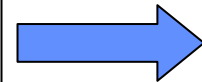

Introduzione a Matlab



Cose è MatLab ?

- ❑ MATLAB (MATrix LABoratory) è un programma interattivo per il calcolo numerico, in cui l'elemento base è la matrice.
- ❑ MATLAB è un ambiente di lavoro che include la possibilità di scrivere un proprio codice, usare una potente libreria grafica, usare un ambiente grafico per simulare sistemi complessi (Simulink), usare una ricchissima libreria di funzioni matematiche (Toolbox), creare delle interfacce user-friendly;



Proprio codice
Libreria grafica
Toolbox
Simulink
GUI

estensioni

.m

.fig

.m

.mdl

.m

Interfaccia Matlab 7.1

Lancia Simulink

Crea GUI

Percorso directory corrente

Directory
corrente

The screenshot shows the MATLAB 7.1 interface. The top menu bar includes File, Edit, Debug, Desktop, Window, and Help. The current directory path is C:\MATLAB6p5\work\Stima3_positivi. The Current Directory browser on the left lists various folders and files. The Command Window on the right contains the following code and output:

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

>> whos A
Name      Size      Bytes  Class
A         3x3         72  double array

Grand total is 9 elements using 72 bytes

>> lambda=eig(A)
lambda =
    16.7745
    -2.6583
     1.8838

>>
```

Command
Window

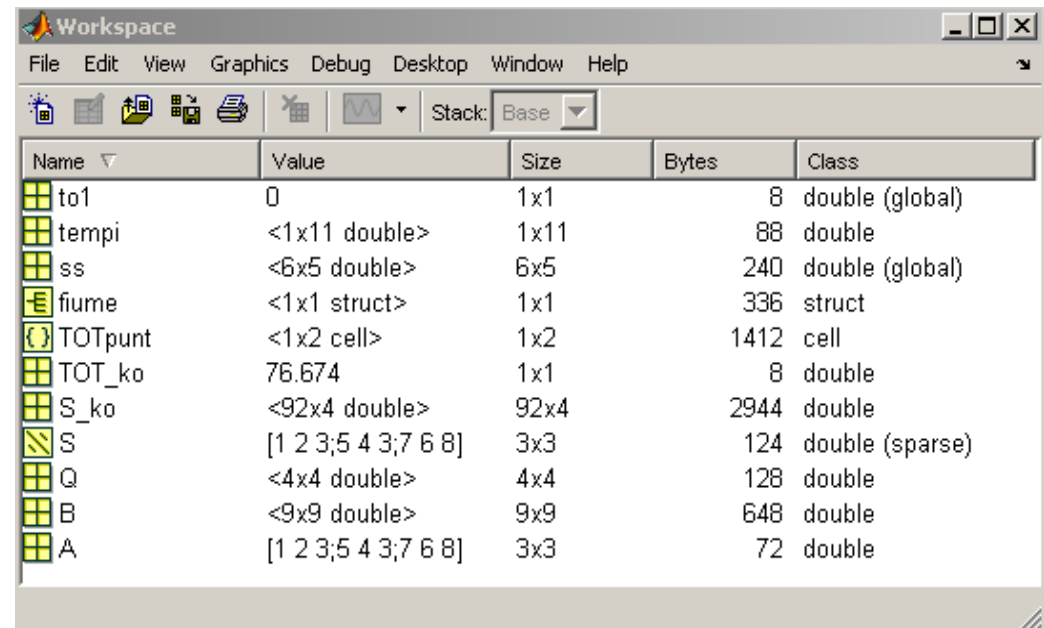
Workspace

- ❑ Le variabili vengono memorizzate nell'area di lavoro Workspace.
- ❑ Di default, Matlab lavora con variabili in doppia precisione. Ogni numero memorizzato in doppia precisione occupa 8 Bytes. Le variabili scalari sono viste come array di dimensione 1x1 Il contenuto di queste variabili può essere variato con una semplice operazione di assegnazione

La finestra Workspace contiene la lista delle variabili e le seguenti informazioni:

- ❑ Name: nome della variabile.
- ❑ Value: valore assegnato alla variabile.
- ❑ Size: dimensione come array (righe per colonne).
- ❑ Bytes: occupazione di memoria in termini di bytes.
- ❑ Class: il tipo di variabile char, double, sparse, cell, struct, uint8.

Di default, Matlab lavora con variabili in doppia precisione.



Name	Value	Size	Bytes	Class
to1	0	1x1	8	double (global)
tempi	<1x11 double>	1x11	88	double
ss	<6x5 double>	6x5	240	double (global)
fiume	<1x1 struct>	1x1	336	struct
TOTpunt	<1x2 cell>	1x2	1412	cell
TOT_ko	76.674	1x1	8	double
S_ko	<92x4 double>	92x4	2944	double
S	[1 2 3;5 4 3;7 6 8]	3x3	124	double (sparse)
Q	<4x4 double>	4x4	128	double
B	<9x9 double>	9x9	648	double
A	[1 2 3;5 4 3;7 6 8]	3x3	72	double

Comandi base fondamentali

- ❑ Il punto e virgola al termine di un'istruzione MATLAB impedisce la visualizzazione del risultato dell'istruzione stessa.
- ❑ MATLAB è case-sensitive
- ❑ In MATLAB le variabili non devono essere dichiarate. La dichiarazione coincide con il primo assegnamento. Es. $A=[1,3;4,2]$; matrice 2x2
- ❑ Il carattere % consente di introdurre commenti. *I commenti sono la parte nobile di qualsiasi programma.*
- ❑ Due o più punti . . . consentono di estendere un'istruzione alla riga successiva.
- ❑ L'apice “ ’ ” associato ad una variabile ne calcola la trasposta.
- ❑ è disponibile un HELP in linea. Basta digitare `help < topic >`. Es. *help spline*

Comandi base fondamentali

- ❑ All'avvio di MATLAB appare il prompt “**>>**” (stesso significato del vecchio DOS)
- ❑ Vi sono due tipi di istruzioni:
 - ❑ assegnamenti “**>>** *variabile = espressione*” Es. A=3;
 - ❑ valutazione di espressioni “**>>** *espressione*” Es. valuta_ottimo
- ❑ La valutazione di un'espressione genera una matrice che viene assegnata alla variabile indicata. Quando nell'istruzione non si specifica la variabile a cui assegnare il risultato, la valutazione dell'espressione viene assegnata alla variabile di sistema “*ans*”.
- ❑ Costanti predefinite:
 - ❑ pi pigreco
 - ❑ Inf infinito
 - ❑ NaN Not a Number (generata da 0/0, o Inf/Inf)

Comandi base fondamentali

- ❑ `whos` elencano le variabili in uso. Es. *whos A*
- ❑ `dir` elenca i files contenuti nel direttorio corrente
- ❑ `clear all` elimina tutte le variabili della sessione corrente
- ❑ `clear var1 var2` elimina le variabili `var1` e `var2`
- ❑ `close all` chiude tutte le figure aperte
- ❑ `clc` pulisce la command window

Comandi base fondamentali

- Una matrice può essere definita con la sintassi seguente: uno spazio o una virgola delimitano gli elementi di un stessa riga; un punto e virgola o un cambio di riga indicano la fine di una riga

$A=[4\ 7\ 3]$ e $A=[4,7,3]$ producano lo stesso vettore riga

$B=[5;3;8]$ e $B = \begin{bmatrix} 5 \\ 3 \\ 8 \end{bmatrix}$ producano lo stesso vettore colonna

- Sono presenti funzioni predefinite per la generazione di particolari matrici:

“*zeros(n,m)*” matrice di zeri

“*ones(n,m)*” matrice di uni

“*eye(n,m)*” matrice identità

“*rand(n,m)*” matrice di numeri casuali

“*diag([a11, a22, a33, ..., aNN])*” matrice diagonale

Comandi base fondamentali

❑ Per accedere agli elementi di una matrice:

```
» A = [7 8; 8.9 7; 9 8]
```

```
A =
```

```
7.0000 8.0000
```

```
8.9000 7.0000
```

```
9.0000 8.0000
```

```
» A(1,2)
```

```
ans =
```

```
8
```

```
» A(2,:) 
```

```
ans =
```

```
8.9000 7.0000
```

```
» A(:,1)
```

```
ans =
```

```
7.0000
```

```
8.9000
```

```
9.0000
```

❑ $A(n,m)$ estrae l'elemento (n,m) della matrice A

❑ $A(n,:)$ estrae l'n-esima riga della matrice A

❑ $A(:,m)$ estrae l'm-esima colonna della matrice A

Comandi base fondamentali

- Un vettore può essere creato con la stessa sintassi utilizzata per le matrici oppure con le istruzioni:

```
» x = 1:6
```

```
x =
```

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

```
» x = 0.5:0.1:0.9
```

```
x =
```

```
0.5000 0.6000 0.7000 0.8000 0.9000
```

```
» x = linspace(-1,1,4)
```

```
x =
```

```
-1.0000 -0.3333 0.3333 1.0000
```

- `a : [step :] b` crea un vettore riga di estremi a e b. Il parametro opzionale `step` indica l'intervallo tra ciascun elemento del vettore
- `linspace(a,b,N)` crea un vettore riga di estremi a e b, costituito da N punti equispaziati

Funzioni elementari per scalari

- ❑ Gli operatori aritmetici presenti in MATLAB sono:
+ (somma) , - (differenza), * (prodotto), / (quoziente), ^ (elevamento a potenza)
- ❑ Funzioni matematiche elementari:
 - ❑ abs valore assoluto di un numero complesso
 - ❑ angle fase di un numero complesso
 - ❑ conj complesso coniugato
 - ❑ exp esponenziale in base e
 - ❑ real, imag parte reale e parte immaginaria di un numero complesso
 - ❑ log, log10 logaritmo naturale ed in base 10
 - ❑ sqrt radice quadrata
- ❑ Funzioni trigonometriche
 - ❑ sin, cos seno, coseno
 - ❑ tan tangente
 - ❑ asin, acos arco seno, arco coseno
 - ❑ atan arco tangente

Funzioni elementari per matrici

□ Gli operatori elementari sono:

$+$, $-$, $*$, $/$, \backslash , $.*$, $./$, $.^$

□ L'operazione di somma o di sottrazione è definita tra matrici aventi le stesse dimensioni. Se uno dei due operandi è uno scalare, esso viene sommato o sottratto a tutti gli elementi della matrice.

□ $X = B/A$ è la soluzione dell'equazione $X*A = B$

□ $X = A \backslash B$ è la soluzione dell'equazione $A*X = B$

□ $.*$, $./$ e $.^$ effettuano le corrispondenti operazioni sui singoli elementi delle matrici coinvolte.

□ Le funzioni matematiche elementari e trigonometriche, quando applicate alle matrici, si riferiscono ai singoli elementi della matrice

□ Principali operazioni matriciali:

□ Matrice trasposta A'

□ Matrice inversa $\text{inv}(A)$

Funzioni elementari per matrici

```
A=[5,7,8,6;3,4,1,0;3,5,8,9;2,4,4,7]
```

```
A =
```

```
 5  7  8  6
 3  4  1  0
 3  5  8  9
 2  4  4  7
```

```
>> det(A)
```

determinante

```
ans =
```

```
-6
```

```
>> eig(A)
```

autovalori

```
ans =
```

```
18.3138
```

```
-0.0361
```

```
2.8612 + 0.9402i
```

```
2.8612 - 0.9402i
```

```
>> v = poly(A)
```

$\det(\lambda I - A) = c_1 \lambda^n + \dots + c_n \lambda + c_{n+1}$

```
ans =
```

```
1.0000 -24.0000 113.0000 -162.0000 -6.0000
```

```
>> rank(A)
```

rango

```
ans =
```

```
4
```

```
>> size(A)
```

dimansioni

```
ans =
```

```
4  4
```

```
>> roots(v)
```

Radici del polinomio

```
ans =
```

```
18.3138
```

```
2.8612 + 0.9402i
```

```
2.8612 - 0.9402i
```

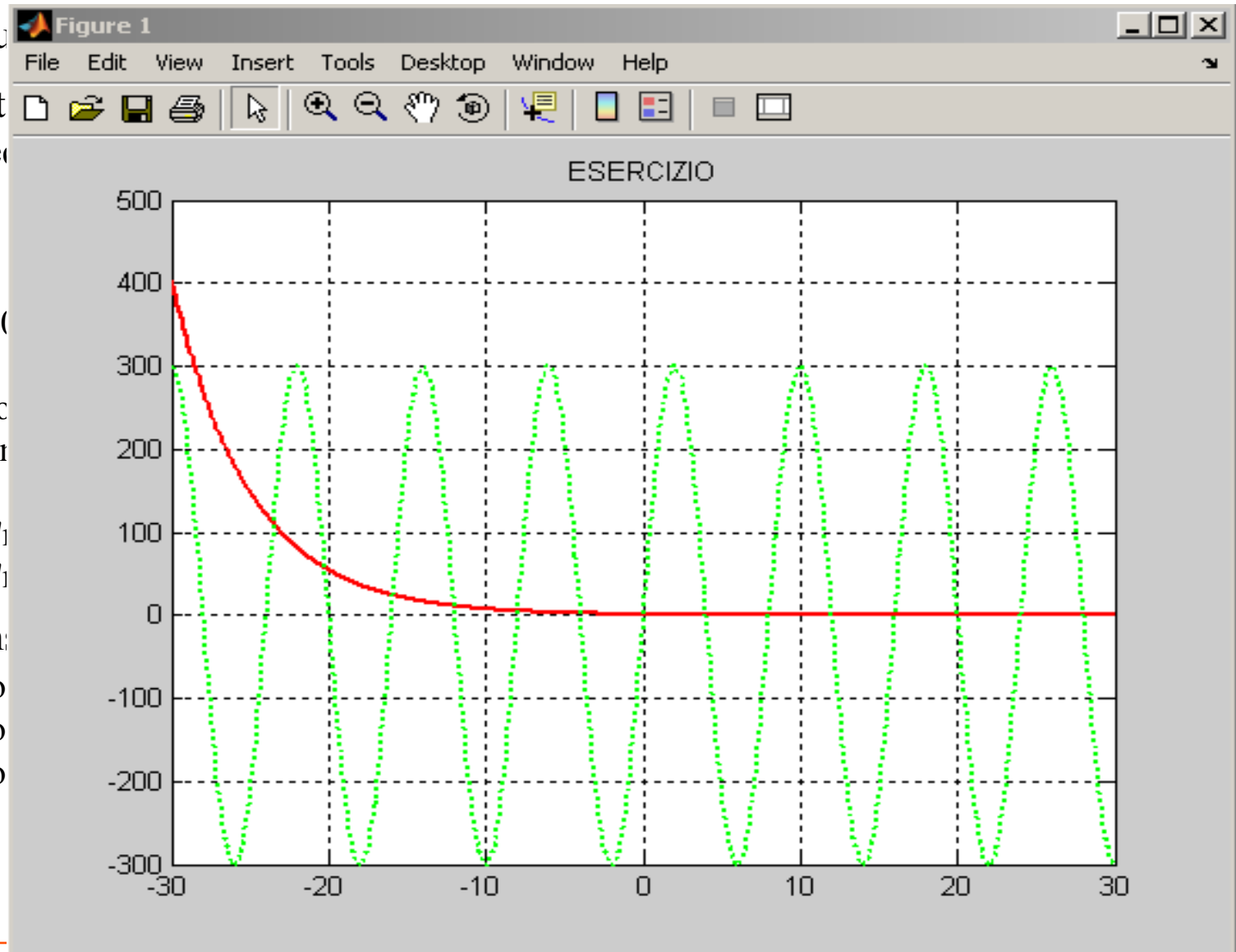
```
-0.0361
```

Visualizzazione grafici

- ❑ La funzione *plot* produce un grafico
- ❑ E' possibile rappresentare più funzioni nello stesso grafico, aggiungere etichette per gli assi, e titoli
- ❑ Esempio:

```
» Tempo = [-30:10:30];  
» Tau = -5;  
» y = exp(Tempo/Tau);  
» z = 300*sin(Tempo);  
» plot(Tempo,y);  
» plot(Tempo,y,'r');  
» plot(Tempo,y,'r');
```

- ❑ Altre funzioni che consentono di creare:
 - ❑ loglog grafico a scala log-log
 - ❑ semilogx grafico a scala log-lineare
 - ❑ semilogy grafico a scala log-lineare



Comandi fondamentali per i grafici

□ Altri tipi di grafico

- bar grafico a barre
- stairs grafico a scala
- mesh grafico 3D
- surf disegna la superficie
- contour disegna le curve di livello

□ Altre funzioni di utilità sono

- title inserisce in una figura il titolo in alto al centro
- xlabel impostazione dell'etichetta dell'asse x
- ylabel impostazione dell'etichetta dell'asse y
- grid grid on visualizza la griglia sulla figura
- axis comando che gestisce gli assi della figura
- hold comando per gestire il mantenimento di una specifica figura
- clf pulisce la figura
- subplot inserisce in una figura più grafici con più assi

Sovrapporre grafici

Per sovrapporre dei grafici nello stesso asse usare il comando **hold**, che permette di non cancellare i grafici già presenti nell'asse considerato.

```
>> hold on
```

```
>> hold off
```

Esempio:

```
>> x = 1:9;
```

```
>> y = sin(x);
```

```
>> ye = abs(rand(1,9));
```

```
>> errorbar(x, y+ye/2, ye, 'o')
```

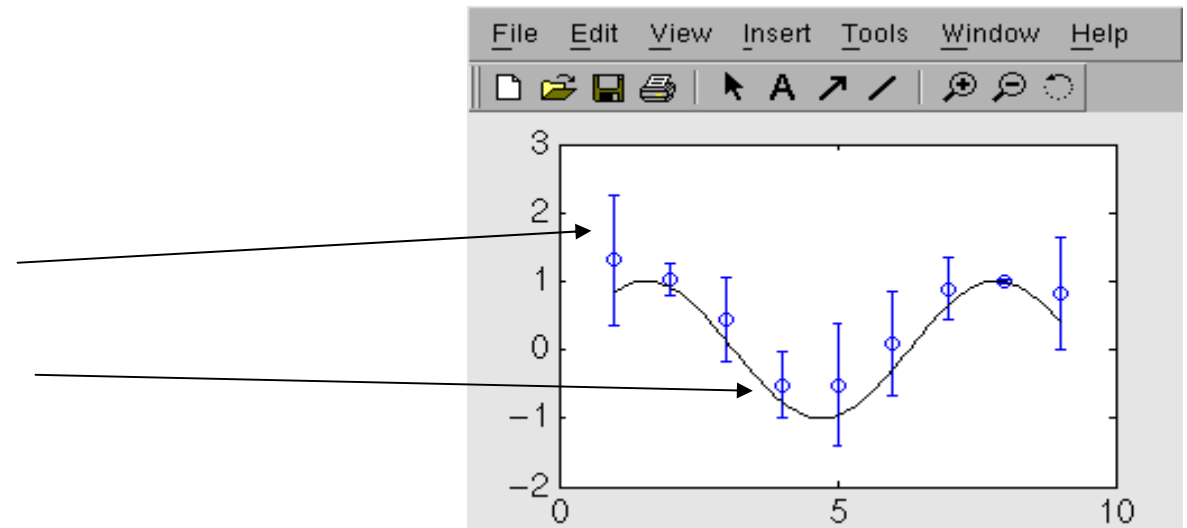
```
>> hold on
```

```
>> plot(1:0.1:9,sin(1:0.1:9),'k')
```

```
>> hold off
```

Nota:

Il comando hold cambia la proprietà '**NextPlot**' dell'asse e della figura corrente.



Contour plot

Le funzioni sono: **contour**, **contourf**. La funzione **contourf**, si differenzia dalla **contour** solo perché le curve di livello sono riempite. Uso:

```
>> contour(x, y, z, nc)
```

```
>> pcolor(x, y, z)
```

Se **nc** è un numero indica il numero di contorni, se è un vettore, i livelli a cui devono essere disegnati i contorni. Esempio:

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

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

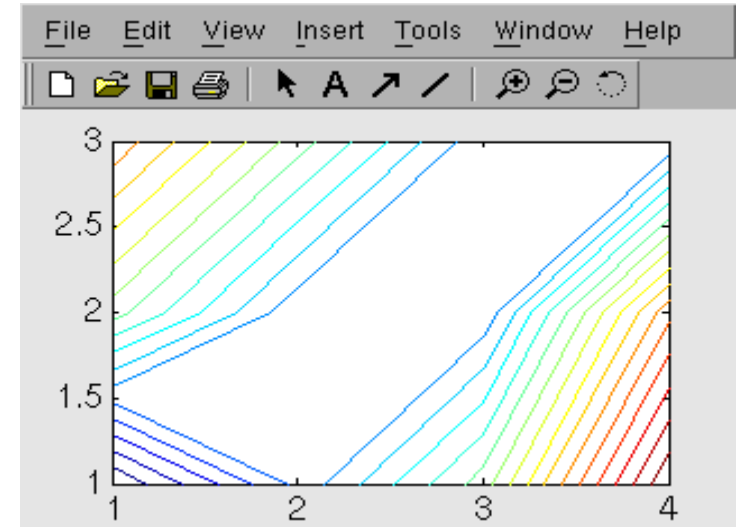
```
>> z = [4,5,6,8; 6,5,5,7; 7,6,5,5]
```

```
z = 4  5  6  8
     6  5  5  6
     7  6  5  5
```

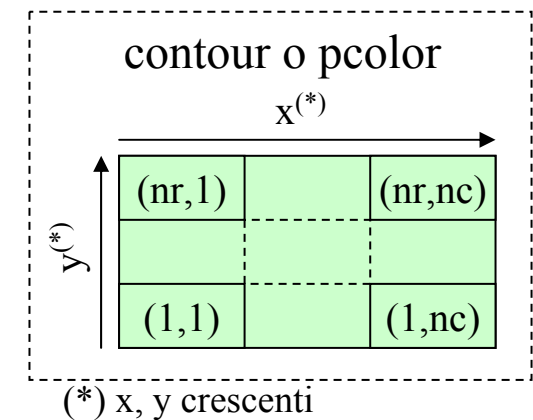
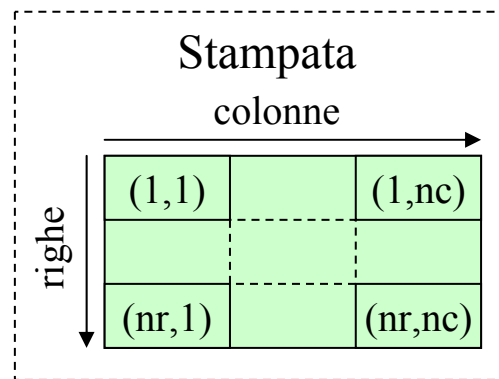
```
>> whos
```

Name	Size	Bytes	Class
x	1x4	32	double array
y	1x3	24	double array
z	3x4	96	double array

```
>> contour(x,y,z,20)
```



Attenzione all'orientamento della matrice **z**. La coordinata **y** scorre lungo le colonne.



Label, leggende, etc.

```
>> x=0:0.1:20;
```

```
>> plot(x,sin(x),x,cos(x))
```

```
>> xlabel('Tempo')
```

```
>> ylabel('Valore')
```

```
>> title('Seni e Coseni')
```

```
>> legend('Seno','Coseno')
```

```
>> grid on
```

```
>> ylim([-1.1,1.3])
```

```
>> xlim
```

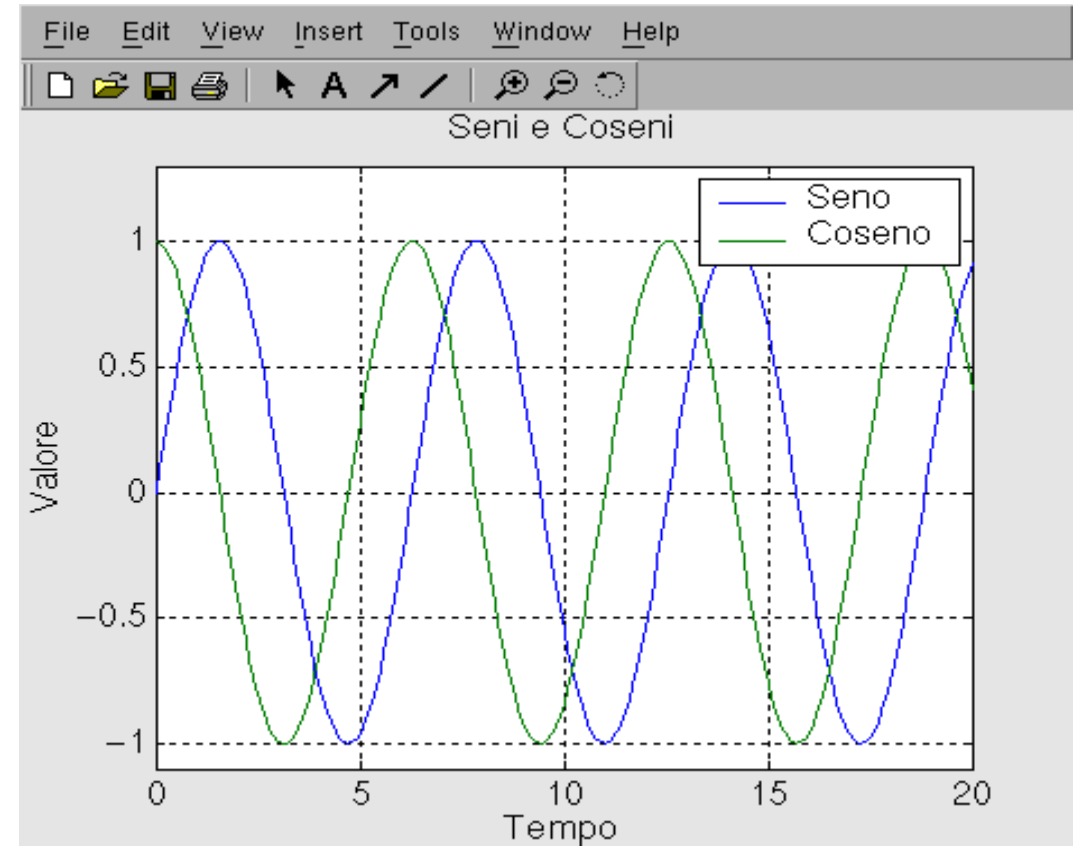
```
ans =
```

```
0 20
```

```
>> axis
```

```
ans =
```

```
0 20.000 -1.1000 1.3000
```



Il comando **axis** permette di impostare diverse proprietà dell'asse corrente.

Gli handle

- ❑ Ogni oggetto grafico ha assegnato un handle, ovvero un numero reale che si riferisce a quell'oggetto.
- ❑ Il numero **0** è l'handle del'oggetto **root** che descrive lo schermo e alcune caratteristiche generali di matlab.
- ❑ Normalmente gli handle delle figure sono numeri interi a partire da **1**.
- ❑ Attenzione! Non confondere le figure (che appunto hanno gli handle) con le finestre del sistema di sviluppo interattivo.

```
>> h = figure
```

```
h = 1
```

```
>> gcf
```

```
ans = 1
```

```
>> hp = plot([1,2],[1,1])
```

```
hp = 3.0010
```

```
>> gca
```

```
ans = 101.0009
```

handle della figura: è un numero intero

handle della linea disegnata

handle del nuovo asse

molte funzioni grafiche se chiamate con dei parametri in uscita, ritornano degli handle agli oggetti grafici appena creati.

Le due funzioni:

gcf Get Current Figure

gca Get Current Axes

ritornano la figura e l'asse corrente

Lettura e scrittura su file

Matlab mette a disposizione diversi comandi per la lettura e la scrittura su file ma i più facili da usare sono Load e Save

❑ Load : legge un file la sintassi è `load filename`

❑ Save : salva un file nella cartella del percorso corrente la sintassi è `save filename`

Esempio : **esempi_save_load.m**

Cell array

```
>> a = {1, 'Ciao', [1,2,3]}
```

```
a =
```

```
[1] 'Ciao' [1x3 double]
```

array di celle: può contenere oggetti di diversa natura. Si creano con le parentesi graffe.

```
>> b = a(2);
```

è ancora un array di celle composto di una sola cella.

```
>> c = a{2};
```

è l'elemento nella cella

Per inserire un elemento in un array di celle preesistente:

```
>> a(2) = {'FTU'}
```

```
>> a{2} = 'FTU'
```

```
>> a(2:3) = {'FTU',123}
```

```
>> a{2:3} = ?? errato
```

```
>> whos
```

Name	Size	Bytes	Class
a	1x3	400	cell array
b	1x1	128	cell array
c	1x4	8	char array

```
>> st = {'pippo','pluto','topolino'};
```

specialmente utile con stringhe di diversa lunghezza

Strutture

```
>> a.x = 1;  
>> a.y = [1,2,3];  
>> a.s = 'Ciao'
```

```
a =
```

```
  x: 1
```

```
  y: [1 2 3]
```

```
  s: 'Ciao'
```

```
>> a(2).x = 'pluto'
```

```
a =
```

```
1x2 struct array with fields:
```

```
  x
```

```
  y
```

```
  s
```

```
>> a(1).y
```

```
ans =
```

```
 1  2  3
```

Un'unica variabile **a** è composta di diversi campi (**x**, **y**, **s** nell'esempio), che possono essere variabili di qualsiasi tipo.

Si può avere un array di strutture.

Stessa sintassi per leggere il valore di un campo

Cosa è un m-file

- ❑ Un M-file è un file ASCII che rispetta la sintassi MATLAB.
- ❑ Un utente può aggiungere delle nuove istruzioni aggiungendo i propri M-files, come < nomefile >.m. Essi verranno eseguiti semplicemente digitando < nomefile >. Es. *pippo.m* nella command window digitare solo pippo per eseguire il programma.
- ❑ I toolboxes di MATLAB sono delle collezioni di M-files che risolvono particolari problemi (Es. Spline toolbox; Fuzzy toolbox etc).
- ❑ Il debug è fondamentale in qualsiasi programma, in Matlab è facilissimo da usare.

Differenza tra script e function

Script

- ❑ Opera su dati presenti nel workspace
- ❑ Non accetta variabili in input.
- ❑ Non ha variabili di output.
- ❑ Utile per automatizzare una serie di istruzioni che si devono eseguire più volte.

Function

- ❑ Le variabili interne sono locali.
- ❑ Può accettare variabili in input.
- ❑ Può avere variabili in output.
- ❑ Utile per estendere il linguaggio MATLAB alle applicazioni personali.

Script

- ❑ E' il tipo più semplice di M-file perchè non ha variabili di input e output.
- ❑ Serve per **automatizzare** una serie di comandi MATLAB che devono essere eseguiti più volte.
- ❑ Opera sui dati esistenti nell'ambiente di lavoro di base, oppure può creare nuovi dati o caricarne dall'esterno. Es. : *load dati_DO.txt*
- ❑ I dati che vengono generati rimangono nell'ambiente di lavoro di base e possono essere riutilizzati per altri calcoli oppure salvati su specifici file Es. : *save dati_T.txt*

Un script viene eseguito semplicemente digitando il suo nome senza l'estensione m.

Schema di uno script (linee guida)

Nelle prime righe è utile descrivere cosa farà il programma che segue
Es. Nello script `esempi_save_load.m` la prima riga è
%Esempi per esercitarsi con il save il load e i vari tipi di variabili di Matlab; questo commento verrà visualizzato tutte le volte che digiterete *help esempi_save_load* sulla command window

Corpo dell'algoritmo:

- ❑ può contenere cicli, assegnazioni, chiamate a function predefinite di matlab oppure create da voi
- ❑ visualizzazione grafica dei risultati (opzionale)
- ❑ Salvataggio dei risultati (opzionale)






N.B : La parte nobile di un programma sono i commenti, quindi i *programmi vanno commentati*

Debug di un m-file

Il Matlab Debugger ti aiuta ad identificare gli errori della programmazione nel codice Matlab. Usando il Debugger, si può:

- ❑ Vedere il contenuto del workspace in ogni momento durante l'esecuzione del programma.
- ❑ Eseguire m-file riga per riga
- ❑ Sapere quali sono via via le funzioni chiamate

Breakpoints: sono dei puntini rossi che si possono collocare lungo il codice in modo da visualizzare il contenuto del workspace da quel punto del programma in poi.

	Set/Clear Breakpoint	Inserisce o cancella i breakpoint
	Step In	Esegue la linea corrente del m-file se trova una function ci entra
	Single Step	Esegue la linea corrente
	Continue	Continua ad eseguire fino alla fine del programma o finchè non trova un altro breakpoint
	Quit Debugging	Esce dalla modalità debugging

Come si scrive una Function?

- ❑ Vengono definite attraverso M-file, detti file di funzione.
- ❑ Nei file di funzione tutte le variabili sono locali: ossia i loro valori sono disponibili solo all'interno della funzione.
- ❑ Essi sono particolarmente utili quando occorre ripetere una serie di comandi più volte.

Definizione della funzione

La prima riga di un file di funzione deve iniziare con la definizione della funzione con la quale:

- ❑ si distingue il file come file di funzione dai file script
- ❑ si nomina la funzione
- ❑ si definisce l'elenco dei parametri di input e di output

Come si scrive una Function?

Le variabili di input specificate nella riga di definizione della funzione sono locali per quella funzione, questo:

- ❑ consente di usare altri nomi di variabili quando viene chiamata la funzione
- ❑ fa sì che tutte le variabili all'interno della funzione vengono cancellate dopo che la funzione è stata eseguita, tranne le variabili che figurano nell'elenco della variabili di output utilizzate nella chiamata di funzione oppure le variabili definite *global*

Riga di definizione

function [output] = nome_function (input)

Output una sola variabile in uscita x: [output] → x

più variabili in uscita x, y, z: [output] → [x,y,z]

nessuna variabile in uscita: [output] → []

Input

Le variabili in input possono essere array (scalari, vettori, matrici) ma anche il nome di altre function:

Sintassi delle Function

Occorre digitarla nel file principale in corrispondenza della riga di programma nella quale si vuole richiamare la function

[var. di output]=nome_funzione(var. di input)

- ❑ Le variabili di output devono essere racchiuse tra parentesi quadre
- ❑ Le variabili di input devono essere racchiuse tra parentesi tonde
- ❑ Il nome della funzione deve essere uguale al nome del file con estensione .m, in cui sarà salvata la funzione.

Esempio function

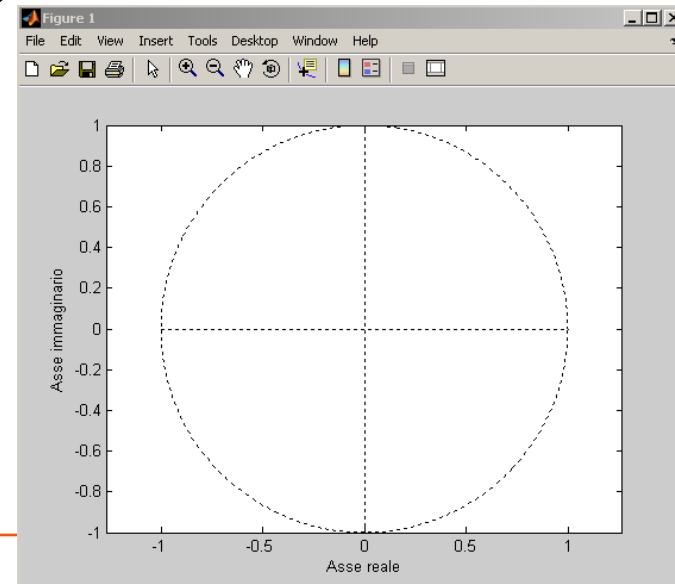
```
function unit_circle(xlim,ylim)
%Draw unit circle
angolo=(0:0.01:2*pi);
x=cos(angolo);
y=sin(angolo);
plot(x,y,':k')
axis([-xlim xlim -ylim ylim]);
axis equal
xlabel('Asse reale')
ylabel('Asse immaginario')
hold on
plot([0 0],[-ylim ylim],':k')
plot([-xlim xlim],[0 0],':k')
hold on
axis(axis)
```

Intestazione funzione con 2 ingressi e nessuna uscita numerica, solo grafica

crea un vettore

Plotta il cerchio unitario

Plotta gli assi tratteggiati



Istruzioni di controllo

- ❑ *for* ripetizione di un insieme di istruzioni per un numero predeterminato di interazioni (deve terminare con *end*)
- ❑ *while* ripetizione di un insieme di istruzioni finchè una determinata condizione rimane vera (deve terminare con *end*)
- ❑ *if* istruzione condizionale (deve terminare con *end*) può utilizzare *else* e *elseif*
- ❑ *else* identifica un blocco di istruzioni alternative
- ❑ *elseif* esegue un blocco di istruzioni se è soddisfatta una condizione alternativa
- ❑ *end* termina le istruzioni *if, for e while*
- ❑ *break* termina l'esecuzione di un ciclo for o while
- ❑ *switch* indirizza il controllo di un programma confrontando l'espressione di input con le espressioni associate alle clausole *case*
- ❑ *case* utilizzato con *switch* per gestire l'esecuzione di un programma

Operatori relazionali

- $<$ minore
- \leq minore o uguale
- $>$ maggiore
- \geq maggiore o uguale
- $==$ uguale
- \sim diverso

P.S Il simbolo \sim si digita tenendo premuto alt e digitando 126 sul tastierino numerico

Operatori logici e funzioni logiche

Matlab ha 4 operatori logici o booleani e una serie di funzioni logiche

Operatori logici

1. \sim NOT l'istruzione $\sim A$ rende una array delle stesse dimensioni di A; con elementi pari a 1 se quelli corrispondenti di A sono nulli altrimenti sono pari a 0
2. $\&$ AND l'istruzione $A \& B$ restituisce un array delle stesse dimensioni di A e B; gli elementi del nuovo array sono pari a 1 se i corrispondenti elementi di A e B sono entrambi diversi da 0 altrimenti sono pari a 0.
3. OR l'istruzione $A | B$ rende una array delle stesse dimensioni di A e B; con elementi pari a 1 se almeno 1 dei due elementi corrispondenti di A e B è diverso da 0; e pari a 0 se entrambi gli elementi di A e B sono nulli.
4. $\text{xor}(A,B)$ OR esclusivo l'istruzione di $\text{xor}(A,B)$ restituisce un array delle stesse dimensioni di A e B; gli elementi del nuovo array sono pari a 1 se uno solo dei due elementi corrispondenti di A e B è diverso da 0 (non entrambi); sono pari a 0 se entrambi gli elementi di A e B sono nulli o diversi da 0.

Funzioni logiche

- ❑ *any(x)* restituisce uno scalare, che è pari a 1 se almeno uno degli elementi del vettore x è diverso da zero, 0 negli altri casi
- ❑ *any(A)* restituisce un vettore riga che ha lo stesso numero di colonne della matrice A e che contiene 1 e 0 in funzione del fatto che la corrispondente colonna di A contiene oppure almeno un elemento diverso da 0.
- ❑ *all(x)* restituisce uno scalare, che è pari a 1 se tutti gli elementi del vettore x sono diversi da 0, 0 negli altri casi
- ❑ *find(x)* crea un array che contiene gli indici degli elementi non nulli degli array x $[u,v,w]=\text{find}(A)$ crea gli array u e v che contengono gli indici delle righe e delle colonne degli elementi non nulli della matrice A , e l'array w che contiene i valori degli elementi non nulli. L'array w può essere omesso.
- ❑ *finite(A)* restituisce un array della stessa dimensione di A i cui elementi sono pari a 1 se i corrispondenti elementi di A sono valori finiti, 0 negli altri casi.
- ❑ *isnan(A)* restituisce un array della stessa dimensione di A i cui elementi sono pari a 1 se i corrispondenti elementi di A sono pari a "NaN" (Not a Number, numero non definibile), 0 negli altri casi
- ❑ *isinf(A)* restituisce un array della stessa dimensione di A i cui elementi sono pari a 1 se i corrispondenti elementi di A sono pari a "INF", 0 negli altri casi
- ❑ *isempty(A)* restituisce 1 se A è una matrice vuota, 0 altri casi.
- ❑ *isreal(A)* restituisce 1 se A non contiene elementi parti immaginarie, 0 negli altri casi

Esempio script Matlab

.....\esempi_matlab\Sistemi_td\Sist_td_Eigval.m

Esempio cercare gli zeri di una funzione

Calcola gli zeri di una funzione reale di variabile reale con la seguente sintassi

$$[x, fval] = fzero('fun', x0)$$

- ❑ Input fun nome della function che contiene la funzione f
- ❑ Input x0 iniziale per la ricerca dello zero
- ❑ Output x approssimazione dello zero calcolato
- ❑ Output fval valore di f in x.

Si possono ottenere delle informazioni complete sulle iterazioni usando il comando

- ❑ `fzero('fun',x0, optimset('disp','iter'))`

fminsearch

Il comando fminsearch permette di trovare un minimo locale di un problema non lineare non vincolato del tipo

$$\begin{cases} \min f(x) \\ x \in \mathbb{R}^n \end{cases} \quad \text{dove } f : \mathbb{R}^n \rightarrow \mathbb{R} \quad \text{f è la funzione obiettivo}$$

La forma più semplice del comando è `x=fminsearch('f',x0)` dove `f` è una stringa che contiene il nome della funzione e `x0` è il punto di partenza che utilizza l'algoritmo.

Esempio

fun_pippo.m

```
function y = fun_pippo(x)
y=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

Linea di comando

```
>> x=fminsearch('fun_pippo',[0;0])
```

```
>> x =
```

```
1.0000
```

```
1.0000
```

Introduzione al Simulink

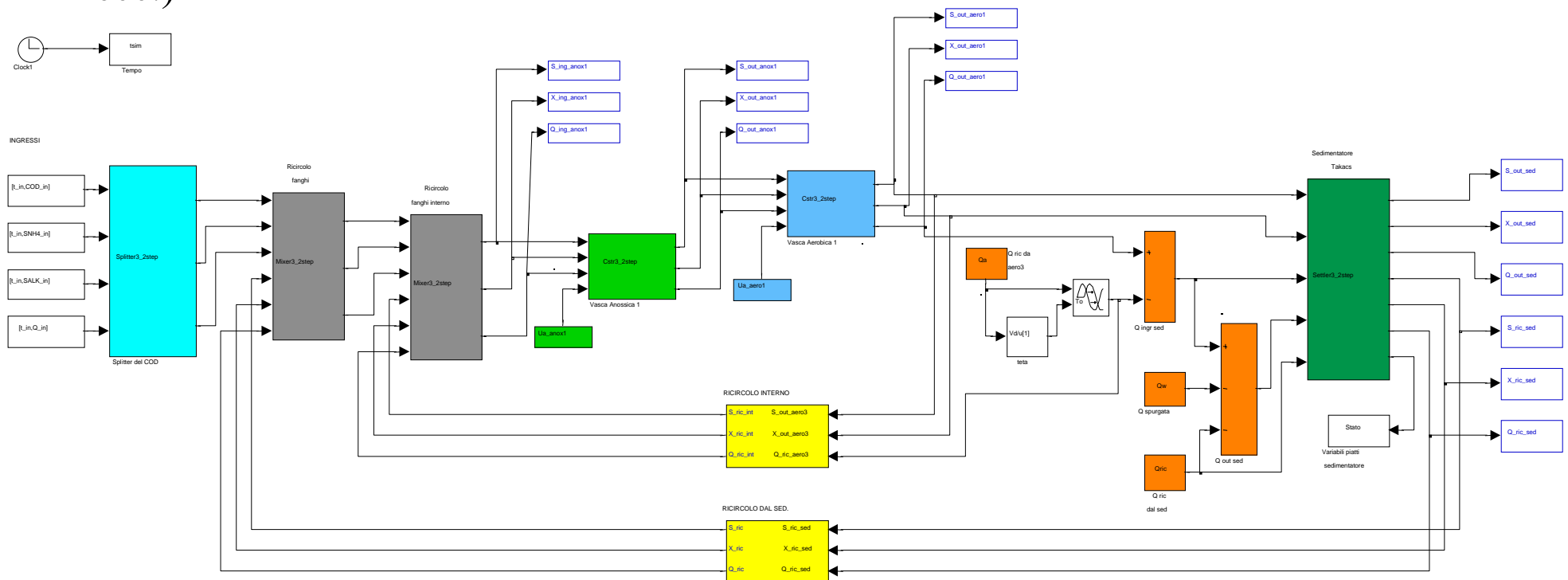


L'ambiente Simulink

- ❑ Simulink è un **ambiente grafico** per la simulazione di sistemi complessi
- ❑ Simulink è composto da una **libreria di blocchi** che descrivono elementi statici e dinamici elementari
- ❑ L'utente compone lo schema a blocchi del sistema da simulare mediante l'interconnessione dei blocchetti elementari
- ❑ Simulink genera automaticamente le equazioni e risolve il problema numerico di simulazione desiderato.
- ❑ I modelli costruiti in Simulink possono essere modelli gerarchici: ogni blocco del sistema può essere a sua volta un sottosistema complesso.
- ❑ Simulink interagisce con Matlab attraverso il Workspace; i modelli Simulink possono contenere variabili del Workspace; allo stesso modo il risultato delle simulazioni può essere esportato nel Workspace e analizzato con Matlab.

La forza del simulink stà nella modularietà

- Modularità: un sistema complesso può essere rappresentato come interconnessione di svariati sottosistemi. Ciascuno di essi può a sua volta (in maniera ricorsiva) essere composto da sottosistemi di complessità via via inferiore, sino ad arrivare a blocchi che descrivano sottosistemi elementari, cioè descritti da una sola relazione matematica (equazione differenziale oppure alle differenze, un'equazione algebrica ecc.)

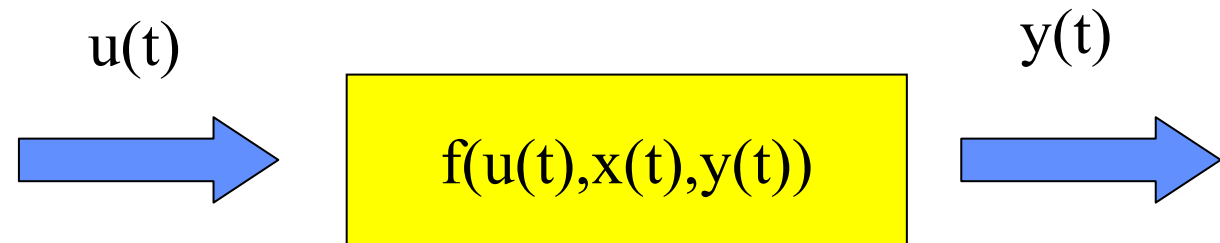


Blocco elementare

Ogni blocco rappresenta un sistema dinamico elementare.

Un blocco comprende le seguenti grandezze (non sempre sono tutte presenti):

- un insieme di ingressi
- un insieme di stati
- un insieme di uscite



Le grandezze di uscita sono una funzione del tempo e delle grandezze di ingresso e degli stati del sistema.

$$y(t) = f(u(t), x(t), t)$$

Si parla di blocco continuo se l'uscita è una funzione continua del tempo. Si parla di blocco discreto se il segnale di uscita è valutato solo in specifici istanti.

Metodi di integrazione numerica

- ❑ I metodi numerici per la soluzione di una generica equazione differenziale individuano all'interno dell'intervallo di integrazione un certo numero di “istanti di integrazione” $T_0=t_1 < t_2 \dots < t_n=T_f$ e calcolano la soluzione $x(t_i)$ in corrispondenza di questi punti.

$$\dot{x} = f(x, t) \quad t \in [T_0, T_f] \quad x(T_0) = x_0$$

- ❑ La soluzione all'istante t_{i+1} è determinata sulla base delle soluzioni calcolate negli istanti precedenti, con una formula del tipo:

$$x(t_{i+1}) = x(t_i) + h \Delta(t_i, x, h, f)$$

- ❑ con $h_i = t_{i+1} - t_i$ è il passo di integrazione
- ❑ Δ è un'opportuna funzione

Metodi di integrazione numerica

I vari metodi di integrazione si distinguono per una diversa funzione Δ e un'opportuna scelta di h .

- ❑ Metodi a passo fisso: h è costante
- ❑ Metodi a passo variabile: h è scelto in modo opportuno ad ogni passo.

Trovata la soluzione x_{i+1} , viene valutato l'errore nei confronti della soluzione esatta. Se questo supera il limite prefissato si riduce h e si ricalcola la soluzione. Se l'errore è inferiore al passo successivo h viene incrementato. In questo modo i tempi di calcolo si riducono.

Metodi di integrazione numerica

Simulink possiede diversi metodi di integrazione, sia a passo fisso che a passo variabile.

Metodi a passo variabile

- ode45(default) non appropriato per sistemi stiff(RungeKutta4,5)
- ode23più veloce, ma meno preciso di ode45, anche per stiff
- ode113 per soluzioni accurate, ma più lento
- ode15s efficiente per sistemi stiff
- ode23s meno preciso, ma più efficiente del precedente
- ode23t per sistemi moderatamente stiff
- ode23tb meno preciso di ode15s
- discrete (default) per sistemi discreti

Metodi a passo fisso

- ode5(default) versione a passo fisso della ode45
- ode4 Metodo di RungeKuttadel 4 ordine
- ode3 versione a passo fisso della ode23
- ode1 metodo di Eulero

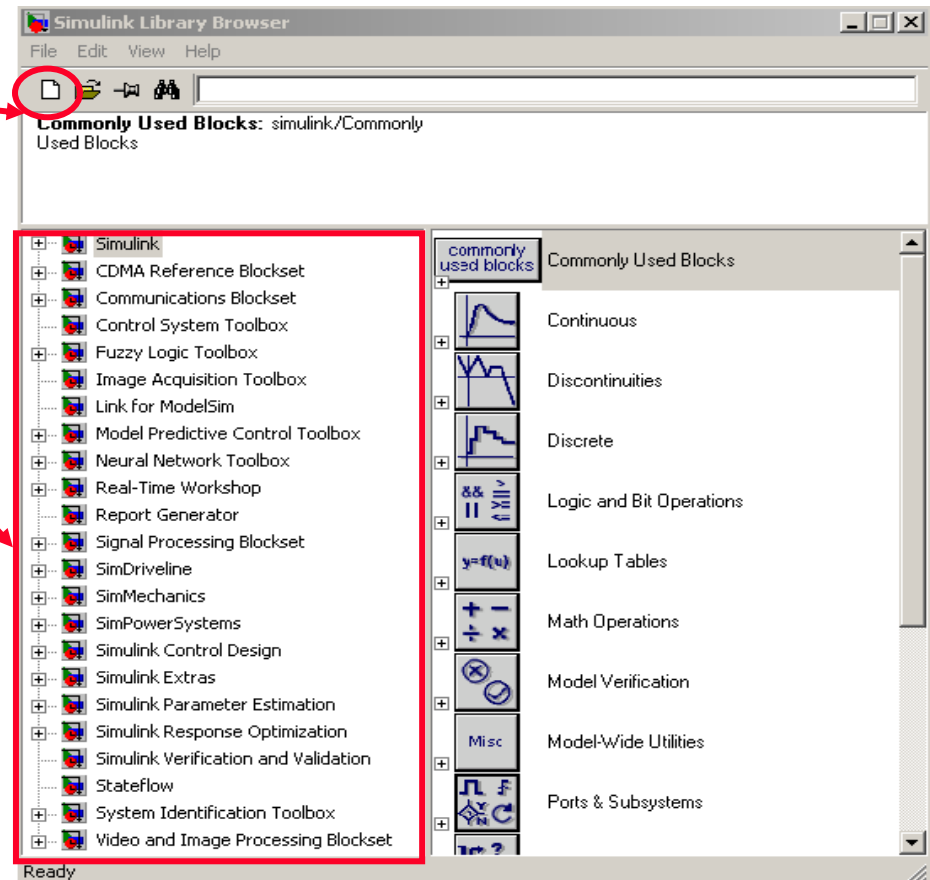
L'interfaccia grafica

- ❑ Digitando simulink sul prompt di Matlab appare la libreria grafica
- ❑ Da qui è possibile creare un nuovo modello (foglio bianco) e comporre il sistema da simulare mediante i diversi blocchi elementari già disponibili.

Crea nuovo modello

Libreria grafica
Comprende anche blocchi
elementari già pronti all'uso

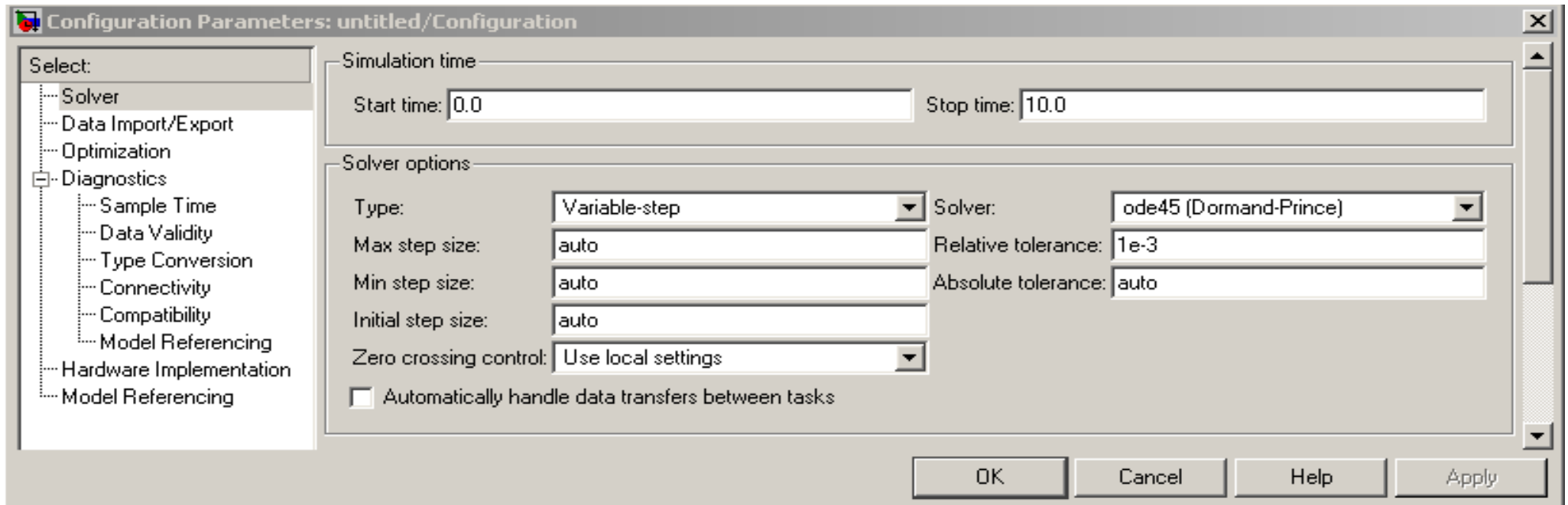
I blocchi-base sono raggruppati in categorie. La maggior parte dei blocchi che useremo è contenuta nella categoria Simulink



Parametri della simulazione

L'utente deve definire:

- Istanti di inizio e fine della simulazione;
- Tipo di solutore numerico (se il problema richiede metodi particolari);
- Parametri del solutore.



Parametri della simulazione

- ❑ **Start time** istante di inizio
- ❑ **Stop time** istante di fine
- ❑ **Type** metodo a passo fisso o a passo mobile

In presenza di un metodo a passo variabile

- ❑ **Max step size**: è la massima ampiezza del passo di integrazione. Se è auto allora è $1/50$ dell'intervallo di integrazione. Occorre fissarlo minore di $0.1 \tau_{\max}$ (costante di tempo più grande).
- ❑ **Min step size**: è la minima ampiezza del passo di integrazione. Occorre fissarlo minore di $0.01 \tau_{\min}$ (costante di tempo più piccola).
- ❑ **Relative tolerancee Absolute tolerance**: definiscono il massimo errore relativo e assoluto.

In presenza di un metodo a passo fisso

- ❑ **Fixedstep size**: è l'ampiezza del passo di integrazione. Il valore di default auto è pari ad un cinquantesimo dell'intervallo di integrazione.

Simulink lanciato da Matlab

- ❑ Un programma simulink può essere lanciato da uno script o una function (m-file)
- ❑ I vantaggi consistono nel settaggio di tutte le variabili nello script stesso
- ❑ La sintassi è molto semplice: supponiamo di avere un simulink che si chiama “Modello.mdl”
- ❑ Lo script in matlab dovrà contenere il seguente comando:

```
[T,X,Y]=sim('Modello',Timespan,options,UT)
```

Dove

T restituisce il vettore dei tempi dopo la simulazione

X restituisce la matrice degli stati

Y restituisce le uscite

Timespan [Tfinal]

[Tinit Tfinal]

Options da simset

UT (opzionale) input esterni

Esempio Lancio Simulink da Matlab

```
%Lanciare una logistica fatta su simulink da matlab e visualizzare i  
%risultati in un plot
```

```
clc
```

```
clear
```

```
close all
```

```
global r K
```

```
r=1.03; %rateo di crescita
```

```
K=23; %capacità portante
```

```
y0=8; %popolazione iniziale
```

```
tempofin=10;
```

```
% implementazione in Simulink
```

```
[T_sim,X_sim,Y_sim]=sim('logistica_sim',[0 tempofin]);
```

```
hold on % tengo attivo il disegno precedente sulla figura(1)
```

```
plot(T_sim,OUT_logistica,'m')
```

```
legend('logistica calcolata con Simulink','Location','SouthEast')
```

```
grid on
```

