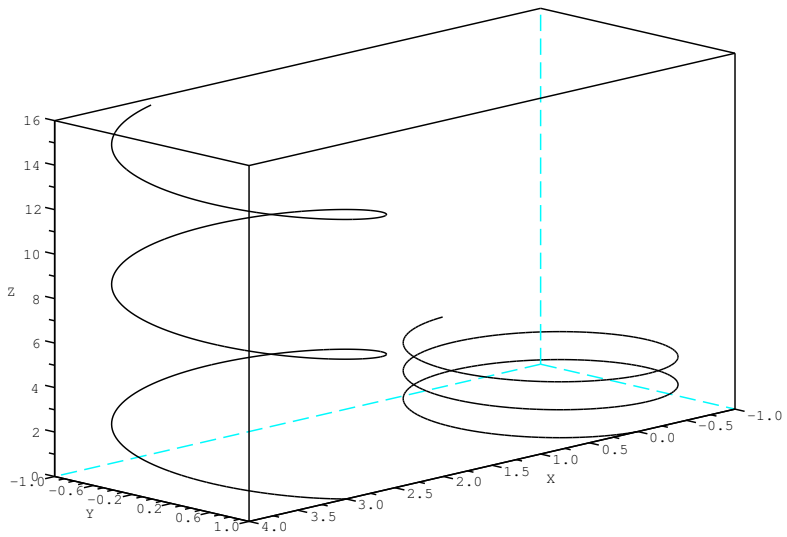


Figura 3.13: Esempio di curve in 3D ottenute tramite il comando `param3d1()`

3.3.3 Disegnare una superficie

Il modo più semplice di disegnare una superficie è quello di discretizzare la relazione

$$z = f(x, y) \tag{3.1}$$

e di considerare il dominio di rappresentazione rettangolare. Come esempio si consideri la funzione

$$f(x, y) = \cos(x)\cos(y) \quad \text{per } (x, y) \in [0, 2\pi] \times [0, 2\pi] \tag{3.2}$$

Nella implementazione più semplice è sufficiente definire due vettori per il dominio ed una matrice `z(i,j)=f(x(i),y(j))`:

```
x = linspace(0,2*%pi,31);
y = x;
z = cos(x)'*cos(y);
```

ed il comando `plot3d(x,y,z,[theta,alpha,leg,flag,ebox])` per il quale, sempre facendo riferimento alla modalità di implementazione più semplice, può essere utile utilizzare fornendo il solo parametro `flag`, vettore di 3 elementi, che condiziona pesamente la modalità di rappresentazione della curva. In particolare

- `flag=[mode, type, box]`:
  - `mode`: intero per gestire i colori
    - `mode>0`: la superficie è colorata con il valore fornito in `mode` e con il *reticolo*;
    - `mode=0`: la superficie non è colorata ma ottenuta disegnando il solo *reticolo* (mesh);
    - `mode<0`: la superficie è colorata con il valore fornito in `mode` senza *reticolo*;
  - `type`: intero da 0 a 6 per gestire la scala (dettagli nell'help in linea)
  - `box`: intero da 0 a 4 il riquadro attorno al disegno (dettagli nell'help in linea)

In definitiva, il codice seguente genera la figura 3.14 in cui sono stai utilizzati due valori del parametro `mode`:

```
subplot(121)
plot3d(x,y,z,flag=[2 4 4])
subplot(122)
plot3d(x,y,z,flag=[0 4 4])
```

Il problema evidente della figura 3.14 è la difficoltà nel percepire l'altezza dei punti della superficie. Può essere opportuno utilizzare un colore proporzionale al valore della funzione. Per farlo è ncessario utilizzare il comando `plot3d1()`. È anche necessario, però, modificare la mappa dei colori come illustrato nella Sezione 3.2.3.

Le figure 3.15 e 3.16 sono ottenute caricando le mappe di colori predefinite sul rosso e la scala di grigi utilizzando il codice seguente

```
h=figure();
h.color_map = hotcolormap(32);
subplot(121)
plot3d1(x,y,z,flag=[1 4 4])
subplot(122)
plot3d1(x,y,z,flag=[-1 4 4])

h=figure();
h.color_map = graycolormap(32);
subplot(121)
plot3d1(x,y,z,flag=[1 4 4])
subplot(122)
plot3d1(x,y,z,flag=[-1 4 4])
```

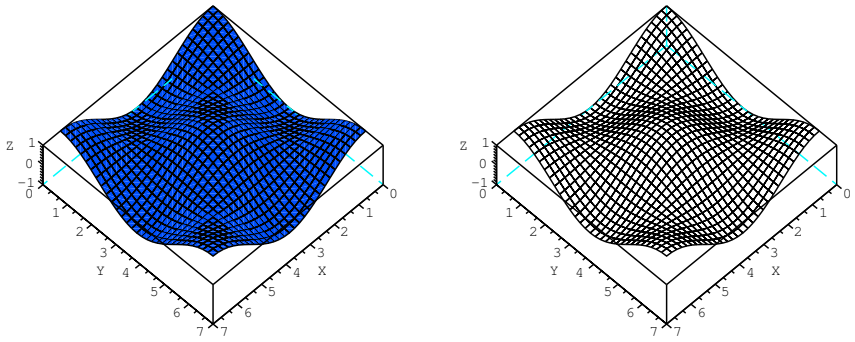


Figura 3.14: Esempio di superficie in 3D ottenuta tramite il comando `plot3d()` con `mode=2` (sinistra) e `mode=0` (destra)

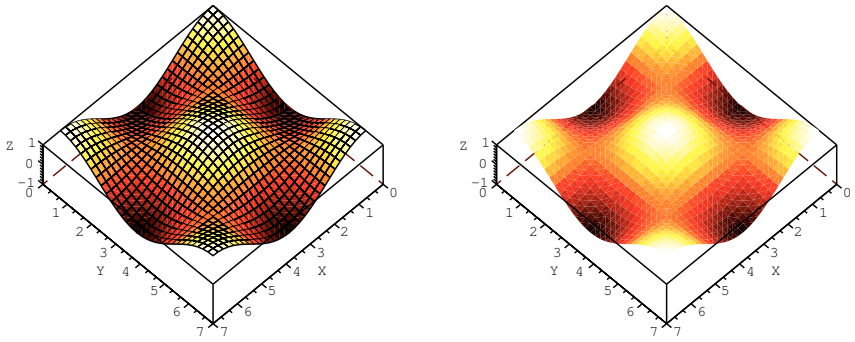


Figura 3.15: Esempio di superficie in 3D ottenuta tramite il comando `plot3d1()` e `hotocolormap` con `mode>0` (sinistra) e `mode<0` (destra)

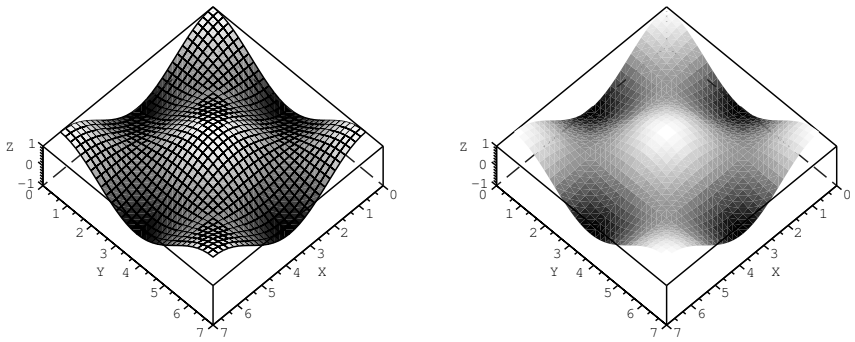


Figura 3.16: Esempio di superficie in 3D ottenuta tramite il comando `plot3d1()` e `graycolormap` con `mode>0` (sinistra) e `mode<0` (destra)

3.4 Scoprire le proprietà degli oggetti grafici

I comandi grafici hanno una serie di opzioni che permettono di personalizzare l'aspetto degli ogetti presenti nella figura stessa. Non tutte le possibilità sono però raggiungibili tramite le opzioni del comando si pensi, ad es., al colore del simbolo, tramite il comando non è possibile modificarlo.

È quindi necessario accedere alle proprietà dell'oggetto tramite altra via. Disegnamo un grafico con due assi e del testo aggiunto:

```
h=scf(9);
theta=0:.1:2*%pi;
x=cos(theta);
y=sin(theta);
subplot(211);
plot2d(x,y);
h1=gca();
h1.font_size=nice_font;
subplot(212);
h2=gca();
h2.font_size=nice_font;
plot2d(x,y,-7,frameflag=4);
xstring(-.2,0,'del testo')
```

riprodotto in figura 3.17.

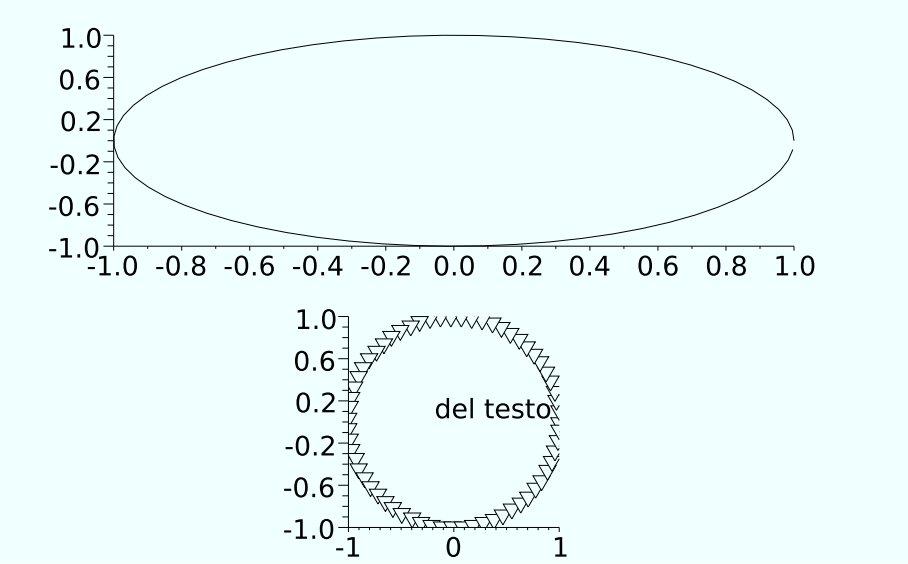


Figura 3.17: Un semplice grafico composto da due assi

Il comando `gcf()` (get current figure) fornisce un *handle* alla figura:

```
-->h=gcf()
 h  =

Handle of type "Figure" with properties:
========================================
children: ["Axes";"Axes"]
```

```
figure_position = [200,200]
figure_size = [624,602]
axes_size = [613,456]
auto_resize = "on"
viewport = [0,0]
figure_name = "Graphic␣window␣number␣%d"
figure_id = 0
info_message = ""
color_map= matrix 32x3
pixmap = "off"
pixel_drawing_mode = "copy"
immediate_drawing = "on"
background =  -2
visible = "on"
rotation_style = "unary"
event_handler = ""
event_handler_enable = "off"
user_data = []
tag = ""
```

da cui si evince che ci sono due `children`, corrispondenti ai due assi disegnati. Per visualizzare le proprietà di, per es., il primo è sufficiente scrivere:

```
-->h.children(1)
 ans  =

Handle of type "Axes" with properties:
======================================
parent: Figure
children: ["Text";"Compound"]

visible = "on"
... [continua per una ventina di righe]
```

ed eventualmente modificare i parametri che interessano, ad esempio aggiungendo solo le griglie verticali:

```
-->h.children(1).grid=[1 -1]
```

È chiaro a questo punto come anche questo oggetto abbia due children, uno dei quali è proprio la *curva* rappresentata tramite simboli cui poter cambiare il colore:

```
-->h.children(1).children(2).children.mark_foreground=5
```

oppure lo stile e lo spessore per la curva dell'asse superiore:

```
-->h.children(2).children.children.line_style=2;
-->h.children(2).children.children.thickness=2;
```

ottenendo l'effetto in figura 3.18.

Modificare le proprietà dei grafici tramite gli *handle* è opportuno quando l'operazione deve essere ripetibile, per modifiche *una tantum* si può ricorrere all'interfaccia grafica `Figure Editor` che compare selezionando la voce `Figure properties` nel menu `Edit` della finestra grafica. La

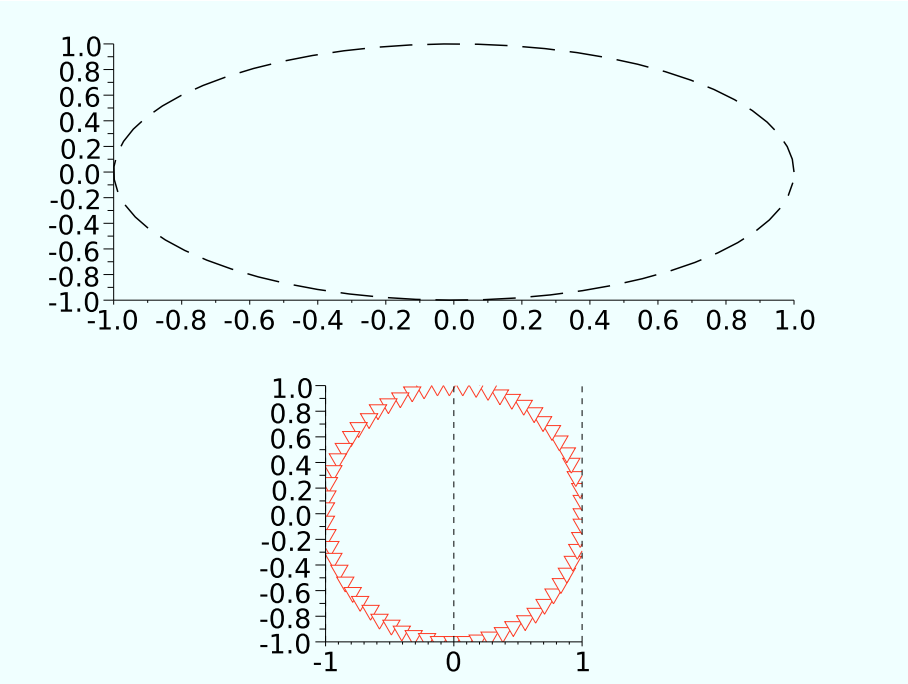


Figura 3.18: Un semplice grafico composto da due assi modificato tramite gli handles

finestra di dialogo `Figure Editor` consente di accedere all'`Axes Editor` mediante selezione della voce `Axes(1)` nella sezione `Objects Browser` (vedi Figure 3.19 e 3.20).

Altri due comandi danno accesso all'handle di un oggetto e posso essere utili:

```
h = gca();
h = gce();
```

che si riferiscono, rispettivamente, al corrente asse o entità. Sul primo comando non c'è molto da aggiungere a quanto detto per il comando `gcf` che ne rappresenta la versione *parent*. Il secondo si riferisce all'ultimo oggetto creato e può essere utile per scrivere codice più snello.

3.5 Esportare grafici

È possibile salvare un grafico nei più comuni formati utilizzando i comandi `xs2...`. In particolare, `xs2bmp`, `xs2eps`, `xs2fig`, `xs2gif`, `xs2ppm`, `xs2ps` e, solo per Windows, `xs2emf`.

Un baco del comando `xs2eps` è riportato in figura 3.5. In genere, vale anche per Matlab, il rapporto fra le dimensioni dei font da schermo a file può cambiare durante il salvataggio ed è quindi necessario modificarle.

Per modificare le dimensioni in pixel dell'immagine è possibile agire sul parametro corrispondente. Ad esempio, per imporre un'imagine di 640×480 pixel è sufficiente scrivere:

```
h=gcf();
h.axes_size=[640 480];
```

verificando che i pixel in `h.figure_size` siano maggiori.

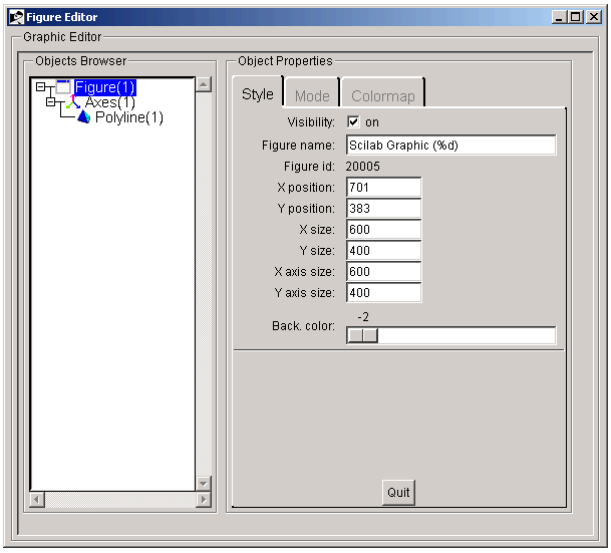


Figura 3.19: La finestra di dialogo `Figure Editor`.

Tabella 3.6: Oggetti definibili con il comando `uicontrol`

`Pushbutton`	bottone generalmente usato per lanciare un callback
`Radiobutton`	bottone con due stati mutualmente esclusivi
`Checkbox`	bottone con due stati (usato per scelte multiple indipendenti)
`Edit`	una zona di tipo stringa editabile
`Text`	una zona di tipo stringa generalmente statica
`Slider`	barra di controllo per impostare un valore in un dato intervallo con il mouse
`Frame`	una zona per raggruppare controlli similari
`Listbox`	una lista che può essere selezionata con il mouse
`Popupmenu`	una lista che può essere selezionata con il mouse ed appare al click del mouse

3.6 Programmare una GUI

In Scilab è possibile costruire un'interfaccia grafica per applicazioni interattive, ossia una GUI (Graphical User Interface). Alla GUI è possibile aggiungere pulsanti, barre, testo e quant'altro è normalmente disponibile in altri linguaggi per lo stesso scopo. Dalla versione 5.1 c'è stato un miglioramento significativo nella gestione della GUI rispetto alle versioni precedenti; è possibile, ad esempio, costruire oggetti in una finestra in cui sono presenti dei grafici, cosa precedentemente non possibile.

Il comando principale con cui gestire le GUI è `uicontrol`, permette la definizione degli oggetti con cui interagire. La tabella 3.6 riporta i principali oggetti/comandi che possono essere creati con questo comando.

In questa sezione, tramite un semplice esempio, si forniranno le basi per impostare e costruire una GUI personalizzata utilizzando lo `Slider`, il `Pushbutton`, il `Text` ed il `Radiobutton`. Il listato che segue (`manuale_gui.sci`) permette di costruire una finestra in cui, nella parte destra, c'è un semplice grafico di una sinusoide e, nella parte sinistra, lo spazio per alcuni comandi con cui modificare interattivamente il grafico. La figura 3.21 mostra la GUI all'avvio.

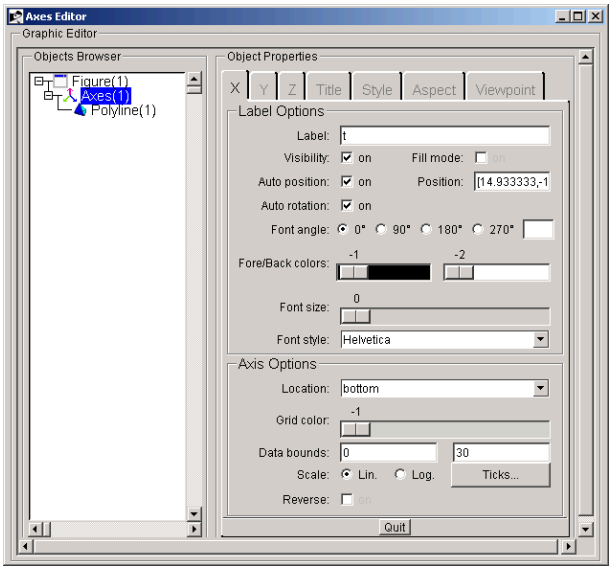


Figura 3.20: La finestra di dialogo `Axes Editor`.

```
//
// exec('manuale_gui.sci');
//


// aggiorna grafico per movimento slider
function update_slider()

  mydraw();

endfunction

// aggiorna grafico per linea solida/tratteggiata
function update_radio()

  mydraw();

endfunction

// disegna plot
function mydraw()

  t = linspace(0,7,200)

  hf = gcf();
  hf.immediate_drawing="off";
  h=gca();
  if (h.children~=[])
    delete(h.children);
  end
  w = h_slider.value/10;
  plot2d(t,sin(w.*t));
```

```
  h.axes_bounds=[0.15,0,.9,1]
  xtitle("sin␣(wt)")
  xlabel("time␣[s]")
  h.font_size = 4;
  h.x_label.font_size = 4;
  h.title.font_size = 4;

  if (h_radio.value==0)
    h.children.children.polyline_style=1;
  else
    h.children.children.polyline_style=2;
  end

  hf.immediate_drawing="on";

endfunction

// disegna plot iniziale
function myInitDraw()

  t = linspace(0,7,200)

  w = 5;
  plot2d(t,sin(w.*t));
  h = gca();
  h.axes_bounds=[0.15,0,.9,1]
  xtitle("sin␣(wt)")
  xlabel("time␣[s]")
  h.font_size = 4;
  h.x_label.font_size = 4;
  h.title.font_size = 4;

endfunction


//
// MAIN
//
xdelall();
funcprot(0);

screen_size = get(0,"screensize_px");
sizex = .75*screen_size(3);
sizey = .75*screen_size(4);
h_graf = scf(0);
h_graf.figure_size = [sizex sizey];
h_graf.figure_position = [(screen_size(3)-sizex)/2 (screen_size(4)-sizey)
   /2];

myInitDraw();
testo = strcat(["w␣=␣5␣rad/s"]);

// stop
```

```
h_stop=uicontrol(h_graf,...
  "style","pushbutton",...
  "string","STOP",...
  "fontsize",14,...
  "backgroundcolor",[1 0.5 0],...
  "position",[10 sizey-210 50 50],...
  "callback","xdel(0)");

// slider
h_text_slider = uicontrol(h_graf,...
  "style","text",...
  "horizontalalignment","center",...
  "string",testo,...
  "fontsize",16,...
  "backgroundColor",[1 1 1],...
  "position",[10 sizey-280 200 20]);
h_slider=uicontrol(h_graf,...
  "style","slider",...
  "Min",0,...
  "Max",100,...
  "value",50,...
  "position",[10 sizey-250 200 20],...
  "callback", "update_slider();testo␣=␣strcat(['' w␣=␣'' ␣string(h_slider.
     value/10)␣'' ␣rad/s'']);␣h_text_slider.string=testo");

// radiobutton
h_text_radio = uicontrol(h_graf,...
  "style","text",...
  "horizontalalignment","left",...
  "string","linea␣ZOH/interpolata",...
  "backgroundColor",[1 1 1],...
  "fontsize",14,...
  "position",[40 sizey-310 170 20]);
h_radio = uicontrol(h_graf,...
  "style","radiobutton",...
  "Min",0,...
  "Max",1,...
  "value",0,...
  "backgroundColor",[1 1 1],...
  "position",[10 sizey-310 20 20],...
  "callback", "update_radio()");

//
// FINE MAIN
//
```

Per prima cosa vediamo la struttura del codice, ci sono alcune funzioni definite prima dello script ed infine i comandi dello script principale, delimitati con dei commenti come *main*. Le funzioni che si è ritenuto definire sono due funzioni *dummy* che vengono chiamate ad ogni interazione con i due comandi prescelti, uno `Slider` ed un `Radiobutton`, una funzione di inizializzazione del grafico ed una funzione di aggiornamento del grafico stesso. Per il comando `Pushbutton` non è necessario utilizzare una funzione ma si può utilizzare la sola stringa di callback.

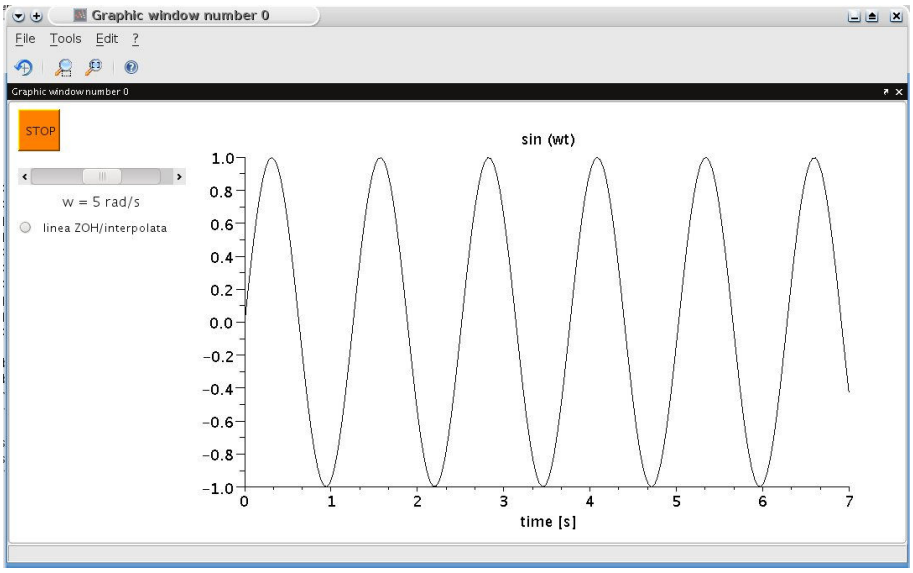


Figura 3.21: Finestra d'avvio del listato mostrato in sezione 3.6.

Le prime due funzioni del main

```
xdelall();
funcprot(0);
```

sono due comandi per chiudere tutte le finestre aperte (si veda la sezione 3.1) e per evitare messaggi di warning dovuti alla ridefinizione delle funzioni. Non sono ovviamente necessari.

I comandi successivi

```
screen_size = get(0,"screensize_px");
sizex = .75*screen_size(3);
sizey = .75*screen_size(4);
h_graf = scf(0);
h_graf.figure_size = [sizex sizey];
h_graf.figure_position = [(screen_size(3)-sizex)/2 (screen_size(4)-sizey)
   /2];
```

servono per definire la finestra grafica in cui costruire la GUI legandola alle dimensioni dello schermo attualmente in uso e centrando la finestra stessa. Dopo aver inizializzato il grafico tramite `myInitDraw()` si definiscono i comandi necessari, il primo è il `Pushbutton`:

```
// stop
h_stop=uicontrol(h_graf,...
  "style","pushbutton",...
  "string","STOP",...
  "fontsize",14,...
  "backgroundcolor",[1 0.5 0],...
  "position",[10 sizey-210 50 50],...
  "callback","xdel(0)");
```

in cui si deve fornire *l'handle* alla finestra grafica in cui definire l'oggetto, il tipo di oggetto, parametri di formattazione ed infine la *callback*, ossia il comando da eseguire quando l'utente

clicca su pulsante stesso. In questo caso il comando è una banale chiusura dell'unica finestra aperta: `xdel(0)`.

La definizione degli altri oggetti è concettualmente analoga, l'unica differenza è che nell'esempio il callback richiama una funzione d'utente (inutile in questo caso specifico). È di interesse vedere alcuni comandi della funzione `MyDraw()` che viene richiamata ogni volta che l'utente interagisce con gli oggetti grafici. La prima cosa da notare è che tale funzione inviluppa tutti i comandi grafici in questo modo:

```
hf = gcf();
hf.immediate_drawing = "off";
...
// comandi
...
hf.immediate_drawing = "on";
```

è bene conservare questa struttura per non avere problemi di resa dell'immagine (sfarfallio e non solo). Un altro comando notevole serve per cancellare la curva attualmente presente sullo schermo e si ottiene tramite:

```
h=gca();
if (h.children~=[])
  delete(h.children);
end
```

in cui si utilizza il concetto di *children* presentato nella sezione 3.4.

La cosa importante da osservare è che la funzione `MyDraw()` accede allo stato degli oggetti grafici è può quindi utilizzarlo per i suoi scopi. In questo semplice esempio si utilizza il valore dello `Slider` cui si accede tramite la variabile `h_slider.value` e del `Radiobutton` tramite la variabile `h_radio.value`.

La figura 3.22, infine, riporta lo stato della GUI dopo aver modificato lo `Slider` e selezionato il `Radiobutton`.

Un elenco di comandi per gestire una GUI è:

```
about      - show 'about scilab' dialog box
addmenu    - interactive button or menu definition
clipboard - Copy and paste strings to and from the system clipboard.
close      - close a figure
delmenu    - interactive button or menu deletion
exportUI   - Call the file export graphical interface
figure     - create a figure
findobj    - find an object with specified property
gcbo       - Handle of the object whose callback is executing.
getcallbackobject - Return the handle of the object whose callback is
    executing.
getinstalledlookandfeels - returns a string matrix with all Look and
    Feels.
getlookandfeel  - gets the current default look and feel.
getvalue        - xwindow dialog for data acquisition
messagebox      - Open a message box.
printfigure     - Opens a printing dialog and prints a figure.
printsetupbox   - Display print dialog box.
```

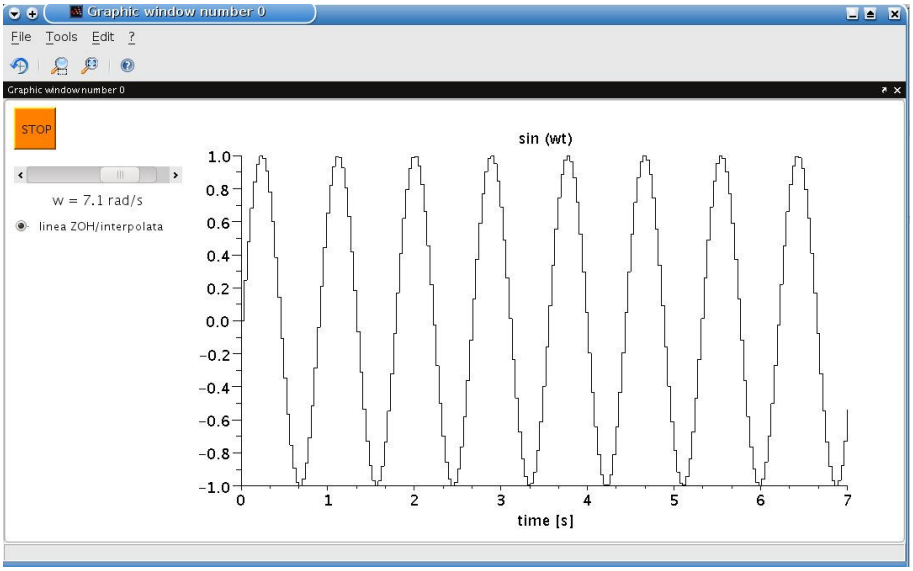


Figura 3.22: Stato della GUI dopo aver modificato lo `Slider` e selezionato il `Radiobutton`.

```
progressionbar  - Draw a progression bar
root_properties - description of the root object properties.
setlookandfeel  - sets the current default look and feel.
setmenu         - interactive button or menu activation
toolbar         - show or hide a toolbar
toprint         - Send text or figure to the printer.
uicontrol       - create a Graphic User Interface object
uigetcolor      - Opens a dialog for selecting a color.
uigetdir        - dialog for selecting a directory
uigetfile       - dialog window to get a file(s) name(s), path and filter
    index
uigetfont       - Opens a dialog for selecting a font.
uimenu          - Create a menu or a submenu in a figure
unsetmenu       - interactive button or menu or submenu de-activation
usecanvas       - Get/Set the main component used for Scilab graphics.
waitbar         - Draw a waitbar
x_choices       - interactive Xwindow choices through toggle buttons
x_choose        - interactive window choice (modal dialog)
x_choose_modeless - interactive window choice (not modal dialog)
x_dialog        - Xwindow dialog
x_matrix        - Xwindow editing of matrix
x_mdialog       - Xwindow dialog
x_message       - X window message
x_message_modeless - X window modeless message
xgetfile        - dialog to get a file path
```

3.7 Creare animazioni

Per creare delle animazioni evitando lo sfarfallio dell'immagine si deve ricorrere alla nota tecnica di grafica del doppio buffer. Il modo più semplice, e non ottimizzato per ottenere una semplice animazione (una palla rossa che si muove sullo schermo) è dato dal codice seguente (`manuale_animazione.sce`). Si nota come siano necessarie solo alcunicomandi relativi all'attivazione del double buffering, la pulizia della figura ad ogni iterazione e l'esplitica richiesta di aggiornare la pixmap.

```
N  = 1000;
r1 = .2; r2 = .2;
scf();
f = gcf();
f.pixmap = "on";   // double buffer
for i=1:N
    clf();
    plot2d(%inf,%inf, frameflag=3, rect=[-2,-2,2,2], axesflag=0)
    xtitle('Animazione')
    theta = i*2*%pi/N;
    theta2 = i*20*%pi/N;
    c = [cos(theta)+r2*cos(theta2) sin(theta)+r2*sin(theta2)];
    xfarcs([c(1)-r1 c(2)+r1 2*r1 2*r1 0 360*64]',5);
    show_pixmap();
end
f.pixmap = 'off';
```

3.8 Rassegna dei comandi grafici

Un'utile rassegna così come riportata nella guida in linea Scilab:

- 2d plotting

```
  plot2d      : plot a curve
  plot2d2     : plot a curve as step function
  plot2d3     : plot a curve with vertical bars
  plot2d4     : plot a curve with arrows
  fplot2d     : plot a curve defined by a function
  champ       : 2D vector field
  champ1      : 2D vector field with colored arrows
  fchamp      : direction field of a 2D first order ODE
  contour2d   : level curves of a surface on a 2D plot
  fcontour2d  : level curves of a surface defined by a function on a 2D
      plot
  grayplot    : 2D plot of a surface using colors
  fgrayplot   : 2D plot of a surface defined by a function using colors
  Sgrayplot   : smooth 2D plot of a surface using colors
  Sfgrayplot  : smooth 2D plot of a surface defined by a function using
      colors
  xgrid       : add a grid on a 2D plot
  errbar      : add vertical error bars on a 2D plot
  histplot    : plot a histogram
  Matplot     : 2D plot of a matrix using colors
```

- 3d plotting

```
plot3d   : plot a surface
plot3d1  : plot a surface with gray or color level
fplot3d  : plot a surface defined by a function
fplot3d1 : plot a surface defined by a function with gray or color
   level
param3d  : plot one curve
param3d1 : plots curves
contour  : level curves on a 3D surface
fcontour : level curves on a 3D surface defined by a function
hist3d   : 3D representation of a histogram
genfac3d : compute facets of a 3D surface
eval3dp  : compute facets of a 3D surface
geom3d   : projection from 3D on 2D after a 3D plot
```

- Line and polygon plotting

```
xpoly   : draw a polyline or a polygon
xpolys  : draw a set of polylines or polygons
xrpoly  : draw a regular polygon
xsegs   : draw unconnected segments
xfpoly  : fill a polygon
xfpolys : fill a set of polygons
```

- Rectangle plotting

```
xrect  : draw a rectangle
xfrect : fill a rectangle
xrects : draw or fill a set of rectangles
```

- Arc plotting

```
xarc   : draw a part of an ellipse
xarcs  : draw parts of a set of ellipses
xfarc  : fill a part of an ellipse
xfarcs : fill parts of a set of ellipses
```

- Arrow plotting

```
xarrows : draw a set of arrows
```

- Strings

```
xstring   : draw strings
xstringl  : compute a box which surrounds strings
xstringb  : draw strings into a box
xtitle    : add titles on a graphics window
titlepage : add a title in the middle of a graphics window
xinfo     : draw an info string in the message subwindow
```

- Frames and axes

```
xaxis     : draw an axis
graduate  : pretty axis graduations
plotframe : plot a frame with scaling and grids
```

- Coordinates transformations

```
isoview : set scales for isometric plot (do not change the size of
   the window)
square  : set scales for isometric plot (change the size of the
   window)
scaling : affine transformation of a set of points
rotate  : rotation of a set of points
xsetech : set the sub-window of a graphics window for plotting
subplot : divide a graphics window into a matrix of sub-windows
xgetech : get the current graphics scale
xchange : transform real to pixel coordinates
```

- Colors

```
colormap     : using colormaps
getcolor     : dialog to select colors in the current colormap
addcolor     : add new colors to the current colormap
graycolormap : linear gray colormap
hotcolormap  : red to yellow colormap
```

- Graphics context

```
xset      : set values of the graphics context
xget      : get current values of the graphics context
xlfont    : load a font in the graphics context or query loaded font
getsymbol : dialog to select a symbol and its size
```

- Save and load

```
xsave   : save graphics into a file
xload   : load a saved graphics
xbasimp : send graphics to a Postscript printer or in a file
xs2fig  : send graphics to a file in Xfig syntax
xs2gif  : send graphics to a file in Gif syntax
xs2ppm  : send graphics to a file in PPM syntax
```

- Graphics primitives

```
xbasc   : clear a graphics window and erase the associated recorded
   graphics
xclear  : clear a graphics window
driver  : select a graphics driver
xinit   : initialisation of a graphics driver
xend    : close a graphics session
xbasr   : redraw a graphics window
replot  : redraw the current graphics window with new boundaries
```

```
xpause  : suspend Scilab
xselect : raise the current graphics window
xclea   : erase a rectangle
xclip   : set a clipping zone
xdel    : delete a graphics window
winsid  : return the list of graphics windows
xname   : change the name of the current graphics window
```

- Mouse position

```
xclick    : wait for a mouse click
locate    : mouse selection of a set of points
xgetmouse : get the current position of the mouse
```

- Interactive editor

```
edit_curv : interactive graphics curve editor
gr_menu   : simple interactives graphic editor
sd2sci    : gr_menu structure to scilab instruction convertor
```

Capitolo 4

Campi di applicazione: Sistemi dinamici

4.1 Definire sistemi dinamici lineari

Il comando per definire un sistema dinamico lineare è `syslin`. Può essere utilizzato sia per definire funzioni di trasferimento che rappresentazioni in spazio di stato.

4.1.1 La funzione di trasferimento

Dal punto di vista matematico una funzione di trasferimento (fdt) è una funzione razionale fratta, non sorprenderà, quindi, che la sua definizione sia analoga a quanto visto nella sezione 2.10, e che il tipo di una fdt sia proprio un `rational`. Un esempio:

```
-->s=%s;

-->num = 1+s;

-->den = (s+2)*(s+3);

-->Stf = syslin('c',num,den)
 Stf  =

      1 + s
    ---------
              2
    6 + 5s + s

-->typeof(Stf)
 ans  =

 rational
```

L'opzione `'c'` del comando `syslin` indica la definizione di un sistema tempo continuo, per una fdt tempo discreto si ha, analogamente:

```
-->z=%z;

-->Stfd = syslin('d',1,z-0.5)
 Stfd  =

        1
    -------
  - 0.5 + z

-->typeof(Stfd)
 ans  =

 rational
```

4.1.2 La rappresentazione in spazio di stato

La definizione di un sistema dinamico lineare in forma di spazio di stato passa sempre attraverso l'uso del comando `syslin` ma con una sintassi differente: `syslin('c',A,B,C,D)`. Ad esempio, il sistema:

$$\begin{cases} \dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}u \\ y &= \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}u \end{cases}$$

con

$$\begin{aligned} \boldsymbol{A} &= \begin{bmatrix} -5 & -1 \\ 6 & 0 \end{bmatrix} \\ \boldsymbol{B} &= \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ \boldsymbol{C} &= \begin{bmatrix} -1 & 0 \end{bmatrix} \\ \boldsymbol{D} &= 0 \end{aligned}$$

con condizioni iniziali $\boldsymbol{x}(0) = \begin{bmatrix} 0 & 0 \end{bmatrix}^{\mathrm{T}}$, si ottiene tramite il codice:

```
-->A = [-5 -1
-->      6  0];
-->B = [-1; 1];
-->C = [-1 0];
-->D =0;

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

       Sss(1)   (state-space system:)

!lss  A  B  C  D  X0  dt  !

       Sss(2) = A matrix =
  - 5.   - 1.
    6.     0.
```

```
        Sss(3) = B matrix =
  - 1.
    1.

        Sss(4) = C matrix =
  - 1.    0.

        Sss(5) = D matrix =
    0.

        Sss(6) = X0 (initial state) =
    0.
    0.

        Sss(7) = Time domain =
 c

-->typeof(Sss)
 ans  =

 state-space
```

dove non è stato necessario specificare lo stato iniziale posto nullo per default.

4.1.3 Conversioni fra spazio di stato e funzione di trasferimento

È possibile convertire un sistema dinamico dalla sua forma in spazio di stato alla rappresentazione in forma razionale e viceversa. La conversione non è indolore e si rimanda il lettore ad un testo di sistemi dinamici per approfondimenti. I comandi Scilab corrispondenti sono `tf2ss` e `ss2tf`. I due esempi precedenti sono sistemi dinamici con lo stesso comportamento ingresso-uscita, ci si aspetta, quindi, che le due conversioni forniscano gli stessi coefficienti usati nella definizione. Questo è apparentemente smentito:

```
-->ss2tf(Sss)
 ans  =

      1 + s
    ---------
               2
    6 + 5s + s


-->tf2ss(Stf)
 ans  =


        ans(1)   (state-space system:)

!lss   A   B   C   D   X0   dt   !

        ans(2) = A matrix =
  - 5.   - 1.
```

```
    6.   - 4.441D-16

       ans(3) = B matrix =
  - 1.
    1.

       ans(4) = C matrix =
  - 1.     1.110D-16

       ans(5) = D matrix =
    0.

       ans(6) = X0 (initial state) =
    0.
    0.

       ans(7) = Time domain =
 c
```

si nota, infatti, la presenza di due termini che ci si aspettava fossero nulli ed assumono un valore di ordine di grandezza 10^{-16}. Si intuisce che si tratta di un valore *piccolo*, praticamente nullo, ma prima di considerarlo tale è bene leggere la sezione 2.23 relativa all'accuratezza numerica.

4.1.4 Rappresentazioni minime

Il comando `minreal` permette di calcolare la rappresentazione minima bilanciata di un sistema lineare nella rappresentazione in spazio di stato. Lo stesso comando può anche essere utilizzato per le funzioni di trasferimento ed effettua delle cancellazioni fra poli e zeri uguali. Nell'esempio che segue la fdt $F1$ ha un polo ed uno zero in $s = 2$, Scilab esegue la cancellazione dopo aver convertito la fdt in spazio di stato (anche l'uscita di `minreal` è in spazio di stato):

```
-->s=%s;
-->num=(s-2)*(s-3);den=(s-2)*(s-4)*(s-5);
-->F1=syslin('c',num,den)
 F1  =
                    2
         6 - 5s + s
    ------------------
                    2   3
  - 40 + 38s - 11s + s

-->F2=minreal(tf2ss(F1));
-->ss2tf(F2)
 ans  =

     - 3 + s
    -----------
               2
    20 - 9s + s
```

Il comando per la realizzazione minima è `minss`.

4.1.5 Estrarre informazioni da una variabile rappresentante un sistema lineare

Sia che il nostro sistema lineare sia definito come fdt che come spazio di stato può essere utile estrarne delle informazioni. per quanto concerne l'accesso al numeratore, al denominatore, ai poli o agli zeri della fdt la procedura è identica a quella già vista per le funzioni razionali. Quindi, per esempio, i poli di `Stf` sono accessibili con:

```
-->poli = roots(Stf.den)
 poli  =

  - 2.
  - 3.
```

Se il sistema è stato definito tramite una variabile di tipo `state-space` è possibile accedere alle sue matrici $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$, oltre ad un eventuale stato iniziale:

```
-->A = Sss.A
 A  =
  - 5.   - 1.
    6.     0.

-->typeof(A)
 ans  =
 constant
```

si noti come il tipo della variabile `A` sia `constant`, vale a dire una matrice su cui è possibile effettuare le operazioni corrispondenti.

È anche possibile estrarre tutte le matrici di un sistema dinamico in spazio di stato tramite il comando `abcd`:

```
-->[A,B,C,D]=abcd(Sss)
 D  =
    0.

 C  =
  - 1.     0.

 B  =
  - 1.
    1.

 A  =
  - 5.   - 1.
    6.     0.
```

4.1.6 Visualizzare sistemi in spazio di stato

Per visualizzare sistemi in spazio di stato esiste un comando che fornisce un'uscita più leggibile rispetto a quanto visto negli esempi precedenti:

```
-->ssprint(Sss)
```

```
.    |-5  -1 |     |-1 |
x = | 6   0 |x + | 1 |u

y = |-1   0 |x
```

oppure, nel tempo discreto:

```
-->ssprint(tf2ss(Stfd))

+
x = | 0.5 |x + | 1 |u

y = | 1 |x
```

4.2 La tabella di Routh-Hurwitz

Il comando `routh_t` permette il calcolo della tabella di Routh-Hurwitz utile ai fini dell'analisi di stabilità di sistemi dinamici per i cui dettagli si rimanda alla corrispondente letteratura.

L'esempio che segue mostra la tabella applicata ad un polinomio, vale a dire il denominatore della funzione di trasferimento di un sistema lineare:

```
-->s=%s;

-->d = (s+3)*(s-1)
 d  =

               2
  - 3 + 2s + s

-->routh_t(d)
 ans  =

    1.   - 3.
    2.     0.
  - 3.     0.
```

È anche possibile utilizzare il comando per calcolare la tabella di un sistema chiuso in retroazione lasciando un parametro k come grado di libertà. In questo caso l'ingresso deve essere una variabile di tipo `rational`, vale a dire una funzione di trasferimento. Supponendo di utilizzare il denominatore appena definito, un esempio di applicazione è dato da:

```
-->F = 1/d;

-->k=poly(0,'k')
 k  =

    k

-->routh_t(F,k)
```

```
 ans   =

    1          - 3 + k

    2            0

  - 6 + 2k       0
```

da cui si evince che, con $k > 3$, il sistema chiuso in retroazione può essere reso asintoticamente stabile.

4.3 Rappresentazioni grafiche sul piano complesso

4.3.1 La mappa poli-zeri

È spesso utile avere una rappresentazione grafica degli zeri e dei poli di un sistema dinamico. Il comando `plzr` accetta in ingresso sia variabili di tipo `rational` che `state-space`. Se seguito dal comando `sgrid`, inoltre, disegna dei luoghi geometrici utili alla lettura delle costanti di tempo e dei coefficienti di smorzamento. Il risultato del codice:

```
-->plzr(Ssss)
-->sgrid
```

è mostrato in figura 4.1. Si noti come, nella sola versione 5.2, la legenda fosse sbagliata, baco non presente nelle versioni precedenti ed in quella corrente.

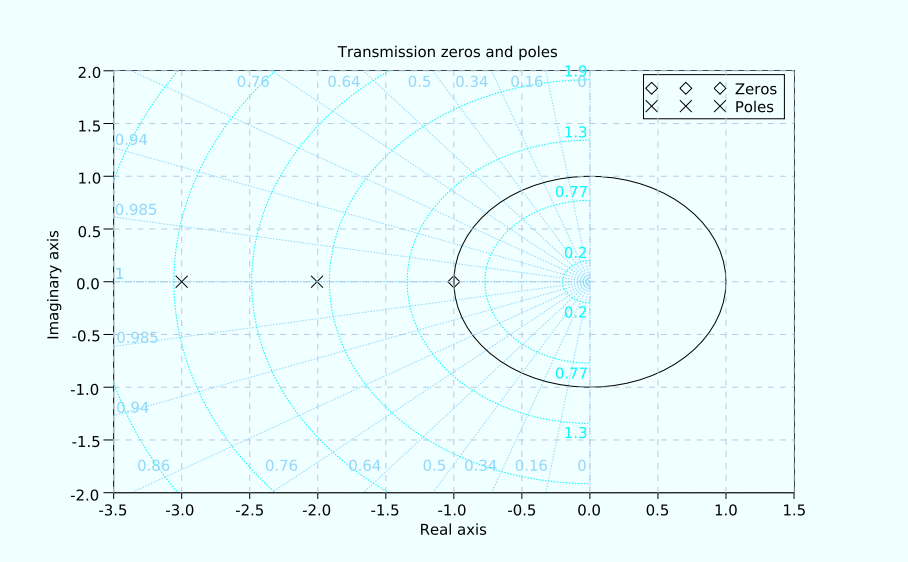


Figura 4.1: Esempio dell'uso del comando `plzr`

Fino alla versione 4.1.1 questo comando era abbastanza minimale, si noti come, per il caso tempo continuo, in assenza del comando `sgrid` venga disegnata anche la circonferenza unitaria priva di utilità.

Per il tempo discreto si utilizza lo stesso comando per disegnare la mappa ma il comando `zgrid` per la griglia come mostrato in figura 4.2 relativa all'esempio:

```
-->plzr(Stfd)
-->zgrid
```

in cui l'unico polo è visibile in $z = 0.5$.

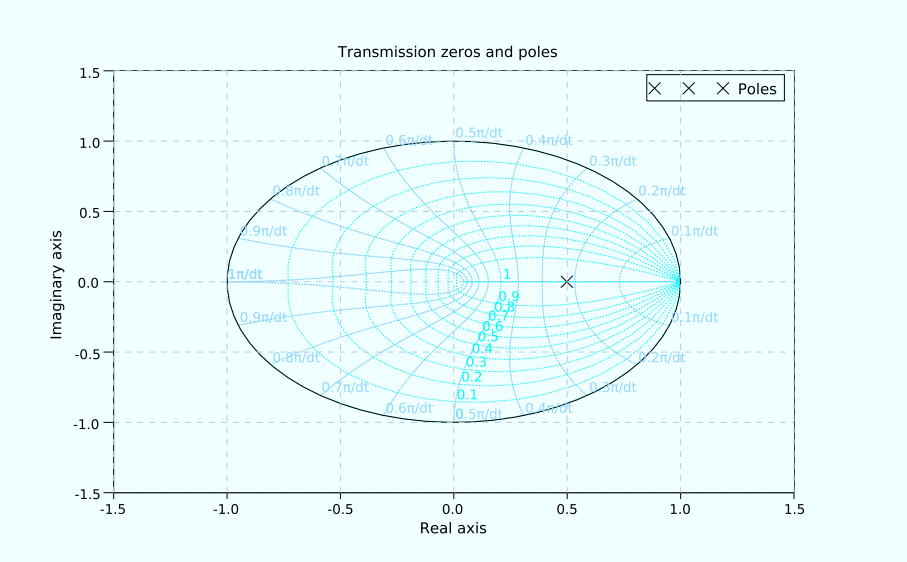


Figura 4.2: Esempio dell'uso del comando `plzr` per il caso tempo discreto

4.3.2 Il luogo delle radici

Il luogo delle radici è il luogo geometrico dei punti per cui

$$1 + kF(s) = 0$$

con $F(s)$ funzione di trasferimento d'anello. Si può graficamente rappresentare tramite il comando `evans`, come riportato nell'esempio che segue, relativo alla fdt d'anello

$$F(s) = \frac{s + 10}{(s + 2)(s + 3)}$$

con la corrispondente figura 4.3:

```
s=%s;
num = 10+s;
den = (s+2)*(s+3);
Stf = syslin('c',num,den)
evans(Stf)
```

Anche per questo comando la legenda era sbagliata nella sola versione 5.2. La scelta degli assi per la visualizzazione di un luogo delle radici non è sempre ottimale e può quindi essere necessario ricorrere allo strumento *zoom* per inquadrare la regione di interesse del luogo delle radici. Capita,

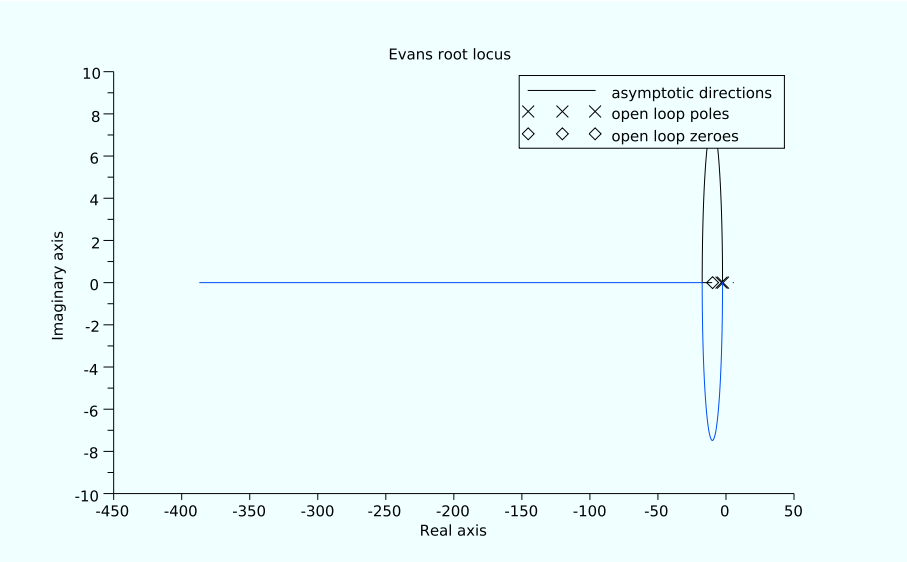


Figura 4.3: Luogo delle radici di una fdt con due poli ed uno zero

inoltre, che le etichette possano coprire parte del grafico di interesse. Nel caso in esame una scelta degli assi probabilmente più opportuna sarebbe stata quella mostrata in figura 4.4.

Un altro problema relativo al comando `evans` è dato dalla discretizzazione scelta da Scilab per il disegno. Il comando nell'aiuto in linea, infatti, viene presentato come

```
evans(H [,kmax])
```

da cui si evince come il parametro `kmax` sia facoltativo. Il valore di default non sempre è soddisfacente. Si potrebbe rimediare utilizzando la sintassi che permette di imporre il massimo k; questo però, richiede di sapere *a-priori* che il grafico era sbagliato...

Le insidie di questo comando possono essere apprezzate in maniera specifica dai 4 grafici in figura 4.5 in cui sono riportari i luoghi delle radici delle fdt:

$$P_1 = \frac{s+1}{s(s+0.1)} \quad P_2 = \frac{s+10}{s(s+1)} \quad P_3 = \frac{s+100}{s(s+10)} \quad P_2 = \frac{s+1000}{s(s+100)}.$$

Dalla teoria è noto che *qualitativamente* dovrebbero apparire indistinguibili mentre l'uscita del comando dà risultati spiacevolmente diversi fra loro.

La rappresentazione del luogo delle radici in z è formarmente identica al caso in s ed utilizza quindi gli stessi comandi.

4.3.3 Sintesi tramite il luogo delle radici

Il comando `evans` può offrire un'utile strumento numerico per tarare la sintesi di un controllore tramite il luogo delle radici. Una volta disegnato il luogo delle radici, infatti, è necessario sapere per quale valore di k i poli a ciclo chiuso si trovano in un determinato punto.

Ci sono due possibili strade grafiche, la prima passa attraverso l'utilizzo del comando `horner`, al seconda attraverso lo strumento `datatip` discusso in sezione xxx.

Senza entrare nel dettaglio algoritmico, la prima strada prevede l'utilizzo del comando:

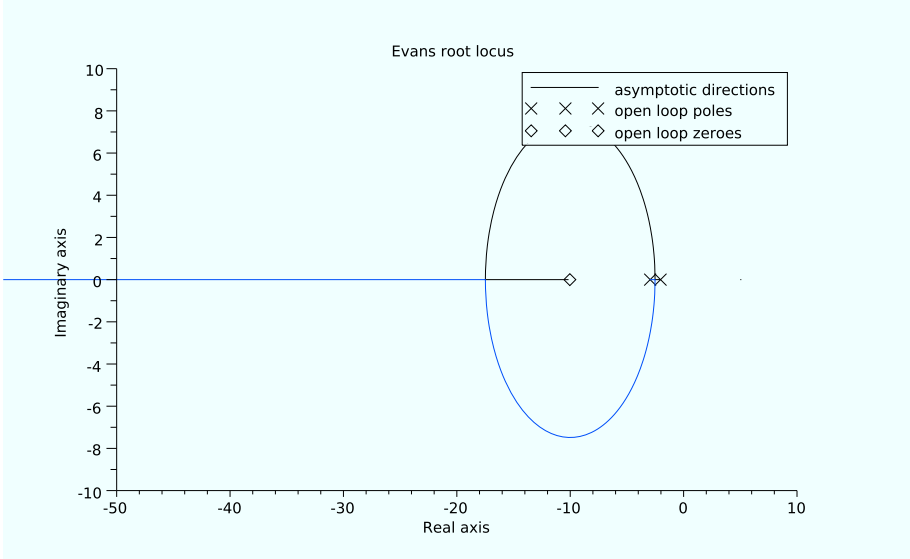


Figura 4.4: Luogo delle radici di una fdt con due poli ed uno zero con un diverso intervallo degli assi

```
-->k=-1/real(horner(Stf,[1,%i]*locate(1)))
```

che permette di cliccare sul grafico nel punto desiderato (comando `locate`) ed utilizzare questo punto per risolvere in k l'equazione $1 + kF(s) = 0$. Non è necessario cliccare *esattamente* sulla curva ma può essere utile allargare prima gli assi con lo strumento *zoom*.

Anche per la sintesi, il caso in z è formarmente identico al caso in s ed utilizza quindi gli stessi comandi.

4.4 Le rappresentazioni frequenziali

4.4.1 Il diagramma di Nichols

Il diagramma di Nichols rappresenta la funzione di risposta armonica $F(j\omega)$ su un piano in cui in ascissa è riportata la fase, generalmente espressa in gradi, ed in ordinata il modulo, generalmente espresso in decibel[1] (dB).

Il comando da utilizzare per rappresentare il diagramma di Nichols è `black`, per sovrimporre i luoghi a modulo e fase costante è poi possibile utilizzare il comando `chart`.

Il comando `chart()`, da solo, produce uno schema di Nichols in cui l'origine degli assi, come usuale, è posta in $(-90°, 0)$ ed in cui sono rappresentati i luoghi a modulo e fase costanti. Un esempio è riportato in figura 4.6.

Il comando `chart` può essere utilizzato anche per disegnare solo alcune curve di interesse oppure per distinguerle dalle altre, si veda l'help in linea per i dettagli. Il codice che segue mostra il

[1] $X_{\text{dB}} = 20 \log_{10}(X)$

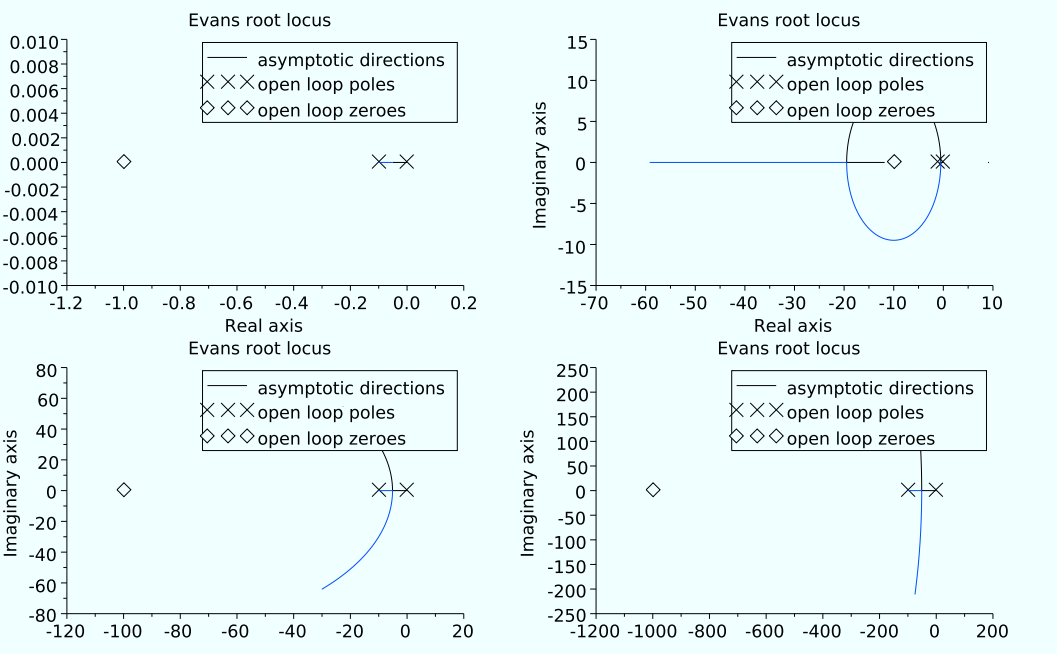


Figura 4.5: Luogo delle radici di 4 fdt che *dovrebbero* apparire qualitativamente uguali

diagramma di Nichols della funzione di risposta armonica

$$F(j\omega) = \frac{10 + j\omega}{j\omega(j\omega + 3)}$$

in cui si è deciso di disegnare in rosso la curva corrispondente al luogo a modulo costante di valore $-3\,\text{dB}$. La figura corrispondente è la 4.7:

```
s=%s;
num = 10+s;
den = s*(s+3);
Stf = syslin('c',num,den);
chart();
black(Stf)
chart(-3,[],list(1,0,5))
xgrid
```

Anche in questo caso lo strumento *zoom* permette di scegliere gli assi desiderati. Si noti come la curva sia parametrizzata in Hz. Per il tempo continuo sulla curva sono riportati alcuni valori appartenenti all'intervallo $[10^{-3}, 10^{3}]\,\text{Hz}$.

Per il tempo discreto le rappresentazioni frequenziali si calcolano per

$$z = e^{j\theta}$$

con $\theta \in]-\pi, \pi]\,\text{rad}$. Per effetto della simmetria è spesso sufficiente disegnare la curva per $\theta \in [0, \pi]\,\text{rad}$. Il comando assume in ingresso una variable del tipo

$$z = e^{2\pi j\omega T}$$

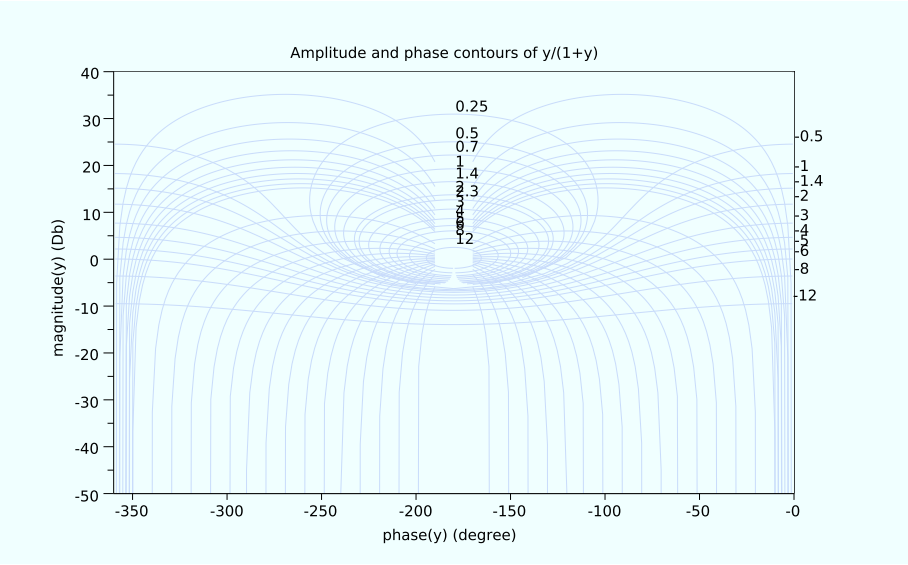


Figura 4.6: Grafico ottenuto con il comando `chart()`

vale a dire la sintassi relativa ai sistemi a dati campionati. L'ingresso del comando è ω ed è quindi espresso in Hz. L'intervallo di default è $\omega \in [10^{-3}, 0.5]$ Hz. Per sistemi tempo discreto, non originati da alcun campionamento, la variable $T = 1$ e per avere un grafico fino a $\theta = \pi$ si deve scegliere proprio $\omega = 0.5$.

4.4.2 Il diagramma di Bode

Il diagramma di Bode si ottiene tramite il comando `bode` come nell'esempio seguente:

```
s=%s;
num = 20*(10+s);
den = (s+5)*(s+3);
Stf = syslin('c',num,den);
bode(Stf)
```

che rappresenta la funzione:

$$F(j\omega) = \frac{20(10 + j\omega)}{(j\omega + 3)(j\omega + 5)}$$

ed è visualizzabile in figura 4.8. Si faccia attenzione alla scala per l'asse delle ascisse, Scilab utilizza le frequenze in Hz mentre altri programmi, ad es., Matlab, utilizzano le pulsazioni in rad/s. Questo significa che se dal diagramma si legge, ad esempio, 4, va interpretato in Hz e corrisponde a $4 \cdot 2\pi \approx 25$ rad/s; al contrario, se si cerca la pulsazione di 10 rad/s si deve cercare un ascissa del valore $10/2\pi \approx 1.6$ Hz.

Il solo diagramma dei moduli si ottiene con il comando `gainplot`. Il codice seguente riporta un caso in cui le frequenze di default di Scilab non sono ottimali per la rappresentazione della funzione. Prendendo, ad esempio, la funzione:

$$F(j\omega) = \frac{10^6 + j\omega}{10^5 + j\omega}$$

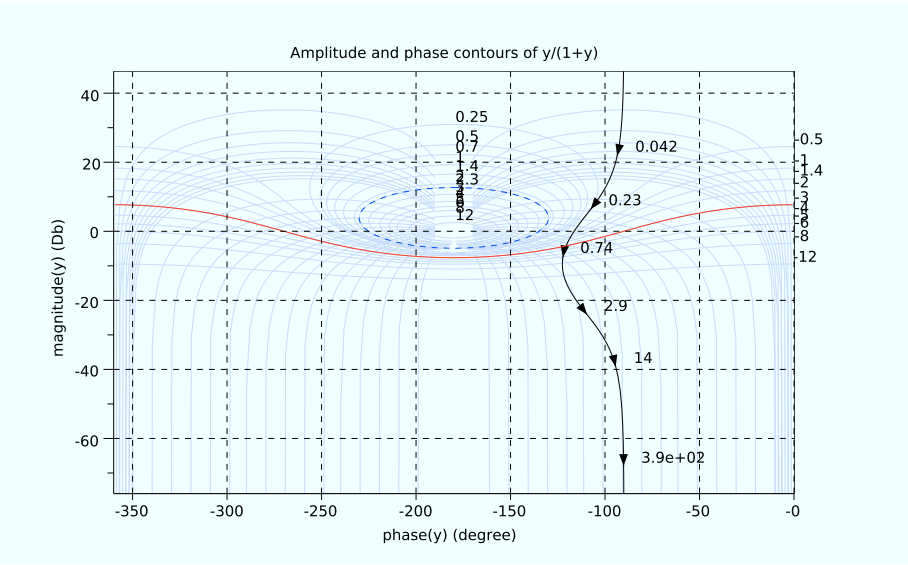


Figura 4.7: Esempio di diagramma di Nichols

disegnata con il codice

```
s=%s;
num = (s+1e6);
den = (s+1e5);
Stf = syslin('c',num,den);
gainplot(Stf)
```

da luogo alla funzione poco significativa riportata in figura 4.9.

Imponendo un intervallo opportuno, anche sulla base della conoscenza della funzione stessa, si ottiene la figura ben più significativa di figura 4.10 ottenuta tramite il codice:

```
s=%s;
num = (s+1e6);
den = (s+1e5);
Stf = syslin('c',num,den);
f=logspace(-1, 8, 50);
gainplot(Stf,f)
```

4.4.3 Il diagramma di Nyquist

Il diagramma di Nyquist, ed i corrispondenti luoghi dei punti a modulo costante, si ottengono tramite i comandi `nyquist` e `m_circle` il cui utilizzo è del tutto analogo ai comandi utilizzati per il diagramma di Nichols. Anche in questo caso, quindi, la curva è parametrizzata in Hz.

Si noti come, anche in questo caso, il grafico utilizzi un intervallo frequenziale di default che potrebbe risultare inadeguato a rappresentare opportunamente la funzione in esame. Funzioni con poli nell'origine, per il tempo continuo, o poli in 1, per il tempo discreto, risultano di difficile lettura e lo strumento *zoom* potrebbe essere necessario.

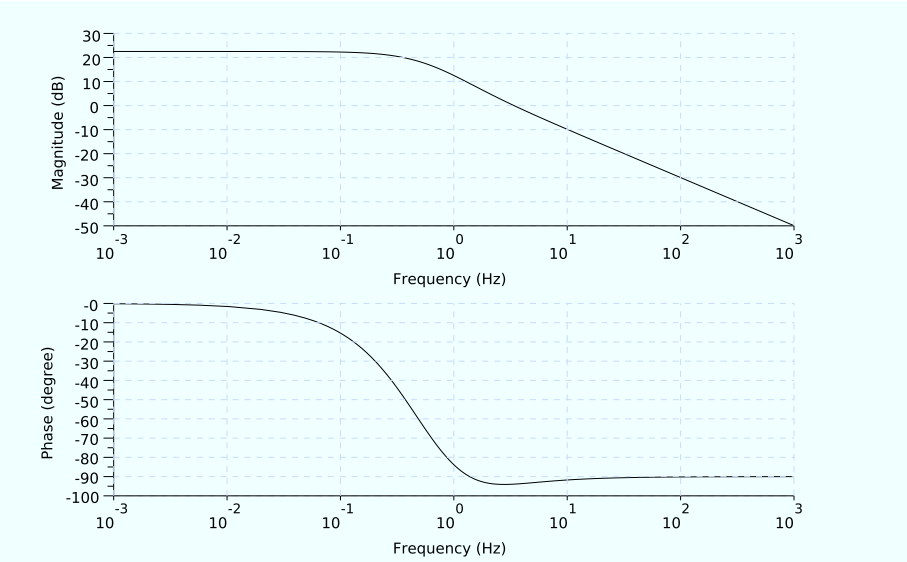


Figura 4.8: Esempio di diagramma di Bode

4.4.4 Calcolare modulo e fase in punti specifici

Il comando `horner` permette di calcolare il valore di una funzione razionale in uno o più punti. Può essere utilizzato per calcolare il valore di una funzione di risposta armonica per poi convertirlo in modulo, espresso in dB, e fase, espressa in gradi.

Nell'esempio che segue la funzione di risposta armonica

$$F(j\omega) = \frac{1}{j\omega + 3}$$

viene calcolata in $\omega = 3\,\text{rad/s}$:

```
-->s=%s;

-->num = 3;

-->den = (s+3);

-->Stf = syslin('c',num,den);

-->out = horner(Stf,3*%i)
 out  =

    0.5 - 0.5i

-->[phi,db]=phasemag(out)
 db  =

  - 3.0103
 phi  =
```

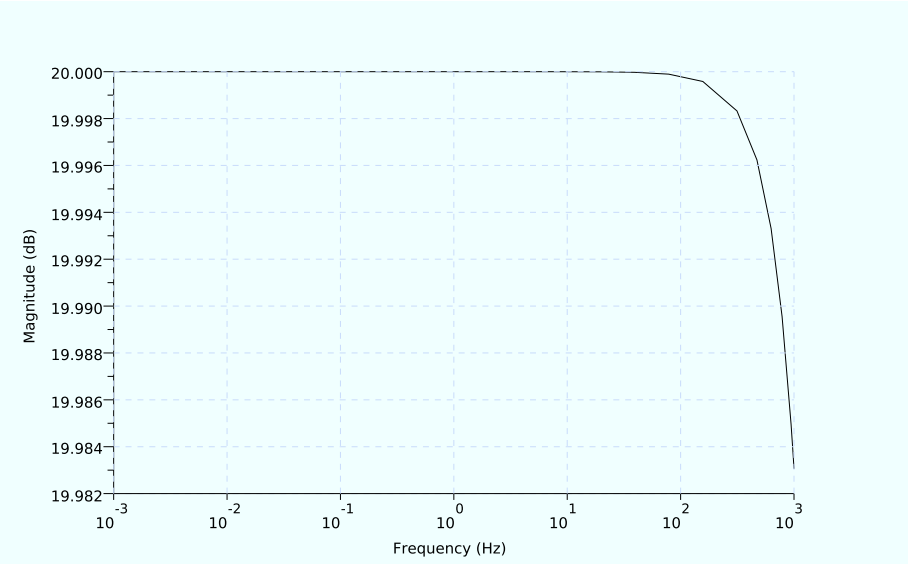


Figura 4.9: Esempio di diagramma di Bode (solo moduli) in cui l'intervallo frequenziale di default non è significativo.

```
- 45.
```

ed il risultato, come noto, è che nel punto di rottura il modulo è -3 dB sotto il regime con fase -45 gradi.

Esistono diversi comandi per ottenere informazioni numeriche sulla risposta in frequenza di una funzione di trasferimento in un intervallo di punti, in particolare si vedano `repfreq` e `dbphi`. Si consiglia di verificare sempre in che unità di misura è necessario fornire gli ingressi, se Hz o rad/s.

4.4.5 Calcolare i margini di stabilità

I comandi necessari al calcolo dei margini di stabilità sono `g_margin` e `p_margin` per il margine di guadagno e di fase, rispettivamente.

4.5 Conversione tempo continuo-discreto

È possibile convertire un sistema tempo continuo nel corrispondente tempo discreto con il comando `cls2dls`. Questo comando implementa la trasformazione bilineare:

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

in cui T è il passo di campionamento. Questo comando ammette in ingresso solo sistemi in spazio di stato e fornisce in uscita un sistema in spazio di stato:

```
Sd = cls2dls(Sc, T [,fp])
```

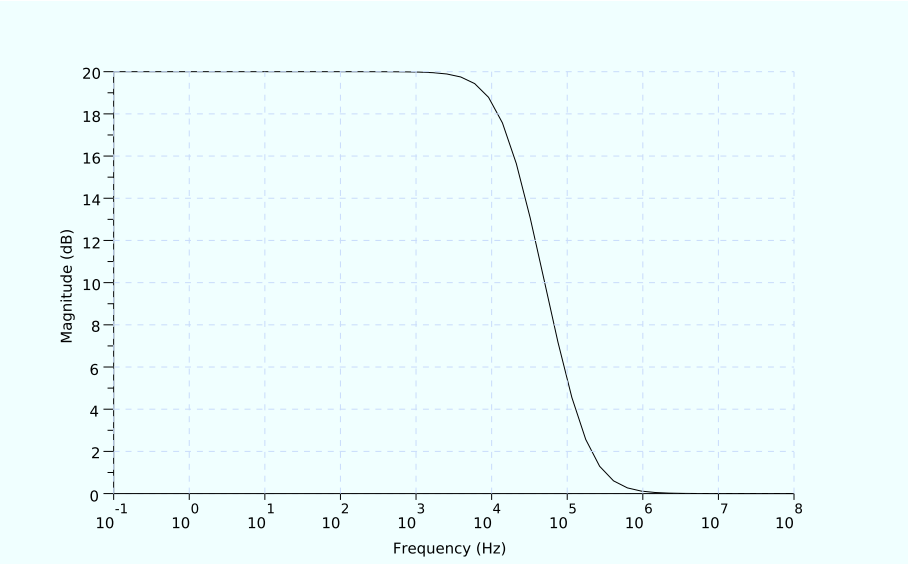


Figura 4.10: Comando `gainplot` usato sulla stessa funzione di figura 4.9 con un opportuno intervallo frequenziale.

in cui `sl` è il sistema in ingresso, `T` il passo di campionamento e l'ingresso opzionale `fp` la frequenza di prewarp.

Nel caso, frequente, in cui si stia lavorando con sistemi ingresso-uscita e quindi con funzioni di trasferimento è necessario provvedere ad una scomoda doppia conversione ed utilizzare la sintassi:

```
Fd = ss2tf(cls2dls(tf2ss(Fc),T))
```

Si noti come la conversione si possa ottenere tramite il comando `horner` descritto in Sezione 2.9.6. Il comando `horner` permette anche di implementare le trasformazioni per integrazione all'avanti e all'indietro.

4.6 Simulare un semplice sistema dinamico

Per simulare sistemi dinamici complessi è opportuno, sebbene non indispensabile, utilizzare Xcos. Per visualizzare le risposte ad ingressi canonici o meno di un sistema dinamico la cui espressione è semplice è possibile utilizzare degli appositi comandi sotto Scilab.

Nel tempo continuo il comando da utilizzare è `csim`:

```
[y [,x]]=csim(u,t,sl,[x0 [,tol]])
```

in cui

```
u   : function, list or string (control)
t   : real vector specifying times with, t(1) is the initial time ( x0=x(
   t(1)) ).
sl  : list ( syslin )
y   : a matrix such that y=[y(t(i)] , i=1,..,n
x   : a matrix such that x=[x(t(i)] , i=1,..,n
```

```
tol : a 2 vector [atol rtol] defining absolute and relative tolerances
   for ode solver (see ode)
```

Si noti come il sistema dinamico debba essere espresso in spazio di stato (e in questo caso sia possibile anche visualizzare l'andamento dello stato). L'ingresso `u` può essere una funzione, un vettore rappresentante l'ingresso nei punti corrispondenti oppure una stringa dal valore `'impulse'` o `'step'` per generare gli ingressi canonici.

Il corrispondente comando nel tempo discreto è `dsimul`, curiosamente la sintassi di questo comanto è differente dal corrispondente comando tempo continuo:

```
y=dsimul(sl,u)
```

in cui `sl` è ancora un sistema dinamico in spazio di stato ed `u` l'ingresso. In questo caso, però, l'unico ingresso ammissibile è un vettore di dimensioni opportune con cui costruire, oltre ad un ingresso generico, anche gli ingressi canonici. Questo comando fornisce solo l'uscita, per avere l'andamento dello stato è necessario utilizzare il comando:

```
[X]=ltitr(A,B,U,[x0])
[xf,X]=ltitr(A,B,U,[x0])
```

4.7 Le equazioni differenziali

I programmi di calcolo numerico con funzioni dedicate ai sistemi dinamici sono corredate da una libreria di funzioni dedicate alla soluzione di equazioni differenziali. In Scilab un comando in tal senso è `ode`. Per i dettagli si rimanda all'aiuto in linea o a testi specifici, per la simulazione di sistemi dinamici semplici si suggerisce di utilizzare i comandi più semplici appena esposti oppure Xcos, descritto nel Capitolo 5.

4.8 Rassegna sui comandi per i sistemi dinamici

- Una rassegna sui disponibili per i sistemi dinamici ed il controllo:

```
  abcd         - state-space matrices
  abinv        - AB invariant subspace
  arhnk        - Hankel norm approximant
  arl2         - SISO model realization by L2 transfer approximation
  balreal      - balanced realization
  bilin        - general bilinear transform
  cainv        - Dual of abinv
  calfrq       - frequency response discretization
  canon        - canonical controllable form
  cls2dls      - bilinear transform
  colregul     - removing poles and zeros at infinity
  cont_frm     - transfer to controllable state-space
  cont_mat     - controllability matrix
  contr        - controllability, controllable subspace, staircase
  contrss      - controllable part
  csim         - simulation (time response) of linear system
```

```
  ctr_gram    - controllability gramian
  dbphi       - frequency response to phase and magnitude
     representation
  ddp         - disturbance decoupling
  des2tf      - descriptor to transfer function conversion
  dscr        - discretization of linear system
  dsimul      - state space discrete time simulation
  dt_ility    - detectability test
  equil       - balancing of pair of symmetric matrices
  equil1      - balancing (nonnegative) pair of matrices
  feedback    - feedback operation
  flts        - time response (discrete time, sampled system)
  frep2tf     - transfer function realization from frequency response
  freq        - frequency response
  freson      - peak frequencies
  g_margin    - gain margin
  gfrancis    - Francis equations for tracking
  imrep2ss    - state-space realization of an impulse response
  invsyslin   - system inversion
  kpure       - continuous SISO system limit feedback gain
  krac2       - continuous SISO system limit feedback gain
  lin         - linearization
  linmeq      - Sylvester and Lyapunov equations solver
  lqe         - linear quadratic estimator (Kalman Filter)
  lqg         - LQG compensator
  lqg2stan    - LQG to standard problem
  lqr         - LQ compensator (full state)
  ltitr       - discrete time response (state space)
  markp2ss    - Markov parameters to state-space
  minreal     - minimal balanced realization
  minss       - minimal realization
  obs_gram    - observability gramian
  obscont     - observer based controller
  observer    - observer design
  obsv_mat    - observability matrix
  obsvss      - observable part
  p_margin    - phase margin
  pfss        - partial fraction decomposition
  phasemag    - phase and magnitude computation
  ppol        - pole placement
  projsl      - linear system projection
  repfreq     - frequency response
  ricc        - Riccati equation
  rowregul    - removing poles and zeros at infinity
  rtitr       - discrete time response (transfer matrix)
  sm2des      - system matrix to descriptor
  sm2ss       - system matrix to state-space
  specfact    - spectral factor
  ss2des      - (polynomial) state-space to descriptor form
  ss2ss       - state-space to state-space conversion, feedback,
     injection
  ss2tf       - conversion from state-space to transfer function
  st_ility    - stabilizability test
```

```
  stabil      - stabilization
  svplot      - singular-value sigma-plot
  sysfact     - system factorization
  syssize     - size of state-space system
  tf2ss       - transfer to state-space
  time_id     - SISO least square identification
  trzeros     - transmission zeros and normal rank
  ui_observer - unknown input observer
  unobs       - unobservable subspace
  zeropen     - zero pencil
```

Capitolo 5

Xcos

5.1 Introduzione

Xcos è un'applicazione Scilab per la simulazione grafica di sistemi dinamici. Contiene una libreria di *blocchi* che forniscono ingressi, funzioni e uscite utili per un'ampia casistica di simulazioni sia per sistemi tempo continuo che tempo discreto. È l'evoluzione di Scicos, utilizzato fino alla versione di Scilab 5.1 ed una sorte di clone di Simulink, il famoso pacchetto di Matlab.

È utile pensare a Xcos come ad una sorta di linguaggio di programmazione grafico, come tutti i linguaggi, oltre ad offrire una serie di comandi generici, permette all'utente di aggiungere blocchi personalizzati che sono concettualmente analoghi alle funzioni. Xcos, in particolare, permette di aggiungere blocchi che utilizzino il linguaggio Scilab, C o Fortran.

Un'altra utile funzione di Scilab è la generazione automatica di codice C. Una volta realizzato il proprio modello Xcos, infatti, può essere utile prevedere di utilizzarlo in applicazioni *embedded*. È possibile utilizzare Xcos per realizzare tutti i passaggi per la progettazione e l'implementazione di un sistema di controllo, dalla simulazione, alle prove HIL (Hardware-In-the-Loop) fino a generare un eseguibile in hard real-time.

Un semplice modello Xcos appare come quello mostrato in Figura 5.1.

5.1.1 Cosa significa simulare un sistema tempo continuo con un sistema digitale?

Un calcolatore è una macchina basata su tecnologia digitale, cosa vuol dire simulare un sistema dinamico tempo continuo con una macchina digitale? La simulazione consiste nell'approssimare numericamente le equazioni differenziali che caratterizzano il modello tempo continuo con delle equazioni alle differenze, o alle ricorrenze, che caratterizzano il modello tempo discreto. In quanto approssimazione, la simulazione numerica è caratterizzata da errori, un'altra fonte di errore consiste nell'utilizzare variabili implementate con un numero finito di bit per rappresentare numeri reali. Questo procedimento, noto come discretizzazione nelle applicazioni dell'Automatica, è ampiamente trattato nei libri di testo ai quali si rimanda per approfondimenti.

Per chi si appresta ad utilizzare Xcos per simulazioni di semplici modelli dinamici, magari lineari e stazionari, la discretizzazione non dovrebbe comportare nessun errore significativo a meno di non simulare sistemi che contengano dinamiche molto diverse fra di loro (autovalori o

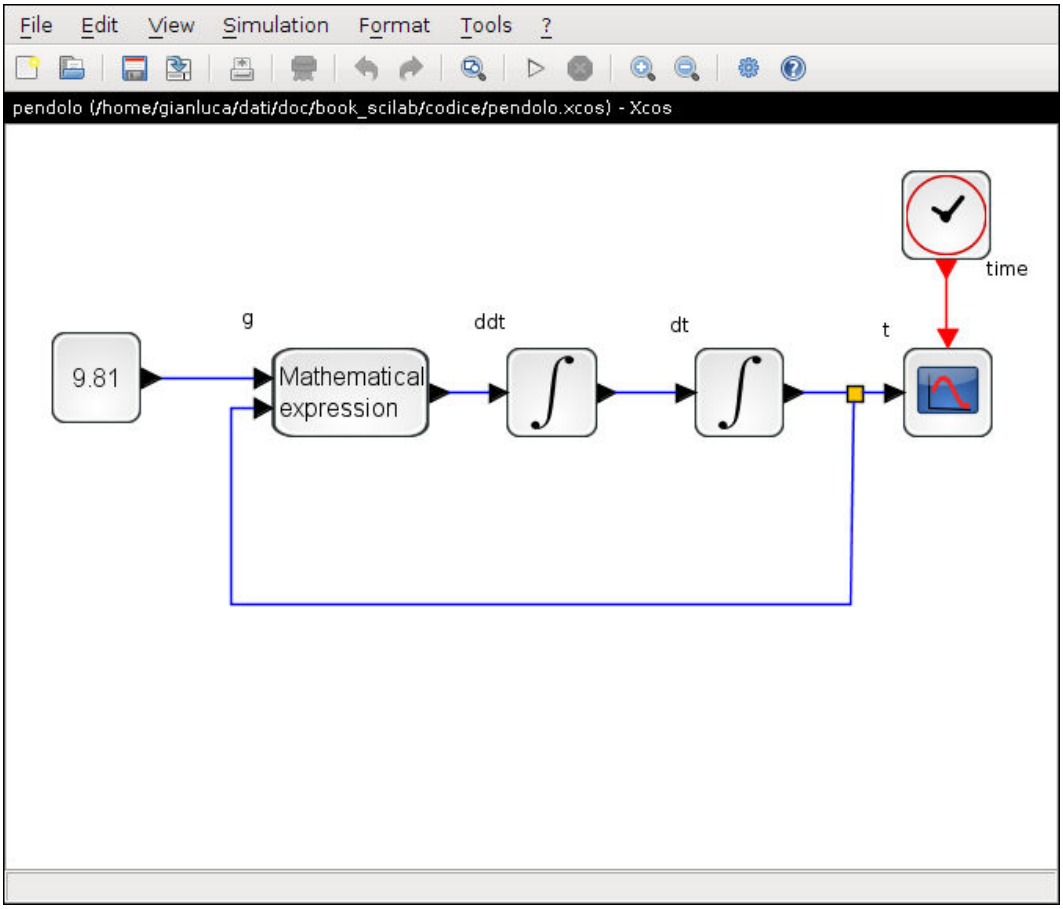


Figura 5.1: Un esempio di schema Xcos.

poli distanti fra di loro diversi ordini di grandezza).

Si noti che l'errore di discretizzazione è diverso da un'errato settaggio dell'oscilloscopio (si veda sezione 5.2.2).

5.1.2 Lanciare Xcos

Per avviare l'applicazione Xcos è sufficiente digitare, dalla linea di comando Scilab:

```
--> xcos;
```

od utilizzando il mouse seguendo il percorso `Menu -> Applications -> Xcos` come mostrato in Figura 5.2.

A questo punto si apre una finestra per la costruzione dello schema di simulazione che assume il nome di default `Untitled` (vedi Figura 5.3); tale nome può essere cambiato all'atto del salvataggio in un file di tipo `cos`; volendo, ad esempio, rinominare lo schema in `mioschema`, basterà eseguire il comando `Save As` dal menu `File` della finestra `Untitled` indicando il nome file `mioschema.cos`.

5.1.3 Cos'è un blocco?

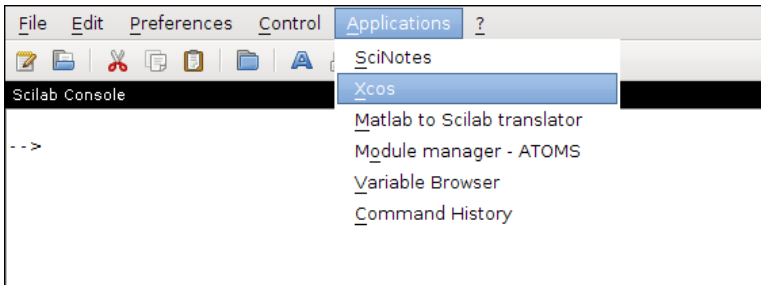


Figura 5.2: Dettaglio sulla finestra di avvio di Scilab e possibilità di avvio di Xcos.

Figura 5.3: La finestra bianca per la costruzione del modello di simulazione.

Un blocco è una funzione grafica che può essere caratterizzata dai seguenti elementi:

- Ingresso
- Stato
- Uscita
- Quantità scalari/vectoriali
- Tempo continuo/discreto
- Ingressi di attivazione
- Uscite di attivazione

dove i primi elementi sono evidenti. Gli ingressi di attivazione stanno ad indicare che la funzione può eventualmente essere valutata solo quando richiesto agendo sull'ingresso di attivazione.

Questo permette di simulare degli *eventi* non legati allo scorrere del tempo ma al valore di alcune variabili. In maniera analoga un blocco può generare un segnale che sia di attivazione per un altro blocco. Ingressi ed uscite sono rappersentati da triangoli neri, ingressi ed uscite di attivazione da triangoli rossi.

5.2 Le palette

Con il termine palette si intende la libreria di blocchi disponibile per costruire un modello Xcos.

Il *browser* delle palette si apre di default all'avvio di Xcos ed è mostrato in figura 5.4.

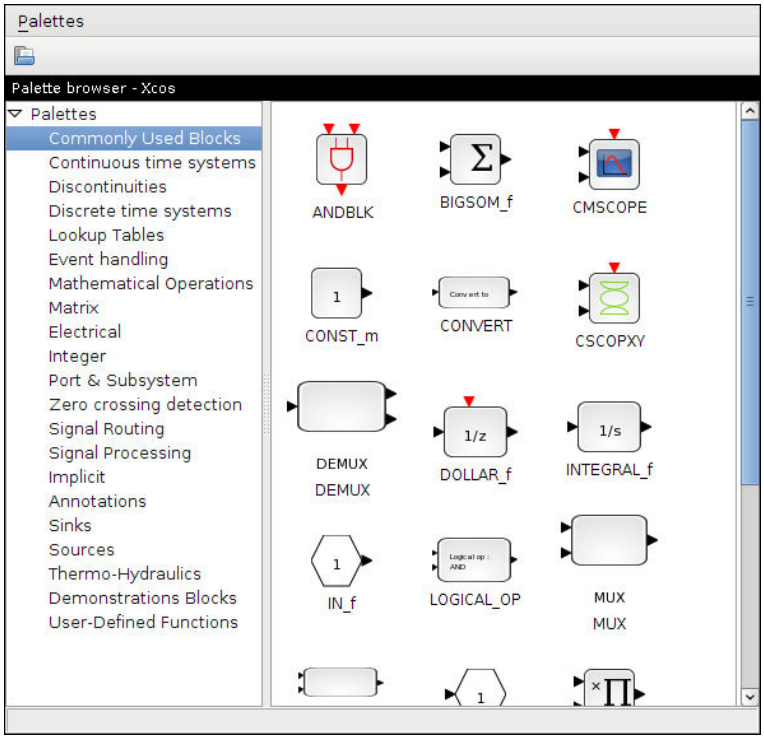


Figura 5.4: La finestra `Palette browser`.

Una volta individuato un blocco di interesse in una delle librerie disponibili, bisogna trascinarlo (ovvero effettuare un *drag-and-drop*) dalla finestra in cui è rappresentato alla propria finestra.

L'albero è organizzato secondo la funzionalità dei blocchi in: sorgenti, destinazioni, lineri, non-lineari, matriciali, interi, eventi, ecc.

5.2.1 Sorgenti

La cartella `sources`, mostrata in Figura 5.5, contiene tutti i blocchi in grado di generare segnali e quindi utili come sorgenti in uno schema Xcos. Oltre a poter generare segnali con funzioni analitiche quali la costante, la sinusoide, il dente di sega, ecc. è anche possibile caricare un segnale dalla spazio di lavoro o da un file. L'utente può quindi generare un segnale di ingresso qualsivoglia.

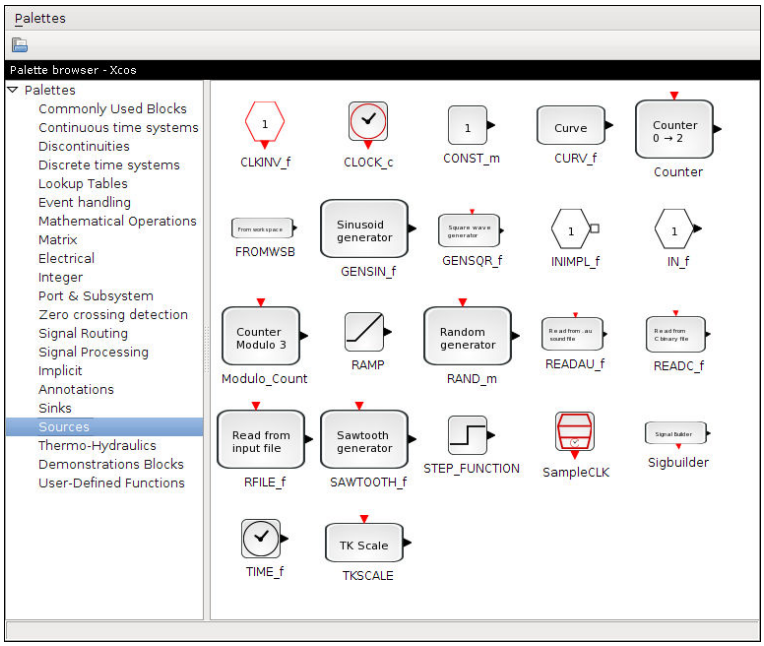


Figura 5.5: Blocchi sorgenti di Xcos.

5.2.2 Destinazioni

Le destinazioni (`sinks`) rappresentano tutti quei blocchi utili per visualizzare o salvare i risultati di una simulazione. Esistono dei blocchi grafici, quali l'oscilloscopio, o numerici, che permettono di salvare i dati nello spazio di lavoro oppure in un file.

5.2.3 Funzioni per i sistemi dinamici

Molti blocchi utili per la simulazione di sistemi dinamici lineari sono disponibili nelle cartelle `continuous time sistems` e `discrete time sistems`). È possibile, infatti, simulare il comportamento di un sistema ingrasso-stato-uscita, o ingresso-uscita, fornendo le matrici del sistema dinamico e la sua funzione di trasferimento. In questa cartella ci sono anche blocchi destinato al calcolo dell'integrale e la derivata temporale.

5.2.4 Altri blocchi

Il modo migliore per conoscere i blocchi disponibili è quello di sfogliare le cartelle e vederne il corrispondente aiuto in linea. Molti blocchi utili zono in `mathematical operations` e `signal routing`. Fra i vari blocchi si segnalano i blocchi per eseguire codice scritto in C, in Fortran o in Scilab stessoo il blocco per realizzare una qualsiasi espressione matematica.

5.3 Cambiare i parametri dei blocchi

Una volta individuato il blocco da utilizzare è sufficiente trascinarlo sullo schema d'utente e collegarlo in maniera opportuna. Collegare un blocco significa individuare quale segnale rappresenta il

suo eventuale ingresso e cosa fare della sua eventuale uscita. Il blocco, come detto, è una funzione grafica; per qualsiasi blocco, prima del suo utilizzo, vanno definiti i parametri di simulazione. Se il blocco rappresenta un ingresso sinusoidale, ad esempio, vanno scelti l'ampiezza, la frequenza e l'eventuale fase iniziale del seno. Molti blocchi hanno dei parametri di default, si deve fare attenzione a non lanciare una simulazione senza aver prima modificato i parametri di tutti i blocchi, la presenza dei parametri di default, infatti evita un errore di sintassi ma favorisce errori di semantica.

Per accedere ai parametri di un blocco si deve cliccare con il tasto destro sul blocco e scegliere `Block parameters` dal menu a tendina. In alternativa, la stessa finestra di dialogo si attiva con un doppio clik sulla corrispondente icona rappresentativa.

In Figura 5.6 è riportata la finestra di dialogo del blocco `CONST_m`; nella riga è possibile immettere direttamente il valore numerico della costante oppure un'espressione (si veda in particolare la Sezione 5.5) che ne consente il calcolo.

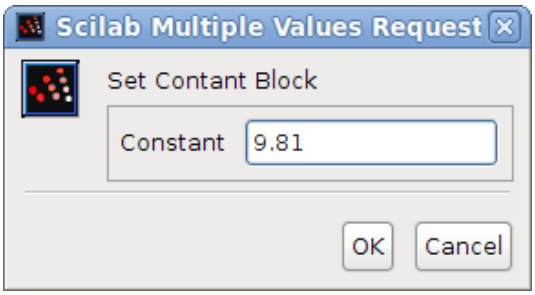


Figura 5.6: La finestra di dialogo del blocco `CONST_m`.

5.4 Opzioni di simulazione

La stessa simulazione richiede di impostare dei parametri ed anche in questo caso ogni schema nasce con dei parametri di default. Per accedere ed eventualmente modificare questi parametri si deve selezionare `Simulation->Setup` e si ottiene una finestra come quella mostrata in Figura 5.7:

Per modelli *semplici* l'unico parametro da settare è il tempo finale di simulazione (`final integration time`) lasciando inalterati i parametri di default. Per il tempo finale è opportuno fare delle considerazioni basate sulla conoscenza del modello da simulare prima di lanciare la simulazione stessa, un'errata impostazione del tempo finale, infatti, può dare origine a grafici di difficile interpretazione. Si supponga, ad esempio, di voler simulare un sistema elettromeccanico con costante di tempo dell'ordine dei millisecondi e di settare il tempo finale in 100 secondi, ciò che si vedrà sarà il solo regime con il rischio di fraintendere il risultato della simulazione stessa. Considerazioni analoghe possono essere fatte per i parametri dell'oscilloscopio.

5.5 Gestione delle variabili

È opportuno evitare di usare delle costanti come parametri sia dei singoli blocchi che della simulazione. Per minimizzare la possibilità di errori, infatti, è bene centralizzare la definizione di tutte le variabili ed utilizzare variabili simboliche nei blocchi; ad esempio, è opportuno utilizzare

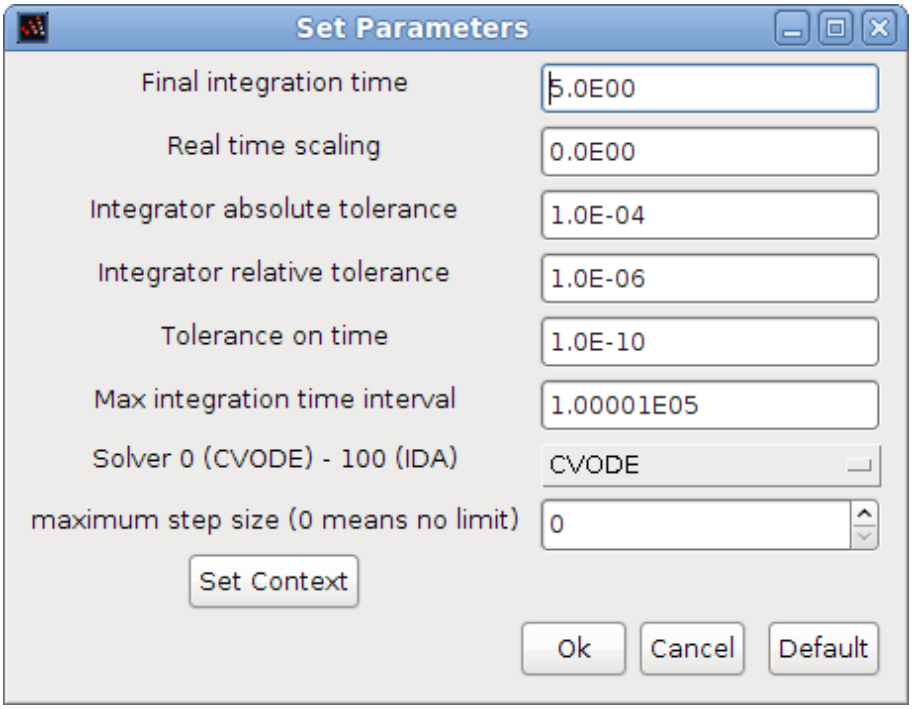


Figura 5.7: Finestra delle opzioni di simulazione.

la stessa variabile simbolico come intervallo di visualizzazione degli oscilloscopi presenti nello schema.

Un modo opportuno è quello di utilizzare il concetto di `Context` nel modo seguente:

- Si mettano tutte le variabili in un file chiamandolo, ad esempio, `var.sce`;
- Si apra `Diagram -> Context` scrivendo `exec('var.sce');`
- Si usino le variabili simboliche come parametri.

Si noti come le variabili del *context* vengano viste solo dai blocchi, non possono essere usate, quindi, per settare i parametri di simulazione visti nella sezione 5.4.

Un modo per settare il tempo finale della simulazione (`Simulation->setup`, poi agendo sulla variabile `final integration time`) è quello di utilizzare il blocco END disponibile nella cartella `sinks` della libreria di blocchi. Si immagini un file `var.sce` con la sola riga: `tf=3;`, a questo punto si usa la variabile simbolica `tf` come parametro del blocco END.

Un'ultima possibilità, infine, è quella di usare variabili globali.

5.6 Lanciare la simulazione

Per lanciare la simulazione il modo più semplice è farlo in maniera grafica selezionando il comando `Simulation->Run`.

5.7 Costruire un semplice modello

Si vuole realizzare la simulazione di un sistema dinamico tempo continuo del quale si conosce la funzione di trasferimento:

$$F(s) = \frac{1}{s^2 + s + 1}$$

Per farlo si ha bisogno del segnale di ingresso, per il quale si decide arbitrariamente di considerare un segnale costante, di un blocco che implementi la $F(s)$ e di un'uscita. In questo caso si decide per un semplice oscilloscopio.

La figura 5.5 mostra la cartella delle sorgenti dalla quale si sceglie facilmente il blocco costante e lo si *trascina* nello schema dell'utente.

In maniera analoga è possibile selezionare il blocco CLR (Continuous Transfer Function) che è rintracciabile nella cartella delle funzioni continue. Si deve infine scegliere l'oscilloscopio dalla cartella destinazioni; questo blocco ha un ingresso di attivazione al quale è opportuno connettere un *clock di attivazione* ottenendo così un campionamento regolare del segnale di uscita della funzione di trasferimento.

Lo schema costruito è mostrato in Figura 5.8.

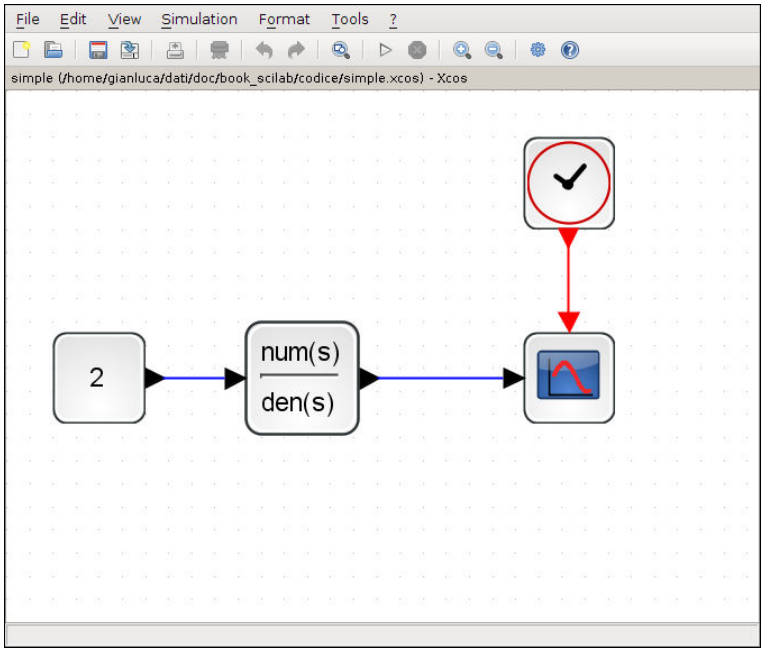


Figura 5.8: Un semplice schema Xcos.

È necessario ora impostare tutte i parametri dei blocchi e della simulazione come indicato sommariamente nelle sezioni precedenti e poi lanciare la simulazione ottenendo l'uscita grafica dell'oscilloscopio mostrata in Figura 5.9.

5.8 Importare ed esportare dati

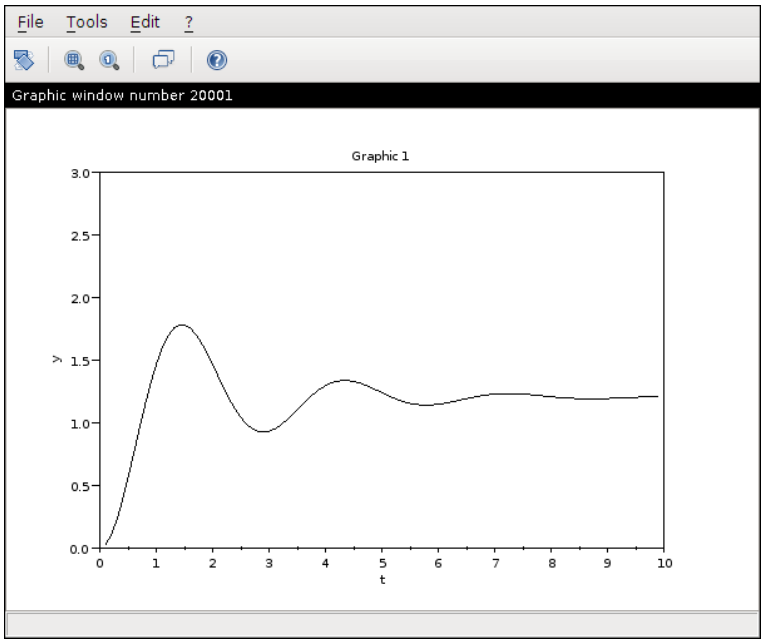


Figura 5.9: Risultato della risposta indiciale per lo schema mostrato in Figura 5.8.

Gli schemi Xcos possono utilizzare e salvare dati esterni a Xcos stesso; esistono due possibilità, interagire con un file o con la spazio di lavoro Scilab. È possibile scrivere (`WRITEC_f` e `WFILE_f`) e leggere (`READC_f` e `RFILE_f`) da file seguendo una sintassi opportuna così come è possibile scrivere e leggere in variabili dello spazio di lavoro Scilab tramite i blocchi `TOWS_c` e `FROMWS_c`.

È utile notare come, rispetto all'ultima versione di Scicos associata a Scilab 5.1, ci sia stato un'importante miglioramento nella possibilità, per Xcos, di *vedere* lo spazio di lavoro Scilabmentre Xcosè aperto. Questo vale anche per i risultati salvati da una simulazione con il blocco `TOWS_c`, ora immediatamente accessibili in Scilabpur tenendo aperto lo schema Xcoscorrispondente.

5.9 Costruire un blocco personalizzato

In maniera analoga alla programmazione tradizionale, anche in Xcos può essere utile creare dei blocchi personalizzati e quindi una libreria di blocchi concettualmente analoga ad una libreria di funzioni. Per farlo è sufficiente utilizzare i corrispondenti comandi della finestra `Palette browser`.

5.10 Il superblocco

Il concetto di superblocco indica un insieme di blocchi racchiusi in un solo blocco con opportuni ingressi ed uscite. Per creare un superblocco è sufficiente selezionare la regione di interesse, facendo attenzione a quelli che saranno gli ingressi e le uscite del superblocco, e selezionare `Diagram -> Region to Super Block`. Una volta creato un superblocco è possibile modificarlo cliccando due volte sul blocco stesso: si apre una nuova finestra che *espande* il contenuto del blocco stesso.

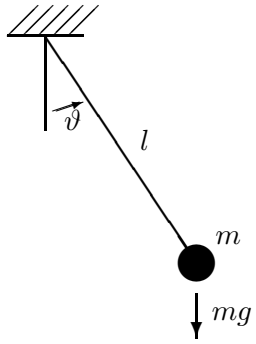


Figura 5.10: Rappresentazione schematica di un pendolo semplice.

5.11 Esempio: Studio in simulazione di un pendolo semplice

5.11.1 Modellistica del pendolo

Con riferimento alla rappresentazione schematica in Figura 5.10, si considerino i seguenti parametri:

- l — lunghezza del pendolo, supposto rigido;
- m — massa (puntiforme) concentrata all'estremità del pendolo;
- μ — momento di inerzia di m rispetto al punto di ancoraggio del pendolo;

e le grandezze

- ϑ — posizione angolare del pendolo rispetto alla verticale (positiva se risulta da rotazione in verso antiorario);
- g — accelerazione di gravità (positiva verso il basso);
- τ_g — momento della forza di gravità rispetto al punto di ancoraggio del pendolo (positivo se causa rotazione in verso antiorario).

Dalla Meccanica, si ha:

$$\begin{aligned} \mu\ddot{\vartheta} &= \tau_g\,, \\ \mu &= ml^2\,, \\ \tau_g &= -mgl\sin(\vartheta)\,. \end{aligned}$$

Assumendo come variabile di uscita ϑ e come variabile di ingresso g, si può facilmente derivare il modello implicito ingresso-uscita

$$\ddot{\vartheta} + \frac{g}{l}\sin(\vartheta) = 0\,.$$

Per poter ricorrere allo schema realizzativo in Figura 5.11 bisogna però riferirsi ad un modello implicito ingresso-stato-uscita, ovvero nella forma generale

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, u) \\ y = g(\boldsymbol{x}, u)\,. \end{cases} \tag{5.1}$$

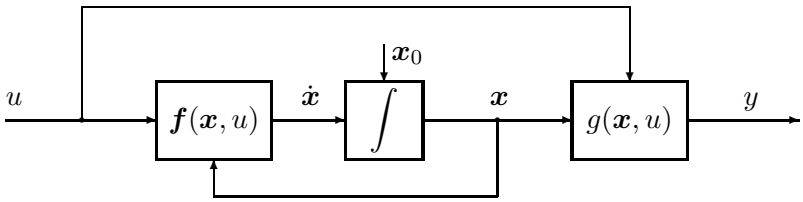


Figura 5.11: Schema realizzativo relativo al modello implicito ingresso-stato-uscita nella forma (5.1).

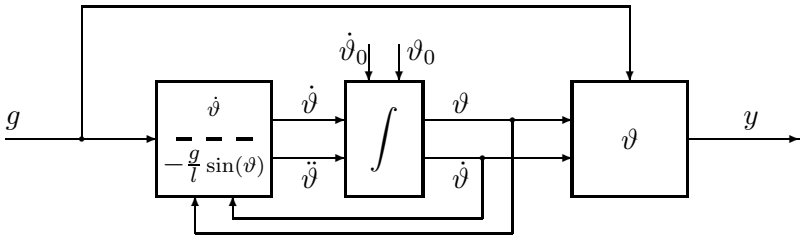


Figura 5.12: Schema realizzativo relativo al modello implicito ingresso-stato-uscita nella forma (5.2).

Nel nostro caso, avendo scelto $y=\vartheta$, $u=g$ ed assumendo

$$\boldsymbol{x} = \begin{pmatrix} \vartheta \\ \dot{\vartheta} \end{pmatrix},$$

si può ricavare

$$\begin{cases} \begin{pmatrix} \dot{\vartheta} \\ \ddot{\vartheta} \end{pmatrix} = \begin{pmatrix} \dot{\vartheta} \\ -\dfrac{g}{l}\sin(\vartheta) \end{pmatrix} \\ y = \vartheta \end{cases} \tag{5.2}$$

cui corrisponde lo schema in Figura 5.12. Si noti che la variabile di stato x_1 è legata all'energia potenziale del pendolo, mentre la variabile di stato x_2 è legata all'energia cinetica del pendolo. La prima equazione, inoltre, non è da leggersi come una banale identità, quanto come l'uguaglianza

$$\dot{x}_1 = x_2$$

che evidenzia il legame integrale tra le due variabili di stato.

Svolgendo lo schema in Figura 5.12 in modo da renderlo unifilare si ottiene lo schema in Figura 5.13.

Da questo un'ulteriore semplificazione grafica conduce allo schema in Figura 5.14.

Nel seguito si farà riferimento ad un pendolo caratterizzato dai parametri

$$m = 1\,\mathrm{kg} \qquad l = 1\,\mathrm{m}$$

in presenza di una accelerazione di gravità costante pari a $g = \hat{g} = 9.81\,\mathrm{m/s^2}$.

5.11.2 Costruzione dello schema Xcos

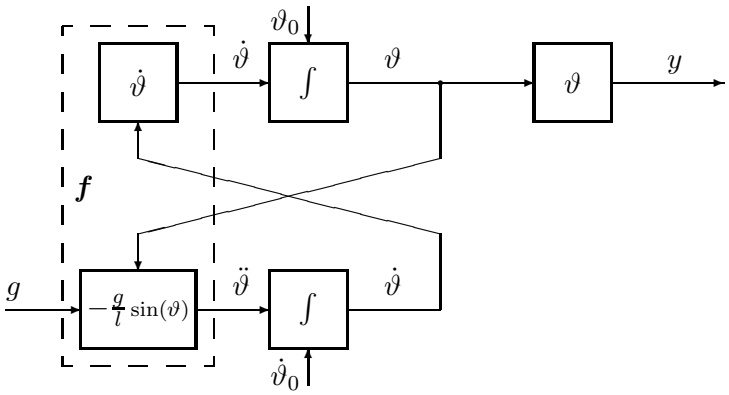


Figura 5.13: Trasformazione dello schema realizzativo in Figura 5.12.

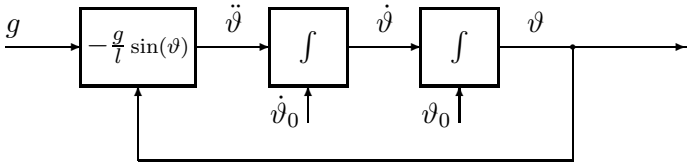


Figura 5.14: Semplificazione dello schema realizzativo in Figura 5.13.

Per la simulazione del pendolo si proceda alla costruzione di uno schema Xcos basato sullo schema realizzativo in Figura 5.14.

A tale scopo si invochi l'interfaccia grafica come mostrato nella sezione 5.1.2 salvando il file con il nome, ad esempio, di `pendolo.xcos`.

Nel nostro caso occorrono i seguenti blocchi:

- un blocco generatore di segnale costante, per fornire la grandezza di ingresso (blocco `CONST_m` nella cartella `sources`);
- un blocco che effettua il calcolo di una funzione algebrica (blocco `EXPRESSION` nella cartella `user-defined functions`);
- due blocchi che effettuano un'operazione di integrazione (blocco `INTEGRAL_m` nella cartella `continuous time systems`);
- un blocco oscilloscopio, per visualizzare l'andamento della variabile di uscita (blocco `CSCOPE` nella cartella `sinks`). Si noti che il blocco `CSCOPE` oltre all'ingresso relativo alla variabile da visualizzare ha bisogno di un ingresso relativo alla base dei tempi; questa può essere fornita da un blocco `CLOCK_c` nella libreria `Sources`.

Per il calcolo della funzione algebrica $-(g/l)\sin(\vartheta)$ occorre produrre in ingresso al blocco `EXPRESSION` due grandezze che forniscano i valori correnti di g e ϑ. Connettendo, ad esempio, la grandezza g alla prima porta e la grandezza ϑ alla seconda porta in ingresso, l'espressione da immettere nella finestra di dialogo del blocco `EXPRESSION`, riportata in Figura 5.15, vale

$$-\frac{g}{l}\sin(\vartheta) \qquad \rightarrow \qquad \texttt{-u1/l*sin(u2)}\,,$$

in cui, in luogo della variabile l da definirsi nel Context, può utilizzarsi direttamente il suo valore numerico.

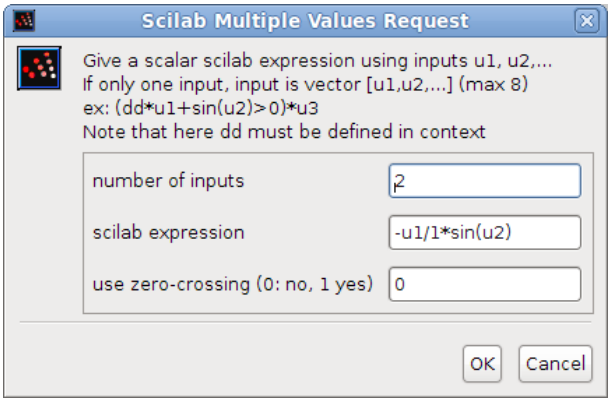


Figura 5.15: La finestra di dialogo del blocco `EXPRESSION`.

I due blocchi `INTEGRAL_m` necessitano delle rispettive condizioni iniziali che possono essere fornite nell'apposita riga della finestra di dialogo (vedi Figura 5.16). Si noti che i valori angolari vanno espressi in radianti, e che può quindi essere comodo introdurre una conversione gradi-radianti nella riga della finestra di dialogo in cui si assegnano le condizioni iniziali. Se, ad esempio, si vuole assegnare un angolo iniziale di 45° la riga `Initial Condition` della finestra di dialogo del secondo integratore può scriversi `45/180*%pi`.

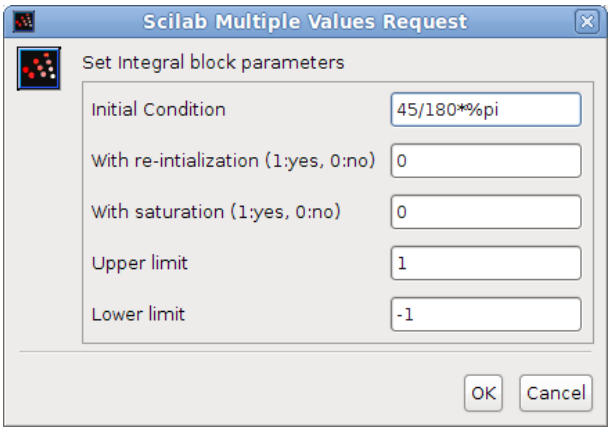


Figura 5.16: La finestra di dialogo del blocco `INTEGRAL_m`.

Effettuando le connessioni tra i blocchi in maniera congruente con lo schema realizzativo in Figura 5.14 si ottiene lo schema Xcos riportato in Figura 5.1.

È opportuno a questo punto effettuare un'operazione di salvataggio dello schema costruito mediante il comando `Save` del menu `File` della finestra `Pendolo` in cui si è operato.

5.11.3 Esecuzione della simulazione

Completata la costruzione dello schema è possibile procedere all'esecuzione della simulazione. A tale scopo è necessario dapprima configurare i parametri del simulatore mediante la finestra di dialogo `setup` del menu `Simulation`.

Dei molti parametri su cui è possibile intervenire è opportuno in prima battuta adeguare soltan-

to la durata della simulazione (`Final integration time`) alla scala temporale di evoluzione del sistema che si sta simulando. Nel nostro caso i valori dei parametri fisici del pendolo comportano delle oscillazioni che, dipendentemente dalle condizioni iniziali, si compiono nell'ordine di qualche secondo; possiamo quindi orientarci su una durata di 5 s.

Conviene inoltre configurare concordemente i parametri del blocco `CLOCK_c`. Infatti, il valore `0.1` di default del `Period` va adeguato alla durata della simulazione in modo da garantire sufficiente risoluzione al grafico visualizzato; con una durata di 5 s un valore `0.01` sembra più adeguato. Inoltre, il valore `0.1` di default del `Init time` esclude la visualizzazione del valore iniziale; conviene quindi portarlo a `0` (vedi Figura 5.17).

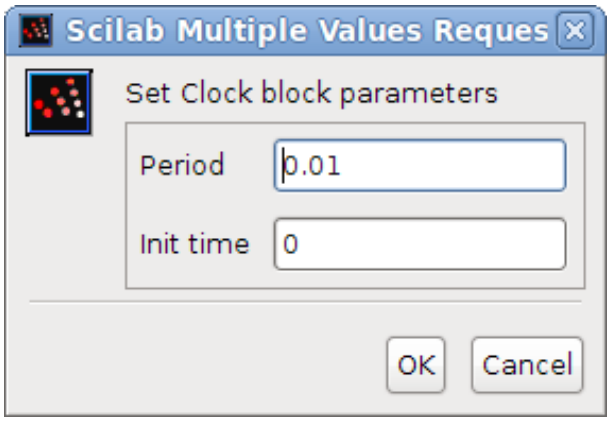


Figura 5.17: La finestra di dialogo del blocco `CLOCK_c`.

A questo punto la simulazione può essere avviata selezionando il comando `Run` del menu `Simulation` ed il risultato della simulazione si rende visibile in una finestra grafica che si attiva per effetto del blocco `CSCOPE`. Ad esempio, con le condizioni iniziali ($\dot{\vartheta} = 0°/\mathrm{s}$ e $\vartheta = 45° = \pi/4\,\mathrm{rad}$) la finestra grafica si presenta come riportato in Figura 5.18.

Si può riconoscere che la scala dell'asse orizzontale del grafico corrisponde all'intervallo temporale del `Refresh period` definito nelle proprietà del blocco `CSCOPE`, mentre quello verticale si estende dal valore `Ymin` al valore `Ymax` ivi definiti (vedi Figura 5.19).

Per meglio adeguare la scala di rappresentazione delle ascisse all'effettiva escursione delle ampiezze rappresentate si può, in alternativa:

- impostare nuovi valori per i parametri `Refresh period`, `Ymin` e `Ymax` nelle proprietà del blocco `CSCOPE` agendo nella finestra di dialogo rappresentata in Figura 5.19. Tale approccio richiede l'avvio di una nuova simulazione ma conserva nello schema i valori per tutte le future esecuzioni e, dopo salvataggio, anche per le future sessioni di simulazione;
- impostare l'intervallo di rappresentazione delle ascisse e delle ordinate nella finestra di dialogo `Axes Editor` che compare selezionando la voce `Axes properties` nel menu `Edit` della finestra grafica (si veda sezione 3.4);

Modificando la scala di rappresentazione in modo da visualizzare tutta la durata della simulazione (5 s) sull'asse delle ascisse ed una escursione da -1 a +1 sull'asse delle ordinate, la finestra grafica relativa al blocco `CSCOPE` si presenta quindi come riportato in Figura 5.20.

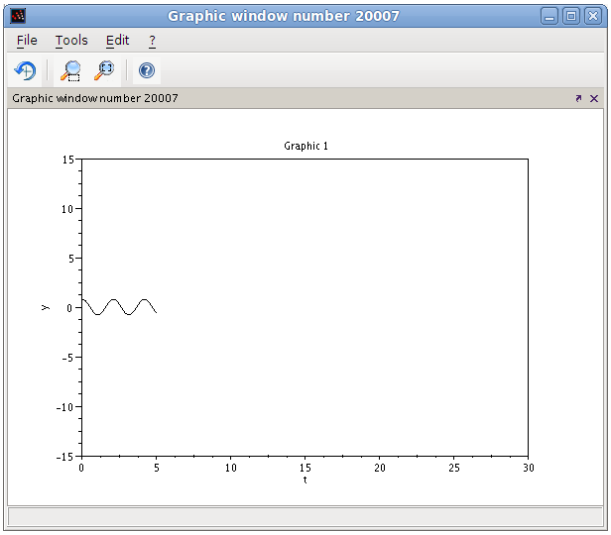


Figura 5.18: Finestra grafica relativa al blocco `CSCOPE` di Figura 5.1 al termine della simulazione con `Final integration time` = `5`, `Initial condition` = `0` per il primo integratore e `Initial condition`=`45/180*%pi` per il secondo integratore.

5.11.4 Linearizzazione

A partire dal modello implicito ingresso-stato-uscita (5.2), ricercando le soluzioni del problema

$$\boldsymbol{f}(\hat{\boldsymbol{x}}, \hat{u}) = \mathbf{0}\,,$$

è facile ricavare che il pendolo possiede equilibri stazionari caratterizzati da

$$\hat{x}_1 = \hat{\vartheta} = k\pi \qquad \hat{x}_2 = \hat{\dot{\vartheta}} = 0 \qquad \hat{u} = \hat{g}$$

con $k \in \mathbb{N}$. Introducendo le variabili di scostamento

$$\delta\vartheta = \vartheta - \hat{\vartheta} \qquad \delta\dot{\vartheta} = \dot{\vartheta} - \hat{\dot{\vartheta}} = \dot{\vartheta}$$

la forma implicita ingresso-stato-uscita del modello linearizzato si scrive

$$\left\{ \begin{array}{rcl} \begin{pmatrix} \dot{\vartheta} \\ \ddot{\vartheta} \end{pmatrix} & = & \begin{pmatrix} 0 & 1 \\ -\dfrac{\hat{g}}{l}\cos(\hat{\vartheta}) & 0 \end{pmatrix} \begin{pmatrix} \delta\vartheta \\ \dot{\vartheta} \end{pmatrix} = \begin{pmatrix} \dot{\vartheta} \\ -\dfrac{\hat{g}}{l}\cos(\hat{\vartheta})\delta\vartheta \end{pmatrix} \\ y & = & \delta\vartheta\,. \end{array} \right. \tag{5.3}$$

Con riferimento all'equilibrio caratterizzato da

$$\hat{\vartheta} = 0 \qquad \hat{\dot{\vartheta}} = 0 \qquad \hat{g} = 9.81\,\mathrm{m/s^2}\,,$$

il modello linearizzato si riduce a

$$\left\{ \begin{array}{rcl} \begin{pmatrix} \dot{\vartheta} \\ \ddot{\vartheta} \end{pmatrix} & = & \begin{pmatrix} \dot{\vartheta} \\ -\dfrac{\hat{g}}{l}\vartheta \end{pmatrix} \\ y & = & \vartheta \end{array} \right. \tag{5.4}$$

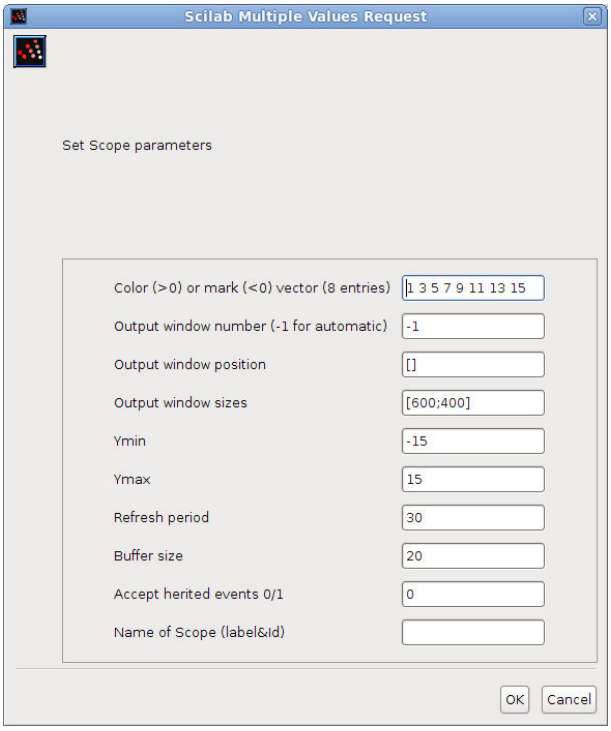


Figura 5.19: La finestra di dialogo del blocco `CSCOPE`.

che, in maniera analoga a quanto visto nella Sezione 5.11.1, può essere ricondotto allo schema in Figura 5.21.

Per la costruzione di uno schema Xcos basato sullo schema realizzativo in Figura 5.21, vista la somiglianza strutturale con lo schema in Figura 5.14, si può procedere per semplice duplicazione ed adattamento dello schema in Figura 5.1. A tale scopo si selezioni dapprima l'intero schema presente nella finestra `Pendolo`, ad esempio mediante l'uso del cursore o più direttamente mediante la combinazione di tasti `Ctrl+A`, e si proceda alla classica sequenza `Ctrl+C`, `Ctrl+V` utile per la duplicazione negli applicativi in ambiente `Linux/Windows`. Per evitare la comparsa dello schema duplicato in sovrapposizione con lo schema originario conviene, dopo il `Ctrl+C`, toccare l'area della finestra indicando con un clik un opportuno punto di inserzione del secondo schema; successivamente al `Ctrl+V` lo schema appena inserito resta selezionato consentendone un'agevole riposizionamento con il mouse se necessario. A questo punto, lo schema ottenuto per duplicazione si adatta alla simulazione dello schema realizzativo in Figura 5.21 semplicemente modificando la funzione implementata dal blocco `EXPRESSION` da

$$-\frac{g}{l}\sin(\vartheta) \quad \rightarrow \quad \texttt{-u(1)/l*sin(u(2))}$$

a

$$-\frac{\hat{g}}{l}\vartheta \quad \rightarrow \quad \texttt{-u(1)/l*u(2)}\,.$$

Si noti che l'operazione di copia fa ereditare agli oggetti duplicati le proprietà dei rispettivi oggetti originali; in particolare, i blocchi integratori duplicati presentano rispettivamente le stesse condizioni iniziali dei blocchi origine consentendo così agevolmente il confronto dei due diversi simulatori a partire dalle stesse condizioni iniziali.

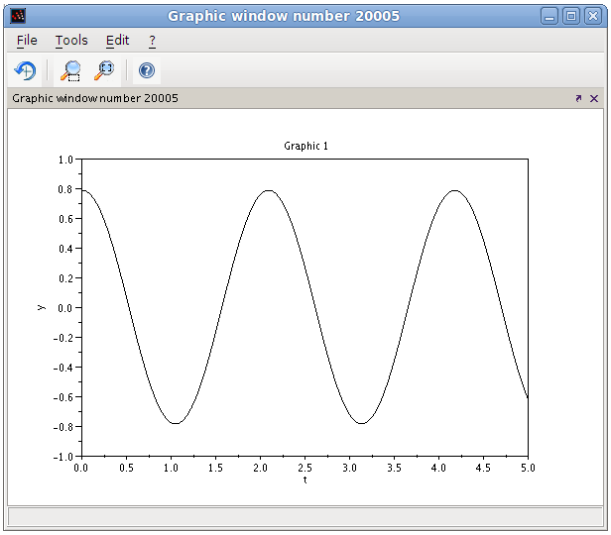


Figura 5.20: Modifica del grafico rappresentato in Figura 5.19 dopo la scalatura degli assi.

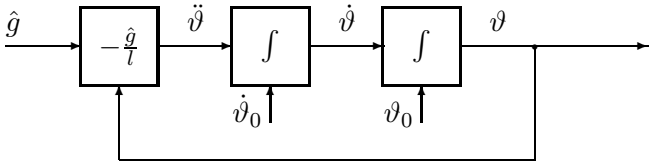


Figura 5.21: Schema realizzativo semplificato relativo al modello linearizzato implicito ingresso-stato-uscita nella forma (5.4).

Al fine di ottenere un confronto diretto dei due risultati di simulazione conviene inviare le due variabili di uscita ad un unico blocco oscilloscopio. Ciò può essere fatto in due modi:

- utilizzando un blocco `CMSCOPE` invece del blocco `CSCOPE`; i due segnali in ingresso vengono così rappresentati in due grafici sovrapposti che possono o meno condividere le scale di rappresentazione;
- convogliando tramite un blocco `MUX` disponibile nella cartella `signal routing` le 2 uscite su un unico filo multivariabile da porsi in ingresso ad un singolo blocco `CSCOPE`. Si ottiene un unico grafico con i due diagrammi sovrapposti.

Adotteremo quest'ultima soluzione poichè nel nostro caso meglio si presta al confronto delle due evoluzioni. Lo schema risultante è rappresentato in Figura 5.22 ed il risultato della simulazione in un tempo di 10 s è in Figura 5.23. È facile riconoscere che, con oscillazioni ampie $\pm 45°$, il risultato della simulazione basata sul modello linearizzato si sgancia molto rapidamente da quello fornito dal modello completo.

Ripetendo invece la simulazione con un angolo iniziale di 5° il risultato ottenuto con il modello linearizzato approssima bene quello fornito dal modello completo (vedi Figura 5.24). In particolare, utilizzando lo strumento `Zoom`, è possibile riscontrare uno sganciamento di circa 4.5 ms tra le due soluzioni dopo quasi 10 s di simulazione.

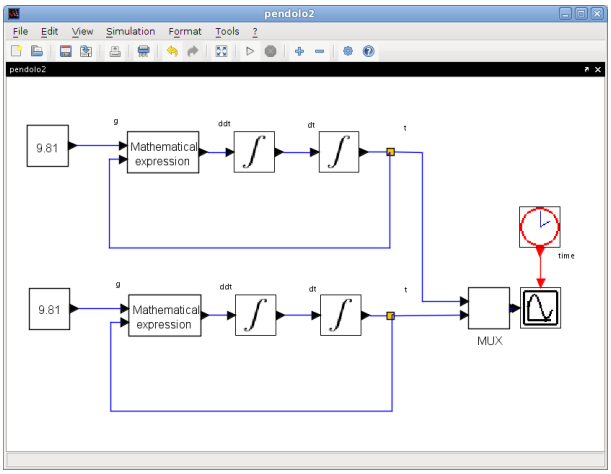


Figura 5.22: Schema Xcos corrispondente all'insieme dello schema realizzativo in Figura 5.14 ed in Figura 5.21.

5.11.5 Rassegna sui blocchi Xcos

Una rassegna non esaustiva sui blocchi Xcosdivisa secondo le cartelle della finestra browser:

- Annotations palette

```
Annotations_pal      Annotations palette
TEXT_f               Text
```

- Commonly used blocks palette

```
Commonlyusedblocks_pal     Commonly used blocks palette
LOGICAL_OP                 Logical operation
RELATIONALOP               Relational operation
```

- Continuous time systems palette

```
Continuous_pal      Continuous time systems palette
CLINDUMMY_f         Dummy
CLR                 Continuous transfer function
CLSS                Continuous state-space system
DERIV               Derivative
INTEGRAL_f          Integration
INTEGRAL_m          Integration
PID                 PID regulator
TCLSS               Continuous linear system with jump
TIME_DELAY          Time delay
VARIABLE_DELAY      Variable delay
```

- Demonstrations blocks palette

```
Demonstrationsblocks_pal     Demonstrations blocks palette
AUTOMAT                      automata (finite state machine)
BOUNCE                       Balls coordinates generator
```

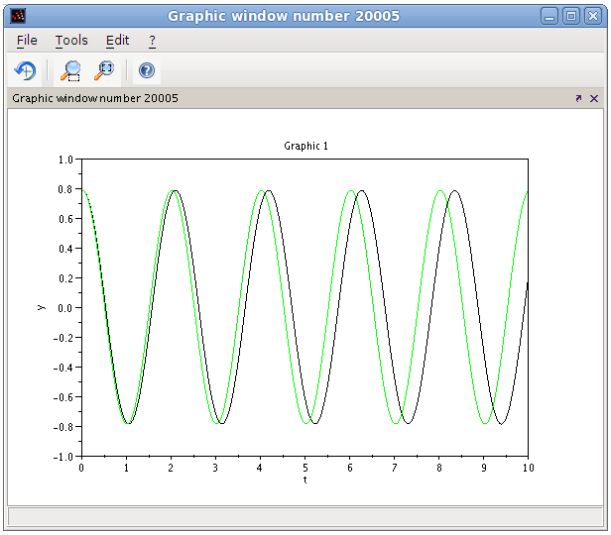


Figura 5.23: Finestra del blocco CSCOPE di Figura 5.22 al termine della simulazione con $\vartheta(0) = 45°$.

```
BOUNCEXY                    Balls viewer
BPLATFORM                   Balls under a platform viewer
PDE                         1D PDE block
```

- Discontinuities palette

```
discontinuities_pal     discontinuities palette
BACKLASH                Backlash
DEADBAND                Deadband
HYSTHERESIS             Hystheresis
RATELIMITER             Rate limiter
SATURATION              Saturation
```

- Discrete time systems palette

```
Discrete_pal      Discrete time systems palette
DELAYV_f          Variable delay
DELAY_f           Discrete time delay
DLR               Discrete transfer function
DLRADAPT_f        Discrete Zero-Pole
DLSS              Discrete state-space system
DOLLAR_f          Delay operator
REGISTER          Shift Register
```

- Electrical palette

```
Electrical_pal        Electrical palette
CCS                   Controllable Modelica current source
CVS                   Controllable Modelica voltage source
Capacitor             Electrical capacitor
ConstantVoltage       Electrical DC voltage source
CurrentSensor         Electrical current sensor
```

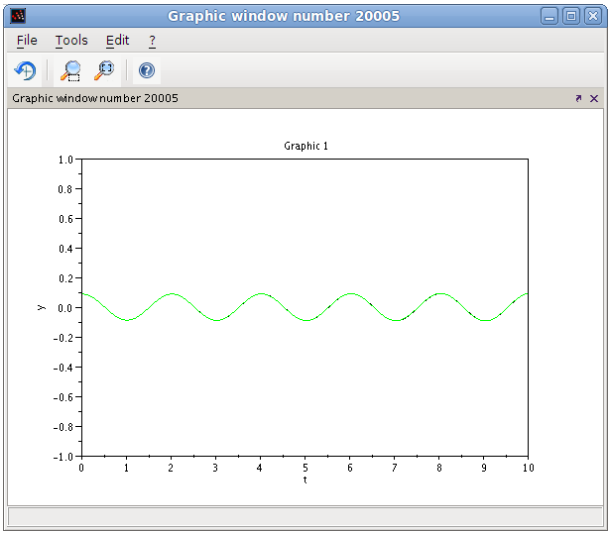


Figura 5.24: Finestra del blocco `CSCOPE` di Figura 5.22 al termine della simulazione con $\vartheta(0) = 5°$.

```
Diode                 Electrical diode
Ground                Ground (zero potential reference)
Gyrator               Modelica Gyrator
IdealTransformer      Ideal Transformer
Inductor              Electrical inductor
NMOS                  Simple NMOS Transistor
NPN                   NPN transistor
OpAmp                 Ideal opamp (norator-nullator pair)
PMOS                  Simple PMOS Transistor
PNP                   PNP transistor
PotentialSensor       Potential sensor
Resistor              Electrical resistor
SineVoltage           Sine voltage source
Switch                Non-ideal electrical switch
VVsourceAC            Variable AC voltage source
VariableResistor      Electrical variable resistor
VoltageSensor         Electrical voltage sensor
VsourceAC             Electrical AC voltage source
```

- Event handling palette

```
Events_pal                Event handling palette
ANDBLK                    Activation and
ANDLOG_f                  Logical and
CEVENTSCOPE               Activation scope
CLKFROM                   Receives data from a corresponding CLKGOTO
CLKGOTO                   Pass block input to CLKFROM block
CLKGotoTagVisibility      Define Scope of CLKGOTO tag visibility
CLKOUTV_f                 Output activation port
CLKSOMV_f                 Activation union
EDGE_TRIGGER              EDGE_TRIGGER block
ESELECT_f                 Synchronous block Event-Select
EVTDLY_c                  Event delay
```

```
EVTGEN_f                Event generator
EVTVARDLY               Event variable delay
Extract_Activation      Extract_Activation block
HALT_f                  Halt
IFTHEL_f                Synchronous block If-Then-Else
MCLOCK_f                MCLOCK_f title
MFCLCK_f                MFCLCK_f title
M_freq                  Multiple Frequencies
freq_div                Frequency division
```

- Implicit palette

```
Implicit_pal     Implicit palette
CONSTRAINT_f     Constraint
DIFF_f           Derivative
```

- Integer palette

```
Integer_pal     Integer palette
BITCLEAR        Clear a Bit
BITSET          Set a Bit
CONVERT         Data Type Conversion
DFLIPFLOP       D flip-flop
DLATCH          D latch flip-flop
EXTRACTBITS     EXTRACTBITS
INTMUL          Integer matrix multiplication
JKFLIPFLOP      JK flip-flop
LOGIC           Combinational Logic
SHIFT           Shift Bits
SRFLIPFLOP      SR flip-flop
```

- Lookup tables palette

```
Lookuptables_pal     Lookup tables palette
INTRP2BLK_f          2D interpolation
INTRPLBLK_f          Interpolation
LOOKUP_f             Lookup table
```

- Math operations palette

```
Mathoperations_pal     Math operations palette
ABS_VALUE              Absolute value
BIGSOM_f               Sum
COSBLK_f               COSBLK
EXPBLK_m               Exponential
GAINBLK_f              Gain
INVBLK                 Inverse
LOGBLK_f               Log
MATMAGPHI              Complex to Magnitude and Angle Conversion
MATZREIM               Complex decomposition
MAXMIN                 Max and Min
MAX_f                  MAX
```

```
MIN_f                   MIN
POWBLK_f                Power
PRODUCT                 Product
PROD_f                  Multiplication
SIGNUM                  Signum
SINBLK_f                SINBLK
SQRT                    Square root
SUMMATION               Matrix Summation
SUM_f                   Addition
TANBLK_f                TANBLK
TrigFun                 Trigonometric function
```

- Matrix operation palette

```
Matrix_pal      Matrix operation palette
CUMSUM          Cumulative Sum
EXTRACT         Matrix Extractor
EXTTRI          Triangular or Diagonal extraction
MATBKSL         left matrix division
MATCATH         Horizontal Concatenation
MATCATV         Vertical Concatenation
MATDET          Matrix Determinant
MATDIAG         Create Diagonal Matrix
MATDIV          Matrix division
MATEIG          Matrix Eigenvalues
MATEXPM         Matrix Exponential
MATINV          Matrix Inverse
MATLU           LU Factorization
MATMUL          Matrix Multiplication
MATPINV         Matrix PseudoInverse
MATRESH         Matrix Reshape
MATSING         SVD decomposition
MATSUM          Matrix Sum
MATTRAN         Matrix Transpose
MATZCONJ        Matrix Conjugate
RICC            Riccati Equation
ROOTCOEF        Coefficient computation
SUBMAT          Sub-matrix extraction
```

- Port & Subsystem palette

```
Portaction_pal      Port & Subsystem palette
IN_f                Input Port
OUTIMPL_f           Output implicit port
OUT_f               Output Port
```

- Signal processing palette

```
Signalprocessing_pal      Signal processing palette
QUANT_f                   Quantization
SAMPHOLD_m                Sample and hold
```

- Signal routing palette

```
Signalrouting_pal       Signal routing palette
DEMUX                   Demultiplexer
EXTRACTOR               Extractor
FROM                    FROM Receives data from a corresponding GOTO
FROMMO                  Receives data from a corresponding GOTOMO
GOTO                    GOTO Pass block input to From block
GOTOMO                  Pass block input to FROMMO block
GotoTagVisibility       Define Scope of GOTO tag visibility
GotoTagVisibilityMO     Define Scope of GOTOMO tag visibility
ISELECT_m               Iselect
MUX                     Multiplexer
M_SWITCH                Multi-port switch
NRMSOM_f                Merge data
RELAY_f                 Relay
SELECT_m                Select
SWITCH2_m               Switch2
SWITCH_f                Switch
```

- Sinks palette

```
Sinks_pal     Sinks palette
AFFICH_m      Display
CANIMXY       y=f(x) animated viewer
CANIMXY3D     z=f(x,y) animated viewer
CFSCOPE       Floating point scope
CMAT3D        Matrix z values 3D viewer
CMATVIEW      Matrix Colormapped viewer
CMSCOPE       Multi display scope
CSCOPE        Single Display Scope
CSCOPXY       y=f(x) permanent viewer
CSCOPXY3D     z=f(x,y) permanent viewer
ENDBLK        END block
END_c         END_c block
TOWS_c        Data to Scilab worspace
TRASH_f       Trash block
WFILE_f       Write to file
WRITEAU_f     Write AU sound file
WRITEC_f      Write binary data
```

- Sources palette

```
Sources_pal       Sources palette
CLKINV_f          Input activation port
CLOCK_c           Activation clock
CONST_m           Constant
CURV_f            Curve
Counter           Counter
FROMWSB           Data from Scilab workspace to Xcos
GENSIN_f          Sin generator
GENSQR_f          Square wave generator
INIMPL_f          Input implicit port
```

```
Modulo_Count      Modulo counter
RAMP              Ramp
RAND_m            Random generator
READAU_f          Read AU sound file
READC_f           Read binary data
RFILE_f           Read from file
SAWTOOTH_f        Sawtooth generator
STEP_FUNCTION     Step function generator
SampleCLK         Sample Time Clock
Sigbuilder        Signal creator/generator
TIME_f            Time
TKSCALE           Adjust constant value with a tk widget
```

- Thermohydraulics palette

```
ThermoHydraulics_pal     Thermal-Hydraulics toolbox
Bache                    Thermal-hydraulic tank (reservoir)
PerteDP                  Thermal-hydraulic pipe
PuitsP                   Thermal-hydraulic drain (well)
SourceP                  Thermal-hydraulic constant pressure source
VanneReglante            Thermal-hydraulic control valve
```

- User defined functions palette

```
Userdefinedfunctions_pal     User defined functions palette
CBLOCK                       New C
EXPRESSION                   Mathematical expression
MBLOCK                       Modelica generic block
SUPER_f                      Super block
c_block                      C language
fortran_block                Fortran
generic_block3               Generic block
scifunc_block_m              Scilab function block
```

- Zero crossing detection palette

```
Zerocrossingdetection_pal     Zero crossing detection palette
GENERAL_f                     GENERAL_f title
NEGTOPOS_f                    Threshold negative to positive
POSTONEG_f                    Threshold positive to negative
ZCROSS_f                      Threshold detection at zero
```

Capitolo 6

Scilab vs Matlab

Matlab è probabilmente il più diffuso programma di calcolo scientifico nelle Università; negli ultimi anni la sua diffusione in ambito industriale è stata anche significativa.

Le prime versioni di Matlab risalgono alla fine degli anni 80 il che fa di questo sw un prodotto certamente testato e sostanzialmente privo di bachi. I vantaggi nell'uso di Matlab sono notevoli:

- È diventato quasi uno standard nell'ambito scientifico universitario;
- Racchiude in un solo ambiente una vasta libreria di funzioni;
- La comunità di utilizzatori Matlab è enorme, dal sito della Mathworks, ad esempio, è possibile accedere ad innumerevoli funzioni scritte da utenti e messe a disposizione degli altri utenti.

Ci sono alcuni svantaggi nell'utilizzo di Matlab:

- È un codice commerciale il cui costo non è banale;
- È dotato di una vasta quantità di *toolbox* aggiuntivi che sono tutti a pagamento;
- Essendo un codice coperto da copyright è installabile su una sola macchina;
- Gli studenti universitari non possono esercitarsi sul proprio computer se non acquistando una licenza o craccando il sw;
- Adotta una strategia di commercializzazione aggressiva, proponendo con frequenza nuove versioni con modifiche spesso solo cosmetiche e non garantendo sempre la compatibilità con le versioni precedenti.

Anche per Scilab, ovviamente, ci sono vantaggi e svantaggi:

- È un codice gratuito, qualsiasi utente può installarlo dove e come vuole senza dover pagare nè compiere azioni illegali;
- Libera l'utente dalla necessità di continui aggiornamenti la cui necessità è spesso discutibile;
- È essenzialmente un *clone* di Matlab, il passaggio da Matlab a Scilab, per quanto non indolore, è abbastanza facile;

Tabella 6.1: Funzioni Matlab emulate in Scilab

mtlb_0	mtlb_a	mtlb_all	mtlb_any
mtlb_axis	mtlb_beta	mtlb_box	mtlb_close
mtlb_colordef	mtlb_conv	mtlb_cumprod	mtlb_cumsum
mtlb_dec2hex	mtlb_delete	mtlb_diag	mtlb_diff
mtlb_dir	mtlb_double	mtlb_e	mtlb_echo
mtlb_eig	mtlb_eval	mtlb_exist	mtlb_eye
mtlb_false	mtlb_fft	mtlb_fftshift	mtlb_find
mtlb_findstr	mtlb_fliplr	mtlb_fopen	mtlb_format
mtlb_fprintf	mtlb_fread	mtlb_fscanfb	mtlb_full
mtlb_fwrite	mtlb_grid	mtlb_hold	mtlb_i
mtlb_ifft	mtlb_imp	mtlb_int16	mtlb_int32
mtlb_int8	mtlb_is	mtlb_isa	mtlb_isfield
mtlb_isletter	mtlb_isspace	mtlb_l	mtlb_legendre
mtlb_linspace	mtlb_load	mtlb_logic	mtlb_logical
mtlb_lower	mtlb_max	mtlb_min	mtlb_more
mtlb_num2str	mtlb_ones	mtlb_plot	mtlb_prod
mtlb_rand	mtlb_randn	mtlb_rcond	mtlb_realmax
mtlb_realmin	mtlb_repmat	mtlb_s	mtlb_save
mtlb_setstr	mtlb_size	mtlb_sort	mtlb_strcmp
mtlb_strcmpi	mtlb_strfind	mtlb_strrep	mtlb_sum
mtlb_t	mtlb_toeplitz	mtlb_tril	mtlb_triu
mtlb_true	mtlb_uint16	mtlb_uint32	mtlb_uint8
mtlb_upper	mtlb_zeros		

- Oltre ad essere gratuito è *open source*, con tutti i pro e i contro di questa scelta;
- La comunità di utenti Scilab ha messo in condivisione diversi pacchetti sw e manuali;
- Non è privo di bachi;
- Ha una comunità ridotta rispetto a Matlab e problemi non banali non sempre trovano soluzione immediata;
- In diversi punti appare *acerbo*, ad esempio alcuni comandi *simili* hanno sintassi completamente diverse fra loro, in generale, alcune sintassi sono troppo di *basso* livello e potrebbero essere rese più *user friendly* con poco sforzo.

Scilab mette a disposizione una serie di funzioni che *emulano* il comportamento di funzioni Matlab, queste funzioni hanno il prefisso `mtlb_` e sono riportare in tabella 6.1. È anche disponibile un tool per tentare la traduzione automatica di codice scritto in sintassi Matlab.

Gli operatori sono uguali in Matlab e Scilab ma la semantica cambia leggermente, ad esempio

```
A>B
```

con A e B matrici vuote ritorna in uscita una matrice vuota in Matlab e un errore in Scilab. Se A e B sono matrici complesse, Matlab lavora sulla parte reale mentre Scilab dà errore. Le maggiori differenze si hanno per le variabili di tipo stringa.

Il tipo stringa, in particolare, è gestito in maniera diferenza in Matlab e Scilab. Matlab considera una stringa di caratteri la dove Scilab considera una matrice di caratteri. Ad esempio `'mystring'` in Matlab diventa `[m,y,s,t,r,i,n,g]` in Scilab. In Scilab, quindi, una stringa di caratteri è un oggetto di tipo stringa di dimensioni 1×1 mentre in Matlab è un vettore con un numero di elementi pari al numero dei caratteri. Un piccolo vantaggio di Scilab è che diventa possibile impilare stringhe di dimensioni diverse cosa che, in Matlab, è possibile solo definendo un `cell of character strings`.

Si deve precisare che, anche a parità di nome del comando, la sintassi può cambiare; non solo, a parità di sintassi può essere diversa la semantica.

La pagina http://www.scilab.org/product/dic-mat-sci/M2SCI_doc.htm contiene un utile dizionario, è sufficiente cliccare sulla funzione Matlab desiderata per accede alla pagina con l'equivalente Scilab o, in caso di assenza di un comando corrispondente, con le righe di codice da implementare per ottenere un comportamento equivalente.

Capitolo 7

GNU Free Documentation License

Version 1.3, 3 November 2008.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within

that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History

section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate"

if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the

option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.