

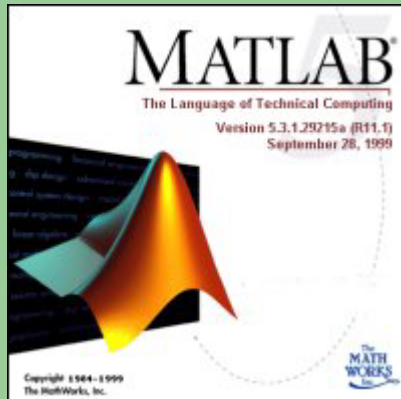
Introduzione all'utilizzo di Matlab® e Simulink®

Ing. Andrea Paoli

apaoli@deis.unibo.it

www-lar.deis.unibo.it/~apaoli

Tel. (051-20) 93045

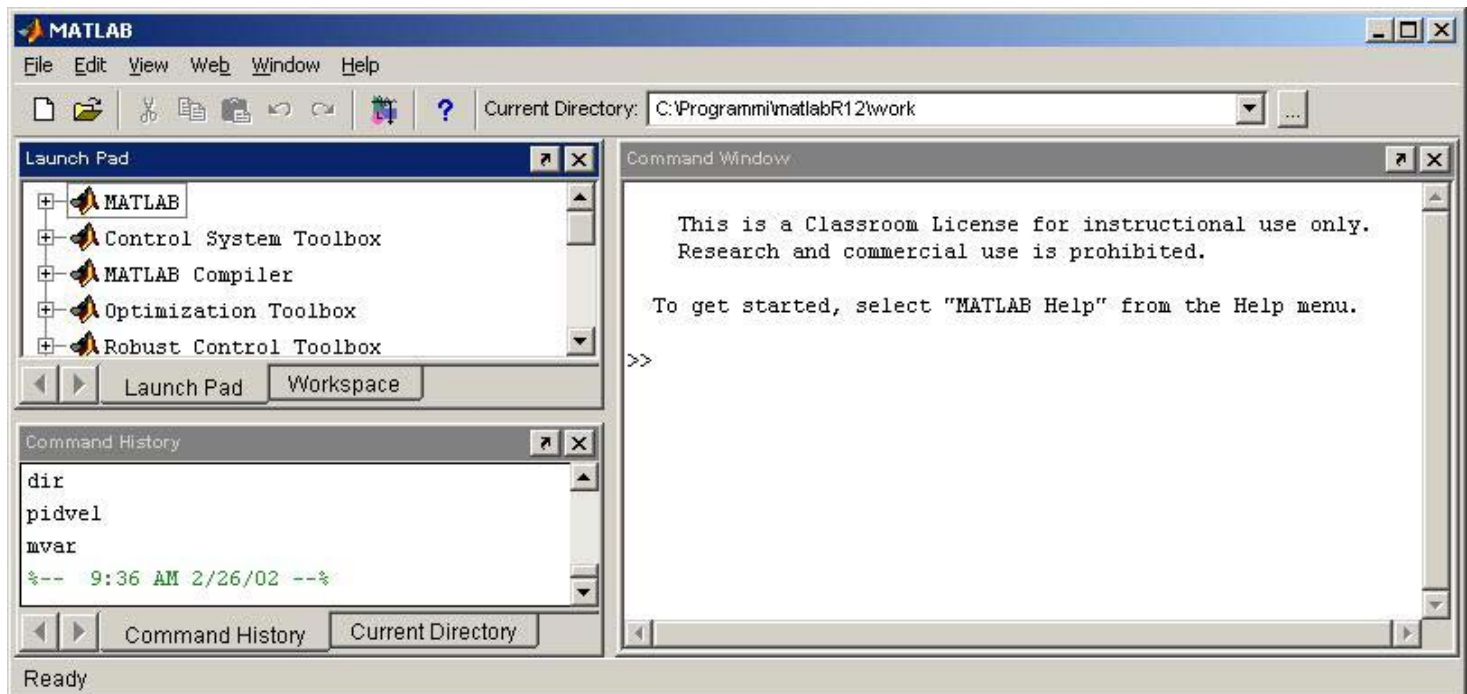


Cosa è Matlab® ?

- Matlab (= Matrix Laboratory) è un linguaggio di programmazione per applicazioni scientifiche e numeriche.
- Vasto set di funzioni predefinite.
- Interprete di comandi.
- Possibilità di scrivere nuove funzioni.
- Librerie di Toolbox per svariate applicazioni (Control System Toolbox, Signal Processing...)

L'interfaccia Matlab®

La command window dà accesso diretto all'interprete (scrittura diretta dei comandi).



Matlab® come calcolatrice.

Per valutare espressioni aritmetiche.

Esempio: calcolare $4 + \sqrt{2} - \sin(0.2\pi)^2 + e^2$

al prompt digitare

```
>> 4+sqrt(2)-sin(0.2*pi)^2+exp(2)
```

```
ans =
```

```
12.4578
```

Il risultato viene scritto nella variabile “ans”.

Definizione di variabili.

- E' possibile definire variabile ed espressioni più complesse:

```
>> a=4; b=2;
```

```
>> a*b
```

```
ans =
```

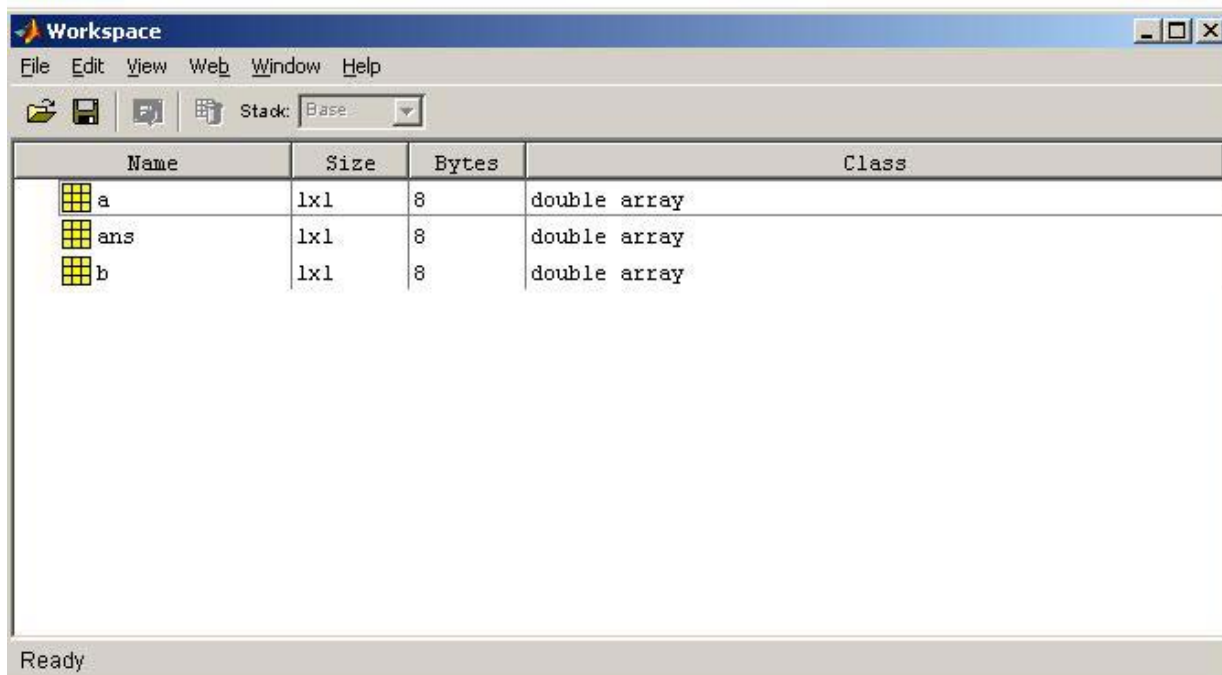
```
8
```

- Per cancellare una variabile (es. a):

```
>> clear a
```

Il workspace.

Ogni variabile definita in questo modo viene conservata in memoria nel workspace.



Letture e scrittura su file.

- Mediante i comandi load e save è possibile salvare su file le variabili del workspace:

`save nomefile variabile1 variabile2 ...` scrive nel file `nomefile.mat` le variabili elencate.

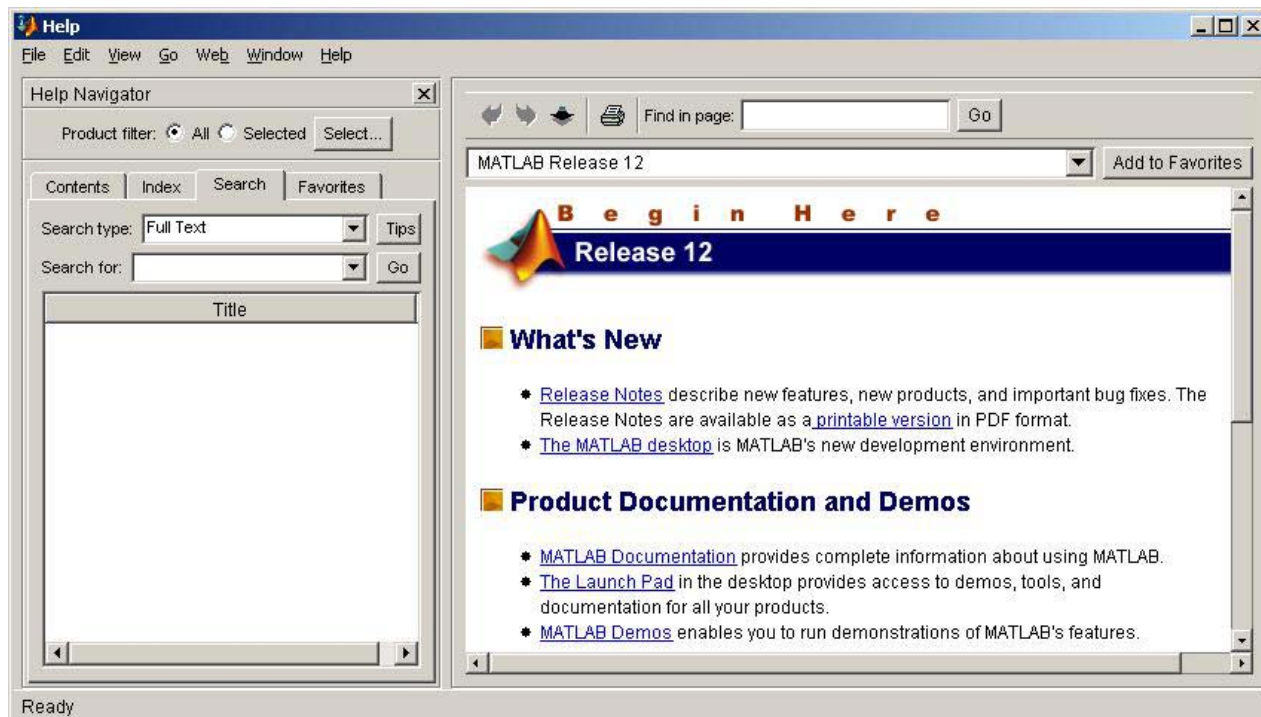
`load nomefile variabile1 variabile2 ...` carica dal file `nomefile.mat` le variabili elencate.

`save nomefile` salva tutto il workspace in `nomefile`.

`load nomefile` carica tutte le variabili in `nomefile`.

Una funzione fondamentale.

Help: fornisce la descrizione completa di tutte le funzioni predefinite ! !



Definizione di matrici.

- Definiamo la matrice 2x2: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

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

```
A =
```

```
    1    2  
    3    4
```

- Accedere agli elementi di una matrice:

```
>> A(1,2)
```

```
ans =
```

```
    2
```

Le wildcard.

- Per accedere a intere righe o colonne di una matrice, si usa la wildcard “:”
- Es: selezionare la prima riga di A

```
>> A(1,:)
```

```
ans =
```

```
1    2
```

- Es: selezionare la seconda colonna di A

```
>> A(:,2)
```

```
ans =
```

```
2
```

```
4
```

Selezionare sottomatrici.

- Se defiamo:

```
>> B=[1 , 2 , 3 ; 4 , 5 , 6]
```

```
B =
```

1	2	3
4	5	6

```
>> B(1:2,2:3)
```

```
ans =
```

2	3
5	6

Operazioni elementari.

- Sono definiti gli operatori $+$, $-$, $*$, $^{\wedge}$.
- Matrice trasposta:

```
>> A'
```

```
ans =
```

```
1 3  
2 4
```

- Matrice inversa:

```
>> inv(A)
```

```
ans =
```

```
-2.0000  1.0000  
1.5000 -0.5000
```

Operazioni elementari.

- Determinante:

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

```
ans =
```

```
-2
```

- Autovalori:

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

```
ans =
```

```
-0.3723
```

```
5.3723
```

Altre operazioni.

- **rank**: calcolo del rango di una matrice
- **trace**: calcolo della traccia di una matrice
- **norm**: calcolo della norma di una matrice
- **size**: per conoscere le dimensioni di una matrice

Matrici speciali.

- `eye(n,n)`: matrice identità $n \times n$;
- `zeros(n,m)`: matrice di zeri $n \times m$;
- `ones(n,m)`: matrice di uni $n \times m$;
- `rand(n,m)`: matrice $n \times m$ con elementi distribuiti uniformemente tra 0 e 1.

I vettori.

I vettori hanno due funzioni fondamentali in Matlab:

- rappresentazione dei polinomi (un polinomio è descritto dal vettore dei suoi coefficienti);
- rappresentazione di segnali (un segnale è rappresentato mediante la sequenza dei valori che assume in un insieme di istanti di tempo, quindi mediante un vettore).

Definizione di vettori.

- `>> v=(0:10)`

`v =`

0 1 2 3 4 5 6 7 8 9 10

- `>> v=(1:0.5:3)`

`v =`

1.0000 1.5000 2.0000 2.5000 3.0000

- `>> v=[3 6 1 7]`

`v =`

3 6 1 7

I polinomi e le operazioni.

- Sono definiti come vettori. Es: $3s^2 + 2s + 1$
 `>> pol=[3 2 1]`
 `pol =`
 3 2 1
- Calcolo delle radici (roots).
 `>> roots(pol)`
 `ans =`
 -0.3333 + 0.4714i
 -0.3333 - 0.4714i

I polinomi e le operazioni.

- Valutazione in un punto (**polyval**)

```
>> polyval(pol,0)
```

```
ans =
```

```
1
```

- Prodotto di polinomi (**conv**) $(s+1)(s+1) = s^2 + 2s + 1$

```
>> pol1=[1 1]; pol2=[1 1];
```

```
>> polprod=conv(pol1,pol2)
```

```
polprod =
```

```
1    2    1
```

Gli M-file: script e funzioni.

M-file: file contenente codice Matlab

- Vengono scritti mediante un qualsiasi editor di testo ed eseguiti chiamandoli dalla linea di comando. Due tipi di M-file: *script* e *funzioni*.
- Gli *script* si usano per automatizzare le sequenze di comandi. Quando viene eseguito uno script, l'esecuzione dei comandi è del tutto equivalente alla scrittura del codice con la tastiera. Non hanno argomenti di input e output, tutte le variabili sono globali.

Gli M-file: script e funzioni.

- Le **function** si usano per estendere le capacità di Matlab. Normalmente generano una o più uscite (matriciali) dipendenti dai parametri in ingresso. Le variabili sono locali alla funzione.
- `function [output]=nomefunction(input)`
 istruzioni;
 return;

Programmare in Matlab.

- Matlab è un linguaggio di programmazione

Esistono comandi per il controllo di flusso:

- if...elseif...else...end
- while...end
- for
- switch
- break

Esempio 1.

La grafica in matlab

- `plot(y)` – visualizza gli elementi del vettore `y` rispetto agli indici del vettore stesso;
- `plot(x,y)` – visualizza il vettore `y` vs. il vettore `x`;
- Per visualizzare una qualsiasi funzione $y=f(x)$ in Matlab, è SEMPRE necessario creare i vettori `x` e `y` nel dominio di interesse;

Scegliere la finestra grafica.

- `figure(n)` specifica su quale figura lavoriamo
- `subplot` permette di suddividere la finestra in più grafici, per visualizzare contemporaneamente diversi segnali
- Es.
`figure(1); subplot(211);`
...
`subplot(211);`
...
● `clf` - pulisce la figura corrente

Tracciare il grafico.

- **plot**: plotta il grafico 2-D con scale lineari per entrambi gli assi;
- **loglog**: plotta il grafico 2-D con scale logaritmiche per entrambi gli assi;
- **semilogx**: plotta il grafico 2-D con scala logaritmica per l'asse x e lineare per l'asse y;
- **semilogy**: plotta il grafico 2-D con scala lineare per l'asse x e logaritmica per l'asse y;
- Sintassi: `plot(x1,y1,x2,y2,...)`.

Grafici 3-D.

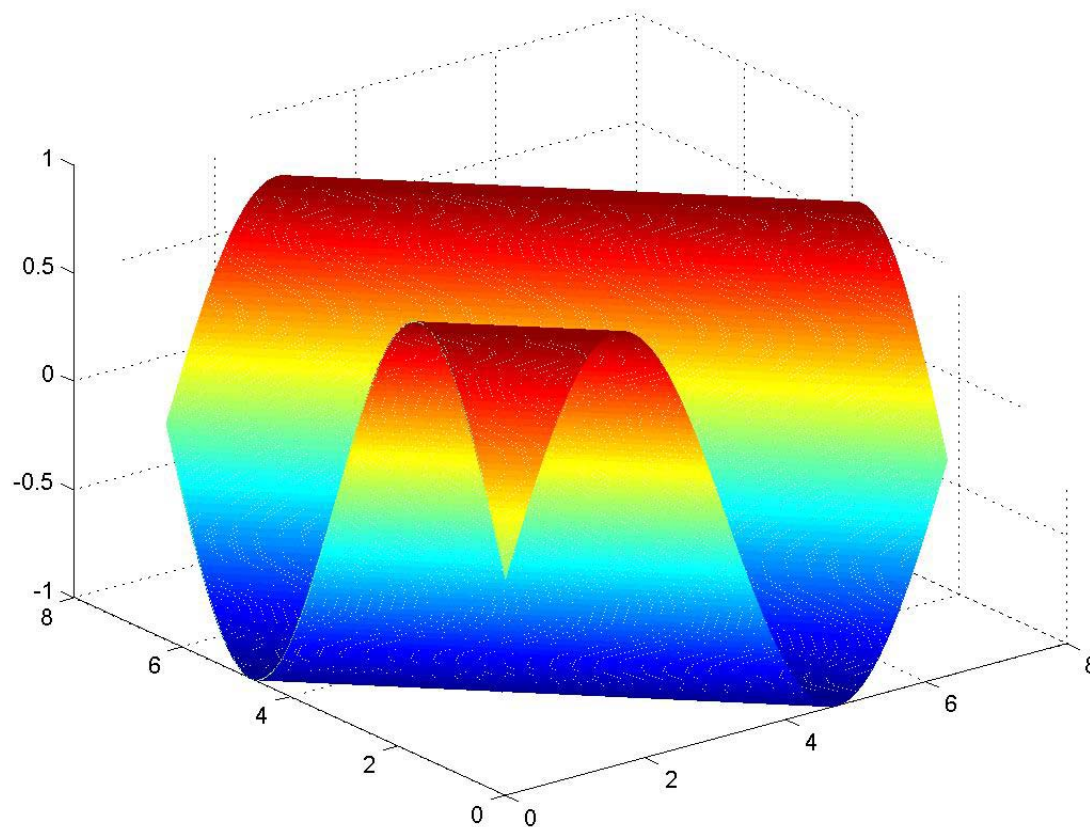
- Possibilità di tracciare grafici tridimensionali (**mesh**).
- ```
>> x=(0:0.01:2*pi); y=(0:0.01:2*pi);
>> for i=1:length(x)
 for j=1:length(y)
 z(i,j)=sin(x(i)+y(j));
 end
 end
>> mesh(x,y,z)
```

# Grafici 3-D.

$$x \in [0, 2\pi];$$

$$y \in [0, 2\pi];$$

$$z = \sin(x + y);$$

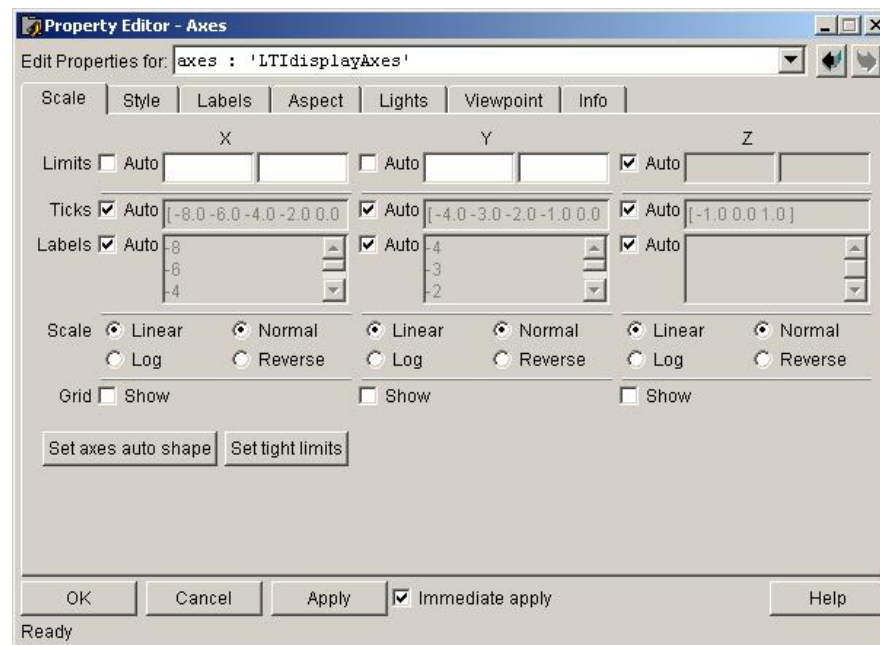


# Elaborare i grafici.

- `axis ( [XMIN XMAX YMIN YMAX] )` imposta la scala degli assi;
- `grid on / grid off` abilita e disabilita la griglia
- `title` inserisce il titolo;
- `xlabel, ylabel` inserisce le etichette negli assi, es. per specificare le unità di misura;
- `legend` inserisce la legenda del grafico.

# Plot editing mode.

- Interfaccia user-friendly per modificare le proprietà delle figure, delle linee, ecc.



# L'ambiente Simulink®.

Simulink: un ambiente grafico per la simulazione di sistemi complessi.

Perche' non basta Matlab?

- E' spesso necessario simulare sistemi complessi, composti da numerosi blocchi interconnessi tra loro;
- Spesso i singoli blocchi sono nonlineari o tempo-varianti;
- Può essere necessario integrare blocchi continui e discreti.

# Come funziona?

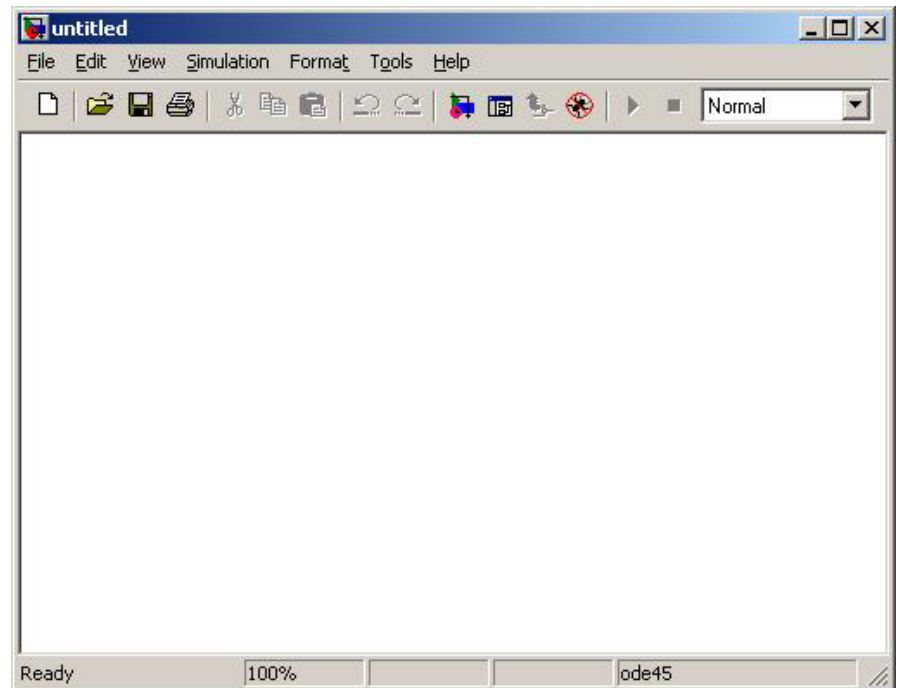
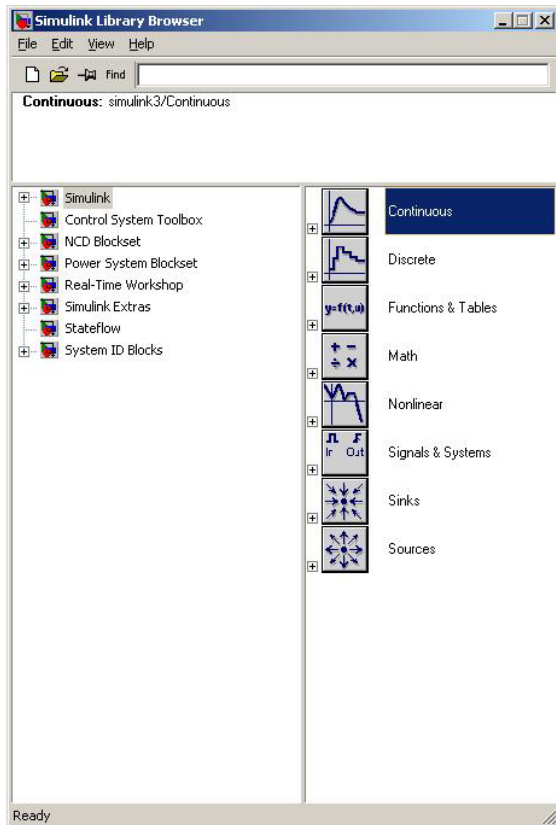
- Simulink contiene una libreria di blocchi che descrivono elementi statici e dinamici elementari;
- L'utente compone lo schema a blocchi del sistema da simulare mediante l'interconnessione dei blocchetti elementari;
- Simulink genera automaticamente le equazioni e risolve il problema numerico di simulazione desiderato.



# Interazione con Matlab®.

- 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.
- Digitando 'simulink' al Matlab prompt si apre la libreria dei modelli.
- Da qui è possibile creare un nuovo modello (foglio bianco) e comporre il sistema da simulare mediante i diversi blocchi.

# Un nuovo modello.



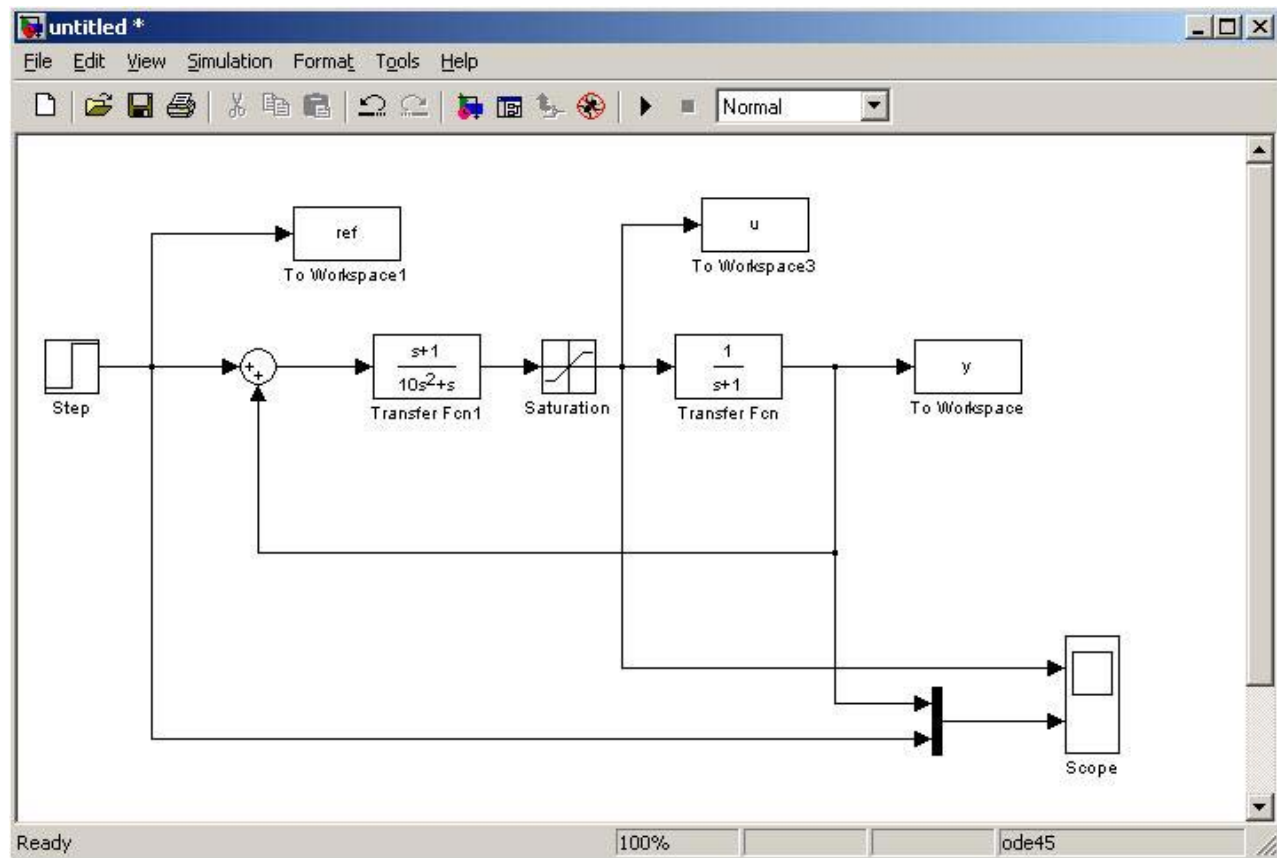
# Librerie Simulink®.

- Continuous (integrator, transfer function, transport delay, zero-pole);
- Discrete (Discrete transfer functions, discrete integrator, zero order hold..);
- Functions & tables (fcn, look-up table, Matlab Fcn);
- Math (gain, sum, product);
- Nonlinear (saturation, switch, dead-zone);

# Librerie Simulink®.

- Signals & Systems (mux, demux);
- Sinks (scope, to workspace);
- Sources (white-noise, clock, constant, pulse generator, repeating sequence, sine, step, from workspace);
- Control System Toolbox (LTI System).

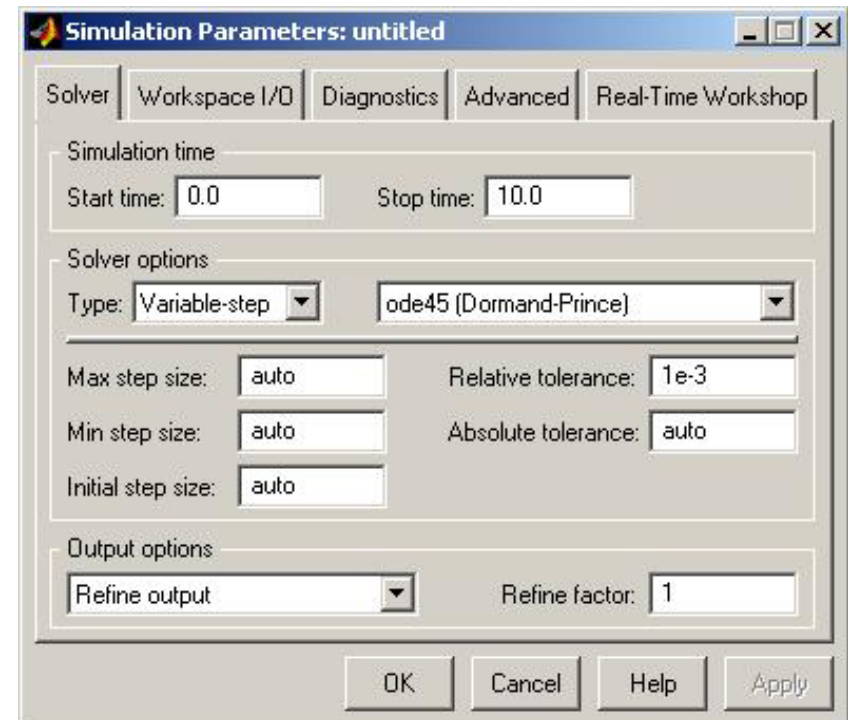
# Esempio di simulazione.



# 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.



## Esempio 2.

---

# Il Control System Toolbox.

- Il Control System Toolbox mette a disposizione del Control Engineer una serie di strumenti classici per la modellazione, l'analisi e il controllo di sistemi dinamici
- Permette di:
  - inserire una f.d.t. in vari modi
  - manipolare sistemi dinamici
  - analizzare risposte temporali e frequenziali
  - progettare un controllore con varie tecniche (es. rlocus)



# Inserire una funzione di trasferimento.

- Inserire 
$$G(s) = \frac{b_m s^m + K + b_1 s + b_0}{a_n s^n + K + a_1 s + a_0} = \frac{s + 1}{s^2 + 2s + 3}$$

```
>> num=[1 1]; den=[1 2 3];
```

```
>> G=tf(num,den)
```

Transfer function:

$s + 1$

-----

$s^2 + 2s + 3$

# Inserire una funzione di trasferimento.

- Inserire

$$G(s) = k \frac{\prod_i (s - z_i)}{\prod_j (s - p_j)}, (z_i, p_j) \in C = 20 \frac{(s + 2)(s - 4)}{(s + 3 + j)(s + 3 - j)}$$

```
>> k=20; Z=[-2 4]; P=[3+i 3-i];
```

```
>> G=zpk(Z,P,k)
```

```
20 (s+2) (s-4)
```

```

```

```
(s^2 - 6s + 10)
```

# Inserire una funzione di trasferimento.

- Più intuitivamente:

$$G(s) = \frac{s + 160}{s^3 + 12s^2 + 30s + 100}$$

```
>> s=tf('s');
```

```
>> G=(s+160)/(s^3+12*s^2+30*s+100)
```

Transfer function:

$s + 160$

-----

$s^3 + 12 s^2 + 30 s + 100$

# Estrarre dati da una f.d.t.

- Estrarre il numeratore e il denominatore (tfdata):

```
>> [num,den]=tfdata(G,'v')
```

```
num =
```

```
 0 0 1 160
```

```
den =
```

```
 1 12 30 100
```

# Estrarre dati da una f.d.t.

- Estrarre zeri poli e guadagno (zpkdata):

```
>> [z,p,k]=zpkdata(G,'v')
```

```
z =
```

```
-160
```

```
p =
```

```
-10.0000
```

```
-1.0000 + 3.0000i
```

```
-1.0000 - 3.0000i
```

```
k =
```

```
1
```

# Proprietà delle f.d.t.

Altre caratteristiche delle f.d.t.

- `damp` pulsazione naturale e coefficiente di smorzamento di poli e zeri;
- `dcgain` guadagno statico;
- `pole` poli della f.d.t. ;
- `zero` zeri della f.d.t. ;
- `pzmap` grafico di poli e zeri nel piano complesso (`sgrid` permette di tracciare i luoghi caratteristici);

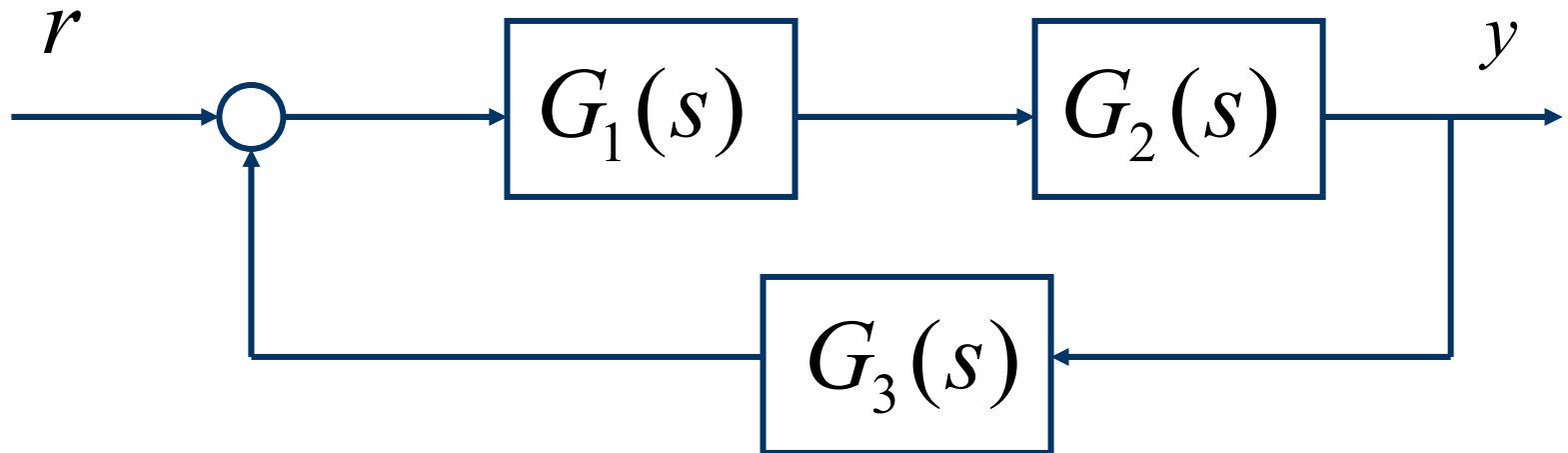
# Interconnessione di sistemi.

- Connessione in serie:  
>>> G=series(G1,G2);
- Connessione in parallelo:  
>>> G=parallel(G1,G2);
- Connessione in feedback:  
>>> Gtot=feedback(G,G3);

# Esempio di interconnessione.

- Calcolare la f.d.t. totale

$$G_1(s) = \frac{1}{s+1}; \quad G_2(s) = \frac{4}{s^2 + 0.8s + 4}; \quad G_3(s) = \frac{2}{s+2};$$





# Esempio di interconnessione.

- ```
>> s=tf('s');  
>> G1=1/(s+1); G2=4/(s^2+0.8*s+4); G3=2/(s+2);  
>> andata=series(G1,G2);  
>> Gtot=feedback(andata,G3)
```

Transfer function:

$$4 s + 8$$

$$s^4 + 3.8 s^3 + 8.4 s^2 + 13.6 s + 16$$

Simulazione di sistemi lineari.

Funzioni disponibili per la simulazione:

- **impulse**: simulazione risposta all'impulso;
- **step**: simulazione risposta a scalino;
- **initial**: simulazione movimento libero;
- **lsim**: simulazione con ingresso qualsiasi e stato iniziale qualsiasi.

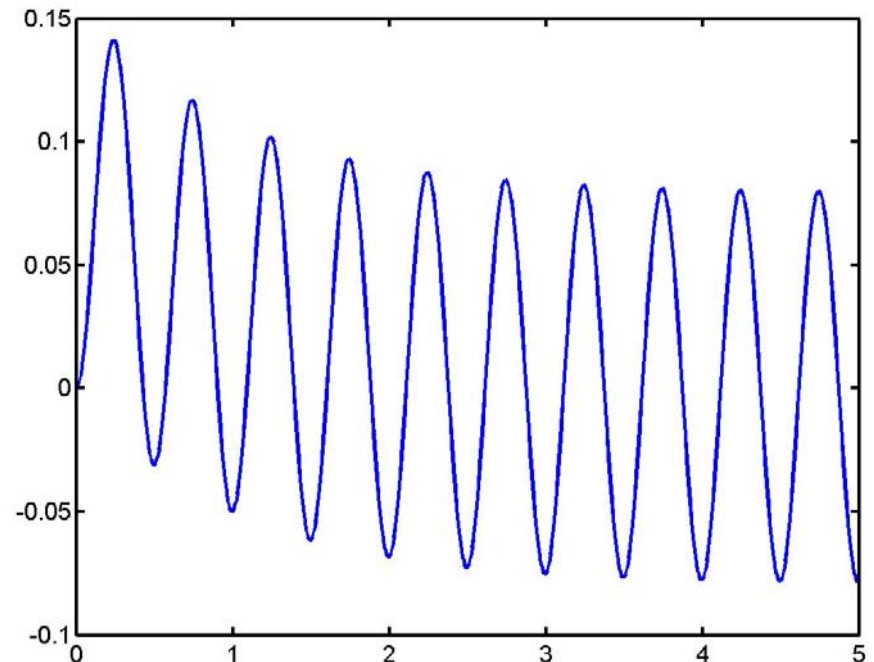
Sintassi :

```
>> [y,t]=step(G);
```

```
>> [y,t]=lsim(G,u,t);
```

Esempio di simulazione.

```
>> s=tf('s'); G=1/(s+1);  
>> t=(0:0.01:5); u=sin(2*pi*2*t);  
>> y=lsim(G,u,t);  
>> plot(t,y)
```



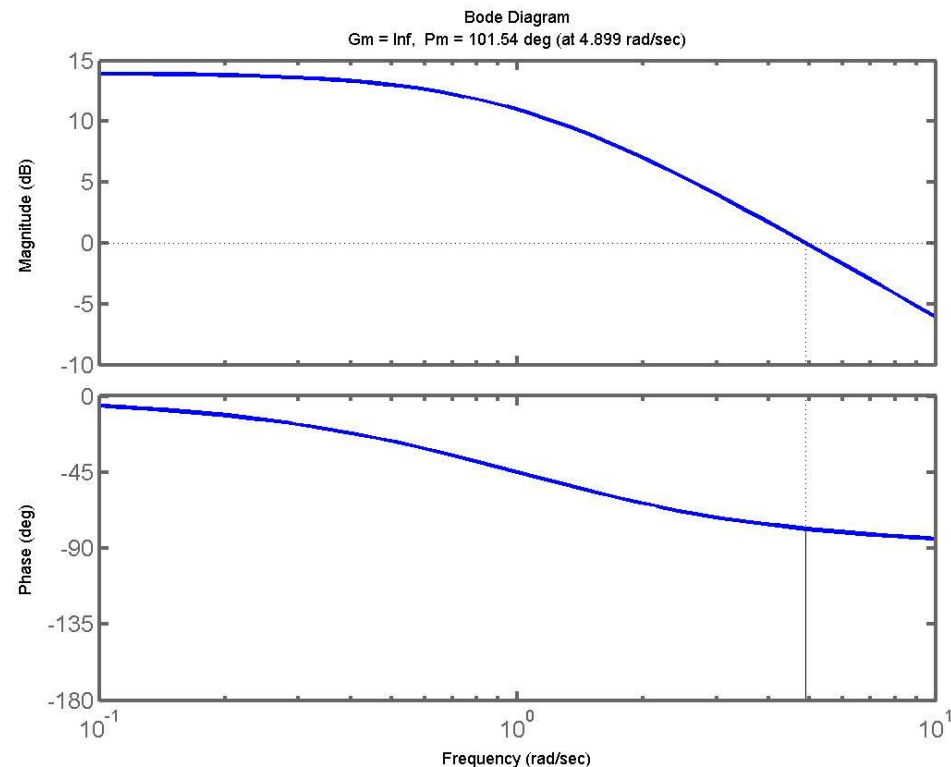
Analisi di sistemi di controllo.

Per i problemi di controllo lineari invarianti SISO esistono le seguenti funzioni:

- `bode(sistema)`: tracciamento diagrammi di Bode della risposta in frequenza;
- `margin(sistema)`: come `bode` ma in più calcola pulsazione critica, margine di fase e margine di guadagno;
- `nyquist(sistema)`: tracciamento diagramma di Nyquist della risposta in frequenza;
- `rlocus(sistema)`: tracciamento luogo delle radici (`rlocfind`);

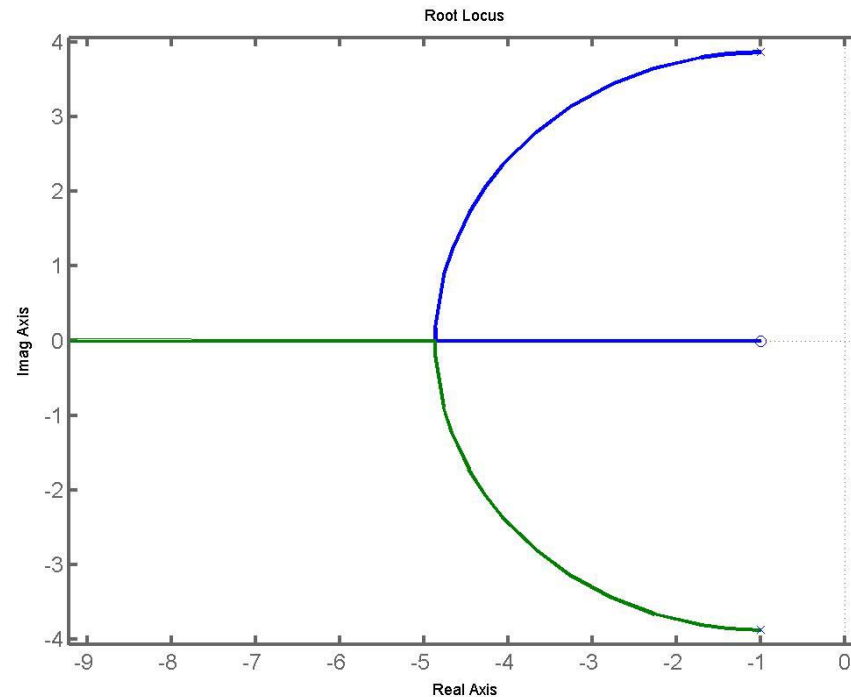
Esempi di analisi.

- `>> s=tf('s'); L=5/(s+1); margin(L)`



Esempi di analisi.

- `>> s=tf('s'); L=(s+1)/(s^2+2*s+16); rlocus(L)`



Esempio 3.
