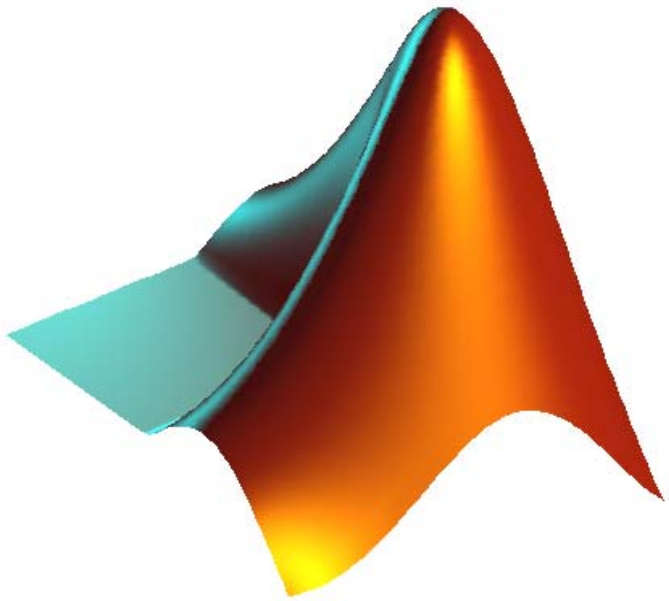
UNIVERSITÀ degli STUDI "MAGNA GRÆCIA"
di CATANZARO

Corso di Laurea in Ingegneria Informatica e Biomedica

Prof. Francesco AMATO

Esercitazioni di MATLAB/Simulink®



Dispensa a cura dell'Ing. Alessio MEROLA

INDICE

Capitolo 1

Generalità

1.1 Matlab overview

MATLAB nasce negli anni settanta come linguaggio di programmazione dedicato al calcolo matriciale, algebrico e per l'analisi numerica (da qui la denominazione che deriva dall'acronimo di MATrix LABoratory). Negli anni, accanto allo sviluppo di routine numeriche del linguaggio sempre più efficienti, MathWorks (software house proprietaria di MATLAB http://www.mathworks.com/) ha aggiunto un ambiente software che utilizza tale linguaggio.
Nella sua veste attuale, l'ambiente interattivo di MATLAB consente di svolgere calcoli matematici, permette di gestire variabili e di importare ed esportare dati, di generare diagrammi e creare grafica avanzata, con la possibilità di ampliare le sue funzioni tramite moduli software aggiuntivi (i Toolbox).

Principali ToolBox di MATLAB

System Identification Toolbox
Control System Toolbox
Model Predictive Control Toolbox
Frequency Domain Identification Toolbox
Nonlinear Control Design Blockset
Neural Network Toolbox
Signal Processing Toolbox
Optimization Toolbox
Robust Control Toolbox

In particolare, MATLAB include un importante ambiente GUI per la modellistica e la simulazione di sistemi dinamici (SIMULINK) che si interfaccia facilmente con le altre funzionalità proprie del software.

1.2 L'interfaccia grafica di MATLAB

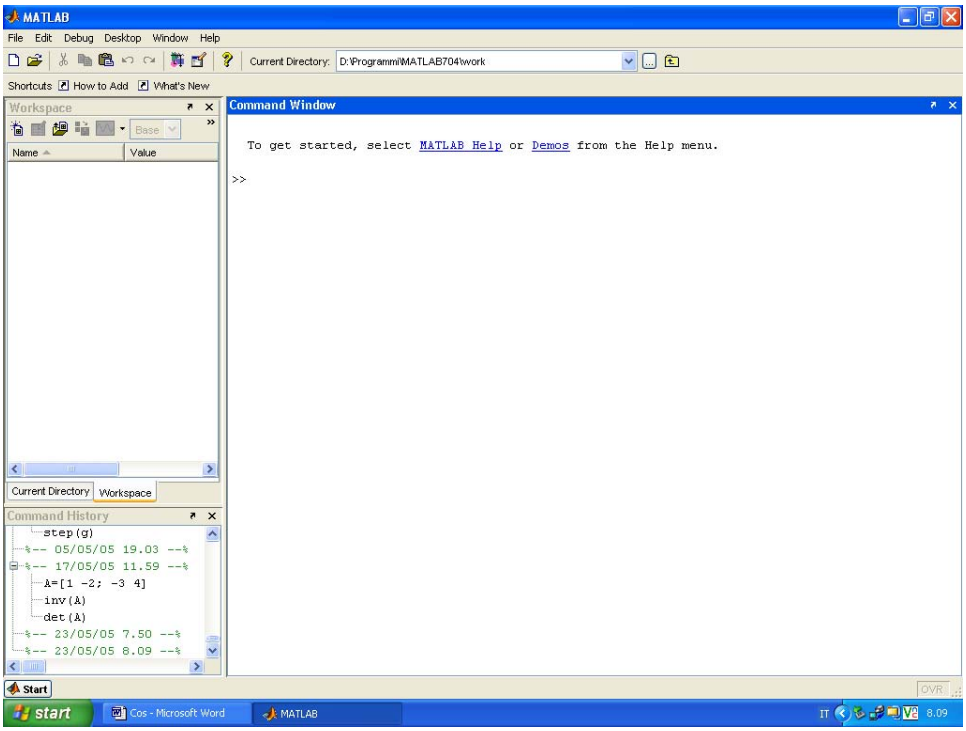


Figura 1.1 Interfaccia grafica di MATLAB

L'interfaccia grafica di MATLAB si compone di più parti, ma quella fondamentale è sicuramente la **`Command Window`** che consente l'immissione dei comandi.
Per visualizzare questa sola finestra a tutto schermo, si entri nel menu a tendina **`Desktop`**→**`Desktop Layout`** e si selezioni l'opzione **`Command Window Only.`**
Immettendo una semplice operazione matematica (ad es. 2+3*6) in corrispondenza del prompt **`>>`** si farà lavorare MATLAB "in modalità calcolatrice".
Il risultato del calcolo verrà comunicato con:

```
>>ans=
       20
```

Ciò significa che MATLAB ha provveduto a salvare il risultato dell'operazione nella variabile temporanea answer.
In alternativa, se durante tutta una sessione di MATLAB si vuole memorizzare il risultato di un'operazione in una variabile permanente, basta effettuare una semplice operazione di assegnazione:

```
>>a=2+3*6;
```

Con questa istruzione è stato assegnato il risultato dell'operazione alla variabile **`a`**, senza peraltro visualizzarne il valore a schermo, avendo posposto il punto e virgola.
Ovviamente, in questo caso la variabile **`a`** è un semplice scalare ma in generale MATLAB tratta diversi tipi di variabili, come verrà ampiamente descritto nel capitolo 2.

MATLAB impone delle regole di assegnazione del nome delle variabili. Infatti, i nomi da assegnare alle variabili devono iniziare con una lettera e devono contenere meno di 20 caratteri.
Per "pulire a schermo" la finestra dei comandi dall'immissione di precedenti istruzioni o dall'output di risultati pregressi, si digiti al prompt:

```
>>clc
```

Nella dichiarazione delle variabili, è importante notare che MATLAB è *case sensitive* cioè distingue tra carattere maiuscolo e minuscolo per la definizione delle variabili.

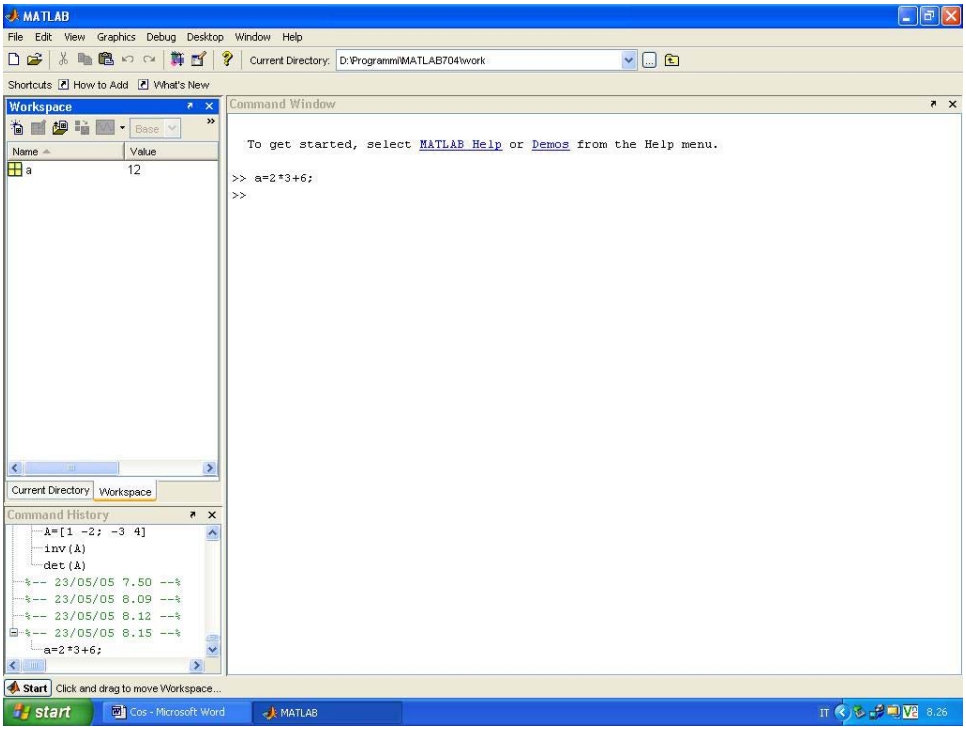


Figura 1.2 Finestra Workspace

Per visualizzare tutte le variabili dichiarate all'interno di una sessione si acceda al menu **Desktop** e si selezioni proprio **Workspace** (una delle finestre che con la precedente operazione era stata deselezionata). In questo modo, attraverso la finestra **Workspace** comparsa a sinistra della **Command Window** a seguito dell'operazione di selezione, si ha una utile panoramica delle variabili correnti nella sessione. Nel caso specifico, all'interno della stessa finestra, è possibile individuare la variabile **a,** settata precedentemente, e il suo valore numerico.
In alternativa, per conoscere il contenuto del workspace, si può digitare al prompt l'istruzione **who.** Analogamente, con **whos** è possibile visualizzare tutte le variabili memorizzate ed in più i loro valori numerici.

1.3 Salvataggio, lettura, eliminazione di variabili

Le variabili possono essere salvate e cancellate tramite due strategie differenti ma omologhe.
Una variabile può essere salvata cliccando sul tasto di salvataggio della finestra **`Workspace`** o in alternativa digitando al prompt dei comandi:

`>>save` *nomeFile var1 var2 var3 varN*

Con questa istruzione si è scelto convenientemente di salvare le variabili prescelte in un file *nomeFile*.
Se invece avessimo digitato solo **`save`**, MATLAB avrebbe autonomamente provveduto a salvare tutte le variabili presenti nel Workspace nel file di default MATLAB.MAT.

Per caricare nel workspace variabili precedentemente salvate è sufficiente digitare:

`>>load` *nomeFile*

Per modificare la directory corrente da tastiera si usi la sintassi:

`>>cd` *nomedir*

Per visualizzare il contenuto della stessa:

`>>dir` *nomedir*

Altrimenti, è possibile rendere corrente una determinata directory avvalendosi del menu a tendina chiamato **`Current Directory:`** presente sulla barra dei menu.
Le variabili correnti nella sessione possono essere eliminate insieme con **`Clear Workspace`** del menu **`Edit`** o in alternativa, molto più semplicemente, digitando al prompt **`clear all.`**

Per l'eliminazione di specifiche variabili:

`>>clear` *var1 var2 var3 varN*

Ancora più semplicemente, attivando la finestra Workspace, tutte le variabili o alcune possono essere eliminate selezionandole e cliccando sull'icona del cestino.

1.4 L'Help di MATLAB

Per accedere alla funzione Help del software è sufficiente entrare nel menu **`Help`** della barra dei menu.
Invece, digitando al prompt **`Help`** verrà invece restituito l'elenco degli argomenti inclusi nella guida di MATLAB e loro descrizione.
Invece, con la sintassi **`Help`** *nomefunzione* è possibile ottenere tutte le informazioni relative alla particolare funzione prescelta.

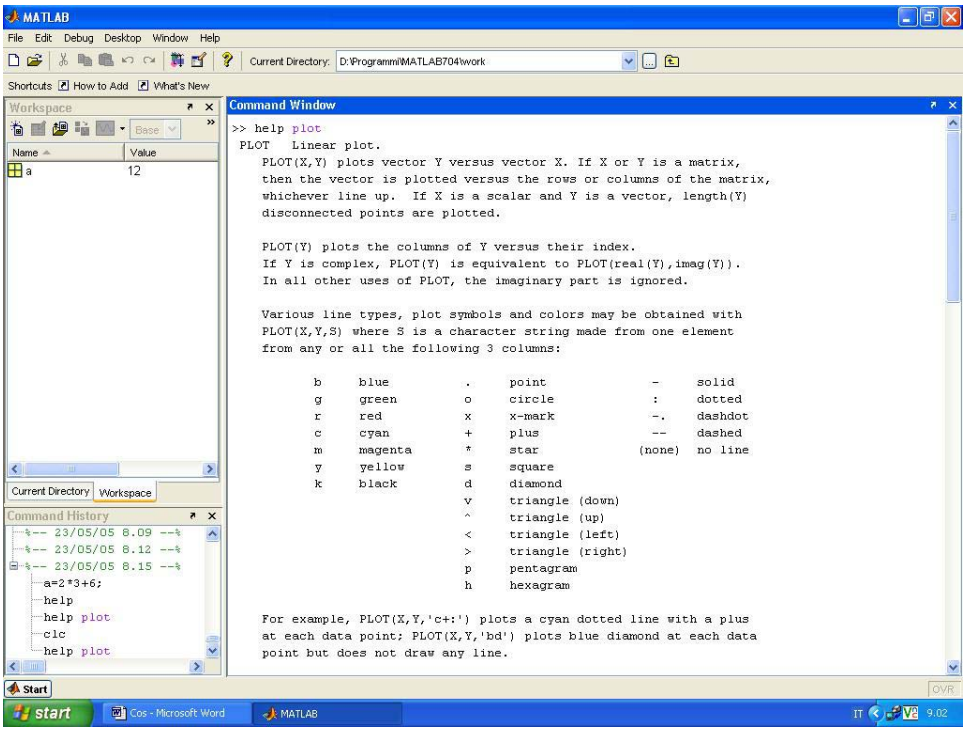


Figura 1.3 Help Plot

Infine, con **`lookfor`** *nomefunzione* è possibile ottenere tutte le ricorrenze della funzione prescelta nella guida

Capitolo 2

Editing di variabili

2.1 Gli array

Gli **array** sono un insieme di scalari ordinati spazialmente.
Array ad una dimensione sono detti *vettori riga* o *colonna*. Quando l'array ha almeno una dimensione è detto *matrice bidimensionale* o *tridimensionale* (in presenza della terza dimensione).
Ogni elemento di un array è identificato tramite un indice.
Ad esempio, l'accesso ad un singolo elemento di un vettore (array monodimensionale) è possibile con la sintassi **`A(2)`**, dove 2 è l'indice dell'elemento del vettore (il secondo elemento in questo caso).
Invece, per identificare univocamente un elemento di una matrice sono necessari un indice di riga e uno di colonna. Per cui, il comando **`B(2,3)=0`** va ad assegnare 0 all'elemento della matrice **`B`** che occupa la posizione in corrispondenza della seconda riga e della terza colonna.
Importante è l'operatore colon (:) che consente la selezione di un intera colonna di un array.
Pertanto con l'istruzione:

```
>>A(:,2)=[]
```

si vanno ad eliminare con l'array vuoto **`[]`** tutti gli elementi appartenenti alla seconda colonna della matrice A.
Invece, con:

```
>>x(1:3)
```

si selezionano i primi tre elementi del vettore x.

Memorizzazione di un vettore

L'operazione più semplice di memorizzazione di un vettore riga è la seguente:

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

La risposta di avvenuta memorizzazione data da MATLAB è la seguente:

```
>>A=
    1   2   3   4
```

Nel caso si voglia creare un vettore colonna, basta interporre tra gli elementi del vettore un segno di punto e virgola.

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

Per definire un vettore i cui elementi sono distribuiti secondo una certa legge, le sintassi per i diversi casi sono le seguenti:

- **vettore con elementi equispaziati ad incremento unitario**
  - **`x=1:n`**
- **vettore con elementi equispaziati ad incremento non unitario**
  - **`x=xmin:incremento:xmax`**
  - **`x=linspace(xmin,xmax,n)`** dove **`n`** è il numero di elementi del vettore
- **vettore con elementi distribuiti logaritmicamente**
  - **`x=logspace(xmin,xmax,n)`**

Memorizzazione di una matrice

Per memorizzare una matrice del tipo:

$$M = \begin{pmatrix} 2 & 5 \\ -3 & 4 \end{pmatrix}$$

È conveniente usare la sintassi che consente l'immissione della matrice per righe, le quali sono separate da un punto e virgola.

```
>>M=[2 5;-3 4]
```

Memorizzazione di un array a più dimensioni

Alla struttura planare di una matrice è possibile aggiungere una o più dimensioni per creare *matrici tridimensionali* o nel caso più generale *matrici ad n-dimensioni*.
Un array tridimensionale consta di due o più "strati planari", cioè di due o più matrici bidimensionali sovrapposte in pila.
Perciò, si deve immettere innanzitutto il primo "strato":

```
>>A=[4 8;3 2];
```

Successivamente, si aggiunge il secondo "strato" con l'istruzione:

```
>>A=(:,:,2)=[5 6; 9 3]
```

In alternativa, è possibile utilizzare il comando **`cat`**, predefinendo due matrici A e B di pari dimensione, le quali andranno a rappresentare rispettivamente il primo e il secondo strato della matrice 3-D.

```
C=cat(3,A,B)
```

Comandi aggiuntivi

Per ottenere informazioni sulle dimensioni o su altre proprietà degli array, ovvero per creare una particolare disposizione dei loro elementi esiste una prima serie di comandi molto utili.

- **`max(A)/min(A)`**

I due comandi restituiscono rispettivamente il più grande e il più piccolo elemento di `A` se questo è un array e possiede tutti gli elementi reali.
In caso **`A`** abbia elementi complessi, i comandi restituiscono l'elemento con la massima/minima ampiezza.
Con le istruzioni:

```
[x,y]=max(A)
[x,y]=min(A)
```

si memorizzano nel vettore **`x`** gli elementi più grandi (o più piccoli con **`min`**) della matrice A e i loro indici nel vettore **`y`**.

- **`size(A)`**
  Restituisce un vettore riga che contiene le dimensioni della matrice **`A`**

- **`sort(A)`**
  Ordina in maniera crescente le colonne dell'array **`A`**

- **`sum(A)`**
  Restituisce un vettore riga in cui ciascun elemento rappresenta la somma della corrispondente colonna di **`A`**, se ha **`A`** è una matrice

- **`squeeze(A)`**
  Operazione per la compattazione di una matrice

2.2 Celle ed array di celle

È un tipo di dati non omogeneo, tramite il quale è possibile creare un insieme di dati in cui uno o più elementi sono diversi tra loro e che, a loro volta, possono essere costituiti da vettori, matrici e stringhe.
Quindi, a differenza dei tipi di dati fin qui visti, con l'array di celle è possibile creare una struttura dati che presenta la proprietà di disomogeneità tra gli elementi concatenati.
Ad esempio, con l'istruzione:

```
>>A={3,[1 4],[3 5;6 1],'stringa'}
```

si è creato l'array di celle **`A`** che annovera tra i suoi elementi uno scalare, un vettore riga una matrice e una stringa.

```
>>A=
   [3] [1x2 double] [2x2 double] 'stringa'
```

Per visualizzare il contenuto di un particolare elemento dell'array di celle (ad es. il secondo), si digiti:

```
A{2}
```

La risposta che si ottiene è la seguente:

```
>>ans=
       1   4
```

In alternativa, per visualizzare il contenuto di tutte le celle si usi **`celldisp`**.
Inoltre, per ottenere una rappresentazione grafica dello stesso contenuto si ha a disposizione **`cellplot`**.

2.3 Struct ed array di struct

Una **struct** rappresenta un tipo di variabile assimilabile al record dei linguaggi di programmazione come C o Pascal. All'interno del record (struct) sono presenti dei campi in cui possono essere memorizzati dei dati tra loro diversi.

Molto brevemente, supponiamo di costruire un elenco di scienziati famosi. Iniziamo così a memorizzare i primi due nodi della lista.

Nome	Albert
Cognome	Einstein
Luogo di nascita	Ulm
Data di nascita	
giorno	14
mese	3
anno	1879

Nome	Enrico
Cognome	Fermi
Luogo di nascita	Roma
Data di nascita	
giorno	29
mese	9
anno	1901

Si parte con la definizione della data che è gia di per sé un tipo di dato strutturato, componendosi dei campi giorno, mese, anno.

```
dataS1.giorno=14;
dataS1.mese=3;
dataS1.anno=1879;

S1.nome='Albert';
S1.cognome='Einstein';
S1.luogo='Ulm';
S1.nascita=dataS1;
```

Ugualmente per il secondo nodo:

```
dataS2.giorno=29;
dataS2.mese=9
dataS2.anno=1901

S2.nome='Enrico';
S2.cognome='Fermi';
S2.luogo='Roma';
S2.nascita=dataS2;
```

Per concatenare i due nodi e creare la lista di scienziati:

```
scienziati(1)=S1;
scienziati(2)=S2;
```

Come verifica dell'avvenuta memorizzazione si digiti al prompt:

```
>>scienziati
```

MATLAB restituisce:

```
1x2 struct array with fields:
   nome
   cognome
   luogo
   nascita
```

Per conoscere, ad esempio, il luogo di nascita di Albert Einstein è sufficiente digitare al prompt:

```
scienziati(1).luogo
```

Capitolo 3

Calcolo numerico

3.1 Operazioni con i numeri

MATLAB permette di effettuare tutte le operazioni numeriche adoperando i classici simboli della programmazione standard **(+ - * / ^)** e consente di applicare tali operatori a singoli operandi, quali che siano semplici scalari o più elementi appartenenti a un vettore o di una matrice.
MATLAB segue in generale le regole standard di precedenza nelle operazioni, ma ha in più l'operatore di divisione sinistra (****) che consente di anteporre il divisore rispetto al dividendo.
Ad esempio **2\5=2.5** è equivalente alla classica operazione **5/2=2.5**.

Alcuni semplici accorgimenti nel trattamento dei numeri complessi

Il trattamento dei numeri complessi in MATLAB si deve avvalere di alcuni semplici accorgimenti precauzionali, per non incorrere in errori di calcolo derivanti da alcune situazioni particolari.
Infatti, anche in MATLAB, i numeri complessi vengono identificati nella loro parte immaginaria secondo la comune convenzione di assegnare all'unità immaginaria la costante **i** o **j**.
Pertanto, siccome le due costanti predette sono già definite e non protette da scrittura, per l'utente, ai fini della correttezza del calcolo, non è conveniente definire altre variabili con lo stesso nome.
Quindi, se il coefficiente della parte immaginaria di un numero complesso è intero, si digita - ad esempio - 3+2**i** per definire un numero complesso avente parte reale 3 e parte immaginaria 2.
Invece, volendo definire un numero complesso a parte immaginaria frazionaria (es. 3/2), si dovrà digitare **3/2*i** e non **3/2i,** la quale ultima scrittura effettua l'operazione di divisione tra **3** e **2i**.

3.2 Variabili e costanti predefinite

Oltre a quelle già viste (**ans**, **i** e **j**), diverse sono le variabili predefinite in MATLAB.

Variabile	Descrizione
ans	variabile di default
i,j	unità immaginaria $\sqrt{-1}$
eps	precisione dei numeri decimali
pi	pi greco
inf	infinito
NaN	forma di indeterminazione (0/0)
realmax	massimo numero in virgola mobile
realmin	numero più prossimo allo zero macchina
date	data
clock	orologio
cputime	tempo di elaborazione

3.3 Funzioni matematiche di base

Per avere una panoramica delle funzioni matematiche incluse in MATLAB è sufficiente digitare al prompt:

- **`help elfun`** (funzioni matematiche di base)
- **`help specfun`** (funzioni matematiche specifiche)

Funzioni matematiche numeriche

Funzione	Descrizione
`round`	arrotondamento all'intero più vicino
`fix`	troncamento all'intero più vicino verso 0
`floor`	arrotondamento per difetto all'intero più vicino
`ceil`	arrotondamento per eccesso all'intero più vicino
`sign`	calcolo della funzione sign(x) +1 se x>0; 0 se x=0; -1 se x<0
`rem`	resto in una divisione intera
`rat`	espansione razionale
`rats`	approssimazione razionale
`gcd`	massimo comun divisore
`lcm`	minimo comune multiplo

Funzioni esponenziali e logaritmiche

Funzione	Descrizione
`exp`	esponenziale in base e
`pow2`	esponenziale in base 2
`sqrt`	radice quadrata
`log`	logaritmo naturale o neperiano
`log2`	Logaritmo in base 2
`log10`	logaritmo decimale o in base 10

Funzioni per numeri complessi

Funzione	Descrizione
`abs`	modulo
`angle`	angolo
`conj`	coniugato
`imag`	parte immaginaria
`real`	parte reale

Funzioni trigonometriche

Funzione	Descrizione
`sin`	seno
`cos`	coseno
`tan`	tangente
`cot`	cotangente
`sec`	secante
`csc`	cosecante
`asin`	arcoseno
`acos`	arcocoseno
`atan`	arcotangente
`atan2(y,x)`	arcotangente a 4 quadranti
`acot`	cotangente
`asec`	secante
`acsc`	cosecante

Tali funzioni accettano in ingresso, per default, variabili espresse in radianti.

Capitolo 4

Calcolo algebrico

4.1 Operazioni con gli array

Si intende per *operazioni con gli array* l'insieme delle operazioni effettuate elemento per elemento ed identificate premettendo a tutti gli operatori aritmetici - tranne che per addizione e sottrazione in cui esso non necessita - il carattere "**.**".
Infatti, utilizzando il semplice operatore di moltiplicazione (*), MATLAB esegue la moltiplicazione tra due array $\mathbf{x=(x_1,x_2,x_3)}$ e $\mathbf{y=(y_1,y_2,y_3)}$ come ***prodotto interno*** **o** ***scalare***.
Quindi, l'operazione **x*y** dà come risultato uno scalare di valore $\mathbf{ans=x_1y_1+x_2y_2+\ x_3y_3}$.

```
>> x=[1,2,3];
>> y=[3;2;1];
>> x*y

ans =

    10
```

Si noti che per poter effettuare il prodotto, il numero di righe di **y** deve essere pari al numero di colonne di **x**.
Diversamente, per ottenere una moltiplicazione di array in cui ogni elemento della matrice C è ottenuto effettuando il prodotto elemento per elemento delle matrici di partenza A e B (cioè $c_{ij}=a_{ij}b_{ij}$), si segua il seguente esempio:

```
>> A=[1,2;3,4];
>> B=[2,1;4,3];
>> C=A.*B

C =

     2     2
    12    12
```

Più in generale, nella tabella successiva sono raccolti tutti i tipi di operazione elemento per elemento che si possono effettuare in MATLAB.

Simbolo	Descrizione	Operazione
+	Somma array-scalare	**A+b**
-	Sottrazione array-scalare	**A-b**
+	Somma	**A+B**
-	Sottrazione	**A-B**
.*	Moltiplicazione elemento-elemento	**A.*B**
./	Divisione a destra elemento-elemento	**A./B**
.\	Divisione a sinistra elemento-elemento	**A.\B**
.^	Elevamento a potenza	**A.^B**

Operazioni con gli array elemento per elemento

In tutte le operazioni condotte elemento per elemento, gli array devono avere le stesse dimensioni.

4.2 Calcolo matriciale

Vale la pena di ricordare che nelle operazioni di moltiplicazione e divisione tra matrici valgono comunque le comuni regole algebriche per le quali:

- due o più matrici possono essere tra loro sommate o sottratte se hanno le stesse dimensioni;
- due matrici possono essere moltiplicate o divise se il numero di colonne della prima matrice è pari quello di righe della seconda.

Trasposizione di una matrice

L'operatore di trasposizione è dato dal carattere (').

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

A =

     1     2
     3     4

>> A'

ans =

     1     3
     2     4
```

Per una matrice ad elementi complessi, l'operatore di trasposizione crea la matrice *coniugata trasposta*. Pertanto, nel caso in cui serva la sola matrice coniugata è opportuna premettere all'operatore di trasposizione il carattere (**.**).

Moltiplicazione tra matrici

Con l'operatore ***** si chiede a MATLAB il prodotto scalare di due matrici *m x p* e *p x n*, da cui risulta una nuova matrice *m x n* in cui ciascun elemento è ottenuto come somma del prodotto scalare riga-colonna delle due matrici.
La stessa regola può essere ristretta al caso di moltiplicazione tra vettore e matrice.

Divisione tra matrici

L'operazione di divisione tra matrici porta con sé delle implicazioni di tipo algebrico nel trattamento delle singole matrici. Infatti, le operazioni:

- **divisione a sinistra**
  `X=A\B` ovvero $X=A^{-1}B$

- **divisione a destra**
  `X=B/A` ovvero $X=BA^{-1}$

sono ammissibili se la matrice A è quadrata ed invertibile (cioè a determinante non nullo).
In particolare, l'uso della prima operazione può essere riferito alla risoluzione di un sistema di equazioni algebriche lineari, per il quale la soluzione esiste ed è unica se il numero delle equazioni è pari a quello delle incognite e se il determinante della matrice dei coefficienti ha valore non nullo.
In verità, con MATLAB è possibile la risoluzione guidata di sistemi sia indeterminati che sovradeterminati, per la quale si rimanda più avanti nella trattazione.
Comunque, per la risoluzione di sistemi determinati, è conveniente adottare la divisione a sinistra, da preferire anche all'omologa operazione **`X=inv(A)*B`**, più pesante in termini di complessità computazionale.

Elevamento a potenza di matrici

`A^b`

dove **`A`** deve essere una matrice quadrata e **`b`** è uno scalare

Altre funzioni esponenziali

- **`expm(A)->`**e^A
- **`logm(A)->`**$\log(A)$
- **`sqrtm(A)->`**$\sqrt{A}$ (per matrici definite positive)

Funzioni matriciali avanzate

MATLAB mette a disposizione dei comandi che permettono di elaborare le matrici con le particolari funzioni elencate in tabella.

Funzione	Descrizione
`det(A)`	Calcola il determinante
`diag(A)`	Restituisce gli elementi sulla diagonale principale
`eig(A)`	Calcola gli autovalori
`poly(A)`	Calcola i coefficienti del polinomio caratteristico
`rank(A)`	Calcola il rango
`trace(A)`	Restituisce la traccia di A

Matrici notevoli

Per conoscere tutte le matrici notevoli preimpostate di cui MATLAB dispone, digitare al prompt:

```
>>help specmat
```

Le matrici notevoli di più largo uso sono le seguenti:

Matrice	Descrizione
`diag(m,n)`	Matrice diagonale
`eye(m,n)`	Matrice identità
`ones(m,n)`	Matrice di uno
`ran(m,n)`	Matrice pseudo-casuale a distr. uniforme
`randn(m,n)`	Matrice pseudo-casuale a distr. normale
`zeros(m,n)`	Matrice di zeri

Nel caso di matrici identità quadrate è sufficiente mettere per argomento della funzione la sola dimensione `n`.

4.3 Calcolo polinomiale

Un polinomio, rappresentato nella sua scrittura algebrica, ad esempio, come:

$$p(x) = 5x^4 + 2x^2 + x + 4$$

ha la sua corrispondente definizione in MATLAB come un vettore i cui elementi sono i coefficienti del polinomio ordinati in maniera decrescente.
L'array dei coefficienti del polinomio di cui sopra è:

```
p=[5,0,2,1,4]
```

Si noti che, essendo il polinomio mancante del termine di terzo grado, è stato messo uno zero in corrispondenza dell'elemento del vettore che rappresenta proprio il coefficiente di terzo grado.
Ciò è utile, in particolare, nelle operazioni di addizione e sottrazione di polinomi, i quali per poter essere sommati devono essere dello stesso grado o, almeno, devono essere rappresentati mettendo uno zero per indicare, all'interno del vettore dei coefficienti, i termini di grado mancante.
Tale accortezza non deve essere usata per le operazioni di moltiplicazione e divisione.
Molto brevemente, ora compendiamo i principali comandi da utilizzare con i polinomi, lasciando a capitoli successivi la trattazione delle operazioni di interpolazione che si possono effettuare con i polinomi.

Comando	Descrizione
`conv(a,b)`	Esegue il prodotto tra i polinomi a e b, creando un nuovo vettore i cui elementi sono i coefficienti del polinomio prodotto ordinati in modo decrescente.
`[q,r]=deconv(p1,p2)`	Effettua la divisione tra due polinomi, restituendo gli array quoziente p1 e resto p2.
`poly(p)`	Determina i coefficienti di un polinomio a partire dalle sue soluzioni immesse col vettore p.
`polyval(p,x)`	Valuta il valore assunto da un polinomio in corrispondenza di x
`roots(p)`	Calcola le radici del polinomio p.
`[r,p,k]=residue(n,d)`	Esegue lo sviluppo in fratti semplici.
`q=polyder(p)`	Effettua la derivata del polinomio p.
`q=polyder(n,d)`	Effettua la derivata del prodotto dei polinomi n e d.
`q=polider(n,d)`	Effettua la derivata del rapporto n(x)/d(x)

Esempio sviluppo in fratti semplici

Data la funzione razionale fratta:

$$\frac{n(s)}{d(s)} = \frac{2s^4 + 3s^3 + 15s^2 + 18s + 1}{22s^2 + 2s + 1}$$

Si ottiene lo sviluppo in fratti semplici della stessa, creando separatamente gli array dei coefficienti del numeratore n(s) e del denominatore d(s) e digita successivamente al prompt l'istruzione:

```
>> [r,p,k]=residue(n,d)
```

MATLAB risponde con:

```
r =

   0.7938 - 1.7486i
   0.7938 + 1.7486i
  -1.6631
   0.0755


p =

  -0.1154 + 2.6556i
  -0.1154 - 2.6556i
  -1.1456
  -0.1235


k =

     []
```

Siccome compaiono 4 radici complesse coniugate a due a due, è opportuno selezionarle per effettuare un'espansione separata. Pertanto:

```
>> [n1,d1]=residue(n(1:2),d(1:2),[])

n1 =

    24   -70


d1 =

     1    -5     6
```

Così, riprendendo le radici reali precedentemente trovate, si ha uno sviluppo in fratti semplici con la scrittura più "consona":

$$-\frac{1.66}{s+1.15} + \frac{0.08}{(s+0.12)^2} + \frac{24s-70}{s^2-5s+6}$$

Capitolo 5

Diagrammi

5.1 Le potenzialità grafiche di MATLAB

MATLAB, oltre agli strumenti di calcolo numerico e algebrico già visti, dispone di diversi comandi per la creazione di vari tipi di diagrammi a cui poter assegnare diverse proprietà.
La creazione di un grafico più essere divisa negli step fondamentali riportati nel prosieguo del capitolo come intestazione dei paragrafi.

5.2 Scelta del tipo di grafico

Per avere una panoramica dei tipi di grafico che si possono creare con MATLAB, è utile osservare la tabella sottostante, che guida velocemente nella scelta del comando da utilizzare per la creazione di un diagramma cartesiano specifico. Più avanti, la descrizione estesa, tramite esempi, delle funzioni associate.

	Origine dati (Scala lineare)	
	2-D	**3-D**
Singolo	• **grafico semplice** plot(x,y) • **grafico "accurato"** fplot('funzione',[xmin xmax])	• **a linea (parametrico)** plot3(x,y,z) • **a superficie** mesh(x,y,z); surf(x,y,z) • **a contorno** contour(x,y,z)
Sovrapposti	• **uguale scala asse x** plot(x,A) • **scala asse x differente** plot(X,Y)	
Affiancati	subplot(m,n,p)	subplot(m,n,p)

	Origine dati (Scala non lineare)	
	Scala semilogaritmica	**Scala logaritmica**
Singolo	• **asse x logaritmico** semilogx(y,x) • **asse y logaritmico** semilogy(y,x)	loglog(y,x)
Affiancati	subplot(m,n,p)	subplot(m,n,p)

- **`plot(x,y)`**

Il comando "base" per il tracciamento dei diagrammi, prevede una prima serie di istruzioni per la creazione dei vettori aventi come elementi i valori delle variabili indipendenti `x` e delle variabili dipendenti y.
In questa modalità, per avere una rappresentazione grafica valida della curva, è consigliabile scegliere, nella definizione delle ordinate, un passo abbastanza piccolo. Pertanto, volendo rappresentare la funzione sin(x)*cos(x), si digiti:

```
>> x=[0:0.1:pi];
>> y=sin(x).*cos(x);
>> plot(x,y)
```

Il risultato ottenuto è rappresentato nella figura sottostante.

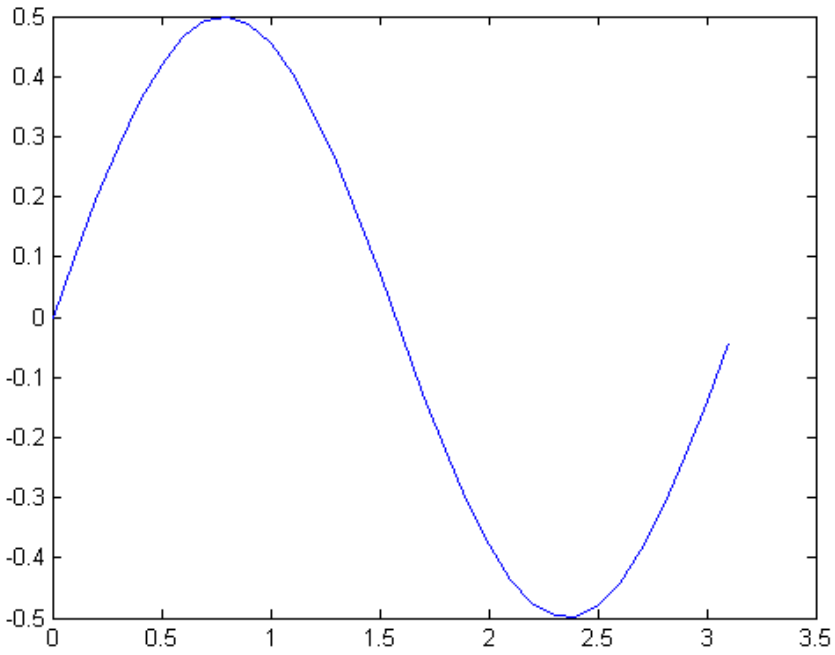


Figura 5.1 Plot funzione

- **`fplot('funzione',[xmin xmax])`**

Utilizzando non serve definire il passo per la rappresentazione del vettore delle ordinate, visto che questo è determinato in automatico in modo da ottenere una rappresentazione accurata della curva.
Ad esempio, per rappresentare la funzione tan(sen(x)), la sintassi è la seguente:

```
>> f='tan(sin(x))';
>> fplot(f,[0 pi])
```

Inoltre, si può stabilire un opportuno intervallo di rappresentazione anche lungo y, tramite la scrittura:

```
>> fplot(f,[0 pi 0 2])
```

Così, si ottiene come risultato che il grafico in figura 5.3 ha il valore 2 come limite massimo delle ordinate.

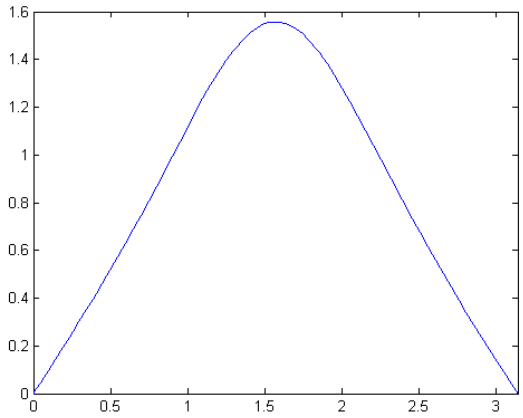


Figura 5.2 Plot con ordinata massima 1,6

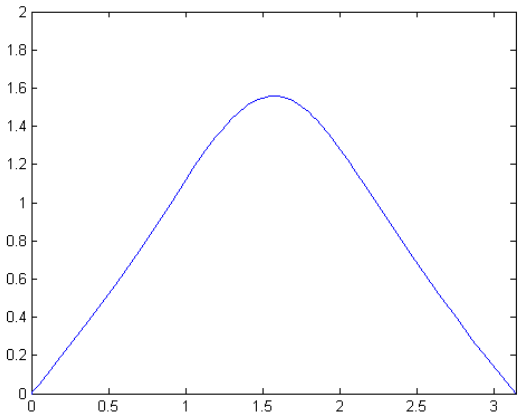


Figura 5.3 Plot con ordinata massima 2

- **`Diagrammi sovrapposti`**

Dovendo rappresentare dati derivanti da due o più funzioni i cui comportamenti sono confrontabili, si può scegliere, in prima istanza, di memorizzare le ordinate da rappresentare in una matrice A delle variabili dipendenti, definendo dapprima il vettore delle ascisse:

```
>> x=[0:0.1:pi];
```

Successivamente, si predispone un'area di memoria di dimensioni pari alla matrice A:

```
>> A=zeros(32,2);
>> A(:,1)=sin(x');
>> A(:,2)=cos(x');
>> plot(x,A)
```

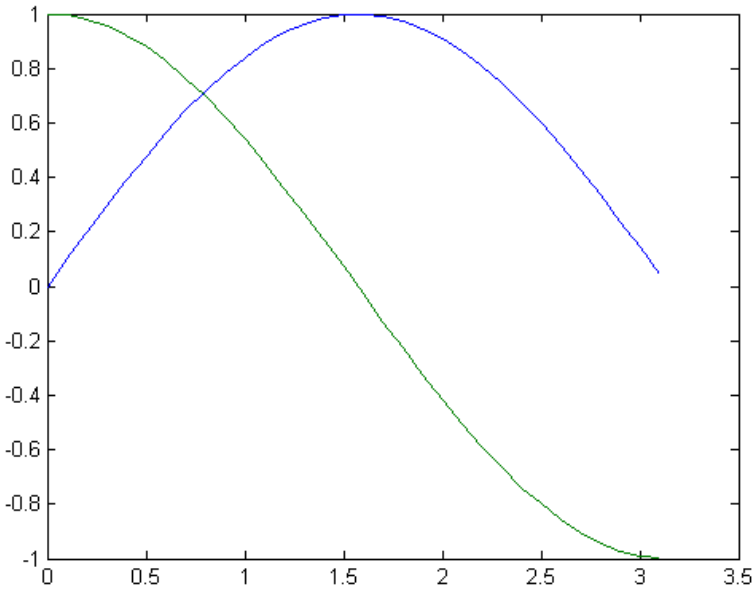


Figura 5.4 Esempio di diagramma multiplo

Esistono, però, altre possibilità, che evitano di passare attraverso la definizione della matrice di cui sopra:

```
>> x=[0:0.1:pi];
>> y1=sin(x);
>> y2=cos(x);
>> plot(x,y1)
>> axis(axis)
>> hold
>> plot(x,y2)
```

```
>> x=[0:0.1:pi];
>> y1=sin(x);
>> y2=cos(x);
>> plot(x,y1,x,y2)
```

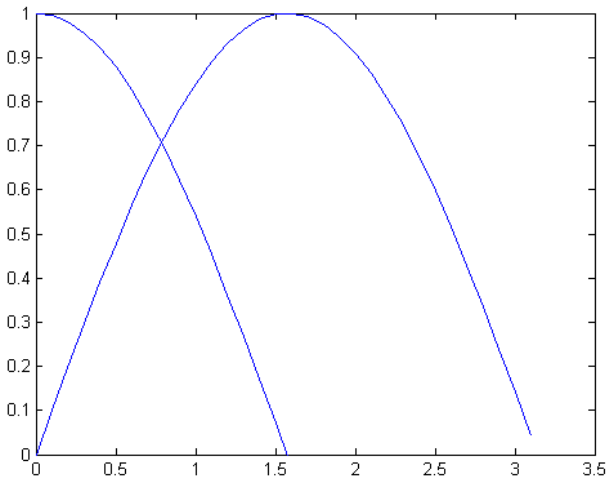


Figura 5.5 Esempio di diagramma multiplo

Quando, però, i dati di più funzioni sono poco confrontabili in termini di ordine di grandezza e, conseguentemente, le ordinate molto grandi di una delle due funzioni rendono poco distinguibile gli andamenti delle restanti funzioni, è opportuno utilizzare il seguente script:

```
>> t=[0:0.1:10];
>> plot(t,sin(t));
>> axis(axis)
>> hold
>> plot(t,[t;exp(t)])
```

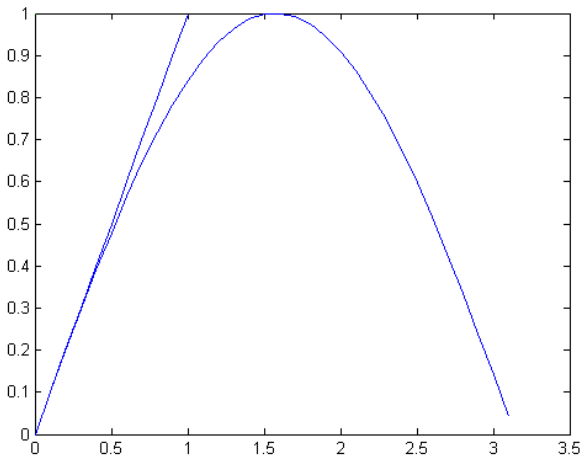


Figura 5.6 Rappresentazione di dati poco confrontabili

Nel grafico risultante di figura 5.6 è stato così possibile "lasciare in primo piano" la prima funzione **`sin(t)`**, la quale è stata resa confrontabile nel suo intervallo di definizione con l'esponenziale.

- **`Diagrammi affiancati`**

Il comando **`subplot(m,n,p)`** crea una rappresentazione matriciale *mxn* di diagrammi (siano essi piani che tridimensionali), iniziando a disegnare dall'angolo in alto a sinistra e finendo in basso a destra.
Ad esempio, volendo rappresentare le funzioni y=sin(x) e y=cos(x) affiancate, si ha:

```
>> x=[0:0.1:pi];
>> y1=sin(x);
>> subplot(1,2,1)
>> plot(x,y1)
>> y2=cos(x);
>> subplot(1,2,2)
>> plot(x,y2)
```

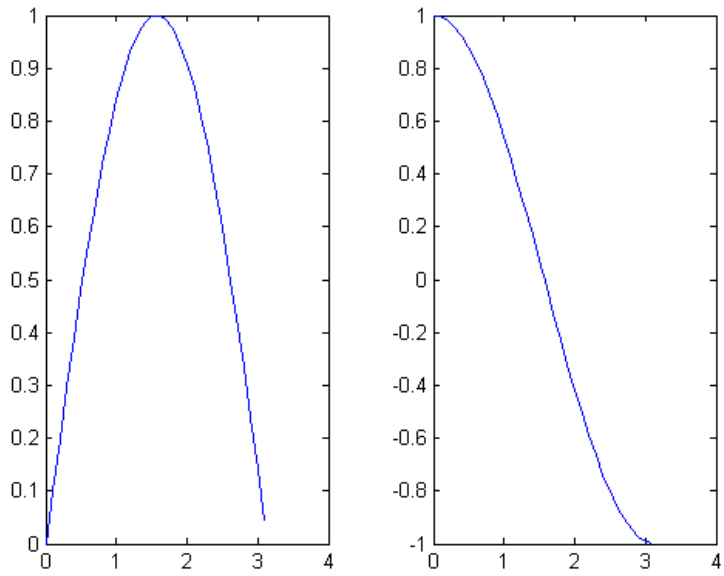


Figura 5.7 Diagrammi affiancati

Per semplicità, le due funzioni sono state rappresentate con un semplice **`plot(x,y)`**, ma è appena il caso di precisare che la modalità di rappresentazione affiancata supporta tutti gli altri comandi grafici.

Diagrammi tridimensionali

- **`plot3(x,y,z)`**

La seguente scrittura rappresenta l'equazione parametrica di un'elica cilindrica:

x=sen(t)
y=cos(t)
z=t

La quale ha la sua corrispondente in MATLAB:

```
>> t=[0:0.1:5*pi];
>> plot3(sin(t),cos(t),t)
>> plot3(sin(t),cos(t),t)
```

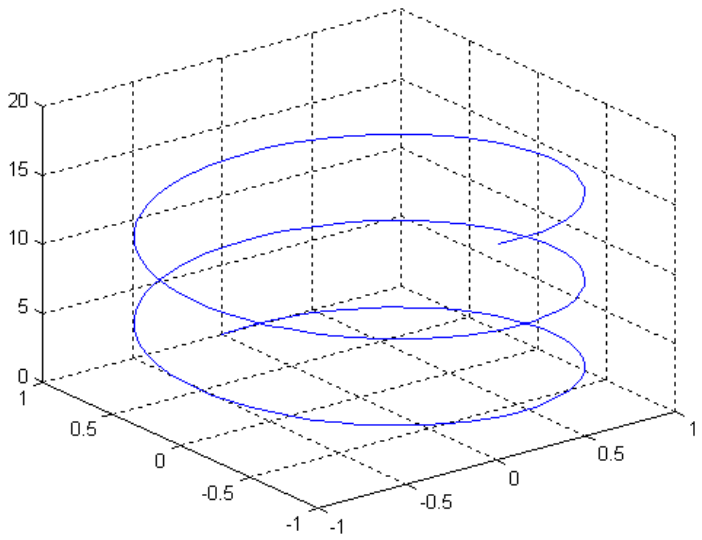


Figura 5.8 Rappresentazione elica cilindrica

- **`mesh(X,Y,Z)`**

Con questo comando, una funzione viene rappresentata secondo un reticolato tridimensionale. Perciò, si deve prima creare una griglia di punti che rappresentano le variabili indipendenti appartenenti al dominio della funzione da rappresentare.

Ad esempio, con la scrittura:

```
x=[0:0.1*pi:2*pi];
y=[1:0.1*pi:2*pi];
[X,Y]=meshgrid(x,y);
```

si è definito un dominio di x appartenenti all'intervallo tra 0 e 2*pi e di y comprese tra 1 e 2*pi.

Invece, volendo creare un dominio quadrato, si ha:

```
[X,Y]=meshgrid(-2*pi:.1*pi:2*pi);
```

Per ottenere la rappresentazione della funzione sottoindicata, si digiti:

```
Z=sin(X+Y)-cos(X-Y);
mesh(X,Y,Z)
```

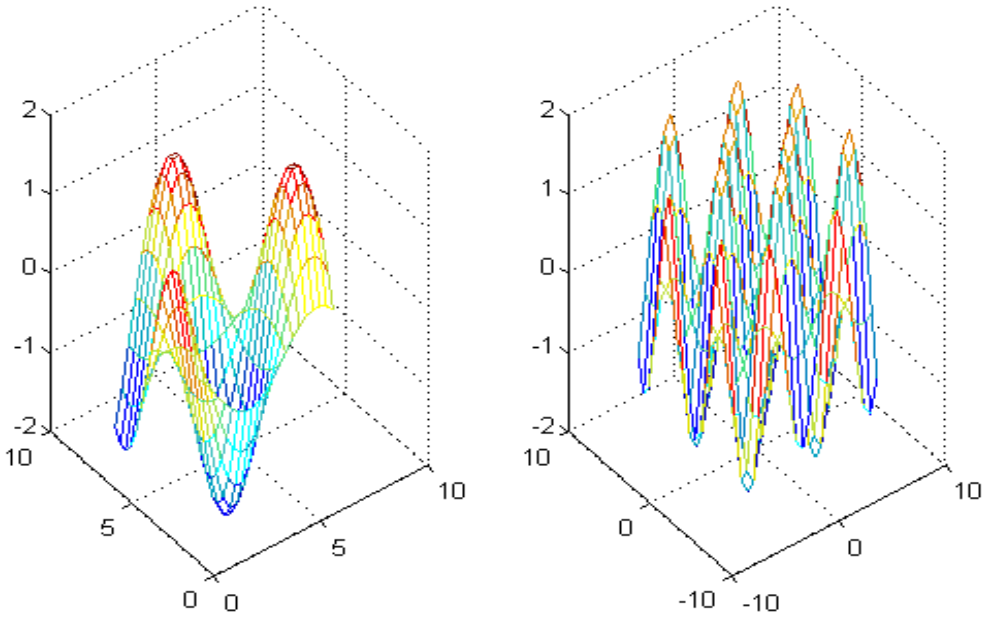
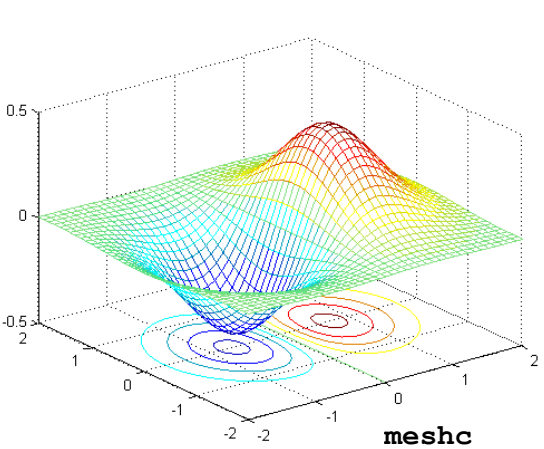


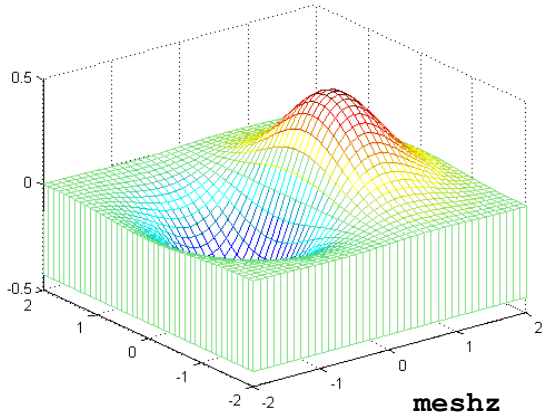
Figura 5.9 Rappresentazione della funzione negli intervalli di definizione prescelti

Per aggiungere al di sotto della curva le linee di livello, in modo da avere una cognizione più accurata dei massimi e minimi, si usi **meshc**.
Inoltre, esistono i comandi **meshz** (che crea al disotto della superficie una base di linee verticali congiungenti il piano cartesiano con la superficie) e **waterfall** (rappresentazione del grafico utilizzando una griglia unidirezionale).

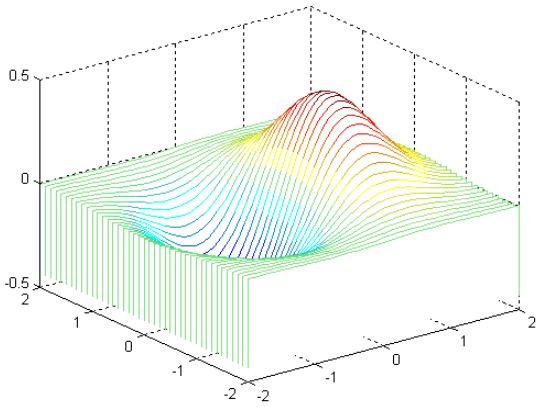
```
>> [X,Y]=meshgrid(-2:0.1:2);
>> Z=X.*exp(-(X.^2+Y.^2));
>> meshc(X,Y,Z)
>> meshz(X,Y,Z)
>> waterfall(X,Y,Z)
```



meshc



meshz



waterfall

- **surf(X,Y,Z)**

La stessa funzione rappresentata prima con **mesh** è ora disegnata a facce piane. Come nel caso precedente, **surfc** crea le curve di livello al di sotto della superficie.

```
>> [X,Y]=meshgrid(-2:0.1:2);
>> Z=X.*exp(-(X.^2+Y.^2));
>> surf(X,Y,Z)
```

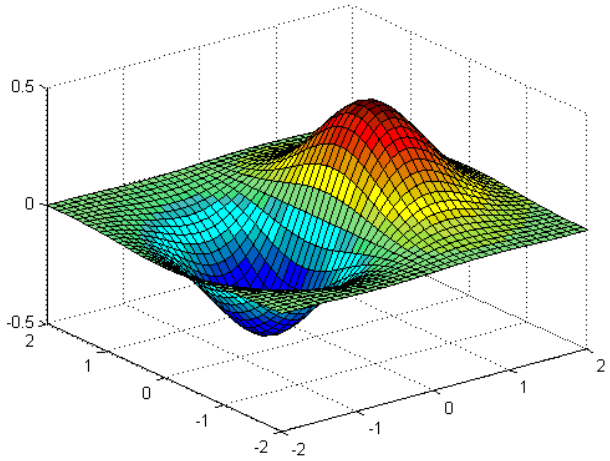
- **contour(X,Y,Z)**

Crea un diagramma a curve di livello in 3-D, tramite la sintassi:

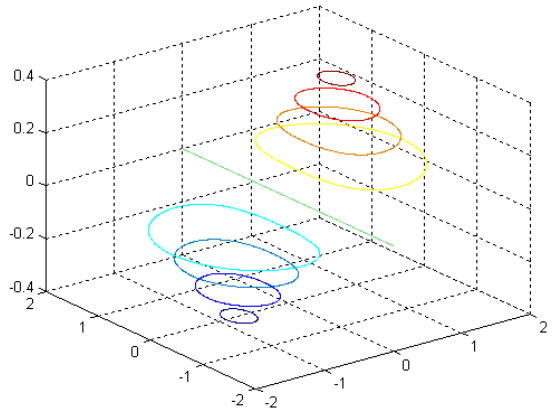
```
>> contour3(X,Y,Z)
```

O, in alternativa, permette di definire la quota del piano di sezione su cui visualizzare le curve di livello, tramite la scrittura:

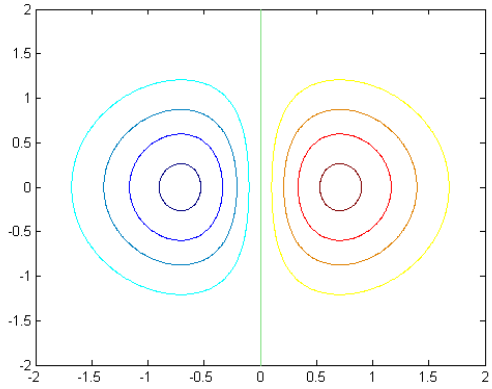
```
>> contour(X,Y,Z,0.1)
```



surf



contour3



contour

5.3 Opzioni del grafico

Titoli

Esiste una serie di comandi che permette di aggiungere al grafico ulteriori informazioni. Tutti questi nuovi elementi sono sottoriportati in tabella.

Comando	Descrizione
`title('titolo')`	Permette di aggiungere il titolo ad un grafico.
`xlabel('x')`	Consente di denominare l'asse delle ascisse.
`ylabel('y')`	Consente di denominare l'asse delle ordinate.
`text(x,y,'testo')`	Permette di aggiungere un commento al grafico alla coordinata indicata.
`gtext('testo')`	Posiziona la stringa testo nel punto indicato con il mouse.
`legend('dati1',…,'datin')`	Consente di associare un nome ad ogni serie di dati in un grafico multiplo.

Per applicare i comandi fin qui descritti, supponiamo di dover rappresentare i dati sperimentali ottenuti da un ipotetico esperimento (ipotizziamo, per semplicità, che si tratti della caduta di un grave da un palazzo di tre piani).

```
>> a=9.81
>> t=0:0.1:1.35;
>> x=1/2*a*(t.^2);
>> v=a*t;
>> plot(t,x,t,v),title('Caduta di un grave'),xlabel('tempo'),...
ylabel('dati cinematici rilevati'),...
gtext('Impatto al suolo'),legend('spazio percorso [m]','velocità [m/s]')
```

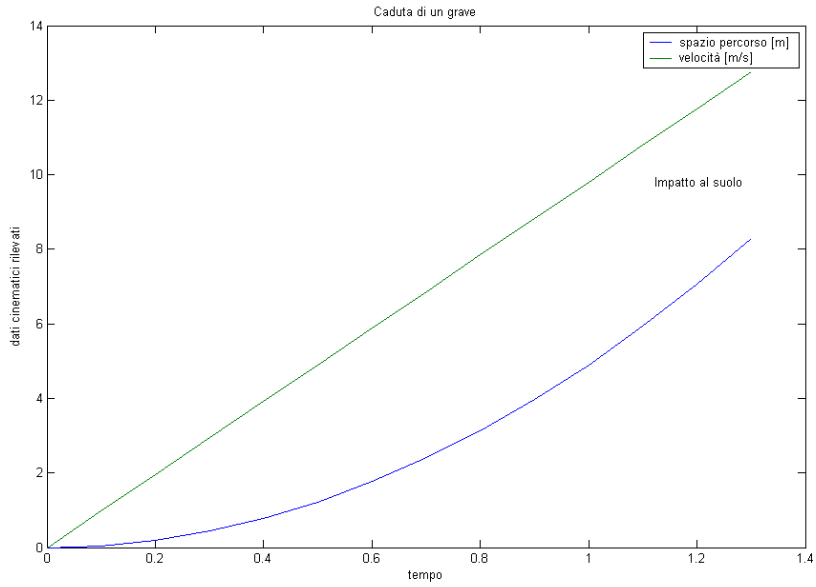


Figura 5.10 Andamento delle variabili cinematiche

Assi coordinati

Le opzioni disponibili per la modifica delle proprietà degli assi cartesiani sono le seguenti:

- **Attivazione degli assi e della griglia**
  - **Axis ('on') ('off')**
  - **Grid ('on) ('off')**
- **Modifica fattore di scala e dimensioni degli assi**
  - **axis ('equal')** uguale fattore di scala per i due assi
  - **axis ('square')** delimita un'area grafica quadrata
  - **axis ('square')** estende l'area grafica fino ai limiti della curva rappresentata
  - **axis ('auto')** scalatura automatica degli assi

- **Modifica incremento dei segni di graduazione**
    - **`set(gca,'Xtick',[xmin:dx:xmax],'Ytick',[ymin:dy:ymax])`**

Con questa sintassi è possibile definire l'intervallo di rappresentazione lungo gli assi (xmin, xmax, ymin, ymax) e la loro spaziatura con dy.

Formato grafico dei dati

Un diagramma è ancora più significativo quando è arricchito di ulteriori caratterizzazioni grafiche. Gli attributi da poter assegnare ai tipi di linea e ai singoli dati sono i seguenti:

Stile linea		Stile punto		Colore	
CONTINUA	-	Punto	.	Giallo	y
	--	Più	+	Magenta	m
Tratteggiata	:	Asterisco	*	Ciano	c
Punteggiata	-.	Cerchio	o	Rosso	r
A tratto-punto		Croce	x	Verde	g
		Quadrato	s	Blu	b
		Rombo	d	Bianco	w
		Triangoli	v	Nero	k
		Stelle	p		

Ad esempio, con le istruzioni sottoriportate è possibile effettuare una rappresentazione di dati sparsi, evidenziando ogni dato con un marcatore circolare (**`'Marker','o'`**) di colore ciano (**`'MarkerEdgeColor','c'`**)

```
>> x=[0:0.1:pi];
y=sin(x).*cos(x);
>> plot(x,y,'Marker','o','MarkerSize',2,'MarkerEdgeColor','c')
```

dove **`MarkerSize`** identifica la taglia del marcatore.

In alternativa, è possibile adottare una scrittura semplificata che rappresenta i singoli dati senza congiungerli tra loro.

```
>> plot(x,y,'oc')
```

Le due scritture conducono ai risultati in figg 5.11 e 5.12.

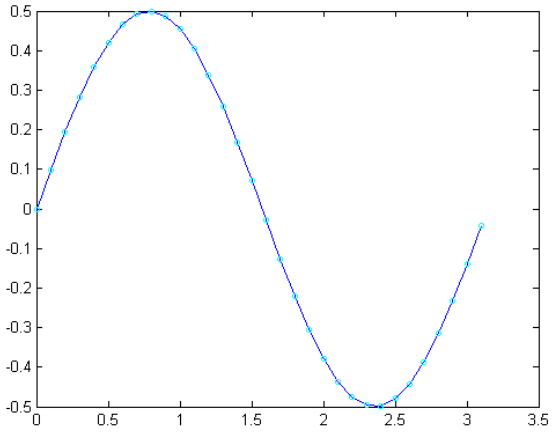


Figura 5.11 Uso di marcatori per evidenziare i dati

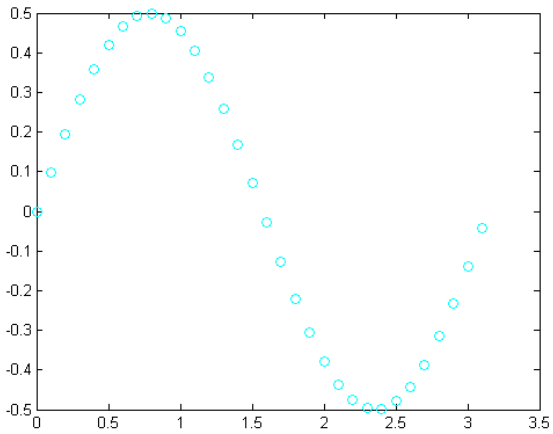


Figura 5.12 Rappresentazione di dati sparsi

Inoltre, è possibile contemporaneamente identificare i dati singoli con un marcatore ed evidenziare l'andamento della curva con un certo stile di linea e colore con la sintassi:

```
>> plot(x,y,'m:',x,y,'sg')
```

Il medesimo risultato può essere ottenuto utilizzando la scrittura estesa per la modifica degli attributi degli oggetti grafici:

```
plot(x,y,'Marker','s','MarkerSize',4,'MarkerEdgeColor','g','LineStyle',':
','Color','m')
```

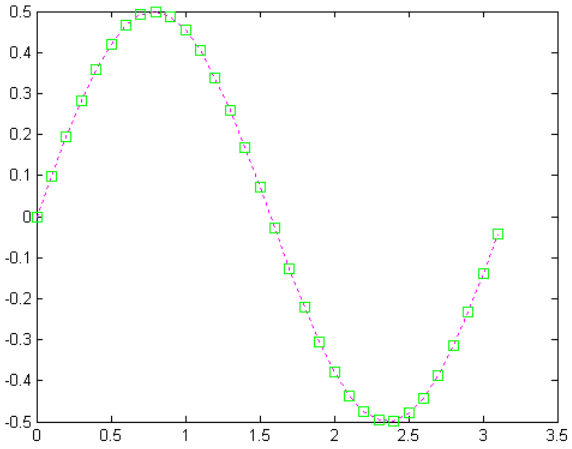


Figura 5.13 Modifica degli attributi del grafico

Stampa del grafico

L'iter più veloce da utilizzare per stampare un grafico è quello che consiste nell'utilizzare la voce **`Print`** del menu **`File`** della finestra grafica.
Per esportare il grafico in formato file di stampa o immagine, nella stessa finestra, si selezioni la voce **`Export...`** del medesimo menu.

Capitolo 6

Programmazione strutturata

6.1 MATLAB come linguaggio di programmazione

Nella trattazione precedente, MATLAB è stato utilizzato come una "calcolatrice grafica", utilizzando semplici istruzioni per compiere operazioni numeriche o rappresentazioni grafiche.
In realtà, con MATLAB è possibile programmare e, pertanto, operazioni ripetitive di calcolo possono essere implementate in un programma secondo le tre strutture fondamentali dei linguaggi di programmazione:

- *sequenza*
- *selezione*
- *iterazione*

Inoltre, in questo capitolo si vedrà come le stesse istruzioni, che finora sono state viste vivere limitatamente all'interno della sessione di lavoro corrente, possono essere memorizzate a permanenza su file.

6.2 File

Le istruzioni fin qui viste sono state editate all'interno della **`Command Window`**.
In realtà, se si prevede di riutilizzare le stesse istruzioni, è opportuno creare un M-file in cui si memorizzano le istruzioni digitate da tastiera per poi richiamarle in qualsiasi momento, unitamente alla dichiarazione delle variabili usate.
Per creare un M-file (script file) basta cliccare sull'icona (New M-File) rappresentante una pagina bianca presente a sinistra sulla barra dei menu.
E' da notare che tutte le variabili memorizzate su questo "foglio bianco", all'atto dell'esecuzione dello script file (clic sull'icona Run), divengono visibili nel workspace come variabili globali.
Pertanto, è bene far presente che in alcuni casi si deve rendere conto con la possibilità di omonimia tra variabili.
Per salvare un m-file è sufficiente andare alla voce **`Save`** del menu **`File`**.

6.3 Funzioni

Una sequenza di comandi usata per ottenere un determinato risultato è identificata tramite una function, secondo la sintassi:

```
function var_output = nome_function (var_input)
   blocco_istruzioni;
```

La stessa function può essere salvata all'interno di uno script file, denominato con lo stesso nome della function.
E' da notare, però, che esiste una sostanziale differenza tra script e function nell'ambito della visibilità delle variabili. Infatti, i parametri passati alle function hanno **visibilità locale**, cioè esistono solo all'interno della funzione stessa.
Quindi, se all'atto dell'esecuzione della function, una variabile ausiliaria (utilizzata solo nei calcoli della function) ha lo stesso nome di una variabile preesistente nel workspace, quest'ultima, una volta eseguita la function, non viene modificata nel suo valore.
Per forzare MATLAB a trattare le variabili in uso in una function come globali, si deve effettuare una dichiarazione delle variabili secondo la scrittura:

```
global nome_var
```

Inoltre, quando non si conosce a priori il numero di argomenti in ingresso a una variabile si utilizzi la definizione della funzione:

```
function var_output = nome_function (varargin)
```

Parimenti, per le variabili di output,si usi **`varargout`.**

Nella scrittura di funzioni e script può essere utile inserire commenti. Ciò è possibile usando la sintassi:

```
% questo è un commento
```

Inoltre, per continuare una stessa istruzione su un'altra riga, si utilizzino i tre punti "**`...`**".

6.4 Streaming delle variabili

Con *streaming delle variabili* si intende l'insieme delle operazioni di input di variabili e loro output unitamente alla modifica del formato di visualizzazione.

Input

```
nome_var = input ('testo');
```

Con la sintassi di cui sopra, MATLAB, visualizzando la stringa **`'testo'`** chiede all'utente di immettere da tastiera il valore della variabile **`nome_var`**.

Qualora si voglia memorizzare una variabile come stringa si utilizzi la scrittura:

```
nome_var = input ('testo','s');
```

Output

Per visualizzare il risultato di una operazione con un determinato formato si utilizzi la sintassi:

```
fprintf('-largh_c.cifre_dec cod \c',nome_var)
```

La sintassi si compone di più elementi aventi le seguenti funzioni rispettivamente:

- **`[-]`**
  allineamento numerico a sinistra

- **`largh_c`**
  numero di cifre da destinare per la rappresentazione del numero.

- **`cifre_dec`**
  numero di cifre decimali dopo la virgola

- **`cod`**
  notazione per la rappresentazione del numero
  - **%f:** formato decimale
  - **%e:** notazione scientifica con e minuscola
  - **%E:** notazione scientifica con E maiuscola
  - **%g:** notazione decimale in cui gli zeri dopo la virgola non vengono visualizzati

- **`\c`**
  - **`\n`:** avvia una nuova riga
  - **`\b`:** backspace
  - **`\t`:** tab

Esempio

```
>> a=pi

a =

    3.1416

>> fprintf('Pi greco a due cifre decimali: %3.2f\n',a)
Pi greco a due cifre decimali: 3.14
>> fprintf('Pi greco a cinque cifre decimali: %6.5f\n',a)
Pi greco a cinque cifre decimali: 3.14159
```

Formato delle variabili

I principali formati numerici adottati in MATLAB sono i seguenti:

- **`format short:`** 4 cifre decimali
- **`format long:`** 16 cifre decimali
- **`format short e:`** notazione esponenziale a 4 cifre decimali
- **`format long e:`** notazione decimale
- **`format rat:`** approssimazione razionale

Per gli altri formati numerici si digiti al prompt **`>> help format`**

6.5 Operatori relazionali e logici

Operatore	Descrizione
<	minore
<=	minore uguale
>=	maggiore uguale
==	uguale
~=	diverso

La scrittura **`var=a<b`** definisce una variabile booleana **`var`** che restituisce 1 se la relazione è vera, altrimenti dà 0.

Gli **operatori logici** si distinguono in due classi.
Alla prima appartengono tutti quegli operatori (detti binari) che determinano operazioni logiche tra **due** variabili scalari o tra gli elementi di **due** array (primo tipo di operatori tabellati); alla seconda attengono le operazioni logiche effettuate su **una** singola variabile.

Operatore binario	Descrizione
&	**AND:** restituisce 1 se entrambi gli operandi sono diversi da 0.
!	**OR:** restituisce vero se almeno uno degli operandi è diverso da 0.
XOR	Ritorna falso se entrambi i valori degli operandi sono uguali o diversi da 0.

Operatore monario	Descrizione
NOT(a)	Restituisce vero (1) se il valore di **a** è uguale a 0; falso (0) altrimenti.
ANY(A)	Restituisce un vettore nx1 di 1 se tutti gli elementi dell'array mxn sono diversi da 0.
ALL(A)	Restituisce un vettore nx1 di 1 se almeno un elemento in ciascuna colonna dell'array è diverso da 0.

6.6 Istruzioni di selezione

```
if (espressione logica)
      istruzioni
elseif (espressione logica)
      istruzioni
else
      istruzioni
end
```

La sintassi sopra riportata rappresenta la struttura più completa per eseguire istruzioni condizionali, nelle quali lo svolgimento di una sequenza di istruzioni è condizionato dal soddisfacimento di una determinata condizione.
Nel caso particolare, se non è soddisfatta la prima condizione, MATLAB procede ad esaminare il secondo blocco di istruzioni se la seconda condizione è verificata, altrimenti compie le istruzioni non condizionate del terzo blocco.
Esistono anche due strutture condizionali più semplici, costruite con gli stessi comandi **if**, **elseif** ed **else** che permettono di condizionare l'esecuzione di alcune istruzioni al soddisfacimento di più di una condizione per volta.

```
if (espressione logica)
      istruzioni
elseif (espressione logica)
      istruzioni
end

if (espressione logica)
      istruzioni
else
      istruzioni
end
```

Inoltre, nel caso in cui esista un ampio ventaglio di condizioni, al cui soddisfacimento è legata l'esecuzione di alcuni blocchi d'istruzioni, è disponibile il costrutto **switch**.

```
switch nome_variabile
case valore_1
     istruzioni
case valore_2
     istruzioni
otherwise
     istruzioni
end
```

6.7 Istruzioni di iterazione

Come in altri linguaggi di programmazione, i cicli iterativi condizionati per l'esecuzione di una sequenza di istruzioni, sono governati dai comandi **while** e **for**.

```
while condizione
     istruzioni
end
for i=1:n
     istruzioni
end
```

Esempio di file script

```
%file-m conv10/2
%script che trasforma in base 2 un numero espresso in base 10
clear all; ris=2;
resto=0;
num=input('Inserisci il numero da convertire: ');
i=0;
%predisposizione di un'area di memoria per contenere l'output che può
%essere costituito da 24 cifre al massimo
num_binario(24)=0;
%"cuore" del programma per il calcolo della conversione
while ris>1
%troncamento del risultato della divisione per 2
    ris=fix(num/2);
%calcolo del resto
    resto=rem(num,2);
    num=ris;
   num_binario(24-i)=resto;
   i=i+1;
end
num_binario(24-i)=1;
%selezione tra gli elementi del vettore v delle sole cifre del numero
%binario
n=1;
while num_binario(n)==0
n=n+1;
end
%cancellazione degli zeri antecedenti le cifre del numero binario
num_binario(1:(n-1))=[]
```

6.8 Debug dei programmi

In generale, per operazione di debug si intende la procedura manuale o, nel caso particolare, automatizzata con la quale si individuano errori di tipo sintattico ed errori di tipo logico.
L'indubbio vantaggio derivante dall'eseguire una sessione di debug, piuttosto che dal riconoscimento di errori sintattici, peraltro segnalati da MATLAB in automatico, è rappresentato sicuramente dalla **possibilità di individuare errori di tipo logico** e cioè derivanti dall'implementazione dell'algoritmo risolutivo del problema, seguendo l'evoluzione delle variabili nel loro cambiamento di valore.
Ciò è possibile mettendo dei punti di interruzione con un clic in corrispondenza del numero di riga.
Appare così un punto rosso a partire dal quale l'esecuzione da parte del compilatore viene interrotta e da cui la stessa può essere avanzata manualmente riga per riga tramite il bottone **`step`** (esiste anche l'omologo **`step-in/step-out`** che serve per penetrare all'interno del codice di una function).
E' da notare che, in questo caso, con il tasto **`Run`** il codice viene eseguito all'interno di spezzoni di codice compresi tra due punti di interruzione.
Per cancellare il punto di interruzione immesso basta cliccare sullo stesso ulteriormente o utilizzare il bottone **`set/clear breakpoint`**.
`Clear all breakpoints` serve invece per cancellare tutti i punti di interruzione precedentemente immessi.
In questa modalità di avanzamento step by step è possibile visualizzare contemporaneamente il valore assunto dalle variabili del programma cliccando due volte su di esse all'interno della finestra workspace.

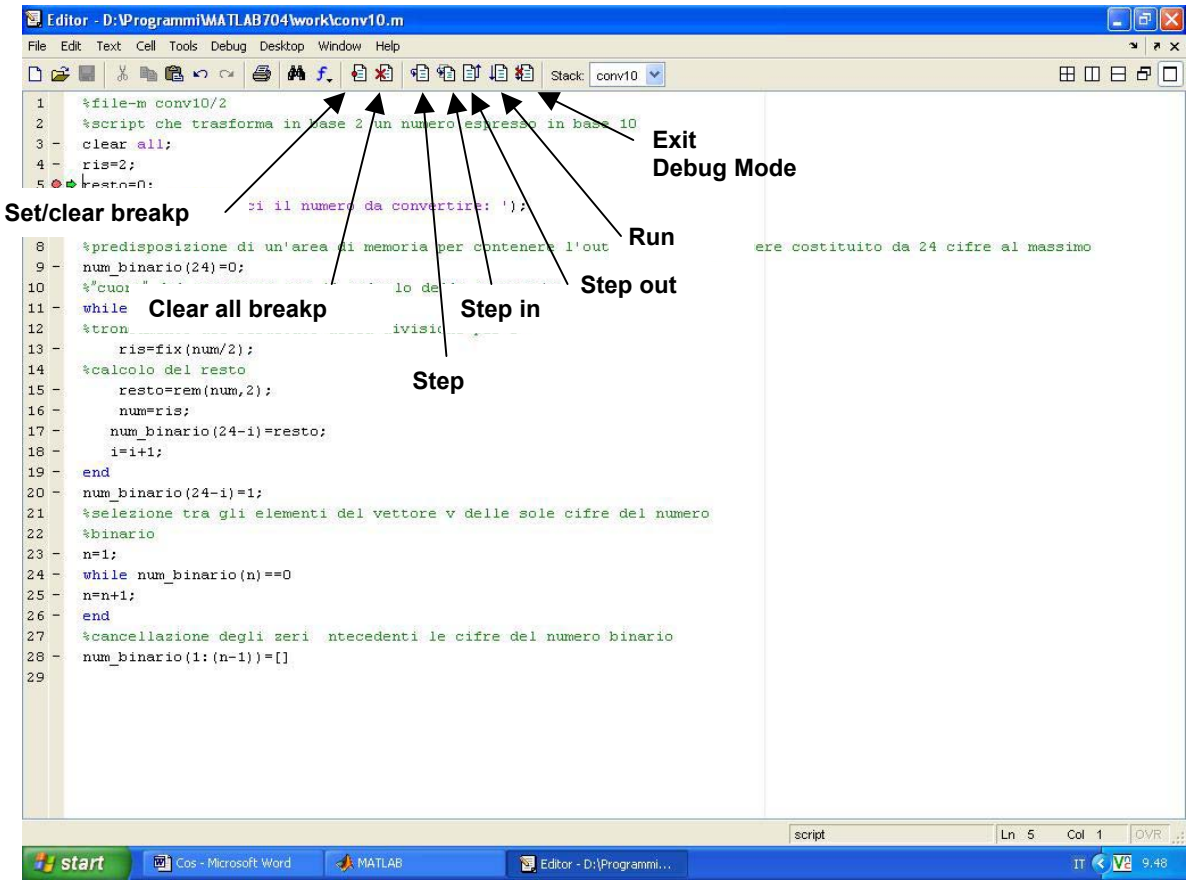


Figura 6.1 Debug Mode

Capitolo 7

Funzioni matematiche avanzate

7.1 Risoluzione di problemi di matematica avanzata

Accanto alle funzioni matematiche elementari, MATLAB possiede diversi comandi utili che conducono alla risoluzione di problemi pertinenti ad applicazioni matematiche più avanzate.
Pertanto, in questo capitolo, verranno presentate le funzioni per operazioni di interpolazione di dati sparsi, gli strumenti per la risoluzione di sistemi di equazioni lineari e per lo studio analitico di funzioni matematiche (ricerca di punti di stazionarietà e di zero, derivazione ed integrazione), nonché per la risoluzione di equazioni differenziali.

7.2 Interpolazione

MATLAB dispone di diversi comandi per operazioni di interpolazione di dati, da effettuarsi sia nel piano che nello spazio.
Di seguito è riportata la spiegazione dei principali comandi, coadiuvata da esempi.

- **griddata**

Restituisce una matrice di z interpolate, calcolate in corrispondenza di determinati punti sul piano, specificati tramite XI (vettore riga delle ascisse) e YI (vettore colonna delle ordinate), nota l'equazione della superficie. Essendo gli elementi di entrambi i vettori equispaziati, è possibile utilizzare il comando **surf** per la rappresentazione della superficie.

```
%creazione di 100 punti random compresi tra -90° e +90°
x = rand(100,1)*3.14-1.57;
y = rand(100,1)*3.14-1.57;
z=cos(x.^2+y.^2)
%creazione dei vettori equispaziati XI e YI
i = -1.57:.1:1.57;
[XI,YI] = meshgrid(i,i);
ZI = griddata(x,y,z,XI,YI);
surf(XI,YI,ZI), hold
plot3(x,y,z,'o'), hold off
```

In alternativa è possibile definire il metodo di interpolazione sostituendo la riga:
`ZI = griddata(x,y,z,XI,YI);`
con:
`ZI = griddata(x,y,z,XI,YI,'cubic');`
in cui **cubic** afferisce alla scelta del metodo di interpolazione lineare usato tra gli altri possibili, di cui si parlerà più estesamente nel paragrafo successivo.
Il frammento di codice sopra riportato genera il risultato rappresentato nella successiva figura 7.1.

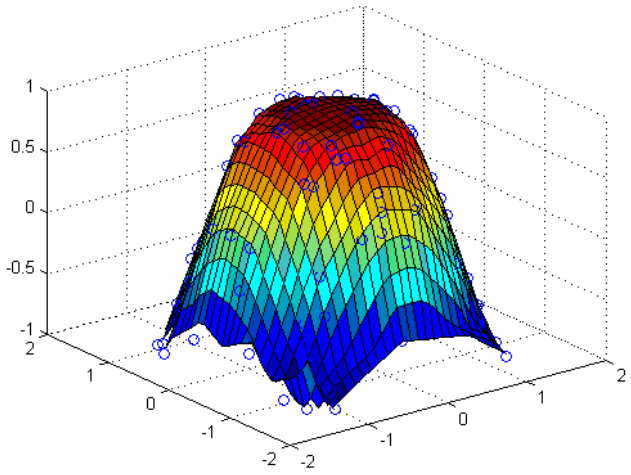


Figura 7.1 Risultato di `griddata`

- **`interp1`**

Per ottenere una interpolazione lineare, si ha a disposizione la sintassi:

```
yi=interp1(x,y,xi)
```

che restituisce un vettore **`yi`** i cui elementi sono i valori delle ordinate interpolate ottenute in corrispondenza dell'ascissa **`xi`**, note le ascisse **`x`** e le ordinate **`y`** che concorrono a definire la curva interpolante.
In particolare, usando la scrittura:

```
yi=interp1(x,y,xi,metodo)
```

È possibile scegliere il metodo di interpolazione:

- **`linear`**: interpolazione tramite spline cubiche
- **`spline`**: interpolazione tramite spline cubiche
- **`cubic`**: interpolazione con cubiche di Hermite
- **`pchip`**: interpolazione con cubiche di Hermite
- **`nearest`**: interpolazione al punto più vicino

È da notare però che **`nearest`** e **`linear`**, a differenza degli altri metodi, non consentono operazione di estrapolazione, cioè non permettono di calcolare ordinate al di fuori dell'intervallo di ascisse definito.
In tal caso, si deve specificare esplicitamente l'opzione di estrapolazione, scrivendo:

```
yi=interp1(x,y,xi,metodo,'extrap')
```

Nel caso contrario, se si vogliono "filtrare" punti al di fuori dell'intervallo di ascisse definito si usa la sintassi:

```
yi=interp1(x,y,xi,metodo,extrapval)
```

La scrittura di cui sopra pone a 0 le ordinate interpolate al di fuori dell'intervallo di ascisse definito.
Tutti questi metodo funzionano se il vettore **x** è monotono crescente.
In presenza di dati non equispaziati, è preferibile usare **interp1q** in alternativa a **interp1**, visto che esso conduce a tempi di elaborazione più brevi.
A titolo di esempio, si riporta un grafico che dimostra come l'uso del metodo **spline** porta a migliori risultati nel caso di funzioni monotone.
Quindi per funzioni non strettamente monotone è consigliabile utilizzare il metodo **pchip**.

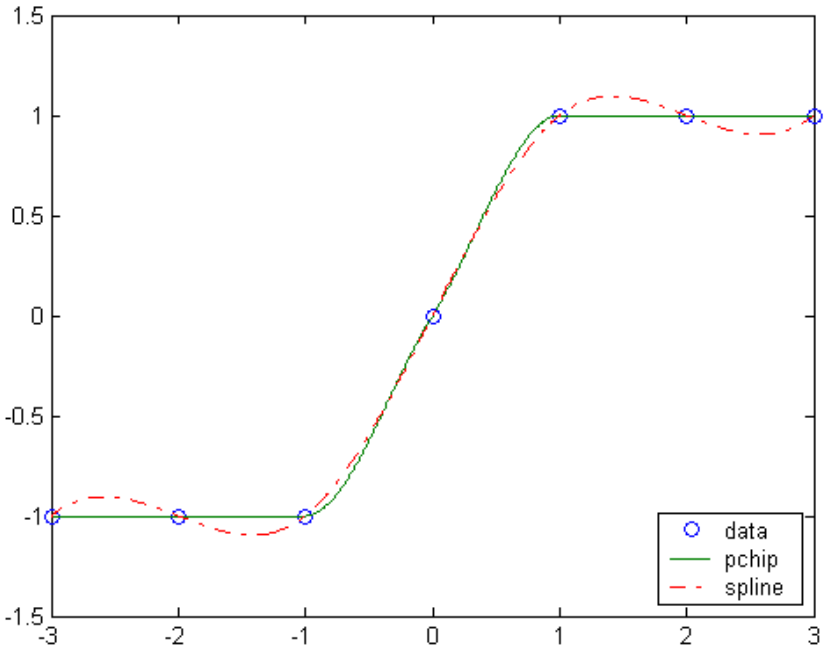


Figura 7.2 Esempio di interpolazione di dati

- **interp2**

zi=interp2(x,y,z,xi,*metodo*)

esegue un interpolazione su tabella bidimensionale. Tra i metodi è possibile scegliere **linear**, **cubic** e **nearest**.

- **polyfit**

I comandi appena visti consentono la determinazione di ordinate interpolate in corrispondenza di determinati valori di ascisse. In alternativa è possibile conoscere una funzione analitica approssimante, ottenuta utilizzando il metodo ai minimi quadrati, dato un vettore di variabili indipendenti x e un vettore y con i valori da approssimare.
Con l'istruzione:

p=polyfit(x,y,n)

Si chiede a MATLAB di trovare un polinomio di grado **n** approssimante i dati.

- **`Basic Fitting Interface`**

MATLAB dispone, per la risoluzione di problemi di interpolazione, di un'interfaccia grafica associata ad ogni **`figure`** che consente, una volta generato il grafico a partire da dati sparsi, di scegliere, tramite una intuitiva interfaccia grafica, la curva che meglio approssima i dati, potendola selezionare tra polinomiali, spline e cubiche.
A questa interfaccia si accede tramite il menu **`Tool`** appartenente alla finestra del grafico e selezionando la voce **`basic fitting`**.
La **`Basic Fitting Interface`** consente di determinare l'equazione dell'interpolante (selezionando **`show equation`**), nonché permette di ottenere la rappresentazione degli scarti tra valore reale e valore atteso selezionando l'opzione **`plot residual`**.
Infine, all'interno della sezione appartenente al terzo pannello, si possono salvare i risultati ottenuti nel workspace cliccando sul bottone corrispondente, o plottarli selezionando il checkbox **`plot results`**.
Per accedere da una sezione all'altra all'interno della finestra, si clicchi sulla freccia in basso a destra.

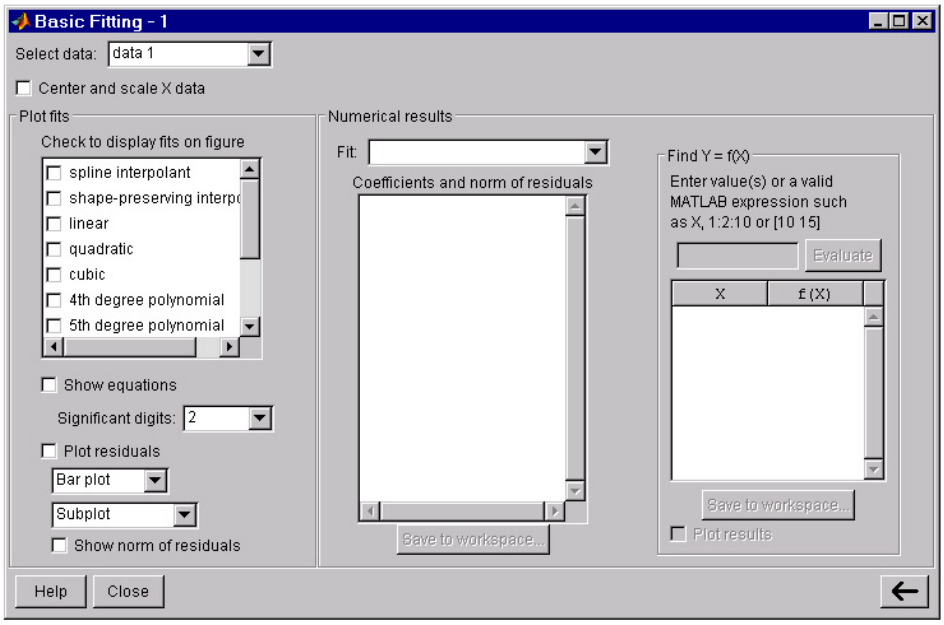


Figura 7.3 Basic Fitting Interface

7.3 Risoluzione di sistemi di equazioni lineari

Spesso capita di dover affrontare la risoluzione di sistemi di equazioni lineari che, nel caso di molte applicazioni, sono caratterizzati da un numero considerevole di incognite, rendendo complessa la risoluzione manuale.
Ancora una volta, MATLAB viene in aiuto, visto che il software, così come è stato pensato, con la sua struttura dati a vettori e matrici, riesce agevolmente a risolvere questo tipo di problemi.
Dall'algebra, è noto che un sistema di equazioni lineari è scritto secondo:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ \qquad\qquad ... \\ a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m \end{cases}$$

La stessa scrittura può essere tradotta in forma matriciale:

[A][x]=[b]

dove [A] è la matrice dei coefficienti, [x] il vettore (mx1) delle incognite e [b] il vettore (nx1) dei termini noti.
Per sapere se un sistema di equazioni conduce all'esistenza di soluzioni, si confrontano i ranghi della matrice dei coefficienti [A] e della matrice completa [Ab], chiamati rispettivamente p e p'.
Allora, per tale condizione:

se $p \neq p'$ il sistema non è risolubile

se p=p' il sistema è risolubile e se $\begin{cases} p = n \Rightarrow unica\_soluzione \\ p < n \Rightarrow \infty^{n-p}\_soluzioni \end{cases}$

Come già visto, è possibile risolvere sistemi di equazioni con l'operatore di divisione a sinistra (\) **`x=A\b`**, siano essi indeterminati che sovradeterminati.
Infatti, nel primo caso (numero delle incognite superiore a quello delle equazioni), tale operatore fornisce una delle infinite soluzioni ponendo a zero una variabile; nel secondo caso (numero delle equazioni superiore a quello delle incognite), si ottiene una soluzione ai minimi quadrati.

Sistemi sovradeterminati

Qualora si voglia determinare – ad esempio – l'equazione di una retta di regressione, dati 3 punti, si sa che il sistema, ovviamente, non conduce ad una soluzione esatta, poiché il rango della matrice completa è diverso da quello della matrice dei coefficienti.

x	y
0	3
4	5
10	13

```
>> A=[0,1;4,1;10,1];
>> b=[3,5,13]
>> b=b'

b =

     3
     5
    13

>> rank(A)

ans =
     2

>> rank([A b])

ans =
     3
```

In tal caso, adoperando l'operatore di divisione a sinistra, esso fornisce una soluzione a minimi quadrati del sistema, restituendo l'equazione della retta di regressione y=1.03x+2.21, dati i tre punti.

```
>> A\b

ans =
    1.0263
    2.2105
```

In alternativa, per ottenere la soluzione approssimata del sistema, si sarebbe potuto usare l'operatore **`pinv(A)`** che effettua la pseudo-inversa di Moore-Penrose della matrice dei coefficienti e che minimizza la norma euclidea del vettore delle soluzioni.
Più in generale, tale operatore si può utilizzare quando la matrice A non è invertibile (cioè essa non è quadrata o a rango pieno) e ciò comporta la mancanza di unicità o di esistenza della soluzione (sistemi indeterminati o sovradeterminati).
Per esigenze di calcolo più avanzate, esistono operatori omologhi ma dagli algoritmi di approssimazione più raffinati (**`lscov e lsqnonneg`**); si veda a tal proposito l'help digitando al prompt **`help matfun`**.

Sistemi indeterminati

Un sistema può avere un numero infinito di soluzioni quando, pur avendo un numero di equazioni pari a quello delle incognite, ha la matrice dei coefficienti non singolare (cioè vi sono delle equazioni linearmente dipendenti tra loro).
Ad esempio, dato il sistema singolare:

$$\begin{cases} x + 2y = 5 \\ 3x + 6y = 15 \end{cases}$$

se si cerca di risolverlo con l'operatore di divisione sinistra, MATLAB risponde con un messaggio di errore che evidenzia come la matrice A non sia invertibile perché singolare.

```
>> A\b
Warning: Matrix is singular to working precision.
(Type "warning off MATLAB:singularMatrix" to suppress this warning.)

ans =

   Inf
   Inf
```

Utilizzando invece l'operatore **pinv** viene scelta, tra le infinite, una soluzione a norma euclidea minima.

```
>> pinv(A)*b

ans =

    1.0000
    2.0000
```

In realtà, la soluzione corretta deve essere una espressione parametrica in funzione di un parametro che descrive le infinite soluzioni.
Ciò si può ottenere con il comando **rref**

```
>> rref([A b])

ans =
     1     2     5
     0     0     0
```

Come è possibile verificare anche dai calcoli a mano, la seconda equazione è linearmente dipendente dalla prima per un fattore moltiplicativo 3 e, pertanto, è possibile ridurre il sistema ad una sola equazione parametrica x+2y=5, per la quale si può prendere x come variabile libera.
Ora, si riporta un ulteriore esempio di sistema indeterminato in cui però, questa volta, la matrice A è non singolare:

$$\begin{cases} x + 2y - 3z = 1 \\ 2x + 4y + z = 2 \end{cases}$$

```
>> rank(A)

ans =

     2

>> rank([A b])

ans =

     2

>> size(A,2)
ans =
     3
```

Poiché p=p' e p<n (2<3) , come determinato anche con le istruzioni MATLAB, si hanno ∞^1 soluzioni.
Pertanto, è possibile trovare la forma parametrica del sistema:

```
>> rref([A b])

ans =

     1     2     0     1
     0     0     1     0
```

Essa conduce alla soluzione: x=1-2k y=k z=0.

Esempio di file script per la risoluzione di sistemi

```
% File script ris_sist.m
% Risolve un sistema di equazioni sia esso determinato o indeterminato
% Fornisce una soluzione a minimi quadrati nel caso sovradeterminato
clear all
% Inserimento matrice dei coefficienti
A=input('Immettere la matrice dei coefficienti: ');
%Inserimento vettore dei termini noti
b=input('Immettere il vettore dei termini noti: ');
%Verfica di risolubilità del sistema
if rank(A)==rank([A b])
    %caso positivo-> sistema determinato o indeterminato
    if rank(A)==size(A,2) %condizione di unicità della soluzione
        %sistema singolare: numero incognite = numero equazioni ma det(A)
        %diverso da 0
        if det(A)==0
        disp ('Sistema singolare: calcolo della forma parametrica')
        rref([A b])
        else
        % sistema determinato
        disp('Sistema determinato')
        x=A\b
        end
    else
        % sistema indeterminato
        disp ('Sistema indeterminato: calcolo della forma parametrica')
        rref([A b])
    end
else
    %caso negativo->sistema sovradeterminato
    disp('Sistema indeterminato: ricerca della retta della soluzione
approssimata con il metodo dei minimi quadrati')
    x=A\b
end
```

7.4 Ricerca di zeri e punti di stazionarietà di una funzione

- **fzero**

La funzione **sol=fzero(@funzione,x0)** trova lo zero reale più vicino ad **x0** della funzione definita da un function file.

Perciò, si deve definire prima di tutto un file .m con le istruzioni seguenti:

```
function y = funzione(x)
y=3*x.^2-2*cos(x);
```

Successivamente, al prompt, si predispone l'istruzione:

```
sol=fzero(@funzione,0)
```

Così, MATLAB risponde trovando la soluzione più prossima a 0:

```
sol =

   -0.7108

>>
```

- **fminbnd - fminsearch**

MATLAB permette la ricerca di punti di stazionarietà di una funzione a una variabile (**fminbnd**) e a più variabili (**fminsearch**). Le ulteriori funzioni di ottimizzazione appartengono tutte al Toolbox Optimization, per la cui trattazione si rimanda ad altri testi.

La ricerca del punto di minimo di una funzione (ad es. la funzione definita nel function file del precedente paragrafo) nell'intervallo [-1,1] è ottenuta tramite le istruzioni:

```
min=fminbnd(@funzione,-1,1)
```

Nel caso di una funzione a due variabili, la sintassi da utilizzare è la seguente:

```
min=fminsearch(@funzione,x0)
```

dove **x0** è il vettore riga che definisce il punto nel cui intorno si cerca il minimo.

Invece, per determinare i massimi locali della funzione, basta rieditarla preponendo un segno meno nella sua espressione.

7.5 Derivazione e integrazione

- **diff**

La differenza tra gli elementi adiacenti di un vettore può essere calcolata con:

```
Y=diff(X)
```

Per calcolare la derivata numerica (rapporto incrementale) di y, il passo è breve:

```
dy=diff(y)./diff(x)
```

X può essere anche una matrice: in tal caso le differenze sono calcolate tra gli elementi presenti in colonne adiacenti.
Si può anche ottenere una differenza applicandola ricorsivamente, con l'istruzione:

```
diff(x,n)
```

- **gradient**

```
FX=gradient(F)
```

determina il gradiente della funzione **F** in due variabili lungo la direzione x.
Qualora si ricerchi il valore del gradiente della funzione nelle direzioni **x** e **y**, si scriva:

```
[FX,FY]=gradient(F)
```

E' possibile inoltre definire l'incremento lungo le due direzioni per il quale è calcolato il gradiente con la sintassi:

```
[FX,FY]=gradient(F,incr_x,incr_y)
```

Per rappresentare il gradiente come vettore avente modulo, direzione e verso, ci si può avvalere delle seguenti righe di codice:

```
i = -1:0.1:1;
[x,y] = meshgrid(i);
z = 10*cos(x.^2 + y.^2);
[fx,fy] = gradient(z,.2,.2);
contour(i,i,z)
hold on
%quiver rappresenta il gradiente come vettore avente modulo, direzione e
%verso
quiver(i,i,fx,fy)
hold off
```

Il risultato ottenuto è visualizzabile nella seguente figura 7.4.

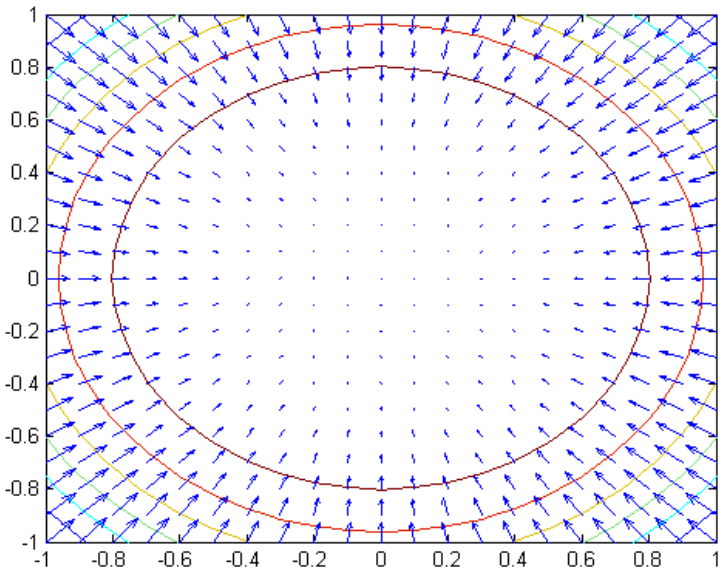


Figura 7.4 Rappresentazione del vettore gradiente

- **quad**

Effettua l'integrazione numerica secondo il metodo di Simpson.

Utilizzando una function handle, possiamo creare la funzione da integrare all'interno del file **fun_da_int.m**:

```
function y = fun_da_int(x)
y = (x.^5+x.^2+3);
```

Per calcolare l'integrale definito della funzione **fun_da_int(x)** tra 0 e 3 è sufficiente digitare al prompt la sintassi:

```
F = quad(@fun_da_int,0,3)
```

E' possibile definire anche un integrale di superficie con la sintassi:

```
F = dblquad(@fun_da_int,xmin,xmax,ymin,ymax)
```

in cui **xmin**, **xmax**, **ymin** e **ymax** rappresentano le ascisse e le ordinate estreme del dominio di integrazione.
Nel calcolo dell'integrale, sia esso semplice o doppio, è possibile definire una tolleranza che per default è fissata a 10^{-6}.

```
F = quad(@fun_da_int,0,3,tol)
```

7.6 Risoluzione di equazioni differenziali ordinarie

Un'equazione del tipo:

$y'=f(t,y)$

rappresenta un'equazione differenziale lineare del primo ordine.
Per conoscerne la soluzione è necessario avere a disposizione una condizione iniziale, per la quale:

$y(t_0)=y_0$

MATLAB riesce a trovare numericamente la soluzione delle sole equazioni differenziali lineari del primo ordine tramite il comando ODE.

Pertanto, per risolvere equazioni di ordine superiore è necessario effettuare delle sostituzioni per ridurne l'ordine, costruendo un sistema di equazioni del primo ordine.
Infatti, a partire da:

$y^{(n)}=f(t,y,y',...,y^{(n-1)})$

con le sostituzioni:

$y_1=y$
$y_2=y'$
...
$y_n=y^{(n-1)}$

Risulta:

$y_1'=y_2$
$y_2'=y_3$
$y_n'= f(t,y_1,y_2,...,y_n)$

La sintassi più semplice per risolvere un'equazione differenziale prevede la definizione dell'intervallo di tempo su cui integrare (vettore **`tempo`**) e di un vettore di condizioni iniziali (**`x0`**).

```
[t,x]=ode45(@funzione,tempo,x0);
```

Il metodo di integrazione scelto è in questo caso **`ode45`** che utilizza per la risoluzione dell'equazione differenziale l'algoritmo di Runge-Kutta.
In realtà, per equazioni differenziali stiff, cioè difficili da risolvere perché caratterizzate da dinamiche molto veloci, esistono anche altri metodi compendiati nella tabella successiva.

Solutore	Grado di stiffness equazione
ode45	basso
ode23	basso
ODE113	basso, richiesta accuratezza della soluzione
ODE15S	alto
ODE23S	alto, requisiti di accuratezza poco stringenti
ODE23T	medio alto
ODE23TB	alto, requisiti di accuratezza poco stringenti

E' facile accorgersi del fatto che un'equazione sia stiff, visto che la risoluzione di un'equazione di questo tipo, condotta con un metodo di integrazione non specificamente dedicato, conduce a tempi computazionali spesso ragguardevoli.

Esempio

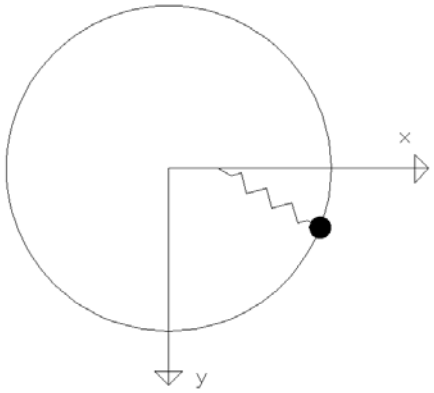


Figura 7.5 Sistema massa molla traslante su guida circolare

Risolviamo ora un'equazione differenziale lineare del secondo ordine che rappresenta l'equazione del moto del sistema meccanico in figura costituito da un peso connesso ad una molla e traslante su una guida circolare

$$m\ddot{\theta} = -mgsen\theta - \frac{1}{4}k\cos\theta$$

Il primo step da eseguire è la riduzione ad un sistema del primo ordine:

$$\dot{\theta}_1 = \theta_2$$

$$\dot{\theta}_2 = -mgsen\theta_1 - \frac{1}{4}k\cos\theta_1$$

Successivamente, si scrive il function file associato all'equazione:

```
function tetapunto=eq_moto(t,teta)
tetapunto=[teta(2);-9.81*sin(teta(1))-(10/4)*cos(teta(1))];
```

In questa prima fase, per semplicità, sia la costante elastica della molla che la massa del grave sono state assegnate direttamente (k=10; m=1) all'interno dell'equazione. Più avanti, vedremo in che modo è possibile definire delle equazioni differenziali parametriche.
Ora simuliamo l'equazione differenziale nell'intervallo di tempo compreso tra 0 e 10, con condizione iniziale x0=(0,0):

```
>> [t,teta]=ode45(@eq_moto,[0 10],[0;0]);
>> plot(t,teta(:,1),'-',t,teta(:,2),':')
```

La precedente equazione differenziale può essere riscritta mettendo in evidenza i contributi delle costanti.
Pertanto, il function file deve essere così modificato:

```
function tetapunto=eq_moto_p(t,teta)
global m k
tetapunto=[teta(2);-m*9.81*sin(teta(1))-(1/4)*k*cos(teta(1))];
```

Al prompt si digiti:

```
>>global m k
>>m=1;
>>k=10;
>> [t,teta]=ode45(@eq_moto_p,[0 10],[0;0]);
>> plot(t,teta(:,1),'m-',t,teta(:,2),'m--')
>> hold on
>>m=10;
>>k=10;
>> [t,teta]=ode45(@eq_moto_p,[0 10],[0;0]);
>> plot(t,teta(:,1),'m-',t,teta(:,2),'m--')
```

Il grafico risultante consente il confronto delle soluzioni avendo modificato il valore della massa da 1 a 10.

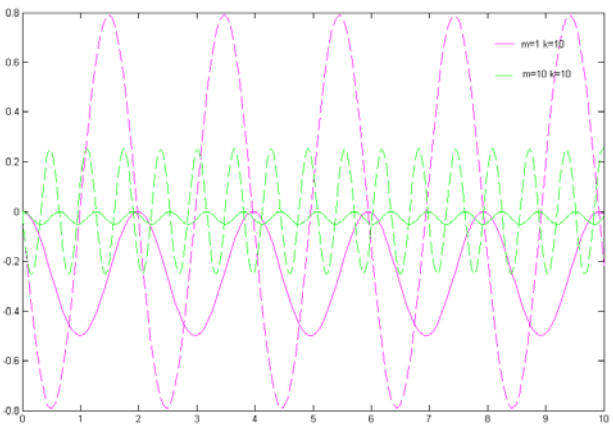


Figura 7.6 Soluzioni dell'equazione differenziale

Problemi alla frontiera

MATLAB riesce a risolvere problemi alla frontiera per equazioni differenziali ordinarie note le condizioni appartenenti a due punti. Si ottiene così una soluzione continua coma la sua derivata prima.
Per risolvere la seguente equazione:

$y''=2xy/(1+x^2)+(x+x^3)*\sin(x)$ $0<x<1$ $y(0)=0$; $y(1)=1$

si crei prima di tutto un function file per definire la funzione differenziale:

```
function ypunto=eq_diff_cc(x,y)
ypunto=[y(2);2*x*y/(1+x.^2)+(x+x.^3)*sin(x)];
```

Poi, si definisca una funzione per indicare le condizioni al contorno nella forma f(y)=0:

```
function res=cc(ya,yb)
res=[ya(0);yb(1)-1];
```

La sintassi che conduce alla risoluzione del problema è la seguente:

```
sol=bvp4c(@eq_diff_cc,@cc,solinit);
```

Nella stessa, come è evidente, si è fatto uso delle function handle precedentemente definite e della struct **`solinit`** che contiene l'informazione delle funzioni da adoperare come ipotesi di soluzione. A tale scopo, si devono prima definire le funzioni di tentativo con il file.m **`fun_tent`**:

```
function y_tent=fun_tent(x)
y_tent=[sin(x);cos(x)];
```

La scelta delle funzioni di tentativo è spesso suggerita dalla conoscenza del comportamento fisico che si ha del problema in esame.

Per definire l'ipotesi iniziale, si adoperi la scrittura:

```
solinit=bvpinit(linspace(0,1,10),@fun_tent);
```

La soluzione trovata può essere ora plottata con le istruzioni:

```
xint=linspace(0,1);
%valutazione della soluzione dell'eq. differenziale tra 0 e 1
sxint=deval(sol,xint);
plot(xint,sxint(1,:))
```

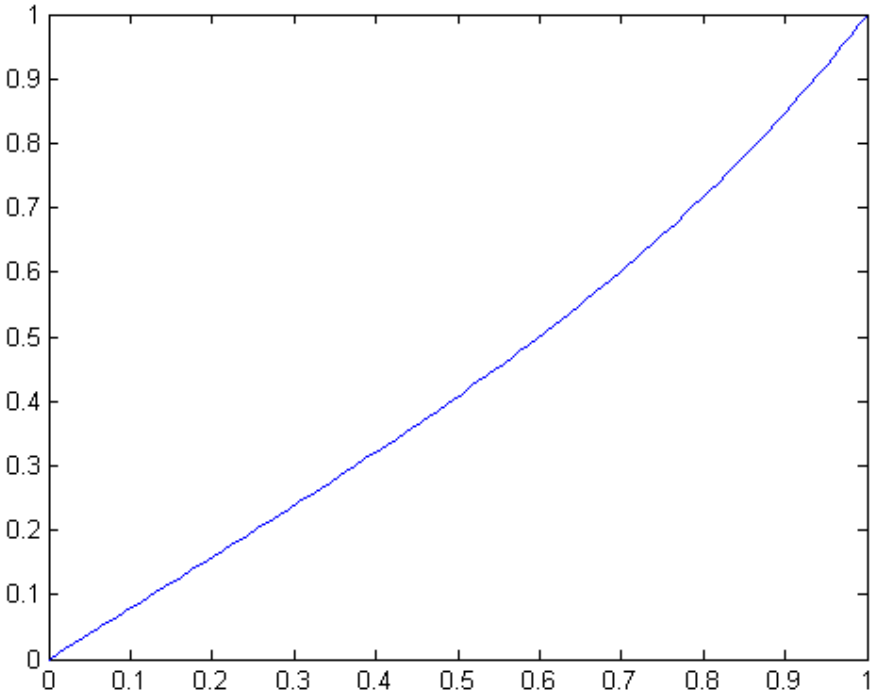


Figura 7.7 Andamento soluzione equazione differenziale

Capitolo 8

Control System Toolbox

8.1 Utilità del toolbox dei controlli

MATLAB, attraverso il tool **`Control System Toolbox`**, risulta particolarmente utile per l'analisi di sistemi dinamici e per la progettazione di sistemi di controllo.
In generale, il **`Control System Toolbox`** mette a disposizione diversi strumenti per la rappresentazione dei sistemi dinamici lineari e stazionari (in seguito chiamati LTI) e analisi del loro comportamento sia in tempo continuo che in tempo discreto, oltre che per l'implementazione dei sistemi di controllo secondo tecniche ordinarie e avanzate.
Pertanto, nel prosieguo del capitolo, verranno presentati gli strumenti per l'analisi dei sistemi lineari nel tempo e in frequenza, applicabili anche ai sistemi dinamici controllati.

8.2 Definizione di sistemi LTI

Per definire un sistema, il Toolbox dei controlli mette a disposizione le rappresentazioni secondo funzione di trasferimento, ISU e zeri-poli-guadagno attraverso l'utilizzo dei comandi elencati nella seguente tabella.

Rappresentazione		Comando
ISU	$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$	`sis=ss(A,B,C,D)`
Funzione di trasferimento	$G(s) = \frac{NUM(s)}{DEN(s)}$	`sis=tf(num,den)`
Zeri-poli-guadagno	$G(s) = K\frac{(s-z_1)(s-z_2)...(s-z_m)}{(s-p_1)(s-p_2)...(s-p_n)}$	`sis=zpk([z1,z2,…zm],[p1,p2,…,pn],K)`

Rappresentazione ISU

A titolo di esempio, si vuole creare la rappresentazione ISU in MATLAB del sistema LTI caratterizzato dai seguenti array:

$$A=\begin{bmatrix}3 & 5 & 2\\ 0 & 1 & 0\\ 1 & 0 & 0\end{bmatrix} \qquad B=\begin{bmatrix}0\\ 1\\ 0\end{bmatrix}$$

C=[2 7 1] D=0

Pertanto, al prompt di MATLAB si devono prima di tutto definire i suddetti array:

```
>> A=[3 5 2;0 1 0;1 0 0];
>> B=[0;1;0];
>> C=[2 7 1];
>> D=0;
```

Definite queste variabili, si può ora utilizzare il comando **ss** per la creazione della rappresentazione ISU del sistema.

```
>> sis=ss(A,B,C,D)
```

Matlab dà la seguente uscita:

```
a =
       x1  x2  x3
   x1   3   5   2
   x2   0   1   0
   x3   1   0   0


b =
       u1
   x1   0
   x2   1
   x3   0


c =
       x1  x2  x3
   y1   2   7   1


d =
       u1
   y1   0

Continuous-time model.
```

Naturalmente, al posto di array numerici si possono utilizzare matrici parametriche che utilizzano parametri significativi del comportamento fisico dell'applicazione.
In questo modo, è possibile ottenere un'utile variazione parametrica del modello, in modo da valutare i differenti comportamenti del sistema al variare dei parametri, come evidenziato in seguito.
A questo proposito, si crea la rappresentazione ISU del modello di un motore a c.c ad eccitazione costante nel quale, variando la tensione di armatura, si ottiene il controllo della velocità del suo asse di rotazione.
Il modello matematico del motore è definito da alcuni parametri afferenti alla dinamica elettrica e meccanica.

```
R= 2.0 % Ohm
L= 0.5 % Henry
Kt = .015 % costante di coppia
Kv = .015 %  costante di velocità
Kf = 0.2 % coefficiente di attrito viscoso
J= 0.02 % kg.m^2/s^2 %momento di inerzia del rotore

%Rappresentazione ISU
A = [-R/L -Kv/L; Kt/J -Kf/J]
B = [1/L; 0];
C = [0 1];
D = [0];
motore_ss = ss(A,B,C,D)
```

Variazione parametrica del modello

Definendo un vettore di parametri **K,** è possibile indagare il comportamento del sistema al variare di alcuni parametri del modello attraverso, ad esempio, la risposta al gradino.

```
K = [0.1 0.15 0.2]; % variazione dei valori di Kt e Kv
A1 = [-R/L -K(1)/L; K(1)/J -Kf/J];
A2 = [-R/L -K(2)/L; K(2)/J -Kf/J];
A3 = [-R/L -K(3)/L; K(3)/J -Kf/J];
motore_ss(:,:,1)= ss(A1,B,C,D);
motore_ss(:,:,2)= ss(A2,B,C,D);
motore_ss(:,:,3)= ss(A3,B,C,D);
motore_ss
step(motore_ss)
```

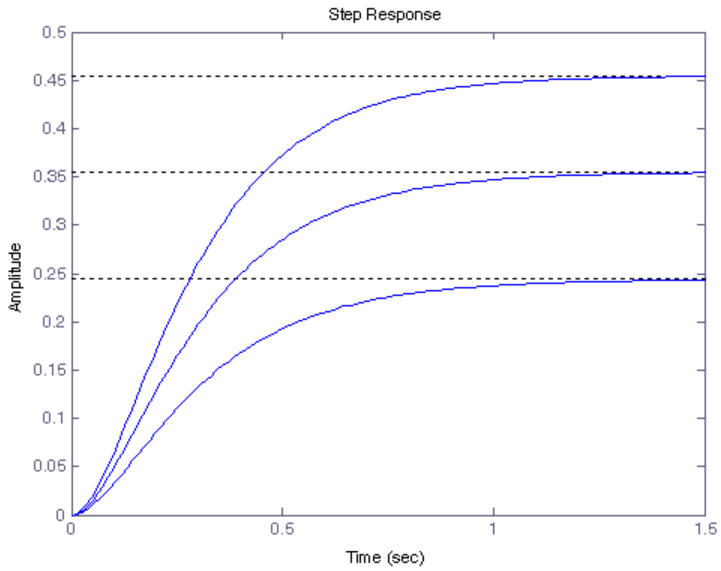


Figura 8.4 Risposte al gradino ottenute al variare dei parametri

Rappresentazione secondo funzione di trasferimento

Data la f.d.t. del sistema:

$$G(s) = \frac{s^2 + 4s + 5}{s^3 + 3s^2 + 4s + 1}$$

in MATLAB si ottiene la corrispondente rappresentazione con il comando **`tf`**, definendo i polinomi del numeratore e del denominatore.

```
>>sis=tf([1,4,5],[1,3,4,1])
```

In aggiunta, avendo definito la f.d.t. del sistema, è possibile ottenere direttamente zeri, poli e guadagno rispettivamente con i comandi:

```
>>zero(sis)

>>pole(sis)

>>dcgain(sis)
```

Rappresentazione secondo zeri-poli-guadagno

Dalle rappresentazioni ISU e secondo f.d.t. è possibile passare alla corrispondente rappresentazione secondo zeri-poli-guadagno con le seguenti istruzioni applicate al caso del motore:

```
>>motore_zpk = zpk(motore_ss)
```

8.3 Conversione di rappresentazioni - Scelta del tipo di variabili

MATLAB permette di raccogliere il risultato di ogni tipo di rappresentazione di un modello del sistema in array di celle. E'possibile definire un parametro aggiuntivo **Ts** afferente al tempo di campionamento.

```
[num,den,Ts] = tfdata(sys)
[z,p,k,Ts] = zpkdata(sys)
[a,b,c,d,Ts] = ssdata(sys)
[response,frequency,Ts] = frdata(sysfr)
```

Con l'istruzione seguente è possibile scegliere il tipo vettoriale di variabile con cui vengono salvati i valori numerici degli zeri, poli e guadagno.

```
[z,p,k,Ts] = zpkdata(sys,'v')
```

8.4 Visualizzazione e modifica delle proprietà dei sistemi LTI

I comandi appena visti creano la rappresentazione dei sistemi LTI secondo oggetti avente determinate proprietà.

Per visualizzare una proprietà particolare del modello si usa:

```
PropertyValue = get(sys,PropertyName)
```

Per modificare le proprietà associate all'oggetto si ha a disposizione la sintassi generale:

```
set(sys,'Property',Value)
```

Ad esempio, in tal modo è possibile assegnare il nome all'ingresso e all'uscita del modello del motore elettrico.

```
set(motore_ss,'InputName',V_arm,'OutputName',omega)
```

La verifica della variazione effettuata si ottiene ridigitando il comando **get**

```
>> get(motore_ss)
            a: [2x2 double]
            b: [2x1 double]
            c: [0 1]
            d: 0
            e: []
    StateName: {2x1 cell}
           Ts: 0
      ioDelay: 0
   InputDelay: 0
  OutputDelay: 0
    InputName: {'V_arm'}
   OutputName: {'omega'}
   InputGroup: [1x1 struct]
  OutputGroup: [1x1 struct]
        Notes: {}
     UserData: []
```

8.5 Rappresentazione della risposta di sistemi LTI

In MATLAB è possibile ottenere la rappresentazione della risposta nel tempo e in frequenza di sistemi LTI secondo due modalità differenti.
Nella prima modalità, quella più classica, si utilizza il prompt dei comandi, avendo disposizione i comandi elencati nella seguente tabella.

Comando	Rappresentazione
impulse	Risposta all'impulso
step	Risposta al gradino
bode	Diagrammi di Bode
plotnyquist	Diagrammi di Nyquist
nichols	Diagrammi di Nichols
freqresp	Risposta in frequenza
gensig	Generazione di un segnale di input
initial	Evoluzione libera
iozmap	Mappa poli/zeri per ciascuna coppia di input/output
lsim	Risposta ad un ingresso generico
margin	Rappresentazione margine di guadagno e margine di fase
plotpzmap	Mappa poli-zeri

Ad esempio, dal prompt, è possibile calcolare l'evoluzione libera e la risposta forzata ad un segnale generico di un sistema LTI precedentemente definito.
La funzione **`initial`** diagramma l'evoluzione libera del sistema, a partire da un determinato stato iniziale.

```
>> initial(motore_ss,[1 0])
```

Invece, per calcolare la risposta del modello del motore ad un ingresso sinusoidale, tra 0 e 10 secondi, si utilizzi:

```
>>t = 0:0.01:10;
>> u = sin(t);
>> y=lsim(motore_ss,u,t);
>>plot(t,y)
```

LTI Viewer

Spesso è consigliabile utilizzare una seconda modalità di rappresentazione, più *user friendly*, attraverso la quale si possono sfruttare tutti i vantaggi dell'interattività dell'interfaccia grafica propria di **`LTI Viewer`**.
Tale ambiente GUI di MATLAB è appositamente predisposto per il plot di sistemi lineari, consentendo le stesse rappresentazioni ottenibili con i comandi appena visti:

- Risposta al gradino e all'impulso;
- Diagrammi di Bode e Nyquist;
- Diagramma di Nichols;
- Diagramma delle ampiezze della risposta in frequenza a valori singolari;
- Rappresentazione poli/zeri;

Per accedere a **`LTI Viewer`**, si digiti al prompt:

```
>>ltiviewer
```

MATLAB risponde mostrando la finestra grafica associata al comando.

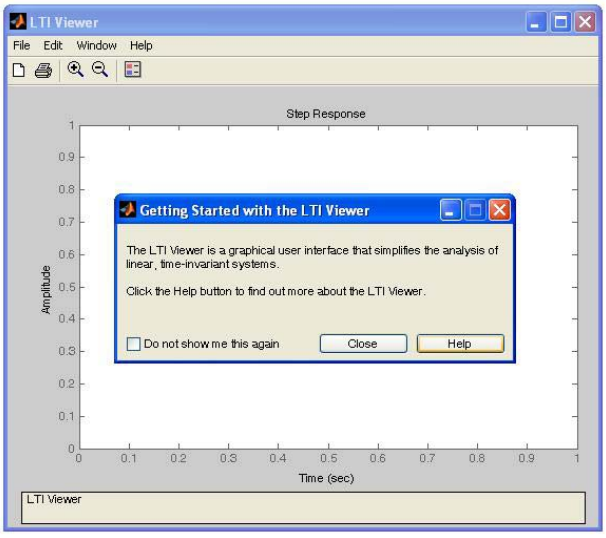


Figura 8.5 Interfaccia grafica di LTI Viewer

Come primo passo per la rappresentazione di un sistema LTI, si deve prima di tutto indicare a MATLAB il modello da rappresentare scegliendolo eventualmente tra quelli presenti nel workspace o caricandolo da una determinata directory.
Pertanto, si selezioni la voce **`Import`** dal menu **`File`** e si indichi a MATLAB il modello del sistema da rappresentare.
Ad esempio, si selezioni la rappresentazione ISU del modello del motore elettrico prima definito.
LTI Viewer rappresenta la risposta al gradino del sistema.
Continuando con questa rappresentazione, è possibile far calcolare a MATLAB e visualizzare i parametri caratteristici della risposta al gradino.

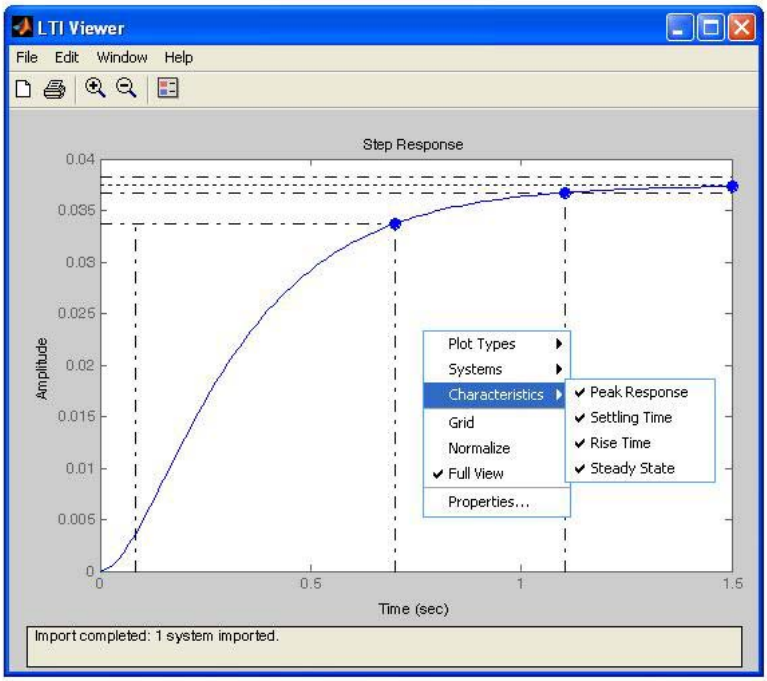


Figura 8.6 Parametri caratteristici della risposta al gradino

Ciò è ottenuto cliccando con il tasto destro sull'area del grafico e scegliendo l'opzione **`Characteristics.`**
Tramite questa opzione, selezionando i nomi dei parametri caratteristici della risposta al gradino indicati nel sottomenu, è possibile visualizzare sull'area del grafico il valore in scala dei parametri stessi.
Invece, tramite la voce **`Plot Types`**, è possibile scegliere la rappresentazione grafica da diagrammare nella finestra.

Capitolo 9

Introduzione all'uso di Simulink

9.1 Uno strumento fondamentale per la simulazione di sistemi dinamici

`Simulink` è uno strumento di lavoro fondamentale e imprescindibile per chi studia la modellistica e simula il comportamento dei sistemi dinamici.
Come già evidenziato nel nome, che si compone delle parole *Simulation* e *Link*, questo ambiente software di MATLAB consente la simulazione di sistemi dinamici secondo una rappresentazione in schemi a blocchi, siano essi lineari, nonlineari, in forma continua o discreta.
Nel prosieguo del capitolo, si farà una prima introduzione all'uso di Simulink, la quale trattazione comunque non sarà esaustiva di tutte le potenzialità del toolbox che sono veramente numerose.
Perciò, per avviare velocemente all'utilizzo di Simulink, la trattazione di questo capitolo sarà svolta *by examples*, guidando alla costruzione, alla simulazione e all'analisi dei risultati di simulazione di semplici modelli di sistemi dinamici appartenenti al dominio meccanico e a quello elettrico.

9.2 Primi passi in Simulink

Per lanciare Simulink, si digiti alla barra dei comandi:

```
>>simulink
```

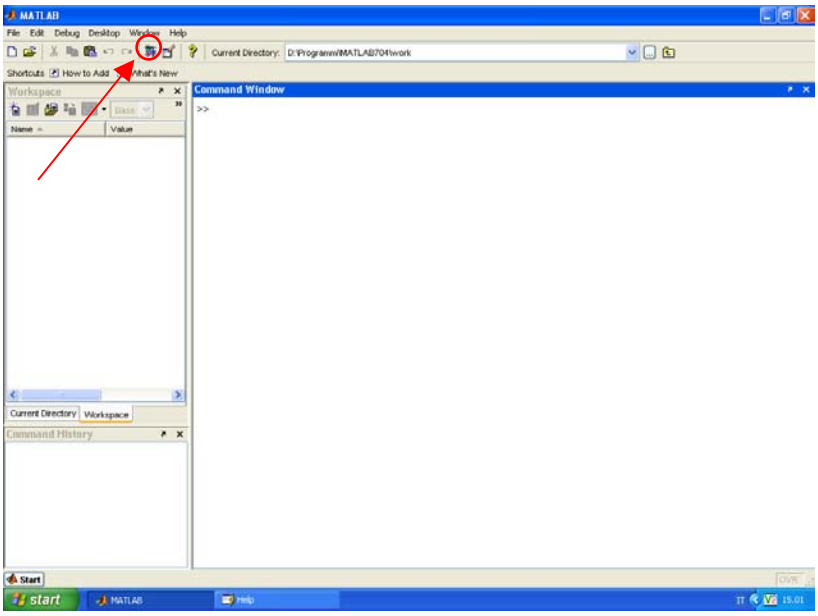


Figura 9.1 Avvio di Simulink

In alternativa, più velocemente, si clicchi sul corrispondente tasto evidenziato in Figura 9.1. MATLAB risponde al comando visualizzando la libreria dei blocchi.

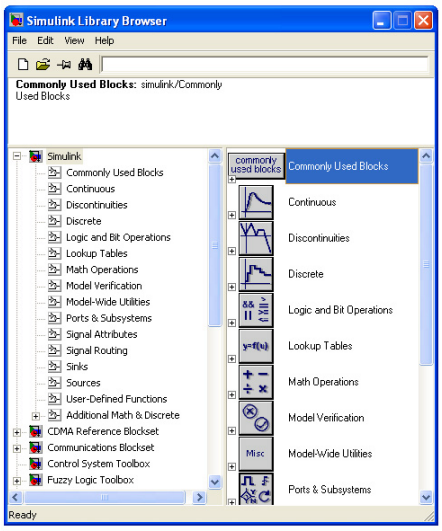


Figura 9.2 Simulink Library Browser

Ad esempio, accedendo all'interno della libreria dei blocchi ***Continous***, si trovano cinque blocchi che sono fondamentali per l'analisi dei sistemi dinamici:

- **`Derivative`**
- **`Integrator`**
- **`State space`**
- **`Transfer Fcn`**
- **`Zero-pole`**

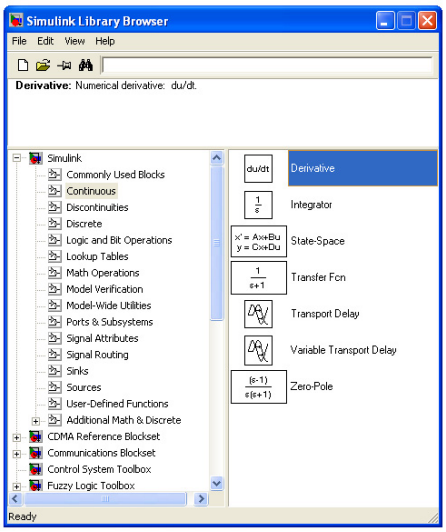


Figura 9.3 Libreria Continous

Come si osserva dalla notazione adottata nella rappresentazione dei blocchi, anche in Simulink si preferisce lavorare nel dominio di Laplace.

Si ricorda che, adottando la variabile di Laplace **s**, l'operatore $\frac{1}{s}$ corrisponde all'operatore di integrazione $\int$ nel dominio del tempo.

Gli esempi di schemi Simulink di seguito riportati utilizzano, in generale, blocchi reperibili nella librerie ***Continous*** e ***Commonly used Blocks***.

Per creare un nuovo modello Simulink (avente estensione del file .mdl) si clicca sull'icona raffigurante una pagina bianca e presente nella barra degli strumenti della finestra ***Simulink Library Browser*** visualizzabile in figura 9.2.

Esempio 1: Simulazione del sistema massa – molla

Un semplice esempio di sistema dinamico è costituito dal sistema massa-molla rappresentato in figura 9.4.

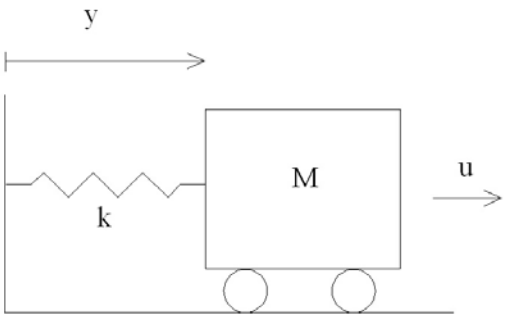


Figura 9.4 Schema del sistema massa-molla

Sulla massa è applicata una forza esterna **u(t)** che genera lo scorrimento della massa **M**. A seguito dello spostamento della stessa massa di 1 kg, sulla molla (avente costante elastica **k=1 N/mm**) si ha un allungamento **y**.

Si ipotizza la presenza di un attrito viscoso tra massa e piano di scorrimento di coefficiente **B=0.1**.

Per poter creare lo schema Simulink, si devono prima di tutto individuare quali sono gli ingressi e le uscite del sistema.

La forza esterna **u(t)** rappresenta l'ingresso e l'elongazione della molla **y** è l'uscita del sistema.

Il modello dinamico del sistema è ottenuto individuando le forze agenti sull'elemento di massa M e, applicando la seconda legge di Newton, si ha:

$$u(t) - Kx - B\dot{x} = M\ddot{x}$$

Per implementare successivamente il modello dinamico del sistema nello schema Simulink, è opportuno isolare la derivata seconda.

$$\ddot{x} = \frac{1}{M}(u(t) - kx - B\dot{x})$$

Ora si procede a costruire lo schema Simulink dell'equazione dinamica sopra determinata trascinando prima di tutto il blocco corrispondente all'ingresso a gradino (***Step***).

Gli altri blocchi appartenenti allo schema sono tutti reperibili dalla libreria ***Commonly used Blocks***.

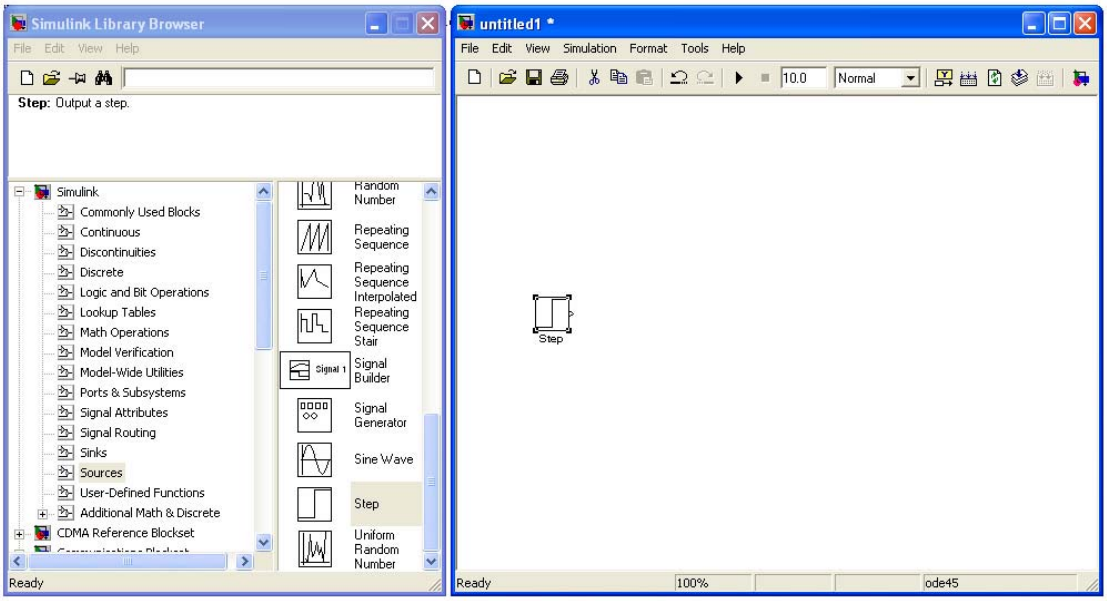


Figura 9.5 Inserimento del blocco Step

Successivamente si inserisce un nodo sommatore (***Sum***) di forma rettangolare avente tre ingressi (+ - -).

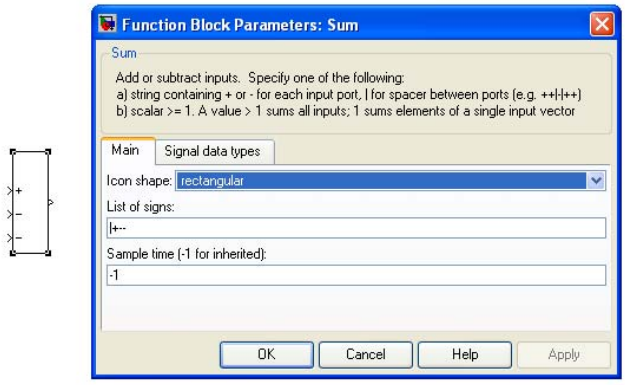


Figura 9.6 Modifica dei parametri del blocco Sum

Alla finestra di settaggio dei parametri del blocco, si accede cliccando due volte sull'icona del nodo sommatore appena inserito.
A questo scopo, si indica l'opzione ***rectangular*** nel campo ***icon shape*** e si immettono i segni **+ - -** nel campo ***list of signs***, come indicato in figura 9.6.

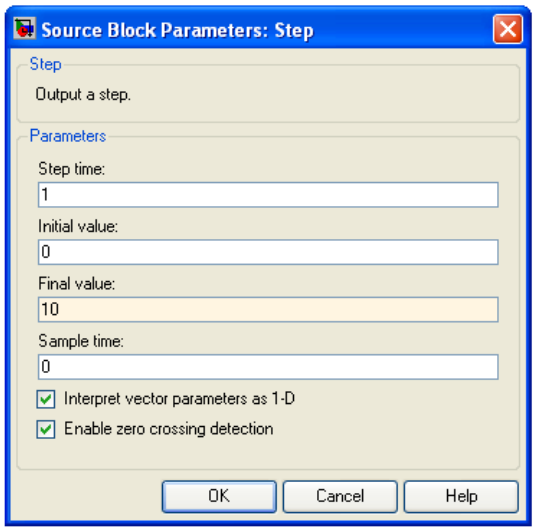


Figura 9.7 Modifica dei parametri del blocco Step

Allo stesso modo, è possibile settare a 10 il valore dell'ampiezza del segnale a gradino utilizzando il campo ***Final value*** della finestra di modifica dei parametri del blocco ***Step***.
Si aggiungono poi due blocchi integratori e un blocco ***Scope,*** il quale permette di visualizzare in una finestra a parte l'andamento del segnale derivante dalla linea di connessione in ingresso al blocco.
Resta da aggiungere un guadagno di valore 1/M tra il nodo sommatore e il blocco integratore.
Si proceda ora alla connessione dei blocchi.

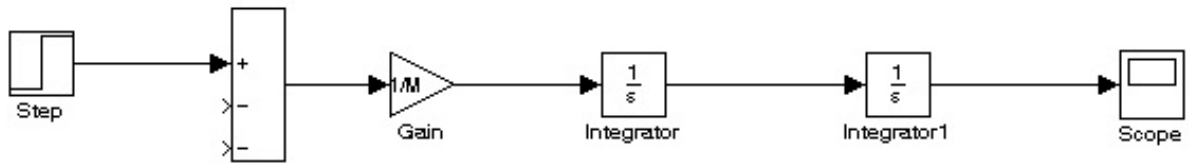


Figura 9.8 Connessione dei blocchi prima inseriti

Due blocchi vengono connessi tra loro muovendo il puntatore del mouse dal punto di uscita del primo blocco fino al punto di inserimento del blocco successivo e tenendo contemporaneamente premuto il tasto sinistro.
Per rendere più leggibile la rappresentazione, è opportuno etichettare le linee di connessione dello schema con il nome del segnale che le attraversa, facendo doppio clic con il tasto sinistro del mouse sopra ogni linea da etichettare e immettendo da tastiera la stringa di commento.

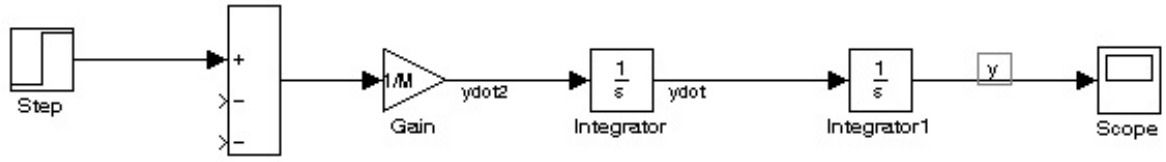


Figura 9.9 Etichettatura delle connessioni

Per concludere lo schema, si devono aggiungere i blocchi corrispondenti ai termini di forza elastica della molla e di forza di attrito viscoso presenti nel modello dinamico.
Pertanto, si devono prima di tutto creare delle diramazioni del segnale di posizione e velocità da connettere poi a due ***gain*** (reperibili nella libreria ***Commonly used Blocks***), per rappresentare i termini di attrito e forza elastica presenti nell'equazione dinamica.
Le diramazioni vengono ottenute ponendosi con il puntatore sopra la linea di ciascuno di questi segnali e trascinando il mouse mentre si tiene premuto il tasto destro.

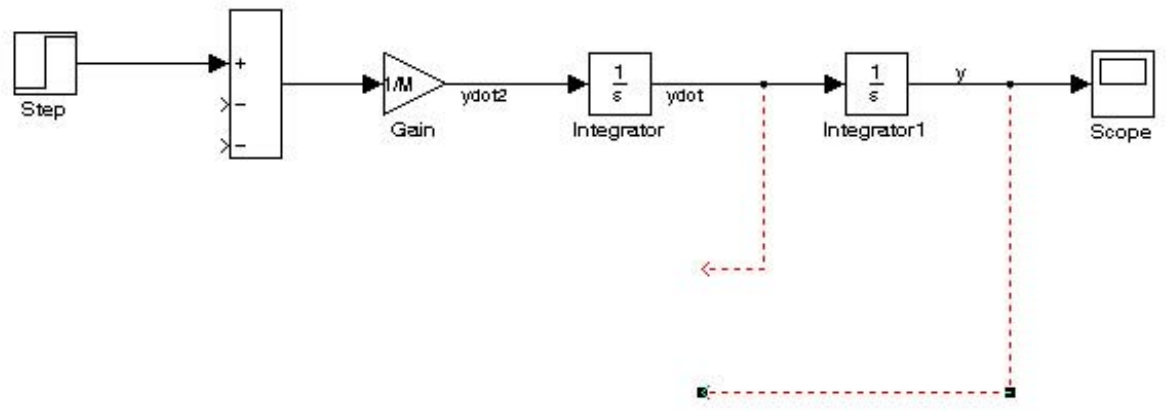


Figura 9.10 Creazione delle diramazioni delle linee di segnale

Ora, si inseriscono i due guadagni (***gain***), provvedendo a ruotarli di 180° in modo da orientare le loro porte di ingresso a destra, così da permettere la connessione dei blocchi all'interno dello schema seguendo il verso delle linee di segnale di posizione e velocità.
La rotazione dei ***gain*** è ottenuta cliccando con il tasto destro su di essi e scegliendo l'opzione del menu contestuale ***flip block***.
Successivamente, si cambino i nomi ai guadagni, rietichettandoli appropriatamente.
Per fare ciò, si clicchi in prossimità dell'etichetta del guadagno – il puntatore assume così la forma di un cursore lampeggiante – e si immetta la nuova stringa.

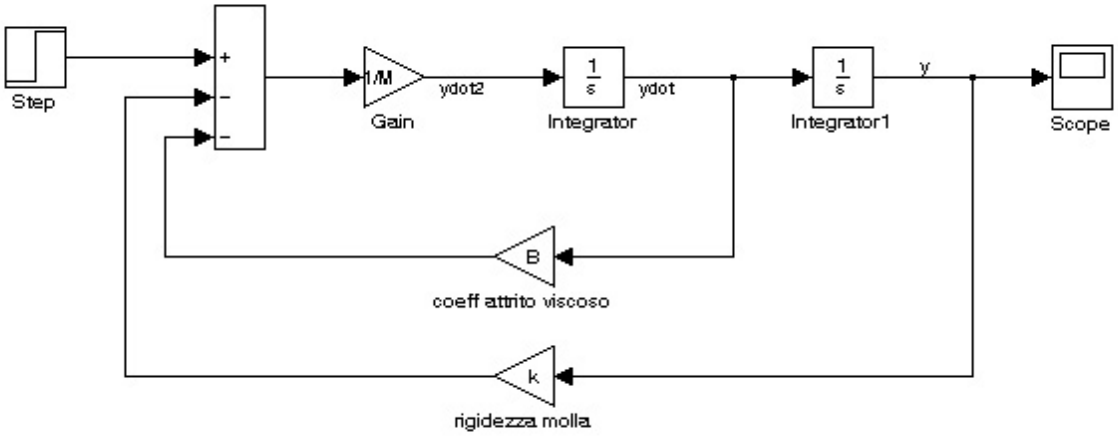


Figura 9.11 Schema Simulink completo

Infine, dalla **`Command Window`** si assegnino i valori ai parametri del modello:

```
>> M=1;
>> B=0.1;
>>K=1;
```

Il modello è ora pronto per la simulazione.
Prima di avviare la simulazione, è spesso utile definire i parametri della simulazione attraverso la finestra **simulation parameters**, accessibile dal menu **Simulation→Configuration parameters**.
E' così possibile scegliere il tipo di integratore da adottare in funzione della stiffness dell'equazione differenziale (come già descritto nel capitolo 7) o, ad esempio, settare il tempo finale di simulazione.

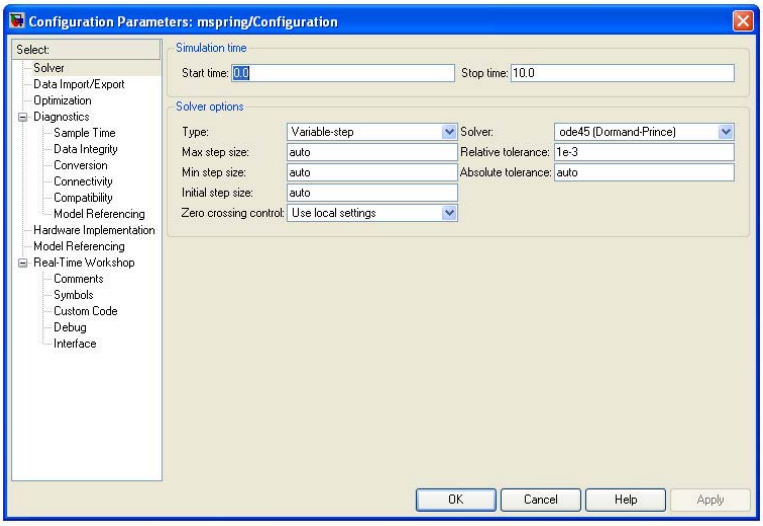


Figura 9.12 Finestra di configurazione dei parametri della simulazione

Per avviare la simulazione si clicchi sul tasto play presente sulla barra dei menu.

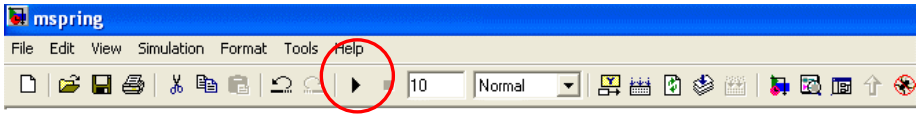


Figura 9.13 Tasto play per l'avvio della simulazione

Infine, per visualizzare l'andamento dell'uscita ***y*** del modello, si clicchi due volte in corrispondenza dello ***Scope.***

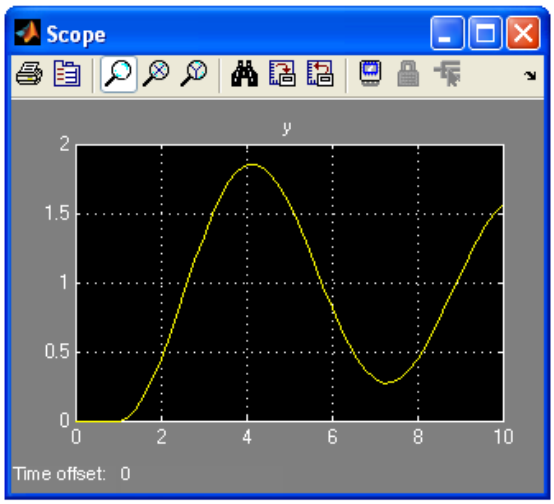


Figura 9.14 Finestra dello Scope

MATLAB risponde visualizzando il grafico dell'uscita in una finestra grafica che ha il layout tipico dello schermo di un oscilloscopio.
Cliccando sull'icona del cannocchiale presente sulla barra degli strumenti della stessa finestra, è possibile scalare automaticamente la finestra del plot.
Inoltre, attraverso i tasti che raffigurano una lente di ingrandimento, è possibile zoomare in avvicinamento o in allontanamento su di una particolare area del grafico o scalare gli assi.

Esempio 2: Modellistica di sistemi LTI- Rappresentazione ISU e secondo f.d.t.

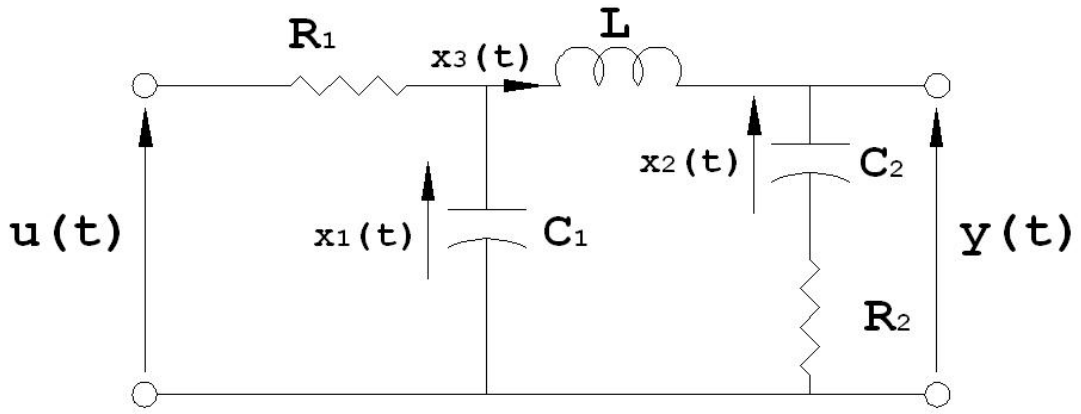


Figura 9.15

In questo esempio, a partire dalla determinazione del sistema dinamico lineare invariante proprio del terzo ordine dello schema elettrico rappresentato in figura, si creeranno in Simulink i modelli ISU e secondo f.d.t. del sistema, utilizzando rispettivamente i blocchi ***State-Space*** e ***Transfer Fcn*** reperibili nella libreria ***Continous***.
A questo scopo, si assumono come variabili di stato **`x1(t)`** e **`x2(t)`** le tensioni ai capi dei condensatori.
Con **`x3(t)`** si identifica come variabile di stato la corrente circolante nell'induttore.
Utilizzando le leggi fondamentali dell'elettrotecnica, si ottengono le seguenti relazioni tra le variabili elettriche di interesse:

$$C_1\dot{x}_1(t) = \frac{u(t) - x_1(t)}{R_1} - x_3(t)$$
$$C_2\dot{x}_2(t) = x_3(t)$$
$$L\dot{x}_3(t) = x_1(t) - x_2(t) - R_2 x_3(t)$$
$$y(t) = x_2(t) + R_2 x_3(t)$$

I parametri del modello hanno i seguenti valori:

$R_1=R_2=200\Omega$
$C_1=1mF$
$C_2=2.5mF$
$L=50H$

Le matrici associate alla rappresentazione ISU del sistema valgono:

$$A=\begin{bmatrix} -1/R_1C_1 & 0 & -1/C_1 \\ 0 & 0 & 1/C_2 \\ 1/L & -1/L & -R_2/L \end{bmatrix} \qquad B=\begin{bmatrix} 1/R_1C_1 \\ 0 \\ 0 \end{bmatrix}$$

$C=[0\ 1\ R_2]$ $D=0$

Dalla **`Command Window`**, o meglio all'interno di un m-file, si creino le variabili associate ai parametri del modello:

```
>> R1=200;
>> R2=200;
>> C1=10^(-3);
>> C2=2.5*10^(-3);
>> L=50;
```

Si costruisca lo schema a blocchi della rappresentazione ISU del sistema, inserendo il blocco ***State-Space*** dalla libreria ***Continous*** e utilizzando come ingresso un gradino di ampiezza 10V.

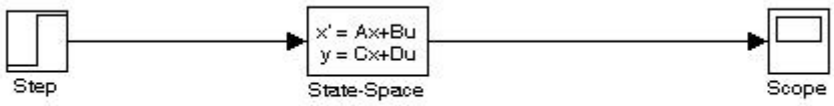


Figura 9.16 Rappresentazione ISU

Successivamente, si acceda alla finestra dei parametri del blocco ***State – Space*** e si riportino le matrici del modello ISU sopra determinato.

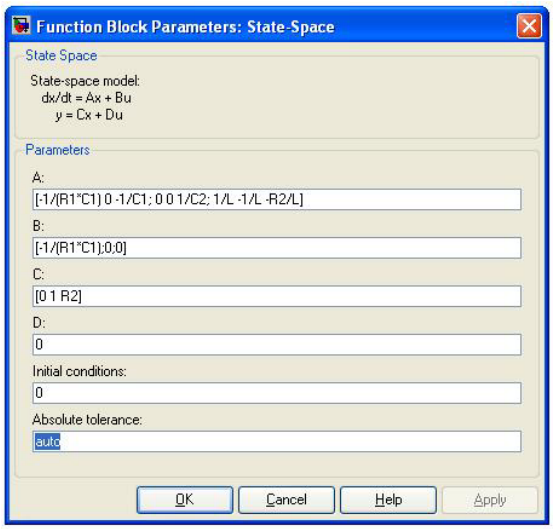


Figura 9.17 Finestra di modifica dei parametri del blocco *State - Space*

Infine, si può avviare la simulazione e ottenere il grafico dell'uscita ***y***.
In alternativa, si può costruire la rappresentazione in Simulink del sistema secondo funzione di trasferimento.
Per trovare la f.d.t. a partire dalle matrici di stato, si utilizza la classica formula di trasformazione:

$$G(s) = C(sI - A)^{-1} B$$

Sostituendo i valori numerici dei parametri elettrici del sistema, si ottiene la seguente f.d.t:

$$G(s) = \frac{20s + 40}{s^3 + 9s^2 + 48s + 40}$$

Questa f.d.t. deve essere implementata nello schema Simulink utilizzando il blocco ***Transfer Fcn*** (reperibile nella libreria ***Continous***)
Si assegnino numeratore e denominatore della f.d.t. dalla finestra dei parametri del blocco.

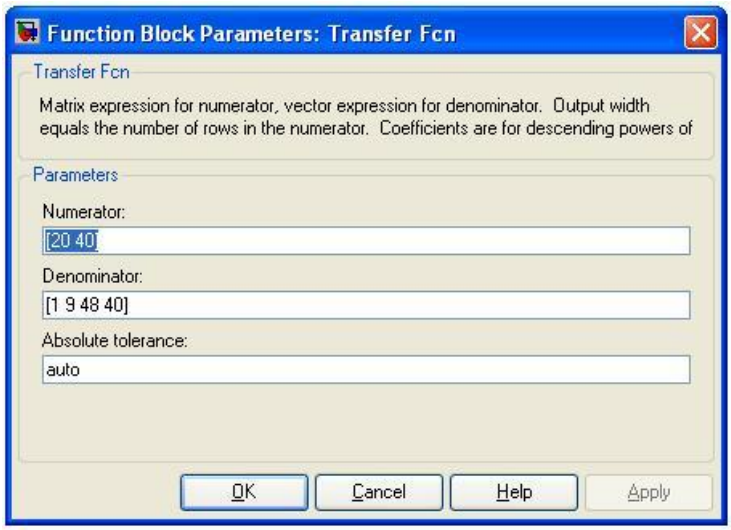


Figura 9.18 Finestra di modifica dei parametri del blocco Transfer Fcn

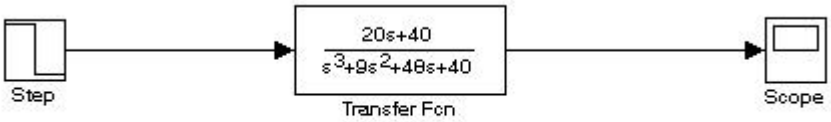


Figura 9.19 Schema Simulink secondo f.d.t.

Esempio 3: Linearizzazione dell'equazione del moto del pendolo

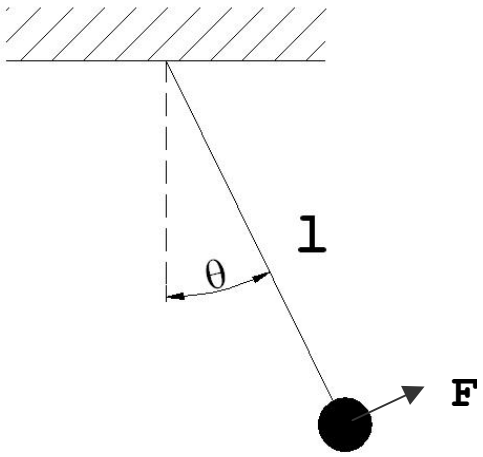


Figura 9.20 Schema del pendolo

Lo studio del moto del pendolo raffigurato in figura 9.19 conduce ad un modello dinamico tipicamente non lineare.

$$Ml^2\ddot{\theta} + B\dot{\theta} + Mgl\sin\theta = Fl$$

Per la costruzione del modello, si assuma una massa di **`M=1kg`**, una forza applicata **`F`**=**`1N`**, una lunghezza **`l`** dell'asta pari a **`1m`** e si consideri, in corrispondenza della cerniera, la presenza di un attrito viscoso avente coefficiente **`B=0.1`**.
In questo esempio, ci si propone di giungere al modello linearizzato del sistema nell'intorno del punto di equilibrio in 0.
Ciò sara fatto utilizzando **`Simulink Control Design`**, un tool di Simulink dotato di GUI che, tra le altre funzionalità, consente di ottenere la linearizzazione di un sistema dinamico implementato in uno schema Simulink.
Pertanto, la creazione dello schema a blocchi del pendolo ripercorre gli stessi passi visti negli esempi precedenti.
Il nuovo schema utilizza gli stessi tipi di blocchi dell'esempio del sistema massa – molla con, in più, il blocco ***sin*** reperibile nella libreria ***Math Operations.***

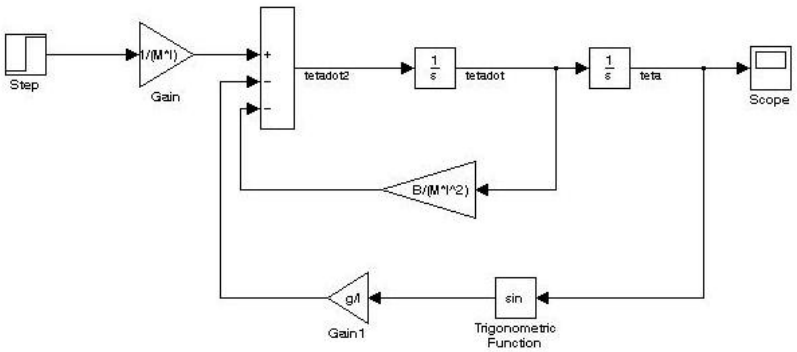


Figura 9.21 Schema a blocchi del modello dinamico del pendolo

In generale, l'analisi lineare del sistema in Simulink parte dalla definizione dello schema a blocchi del sistema.
Successivamente, si devono ricercare i punti di equilibrio attorno cui effettuare la linearizzazione.
A questo scopo, MATLAB mette a disposizione la funzione **`trim`** che fa riferimento, nella sua forma più semplice, al solo nome dello schema Simulink precedentemente costruito.

```
>>[x,u,y,dx] = trim('nomeschema')
```

MATLAB consente altresì di ricercare condizioni di equilibrio specificando il punto operativo.

Ad esempio:

```
>>x0 = [0; 0];
>>u0 = 0;
>>y0 = 1;
```

Inoltre, è possibile indicare quali variabili devono essere fissate ad un determinato valore nell'operazione di linearizzazione.

```
>>ix = [];
>>iu = [];
>>iy = 1;
```

In tal caso, solo la variabile di uscita è stata fissata e posta pari a 1.
Infine, si possono trovare i punti di equilibrio nell'intorno di un punto operativo e fissando l'uscita ad 1 con la sintassi compatta:

```
>>[x,u,y,dx] = trim('nomeschema',x0,u0,y0,ix,iu,iy)
```

È da notare che, quando si studia la linearità di una porzione di uno schema complesso, si estrae dal sistema complessivo un sottosistema, perdendo così il punto operativo originario.
Al contrario, con l'interfaccia grafica di linearizzazione dei sistemi di Simulink è possibile fare lo studio della linearità mantenendo il punto di lavoro invariato.
Lo stesso tool grafico fornisce molte funzionalità applicabili a tale analisi.
Per accedere all'ambiente di lavoro di **`Simulink Control Design`** si usa il menu di Simulink **Tool →Control Design→Linear Analysis**.

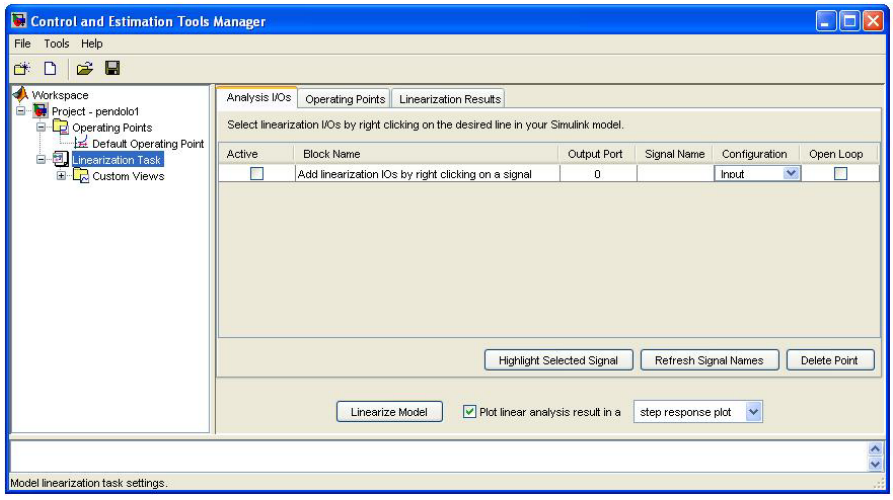


Figura 9.22 Interfaccia di Simulink per la linearizzazione

Come primo step della procedura guidata di linearizzazione, si deve scegliere il punto di linearizzazione all'interno della finestra accessibile attraverso il precedente menu.
Perciò, una volta attivato il flag nel box della scheda **Analysis I/Os** appartenente alla stessa finestra, cliccando con il tasto destro sulla linea di connessione corrispondente al segnale di interesse nello schema Simulink, è possibile indicare il punto da utilizzare per effettuare la linearizzazione.
All'interno del menu contestuale comparso si selezioni **Linearization Points → Input Point.**
Ugualmente, per definire il punto di uscita per l'operazione di linearizzazione si utilizzi l'opzione **Linearization Points → Output Point.**
Si noti che in corrispondenza della linee di segnale marcate per la linearizzazione compare un simbolo a forma di freccia

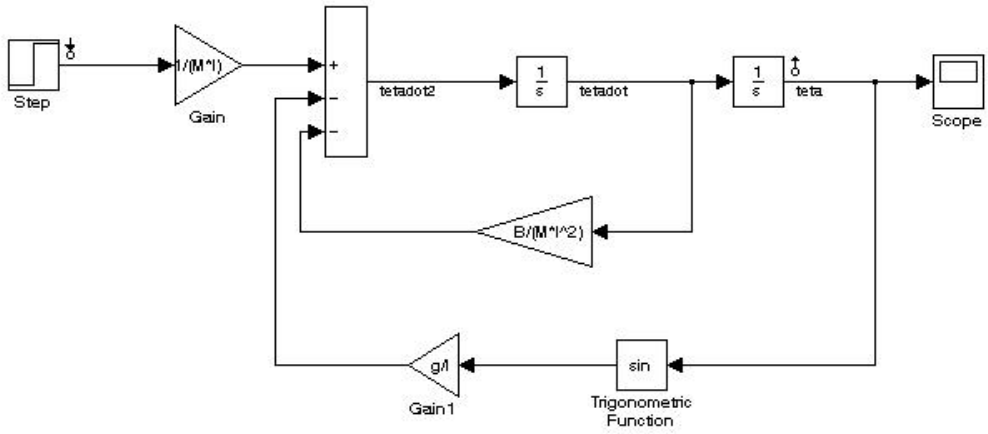


Figura 9.23 Indicazione dei punti di input/output per la linearizzazione

Nel caso di schemi Simulink abbastanza articolati, per "estrarre" da tutto lo schema solo il blocco afferente al sottosistema di interesse, è consigliabile cliccare con il tasto destro sul segnale di uscita e selezionare **Linearization Points → Open Loop**.
Attraverso la struttura ad albero del progetto di linearizzazione corrente caratterizzante l'interfaccia di Simulink per la linearizzazione, è possibile accedere, dalla directory **Operating Point**, alla finestra **Create Operating Point**.

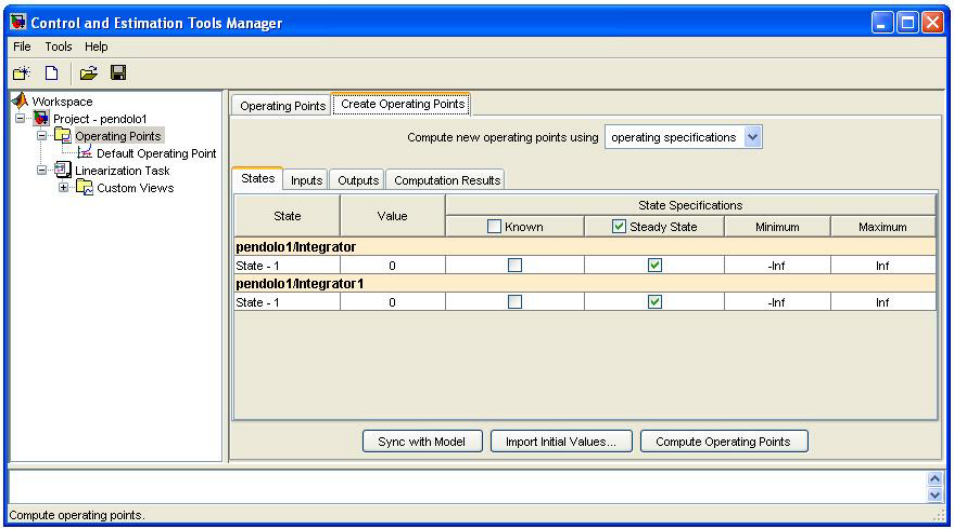


Figura 9.24 Finestra Create Operating Point

La suddetta finestra, attraverso l'opzione **Operating specification** del box **Compute new operating point**, permette di indicare un punto operativo preciso per effettuare la linearizzazione specificando i valori assunti dalle variabili di stato.
Nel caso del presente esempio, non è necessario aggiungere un ulteriore punto operativo, visto che il punto di lavoro scelto per default da Simulink, e pari a [0,0], coincide con il punto di equilibrio inizialmente stabilito per effettuare la linearizzazione del pendolo.
È da notare che, in alternativa, utilizzando l'opzione **simulation snapshot** dello stesso box **Compute new operating point** e' possibile creare un nuovo punto operativo specificando il tempo di simulazione
Ad esempio, è possibile definire un vettore di due elementi per ottenere due punti operativi all'istante **t=0** e **t=10**.

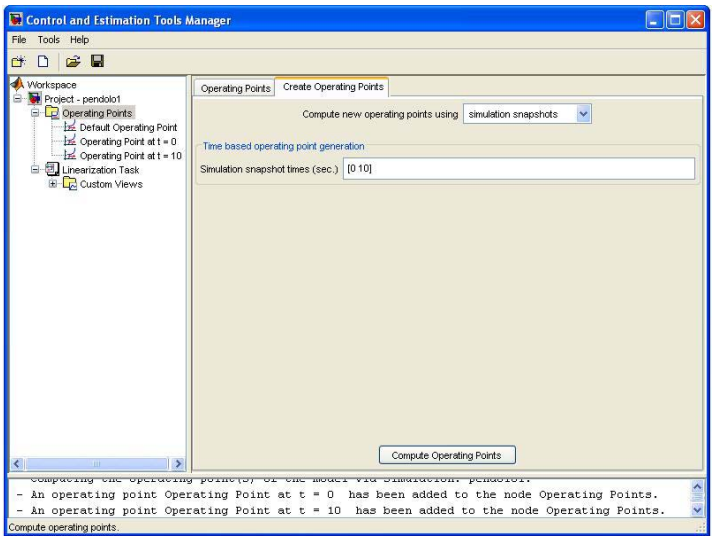


Figura 9.25 Aggiunta di due nuovi punti operativi specificati nel tempo

Dopo l'immissione del vettore all'interno dell'apposito box, per aggiornare lo schema Simulink con la definizione dei nuovi punti di lavoro, si clicchi in corrispondenza del tasto **Compute Operating Points**.
A conferma della validità dell'operazione effettuata, Simulink aggiungerà nella struttura ad albero del progetto di linearizzazione i due nuovi punti operativi.
Inoltre, entrando nella directory **Default Operating Point,** attraverso la finestra **States**, è possibile sia conoscere il punto operativo scelto da Simulink per effettuare la linearizzazione, sia effettuare dinamicamente modifiche alle variabili visualizzate.
Per rendere correnti le modifiche effettuate nella stessa finestra o assicurarsi che il modello sia aggiornato con ultime definizioni dei punti operativi, è opportuno ricordarsi di aggiornare il modello cliccando sul tasto **sync with Model**.

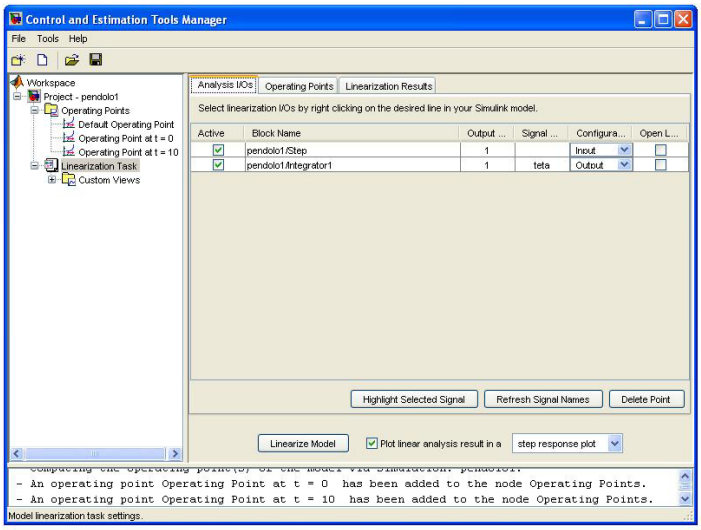


Figura 9.26 Finestra Linearization task

Per ottenere la linearizzazione del modello, si entri nella directory **Linearization Task**. Accedendo alla scheda **Operating Point,** ci si assicuri che il punto operativo selezionato sia proprio **Default Operating Point.**
Poi si ritorni alla scheda **Analysis I/Os** e si prema il tasto **Linearize model**.
Simulink, attraverso l'interfaccia LTI Viewer, provvede a plottare la risposta al gradino del sistema linearizzato, se non si è scelto un altro tipo di rappresentazione.
Infatti, nella stessa finestra, tramite l'apposito menu a tendina etichettato con il commento **Plot linear analysis result in a:**, è possibile scegliere tra i vari di rappresentazione per sistemi lineari.

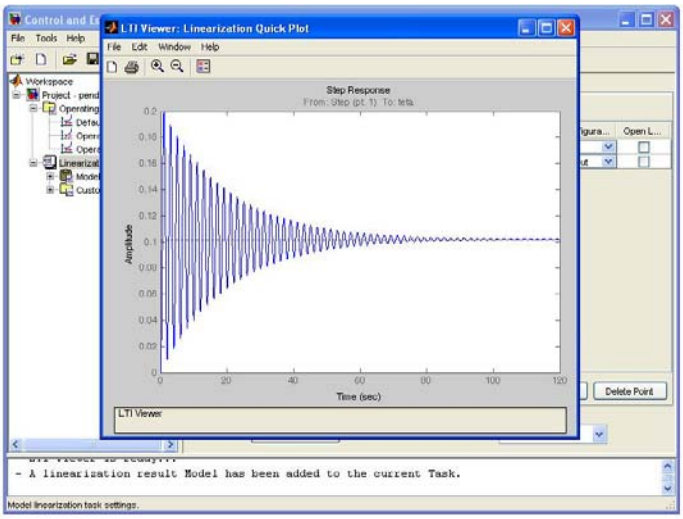


Figura 9.27 Risposta al gradino del pendolo linearizzato

È da notare che la trattazione fin qui fatta si riferisce alla linearizzazione di tutto uno schema Simulink complesso.
D'altra parte, Simulink permette di indicare un solo blocco da linearizzare. Ciò è possibile, cliccando con il tasto destro sul blocco di interesse presente all'interno dello schema Simulink e scegliendo l'opzione **linearize block** del menu contestuale.

Quindi, se si vogliono visualizzare i risultati della linearizzazione blocco per blocco secondo la rappresentazione ISU o in termini di funzione di trasferimento o di zeri-poli-guadagno, si acceda alla sottocartella **model** della directory **Linearization Task**.
La scelta della rappresentazione da utilizzare per i risultati della linearizzazione è effettuabile attraverso il box etichettato con il commento **Display linear result as:.**

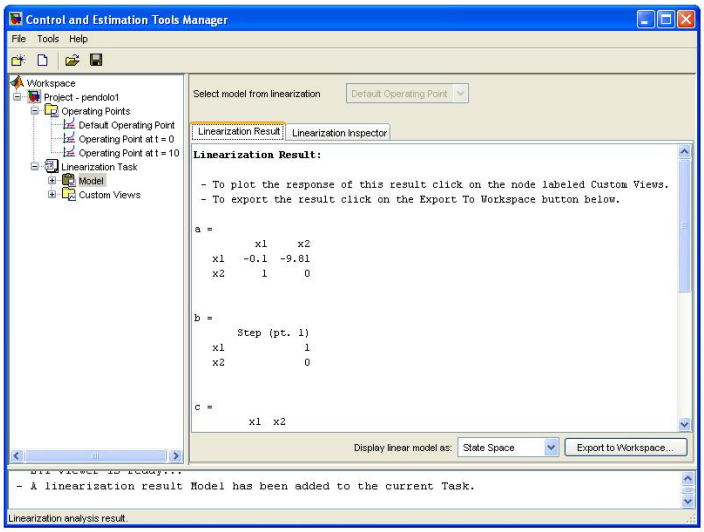


Figura 9.28 Rappresentazione ISU del modello linearizzato

Per esportare il modello linearizzato nel Workspace, si clicchi con il tasto destro sulla sottocartella **model** della directory **Linearization task** e si scelga l'opzione **Export** del menu contestuale.
Alternativamente, una volta attiva la finestra LTI Viewer, la stessa operazione può essere effettuata attraverso il menu **File→Export**.

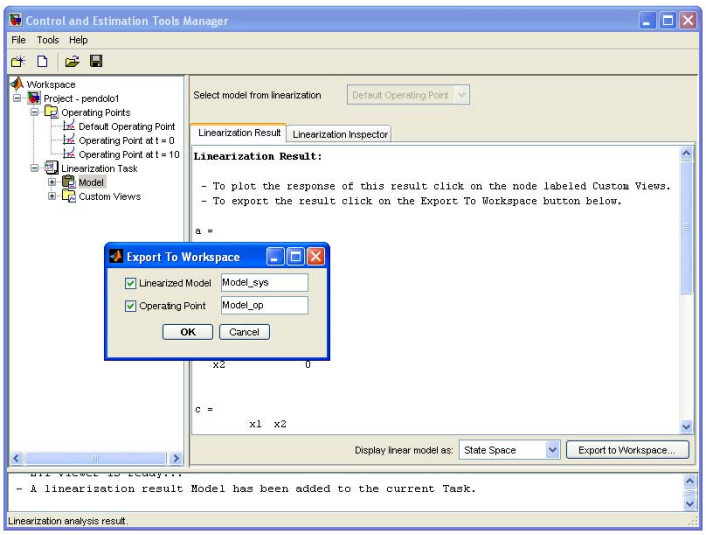


Figura 9.29 Esportazione dei risultati della linearizzazione

Infine, l'ultima sottodirectory **Custom View** della struttura ad albero dell'ambiente di linearizzazione di Simulink, afferisce ad una funzionalità che permette, attraverso l'utilizzo del tool LTI Viewer, di scegliere e di visualizzare i tipi di rappresentazione della risposta del modello linearizzato utili a comprendere il comportamento del sistema nella sua globalità.

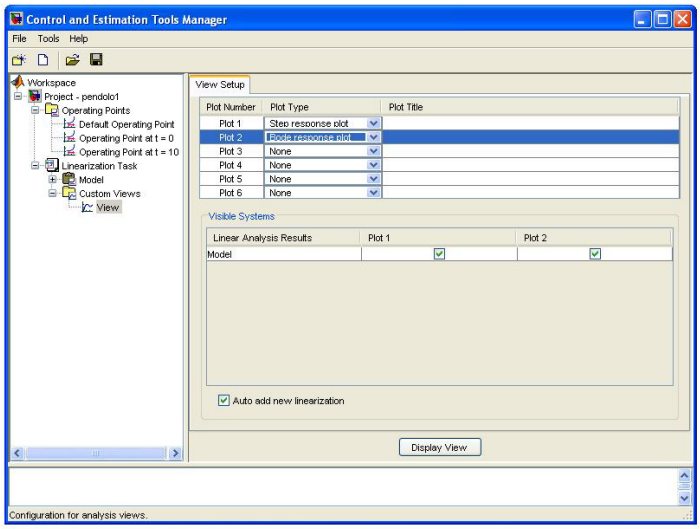


Figura 9.30 Finestra Custom View

La finestra grafica è suddivisa in due aree: la prima permette di scegliere i tipi di grafico con cui rappresentare la risposta del sistema; la seconda consente di assegnare una o più di queste rappresentazioni a ciascun modello di linearizzazione precedentemente creato.
Cliccando su **Display View** si ottengono le rappresentazione grafiche precedentemente definite.

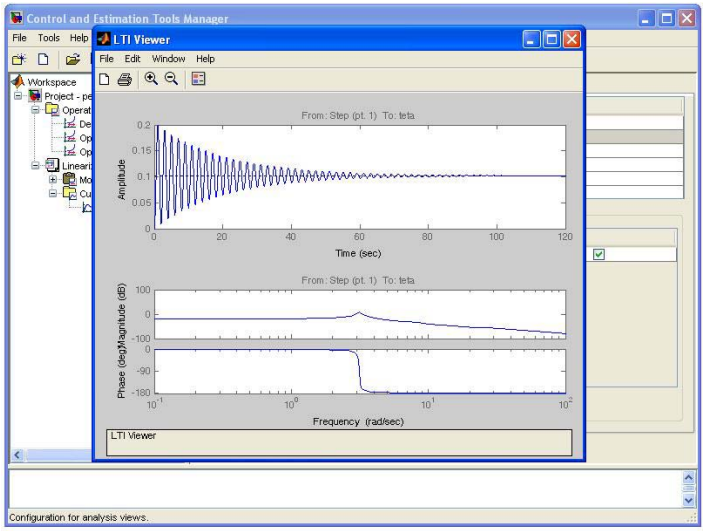


Figura 9.31 Risultati Display View