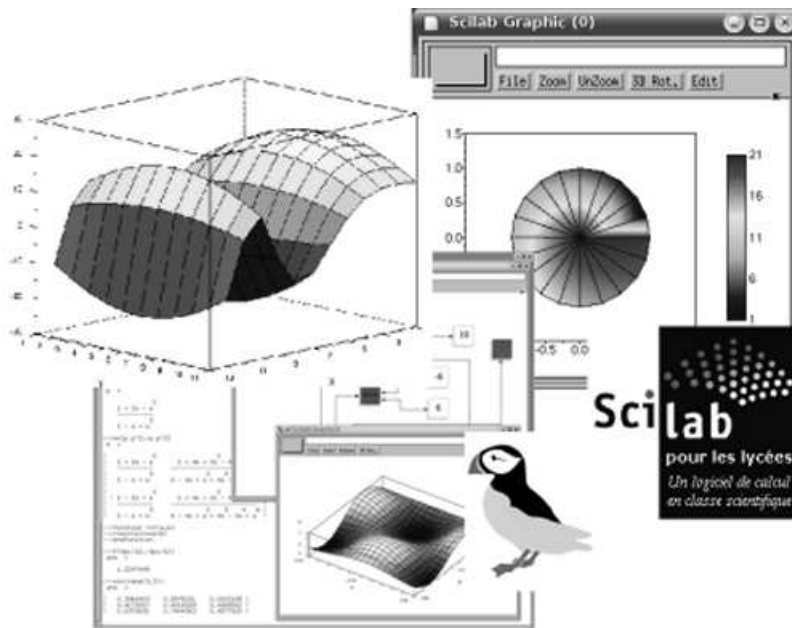


Dispense sull'uso di SCILAB
corso di Progetto dei Sistemi di Controllo

A.A. 2006/2007

Marcello Orlandesi

Gianni Bianchini



Indice

1	Introduzione	3
2	Operazioni base	3
2.1	Variabili Permanenti	5
3	Vettori e Matrici	5
3.1	Funzioni preimpostate	6
3.2	Funzioni matematiche elementari	8
3.3	Funzioni particolari per matrici	9
4	Polinomi	12
5	Grafici	12
5.1	Grafici 2D	13
5.2	Grafici 3D	14
5.2.1	Curve in 3D	14
5.2.2	Superfici in 3D	14
6	Scripting e Funzioni	15
7	Cicli for, while e istruzioni condizionali	16
8	Control Toolbox	18
8.1	Definizione di un sistema lineare	18
8.2	Funzioni per la manipolazione e l'analisi nel tempo e in frequenza di sistemi lineari	19
8.3	Funzioni modificate per la risposta in frequenza	20
9	Esempi Svolti	21

1 Introduzione

Scilab è un programma open source, distribuito dal Consorzio Scilab [1], sviluppato per il calcolo numerico scientifico. È inoltre un potente ambiente di sviluppo per le applicazioni scientifiche e ingegneristiche. È stato sviluppato a partire dal 1990 da INRIA e ENPC e dal 1994 distribuito liberamente su Internet per varie piattaforme e sistemi operativi, completo di codice sorgente [2]. Scilab è stato concepito in modo da essere un sistema aperto, nell'ambito del quale l'utente può definire nuovi tipi di dati, operazioni e funzioni. Contiene centinaia di funzioni matematiche predefinite, e dà la possibilità di utilizzare **routines** scritte in vari linguaggi di programmazione come FORTRAN, C, C++, JAVA. Possiede **strutture** dati sofisticate (inclusive di tipi specifici per liste, polinomi, frazioni razionali, sistemi lineari...), un **interprete** ed un **linguaggio di programmazione** di alto livello [3] [4] [5]. Scilab è un sistema emergente, simile al più noto ambiente proprietario MatlabTM, per l'elaborazione e manipolazione di dati, calcolo numerico e letterale, plotting, grafica 2D e tridimensionale, creazione di funzioni e programmazione scientifica. Maggiori informazioni possono essere reperite sul sito di riferimento, in cui è possibile anche ottenere gratuitamente l'ultima versione stabile (e alcune versioni precedenti), i manuali per l'utilizzo del software ed un buon numero di funzioni sviluppate da terze parti per applicazioni particolari, disponibili come plug-in.

Un'importante caratteristica di questo software è quella di avere una licenza che consente di utilizzarlo, modificarlo e ridistribuirlo in modo pressoché completamente libero. Grazie a questa possibilità gli sviluppatori e la comunità del software libero/open source hanno reso Scilab un importante strumento di calcolo per la didattica e la ricerca.

2 Operazioni base

All'avvio del programma, l'utente può digitare i comandi tramite un'interfaccia a linea di comando denominata *command window*.

L'operazione fondamentale per iniziare a lavorare è quella di *assegnazione a variabile*. Innanzitutto è da notare che questo programma è *case sensitive*, quindi fa distinzione fra lettere maiuscole e minuscole, ad esempio, con l'istruzione

```
--> a = 19
```

```
a =
```

```
19
```

si assegna il valore scalare *19* alla variabile *a*, risulta quindi diversa l'assegnazione

```
--> A = 19
```

```
A =
```

```
19
```

che assegna lo scalare *19* alla variabile *A*.

L'uso del simbolo *;* alla fine del comando rende invisibile l'output dello stesso, quindi nella *command window* non verrà stampato niente.

Le variabili sono sovrascrivibili, cioè, se digitiamo ora la stringa

```
--> A = 16;
```

```
A =
```

```
16
```

il precedente valore *19* viene definitivamente perso.

È inoltre possibile digitare più comandi consecutivi nella stessa stringa, separati dal simbolo `,` oppure `;`

```
--> A = 16, A = 22, b = 15
```

```
A =
```

```
16
```

```
A =
```

```
22
```

```
b =
```

```
15
```

Un comando può essere scritto in più linee attraverso il simbolo `...` come mostrato di seguito:

```
--> 1 + . . .
```

```
--> 2
```

```
ans =
```

```
3
```

Si noti come il programma si avvalga di una variabile di appoggio *ans* in cui viene salvato il risultato dell'ultima operazione effettuata.

Attraverso l'istruzione *format* è possibile modificare il formato di visualizzazione dei risultati ma NON la precisione con cui i calcoli vengono condotti. Il comando ha la seguente sintassi:

```
format(type,long)
```

in cui il campo *type* può assumere i valori

- *'v'*, per avere la visualizzazione in formato *floating point*;
- *'e'*, per avere la visualizzazione in formato *esponenziale*

mentre nel campo *long* va inserito il numero massimo di cifre da considerare (questo parametro è settato a 10 per default). È da notare che nel caso *type='v'* devono essere indicate almeno tre cifre dopo la virgola. Omettendo inoltre uno dei due parametri, questo viene mantenuto al valore attualmente usato.

Il comando *format* senza alcun parametro rende un vettore di due componenti in cui il primo è 0 per il formato *e* ed 1 per il formato *'v'* mentre il secondo elemento indica il numero di cifre. È inoltre possibile che i numeri visualizzati, nel formato esponenziale, abbiano E (se il programma viene usato sotto Linux) al posto di D (sotto Microsoft Windows).

Per avere informazioni sui comandi e funzioni è presente l'istruzione *help*; digitando quindi:

--> help

nel workspace verrà aperta una finestra con il menù dell'help navigabile, mentre digitando

--> help *nome comando*

verrà aperta una nuova finestra contenente tutte le informazioni relative al comando richiesto.

2.1 Variabili Permanenti

In Scilab sono presenti tutte le più comuni costanti matematiche, trigonometriche, etc. già preimpostate e non modificabili, come ad esempio

- %pi: il numero 3.1415927....;
- %i: l'unità immaginaria $\sqrt{-1}$;
- %e il numero 2.7182818... numero di Nepero.
- %eps: precisione di macchina, ovvero il minimo valore tale che, per il calcolatore, risulti $(1+\%eps)>1$, tale valore è dipendente in genere dall'architettura
--> %eps
%eps =
2.220D-16
quindi, per il calcolatore, due numeri che differiscono per %eps, sono uguali, infatti
-->5+%eps/2-5
ans =
0
- %inf, infinito;
- %nan, not-a-number;
- %f, variabile logica *false*;
- %t, variabile logica *true*;

3 Vettori e Matrici

In Scilab sono presenti molte funzioni per calcoli su matrici e vettori. Per definire una matrice è necessario racchiudere i valori fra parentesi quadre, separando i valori correnti di riga con uno spazio e le righe con l'operatore *punto e virgola*:

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

```
A=
```

```
! 1. 5. 9. !
```

```
! 7. 2. 3. !
```

```
! 4. 8. 6. !
```

È possibile dare degli intervalli nome_variabile = [start:step:stop]:

```
a=[1:4]
```

```
a=
```

```
! 1. 2. 3. 4. !
```

```
b=[1:0.1:2]
```

```
! 1. 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2 !
```

 (1)

È possibile quindi eseguire le operazioni base di *somma*, *moltiplicazione*, *divisione* fra matrici e matrici e vettori nonché il calcolo delle seguenti funzioni:

- *determinante*, `det(matrice)`;
- *trasposta*, `matrice'`;
- *inversa*, `inv(matrice)`;
- *autovalori*, `spec(matrice)`;
- *rango*, `rank(matrice)`;
- *traccia*, `trace(matrice)`;
- *autovettori*, `[autovalori,autovettori]=bdiag(matrice)`;

3.1 Funzioni preimpostate

Per vettori e matrici esistono già preimpostate le seguenti funzioni:

- *size*, `size(vettore/matrice)`, restituisce la dimensione del vettore/matrice attraverso due parametri, in uno indica il numero di righe e nell'altro quello delle colonne
--> `v=[1 2 3]`;
--> `size(v)`
`ans =`
`! 1. 3. !`
--> `size(v,1)`
`ans=`
`1.`
--> `size(v,2)`
`ans=`
`3.`
- *length*, `length(vettore)`, restituisce la lunghezza del vettore passato in input (nel caso matriciale restituisce la dimensione massima fra quella di righe e colonne)
--> `v=[1 2 3 4]`;
--> `length(v)`

```
ans=
```

```
4.
```

- *max*, *min*, indicano rispettivamente il valore massimo e minimo del vettore/matrice passato in input

```
--> v= [1000 200 4];
```

```
--> max(v)
```

```
ans=
```

```
1000.
```

```
--> min(v)
```

```
ans=
```

```
4.
```

- *sum*, *prod*, calcolano rispettivamente la somma ed il prodotto fra gli elementi del vettore passato come input

```
--> v=[50 26 8];
```

```
--> sum(v)
```

```
ans=
```

```
84.
```

```
--> prod(v)
```

```
ans=
```

```
10400.
```

- *norm*, *norm(vettore, parametro)*, calcola la norma del vettore in input, nel campo *parametro* è possibile specificare il tipo di norma da calcolare tenendo presente la seguente formula:

$$\|v\|_n = \left(\sqrt[n]{\sum_{i=1}^{\text{lenght}(v)} |v_i|^n} \right)^{\frac{1}{n}} \quad (2)$$

Il parametro può essere settato anche con il valore *'inf'* nel qual caso si calolerà la norma infinito $\|v\|_\infty = \max_{1 < i < \text{lenght}(v)} |v_i|$. Per default Scilab calcola la norma 2 (n=2).

- *abs*, restituisce un vettore avente come componenti il valore assoluto di ciascuna componente del vettore passato in input:

```
--> v=[-5 -9 7 8 -6 ]
```

```
v=
```

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

```
--> abs(v)
```

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

nel caso di numeri complessi il comando `abs` restituisce il modulo complesso del valore in input secondo la formula:

$$|x| = \sqrt{\Re^2 + \Im^2} \quad (3)$$

ad esempio:

```
--> x = 3+4*i;
```

```
--> abs(x)
```

```
ans =
```

```
5.
```

```
--> x = 9+3*i;
```

```
--> abs(x)
```

```
ans =
```

```
9.486833.
```

3.2 Funzioni matematiche elementari

- `sqrt`, calcola la radice quadrata di uno scalare, o dei singoli elementi di vettori e matrici

```
--> a=4;
```

```
--> sqrt(a)
```

```
ans=
```

```
2.
```

```
--> v=[3 4 24];
```

```
--> sqrt(v)
```

```
ans=
```

```
! 1.732  2.000  4.89 !
```

```
--> A=[23 -1 8; 7 33 29; -1 5 9];
```

```
--> sqrt(A)
```

```
ans=
```

```
! 4.795.  i.      2.828. !
```

```
! 2.645.  5.744.  5.385. !
```

```
! i      2.236.  3.000. !
```

- `exp`, `exp(x)` calcola il valore e^x , se il parametro x è un vettore o una matrice, la funzione viene calcolata per ogni singolo elemento.
- `log`, `log(x)` calcola il logaritmo naturale e del parametro x . Se l'input è un vettore o matrice, la funzione viene ripetuta per tutti gli elementi.
- `log10`, `log10(x)`, calcola il logaritmo in base 10 del parametro x ; se l'input è un vettore o una matrice, la funzione viene ripetuta per tutti gli elementi;

- `log2`, `log2(x)`, calcola il logaritmo in base 2 del parametro x ; se l'input è un vettore o una matrice, la funzione viene ripetuta per tutti gli elementi.

Da ricordare la conversione da scala lineare a dB e viceversa:

- lineare \rightarrow dB $\Rightarrow 20\log_{10}x$
 \rightarrow `xdb = 20*log10(x)`
- dB \rightarrow lineare $\Rightarrow 10^{(x/20)}$
 \rightarrow `xlin = 10^(xdb/20)`

- `sin` (`cos`) calcola il seno (coseno) della variabile in input espressa in radianti;
- `asin` (`acos`) calcola l'arcoseno (l'arcocoseno) della variabile in input appartenente all'intervallo $[-1;1]$;
- `tan` calcola la tangente della variabile in input espressa in radianti
- `real()`, calcola la parte reale di un numero complesso;
- `imag()` calcola la parte immaginaria di un numero complesso;
- `dbphi()`, restituisce modulo in dB e fase in gradi di un numero complesso o di un vettore di numeri complessi (`[db,phi] = dbphi(input)`).

3.3 Funzioni particolari per matrici

Esistono funzioni particolari per definire vettori e matrici:

- `zeros(n,m)`, restituisce una matrice di n righe ed m colonne i cui coefficienti sono tutti zero

```
--> v = zeros(1,3)
```

```
v =
```

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

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

```
A =
```

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

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

con il comando `zeros(A)` si genera una matrice di dimensioni pari a quelle di A , di elementi nulli;

- `ones(n,m)`, restituisce una matrice di n righe ed m colonne i cui coefficienti sono tutti uno

```
--> v = ones(1,3)
```

```
v =
```

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

```
--> A = ones(3,3)
```

```
A =
! 1. 1. 1. !
! 1. 1. 1. !
! 1. 1. 1. !
```

con il comando `ones(A)` si genera una matrice, di dimensioni pari a quelle di A , di elementi tutti di valore uno;

- `eye(n,n)`, restituisce la matrice identità di dimensioni n

```
--> A = eye(2,2)
```

```
A =
! 1. 0. !
! 0. 1. !
```

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

```
B =
! 1. 0. 0. !
! 0. 1. 0. !
! 0. 0. 1. !
```

```
--> C = eye(3,5)
```

```
B =
! 1. 0. 0. 0. 0. !
! 0. 1. 0. 0. 0. !
! 0. 0. 1. 0. 0. !
```

- `rand(n,m)`, genera una matrice di dimensioni $n \times m$, di numeri casuali nell'intervallo $[0,1]$, se vengono omessi i valori n ed m di input, la funzione genera un numero casuale;
- `diag(matrice,k)`, restituisce un vettore colonna avente per elementi gli elementi della diagonale k -esima della matrice, dove $k = 0$ indica la diagonale principale, $k > 0$ indica la k -esima sopradiagonale mentre $k < 0$ indica la k -esima sottodiagonale. Se k viene omesso, si suppone $k=0$.
- `diag(v)`, restituisce una matrice avente come diagonale principale gli elementi del vettore v e gli altri elementi a zero;
- operatore `:`, serve per selezionare righe e colonne di matrici; data una matrice A si ottiene:

- `A(:)`, restituisce un vettore colonna composto dagli elementi delle colonne di A ;
- `A(i,:)`, restituisce un vettore contenente la i -esima riga di A ;
- `A(:,j)`, restituisce un vettore contenente la j -esima colonna di A ;
- `A(j:k)`, restituisce $[A(j),A(j+1),\dots,A(k)]$;
- `A(:,j:k)`, restituisce $[A(:,j),A(:,j+1),\dots,A(:,k)]$.

Nella tabella 1 sono riportate alcune operazioni possibili fra matrici e vettori. Alcuni esempi:

$$\rightarrow A = [1 \ 2 \ 3 \ ; \ 9 \ 8 \ 7 \ ; \ 4 \ 6 \ 5 \];$$

$$\rightarrow B = [4 \ 5 \ 9 \ ; \ 7 \ 3 \ 1 \ ; \ 8 \ 2 \ 4 \];$$

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 9 & 8 & 7 \\ 4 & 6 & 5 \end{pmatrix}; \quad B = \begin{pmatrix} 4 & 5 & 9 \\ 7 & 3 & 1 \\ 8 & 2 & 4 \end{pmatrix}; \quad (4)$$

$$A * B = \begin{pmatrix} 42 & 17 & 23 \\ 148 & 83 & 117 \\ 98 & 48 & 62 \end{pmatrix}; \quad B' = \begin{pmatrix} 4 & 7 & 8 \\ 5 & 3 & 2 \\ 9 & 1 & 4 \end{pmatrix}; \quad A^2 = \begin{pmatrix} 31 & 36 & 32 \\ 109 & 124 & 118 \\ 78 & 86 & 79 \end{pmatrix}; \quad (5)$$

$$[A, B] = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 9 \\ 9 & 8 & 7 & 7 & 3 & 1 \\ 4 & 6 & 5 & 8 & 2 & 4 \end{pmatrix}; \quad B(1,:) = (4 \ 5 \ 9); \quad A(:,1:2) = \begin{pmatrix} 1 & 2 \\ 9 & 8 \\ 4 & 6 \end{pmatrix}; \quad (6)$$

[]	definizione di matrice, concatenazione
;	separatore di riga
()	estrazione, m=a(k)
()	inserimento, a(k)=m
'	trasposta
+	somma
-	sottrazione
*	moltiplicazione
\	left division
/	right division
^	esponenziale
.*	moltiplicazione per elemento
.\	left division per elemento
./	right division per elemento
.^	esponenziale per elemento
.*	prodotto di Kronecker
./	right division di Kronecker
.\	left division di Kronecker
\	risolve il sistema lineare Ax=b (x=A\b)

Tabella 1: Operazioni fra matrici

4 Polinomi

In Scilab i polinomi vengono definiti tramite la funzione *poly*:

```
--> p = poly([0 2 3], 'x');
```

definisce un polinomio nella variabile x con radici in 0, 2 e 3, con il coefficiente del termine di grado massimo sempre pari ad uno.

Nel caso in cui non siano note le radici è possibile definire la variabile letterale di interesse

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

e quindi direttamente il polinomio

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

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

```
--> [r] = roots(p)
```

```
r =
```

```
! -1 !
```

```
! -2 !
```

Si ottiene quindi un vettore i cui elementi rappresentano le radici del polinomio.

Come per gli scalari e matrici, anche per i polinomi, si possono effettuare le operazioni di base di addizione, sottrazione, moltiplicazione e divisione

```
--> r = x/p
```

$$\frac{x}{2 + 3x + x^2} \quad (7)$$

```
--> r = x*p
```

$$2x + 3x^2 + x^3 \quad (8)$$

È inoltre possibile selezionare il numeratore ed il denominatore di una funzione razionale rispettivamente con i comandi *polinomio.num* e *polinomio.den*. --> $r = x/p$

$$\frac{x}{2 + 3x + x^2} \quad (9)$$

```
--> r.num
```

```
ans =
```

```
  x
```

```
--> r.den
```

```
ans =
```

```
  2+3x+x^2
```

5 Grafici

Per fare il plot di funzioni, Scilab, offre due comandi fondamentali

- `plot2d(x,y);`
- `plot3d(x,y);`

Queste due funzioni offrono la possibilità di personalizzare alcuni aspetti del grafico quali il font dei caratteri, colori e tratti delle linee, tipi di scala, titoli, griglie e tanto ancora.

5.1 Grafici 2D

Il modo più semplice per tracciare un grafico è tramite il comando `plot(x)` in cui x è un vettore. Si digitino ad esempio nel work space le seguenti linee di codice:

```
--> y = sqrt(linspace(0,4,100));
--> plot(y)
```

Si crea un grafico composto da una linea che unisce i punti che hanno come ordinata gli elementi di y e come ascissa i numeri da 1 alla dimensione di y .

Per tracciare un grafico di una linea specificando due dimensioni, è possibile definire due vettori da usare come valori per le ascisse e per le ordinate:

```
--> x = 1:0.1:10;
--> y = x.^2-5*x;
--> plot(x,y)
```

Commentiamo le seguenti righe di codice da digitare nel work space

```
--> x=[0:.1:10];
--> y=sin(x);
--> plot2d(x,y);
```

con il primo comando viene creato un vettore x di valori da zero a dieci con passo 0.1; viene poi calcolato, con il secondo comando, il *seno* di questi valori, ed infine, con il terzo comando, viene plottato tale funzione, il risultato è mostrato in **Figura 1**

A questo punto è possibile inserire un titolo al grafico

```
--> xtitle('Grafico della funzione f(x)=sin(x)');
```

e la griglia

```
--> xgrid(1);
```

Il risultato lo si può vedere in **Figura 2**

Con il comando `plot2dl(x,y)` è possibile creare un grafico con scala logaritmica.

Con SCILAB, una volta tracciato un grafico, se la finestra del grafico non viene chiusa, la successiva istruzione `plot` disegna nello stesso grafico. Per aprire una seconda finestra grafica basterà dare i comandi `xset('window', 2); xselect(2)` che rispettivamente istanziano una seconda finestra e la selezionano. Per tornare alla prima finestra (che si apre automaticamente con l'uso dei comandi di plot visti prima) basterà impartire il comando `xselect(1)`. Attraverso la funzione `xbasc()` è possibile 'pulire' l'eventuale finestra-grafico già aperta evitando così la sovrapposizione di più grafici.

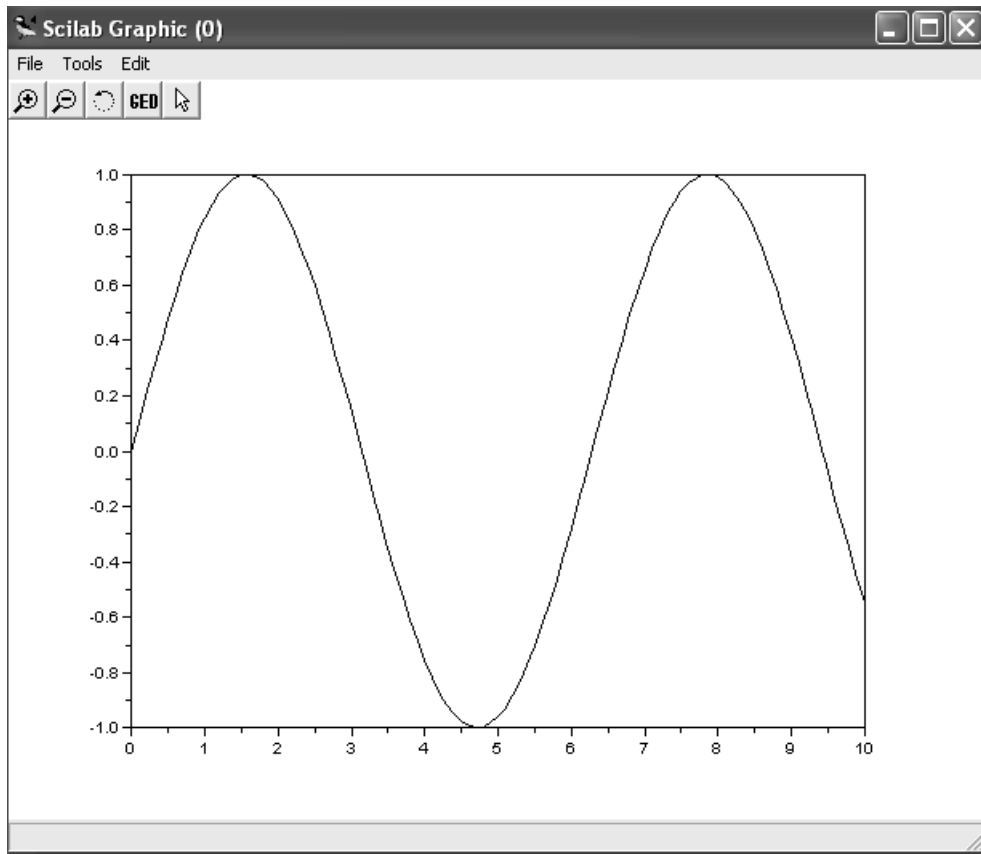


Figura 1: *Grafico di una Sinusoide.*

5.2 Grafici 3D

In 3D è possibile tracciare due tipi di grafici, ossia *curve in tre dimensioni* oppure *superfici in tre dimensioni*;

5.2.1 Curve in 3D

Una curva 3D può essere definita come grafico i cui punti hanno tre coordinate

$$[x(t)y(t)z(t)] \quad t \in \mathfrak{R} \quad (10)$$

in modo da definire la curva parametrica $c : \mathfrak{R} \rightarrow \mathfrak{R}^3$. Per un esempio si tracci la curva parametrica $[\cos(t) \sin(t) \sqrt{t}]$ usando il comando `param3d` come illustrato nell'esempio 4 del capitolo 9.

5.2.2 Superfici in 3D

Una funzione reale di due variabili $f : \mathfrak{R}^2 \rightarrow \mathfrak{R}$ descrive una superficie nello spazio \mathfrak{R}^3 . I punti di questa superficie hanno coordinate $[x \ y \ f(x,y)]$. Attraverso il comando `plot3d(x,y,z)`

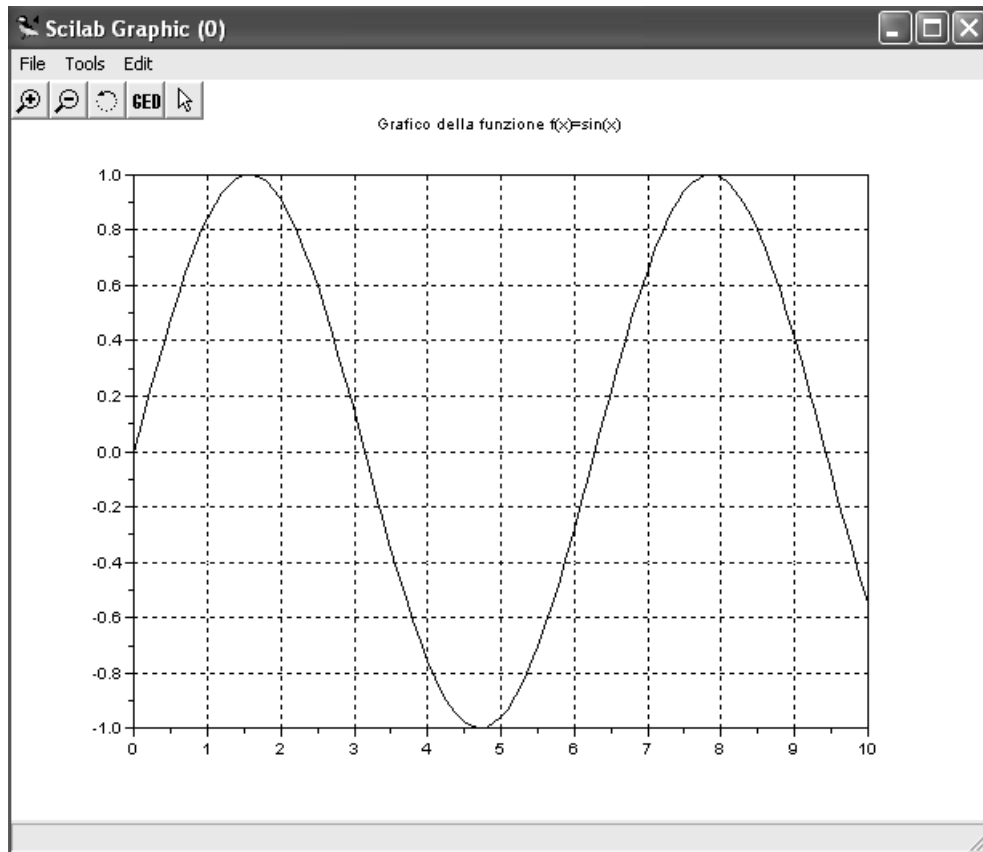


Figura 2: Grafico di una senoide con titolo e griglia.

si crea il grafico di x , y e z matrici di punti in cui z è una funzione di x ed y (si veda l'esercizio 5 del capitolo 9).

6 Scripting e Funzioni

Scilab mette a disposizione un comodo editor in cui è possibile elencare una successione di comandi in un file per poi poterli eseguire sequenzialmente; questi file risultano semplici file di testo con estensione `.sce` che si possono eseguire tramite comando `load into scilab` presente nel menu *Execute* dell'editor stesso oppure da richiamare dal workspace tramite comando `exec`:

```
exec(fun[,mode])
```

in cui *fun* è il nome della funzione di scilab ed il parametro *mode* può assumere uno dei seguenti valori:

- 0 : valore di default;
- -1 : senza visualizzare nessun dato;
- 1 : echo di ogni linea di comando;

- 2 : visualizza il prompt -->;

È inoltre possibile creare librerie di funzioni inserite in file include di estensione *.sci* per essere poi richiamate da altri programmi. Il comando `getf('filename.sci')` carica in memoria le funzioni contenute nella libreria `filename.sci`. Ovviamente, il nome della funzione che dovrà essere utilizzato è quello definito come nome funzione all'interno della libreria e non il nome del file. Un file, peraltro, può contenere più definizioni di funzioni. La definizione di una funzione è molto semplice:

```
function [a,b,...] = nome_funzione(c,d,...)
```

ed è composta da

- la parola chiave *function*;
- il nome della funzione;
- le variabili di input *a,b,...*;
- le variabili di output *c,d,...*;
- variabili locali, da usare quindi soltanto all'interno della funzione;
- variabili globali, da usare anche al di fuori della funzione;
- istruzioni.

Ad esempio, si scriva una funzione che, presi come valori di input due numeri reali x ed y restituisca il valore $z = x^2 - (x - y)^3$. (Si veda l'esempio 6 del capitolo 9)

7 Cicli for, while e istruzioni condizionali

Descriveremo in questo paragrafo i comandi base per eseguire cicli e istruzioni condizionali, dandone una breve descrizione ed alcuni esempi d'uso.

- *select-case*:

```
select x
case1
//istruzioni
case2
//istruzioni
case3
//istruzioni
end //end select x
```

Con questo codice, a seconda del valore assunto dalla variabile x , vengono eseguite determinate istruzioni; lo stesso risultato lo si può ottenere con i seguenti comandi:

```
if x==1
//istruzioni
elseif x==2
//istruzioni
elseif x==3
//istruzioni
end //endif
```

- *if-then-else*:

```
x = 1;
if x > 0
then
y = -x;
else
y = x;
end
```

L'output per questo codice è il seguente:

```
y = -1.
```

- *for*:

```
x = 1;
for k = 1 : 4;
x = x*k
end
```

L'output per questo codice è il seguente:

```
x = 1.
x = 2.
x = 6.
x = 24.
```

- *while*, il ciclo while esegue ripetutamente una sequenza di comandi fino a quando la condizione non è soddisfatta:

```
x = 1;
while x < 14;
x = 2*x
end
```

L'output per questo codice è il seguente:

```
x = 2.
x = 4.
x = 8.
```

x = 16.

8 Control Toolbox

Il control toolbox di Scilab è un *pacchetto* di funzioni che permette l'analisi di sistemi dinamici sia nel continuo che nel discreto. È quindi possibile in particolare definire modelli di sistemi lineari ed analizzarne la risposta nel tempo ed in frequenza, eventualmente allo scopo di sintetizzare sistemi di controllo usando le tecniche classiche di progetto ed analizzandone le prestazioni quali la risposta nel tempo, in frequenza, i margini di stabilità, etc...

8.1 Definizione di un sistema lineare

I sistemi che andremo a definire, sotto forma di funzione di trasferimento, sono di due tipi, a *tempo continuo* e a *tempo discreto*. Per definire una funzione di trasferimento si usa il comando `syslin` in uno dei seguenti modi:

```
[sl]=syslin(dom,A,B,C [,D [,x0] ]);  
[sl]=syslin(dom,N,D);
```

in cui:

- *dom* può assumere il valore 'c' per un sistema a tempo continuo, 'd' per un sistema a tempo discreto, oppure un valore scalare T per definire un sistema a tempo discreto con passo di campionamento associato T ;
- A , B , C , D sono le matrici del sistema in forma di spazio di stato (la matrice D è opzionale, se non specificata assume valore zero);
- x_0 è un vettore contenente lo stato iniziale del sistema (valore di default pari a zero);
- N , D sono polinomi rispettivamente del numeratore e denominatore del sistema in forma di funzione di trasferimento.
- *sl* rappresenta il sistema lineare risultato.

Ad esempio, per definire un sistema a tempo continuo come funzione di trasferimento nella variabile s , si possono usare le seguenti righe di codice:

```
--> s = poly(0,'s');  
--> sl = syslin('c',1,(s * s + 0.2 * s + 1))
```

ottenendo la funzione di trasferimento

$$sl = \frac{1}{s * s + 0.2 * s + 1} \quad (11)$$

8.2 Funzioni per la manipolazione e l'analisi nel tempo e in frequenza di sistemi lineari

Sia definita la funzione di trasferimento di un sistema sl ed un vettore contenente gli istanti di tempo t ,

- Calcolo della *risposta al gradino* in un vettore y :
--> $t = [0:0.1:50]$;
--> $y = \text{csim('step',t,sl)}$
- Calcolo della *risposta impulsiva* e suo grafico:
--> $t = [0:0.1:50]$;
--> $y_i = \text{csim('imp',t,sl)}$
--> $\text{xbasec}()$;
--> $\text{plot2d}(t',y_i')$
- il comando $\text{slstato}=\text{tf2ss}(sl)$, dove sl è un sistema in forma di funzione di trasferimento, calcola una realizzazione di stato slstato del sistema; il viceversa è realizzato tramite il comando $\text{sl}=\text{ss2tf}(slstato)$;
- Interconnessione in *retroazione*:
 $w = \text{sl}/.s2$
questo comando mette il sistema lineare $s2$ in retroazione negativa (per default) con il sistema lineare $s1$ e calcola la funzione di trasferimento ad anello chiuso w ($W = S1 * (I + S2 * S1)^{-1}$);
- *cascata*:
 $w = s2*s1$
questo comando mette il sistema lineare $s2$ in cascata al sistema lineare $s1$;
- *parallelo*:
 $w = s1+s2$
questo comando mette il sistema lineare $s2$ in parallelo con il sistema lineare $s1$;
- $\text{nyquist}(sl)$:
 $\text{nyquist}(sl, \text{frq} [, \text{comments}])$
Traccia il diagramma di Nyquist del sistema lineare sl , in cui frq un vettore di frequenze (espresse in Hertz) e comments un vettore di caratteri di commento;
- $\text{bode}(sl)$:
 $\text{bode}(sl, \text{frq} [, \text{comments}])$
Traccia il diagramma di Bode del sistema lineare sl , in cui frq è un vettore contenente le frequenze (in Hertz) e comments è un vettore di caratteri;
- $\text{black}(sl, \text{list})$:
 $\text{black}(sl, [\text{fmin}, \text{fmax}] [, \text{comments}])$

Traccia il diagramma di Nichols del sistema lineare sl in cui sl è un sistema lineare, $fmin$ ed $fmax$ sono i limiti per la frequenza (espressi in Hertz) e $comments$ un vettore di caratteri;

- `chart(gain, phase, flags)`

Sovrappone la carta di Nichols al diagramma. Il vettore `gains` indica quali luoghi a modulo costante (in dB) vengono tracciati. Il vettore `phase` indica quali luoghi a fase costante vengono tracciati. Il campo `flags` è un vettore di quattro componenti:

- `sup`: 1 per sovrapporre la carta al grafico presente, 0 per nessuna sovrapposizione;
- `leg`: 1 imposta la legenda, 0: grafico senza legenda;
- `cm`: colore per le curve del guadagno (vedi `plot2d`);
- `cphi` colore per le curve della fase (vedi `plot2d`);

- `rf=repfreq(sl,w)`: valuta la risposta in frequenza del sistema lineare `sl` nel vettore di frequenze `w` (in Hz), fornendo in uscita in `rf` per ogni frequenza il numero complesso corrispondente. È naturalmente conveniente convertire il risultato in modulo (dB) e fase (gradi), tramite la funzione

- `[modulodb,fase]=dbphi(rf)`

- `[fasewa,fa]=p_margin(L)`: calcola la fase `fasea` (in gradi) del sistema in retroazione unitaria con guadagno d'anello `L` assunta alla frequenza di attraversamento `fa` (espressa in Hz). Il margine di fase `mf` e la pulsazione di attraversamento `wa` sono allora dati dal seguente calcolo:

```
[fasewa,fa]=p_margin(L);  
wa=2*%pi*fa;  
mf=180+fasewa;
```

- `[mg,fpi]=g_margin(L)`: calcola il margine di guadagno `mg` (in dB) del sistema in retroazione unitaria con guadagno d'anello `L` e la relativa frequenza di attraversamento dell'asse reale `fpi` (in Hz).

8.3 Funzioni modificate per la risposta in frequenza

In scilab, per default, i grafici della risposta in frequenza del control toolbox hanno la scala graduata in *Hertz*; insieme al materiale del corso di PSC sono distribuite le seguenti funzioni

- `r_black`;
- `r_bode`;
- `r_nyquist`;
- `r_repfreq`;

che tarano la scala in radianti al secondo e che quindi possono essere usate al posto di quelle precedenti. Queste funzioni sono tutte presenti in un unico file di nome *psc.sci*; per caricarle è sufficiente usare il comando `get`:

```
-->getf('<path>/psc.sci');
```

dove `<path>` è il percorso su disco che contiene il file *psc.sci*.

9 Esempi Svolti

1. Definizione di un sistema lineare in funzione di trasferimento ed uso delle funzioni fondamentali per la risposta nel tempo:

```
// Definizione di 's' come polinomio, in modo da poterlo usare
// nelle manipolazioni algebriche
```

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

```
// Definizione di due polinomi N(s) e D(s) nella variabile s
```

```
N=s+1
```

```
D=s^2+s+1
```

```
// Pausa di esecuzione. Dare il comando resume da input o da menu
```

```
//per continuare
```

```
pause
```

```
// Definizione della funzione di trasferimento P(s)
```

```
//di un sistema a tempo
```

```
// continuo avente num N e den D
```

```
P=syslin('c',N,D)
```

```
// 'c' significa a tempo continuo
```

```
pause
```

```
// Definizione di un intervallo temporale da 0 a 10 secondi
```

```
// con passo di 0.01 secondi
```

```
// Ne risulta un vettore di 1001 elementi equispaziati, ovvero
```

```
// una matrice 1x1001, come si vede dall'output di size
```

```
t=[0:0.01:10];
```

```
size(t)
```

```
// Simulazione della risposta all'impulso della f.d.t. P(s)
```

```
// ed assegnazione dei valori simulati al vettore pt,
```

```
// che ha le stesse dimensioni di t e contiene in ogni elemento
```

```
// il valore della risposta all'istante del corrispondente
```

```
// elemento di t
```

```
pt=csim('impuls',t,P);
```

```
// Nuova finestra grafica
```

```
xbasc(0);xset('window',0);xselect();
```

```
//Grafico della risposta pt vs. l'asse temporale
```

```

plot2d(t,pt)
pause
// Simulazione della risposta al gradino unitario
pst=csim('step',t,P);
// Nuova finestra grafica
xbasc(1);xset(window,1);xselect();
// Grafico della risposta st vs. l'asse temporale
plot2d(t,pst)
pause
// Definizione dei campioni di un segnale a rampa u(t)= alfa*t
// sui campioni del vettore dei tempi gia' definito
alfa=1;
rampa=alfa * t;
// Simulazione della risposta del sistema ad un ingresso
// qualunque, nel nostro caso la rampa
rt=csim('rampa',t,P);
// Nuova finestra grafica
xbasc(2);xset(window,2);xselect();
// Grafico della risposta alla rampa rt
// leg = Etichetta della figura
plot2d(t,rt,leg='Risposta alla rampa');
pause
//Tante figure insieme!
xbasc(3);xset(window,3);xselect();
plot2d(t,[pt' pst' rt'],leg=Impulso@Gradino@Rampa);

```

2. //Calcolo di interconnessioni di funzioni di trasferimento

```

getf('psc.sci');
// Definizione di s
s=poly(0,'s')
// Definizione di una f.d.t. P1...
P1=syslin('c',s+1,s^2+s+1)
// ... e di una P2
P2=syslin('c',1,s+2)
pause
// Cascata P1*P2
CASCATA=P1*P2
// Parallelo P1+P2
PARALLELO=P1+P2
// Retroazione negativa: P1 in catena diretta
// e P2 in catena inversa

```

```

RETROAZIONE=P1/.P2
// Un impianto P
P=P1
// ed un controllore
C=P2
// Il guadagno d'anello L
L=C*P
// Una f.d.t. di valore unitario
UNO=syslin('c',1,1);
// La f.d.t. L chiusa in retroazione unitaria
W=L/.UNO
// La funzione di sensitivita' dello schema in retroazione
// unitaria avente L come guadagno di anello
S=UNO/.L
// La funzione di sensitivita' dell'uscita ad un disturbo
// sull'ingresso all'impianto
SP=P/.C

3. // Funzioni relative alla risposta in frequenza
// di sistemi lineari a tempo continuo
// Carica le funzioni modificate per esprimere la
// risposta in frequenza in funzione di omega (rad/s)
getf(psc.sci);
s=poly(0,'s')
// Una f.d.t. L(s)
L=syslin('c',(s+10)^2,s*(s+1)^4)
pause
// Diagrammi di Bode
r_bode(L)
pause
// Diagramma di Nyquist
xbasc(1);xset('window',1);xselect();
r_nyquist(L)
pause
// Diagramma di Nyquist nell'intervallo di pulsazioni
// specificato
wmin=0.1; wmax=10;
xbasc(1);xset('window',1);xselect();
r_nyquist(L,wmin,wmax)
pause
// Diagramma di Nichols

```

```

xbase(2);xset(window,2);xselect();
wmin=0.1; wmax=10; r_black(L,wmin,wmax)
pause
// Con carta di Nichols sovrapposta...
chart(list(1,0));

```

4. Comando param3d per grafici 3D

```

// Definizione di un vettore t
t=[0:0.1:20];
// calcolo il coseno, il seno e la radice quadrata di ogni valore di t
x=cos(t);
y=sin(t);
z=sqrt(t);
//con il comando param3d si plotta la curva parametrica
param3d(x,y,z)

```

5. Superfici in 3 dimensioni

```

// inizializzo due vettori x ed y da 0 a 10
//con paso 0,25
x=[0:0.25:10];
y=x;
//creo una funzione in x ed y
deff('w=f(x,y)', 'w=x^2+y');
z=eval3d(f,x,y)
// la funzione eval valuta
// la funzione f di due variabili in una
// griglia di punti definita dai vettori x ed y;
//il comando surf plotta superfici parametriche 3D
//usando una griglia rettangolare di punti definita dai valori
//di X ed Y
surf(x,y,z)
//il comando plot3d traccia il grafico della
//superficie parametrica z=f(x,y)
plot3d(x,y,z)

```

6. //File .sci per una funzione f(x,y) che ha come valori

```

//di input due numeri reali x ed y
//e restituisce il valore  $z = x^2 - (x - y)^3$ 
// Dopo aver aperto l'editor si digiti
function [z] = f(x,y)
// con questo comando viene definita
//una funzione di nome f che prende

```

```
//x ed y come valori di input
//e restituisce in output il valore
//z come di seguito definito:
z = x^2 - (x-y)^3
//Da work space si digiti
--> getf('nome_file.sci')
//digitando fra apici il nome del file .sci per
//caricare la funzione desiderata e, sar  possibile
//iniziare ad usarla ottenendo, ad esempio i seguenti risultati:
-->f(2,3)
ans =
5.
-->f(10,4)
ans =
-116.
```

Riferimenti bibliografici

- [1] "<http://www.scilab.org/events/leaflet/brochure-it.pdf>."
- [2] "<http://www.scilab.org>."
- [3] "<http://www.scilab.org/doc/intro/intro.html>."
- [4] C. Gomez, C. Bunks, J. Chancelier, F. Delebecque, M. Goursat, R. Nikoukhah, and S. Steer., *Engineering and Scientific Computing with Scilab*. Birkhauser, 1999.
- [5] S. L. Campbell, J.-P. Chancelier, and R. Nikoukhah, *Modeling and simulation in Scilab/Scicos*. Springer, 2005.