Agostino De Marco
Domenico P. Coiro

Elementi

di

Dinamica e simulazione di volo

Quaderno 4

Introduzione a Matlab e all'ambiente di simulazione Simulink

Marzo 2019
ver. 2019.a

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

Dichiarazione di Copyright

- Questo testo è fornito per uso personale degli studenti. Viene reso disponibile in forma preliminare, a supporto della preparazione dell'esame di *Dinamica e simulazione di volo*.
- Sono consentite la riproduzione e la circolazione in formato cartaceo o elettronico ad esclusivo uso scientifico, didattico o documentario, purché il documento non venga alterato in alcun modo sostanziale, ed in particolare mantenga le corrette indicazioni di data, paternità e fonte originale.
- Non è consentito l'impiego di detto materiale a scopi commerciali se non previo accordo.
- È gradita la segnalazione di errori o refusi.

Copyright 2010–2017 Agostino De Marco e Domenico P. Coiro,
Dipartimento di Ingegneria Industriale – Università degli Studi di Napoli Federico II.

(Legge italiana sul Copyright 22.04.1941 n. 633)

Quaderno 4

Introduzione a Matlab e all'ambiente di simulazione Simulink

Non ho paura dei computer, ma della loro eventuale mancanza.
– Isaac Asimov

Indice

Uno strumento di lavoro divenuto oramai uno standard *de facto*, utilizzato oggi nei campi più disparati delle scienze e dalla stragrande maggioranza degli studenti di ingegneria, è il software Matlab. In questa appendice si presenteranno sinteticamente le funzionalità principali di questa applicazione. Lo scopo è quello di raccogliere delle informazioni di base sufficienti ad illustrare le caratteristiche dell'*ambiente* Matlab, che permettano soprattutto a chi si accosta per la prima volta ad un simile strumento di iniziare ad impostare e risolvere problemi di calcolo.

4.1 Elementi di base

Matlab è un linguaggio di programmazione di alto livello nato per il calcolo tecnico-scientifico. Le funzionalità di Matlab in quanto applicazione rispecchiano la filosofia con

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

cui essa è stata progettata e sviluppata via via negli anni. L'ambiente di lavoro è strutturato in modo da risultare particolarmente utile nei seguenti campi applicativi: (*i*) Analisi numerica e calcolo, (*ii*) sviluppo di codice e *scripting*, (*iii*) modellistica, simulazione e *prototyping*, (*iv*) analisi di dati, esplorazione e visualizzazione, (*v*) disegno industriale e scientifico, (*vi*) sviluppo di applicazioni *stand-alone* corredate di interfaccia utente (*Graphical User Interface*, GUI).

Matlab, il cui nome deriva da *Matrix Laboratory*, inteso come ambiente di sviluppo, fornisce un linguaggio per il calcolo che ha come struttura di base la matrice. Gli scalari sono matrici 1×1, i vettori riga sono matrici $1 \times n$ ed i vettori colonna sono matrici $m \times 1$. Le due dimensioni principali di questi array vengono dedotte dal contesto delle istruzioni di assegnamento. In momenti successivi è sempre possibile un *reshaping*, cioè un adattamento di una data matrice $m \times n$ a delle nuove dimensioni $m' \times n'$ attraverso il riposizionamento, l'aggiunta o la sottrazione di elementi.

Esiste anche la possibilità, come accade per altri linguaggi di programmazione, di definire ed utilizzare array *multidimensionali*. Questi sono delle generalizzazioni delle matrici e costituiscono delle collezioni di dati accessibili mediante tre o più indici.

Matlab esegue tutti i calcoli numerici con numeri complessi in doppia precisione e la grande maggioranza delle operazioni e delle funzioni predefinite sono implementate in modo da lavorare nativamente su entità matriciali. Ciò è particolarmente vantaggioso nella risoluzione di molti problemi di calcolo dell'ingegneria, in particolare quelli formulati teoricamente nei testi con agevoli notazioni vettorali e matriciali. Tali problemi possono essere implementati in linguaggio Matlab e risolti efficientemente attraverso comandi semplici, con un minimo utilizzo di cicli. Ad esempio, per la soluzione dei tipici problemi dell'algebra lineare è possibile, con poche righe di codice e senza un appesantimento del carico di lavoro del programmatore, richiamare un numero di algoritmi ben collaudati basati sulle funzionalità di librerie di calcolo numerico come LINPACK, LAPACK o EISPACK (si veda `www.netlib.org`).

I codici di calcolo in linguaggio Matlab sono generalmente più snelli rispetto a quelli che sarebbe necessario sviluppare con un linguaggio scalare come il C o il Fortran. Anche se linguaggi di programmazione più evoluti come il C++ e il Java, con il paradigma di programmazione a oggetti, e come il moderno Fortran, con l'uso avanzato dei moduli, permettono ormai un'implementazione snella ed efficiente di una vasta categoria di algoritmi, la popolarità di Matlab, con il suo linguaggio intuitivamente più semplice da imparare e con la sua ben riuscita integrazione tra lo strumento di sviluppo codice, il debugger e l'output grafico, rende il 'laboratorio della matrice' uno strumento più attraente per chi si accosta per la prima volta al mondo della programmazione o per chi ha in mente tempi di sviluppo ridotti.

L'ambiente Matlab si è evoluto durante gli anni grazie al *feedback* ed al contributo dei molti utenti sparsi per il mondo. In ambienti universitari è ormai uno strumento didattico standard per corsi introduttivi e corsi avanzati, nella matematica, nell'ingegneria e nelle scienze.

4.2 Toolboxes e Simulink

L'ambiente possiede dei gruppi tematici di funzioni, denominati *toolboxes* (cassette degli attrezzi), pensati ed implementati *ad hoc* per classi particolari di problemi. Essi risultano

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

molto utili per la maggior parte degli utenti di Matlab e sono pensati per fornire loro degli strumenti di calcolo preconfezionati, efficienti e specializzati. I toolboxes sono collezioni complete di funzioni Matlab tra loro correlate, che estendono l'ambiente di lavoro e permettono di risolvere rapidamente particolari categorie di problemi. Le aree scientifiche per le quali sono disponibili dei toolboxes specifici sono numerose. Alcuni esempi sono: l'elaborazione dei segnali, i sistemi di controllo, le reti neurali, le *wavelets*, la simulazione in generale, il calcolo simbolico, la realtà virtuale.

Di particolare importanza è il pacchetto software Simulink, utilizzabile come modulo dell'ambiente Matlab, e concepito per la modellazione visuale e l'analisi di sistemi dinamici. Simulink supporta sia modelli lineari che non lineari, a tempo continuo e discreto. Si possono modellare sistemi a vettore di stato di dimensione qualsiasi e di tipo *multirate*, in cui diversi sottogruppi di variabili di stato sono aggiornate con frequenze differenti.

Come supporto alla modellazione l'ambiente Simulink fornisce un'interfaccia a finestre, che si avvia digitando il comando `simulink` nella finestra dei comandi di Matlab, figura 4.4, o attraverso l'icona di avvio nell'interfaccia principale. L'uso della GUI di Simulink permette la costruzione di uno schema a blocchi di un sistema dinamico costituito da blocchi funzionali tra di loro interconnessi. Il paradigma di lavoro è simile a quello che si adotterebbe nella fase di analisi preliminare di un sistema (*prototyping*) con l'uso di carta e penna. Simulink è dotato di un'ampia libreria di blocchi e di tipi di connessioni preconfezionati. L'utente può ovviamente personalizzare delle entità già presenti nelle librerie di default o crearne delle proprie. Si rimanda all'help di Matlab per approfondimenti sulle modalità di implementazione delle cosiddette *S-functions*.

Un pacchetto di funzioni Simulink di notevole utilità nel campo aerospaziale ed in particolare nella dinamica del volo è costituito dal toolbox *Aerospace Blockset* per l'analisi e l'integrazione di sistemi dinamici che modellano il funzionamento di velivoli di vario genere e di sistemi propulsivi.

4.3 Sessioni di lavoro

Di solito Matlab viene lanciato selezionando l'applicazione da un ambiente a finestre come Windows o fornendo il comando

```
$ matlab ⟨invio⟩
```

in una delle *shell* di comandi Unix/Linux. Matlab è a sua volta un programma con una riga dei comandi ed un suo prompt (>>). Appena lanciato compare per qualche istante, prima dell'avvio vero e proprio dell'applicazione e durante il caricamento di tutti i moduli software necessari al suo funzionamento, una finestra (*splash screen*) con il logo di Matlab, la versione ed il tipo di licenza. Se ciò accade significa che tutto è stato configurato correttamente, in caso contrario è necessario controllare la validità dell'installazione o della licenza.

Una volta avviata l'applicazione, il *layout* di default della finestra principale di Matlab, come si vede dalla figura 4.1, si presenta come l'insieme di più sotto-finestre: la finestra *Current Folder* per l'esplorazione dei file, la finestra *Command Window* con il *prompt* dei comandi, la finestra *Workspace* con le variabili definite al momento dell'ultimo comando ed una eventuale finestra *pop-up* con la storia dei comandi più recenti (attivata tramite tastiera premendo il tasto Freccia Su: ↑).

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

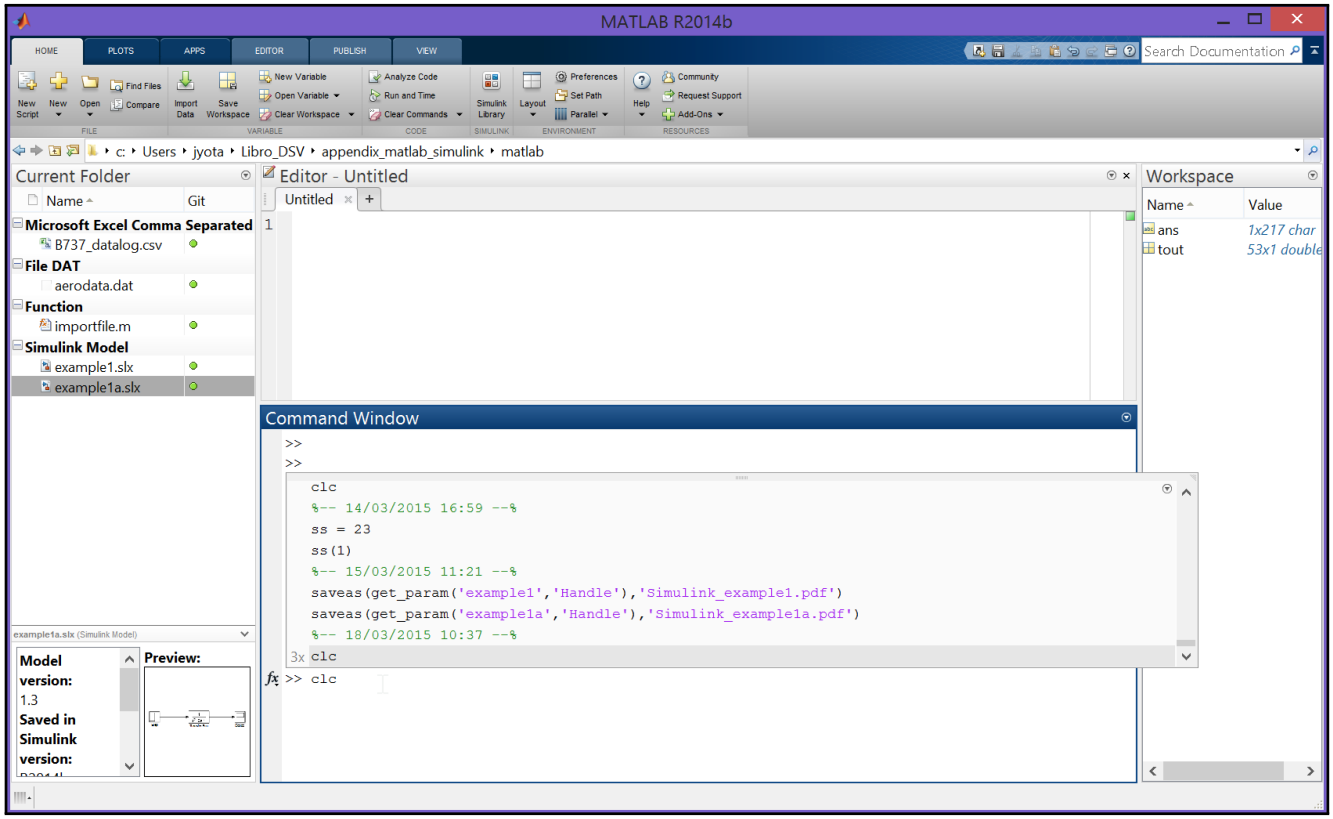


Figura 4.1 Il *layout* di default della finestra principale di Matlab (ver. 2014b). Il menu principale in alto è un ribbon che presenta un insieme di pannelli (sottomenu); nell'immagine è selezionato il menu Home con i suoi pulsanti. Sulla destra il workspace con le variabili definite al momento dell'ultimo comando. In basso il prompt dei comandi e la storia dei comandi più recenti.

Il menu principale di Matlab è un *ribbon* che presenta un insieme di *tabbed panels* (pannelli/sottomenu); ciascuno di essi contiene pulsanti e altri strumenti visuali per la gestione del flusso di lavoro.

La Command Window svolge anche il ruolo di finestra di *log* dei messaggi all'utente. In questo senso si presenta come estensione di una shell come la finestra DOS in Windows o della console in Unix/Linux. Per accedere ad uno dei comandi della *shell* del sistema operativo o anche lanciare un programma eseguibile esterno basta digitare il comando stesso preceduto dal carattere '`!`', detto *shell escape command*. Ad esempio per conoscere velocemente il contenuto della directory di lavoro corrente basta dare il comando

```
>> !dir ⟨invio⟩
```

in Windows oppure

```
>> !ls -l ⟨invio⟩
```

in Unix/Linux (o anche in Windows se è installato il software Cygwin).

Un numero di comandi di Matlab può essere collezionato in un file, di estensione "`.m`", ed interpretato dal *kernel*, che è il nucleo e motore numerico del programma. In gergo informatico tali files, detti *M-files*, sono degli *scripts* di comandi perché eseguiti in maniera sequenziale.

Un comando corrispondente al nome (estensione esclusa) di un M-file è noto all'ambiente di lavoro se il file si trova nel percorso di ricerca predefinito. La ricerca da parte dell'interprete dei comandi avviene in alcune sottocartelle della cartella in cui è instal-

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

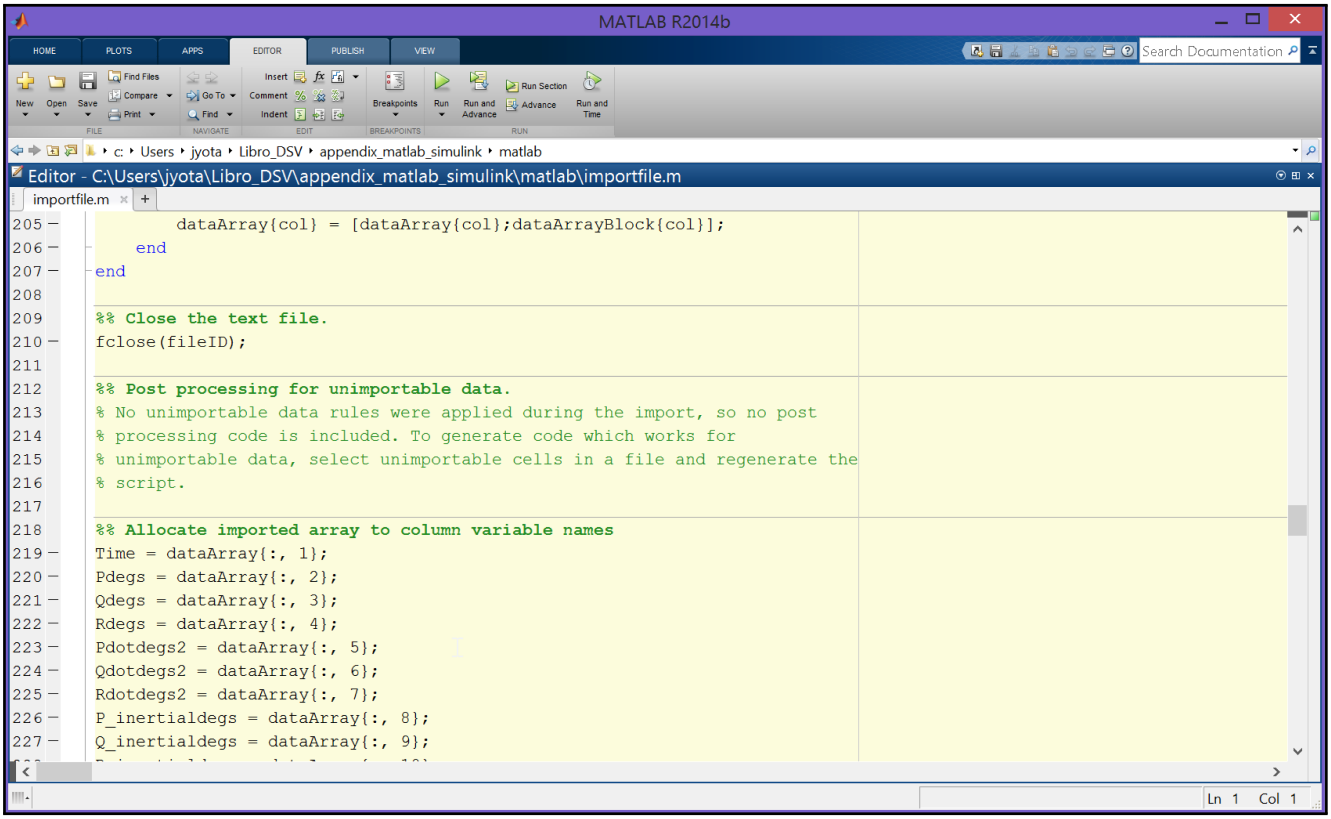


Figura 4.2 L'editor di M-files di Matlab in grado di aprire più di un file di script alla volta. Nel ribbon del menu principale è selezionato il pannello Editor con i tasti per l'avvio e la gestione del debug.

lato il programma e nella cartella di lavoro (*current directory*), che l'utente può definire arbitrariamente a seconda delle sue esigenze.

Oltre alla finestra principale con il prompt dei comandi, fanno parte dell'ambiente di lavoro: (*i*) l'*editor* degli script di comandi (*M-file editor*), si veda la figura 4.2, che si avvia dal menu File ▸ New ▸ M-file, e (*ii*) le finestre dei grafici che l'utente produce nella sua sessione di lavoro, ad esempio con il comando `plot`.

Se si digita `plot(0:0.1:1)` appare la finestra riportata nella figura 4.3. Essa contiene un grafico in cui le ordinate corrispondono ai numeri ottenuti a partire da `0` con incrementi di `0.1` fino ad arrivare a `1` e le ascisse corrispondono agli indici interi che vanno da `1` fino ad `11`. La sequenza di coppie di coordinate viene rappresentata per default unendo i punti con una linea continua. Come vedremo tra breve, l'operatore ":" consente di ottenere rapidamente un vettore di numeri.

4.4 Usare l'help

Se si vogliono avere informazioni su Matlab si può ovviamente ricorrere ai manuali in commercio o liberamente disponibili su internet. oppure utilizzare il suo sistema di help in linea, figura 4.5. L'*help* di Matlab dovrebbe essere la sorgente principale di informazioni sull'uso delle sue funzioni. Esso è relativo alla particolare versione in uso dell'applicazione (quindi non si corre il rischio di usare un comando obsoleto o deprecato dagli sviluppatori perché rimpiazzato da altre funzioni) ed è completo di tutto ciò che serve a farsi un'idea sull'uso di una data funzionalità (soprattutto di esempi).

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

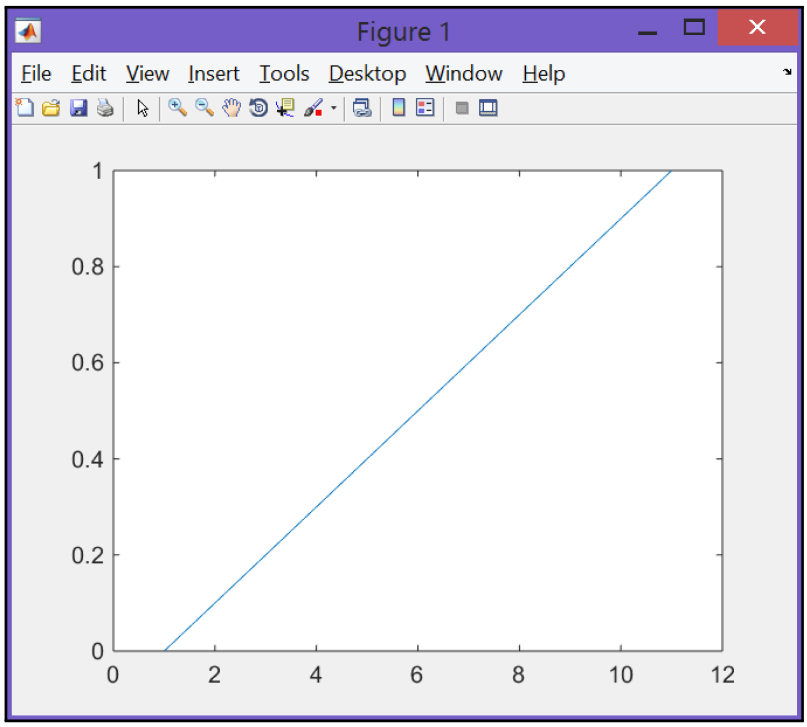


Figura 4.3 Una finestra grafica di Matlab ottenuta con il semplice comando `plot(0:0.1:1)`.

A volte se si vuole sapere rapidamente a cosa serve un comando, ad esempio il comando `plot`, si può scrivere

```
>> help plot ⟨invio⟩
```

nella finestra dei comandi. Comparirà una descrizione accurata di tale comando, sullo stile del comando `man` della shell di Linux. In generale, il comando

```
>> help ⟨nome comando⟩
```

restituisce (se il nome del comando esiste) l'help del comando in questione. Un'altro comando che talvolta si dimostra utile è

```
>> lookfor ⟨parola chiave⟩
```

che controlla se è disponibile l'help per la parola chiave indicata tra tutti i comandi disponibili in Matlab.

In alternativa alla richiesta di help da riga di comando, utile per una consultazione veloce ma a volte poco ricca di esempi, è possibile cercare aiuto nella preziosa *finestra di help a tendina*. Infatti, posizionando il cursore su una parola chiave nella finestra dell'editor on in quella dei comandi e premendo il tasto F1 comparirà una finestra *pop-up* con la descrizione del comando.

4.5 Nozioni e notazioni fondamentali

4.5.1 Le variabili

Come in tutti gli ambienti di programmazione interattivi in Matlab è possibile definire delle variabili, alle quali assegnare determinati valori, ed utilizzarle in calcoli successivi. I nomi delle variabili Matlab possono essere qualsiasi, ma non devono cominciare con un carattere che sia un numero o uno dei caratteri speciali (come quelli di punteggiatura).

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

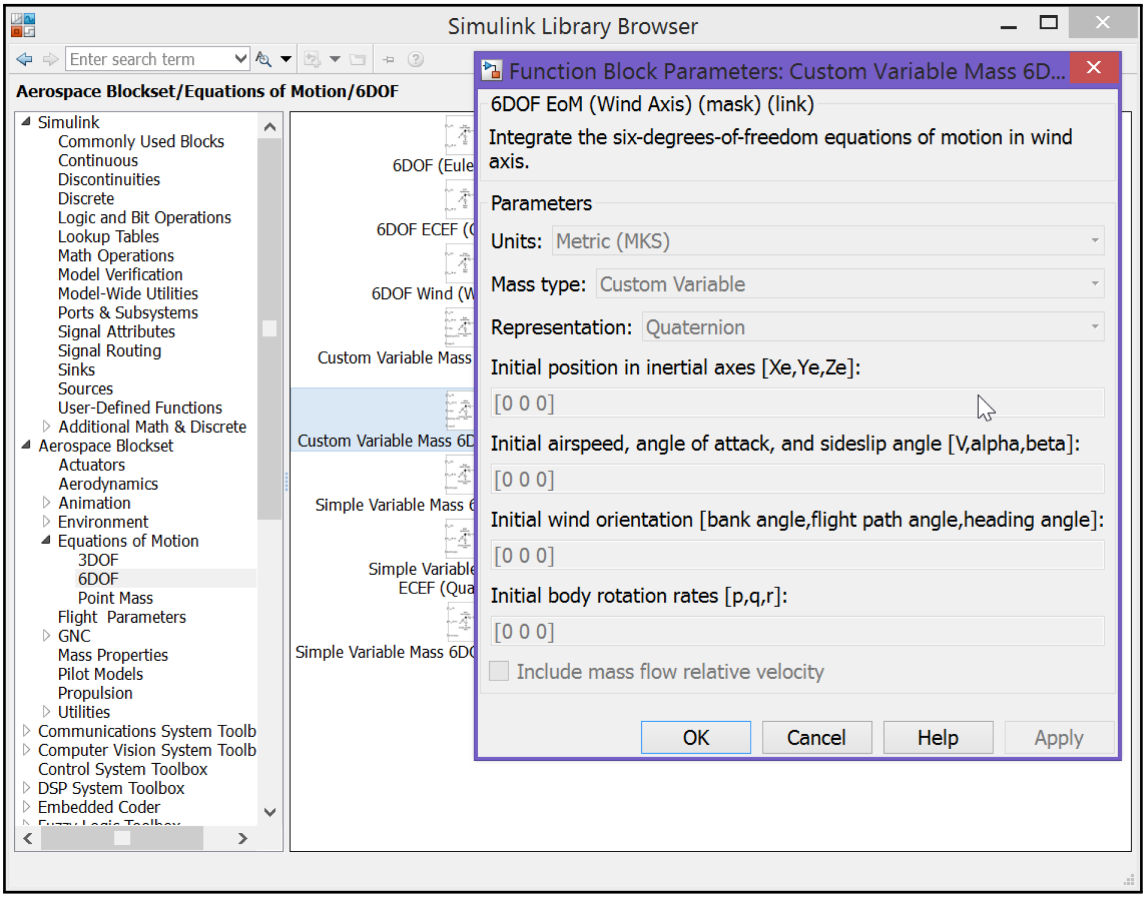


Figura 4.4 L'interfaccia grafica di Simulink per la modellazione di sistemi dinamici.

Se una variabile ha lo stesso nome di un comando Matlab, quest'ultimo non sarà più disponibile a meno che la variabile non venga cancellata con il comando

```
>> clear ⟨nome variabile⟩
```

Per assegnare una variabile si usa l'operatore di assegnamento "=". Il tipo della variabile viene automaticamente costruito in base alla quantità che si assegna.

Negli esempi seguenti si riportano delle sequenze di comandi digitati al prompt di Matlab, ma che potrebbero anche essere contenuti in un M-file. Le righe in cui il prompt ">>" non compare costituiscono ciò che invece la *command window* restituisce all'utente, detto in gergo *message log*. Ad esempio l'assegnazione di una data variabile non terminata dal carattere ";" viene sempre seguita dal messaggio che mostra all'utente il valore della variabile e le dimensioni della matrice. Ciò si verifica anche solo digitando il nome di una variabile ed è utile per conoscerne velocemente il valore o sapere se ne esiste una o se esiste una funzione con quel nome.

Maggiori dettagli sulla gestione delle variabili si evincono dagli esempi seguenti. Si osservi che il carattere "%" è quello che inizia un commento al codice Matlab.

Per cominciare si prenda in esame la sequenza di comandi:

```
>> a=1          % assegna alla variabile a il valore 1
a =
    1

>> b='pippo'    % assegna alla variabile b la stringa pippo
b =
    pippo

>> c=23;        % con il carattere ; a fine comando non viene mostrato
```

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

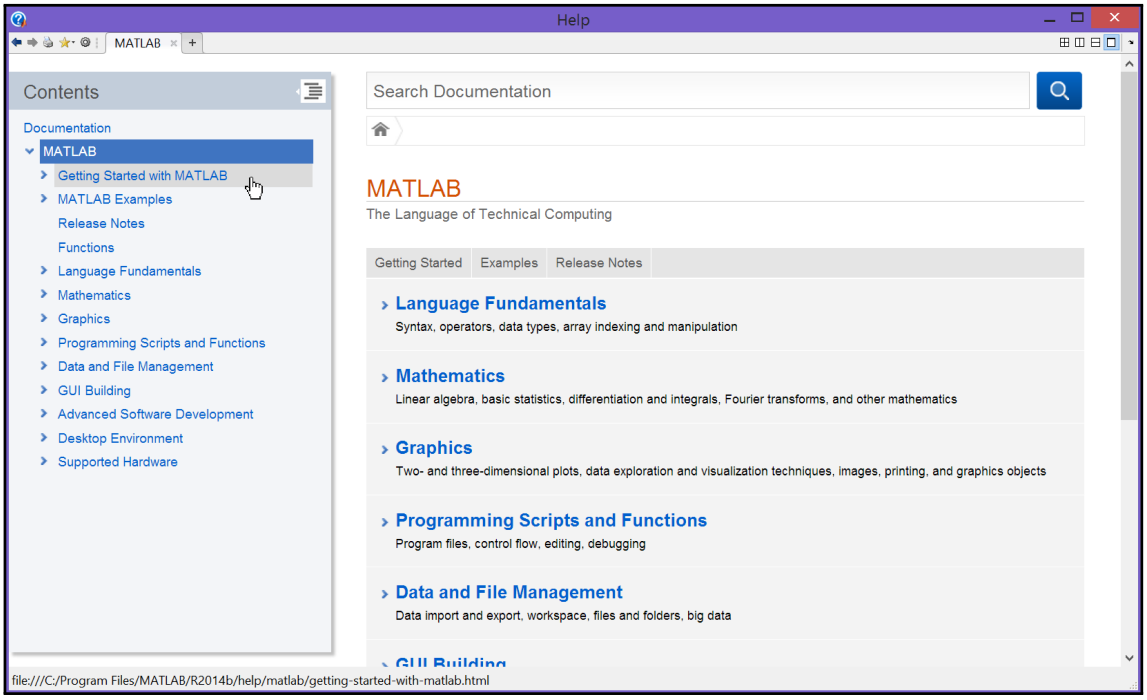


Figura 4.5 La finestra dell'help di Matlab.

```
                % il valore di c

>> 12.5         % 12.5, viene assegnato alla variabile di default ans(wer)
ans =
    12.5000

>> whos         % interroga il sistema sulle variabili dichiarate
    Name        Size        Bytes       Class

    a           1x1         8           double array
    ans         1x1         8           double array
    b           1x5         10          char array
    c           1x1         8           double array

Grand total is 8 elements using 34 bytes
```

Il significato dei comandi e delle operazioni riportate è spiegato dai commenti di cui sono corredati.

I numeri reali sono visualizzati con sole quattro cifre decimali; tuttavia, la rappresentazione interna contiene sempre 16 cifre decimali. Per cambiare il modo di visualizzare i numeri in Matlab, si può utilizzare il comando `format`. Ad esempio, se si danno i comandi:

```
>>format long; pi
ans =
   3.141592653589793
```

si ottiene la rappresentazione a 15 cifre decimali della variabile predefinita `pi` corrispondente a π. Diversamente con i comandi:

```
>>format short; pi
ans =
   3.1416
>>format short e; pi
ans =
   3.1416e+00
```

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

```
>>format long e; pi
ans =
   3.141592653589793e+00
```

si otterranno altre possibili rappresentazioni, rispettivamente, a 4 cifre decimali, scientifica a 4 cifre decimali, scientifica a 15 cifre decimali.

Gli esempi di comandi che seguono illustrano ulteriori aspetti dell'ambiente di programmazione interattivo.

```
>> a='pluto';   % se si riassegna una variabile questa viene riallocata
>> whos
    Name    Size    Bytes   Class

    a       1x5     10      char array
    ans     1x1     8       double array
    b       1x5     10      char array
    c       1x1     8       double array

>> clear a      % si cancella a

>> who          % le variabili dichiarate (senza tipo) ora sono
Your variables are:
    ans     b       c
Grand total is 12 elements using 36 bytes

>> clear all    % si cancellano tutte le variabili
>> who          % nessuna risposta, nessuna variabile definita
```

4.5.2 Matrici e vettori

Una matrice viene delimitata da parantesi quadre. Ogni riga termina con un punto e virgola e deve avere lo stesso numero di elementi (separati da spazi bianchi o da una virgola). Ad esempio, si vedano i comandi seguenti:

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

>> A=[-1, 2, 3;
       4, 5, 6] % il comando termina alla seconda riga con la ]
A =
   -1   2   3
    4   5   6

>> A=[-1 2 3; 4 ... % i 3 punti indicano continuazione su riga successiva
5 6];
```

Essi rappresentano modi equivalenti di introdurre la stessa matrice 2×3. Un vettore riga di dimensione n è una matrice $1 \times n$, un vettore colonna di dimensione m è una matrice $m \times 1$.

Matrici con dimensioni compatibili possono essere sommate o moltiplicate con gli operatori "+", "-" e "*". Si osservi che il carattere "'" (apice) che segue il nome di una variabile equivale all'operatore di trasposizione applicato alla matrice che tale variabile rappresenta. A tal riguardo si prendano in esame i comandi che seguono:

```
>> A=[1 2 3; 4 5 6]; B=[7 8 9; 10 11 12];
>> A+B      % somma di due matrici 2x3
ans =
    8       10      12
    14      16      18
>> A=[1 2 3; 4 5 6]; B=[7 8 9; 10 11 12]';
```

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

```
>> A*B % prodotto di una matrice 2x3 con una 3x2
ans =
    50     68
    122    167
```

Se le matrici che compaiono in un'operazione di prodotto non hanno dimensioni compatibili, viene segnalato un messaggio di errore. Ad esempio:

```
>> A=[1 2 3; 4 5 6]; B=[7 8 9; 10 11 12];
>> A*B      % prodotto di una matrice 2x3 con una 2x3
??? Error using ==> *
Inner matrix dimensions must agree.
```

Se invece si prova ad usare l'operatore "`.*`" al posto del solo "`*`" si ottiene:

```
>> A.*B
ans =
    7      16     27
    40     55     72
```

In questa circostanza Matlab non segnala alcun errore perché ora il prodotto è stato stato eseguito 'elemento per elemento' (*element-wise*). Facendo precedere il carattere "`.`" alle operazioni elementari queste vengono eseguite elemento per elemento. Questa è una proprietà importante ed utile, ad esempio, nella produzione di grafici di funzioni.

Quando si opera con matrici quadrate possono effettuarsi su di esse diverse operazioni particolari che restituiscono le seguenti quantità:

```
>> A=[1 2; 3 4];
>> inv(A)   % l'inversa di A
ans =
    -2.0000     1.0000
    1.5000     -0.5000
>> det(A)   % il determinante
ans =
    -2
>> eig(A) % gli autovalori
ans =
    -0.3723
    5.3723
>> cond(A) % il numero di condizionamento in norma 2
ans =
    14.9330
>> norm(A) % la norma 2
ans =
    5.4650
>> norm(A,inf) % la norma infinito
ans =
    7
```

A questo punto si sperimenti una prima operazione su un'intera matrice. Il comando `A+1` applicato alla matrice precedentemente definita fornisce:

```
>> A+1
ans =
    2   3
    4   5
```

cioè tutti gli elementi di `A` sono stati aumentati di 1. Quindi in questo senso si può sommare un numero ed una matrice.

Per accedere all'elemento `(i,j)` di una matrice `A`, basta scrivere `A(i,j)`, mentre per accedere all'elemento `i`-mo di un vettore `v` (riga o colonna che sia) basta scrivere `v(i)`. Per accedere ad un'intera colonna o ad un'intera riga gli indici corrispondenti possono essere rimpiazzati dai due punti "`:`" come negli esempi seguenti:

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

```
>> A(2,1)
ans =
    4

>> A(:,2) % estrae la seconda colonna
ans =
    3
    5

>> A(1,:) % estrae la prima riga
ans =
    2   3
```

4.5.3 Altri tipi di dati

Una variabile può contenere anche valori non numerici. Un esempio comune è dato dai caratteri e dalle stringhe di caratteri. Le stringhe letterali, come ad esempio `'pippo'`, vengono delimitate con degli apici, e possono comparire a destra di un'assegnazione come nell'esempio seguente:

```
>> str = 'Genny bello!'
str =
pippo
```

in cui si definisce una variabile `str` da intendersi come un vettore i cui elementi sono i singoli caratteri della stringa. Ad esempio il comando `str(5)` restituirà il carattere 'y'.

Matlab fornisce la possibilità di usare particolari tipi di variabili dette *cell arrays* utili a immagazzinare dati eventualmente disomogenei tra loro. Nella sintassi del linguaggio le variabili di tipo cell array sono costruite con una notazione basata sulle parentesi graffe. Un primo esempio è dato dall'assegnazione:

```
M = {a b c};
```

Un'assegnazione equivalente si ha mediante l'uso del costruttore `cell`. L'esempio precedente può riscriversi come:

```
>> M = cell(1,3);              % crea un cell array di valori nulli
M{1} = a; M{2} = b; M{3} = c;  % riempie le celle
```

Le matrici `a`, `b`, e `c` possono essere oggetti qualunque. Ad esempio è perfettamente valida l'istruzione:

```
M = {A, det(A), [det(A);det(B)]}:
```

È possibile inoltre definire delle strutture simili alle `struct` del linguaggio C, ai cui campi si accede con l'operatore "`.`" (punto). Un esempio di definizione di una variabile strutturata `S` è il seguente:

```
>> S.name = 'Pietro Savastano';
>> S.score = 83;
>> S
S =
     name: 'Pietro Savastano'
    score: 83
```

e naturalmente è possibile maneggiare interi array di strutture sfruttando i vantaggi offerti dal linguaggio.

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

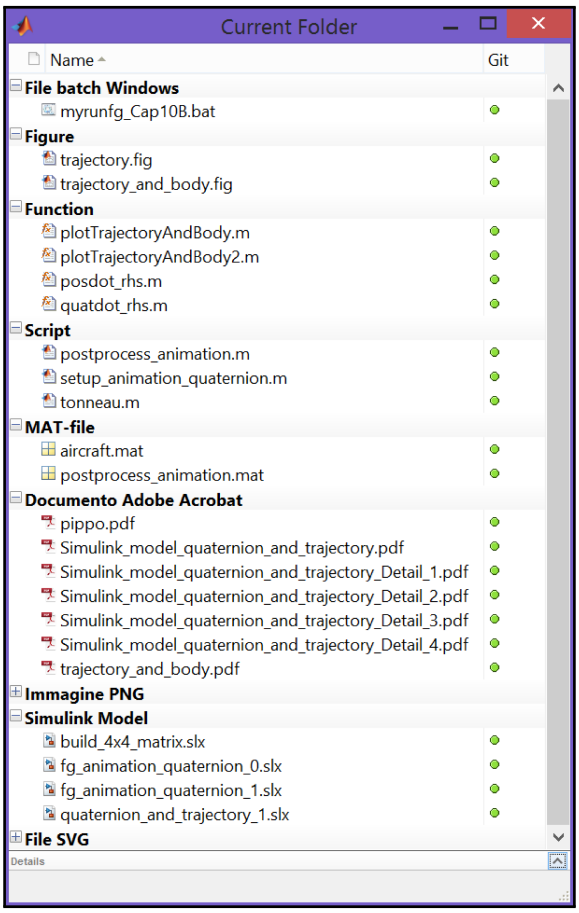


Figura 4.6 La finestra Current Folder per l'esplorazione della cartella di lavoro (mostrata nello stato *Undocked*).

4.5.4 Le funzioni in Matlab

Matlab, è sia un linguaggio di programmazione che un ambiente computazionale interattivo. Come già detto, i files che contengono comandi, cioè righe di codice Matlab, sono chiamati M-files. Dopo aver creato un M-file usando un qualsiasi editor di testo, salvandolo in un appropriato percorso l'utente può disporre di un nuovo comando personalizzato o di una nuova funzionalità, che arricchisce quelle predefinite dell'ambiente Matlab. In particolare esistono due tipi di M-file:

- gli *scripts*, cioè insiemi di comandi eseguiti sequenzialmente, che non hanno argomenti di input o argomenti di output ed operano su dati già presenti nel workspace e/o dati creati contestualmente,
- le *functions*, che possono accettare anche un numero variabile di argomenti di input (come le funzioni C++) ed hanno uno o più argomenti di output.

I programmatori in linguaggio Matlab creano in genere i propri M-files nella cartella di lavoro o in una delle sue sottocartelle. Eventualmente gli M-file potranno essere organizzati in *toolboxes* e salvati in cartelle che l'utente può aggiungere al percorso di ricerca predefinito di Matlab. Se si duplicano nomi di funzioni, cioè si hanno nomi di files identici in più punti del percorso di ricerca, Matlab esegue quello che trova per primo. Normalmente si può modificare il contenuto di un M-file navigando nella cartella di lavoro attraverso la finestra *Current Folder*, figura 4.6, ed aprendo con un doppio click lo script o la funzione desiderata. Si avvierà anche in questo caso l'M-file editor di Matlab.

Quando si richiama uno script dalla riga di comando, Matlab esegue semplicemente i comandi presenti nel file. Gli script possono operare su dati esistenti nel workspace, o possono essi stessi contenere comandi che allocano ed assegnano dati su cui operare.

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

Sebbene gli script non forniscano dati di output, qualsiasi variabile da essi creata rimane nel workspace, per essere usata in calcoli successivi. Inoltre, gli script possono produrre dei grafici, usando funzioni come `plot`.

Per esempio, se si crea un file chiamato `myrmatrix.m` contenente i seguenti comandi:

```
%------------------------------ file: myrmatrix.m
% Calcoli su matrici
r = zeros(1,30);
for k = 1:30
   r(k) = det(rand(7)); ❶
end
r ❷
bar(r) ❸
```

con il comando

```
>> myrmatrix ⟨invio⟩
```

si chiede a Matlab di eseguire le istruzioni contenute nel file una dopo l'altra: ❶ con un ciclo verrà calcolato il determinante di 30 matrici 7×7 i cui elementi verranno generati ad ogni passo in maniera random, ❷ successivamente verrà mostrato l'array `r` contenente i risultati, e infine ❸ ne verrà tracciato un grafico a barre. Dopo l'esecuzione del comando, la variabile `r` rimane nel workspace.

I comandi `eig`, `det`, `inv` e così via, sono esempi di funzioni Matlab. Esse possono essere richiamate in molti modi diversi. Le funzioni possono accettare argomenti in input e forniscono argomenti in uscita. La sintassi più generale per la chiamata di una funzione è la seguente:

```
>> [⟨out⟩1, ... , ⟨out⟩n] = ⟨nome funzione⟩(⟨in⟩1, ... , ⟨in⟩m);
```

dove $\langle in \rangle_1, \ldots, \langle in \rangle_m$ sono gli m parametri di input e $\langle out \rangle_1, \ldots, \langle out \rangle_n$ sono gli n parametri di output. Il numero di parametri di input e output può variare e per chiamare correttamente una funzione è necessario consultarne attentamente l'help (o il codice se disponibile). Se la funzione ha un solo parametro di output, le parentesi quadre possono essere omesse. Il nome di una funzione definita dall'utente deve coincidere con il nome dell'M-file e deve trovarsi nel percorso di ricerca corrente.

Le variabili definite all'interno di una funzione, dette variabili locali, hanno visibilità (*scope*) locale alla funzione quando non sono dichiarate con attributo `global`. È come se le variabili locali di una funzione fossero residenti in un workspace proprio, separato dal workspace principale di Matlab, che viene cancellato dopo la chiamata.

Un semplice esempio è fornito dalla funzione `rank`. L'M-file `rank.m` è disponibile nella cartella `<matlab root>/toolbox/matlab/matfun`. La funzione può essere chiamata dopo aver definito una matrice `A`, digitando al prompt dei comandi `>> rank(A)`. L'M-file in questione è qui di seguito riportato:

```
%------------------------------ file: rank.m
function r = rank(A,tol)
% RANK Matrix rank.
% RANK(A) provides an estimate of the number of linearly
% independent rows or columns of a matrix A.
% RANK(A,tol) is the number of singular values of A
% that are larger than tol.
% RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.
s = svd(A);
if nargin==1
   tol = max(size(A)) * max(s) * eps;
end
r = sum(s > tol);
```

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

La prima linea non commentata di un M-file contenente una funzione comincia con la parola chiave `function` seguita dalla lista degli argomenti di output, in questo caso la sola `r`, dal nome alla funzione, `rank`, e dalla lista di argomenti di input racchiusi tra parentesi tonde, `(A,tol)`. Le righe seguenti la definizione iniziano col carattere di commento "`%`" e rappresentano dei commenti di aiuto che verranno visualizzati al prompt dei comandi quando si digita

```
>>help rank ⟨invio⟩
```

La prima linea del testo di aiuto nel file `rank.m` è la cosiddetta "H1 line" che Matlab espone quando si ricerca aiuto da riga di comando. Il resto del file rappresenta il codice interpretato da Matlab per rendere disponibile la funzione agli utenti.

Nell'esempio appena riportato la variabile `s` presente nel corpo della funzione, così come le variabili che compaiono nella definizione, `r`, `A` e `tol`, sono locali alla funzione e rimangono indipendenti e separate da qualsiasi variabile nel workspace di Matlab. Anche se in quest'ultimo fosse definita una variabile con lo stesso nome non vi sarebbe possibilità di conflitto.

L'esempio illustra un aspetto delle funzioni di Matlab che si trova anche in altri linguaggi di programmazione come il C++, il Java o il Fortran cioè la possibilità di definire funzioni che possano essere chiamate con un numero variabile di argomenti. Il numero di argomenti effettivamente passato alla funzione nel momento della chiamata viene memorizzato di volta in volta nella variabile riservata `nargin`. Una funzione come `rank` può dunque essere usata in modi diversi come si vede dall'esempio seguente:

```
>> A = rand(10); % genera una matrice 10x10 ad elementi random
>> rank(A);
>> r = rank(A);
>> r = rank(A,1.e-6);
```

Rispetto alla gestione del numero di parametri di input e di output, molte funzioni che fanno parte del linguaggio Matlab sono definite in maniera simile. Se nessun argomento di output compare nella chiamata, il risultato è assegnato alla variabile di default `ans`. Tipicamente il primo argomento di input è sempre richiesto e non può essere omesso nella chiamata. Se il secondo argomento di input non è presente, la funzione lo pone uguale ad un valore di default. Tipicamente nel corpo di una funzione sono interrogate le due variabili riservate `nargin` e `nargout`, che contengono il numero di argomenti di input e di output effettivamente utilizzati nelle chiamate. L'effettivo comportamento della funzione ed eventualmente l'incorrere in una condizione di errore di chiamata dipenderà di volta in volta dal valore di queste due variabili.

Per approfondimenti sulla programmazione di funzioni con un numero variabile di argomenti di input e di variabili di output si vedano anche nell'help le voci `varargin` e `varargout`.

4.6 Operazioni di Input/Output

L'ambiente di lavoro offre la possibilità di analizzare dati raccolti eventualmente con altri programmi ed immagazzinati in file di testo (formato ASCII). Ad esempio, si potrebbe avere a disposizione l'insieme di dati seguente:

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

```
 1  2.000     -5.0
 2  0.2500    -9.1
 3  0.0740   -23.0
 4  0.0310   -53.2
 5  0.0160  -105.1
 6  0.0090  -185.5
 7  0.0050  -299.4
 8  0.0030  -453.7
 9  0.0020  -653.0
10  0.0020  -905.3
```

Questi dati possono essere salvati in un file di testo, per esempio `aerodata.dat`, e caricati in Matlab con il comando `load`. In questo esempio, nel workspace di Matlab viene creata una matrice `aerodata` di dimensioni 10×3:

```
>> load aerodata.dat
>> whos

    Name        Size        Bytes       Class
    aerodata    10x3        30          double array

Grand total is 30 elements using 240 bytes
```

Con il comando `save` è possibile salvare le variabili presenti nel workspace in uno o più file. Esistono diversi formati di salvataggio possibili, selezionabili attraverso una opportuna opzione. Un file di dati come `aerodata.dat` potrebbe essere stato prodotto a valle di una sessione di lavoro con la sequenza di comandi:

```
>> id = 1:10;
>> x  = [2.000 0.2500 0.0740 0.0310 0.0160 ...
         0.0090 0.0050 0.0030 0.0020 0.0020];
>> y  = [-5.0 -9.1 -23.0 -53.2 -105.1 ...
         -185.5 -299.4 -453.7 -653.0 -905.3];
>> dati = [id' x' y'];
>> save aerodata.dat dati -ascii  % salva in forma ascii 8 bit
```

I possibili usi del comando di salvataggio sono i seguenti:

```
>> save aerodata.dat dati              % salva in forma binaria
>> save aerodata.dat x y               % salva le matrici riga x e y
>> save aerodata.dat dati -ascii       % salva in forma ascii 8 bit
>> save aerodata.dat dati -double      % salva in forma ascii 16 bit
>> save aerodata.dat dati -ascii -tabs % salva in forma ascii
                                       % con tabulatori
```

Se il nome del file è `stdio` l'output è inviato allo *standard output device*, generalmente lo schermo, che in Matlab è la finestra dei comandi.

Esistono istruzioni di input/output alternative e di più basso livello rispetto a `load` e `save`. I comandi `dlmread` e `dlmwrite` permettono all'utente di lavorare con file di testo nei quali i dati sono organizzati per colonne e i valori nelle singole righe sono separati da un carattere predefinito (il *delimiter*). Ad esempio, non è raro dover manipolare file contenenti dati nel formato *comma separated values* (CSV), valori separati da virgole, tipicamente di estensione "`.csv`". Se si ha un file `dati.csv` in formato CSV, il suo contenuto potrà essere letto ed immagazzinato in una matrice con il comando:

```
>> A = dlmread('dati.csv',',') % legge il file delimitato da ','
```

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

Utilizzatori esperti possono a volte preferire metodi di lettura e scrittura basati su una collezione di comandi Matlab simili a quelli del linguaggio C. Questi comandi sono riportati nella tabella 4.1 e si rimanda all'help per approfondimenti ed esempi pratici sull'argomento.

Tabella 4.1 Comandi di Input/Output di basso livello. Si consulti l'help di Matlab per maggiori dettagli sulla sintassi e per degli esempi d'uso.

`fopen`, `fclose`	apertura e chiusura di un file
`fread`, `fwrite`	I/O per dati non formattati
`fscanf`, `fprintf`	I/O per dati formattati (formattazione con i comandi simili al C)
`fgetl`, `fgets`	I/O dati formattati
`ferror`, `fseek`, `ftell`, `frewind`	Come i corrispondenti comandi in C
`sprintf`, `sscanf`	Conversione di stringhe

Qui si riporta un semplice esempio in cui si crea un file di testo `valori.txt` con una prima riga di intestazione e due colonne di valori, una contenente quelli della variabile indipendente t ed una contenente quelli della funzione $y(t) = 2\sin(t)\cos(t)$:

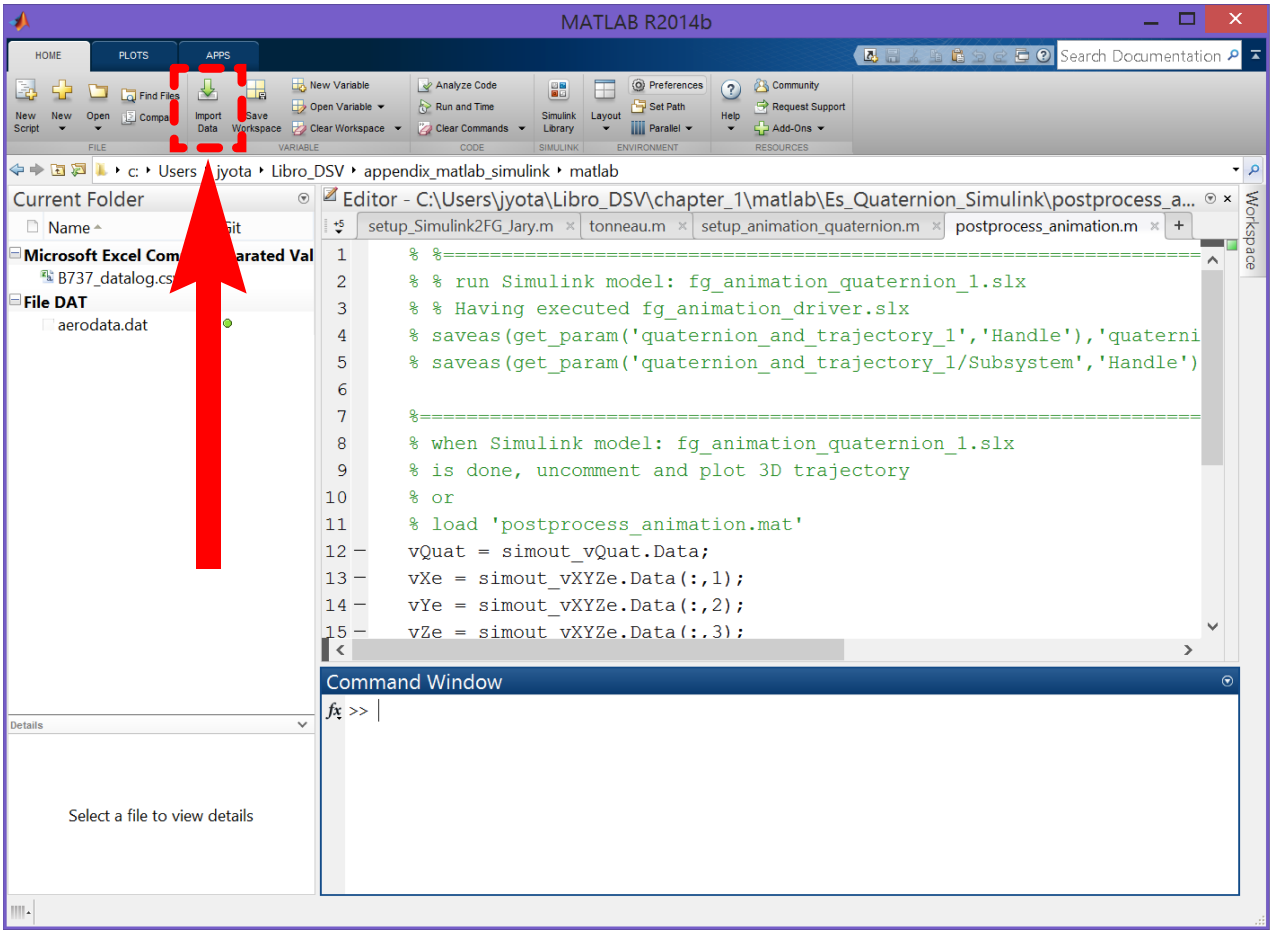
```
% Genera i dati
t = linspace(0,2*pi,40); % ← 40 punti in [0,2π]
y = 2*sin(t).*cos(t); % ← genera y(t) = 2 sin(t) cos(t)
data = [t ; y]; % matrice 2x40
fid = fopen('valori.txt','w'); % apre il canale del file
                               % in scrittura
fprintf(fid,'Valori della funzione sin(t)*cos(t) tra 0 e 2*pi\n\n');
fprintf(fid,'%4.2f %10.6f\n',data); % tipica formattazione C
% N.B.: questa formattazione permette di ottenere
%       un file con i dati su due colonne
fclose(fid) % chiude il canale del file
```

Noto il metodo ed il formato con cui i dati sono stati scritti nel file `valori.txt`, sarà possibile rileggerne il contenuto con comandi analoghi e con l'uso di `fscanf`.

Per finire si ricorda che Matlab dispone di uno strumento visuale per l'importazione dei dati al quale si accede dal menu principale (Home) attraverso l'icona *Import data*, figura 4.7 a fronte. Si tratta di un'interfaccia grafica che permette di selezionare il file che contiene i dati, di analizzarne il contenuto, selezionare il carattere di delimitazione dei valori nelle righe, escludere eventualmente un certo numero di righe iniziali ed infine di stabilire in quante e quali variabili caricare i dati nel workspace.

La figura 4.7 mostra un esempio d'importazione di file. Nell'esempio si è scelto di importare il file dal nome `B737_datalog.csv`. La funzionalità d'importazione ha riconosciuto automaticamente un insieme di 163 colonne di valori numerici separati da virgole e la presenza di una riga testuale iniziale. Attraverso gli aiuti visuali della finestra di dialogo (pulsanti e menu a tendina) è possibile effettuare ulteriori configurazioni della modalità d'importazione. Ad esempio è possibile importare le colonne di numeri in un'unica matrice oppure in matrici colonna separate. Inoltre, è possibile generare il codice di una funzione Matlab con cui riprodurre in maniera programmatica l'importazione effettuate

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro



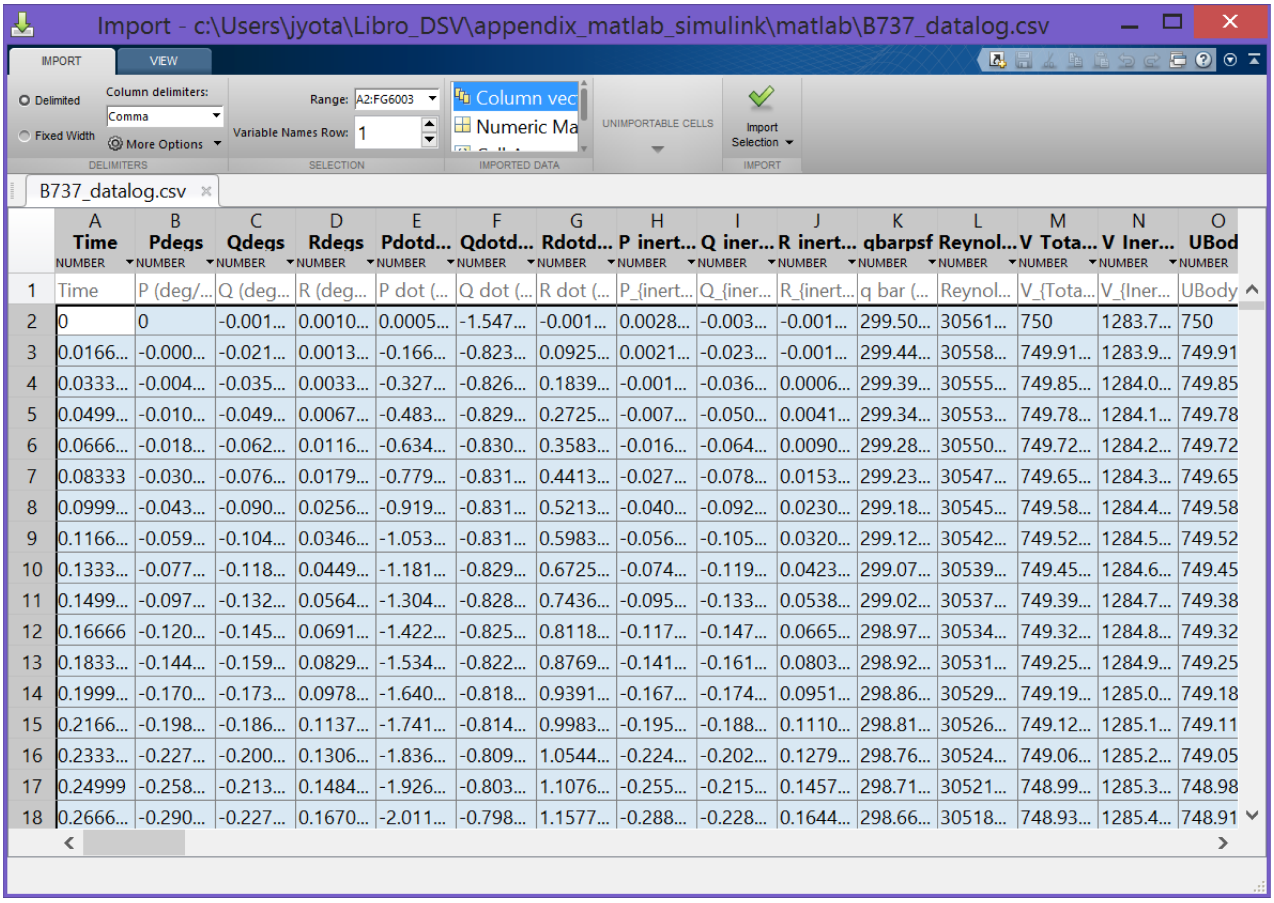


Figura 4.7 In alto, l'icona di avvio della finestra di dialogo *Import data* nel menu principale di Matlab (Home). In basso, la finestra di dialogo dopo aver importato i dati contenuti nel file `B737_datalog.csv`.

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

la prima volta attraverso la finestra *Import data*. Il listato seguente è un adattamento del codice generato automaticamente dallo strumento per l'importazione guidata.

```
function [Time,Pdegs,Qdegs,Rdegs] = importfile(filename, startRow, endRow)
%IMPORTFILE Import numeric data from a text file as column vectors.
%   [Time,Pdegs,Qdegs,Rdegs] = IMPORTFILE(FILENAME) Reads data from
%   text file FILENAME for the default selection.
%
%   [TIME,PDEGS,QDEGS,RDEGS] = IMPORTFILE(FILENAME, STARTROW, ENDROW)
%      Reads data from rows STARTROW through ENDROW of text file FILENAME.
%
% Example:
%   [Time,Pdegs,Qdegs,Rdegs] = importfile('B737_datalog.csv',2, 6003);
%
%   See also TEXTSCAN.

% Auto-generated by MATLAB on 201X/YY/ZZ 17:01:33

%% Initialize variables.
delimiter = ',';
if nargin<=2
    startRow = 2;
    endRow = inf;
end

%% Format string for each line of text:
% 163 columns of double values + carriage return
formatSpec = '%f';
for k = 1:162
   formatSpec = strcat(formatSpec,'%f');
end
formatSpec = strcat(formatSpec,'%[^\n\r]');

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan( ...
               fileID, formatSpec, endRow(1)-startRow(1)+1, ...
               'Delimiter', delimiter, 'EmptyValue', NaN, ...
               'HeaderLines', startRow(1)-1, ...
               'ReturnOnError', false ...
               );
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan( ...
               fileID, formatSpec, endRow(block)-startRow(block)+1, ...
               'Delimiter', delimiter, 'EmptyValue', NaN, ...
               'HeaderLines', startRow(block)-1, ...
               'ReturnOnError', false ...
               );
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
```

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

```
%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Allocate imported array to column variable names
Time = dataArray{:, 1};
Pdegs = dataArray{:, 2};
Qdegs = dataArray{:, 3};
Rdegs = dataArray{:, 4};

% TODO: modify here if you want to return more column vectors
%       taken from dataArray columns
end
```

La funzione `importfile` potrà essere riutilizzata in successive occasioni senza dover passare attraverso l'interfaccia grafica. Eventualmente la funzione generata automaticamente potrà essere rinominata a seconda delle esigenze specifiche dell'utente, ad esempio come `importMyCsvFile`, e salvata in un file opportunamente nominato `importMyCsvFile.m`. L'importazione eseguita attraverso questo metodo richiederà la chiamata:

```
[time,p_degs,q_degs,r_degs] = ...
    importMyCsvFile('B737_datalog.csv', 2, 1001);
```

che produrrà l'allocazione di quattro variabili `time`, `p_degs`, `q_degs` ed `r_degs` corrispondenti a quattro vettori colonna.

4.7 L'ambiente di simulazione Simulink

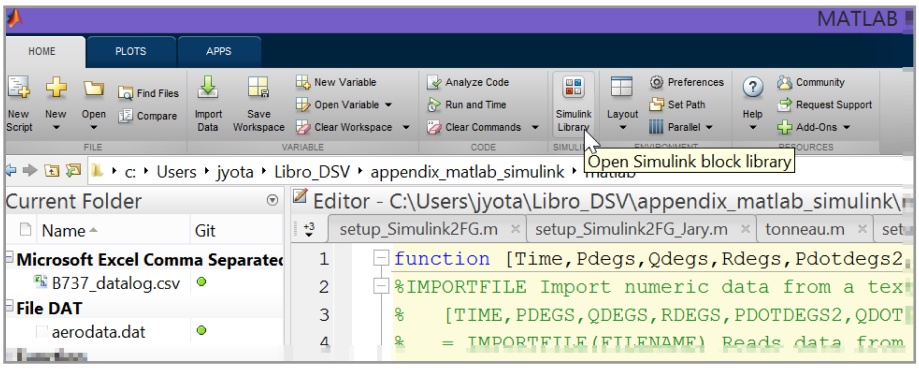
Simulink è un ambiente di lavoro visuale che si presenta come un'estensione grafica di Matlab per la modellazione e la simulazione di sistemi dinamici. In Simulink è possibile costruire il modello di un sistema nella forma di diagramma a blocchi. Concettualmente il sistema stesso è immaginabile come un unico blocco, che riceve segnali in ingresso e ne fornisce altri in uscita. Ogni sistema è scomponibile in un certo numero di sottosistemi, ciascuno schematizzabile a sua volta come un blocco. Mediante gli strumenti visuali di Simulink è possibile costruire modelli anche molto complessi con uno sforzo da parte dell'utente che si limita al tracciamento, su un foglio di lavoro elettronico, di uno schema a blocchi rappresentativo del sistema in esame.

Simulink mette a disposizione una collezione di blocchi predefiniti, configurabili attraverso delle opportune maschere di input, che implementano le funzionalità più utilizzate nei vari campi dell'ingeneria. Esempi molto comuni sono il blocco della funzione di trasferimento o il blocco giunzione (somma algebrica) di due o più segnali. Sono disponibili inoltre dei blocchi di input e output visivo, come ad esempio i generatori di segnale (*Signal Builder*) e gli oscilloscopi (*Scope*).

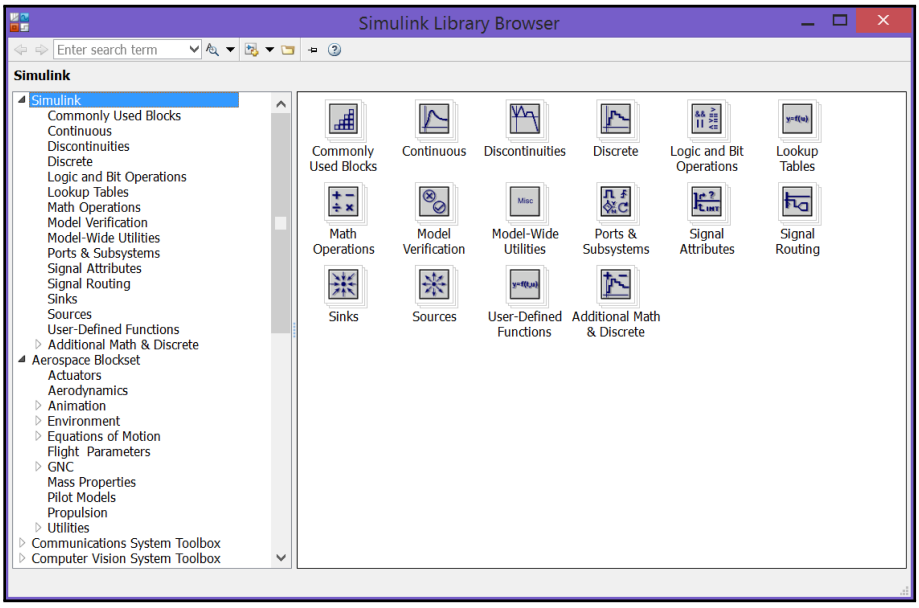
Simulink è integrato in Matlab e i dati possono essere facilmente trasferiti dal workspace di Matlab alla porzione di memoria gestita da Simulink e viceversa.

Simulink può essere lanciato con il comando Matlab:

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro



(a) Pannello Home del menu principale e tasto di avvio di Simulink.



(b) Il Simulink Library Browser.

Figura 4.8 Lancio di Simulink e libreria dei blocchi.

```
>> simulink ⟨invio⟩
```

In alternativa Simulink viene lanciato con l'apposito pulsante, figura 4.8a. L'esecuzione di Simulink comporta l'apertura della finestra mostrata nella figura 4.8b detta *Simulink Library Browser*. Nella libreria è presente un gran numero di blocchi già pronti per l'uso.

4.7.1 Elementi di base

In Simulink un modello è un insieme di blocchi tra loro interconnessi che schematizzano un sistema dinamico. Il modello è composto da due classi di elementi: i blocchi e le linee di collegamento. I blocchi sono usati per generare, modificare, combinare e mostrare i segnali. Le linee sono usate per trasferire i segnali da un blocco all'altro.

Blocchi

La libreria di Simulink dispone di molti *Blockset*, insiemi di blocchi raggruppati per ambito disciplinare (Aerospaziale, Comunicazioni, Controlli, eccetera). Oltre a questi esiste la

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

collezione di blocchi di base di Simulink (figura 4.8b), che si dividono in varie categorie tra cui:

- *Sources*: usati per generare vari tipi di segnale (non hanno ingresso).
- *Sinks*: usati per creare l'output o mostrare il segnale (non hanno uscita).
- *Discrete*: elementi di un sistema lineare e a tempo discreto (funzioni di trasferimento, spazio degli stati, eccetera).
- *Continuous*: elementi di un sistema lineare e a tempo continuo.
- *Math operation*: operazioni matematiche.
- *Signal routing*: somma di segnali, differenza e varie combinazioni.

Linee di collegamento

Le linee sono orientate graficamente per mezzo di una freccia che rappresenta il verso di trasmissione dei segnali. Le linee devono trasmettere sempre dal punto di output di un blocco al punto di input di un altro blocco. L'unica eccezione è una linea che nasce da un'altra: in tal caso l'utente può ramificare il segnale originale trasportandolo in parallelo lungo linee diverse. Le linee non possono mai immettere il segnale in altre linee; per combinarle è necessario usare un blocco appropriato (della categoria *Signal routing*).

I segnali possono essere sia scalari che vettoriali, cioè le linee di connessione tra i blocchi possono trasportare sia grandezzze scalari che vettoriali. Per sistemi SISO (*Single Input–Single Output*) si usano in genere segnali scalari. Per i sistemi MIMO (*Multiple Input–Multiple Output*) si usano i segnali vettoriali. Le linee usate per trasmettere segnali scalari o vettoriali sono identiche, il tipo di segnale che passa in esse è determinato dalle caratteristiche dei blocchi posti alle estremità.

4.7.2 Esempio di modello

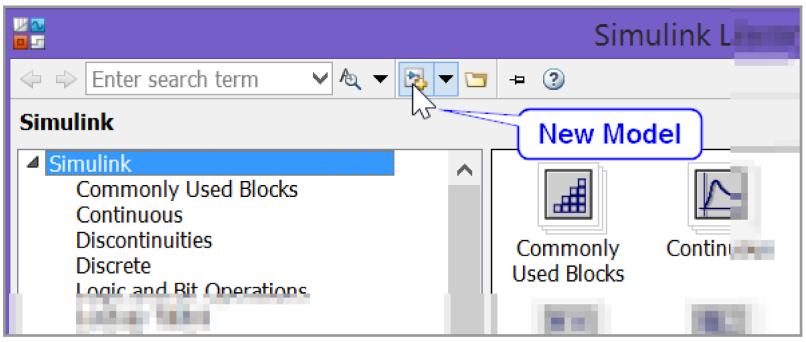
Per cominciare a lavorare in Simulink è necessario creare un progetto vuoto destinato a contenere i blocchi che serviranno via via a confezionare il modello di sistema dinamico desiderato. Un nuovo modello viene creato facilmente con l'apposito tasto mostrato nella figura 4.9a nella pagina seguente. La figura 4.9b mostra la finestra del nuovo progetto che l'utente dovrà popolare opportunamente con blocchi e linee.

Per inserire un blocco predefinito basta selezionarne l'icona nella libreria e trascinarla nella finestra del nuovo modello. Nella figura 4.10 a pagina 25 è rappresentato un semplice esempio di modello Simulink composto da tre blocchi etichettati come: "Step", "Transfer Fcn" e "Scope". Il blocco Step appartiene alla categoria *Sources* e genera un segnale a gradino. Il blocco Transfer Fcn appartiene alla categoria *Continuous* e rappresenta una funzione di trasferimento. Il blocco Scope appartiene alla categoria *Sinks* e permette di rappresentare visivamente la storia temporale di un segnale come accade per un oscilloscopio. Ciascuno di questi blocchi è stato posizionato nel progetto vuoto come nella figura 4.10 e le linee di connessione sono state predisposte trascinando con il mouse il punto d'uscita di un blocco nel punto d'ingresso del blocco adiacente.

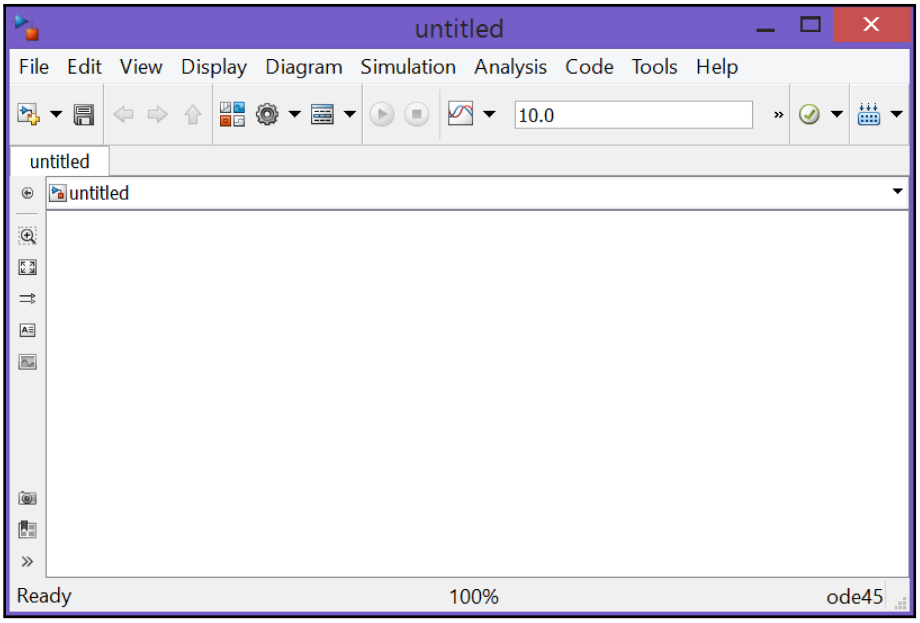
Modificare i blocchi

Tutti i blocchi possono essere opportunamente configurati e le loro caratteristiche di funzionamento modificate a seconda delle esigenze dell'utente. La configurazione viene

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro



(a) Tasto di creazione di un nuovo modello nel Simulink Library Browser.



(b) Finestra di un nuovo modello Simulink.

Figura 4.9 Creazione di un nuovo modello in Simulink.

avviata facendo doppio click con il mouse sull'icona del blocco all'interno del progetto. Ad esempio, con un doppio click sul blocco Transfer Fcn nel modello della figura 4.10 si apre la finestra mostrata nella figura 4.11. Con questa maschera di configurazione l'utente può modificare i polinomi al numeratore e al denominatore della funzione di trasferimento, inserendo i corrispondenti vettori dei coefficienti. Se, ad esempio, si vuole inserire come denominatore il polinomio $(s^2 + 2s + 1)$ sarà necessario inserire il vettore riga `[1 2 1]` all'interno della casella di testo dal nome "Denominator coefficients". Chiudendo la finestra di dialogo il modello apparirà come nella figura 4.12 a pagina 26.

Anche il blocco Step è configurabile. Nella maschera di configurazione di questo blocco, figura 4.13 a pagina 26, l'utente può modificare il tempo iniziale del gradino, il livello di partenza e il livello finale. I valori preimpostati forniscono un segnale scalare a gradino unitario, il cui valore nullo è assunto dal tempo $t_0 = 0\,\mathrm{s}$ fino al tempo $t_1 = 1\,\mathrm{s}$. A partire dal tempo t_1 fino al tempo finale della simulazione (impostato dall'utente nel menu del progetto: Simulation ▸ Model Configuration Parameters) il segnale assumerà valore 1. Questi parametri possono essere tutti modificati a seconda delle esigenze dell'utente.

L'ultimo blocco dell'esempio iniziale è il blocco Scope. Un doppio click sull'icona del blocco apre una finestra come quella della figura 4.14 a pagina 27. Questa finestra è destinata a mostrare la storia temporale del segnale proveniente dal blocco Transfer Fcn.

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

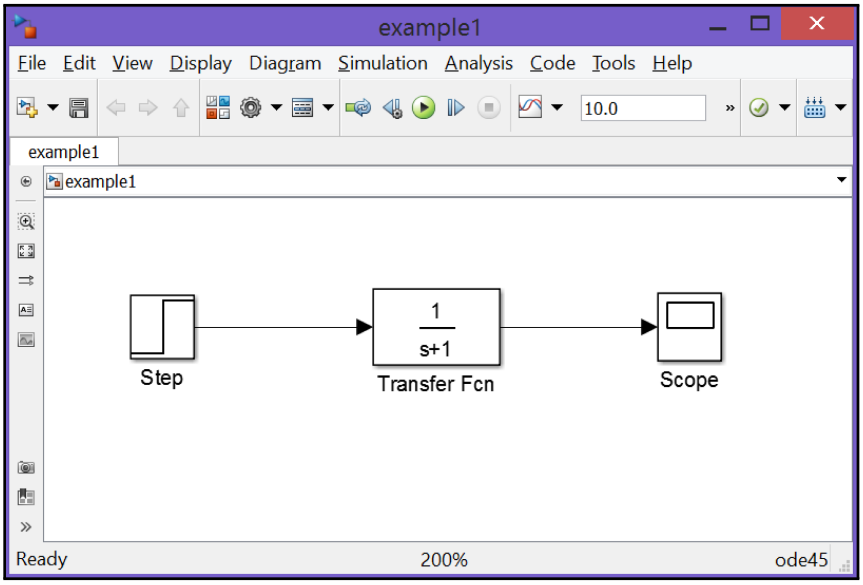


Figura 4.10 Un semplice esempio di modello in Simulink.

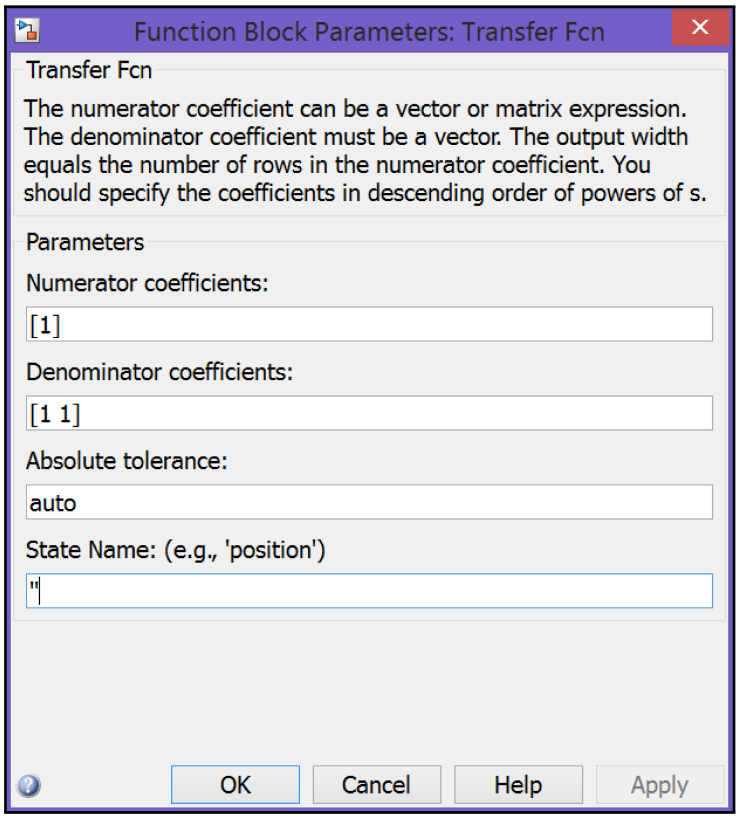


Figura 4.11 Finestra di dialogo del blocco Transfer Fcn.

Se si apre la finestra del blocco Scope subito dopo aver creato il modello, non avendo lanciato alcuna simulazione, essa non conterrà alcun grafico poiché non saranno ancora disponibili i dati provenienti dal canale di input. Dopo aver effettuato una simulazione — avendo cioè invertito la funzione di trasferimento — la finestra del blocco Scope visualizzerà la risposta al segnale di input a gradino. La finestra possiede una comoda funzione detta Autoscale che permette di adattare le scale degli assi ai valori massimo e minimo del segnale ottimizzando la visualizzazione della storia temporale.

Simulazione

Prima di avviare la simulazione, conviene aprire (con un doppio click) e tenere aperta la finestra di dialogo del blocco Scope. Nella finestra del progetto è opportuno salvare il

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

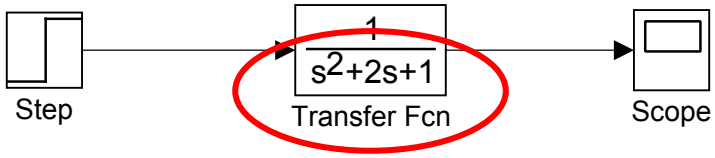


Figura 4.12 Modifica del modello della figura 4.10, ottenuta inserendo i valori `[1 2 1]` come "Denominator coefficients" nella maschera di configurazione del blocco Transfer Fcn.

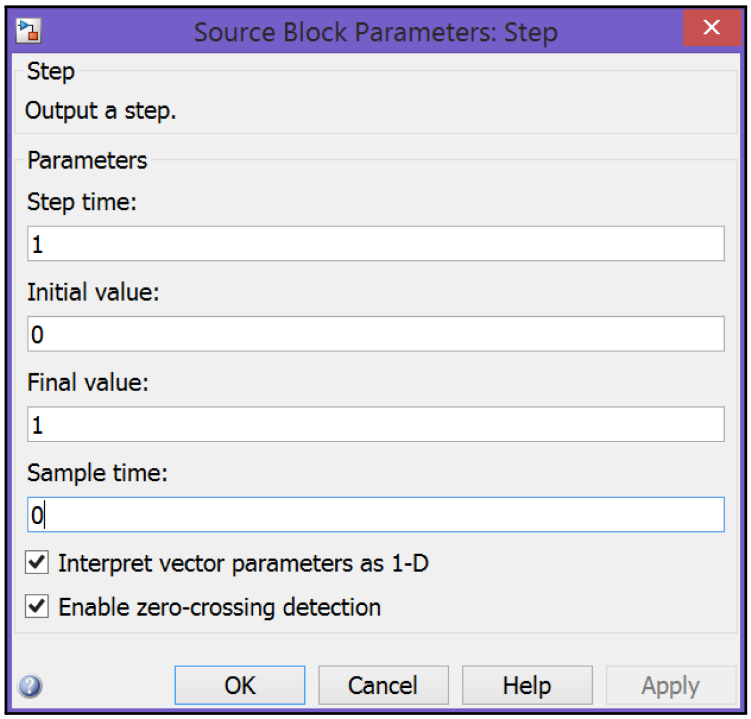


Figura 4.13 Finestra di dialogo del blocco Step.

modello della figura 4.12 in un file. I modelli Simulink hanno estensione "`.slx`". Dunque si potrà nominare il progetto come `example1a` e salvarlo nel file `example1a.slx` attraverso il menu File ▸ Save as (il progetto della figura 4.11 sarà `example1.slx`).

Nella finestra del modello `example1a` la voce di menu Simulation ▸ Model Configuration Parameters permette di impostare i parametri della simulazione. Il menu a tendina è mostrato nella figura 4.15 a pagina 28. Per semplicità si mantengono le impostazioni di default; ad esempio, nella categoria *Solver* si ha un tempo finale della simulatione $t_f = 10\,\mathrm{s}$ e una soluzione numerica delle equazioni del sistema con uno schema del tipo Runge-Kutta (Dormand-Prince) a passo variabile. L'avvio della simulazione è invece mostrato nella figura 4.16 a pagina 29.

Il tempo di esecuzione del modello è molto breve su un computer di medie prestazioni. La risposta nel blocco Scope apparirà come nella figura 4.14. Si osserva che la risposta è diversa da zero solo dopo l'istante $t = 1$. Questo parametro può essere cambiato nel blocco Step.

Per esercizio si provi a cambiare i parametri del sistema e a far ripartire la simulazione. Ad esempio, nel blocco Transfer Fcn si ridefinisca il denominatore della funzione di trasferimento dando l'array `[1 20 400]`. Si mandi in esecuzione la simulazione del modello modificato e si confronti l'output con quello del modello precedente.

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

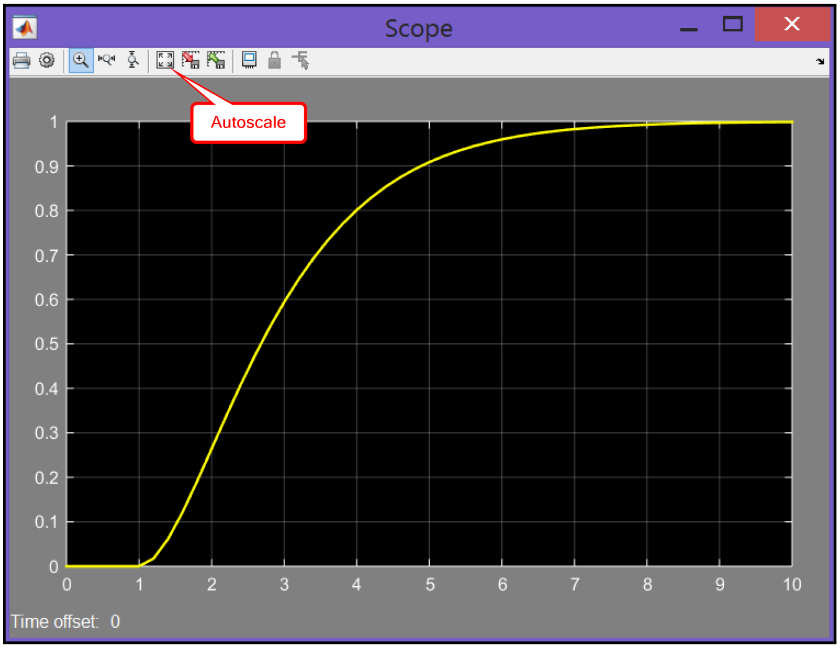


Figura 4.14 Finestra di dialogo del blocco Scope e tasto della funzione Autoscale.

4.7.3 Esempio del modello di un sistema *closed loop*

Un secondo esempio, utile a mostrare altri aspetti interessanti di Simulink, è quello di un semplice modello di sistema *closed loop* o sistema dinamico *a ciclo chiuso*. Per costruirne uno in Simulink si inseriranno in primo luogo alcuni blocchi predefiniti, per poi impostarli opportunamente e collegarli tra di loro come necessario. Infine si otterrà la simulazione del modello.

Inserimento dei blocchi

Per costruire il modello vanno effettuate le seguenti operazioni:

- Si crea un nuovo modello Simulink.
- Si inserisce un blocco Step.
- Si inseriscono blocchi Sum e Gain.
- Si inseriscono due blocchi Transfer Fcn.
- Si inserisce un blocco Scope.
- Si nomina il modello `example2.slx`.

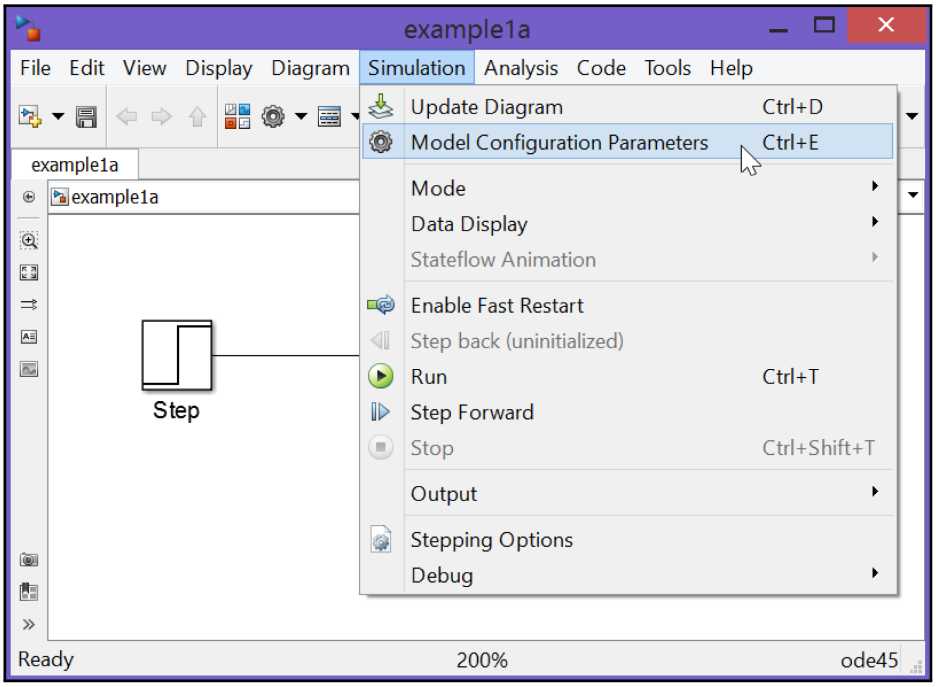
La disposizione e l'impostazione dei blocchi è mostrata nella figura 4.17.

Impostazione, collegamento dei blocchi e simulazione

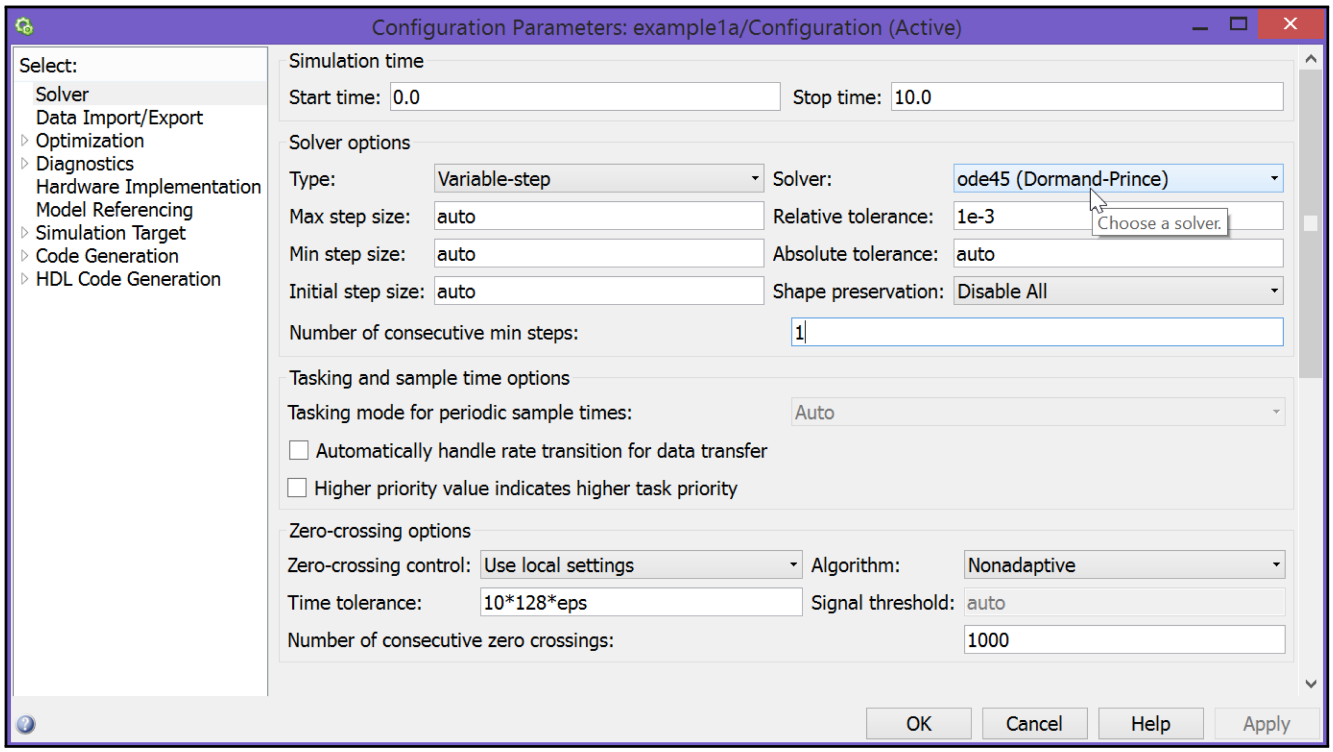
L'aspetto finale del modello è quello della figura 4.18. In esso si osserva quanto segue:

- Nella finestra di configurazione del blocco Sum si è dato `|+-` nel campo "List of signs".
- Nella configurazione del blocco Gain si è dato al guadagno un valore `2.5`.
- Il primo blocco (a sinistra) di tipo Transfer Fcn è stato etichettato "PI Controller" e la funzione di trasferimento è configurata con un numeratore `[1 2]` e un denominatore `[1 0]`.
- Il secondo blocco Transfer Fcn è stato nominato "Plant" e la funzione di trasferimento è configurata con un numeratore `[1]` e un denominatore `[1 2 4]`.

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro



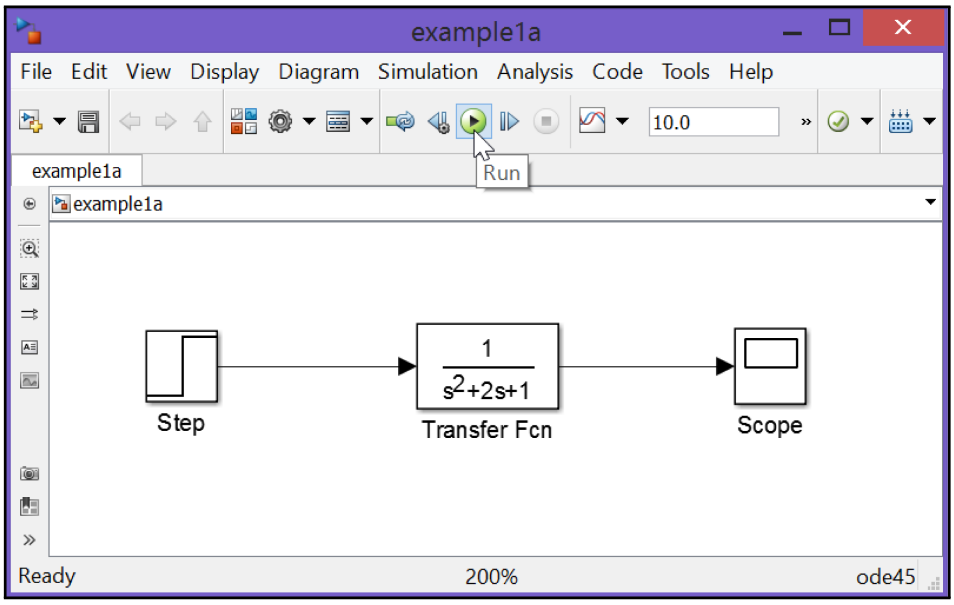
(a) Selezione dal menu Simulation.



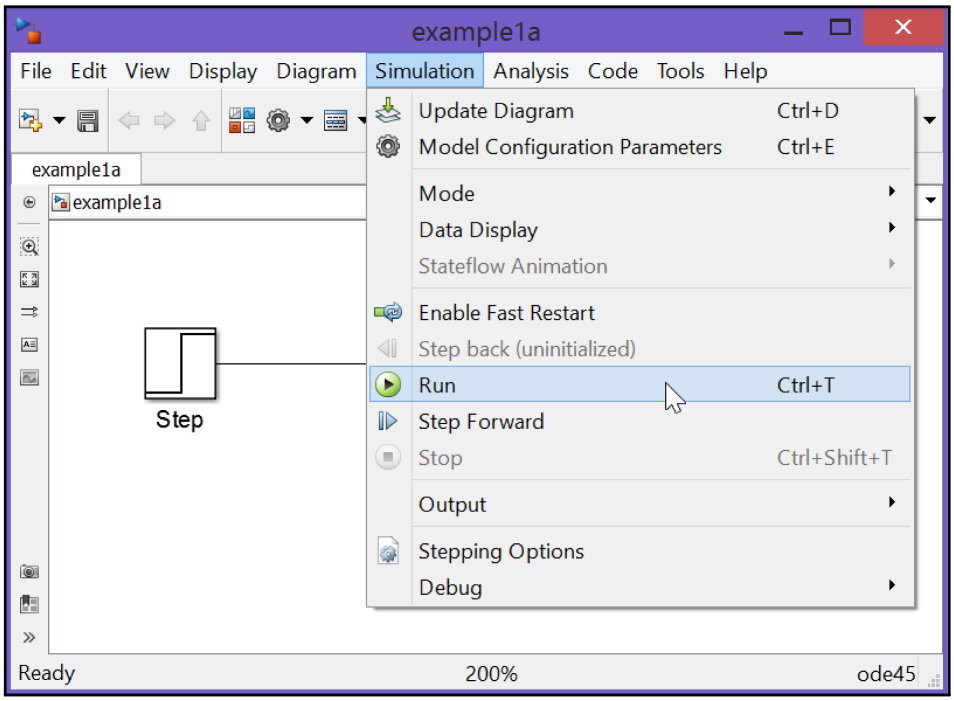
(b) Finestra di configurazione dei parametri della simulazione.

Figura 4.15 Configurazione dei parametri della simulazione in un modello in Simulink.

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro



(a) Tasto rapido di Run.



(b) Esecuzione del modello dal menu del progetto.

Figura 4.16 Lancio di una simulazione di un modello in Simulink.

La figura 4.19 mostra l'output del sistema rappresentato tramite il blocco Scope dopo aver mandato in esecuzione il modello Simulink.

4.7.4 Interazione tra Matlab e Simulink

Simulink è completamente integrato in Matlab, quindi è possibile utilizzare variabili presenti nel workspace di Matlab per impostare i blocchi di un modello Simulink.

Ad esempio, se k è una variabile assegnata tramite il comando:

```
>> k = 2.5;
```

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

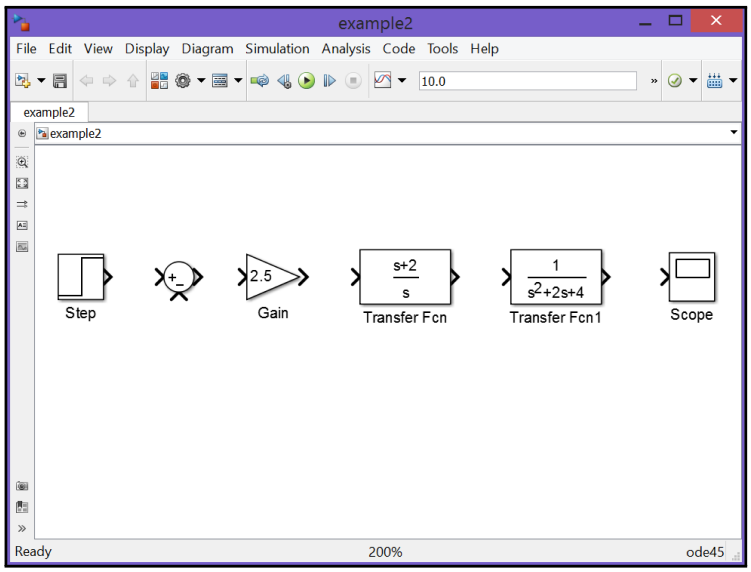


Figura 4.17 Preparazione di un modello di sistema *closed loop* nominato `example2`.

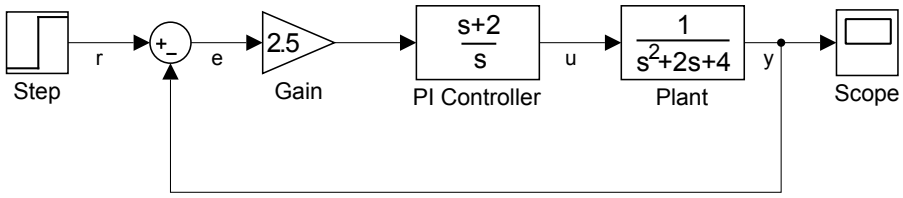


Figura 4.18 Collegamento dei blocchi nel modello `example2`.

nella finestra di configurazione del blocco Gain dell'esempio precedente è possibile dare `k` nel campo "Gain", cioè inserire il nome di una variabile presente nel workspace prima dell'esecuzione del modello anziché un valore numerico. Ciò permetterà di configurare una sola volta il blocco Simulink ed eventualmente di ripetere le simulazioni variando di volta in volta il guadagno con uno script Matlab.

Simulink fornisce anche la possibilità di usare i dati prodotti dall'esecuzione di un modello per costruire variabili ed allocarle nel workspace di Matlab. Per approfondimenti si veda il blocco di tipo To Workspace nel Simulink Library Browser nel percorso Simulink/Sinks.

Un esempio di uscita nel workspace è dato dal modello `example2a` della figura 4.20 nella pagina successiva. Il segnale in uscita dal blocco PI Controller (cioè l'input effettivo del sistema *closed loop*) è canalizzato sia verso il blocco Scope per la rappresentazione grafica (insieme all'output del modello) sia nel blocco To Workspace. La finestra del blocco Scope apparirà come nella figura 4.21. Se si imposta la finestra di dialogo come nella figura 4.22 si otterrà nel workspace di Matlab la variabile strutturata `simout` i cui campi `simout.Time` e `simout.Data` rappresentano i valori discreti del tempo di simulazione e del segnale di input del blocco Plant.

4.8 Simulazione di un modello fisico con Simulink

In Simulink è molto facile modellare sistemi dinamici relativamente complessi a partire dalle leggi fisiche che ne determinano l'evoluzione nel tempo. In questo paragrafo viene presentato un esempio interessante tratto dall'ingegneria dei trasporti.

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

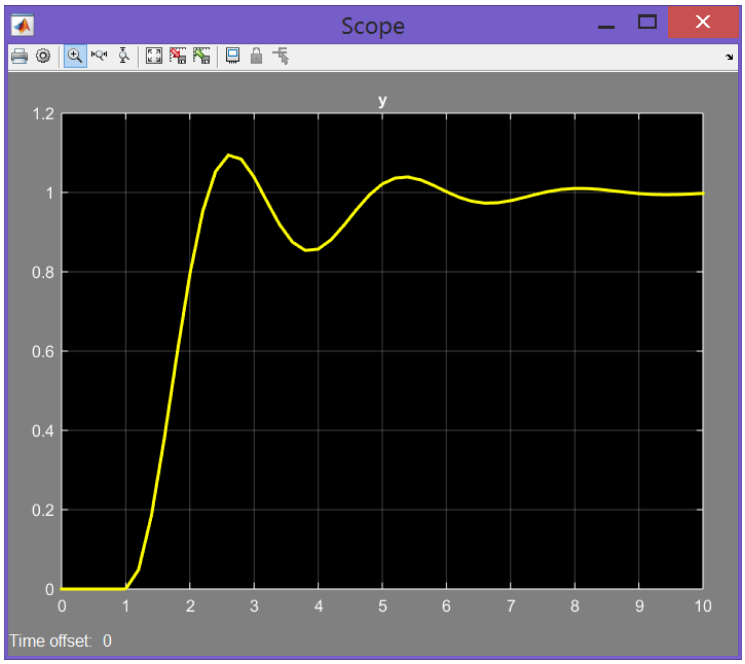


Figura 4.19 Output del modello `example2`.

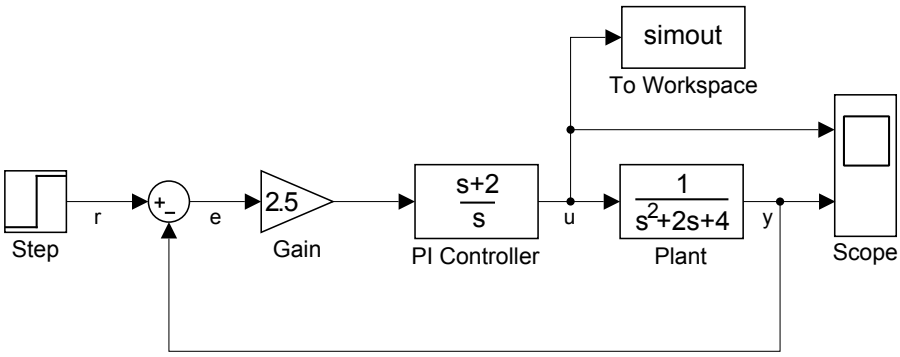


Figura 4.20 L'aspetto del modello `example2a` con i due segnali di input e output canalizzati nel blocco Scope. L'input prodotto dal blocco PI Controller è anche salvato nel workspace di Matlab.

4.8.1 Un semplice modello di treno

Si consideri il *toy train* schematizzato nella figura 4.23 a pagina 33, composto da una motrice e un vagone. Le massa della motrice e del vagone sono date, rispettivamente, dalle costanti M_1 ed M_2. Le due masse, collegate tra di loro tramite una molla ideale di costante elastica k, sono poggiate su un piano di scorrimento orizzontale e possono traslare grazie alla presenza di ruote. Queste ultime sono caratterizzate da un coefficiente di attrito volvente μ. La motrice è tale perché dotata di un motore che applica idealmente una forza $F(t)$ alla massa M_1.

Quando il sistema è a riposo ($t = 0$, $F = 0$) le posizioni x_1 e x_2 sono nulle e l'elongazione della molla è nulla. Se le due masse fossero collegate da una barra rigida allora le storie temporali $x_1(t)$ e $x_2(t)$ sarebbero identiche; la presenza della molla rende possibile una differenza o elongazione $\Delta(t) = x_1 - x_2$ non nulla.

Dalla legge di Newton è noto che la somma delle forze agenti su un corpo è uguale al prodotto della massa per l'accelerazione. In questo caso, le forze agenti su M_1 sono la reazione della molla, la forza d'attrito e la forza F dovuta al motore, mentre le forze agenti su M_2 sono la reazione della molla e la forza di attrito. Lungo la direzione verticale, la forza di gravità è cancellata dalla forza normale applicata dal suolo, quindi saranno nulle le componenti verticali dell'accelerazione. Le equazioni del moto in direzione orizzontale

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

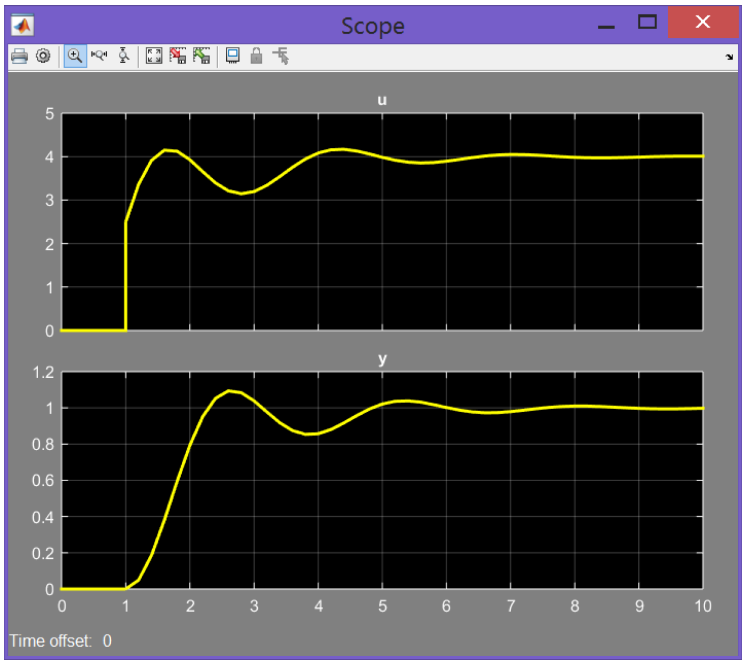


Figura 4.21 I segnali di input e output del blocco Plant nel modello `example2a`.

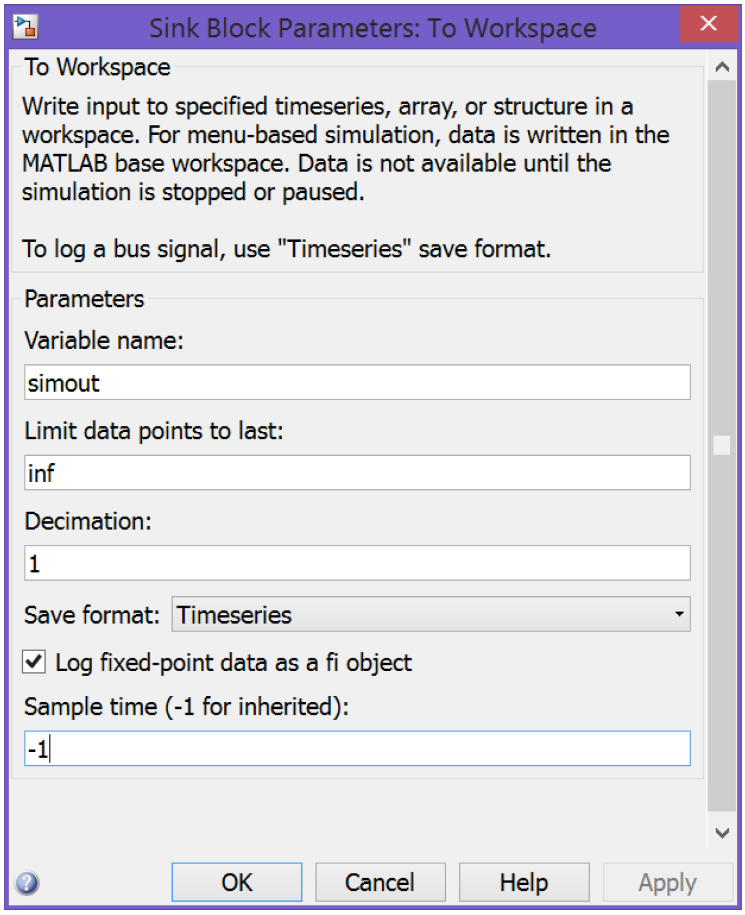


Figura 4.22 La finestra di configurazione del blocco To Workspace nel modello `example2a`.

sono le seguenti:

$$M_1\,\ddot{x}_1 = F - k\,(x_1 - x_2) - \mu\,M_1\,g\,\dot{x}_1 \tag{4.1a}$$

$$M_2\,\ddot{x}_2 = k\,(x_1 - x_2) - \mu\,M_2\,g\,\dot{x}_2 \tag{4.1b}$$

Queste equazioni possono essere riformulate definendo un insieme di variabili di stato e pervenendo ad un sistema di equazioni differenziali del primo ordine dette *equazioni di stato* o *equazioni di evoluzione* del sistema dinamico. Nel caso in esame si hanno quattro

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

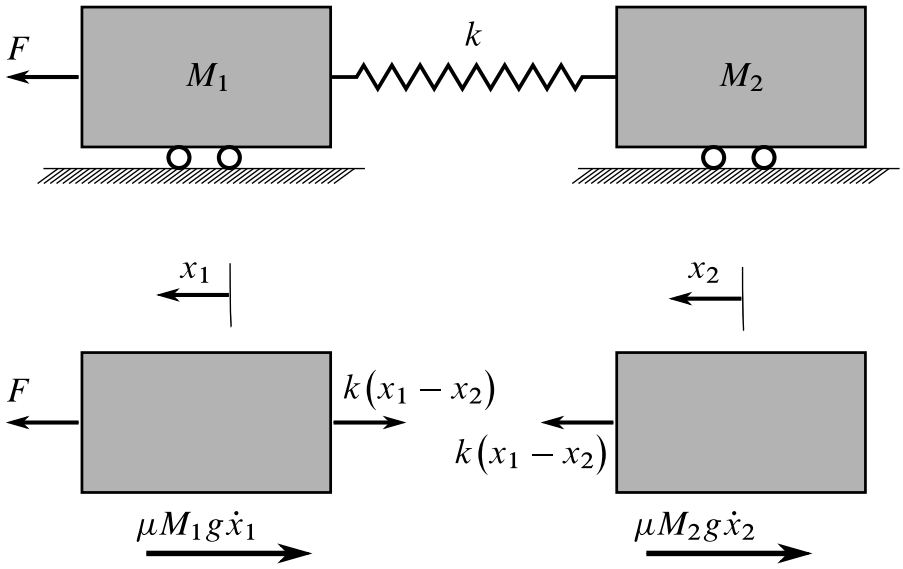


Figura 4.23 In alto, lo schema di due masse rotolanti e collegate da una molla. In basso, il corrispondente *free body diagram*.

variabili di stato: le posizioni, x_1 e x_2, e le velocità, $x_3 = \dot{x}_1$ e $x_4 = \dot{x}_2$. Esse formano il vettore di stato $\boldsymbol{x}$ del sistema dinamico. L'unica variabile di input qui considerata è $u_1 = F$.

Dalle (4.1) si ricavano le seguenti equazioni di stato:

$$\dot{x}_1 = x_3 \tag{4.2a}$$

$$\dot{x}_2 = x_4 \tag{4.2b}$$

$$\dot{x}_3 = -\frac{k}{M_1} x_1 + \frac{k}{M_1} x_2 - \mu\, g\, x_3 + \frac{F}{M_1} \tag{4.2c}$$

$$\dot{x}_4 = \frac{k}{M_2} x_1 - \frac{k}{M_2} x_2 - g\, \mu\, x_4 \tag{4.2d}$$

Ponendo infine come output del sistema la sola velocità della motrice, l'equazione di output associata alle (4.2) è la semplice equazione scalare:

$$y = x_3 \tag{4.3}$$

Dalle (4.2)-(4.3) si osserva che il comportamento del particolare sistema considerato nello spazio degli stati è caratterizzato da equazioni lineari a coefficienti costanti. Esse corrispondono formalmente alle equazioni canoniche

$$\dot{\boldsymbol{x}} = [A]\boldsymbol{x} + [B]\boldsymbol{u}\,, \qquad \boldsymbol{y} = [C]\boldsymbol{x} + [D]\boldsymbol{u} \tag{4.4}$$

In Matlab/Simulink le 4 matrici $[A]$, $[B]$, $[C]$ e $[D]$ possono essere agevolmente definite e utilizzate per la simulazione del sistema. Le (4.4) non sono altro che le equazioni seguenti:

$$\begin{Bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{Bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -k/M_1 & k/M_1 & -\mu g & 0 \\ k/M_2 & -k/M_2 & 0 & -\mu g \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1/M_1 \\ 0 \end{bmatrix} F \tag{4.5}$$

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

$$y = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} + \begin{bmatrix} 0 \end{bmatrix} F \tag{4.6}$$

dalle quali si evincono le espressioni delle matrici del sistema lineare.

Rappresentazione in Matlab

Le equazioni appena trovate si implementano in Matlab con poche semplici istruzioni. Si assuma: $M_1 = 1{,}00\,\mathrm{kg}$, $M_2 = 0{,}50\,\mathrm{kg}$, $k = 1{,}00\,\mathrm{N\,m^{-1}}$, $F = 1{,}00\,\mathrm{N}$, $\mu = 0{,}002\,\mathrm{s\,m^{-1}}$, $g = 9{,}81\,\mathrm{m/s^2}$.

Si crei un nuovo M-file di nome `setup_trainsim.m` inserendovi le istruzioni seguenti:

```
%% Assign data
M1  = 1;     % kg
M2  = 0.5;   % kg
k   = 1;     % N/m
F   = 1;     % N
mu0 = 0.002; % s/m
g   = 9.81;  % m/s^2
```

(continua)

Operando nello spazio degli stati si aggiunga anche:

(continua dal listato precedente)

```
%% Build linear system matrices in State-Space
A = ...
[    0,     0,      1,      0; ...
     0,     0,      0,      1; ...
  -k/M1,  k/M1, -mu0*g,      0; ...
   k/M2, -k/M2,      0, -mu0*g];
B = [0; 0; 1/M1; 0]; % matrix 4x1
C = [0, 0, 1, 0];    % matrix 1x4
D = [0];             % matrix 1x1
train = ss(A,B,C,D);
```

La funzione `ss` è una funzione predefinita di Matlab che accetta in input le quattro matrici del sistema e restituisce una struttura dati che qui è nominata `train`. Essa è utilizzabile successivamente per trovare la risposta del sistema, in ciclo aperto o in ciclo chiuso, per un'assegnata legge del forzamento.

Dopo aver eseguito lo script `setup_trainsim.m` la risposta *open loop* del sistema per un input a gradino d'intensità F è ottenibile con il comando:

```
>> step(F*train, 10)
```

che produce il grafico dell'ampiezza della risposta (nel caso in esame $v_1 = \dot{x}_1 = x_3$) in funzione del tempo. Il risultato è mostrato nella figura 4.24.

Sebbene gran parte delle operazioni si possano effettuare usando sia lo spazio degli stati (SS) la funzione di trasferimento (TF), in Matlab è semplice passare dall'una all'altra rappresentazione se necessario.

Per passare dallo spazio degli stati alla funzione di trasferimento si utilizza il comando:

```
>> [num, den] = ss2tf(⟨matr. A⟩,⟨matr. B⟩,⟨matr. C⟩,⟨matr. D⟩,⟨i_u⟩)
```

dove i_u è l'indice della componente del vettore $\boldsymbol{u}$ degli input per cui si vuole la TF. Nel caso di singolo input, il parametro i_u va omesso. Nell'esempio considerato si ha:

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

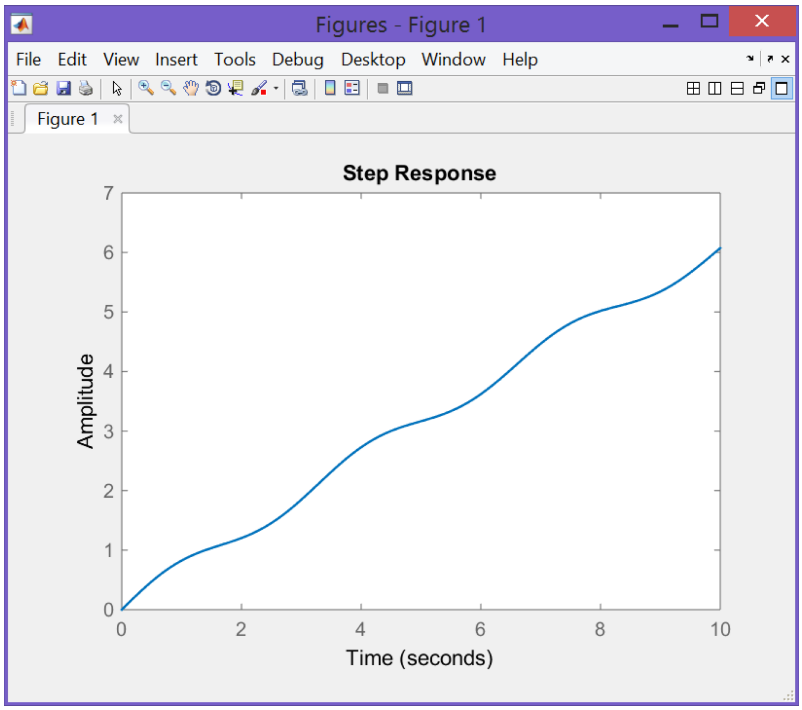


Figura 4.24 Output $v_1(t) = \dot{x}_1$ del modello motrice-vagone modellato in Matlab attraverso la funzione `ss`. Il sistema riceve un ingresso a gradino e la simulazione è ottenuta con il comando `step`.

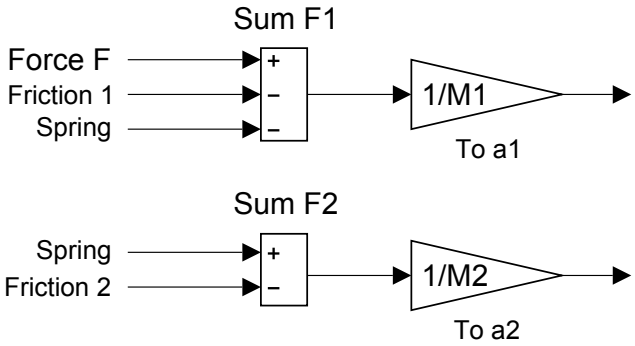


Figura 4.25 Preparazione del modello Simulink `trainsim.slx` con due blocchi di somma che implementano i secondi membri delle (4.7).

```
>>[num, den] = ss2tf(A,B,C,D)
num =
         0    1.0000    0.0196    2.4000         0
den =
    1.0000    0.0392    3.6004    0.0706   -0.0000
```

dove gli array `num` e `den` contengono i coefficienti dei polinomi a numeratore e denominatore, rispettivamente, della funzione di trasferimento del sistema.

Rappresentazione in Simulink

Il modello Simulink del vagone collegato alla motrice si chiamerà `trainsim.slx` e si baserà sull'applicazione della seconda legge di Newton così formulata:

$$\begin{cases} M_1\ddot{x}_1 = \left(\sum_{k=1}^{n_1} F_k\right)_1 \\ M_2\ddot{x}_2 = \left(\sum_{k=1}^{n_2} F_k\right)_2 \end{cases} \tag{4.7}$$

dove n_i è il numero di forze agenti sulla massa M_i. Dal punto di vista concettuale questo sistema di equazioni può essere rappresentato graficamente e senza ulteriori manipolazioni in un modello Simulink.

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

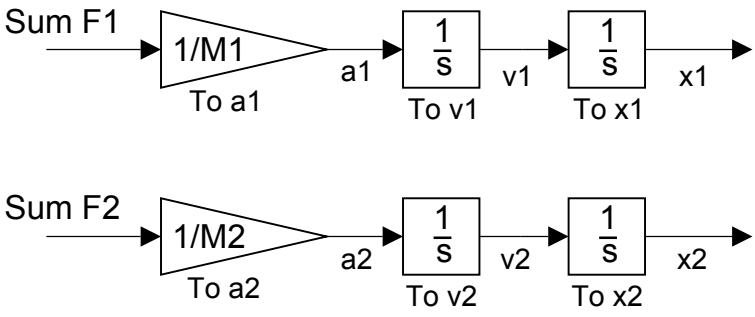


Figura 4.26 Blocchi Integrator in cascata nel modello `trainsim.slx` per l'ottenimento dei segnali $\dot{x}_1$, x_1, $\dot{x}_2$ e x_2.

Per fare ciò vanno costruite due rappresentazioni, una per ogni massa, dell'accelerazione $\ddot{x}_i = \left(\sum_k F_k\right)_i / M_i$. In termini di blocchi e linee di collegamento si avrà lo schema della figura 4.25. Si osservino i blocchi di somma "Sum F1" e "Sum F2" nei quali entrano i segnali corrispondenti alle forze istantanee F_k agenti su ciascuna massa. I blocchi di tipo Gain nominati "To a1" e "To a2" operano la divisione delle due forze per le masse M_1 ed M_2 e forniscono in uscita due segnali corrispondenti alle accelerazioni $\ddot{x}_1$ e $\ddot{x}_2$.

Una volta ottenuti i segnali con le accelerazioni basta applicare, concettualmente, per due volte un operatore di integrazione nel tempo e si otterranno le variabili di stato x_1 e x_2. In termini pratici queste operazioni sono effettuate da due blocchi Simulink di tipo Integrator collegati in cascata. Si vedano i blocchi contassegnati dal simbolo $\frac{1}{s}$ nella figura 4.26. Essi sono stati etichettati opportunamente come "To v1", "To x1", "To v2" e "To x2". Le condizioni iniziali del problema al tempo $t = 0$ sono tutte nulle, cioè le due masse sono inizialmente in quiete con la molla in condizione di riposo: $x_1(0) = x_2(0) = \dot{x}_1(0) = \dot{x}_2(0) = 0$.

A questo punto si hanno tutti i segnali che servono a costruire visualmente le espressioni delle singole forze agenti su ciascuna delle due masse. A partire dallo schema della figura 4.23 si 'disegnano' le connessioni della figura 4.27. Esse convogliano i segnali e li combinano opportunamente in modo da formare i segnali d'ingresso nei blocchi "Sum F1" e "Sum F2". Questa è un'interessante dimostrazione pratica del significato di equazioni del moto *in forma chiusa* assunto dalle (4.2): le forze esterne sono tutte espresse in termini delle variabili di stato o di input. Nel caso di un modello Simulink il termine 'forma chiusa' è anche associabile ai percorsi delle variabili di stato, che dopo essere passati attraverso i blocchi integratori 'ritornano indietro' verso i blocchi che assemblano le azioni forzanti.

Dalla figura 4.27 si osserva che la reazione della molla d'intensità $k(x_1 - x_2)$ si ottiene dapprima sottraendo i segnali "x1" e "x2" con il blocco di tipo Sum etichettato "x1 - x2" e successivamente inviando il risultato al blocco di tipo Gain etichettato "Spring". Il segnale in uscita da quest'ultimo viene diramato ed entra senza cambio di segno nel blocco di somma "Sum F2" e con segno cambiato nel blocco di somma "Sum F1". Analogamente, i segnali "v1" e "v2" associati alle velocità vengono prelevati e utilizzati per costruire attraverso i blocchi "Friction 1" e "Friction 2" le forze d'attrito.

Nel modello `trainsim.slx` l'input è costituito da una forzante a gradino rappresentata dal blocco di tipo Step etichettato "Force F". Esso è configurato in modo da generare un segnale nullo fino al tempo $t = 1$ s, dopo il quale il segnale ha valore costante pari a quello della variabile `F` assegnato dallo script `setup_trainsim.m`.

Nella figura 4.28 sono riportate le storie temporali degli spostamenti x_1 e x_2 e nella figura 4.29 è mostrata l'elongazione $x_1 - x_2$. Nella figura 4.30 si ha infine la velocità $\dot{x}_1$ della motrice. Si confronti questo grafico con quello della figura 4.24 e si osservi che la

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

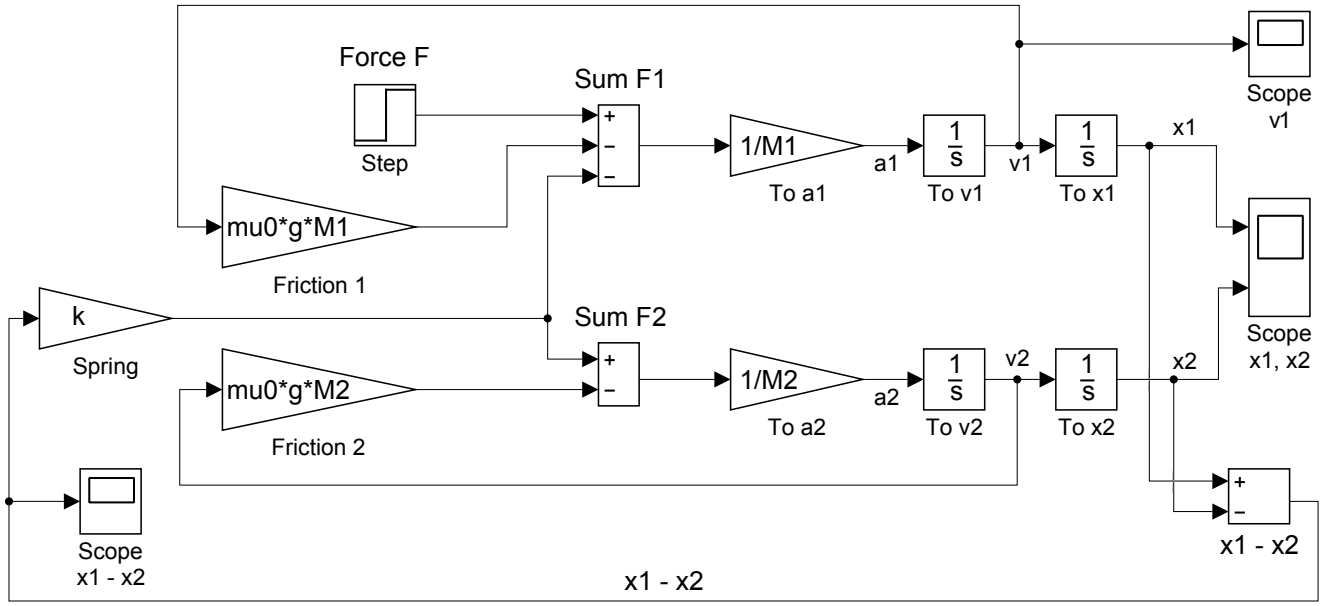


Figura 4.27 L'aspetto del modello di treno `trainsim.slx`.

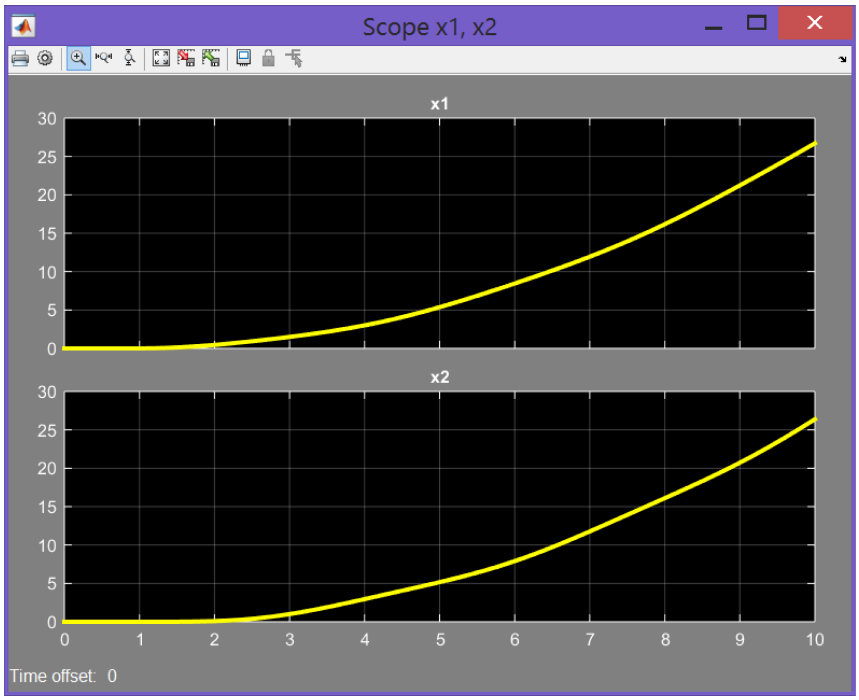


Figura 4.28 Storie temporali degli spostamenti x_1 e x_2 del modello `trainsim.slx`.

crescita di velocità della motrice, dovuta alla continua applicazione di una forza esterna costante, presenta delle oscillazioni dovute alla reazione della molla.

Per esercizio si provi a simulare il sistema motrice-vagone cambiando la legge temporale della forza applicata dalla motrice. Ad esempio, si vedano i blocchi di tipo Signal Generator e Signal Builder nella libreria di Simulink.

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

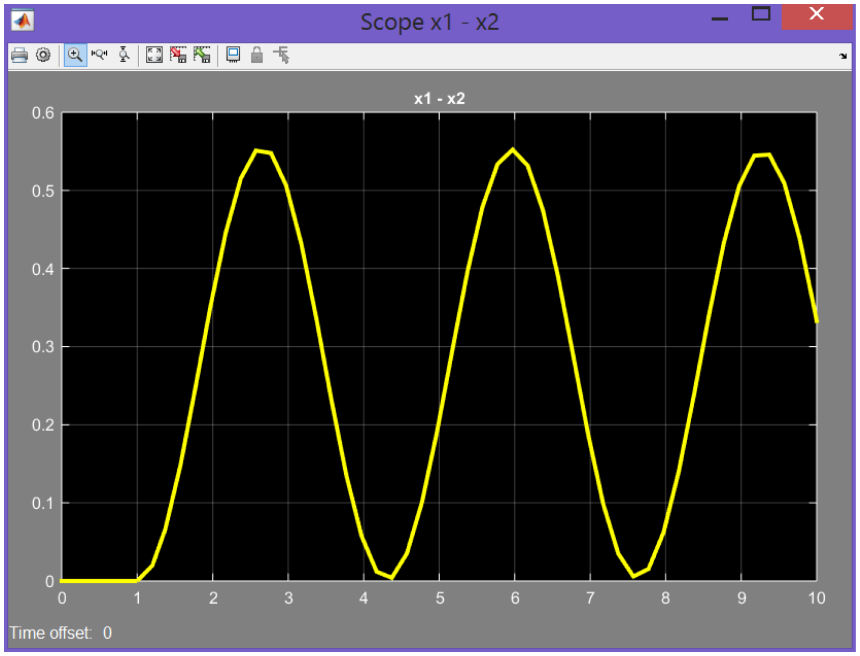


Figura 4.29 Storia temporale della differenza $x_1 - x_2$ (elongazione) del modello `trainsim.slx`.

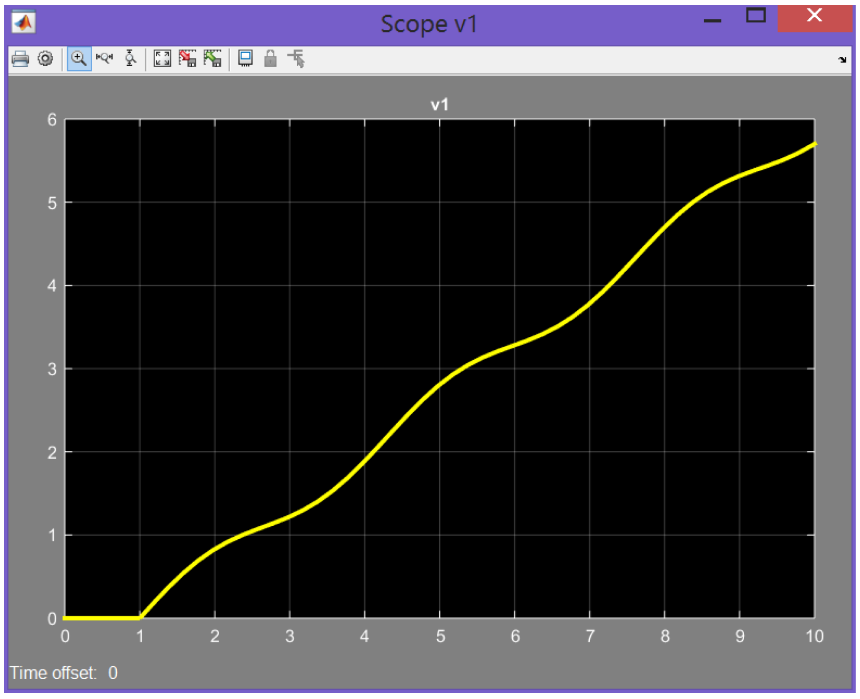


Figura 4.30 Storia temporale dell'uscita $\dot{x}_1$ del modello `trainsim.slx`. La crescita di velocità della motrice presenta delle oscillazioni dovute alla reazione della molla.

DRAFT ver. 2019.a Copyright © A. De Marco, D. P. Coiro

Bibliografia

[1] W. R. Hamilton, *Lectures on Quaternions*, Hodeges & Smith, 1853.

[2] O. Rodrigues, "Des lois géometriques qui régissent les désplacements d'un système solide dans l'espace, et de la variation des coordonnée provenant de ses désplacements considerées indépendamment des causes qui peuvent les produire", *Journal des Mathématiques Pures et Appliquées*, vol. 5, 1840.

[3] E. Salamin, "Application of Quaternions to Computation with Rotations", Working paper, Stanford AI Lab, 1979.

[4] A. P. Yefremov, "Quaternions: Algebra, Geometry and Physical Theories", *Hypercomplex Numbers in Geometry and Physics*, vol. 1, 2004.

[5] Schwab A. L., "Quaternions, Finite Rotations and Euler Parameters", Course notes on Applied Multibody Dynamics, Delft University of Technology, Laboratory for Engineering Mechanics, 2003.
http://tam.cornell.edu/~{}als93/quaternion.pdf.

[6] AIAA/ANSI, *Recommended Practice for Atmospheric and Space Flight Vehicle Coordinate Systems*. R-004-1992, 1992.

[7] G. H. Bryan, *Stability in Aviation: An Introduction to Dynamical Stability as Applied to the Motions of Aeroplanes*. Macmillan and Co., Limited, London, 1911.

[8] D. J. Diston, *Computational Modelling of the Aircraft and the Environment. Volume 1, Platform Kinematics and Synthetic Environment*. John Wiley & Sons, Inc., 2009.

[9] W. F. Phillips, *Mechanics of Flight*. John Wiley & Sons, Inc., 2004.

[10] W. F. Phillips, "Phugoid Approximation for Conventional Airplanes", Journal of Aircraft, Vol. 37, No. 1, January-February 2000.

[11] W. F. Phillips, "Improved Closed-Form Approximation for Dutch-Roll", Journal of Aircraft, Vol. 37, No. 1, May-June 2000.

[12] R. Stengel, *Flight Dynamics*. Princeton University Press, Princeton, 2004.

[13] M. R. Napolitano, *Aircraft Dynamics: From Modeling to Simulation*. John Wiley, 2012.

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

[14] D. K. Schmidt, *Modern Flight Dynamics*. McGraw-Hill, 2010.

[15] B. Stevens, F. Lewis, *Aircraft Control and Simulation*. John Wiley & Sons, Inc., 1992.

[16] D. Stinton, *The Anatomy of the Airplane* (2nd edition). American Institute of Aeronautics and Astronautics, 1998.

[17] B. Etkin, *Dynamics of Flight, Stability and Control*. John Wiley & Sons, New York, 1982.

[18] M. Calcara, *Elementi di dinamica del velivolo*. Edizioni CUEN, Napoli, 1988.

[19] L. V. Schmidt, *Introduction to Aircraft Flight Dynamics*. AIAA Education Series, 1998.

[20] W. J. Duncan, *Control and Stability of Aircraft*. Cambridge University Press, Cambridge, 1952.

[21] R. Jategaonkar, *Flight Vehicle System Identification: A Time Domain Methodology*. Progress in Astronautics and Aeronautics Series, 2006.

[22] C. D. Perkins, R. E. Hage, *Aircraft Performance, Stability and Control*. John Wiley & Sons, New York, 1949.

[23] J. R. Wright, J.. E. Cooper, *Introduction to Aircraft Aeroelasticity and Loads*. John Wiley & Sons, Inc., 2007.

[24] V. Losito, *Fondamenti di Aeronautica Generale*. Accademia Aeronautica, Napoli, 1994.

[25] E. Torenbeek, H. Wittenberg, *Flight Physics*. Springer, Heidelberg, 2009.

[26] P. H. Zipfel, *Modeling and Simulation of Aerospace Vehicle Dynamics*. Second Edition. AIAA Education Series, American Institute of Aeronautics and Astronautics, Reston, VA. 2007.

[27] J. D. Mattingly, *Elements of Propulsion: Gas Turbines and Rockets*. AIAA Education Series, American Institute of Aeronautics and Astronautics, Reston, VA. 2006.

[28] K. Hünecke, *Jet Engines. Fundamentals of Theory, Design and Operation*. Motorbooks International, 1997.

[29] A. Linke-Diesinger, *Systems of Commercial Turbofan Engines*. Springer-Verlag, Berlin Heidelberg, 2008.

[30] F. R. Garza, E. A. Morelli, "A Collection of Nonlinear Aircraft Simulations with MATLAB". NASA-TM-2003-212145, January 2003.

[31] Voce WGS84 su *Wikipedia*:
http://en.wikipedia.org/wiki/World_Geodetic_System

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

[32] Anonimo, *Department of Defense World Geodetic System 1984. Its Definition and Relationship with Local Geodetic Systems*. NIMA TR8350.2, Third Edition, Amendment 2. National Imagery and Mapping Agency, US Department of Defense, 2004.

[33] J. Roskam, *Airplane Flight Dynamics and Automatic Flight Controls*. DARcorporation, 2001.

[34] H. T. Schlichting, E. A. Truckenbrodt, *Aerodynamics of the Aeroplane*. McGraw Hill Higher Education, 2nd edition, 1979.

[35] M. M. Munk, "The aerodynamic forces on airship hulls". NACA-TR-184, 1924.

[36] A. Silverstein, S. Katzoff, "Aerodynamic characteristics of horizontal tail surfaces". NACA-TR-688, 1940.

[37] R. I. Sears, "Wind-tunnel data on the aerodynamic characteristics of airplane control surfaces". NACA-WR-L-663, 1943.

[38] E. Garner, "Wind-tunnel investigation of control-surface characteristics XX: plain and balanced flaps on an NACA 0009 rectangular semispan tail surface". NACA-WR-L-186, 1944.

[39] J. D. Brewer, M. J. Queijo, "Wind-tunnel investigation of the effect of tab balance on tab and control-surface characteristics". NACA-TN-1403, 1947.

[40] S. M. Crandall, H. E. Murray, "Analysis of available data on the effects of tabs on control-surface hinge moments". NACA-TN-1049, 1946.

[41] B. W. McCormick, *Aerodynamics, Aeronautics, and Flight Mechanics*. John Wiley & Sons, 1979.

[42] B. N. Pamadi, *Performance, Stability, Dynamics and Control of Airplanes*. AIAA Education Series, 1998.

[43] A. Tewari, *Atmospheric and Space Flight Dynamics. Modelling and Simulation with Matlab and Simulink*. Birkhäuser, Berlin, 2007.

[44] D. Howe, *Aircraft Loading and Structural Layout*. AIAA Education Series, 2004.

[45] P. Morelli, *Static Stability and Control of Sailplanes*. Levrotto & Bella, Torino, 1976.

[46] L. Prandtl, O. G. Tietjens, *Fundamentals of Hydro and Aeromechanics*. Dover, 1957.

[47] R. K. Heffley, W. F. Jewell, "Aircraft Handling Qualities Data". NASA-CR-2144, December 1972.

[48] H. P. Stough III, J. M. Patton Jr, S. M. SliWa, "Flight Investigation of the Effect of Tail Configuration on Stall, Spin, and Recovery Characteristics of a Low-Wing General Aviation Research Airplane". NASA-TP-1987-2644, February 1987.

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

[49] J. D. Anderson, *Fundamentals of Aerodynamics*. McGraw-Hill, 3rd edition, New York, 2001.

[50] J. J. Bertin, *Aerodynamics for Engineers*. Prentice-Hall, 4th edition, Upper Saddle River, NJ, 2002.

[51] J. Katz, A. Plotkin, *Low-Speed Aerodynamics*. Cambridge University Press, 2nd edition, Cambridge, England, U.K., 2001.

[52] D. E. Hoak, *et al.*, "The USAF Stability and Control Datcom". Air Force Wright Aeronautical Laboratories, TR-83-3048, 1960 (Revised 1978).

[53] R. T. Jones, "A Note on the Stability and Control of Tailless Airplanes". NACA Report 837, 1941.

[54] D. P. Coiro, F. Nicolosi, A. De Marco, N. Genito, S. Figliolia, "Design of a Low Cost Easy-to-Fly STOL Ultralight Aircraft in Composite Material". *Acta Polytecnica*, Vol. 45 no. 4, 2005, pp. 73-80; ISSN 1210-2709.

[55] F. Nicolosi, A. De Marco, P. Della Vecchia, "Flight Tests, Performances and Flight Certification of a Twin-Engine Light Aircraft". *Journal of Aircraft*, Vol 48, No. 1, January-February 2011.

[56] F. Nicolosi, A. De Marco, P. Della Vecchia, "Parameter Estimation and Flying Qualities of a Twin-Engine CS23/FAR23 Certified Light Aircraft". AIAA-2010-7947, AIAA Atmospheric Flight Mechanics Conference, Toronto, 2010.

[57] B. Etkin, *Dynamics of Atmospheric Flight*, Dover Publications, 2005.

[58] L. Mangiacasale, *Flight Mechanics of a μ-Airplane*, Edizioni Libreria CLUP, Milano, 1998.

[59] G. Mengali, *Elementi di Dinamica del Volo con Matlab*, Edizioni ETS, Pisa, 2001.

[60] R. Nelson, *Flight Stability and Automatic Control*, McGraw-Hill, 1989.

[61] Y. Li, M. Nahon, "Modeling and simulations of airship dynamics", *Journal of Guidance, Controls and Dynamics*, Vol 30, No. 6, November-December 2007.

[62] Y. Fan, F. H. Lutze, E. M. Cliff, "Time-Optimal Lateral Maneuvers of an Aircraft", *Journal of Guidance, Controls and Dynamics*, Vol 18, No. 5, September-October 1995.

[63] J. N. Nielsen, *Missile Aerodynamics*, AIAA, Cambridge, MA, 1988.

[64] T. I. Fossen, *Guidance and Control of Ocean's Vehicles*, Whiley, New York, 1998.

[65] J. N. Newman, *Marine Hydrodynamics*, MIT Press, Cambridge, MA, 1977.

[66] E. L. Duke, R. F. Antoniewicz, K. D. Krambeer, "Derivation and Definition of a Linear Aircraft Model". Technical Report NASA Reference Publication RP-1207, Research Engineering, NASA Ames Research Center and NASA Dryden Flight Research Facility, 1988.

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro

[67] G. A. Stagg, *An Unsteady Aerodynamic Model for Use in the High Angle of Attack Regime*. MS thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1998.

[68] Y. Fan, *Identification of an Unsteady Aerodynamic Model up to High Angle of Attack Regime*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1997.

[69] *MATLAB Users' Guide*. The Mathworks, 2003 ed edizioni successive.
http://www.mathworks.com/
http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html

[70] V. Comincioli, *Analisi numerica: metodi, modelli, applicazioni*. McGraw-Hill, 1990, seconda edizione 1995.

[71] E. Kreyszig, *Advanced Engineering Mathematics*. John Wiley & Sons, seventh edition, 1993.

[72] C. de Boor, *A Practical Guide to Splines*. Springer-Verlag, 1978.

[73] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in Fortran: The Art of Scientific Computing*. Cambridge University Press, 1992.

[74] G. Dahlquist, A. Bjorck, *Numerical Methods. Volume I: Fundamentals of Numerical Discretization*. John Wiley & Sons, 1988.

[75] R. D. Richtmyer, K. W. Morton, *Difference Methods for Initial Value Problems*. Wiley-Interscience, 1967.

[76] C. Hirsch, *Numerical Computation of Internal and External Flows*. John Wiley & Sons, 1994.

[77] R. D. Finck, "USAF Stability and Control Datcom". AFWAL-TR-83-3048, October 1960, Revised 1978.

[78] S. R. Vukelich, J. E. Williams, "The USAF Stability and Control Digital Datcom". AFFDL-TR-79-3032, Volume I, April 1979, Updated by Public Domain Aeronautical Software 1999.

[79] W. B. Blake, "Prediction of Fighter Aircraft Dynamic Derivatives Using Digital Datcom". AIAA-85-4070, AIAA Applied Aerodynamics Conference, Colorado Springs, Colorado, 1985.

[80] Autori Vari, Distribuzione ufficiale di Digital Datcom,
sito internet:
http://wpage.unina.it/agodemar/DSV-DQV/Digital-Datcom-Package.zip

[81] B. Galbraith, "Digital Datcom+", Holy Cows, Inc.,
sito internet: http://www.holycows.net/datcom/

DRAFT ver. 2019.a Copyright© A. De Marco, D. P. Coiro