Introduzione a Matlab

Descrizione delle funzionalità di base

Simulazione di sistemi dinamici a tempo continuo

Indice del materiale

- Breve **introduzione** a Matlab
  - Descrizione generale di Matlab (v. 7.x)
  - Quadro delle funzioni predefinite
  - Definizione di matrici, vettori e polinomi
  - Rappresentazione grafica dei dati
  - Cenni alla programmazione in ambiente Matlab

- **Analisi** e **simulazione** di **sistemi dinamici** in ambiente Matlab
  - Rappresentazione di sistemi dinamici tempo-invarianti nell'ambiente Matlab
  - Analisi e simulazione di sistemi dinamici tempo-invarianti: utilizzo della ODE suite

Breve introduzione a Matlab

L'ambiente di lavoro, le istruzioni fondamentali

A cosa serve questa presentazione

- Scopi di questo materiale:
  - fornire le informazioni necessarie per l'uso di **Matlab** in relazione alle esercitazioni del corso;
  - dare una panoramica generale (**tutt'altro che esauriente**) delle potenzialità di **Matlab** per la formulazione e la soluzione di problemi numerici nell'ambito del corso di "Teoria dei Sistemi e del Controllo"

- Questo materiale **NON È un** manuale d'uso di Matlab e non è una guida dettagliata ai metodi numerici di risoluzione di equazioni differenziali oppure alle differenze finite.

Dove trovare altre informazioni?

- Sito web di Mathworks:

  www.mathworks.com

  seguendo i link alla voce "*support*" è possibile trovare i manuali di Matlab in formato Adobe PDF.

  (http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml)

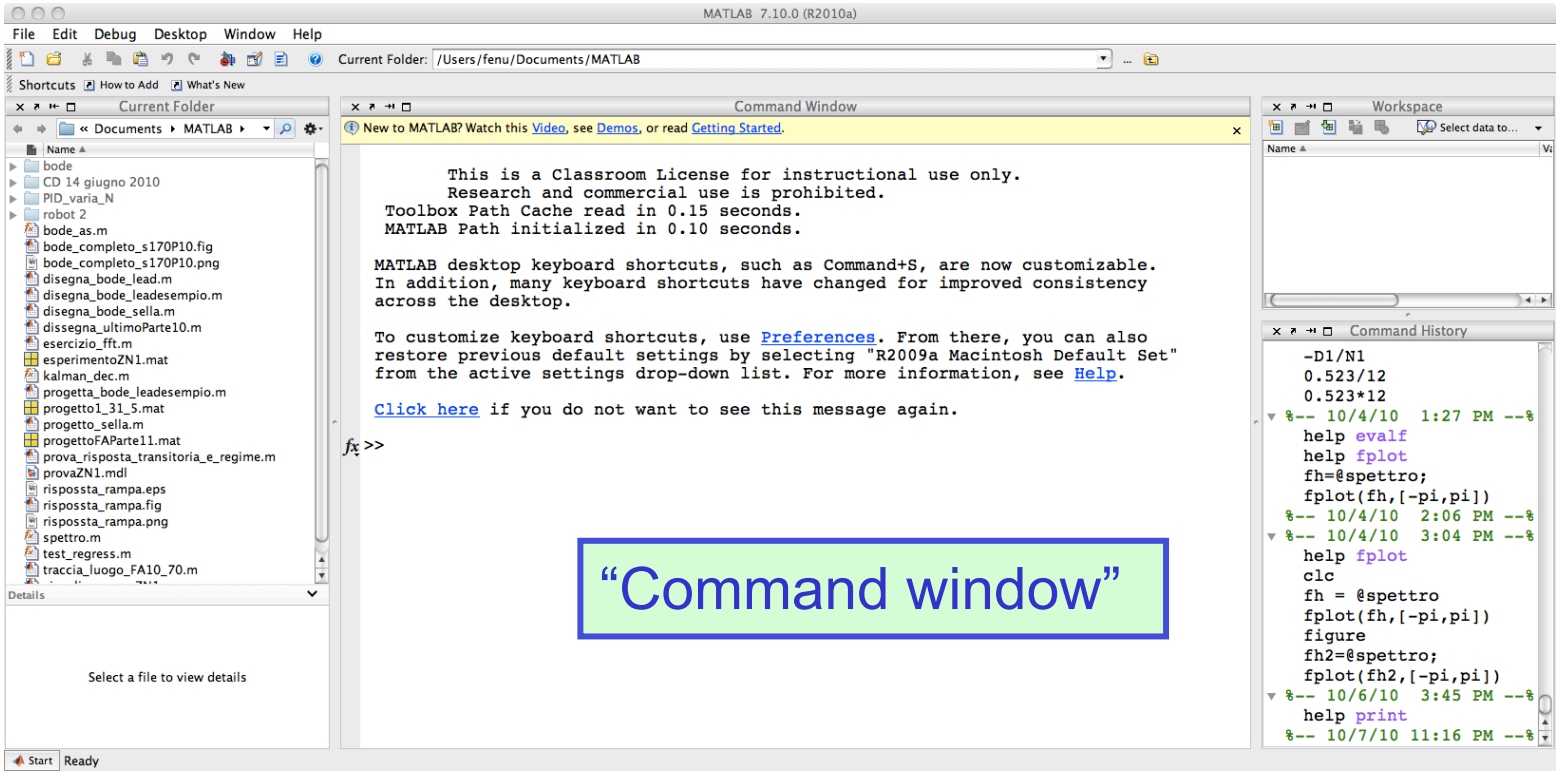
- Un testo in italiano di introduzione a Matlab:

  "***Guida Operativa a MATLAB, SIMULINK e Control Toolbox***" , Alberto Cavallo, Roberto Setola, & Francesco Vasca, Liguori Editore, 1994 (2^a ed. 2002) → in biblioteca

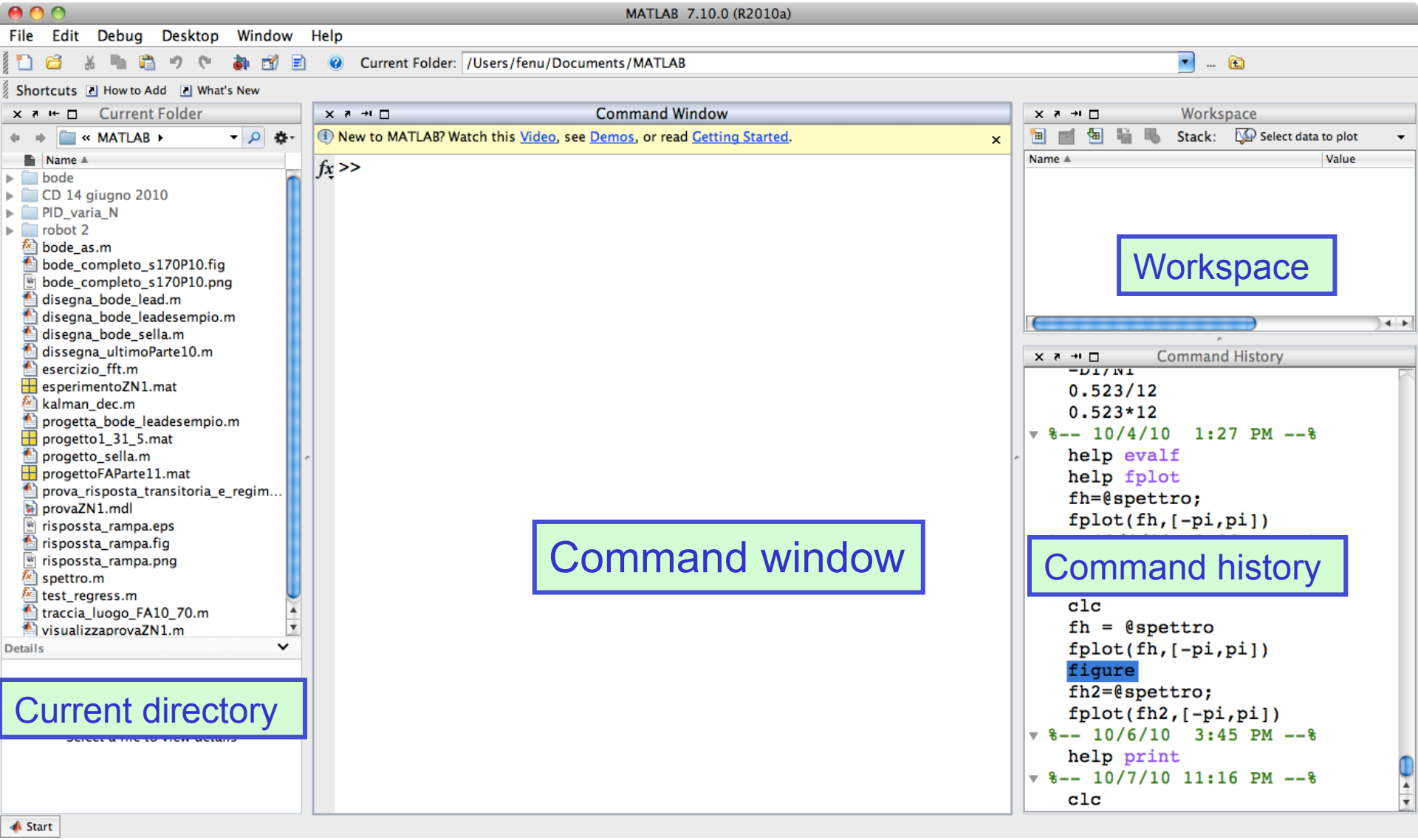
Descrizione generale di Matlab

- MATLAB (= MATrix LABoratory):
  - un **linguaggio di programmazione** per applicazioni scientifiche e numeriche
  - vasto set di **funzioni predefinite**
  - **interprete** di comandi
  - possibilità di scrivere nuove funzioni
  - libreria di **TOOLBOX** per svariate applicazioni; ad es. Signal Processing, Identificazione di modelli, …

L'interfaccia di Matlab



- Interfaccia utente:
  - la ***Command Window*** dà accesso diretto all'interprete
  - scrittura diretta di comandi.



Current directory

Command window

Workspace

Command history

- **Command Window**: è la finestra dell'interprete di comandi di Matlab. Vi si possono scrivere direttamente comandi o lanciare programmi scritti nel linguaggio di programmazione del Matlab.

- **Command History**: è la finestra che contiene l'elenco dei comandi inseriti nella Command Window.

- **Current Directory**: visualizza in forma grafica il contenuto della cartella di lavoro.

- **Workspace**: è la finestra che visualizza il contenuto dell'area di memoria utilizzata dal Matlab (nomi, tipi e dimensioni delle variabili).

Matlab come calcolatrice...

- La modalità di impiego più "semplice": per valutare espressioni numeriche.
- Esempio: per calcolare $4+\sqrt{2}-\sin(0.2\pi)^2+e^2$
  è sufficiente digitare al prompt **»**

```
»4 + sqrt(2) - sin(0.2*pi)^2 + exp(2)
    ans =
       12.4578
```

- Il risultato viene salvato nella variabile *ans*.

Definizione di variabili

- È possibile definire variabili e espressioni non numeriche più complesse.

- Esempio:

```
» 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***.
- Il comando ***whos*** mostra una lista delle variabili definite:

```
» whos
  Name      Size            Bytes  Class
  a         1x1                 8  double array
  ans       1x1                 8  double array
  b         1x1                 8  double array
Grand total is 3 elements using 24 bytes
```

Lettura e scrittura su file

Mediante i comandi ***load*** e ***save*** è possibile salvare su file le variabili del ***workspace***.

- ***load*** *nomefile variabile1 variabile2* **...** carica dal file *nomefile.mat* le variabili elencate
- ***save*** *nomefile variabile1 variabile2* **...** scrive nel file *nomefile.mat* le variabili elencate.
- ***load*** *nomefile* carica tutte le variabili in nomefile.
- ***save*** *nomefile* salva tutto il **workspace** in *nomefile.mat*

Quindi...

- Esiste un insieme (molto vasto) di funzioni predefinite (come ***sin*** e ***sqrt*** nell'esempio precedente).

- A differenza dei normali linguaggi di programmazione (C, Java…) non occorre *dichiarare* le variabili. L'assegnazione coincide con la dichiarazione.

Operazioni matematiche elementari

- Moltiplicazione $a\,b$ ⟷ **a * b**
- Divisione (divisione "a destra") $\frac{a}{b}$ ⟷ **a / b**
- Divisione "a sinistra" $\frac{b}{a}$ ⟷ **a \ b**
- Elevamento a potenza a^b ⟷ **a ^ b**
- Addizione e sottrazione $a + b$, $a - b$ ⟷ **a + b**, **a - b**

Operazioni elementari: precedenza

- La precedenza nello svolgere le operazioni matematiche elementari in Matlab è quella standard, facendo però attenzione alle operazioni di "divisione a destra" e "divisione a sinistra".
- Infatti valgono le regole:
  - L'**elevamento a potenza** viene valutato prima delle operazioni di **moltiplicazione** e **divisione**, che hanno la medesima priorità,
  - L'operazione di "**divisione a destra**" ha priorità rispetto alla "**divisione a sinistra**";
  - Le operazioni di **addizione** e **sottrazione** hanno la priorità più bassa.
  - Per assegnare una diversa precedenza nella sequenza dei calcoli è necessario **utilizzare le parentesi**.

L'operatore di assegnazione "="

- L'operatore "**=**" è utilizzato in Matlab per **assegnare** un valore (numerico, come il risultato di un'operazione di calcolo, oppure testuale o simbolico [ne parliamo più avanti]) ad una variabile.

- Quindi è appropriato pensare all'operatore "=" come ad un'istruzione con la quale si assegna un valore ad una variabile, in un qualsiasi linguaggio di programmazione (C, Fortran ecc.).

- Se l'assegnazione termina con un "**;**" allora il risultato dell'operazione di assegnazione non viene visualizzato, altrimenti si.

```
» x = 2; y = 4; z = x*y
z = 8
```

L'operatore di assegnazione "=" (2)

- Nel caso di assegnazioni lunghe si può utilizzare il simbolo "**...**" per poi proseguire l'assegnazione nella riga seguente

```
» FirstClassHolders = 72;
» Coach = 121;
» Crew = 8;
» TotalPeopleOnPlane = FirstClassHolders + Coach...
+ Crew

TotalPeopleOnPlane =
201
```

Il formato di visualizzazione

- Il risultato di un'operazione numerica viene visualizzato di solito tramite un numero reale, con sole 4 cifre decimali.
- Questo è il formato di visualizzazione standard.

- Esistono altre possibilità e si scelgono tramite il comando **format**:
  - **format short** 4 cifre decimali
    - **short e** 4 cifre decimali, notazione esponenziale
  - **format long** 15 cifre decimali
    - **long e** 15 cifre decimali, notazione esponenziale
  - …

Il formato di visualizzazione

- **format rat** risultato approssimato tramite la più vicina frazione
- **format bank** formato finanziario, con sole due cifre decimali
- …

```
»format long
» x = 3 + 11/16 + 2^1.2
   x =  5.98489670999407
» format short
» x = 3 + 11/16 + 2^1.2
   x =  5.9849
```

Esempi di funzioni predefinite
(di uso piu' comune)

- Funzioni trigonometriche (***sin***, ***cos***, ***tan***, ***acos***, ***asin***, ***atan***…);
- Esponenziale e logaritmo (***exp***, ***log***, ***log10***, ***sqrt***…);
- Numeri complessi (***abs*** ⇒ modulo, ***angle*** ⇒ fase, ***real*** ⇒ parte reale, ***imag*** ⇒ parte immaginaria…);

Alcuni esempi semplici

- Calcolare il modulo di (2+3i):

```
» abs( 2+3*i )
   ans =
      3.6056
```

- Calcolare $20\log_{10}\left(\left|\frac{2+3i}{4+6i}\right|\right)$

```
» 20*log10( abs( (2+3*i) / (4+6*i) ) )
   ans =
      -6.0206
```

Inf e NaN

- Alcune operazioni numeriche possono dare luogo a risultati non corretti, esprimibili soltanto facendo uso di "forme indeterminate", che vengono definite da Matlab tramite le grandezze **Inf** e **NaN**.

- Esempi:

```
» 5/0
Warning: Divide by zero.
ans =
   Inf
```

```
» 0/0
Warning: Divide by zero.
ans =
   NaN
```

Help! Una funzione fondamentale!

- **help** per ottenere la lista dei toolbox installati
- **help** ***nome_toolbox*** per ottenere la lista dei comandi installati in un *toolbox*
- **help** ***nome_comando*** guida 'on-line' di MATLAB sullo specifico comando
- **ver** info sulla versione di MATLAB

Definizione di matrici

- Come si definisce una matrice in Matlab?

  Esempio: definire la matrice 2x2

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

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

- Come si accede agli elementi di una matrice:

```
» A( 1, 2 )
ans =
    2
```

Indici (riga e colonna) dell'elemento di interesse

L'operatore :

- Per accedere a intere righe o colonne di una matrice, si usa il simbolo **:**

- 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 definiamo

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

Sottomatrice di interesse

- scrivendo

```
» B( 1:2 , 2:3 )
ans =
     2     3
     5     6
```

Indici della sottomatrice di interesse

Operazioni (elementari) sulle matrici

- Sono definiti gli operatori **+**, **-**, * e **^**

```
» A = [ 1 2; 3 4 ]; B = [ 5 6; 7 8 ]
» A+B
ans =
    6    8
   10   12

» B-A
ans =
    4    4
    4    4
```

```
» A*B
ans =
   19   22
   43   50

» A^2
ans =
    7   10
   15   22
```

Operazioni (elementari) sulle matrici (2)

- Sono definiti gli operatori **.*, ./** e **.^**, che si applicano elemento per elemento:

```
» A = [ 1 2; 3 4 ]; B = [ 5 6; 7 8 ];
» A.*B
ans =
     5    12
    21    32
» A.^B
ans =
         1        64
      2187     65536
```

Operazioni (elementari) sulle matrici (3)

- Determinante:

```
» det(A)
ans =
    -2
```

- Autovalori:

```
» eig(A)
ans =
   -0.3723
    5.3723
```

- Matrice inversa:

```
» inv(A)
ans =
   -2.0000    1.0000
    1.5000   -0.5000
```

- Matrice trasposta:

```
» A'
ans =
     1     3
     2     4
```

Operazioni (elementari)

- Funzioni matematiche di base **help elfun**
- Funzioni matematiche specifiche **help specfun**
- Funzioni di manipolazione delle matrici **help elmat**

Altre operazioni

- Osservazione importante: NON occorre definire le dimensioni in modo esplicito!

  Per conoscere le dimensioni di una matrice: ***size***

- Altre operazioni:
  - ***rank*** -> calcolo del rango di una matrice
  - ***trace*** -> calcolo della traccia di una matrice
  - ***norm*** -> calcolo della norma di una matrice

Alcune matrici "speciali"

- ***eye*(n,n)** ⇒ matrice identità n x n;
- ***zeros*(n,m)** ⇒ matrice di "0" n x m;
- ***ones*(n,m)** ⇒ matrice di "1" n x m;
- ***rand*(n,m)** ⇒ matrice n x m con elementi distribuiti uniformemente tra 0 e 1.

Vettori

- I vettori hanno due funzioni fondamentali in Matlab:
  - rappresentazione di **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 (1)

- » v=(0:10)

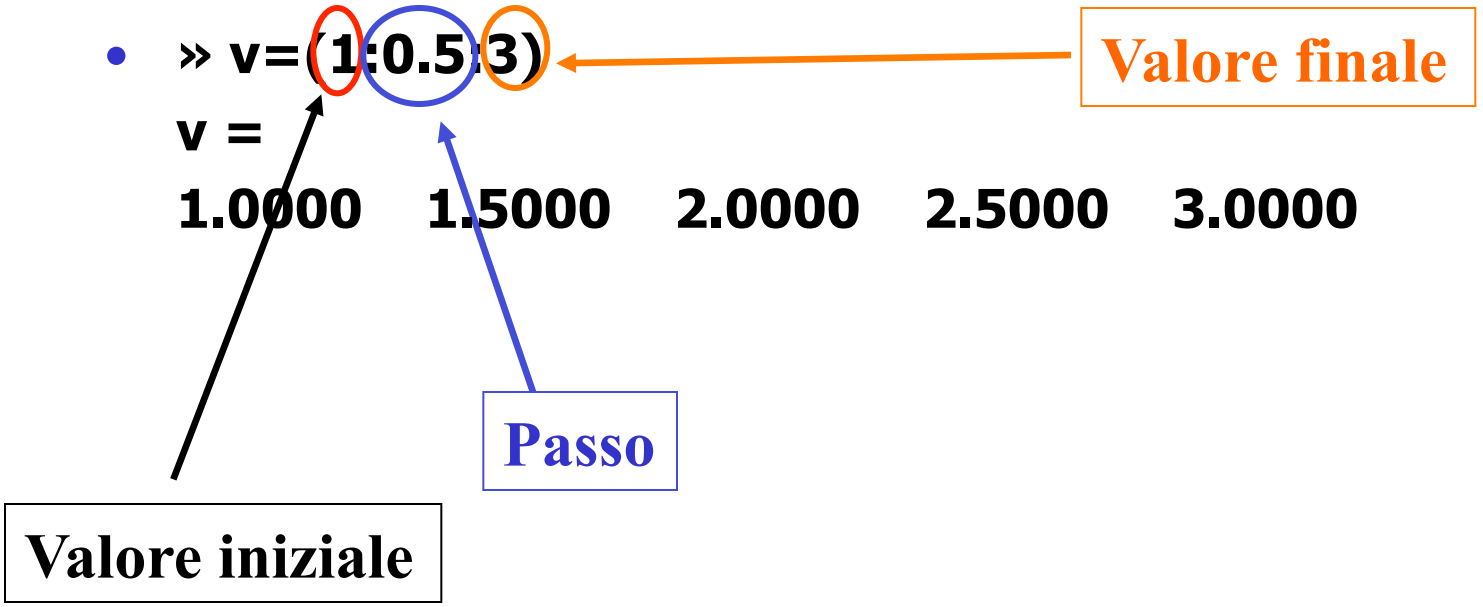
  v =

  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



Definizione di vettori (2)

- Come matrici riga o colonna:

```
» v = [ 3 6 1 7 ]
v =
   3   6   1   7
```

- Polinomi: sono rappresentati come vettori
  Es.:

```
» pol = [ 3 2 1 ]
pol =
   3   2   1
```

$$p(z) = 3z^2 + 2z + 1$$

Operazioni sui polinomi

- Calcolo delle radici ⇒ ***roots***

```
» roots( pol )
ans =
  -0.3333 + 0.4714i
  -0.3333 - 0.4714i
```

- Valutazione in un punto ⇒ ***polyval***

```
» polyval( pol, 0 )
ans =
     1
```

Operazioni sui polinomi (2)

- Prodotto di polinomi ⇒ ***conv***

  Esempio:

$$(z+1)(z+1) = z^2 + 2z + 1$$

```
» pol1 = [ 1 1 ]; pol2 = [ 1 1 ];
» polprod = conv( pol1, pol2 )
polprod =
     1     2     1
```

Rappresentazione grafica

- Grafici 2D:
  - In **scala lineare** ⇒ ***plot***

    plot(x,y) traccia il grafico dei punti che hanno come ascisse (ordinate) gli elementi del vettore x (y).
  - In **scala semilogaritmica o logaritmica** ⇒ ***semilogx***, ***semilogy***, ***loglog***

    stessa sintassi di *plot*
  - Diagrammi **polari** ⇒ ***polar***

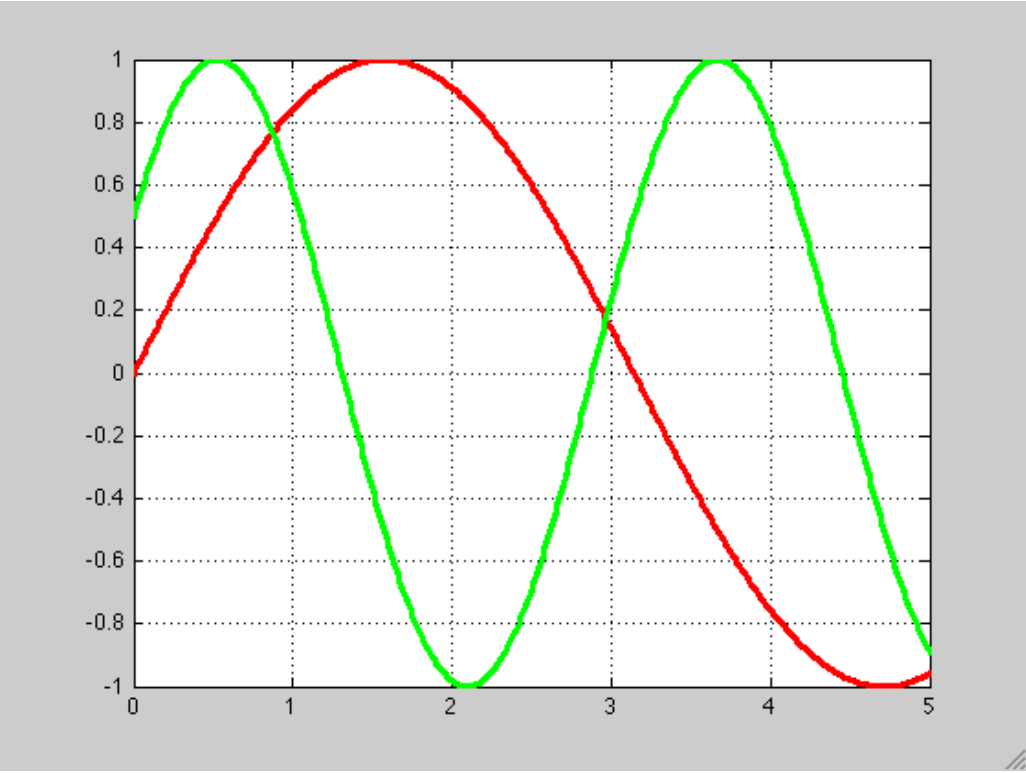
Rappresentazione grafica (2)

- Altre funzioni utili:
  - cambiamenti di scala ⇒ ***axis*([xmin,xmax,ymin,ymax])**
  - sovrapposizione di più grafici ⇒ ***hold***
  - aggiunta di griglia al grafico ⇒ ***grid***
  - titolo al grafico ed etichette agli assi ⇒ ***title('..'), xlabel ('..'), ylabel('..')***
  - più grafici in una finestra ⇒ ***subplot***
  - inserimento di testo in una figura ⇒ ***gtext***

Esempio

- Grafici sovrapposti

```
» t=(0:0.01:5);
» y1=sin(t);
» y2=cos(2*t-pi/3);
» figure;
» plot(t,y1,'r',t,y2,'g');
» grid on;
```



Un tipo di dato "speciale": *cell array*

- Il tipo di dato "cell array" sta ad indicare che si vuole creare un oggetto di tipo **array** i cui **elementi** possono essere **di tipo diverso** (valori numerici, testo, altri tipi di dati).

- Operativamente, un dato di tipo "cell array" si crea in modo simile agli array "classici" in Matlab, semplicemente sostituiendo ai delimitatori classici [ ] i delimitatori { }.

- Esempio: per creare un cell array 2-x-2

```
A = {[1 4 3; 0 5 8; 7 2 9], 'Paolino Paperino';3+7i, -pi:pi/4:pi};
```

Cenni alla programmazione in ambiente Matlab

Operazioni con vettori, caratteri e testo

Cicli, operazioni e condizioni logiche

Funzioni e struttura di un programma in Matlab

Operazioni sui vettori e matrici

- I vettori (e le matrici) possono essere manipolati in Matlab non solo per estrarre parti di essi, ma anche per
  - Concatenare più vettori (matrici) assieme

```
» t1 = [1 :10];
» t2 = [ 20:-1:11];
» t3=[t1, t2];
```

  - Modificare le dimensioni di un vettore (matrice)

```
» help reshape
» t3bis = reshape(t3,2,10);
```

– Cancellare/sostituire elementi in un vettore (matrice)

```
» disp(t3bis);
» t3bis(:,4:6)=[];
» disp(t3bis);
» whos
```

```
» disp(t3bis)
» t3bis(1,4:6)=-3;
» whos
```

Operazioni su caratteri e testo

- Le stringhe di caratteri sono viste come particolari array in Matlab, quindi su di esse possono essere applicate le operazioni di manipolazione sui vettori già descritte:

```
» t1 = 'A'
» t2 = 'BCDE'
» t3 = [t1,t2]
» t4 = [t3,' are the first 5 ';...
'characters in the alphabet.']
```

- A volte è necessario convertire un numero in testo e viceversa: vedere le istruzioni **str2num**, **num2str**, **int2str** e simili.

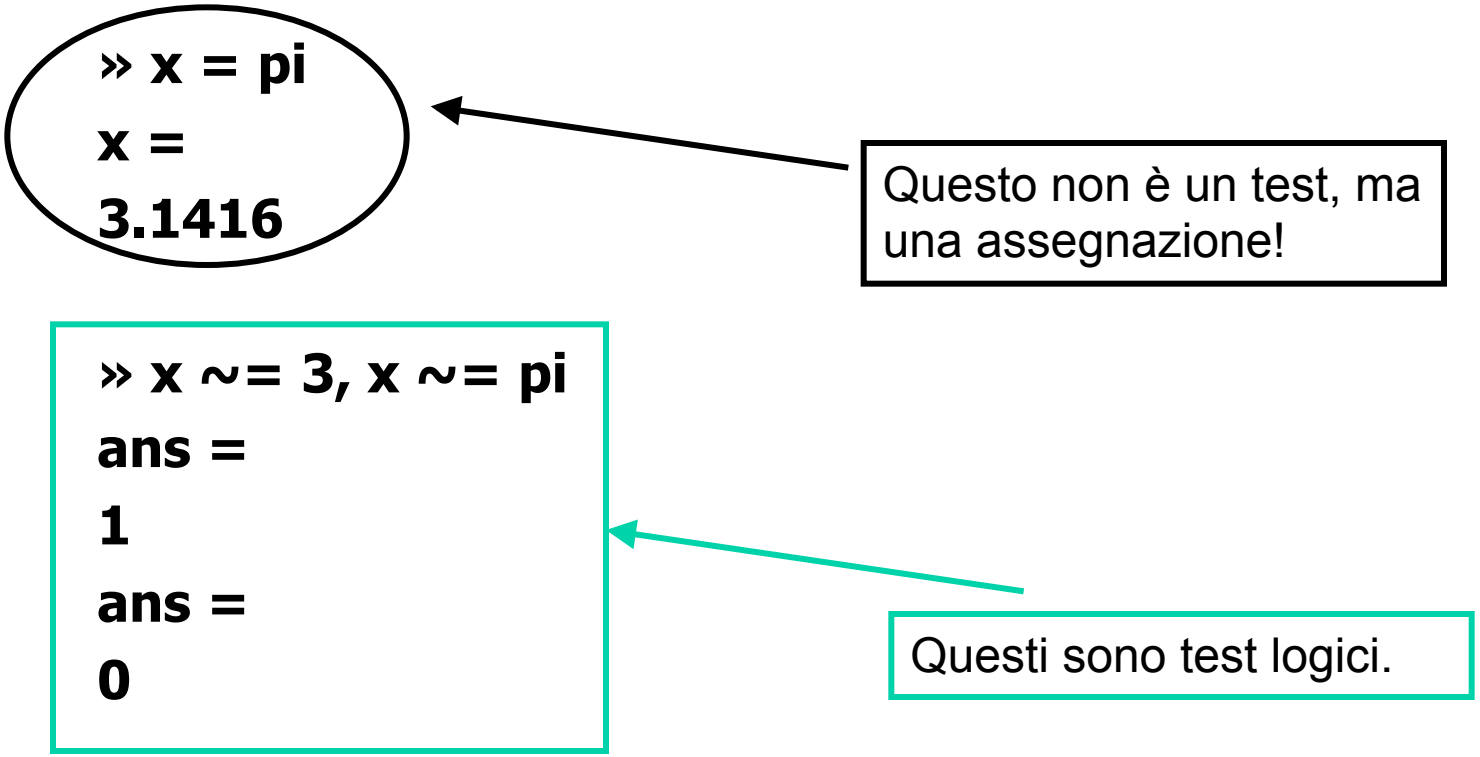
Cicli, operazioni e condizioni logiche

- Operazioni cicliche:

```
for indice = inizio:passo:fine
    fai_qualcosa;
end
```

- Operazioni logiche, condizioni logiche elementari:
  - "vero" e "falso": **true = 1 false = 0**
  - Confronti logici:
    - **x == 2** x è uguale a 2?
    - **x ~= 2** x NON è uguale a 2?
    - **x > 2** x è maggiore di 2?
    - **x < 2** x è minore di 2?
    - **x >= 2** x è maggiore oppure uguale a 2?
    - **x <= 2** x è minore oppure uguale a 2?

- Attenzione a questo esempio:



Composizione di operazioni logiche

- Le operazioni logiche elementari si possono comporre con gli operatori logici “and”, “or”, “xor”, “not”:
  - And **&**
  - Or **|**
  - Xor **xor**
  - Not **~**

- Per una descrizione completa: **help relopt**

Cicli "while"

- La struttura del ciclo è quella classica

while test logico

Comandi_da_eseguire mentre la condizione logica ha valore "true"

end

```
» S = 1; n = 1;
» while S+ (n+1)^2 < 100
    n = n+1; S = S + n^2;
  end
» [n, S]
```

"if" … "then" … "else" …

- La struttura è quella classica

```
if test_logico_1
   Comandi da eseguire se test1 è "true"
elseif test_logico_2
   Comandi da eseguire quando test2 è "true" ma
   test1 è "false"
...
end
```

Un esempio

- Si vuole risolvere l'equazione $x = \cos(x)$
- La trasformiamo in una successione e cerchiamo di trovar a quale valore converge la successione

$$x_n = \cos(x_{n-1}) \quad n \in \mathbf{N}$$

- Il tutto deve essere uno script Matlab.

```
% risolvi x= cos(x)

% metodo 1
% spreca memoria, memorizza anche passi intermedi
x = zeros(1,20); x(1) = pi/4;
n = 1; d = 1;
 while d > 0.001
    n = n+1; x(n) = cos(x(n-1));
    d = abs( x(n) - x(n-1) );
end
n,x


% metodo 2
% migliore memorizza solo l'ultimo dato
 xold = pi/4; n = 1; d = 1;
 while d > 0.001 & n < 20
    n = n+1; xnew = cos(xold);
    d = abs( xnew - xold );
    xold = xnew;
end
[n, xnew, d]
```

Matlab function

- Un m-file è una Matlab function **se e solo se** la prima riga del file contiene il seguente testo

  - **function** [elenco variabili in uscita] =
    nome_della_funzione (elenco variabili in ingresso)

- Non è necessario terminare il file con una parola chiave, anche se esiste l'istruzione "**return**" (vedere **help return**)

Alcuni esempi

- Calcolo dell'area di un triangolo qualsiasi:
  - Vale la formula $$A = \sqrt{s(s-a)(s-b)(s-c)}$$

$$s = \frac{a+b+c}{2}$$

  - con a,b,c, lunghezze dei lati del triangolo (cfr. formula di Erone).

- La funzione allora (nella sua versione più semplice) avrà 3 parametri in ingresso ed 1 solo parametro in uscita

```
function [A] = area_erone(a,b,c)
% area di un triangolo
% lati di lunghezza a, b, c.
% Ingressi:
% a,b,c: lunghezze dei lati
% Uscita:
% A: area del triangolo
% Utilizzo:
% Area = area_erone(2,3,4);

s = (a+b+c)/2;
A = sqrt(s*(s-a)*(s-b)*(s-c));
%%%%%%%%% end %%%%%%%%%%%%
```

- Provare a creare il file "area_erone.m" e poi a scrivere il comando "help area_erone" nella Command Window di Matlab.

- Utilizzo della funzione
  - Assegnando una variabile in uscita

```
» Area = area_erone(10,15,20)
Area =
72.6184
```

  - non assegnando la variabile d'uscita

```
» area_erone(10,15,20)
ans =
72.6184
```

Un esempio: i numeri di Fibonacci

- Calcolare la sequenza:

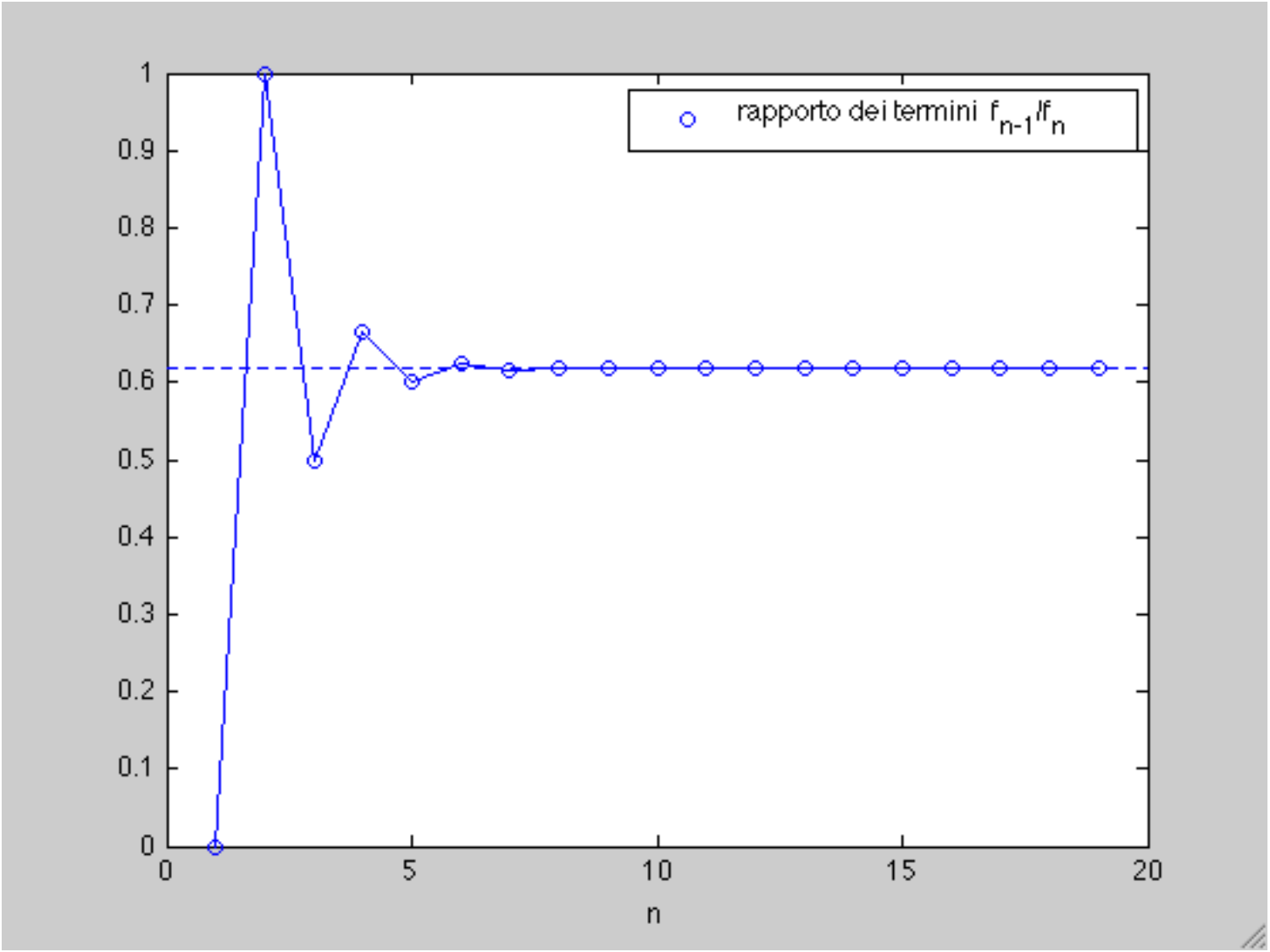
$$f_n = f_{n-1} + f_{n-2} \quad n = 3, 4, 5 \ldots$$

- Calcolare il valore a cui tende il rapporto tra due termini successivi al crescere dell'indice

$$\frac{f_{n-1}}{f_n} = ?$$

$$f(1) = 0 \qquad f(2) = 1$$

```
F(1) = 0; F(2) = 1;
% inizializzazione
for i = 3:20
    F(i) = F(i-1) + F(i-2);
end
% calcolo della sequenza

% grafici
plot(1:19, F(1:19)./F(2:20),'o' )
hold on, xlabel('n')
plot(1:19, F(1:19)./F(2:20),'-' )
legend('rapporto dei termini  f_{n-1}/f_n')
plot([0 20], (sqrt(5)-1)/2*[1,1],'--')
```


rapporto dei termini f_{n-1}/f_n
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
n

Cenni alla "vettorializzazione del codice"

- Per "**calcolo vettorializzato**" si intende un **algoritmo che trae vantaggio dallo sfruttare/eseguire operazioni su interi vettori/matrici** piuttosto che su ciascun singolo componente di essi.

- Si può "vettorializzare" una discreta varietà di strutture di programmazione e tale operazione può portare ad incrementare la velocità d'esecuzione anche di un fattore 10.
- La "vettorializzazione" del codice è **una** delle **pratiche più efficienti** per ottenere **codice** Matlab molto **performante.**

- NB "vettorializzare" NON è sinonimo di "parallelizzare" il codice!

Vettorializzare i cicli

- Vettorializzare in questo caso significa trasformare in operazioni su vettori/matrici le operazioni contenute in cicli for, oppure while.
- Esempio

```
i = 0;
for t = 0:.01:10
  i = i + 1;
  y(i) = sin(t);
end
```

- La versione vettoriale del medesimo codice può essere

```
t = 0:.01:10; y = sin(t);
```

- Quanto si guadagna?

- Lo script "test_ciclo.m" fornisce il seguente risultato

risultati
primo ciclo
  7.6309e-005
secondo ciclo
  5.6040e-005

Incremento di velocità del 25% circa

- Appare evidente che si riesce ad incrementare la velocità di esecuzione.

- **ATTENZIONE**: non sempre è possibile vettorializzare i cicli. In tal caso si può pensare di precompilare il codice "scalare" per ottenere comunque un certo incremento di prestazioni. **NON** affrontiamo l'argomento.

```
% test_ciclo
% script che serve a eseguire il timing su due cicli,
% uno vettorializzato e l'altro no.
%
tic;
 i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
tempo1 = toc;

% secondo ciclo
clear t y
tic;
t = 0:.01:10;
y = sin(t);
tempo2 = toc;

% risultato
messaggio = sprintf('ecco il risultato: primo ciclo %f secondo ciclo
%f',tempo1, tempo2);
disp(messaggio);
```

Vettorializzazione di algoritmi

- Si può incrementare la velocità di esecuzione vettorializzando anche altre strutture di codice.

- Esempio: versione "scalare"

```
function d = minDistance(x,y,z)
% trova in un insieme di punti di R^3
% quello a distanza minima dall'origine e restituisce la distanza min
nPoints = length(x);
d = zeros(nPoints,1); % prealloca (ottimizzazione tempo di calcolo)
for k = 1:nPoints % calcola la distanza per ogni punto
  d(k) = sqrt(x(k)^2 + y(k)^2 + z(k)^2);
end
d = min(d); % trova la distanza minima
```

- versione "vettoriale" del medesimo algoritmo

```
function d = minDistance(x,y,z)
% trova la distanza minima dall'origine in R^3
d = sqrt(x.^2 + y.^2 + z.^2); % calcola la distanza per ogni punto
d = min(d); % trova la distanza minima
```

- Si può migliorare ancora:

```
d = sqrt(min(x.^2 + y.^2 + z.^2)))
```

- Provare le tre versioni su di un insieme di punti (es. 10000 punti a caso), misurando le prestazioni in tutti e tre i casi.

"il mio 1° test" (5-10 minuti)

- Risolvere tramite uno script Matlab e/o una Matlab function il problema di algebra matriciale

$$A\,x \;=\; b$$

- Le matrici del problema vengono assegnate all'inizio dello script.
- È necessario prevedere che la matrice A possa essere non quadrata e che nel caso sia matrice quadrata possa essere non invertibile.

$$A_1 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad b_1 = \begin{bmatrix} 10 \\ 11 \\ 12 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ -1 & 0 & 10 \\ 4 & 5 & 0 \end{bmatrix} \quad b_2 = \begin{bmatrix} 0 \\ 1 \\ 2 \\ -4 \end{bmatrix}$$

- **Soluzione del test**:
  - Molto semplicemente si sfrutta una peculiare istruzione per il calcolo matriciale
  - in ambiente Matlab il comando **A/B** equivale a risolvere il problema

$$A\,x \;=\; b$$

sia nel caso di matrice A quadrata, sia nel caso generale di matrice A non quadrata (quindi soluzione ai minimi quadrati).

- Si veda **help mldivide**

Analisi e simulazione di sistemi dinamici in ambiente Matlab

Definizioni

Proprietà

Risolvere equazioni differenziali in ambiente Matlab

- Equazioni differenziali ordinarie (ODE) con condizioni iniziali assegnate:
  - come si codificano in ambiente Matlab
  - quale risolutore numerico scegliere?
  - analisi critica dei risultati (abbiamo ottenuto quello che ci si aspettava)?
  - Come visualizzare i risultati
- Alcuni esempi

- **NON** si parla in dettaglio di: metodi numerici per la risoluzione di equazioni differenziali, ODE, Matlab in generale.

Equazioni differenziali ordinarie - ODE

- Una ODE è un'equazione differenziale in cui la funzione incognita è funzione di una singola variabile indipendente (per es. il tempo)

$$\dot{y}(t) = f\left(t, y(t)\right)$$

- In molti casi esiste la soluzione ma non in forma chiusa: la si può calcolare tuttavia numericamente, tramite un algoritmo di calcolo.
- Esistono in generale molti risolutori numerici, quindi si tratta di scegliere quello più adatto al particolare problema che si vuole risolvere (per dettagli si rimanda per esempio al corso di "Calcolo numerico")

Risoluzione di una equazione differenziale per via numerica: cenni

- Soluzione per via numerica: il metodo individua un certo numero di "istanti d'integrazione" (= istanti in cui vene stimato il valore della funzione soluzione dell'equazione differenziale) e calcola la soluzione soltanto in questi istanti

$$\dot{y} = f(y,t); \quad t \in [T_0,\ T_f]\,; \quad y\,(T_0) = y_0$$

$$T_0 = t_1 < t_2 < \cdots < t_n = T_f$$

$$y_0,\ \hat{y}(t_2),\ \hat{y}(t_3), \cdots \hat{y}(t_n)$$

Risoluzione di una equazione differenziale per via numerica: cenni (2)

- La soluzione all'istante t_{i+1} è determinata sulla base di valori della soluzione calcolati in campioni precedenti

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

- la bontà della soluzione dipende dal passo h_i e dalla funzione Δ

$$h_i = t_{i+1} - t_i$$

Risoluzione (2): influenza del passo

- Dipendenza da *h* : riducendo *h* si aumenta la precisione del risultato → però così aumentano i tempi di calcolo!
  Inoltre un passo "eccessivamente piccolo" si scontra con i limiti imposti dagli errori di tipo numerico!

- Idea: utilizzare un passo piccolo soltanto dove sia necessario
  → **metodi a passo variabile**

Risoluzione (3): la funzione Δ

- **Metodo di Eulero**: approssima la derivata con il rapporto incrementale

$$y_{i+1} = y_i + h\, f\,(t_i,\, y_i)$$

Accuratezza: metodo d'ordine 1 (l'errore è infinitesimo di ordine 2 rispetto ad *h*)

Risoluzione (4): la funzione Δ

- **Metodi di Runge-Kutta:** la soluzione nel punto t_{i+1} è combinazione lineare della *f* valutata in punti interni all'intervallo $[t_i, t_{i+1}]$

$$y(t_{i+1}) = y(t_i) + h\, f\left(t_i + \frac{h}{2}, y_i + \frac{h}{2} f(t_i, y_i)\right)$$

- Esempio: metodo esplicito di ordine 2

Risoluzione (5): la funzione Δ

- **Metodi di Runge-Kutta espliciti:** espressione generale

$$
\begin{aligned}
y_{i+1} &= y_i + h\,\Delta(t_i,\, y_i,\, h,\, f) \\
\Delta(t_i,\, y_i,\, h,\, f) &= \sum_{q=1}^{m} c_q\, k_q \\
k_1 &= f(t_i,\, y_i) \qquad k_q = f\left(t_i + h\,a_q,\, y_i + h \sum_{j=1}^{q-1} b_{qj}\, k_j\right)
\end{aligned}
$$

Risoluzione (6): la funzione Δ

- **Metodi di Runge-Kutta impliciti:**

$$
\begin{aligned}
y_{i+1} &= y_i + h\,\Delta(t_i,\, y_i,\, h,\, f) \\
\Delta(t_i,\, y_i,\, h,\, f) &= \sum_{q=1}^{m} c_q\, k_q \\
k_1 &= f(t_i,\, y_i) \qquad k_q = f\left(t_i + h\,a_q,\, y_i + h \sum_{j=1}^{m} b_{qj}\, k_j\right)
\end{aligned}
$$

Risoluzione (7): la funzione Δ

- **Metodi "multistep":** nel calcolo di ogni nuovo valore della soluzione y_{i+1} vengono utilizzati diversi valori passati della soluzione y_i, y_{i-1}, y_{i-2},...

$$y_{i+1} = y_i + \sum_{j=0}^{m} \beta_i \, f(t_{i-j}, \, y_{i-j})$$

- Metodi di **Adams – Bashford – Moulton** (noti come metodi PECE [Prediction – Evaluation – Correction – Evaluation])

Panoramica dei risolutori a disposizione

- Metodi per **problemi non-*stiff***:
  - ode45 – algoritmo di Runge-Kutta (Dormand-Prince)
  - ode23 – algoritmo di Runge-Kutta (Bogacki-Shampine)
  - ode113 – metodo predictor-corrector multi-passo di Adams, con ordine da 1 a 13
  - ode15i – metodo per equazioni diff. implicite, ordine variabile
- Metodi per **problemi *stiff***:
  - ode23s - Runge-Kutta (Rosenbrock)
  - ode23t – regola dei trapezi
  - ode23tb - TR-BDF2
  - ode15s – metodo con ordine tra 1 e 5

Quale metodo scegliere?

Solver	Problema	Accuratezza	Quando usarlo
ode45	Non *stiff*	Media	Uso generico; da provare sempre come "*primo tentativo*".
ode23	Non *stiff*	Bassa	Quando è accettabile avere accuratezza bassa oppure per problemi moderatamente *stiff*.
ode113	Non *stiff*	Da bassa ad elevata	Per problemi con tolleranze d'errore molto piccole o per problemi con elevata complessità computazionale.
ode15i	Non *stiff*	Da bassa a media	Per problemi descritti da ODE implicite.

Quale metodo scegliere?

Solver	Tipo di problema	Accuratezza	Quando farne uso
ode15s	*Stiff*	Da bassa a media	Nei casi in cui **ode45** risulta lento, perché il problema è *stiff*.
ode23s	*Stiff*	Bassa	Nei casi in cui è accettabile avere tolleranze d'errore grossolane.
ode23t	Moderatamente *stiff*	Bassa	Per problemi moderatamente *stiff* se si vuole una soluzione priva di oscillazioni (numeriche)
ode23tb	*Stiff*	Bassa	Nei casi in cui è accettabile avere tolleranze d'errore grossolane.

Che cosa si intende per "equazione *stiff* "?

- Un'equazione differenziale viene detta "equazione **stiff** "
  - se si verifica che alcuni metodi numerici utilizzati per risolvere l'ODE sono numericamente instabili, a meno che l'ampiezza del passo con cui si calcola la soluzione non venga resa molto piccola.

- In sostanza una ODE si etichetta come " **stiff** " quando essa contiene dei termini che possono portare a variazioni molto rapide nella soluzione della ODE stessa.

Come si utilizzano i risolutori numerici a disposizione in Matlab

- La sintassi generale è

  [t,y] = solver(@odefun, time interval, y_0, options)

  - odefun: funzione da integrare
  - time interval: l'intervallo di integrazione, assegnato come $[t_0,t_f]$: valori iniziale e finale

  - y_0: il vettore delle condizioni iniziali.
  - options: struttura che contiene parametri opzionali che permettono di variare il comportamento dei risolutori.

Un primo esempio

- Si consideri il problema descritto da: $\begin{cases} \ddot{y} + \dot{y} = 0 \\ y(0) = 2\,, \quad \dot{y}(0) = 0 \end{cases}$
- Si riscrive la ODE come un sistema di equazioni differenziali del primo ordine

$$\begin{cases} \dot{y}_1(t) = y_2(t) \\ \dot{y}_2(t) = -y_1(t) \end{cases} \qquad \begin{cases} y_1(0) = 2 \\ y_2(0) = 0 \end{cases}$$

```
function dy_dt = odefun(t,y)
dy_dt = [y(2); -y(1)];
```

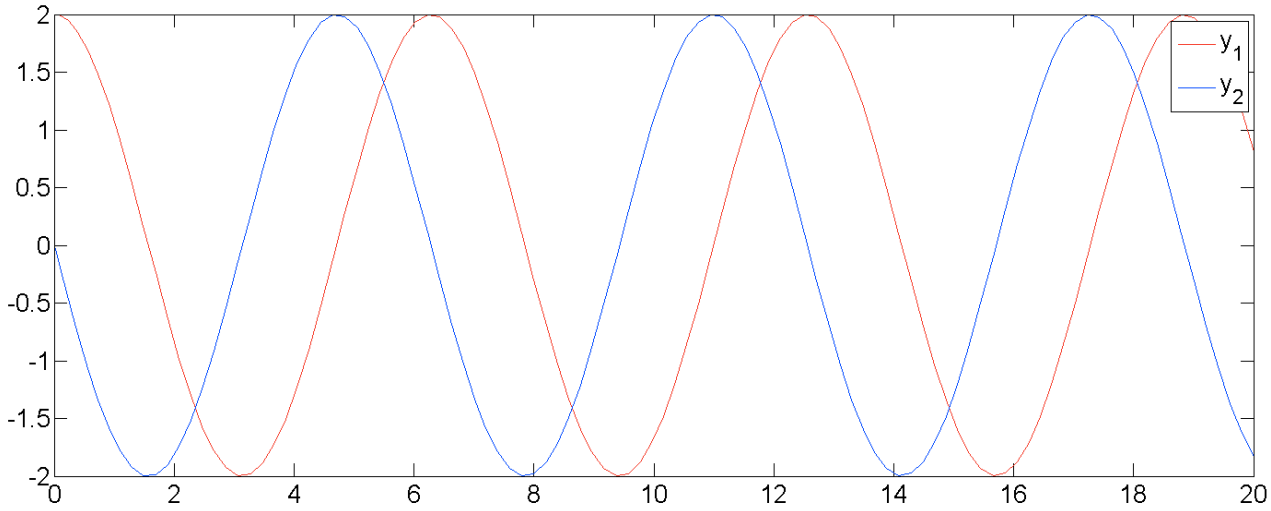
- si applica un risolutore:

```
[t,y] = ode45(@odefun, [0,20], [2,0]);
```

- l'algoritmo determina una sequenza di istanti temporali ed in corrispondenza di questi fornisce i valori della soluzione

- per visualizzare il risultato allora

```
plot(t, y(:,1),'r',t,y(:,2),'b')
title('soluzione della equazione di van der Pol');
xlabel('tempo t'); ylabel('soluzione y');
legend('y_1','y_2')
```



Opzioni

- Esistono diverse possibili opzioni con le quali configurare i vari algoritmi di soluzione:
  - la funzione odeset permette di variare i parametri secondo la sintatti

    opts=odeset('name1','value1','name2','value2',. . .)

  - permette anche di aggiungere opzioni ad un insieme di parametri già esistente

    opts=odeset(old opts,'name','value')
- Se non si assegna alcuna opzione, vengono utilizzati i valori prestabiliti.

nome	significato	valore predefinito
RelTol	tolleranza d'errore relativo	10^{-3}
AbsTol	tolleranza d'errore assoluto	10^{-6}
Refine	fattore di raffinamento dell'uscita	1 (4)
MaxStep	estremo superiore per il passo di risoluzione	
Stats	visualizza la statistica di costo computazionale	off

ad ogni passo si ha che la stima dell'errore commesso deve soddisfare

$$e_k \leq \max\{\mathrm{RelTol}\, y_k, \mathrm{AbsTol}\}$$

dove y_k è il valore della soluzione al passo k

Esempio: un sistema "stiff"

- Consideriamo l'equazione differenziale del secondo ordine

$$a_2 \ddot{y} + a_1 \dot{y} + a_0 y = b_0 u$$

- I coefficienti valgono

$$a_2 = 6.0\,10^{-6},\ a_1 = 1.8,\ a_0 = 0.4,\ b_0 = 0.5$$

- Supponiamo condizioni iniziali tutte nulle e segnale u pari a

$$u(t) = 2 \cdot 1(t)$$

Simulazione

- Provare ad eseguire il modello “esempio_stiff” con un metodo di risoluzione non-stiff (per es. con “ode45”). Notare la lentezza di calcolo e l’inaccuratezza della soluzione proposta.

- Riprovare, utilizzando uno dei metodi di risoluzione per problemi stiff, ad esempio “ode15s”. Notare la rapidità di calcolo e l’andamento della soluzione rispetto a quello ottenuto in precedenza.

Ancora un esempio di problema stiff

- oscillatore di Van der Pol

$$\begin{cases} \dot{y}_1(t) &= y_2(t) \\ \dot{y}_2(t) &= \mu\left(1 - y_1^2(t)y_2(t) - y_1(t)\right) \end{cases}$$

- con μ = 1000:

```
function dy_dt = vdp_ode(t,y,mu)
dy_dt = [y(2); mu*(1-y(1).^2).*y(2)-y(1)];
```

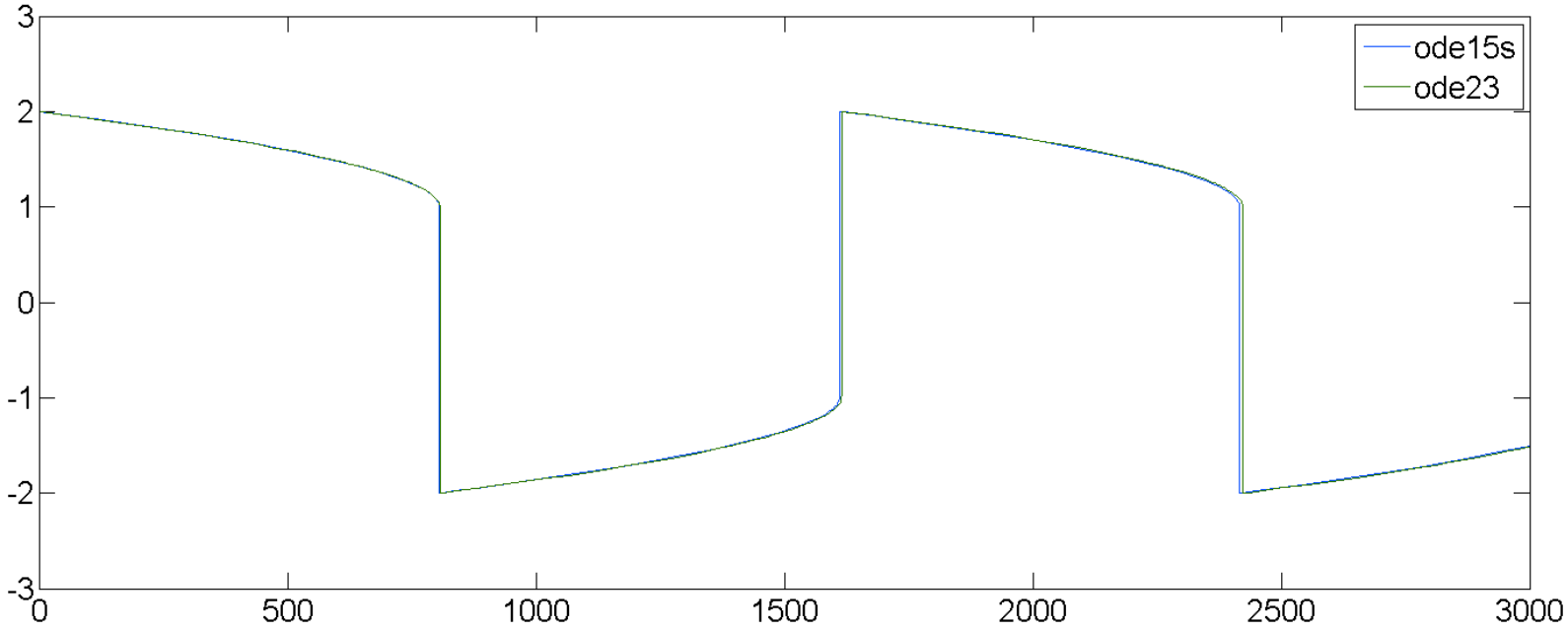
- applicando un primo **risolutore non stiff** (**1087 s** circa)

```
[t,y]=ode23(@(t,y)vdp_ode(t,y,1000),[0,3000],[2,0]);
```

- un **risolutore per problemi stiff** (in circa **737 ms**)

```
[t,y]=ode15s(@(t,y)vdp_ode(t,y,1000),[0,3000],[2,0]);
```

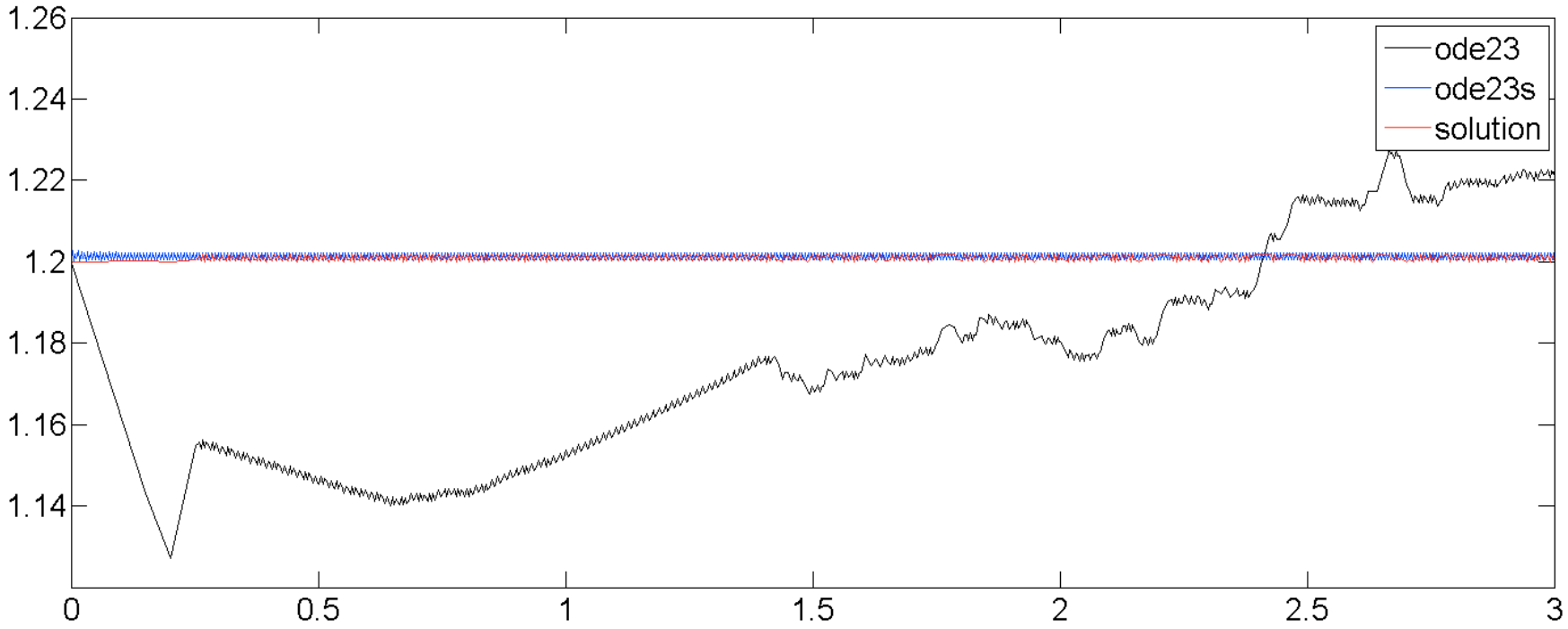
- entrambi forniscono una stima buona della soluzione esatta



Un esempio (con risultato errato!)

- $\dot{y}(t) = \sin(1000\,t)\ , \quad y(0) = 1.2$
- la soluzione analitica è $y(t) = \dfrac{-\cos(1000\,t) + 1201}{1000}$
- utilizzo 2 risolutori differenti ed ottengo 2 soluzioni differenti (una allora è errata!)

```
[t,y]=ode23(@(t,y)sin(1000*t),[0,3],1.2);
[t,y]=ode23s(@(t,y)sin(1000*t),[0,3],1.2);
```


1.26
1.24
1.22
1.2
1.18
1.16
1.14
0
0.5
1
1.5
2
2.5
3
ode23
ode23s
solution

Esercizi

$$\dot{y}(t) + 2y(t) = \sin(t) + \exp(-5t)\,, \quad y(0) = 0$$

$$\dot{y}(t) + y(t) \cdot \tan^{-1}(t) = 5\exp\left(\cos(t)\right)\,, \quad y(0) = 0$$

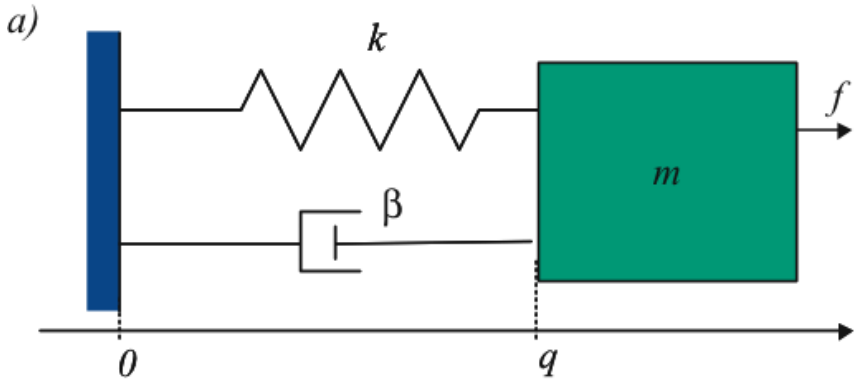
$$\ddot{y}(t) - 4\dot{y}(t) + 5y(t) = \exp\left(-2t\right) \cdot \tan(t)\,, \quad y(0) = 0\,,\ \dot{y}(0) = 0$$

$$2\ddot{y}(t) + y(t) = 2\tan(t)\,, \quad y(0) = 0\,,\ \dot{y}(0) = 1$$

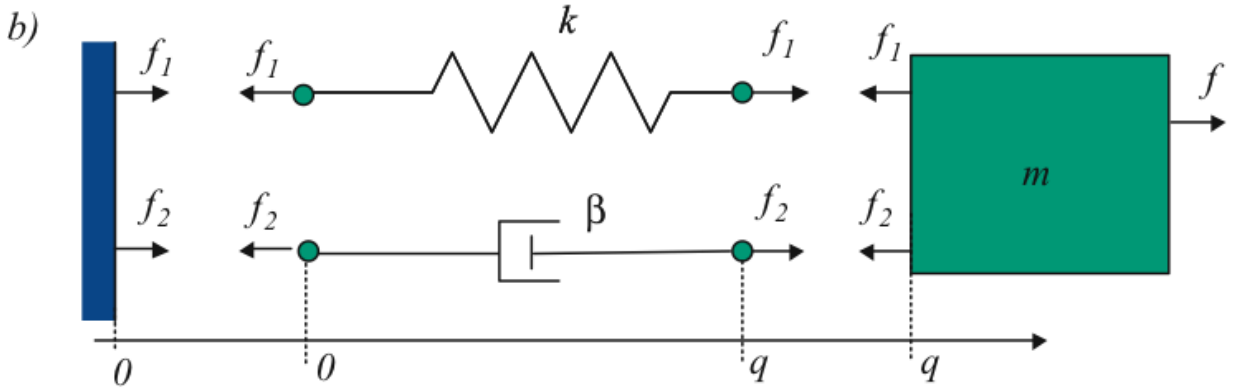
Simulazione di sistemi dinamici in ambiente Matlab

Esempi risolti

Esempio 1 (massa-molla-smorzatore)



(passando a diagrammi corpi liberi)



$$f_1 = k(q - 0)$$

$$f_2 = \beta \dot{q}$$

$$f - f_1 - f_2 = m\ddot{q}$$

$$f - kq - \beta\dot{q} = m\ddot{q}$$

$$m\ddot{q} + \beta\dot{q} + kq = f$$

$$m\ddot{q} + \beta\dot{q} + kq = f$$

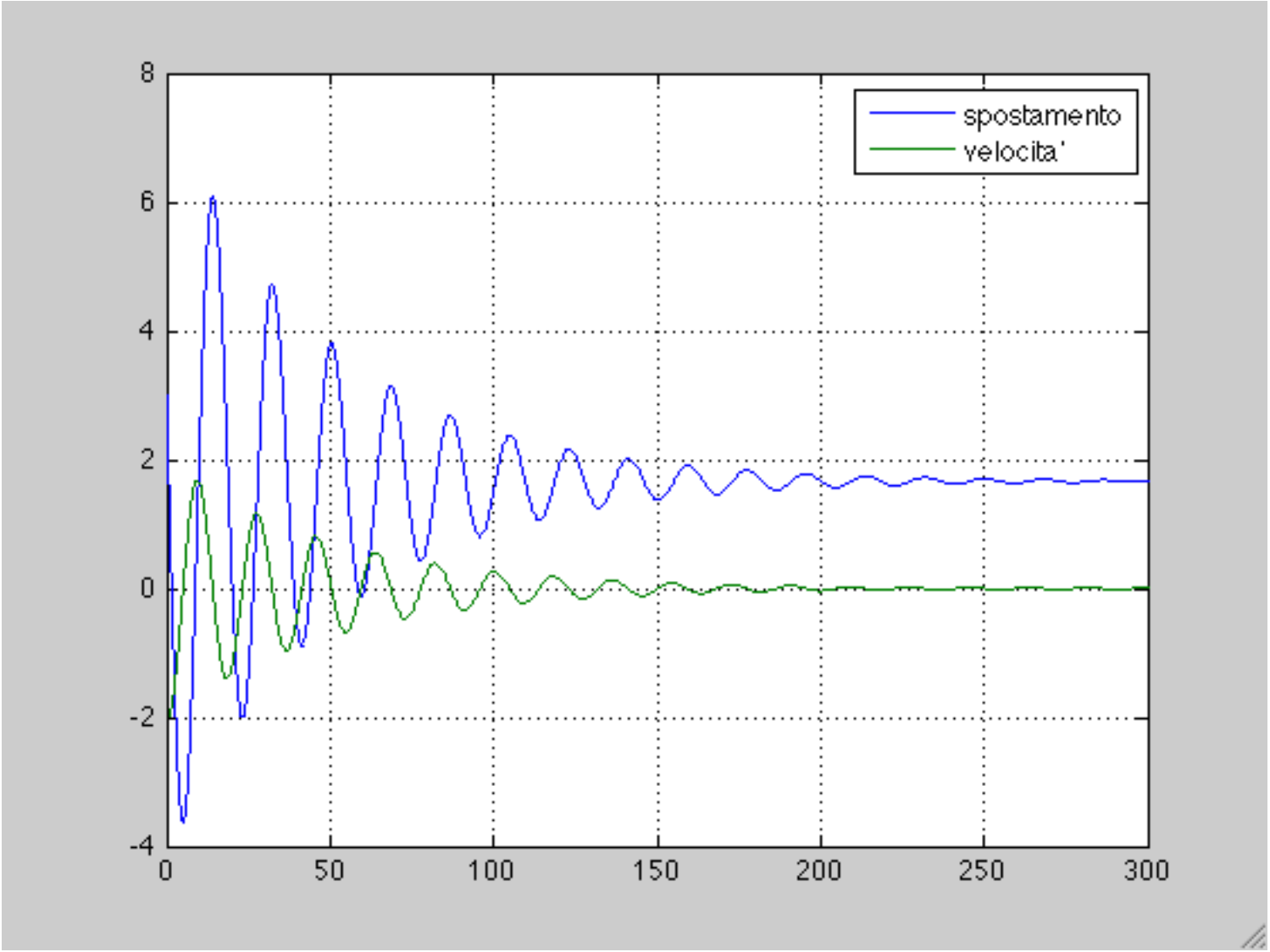
$$\begin{cases} \dot{x}_1(t) &= x_2(t) \\ \dot{x}_2(t) &= -\frac{\beta}{m}x_2(t) - \frac{k}{m}x_1(t) + \frac{1}{m}f(t) \\ y_1(t) &= x_1(t) \\ y_2(t) &= x_2(t) \end{cases}$$

```
function dydt = TDSC_ode_esempio1(t, y, param1, param2)
m = param1(1); beta = param1(2); k = param1(3);
% param2(1) : forzante f
%              0 -> scalino di ampiezza pari al valore di param2(2)
%              1 -> segnale sinusoidale con ampiezza pari al valore di
%                   param2(2) e frequenza [Hz] pari al valore di param2(3)
%              2 -> combinazione dei precedenti; ampiezza dello scalino in
%                   param2(2), della sinusoide in param2(3), frequenza [Hz]
%                   della sinusoide in param2(4)
f_id = param2(1);
switch f_id
    case 0
        f = param2(2);
    case 1
        f = param2(2) * sin(2*pi*param2(3)*t);
    case 2
        f = param2(2) + param2(3) * sin(2*pi*param2(4)*t);
end % case
 dydt = [y(2); (-beta*y(2) - k * y(1) + f)/m];
```

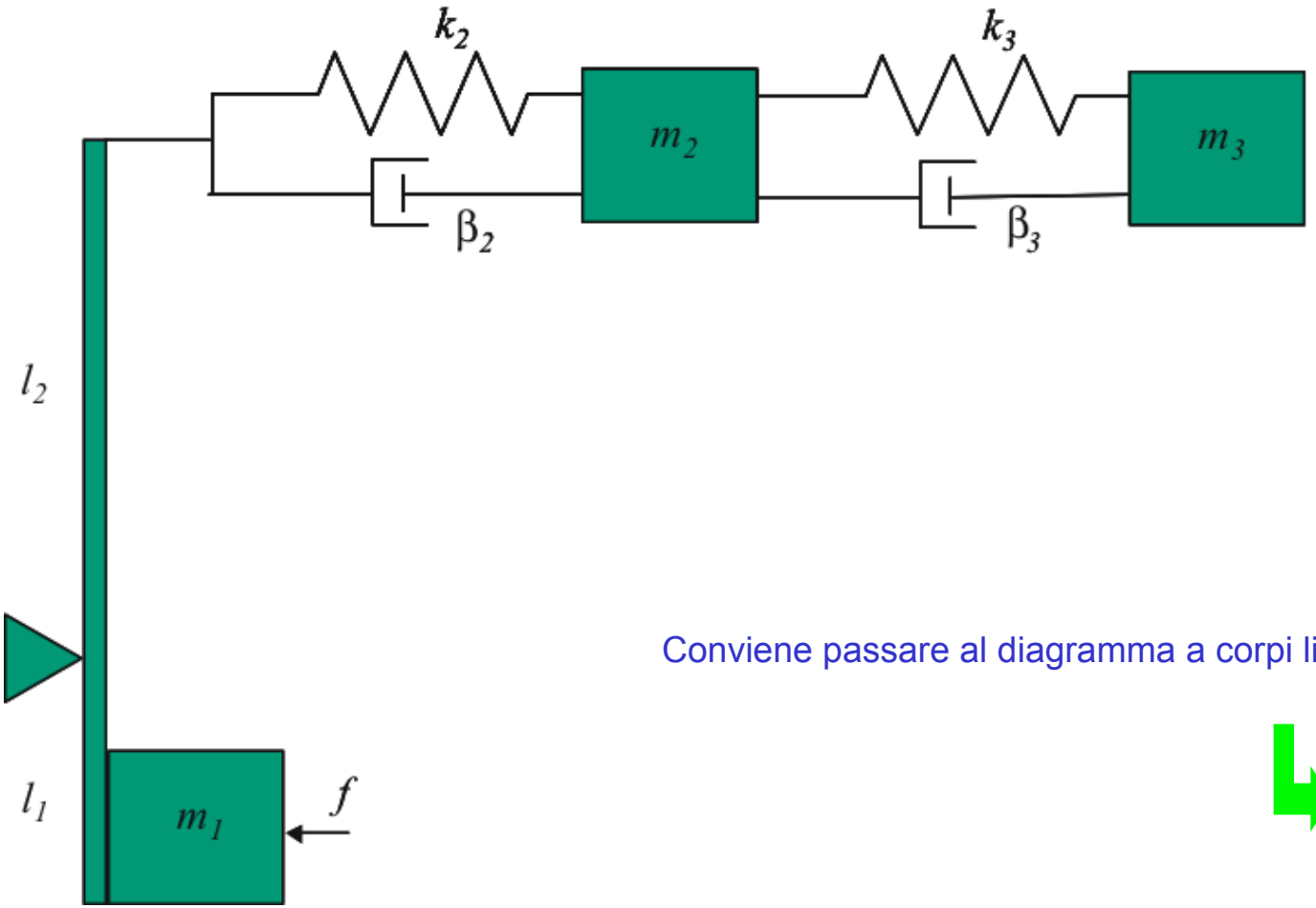
```
% script TDSC_ode_esempio1
%
% corso TDSC - a.a. 2010/2011
%
% script di soluzione della ODE di esempio1

% configurazione del problema
m = 5; % massa [kg]
b = 0.2; % attrito viscoso [N s/m]
k = 0.6; % costante elastica [N/m]
param1 = [m b k];
param2=[0 1];
ode_time = [0 300];
ode_y0 = [3; -2];
ode_options = odeset('MaxStep', 1e-1);

[t,Y] = ode45(@(t,y)TDSC_ode_esempio1(t, y, param1,
param2), ...
    ode_time, ode_y0, ode_options);

% visualizzazione dei risultati
figure;plot(t,Y);grid on
```

Così si impone al risolutore di utilizzare un passo (intervallo di tempo tra 2 campioni successivi della soluzione) non più grande di 0.1 s

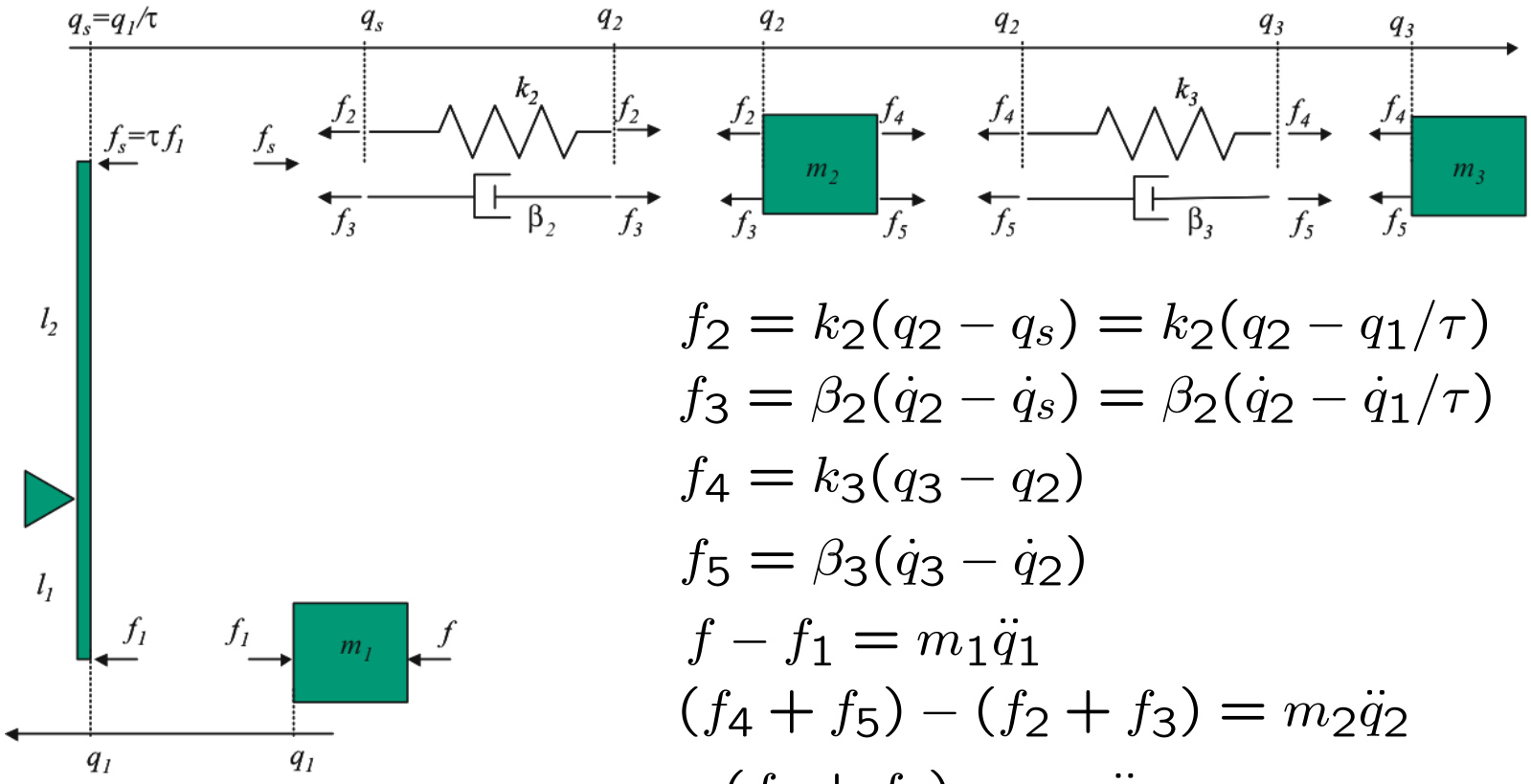
Si noti l'utilizzo di una *anonymous function* @(t, y). Così si possono passare parametri alla function che realizza la ODE.
Per dettagli: nella "*Command window*" di Matlab scrivere **help function_handle**


spostamento
velocita'
8
6
4
2
0
-2
-4
0
50
100
150
200
250
300

Esempio 2 (sist. meccanico tre masse)



Conviene passare al diagramma a corpi liberi



$$f_2 = k_2(q_2 - q_s) = k_2(q_2 - q_1/\tau)$$
$$f_3 = \beta_2(\dot{q}_2 - \dot{q}_s) = \beta_2(\dot{q}_2 - \dot{q}_1/\tau)$$
$$f_4 = k_3(q_3 - q_2)$$
$$f_5 = \beta_3(\dot{q}_3 - \dot{q}_2)$$
$$f - f_1 = m_1\ddot{q}_1$$
$$(f_4 + f_5) - (f_2 + f_3) = m_2\ddot{q}_2$$
$$-(f_4 + f_5) = m_3\ddot{q}_3$$

$$(f_2 + f_3) = \tau f_1$$

$$m_1\ddot{q}_1 = f - \frac{1}{\tau}(k_2(q_2 - q_1/\tau) + \beta_2(\dot{q}_2 - \dot{q}_1/\tau))$$
$$m_2\ddot{q}_2 = (k_3(q_3 - q_2) + \beta_3(\dot{q}_3 - \dot{q}_2)) - (k_2(q_2 - q_1/\tau) + \beta_2(\dot{q}_2 - \dot{q}_1/\tau))$$
$$m_3\ddot{q}_3 = -(k_3(q_3 - q_2) + \beta_3(\dot{q}_3 - \dot{q}_2))$$

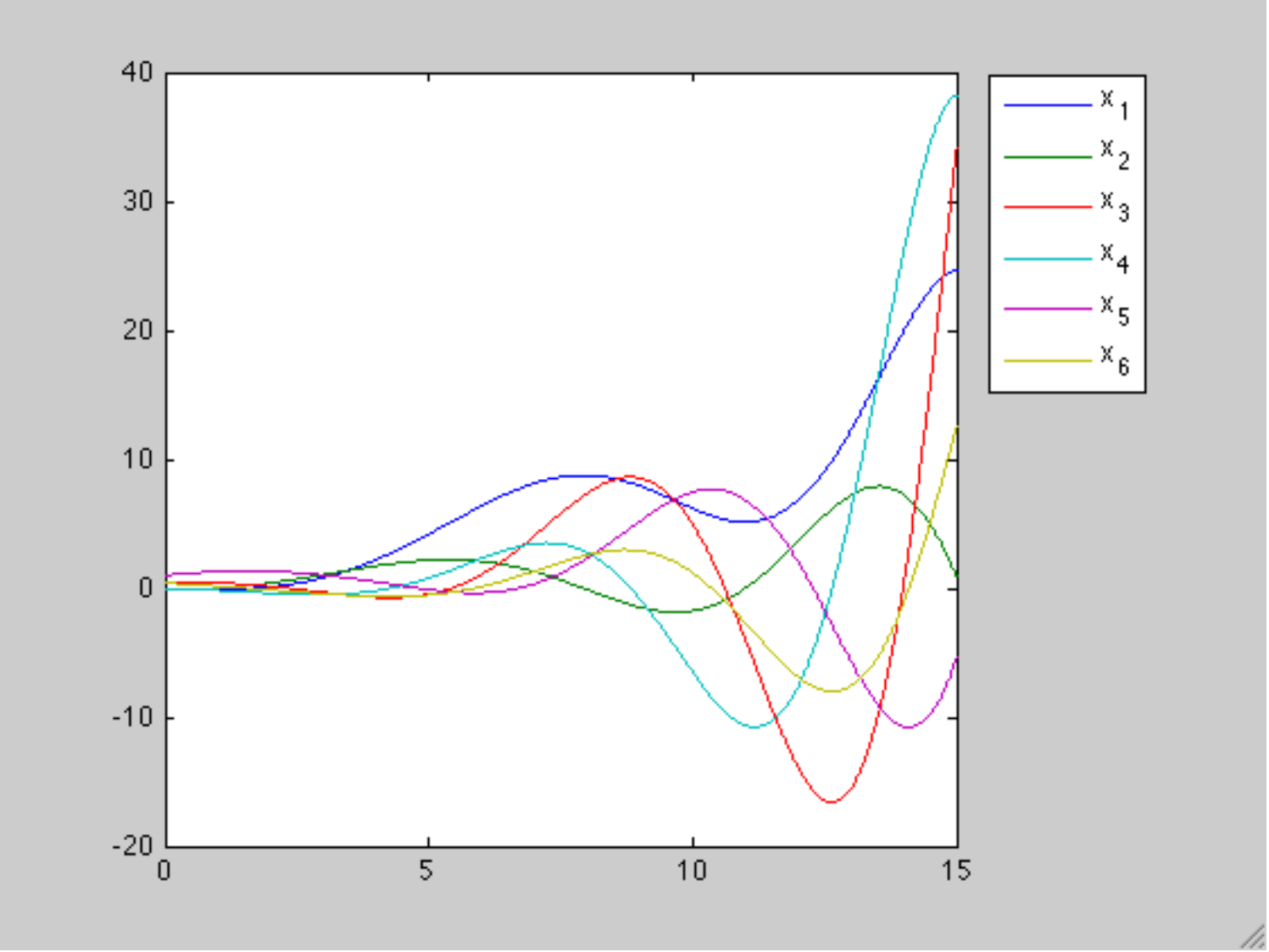
$$m_1\ddot{q}_1 = f - \frac{1}{\tau}(k_2(q_2 - q_1/\tau) + \beta_2(\dot{q}_2 - \dot{q}_1/\tau))$$
$$m_2\ddot{q}_2 = (k_3(q_3 - q_2) + \beta_3(\dot{q}_3 - \dot{q}_2)) - (k_2(q_2 - q_1/\tau) + \beta_2(\dot{q}_2 - \dot{q}_1/\tau))$$
$$m_3\ddot{q}_3 = -(k_3(q_3 - q_2) + \beta_3(\dot{q}_3 - \dot{q}_2))$$

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \frac{1}{m_1} f - \frac{1}{\tau m_1}\left[k_2\left(x_3 - \frac{x_1}{\tau}\right) + \beta_2\left(x_4 - \frac{x_2}{\tau}\right)\right] \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = \frac{1}{m_2}\left[k_3(x_5 - x_3) + \beta_3(x_6 - x_4)\right] - \frac{1}{m_2}\left[k_2\left(x_3 - \frac{1}{\tau}x_1\right)\right] + \frac{\beta_2}{m_2}\left(x_4 - \frac{1}{\tau}x_2\right) \\ \dot{x}_5 = x_6 \\ \dot{x}_6 = -\frac{k_3}{m_3}(x_5 - x_3) - \frac{\beta_3}{m_3}(x_6 - x_4) \end{cases}$$

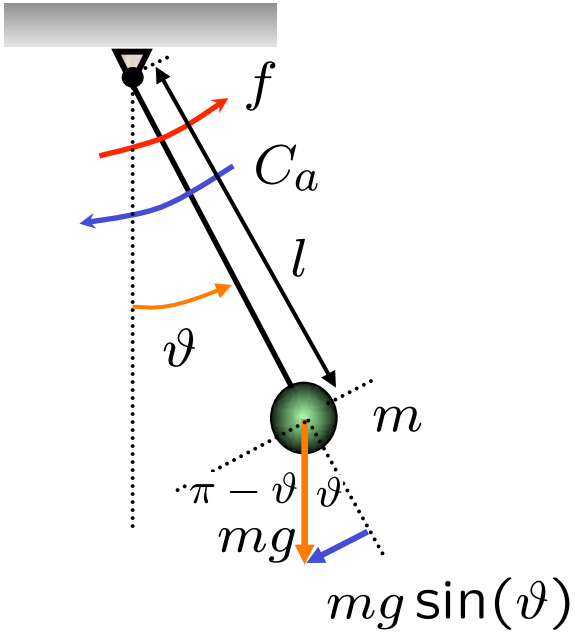
```
function dydt = TDSC_ode_esempio2(t, y, param1, param2)
 % [… omissis : file completo disponibile online]
 dydt = [y(2);...
         f/m1-(k2*(y(3)-y(1)/tau)+b2*(y(4)-y(2)/tau))/(tau*m1);...
         y(4);...
         (k3*(y(5)-y(3)+b3*(y(6)-y(4)))/m2-(k2/m2)*(y(3)-y(1)/tau)+
(b2/m2)*(y(4)-y(2)/tau));...
         y(6);...
         -(k3/m3)*(y(5)-y(3))-(b3/m3)*(y(6)-y(4))];

% ------------------------------------------------
```

```
% script TDSC_ode_esempio2
%
% corso TDSC - a.a. 2010/2011
%
% script di soluzione della ODE di esempio2

% un po' di pulizia
clear all
close all
clc

% configurazione del problema
m1 = 2; m2 = 1.1; m3 = 1.6;
 b2 = 1.0; b3 = .5;
 k2 = 0.8; k3 = .3;
 tau = 3;
param1 = [m1 m2 m3 b2 b3 k2 k3 tau];
param2=[1 1 0.1];
ode_time = [0 15];
ode_y0 = [0;-.1;0.5;0;1;0.4]; %[3; -2];
ode_options = odeset('MaxStep', 1e-2,'OutputFcn',@odeplot);
[t,Y] = ode113(@(t,y)TDSC_ode_esempio2(t, y, param1, param2),...
        ode_time, ode_y0,ode_options);
```

Assegnare **odeplot** come "*output function*" fa sì che il risolutore visualizzi la soluzione man mano che essa viene calcolata (aggiornando il grafico man mano che vengono determinati nuovi valori della soluzione).


40
30
20
10
0
-10
-20
0
5
10
15
x 1
x 2
x 3
x 4
x 5
x 6

Esempio 3 (pendolo semplice)



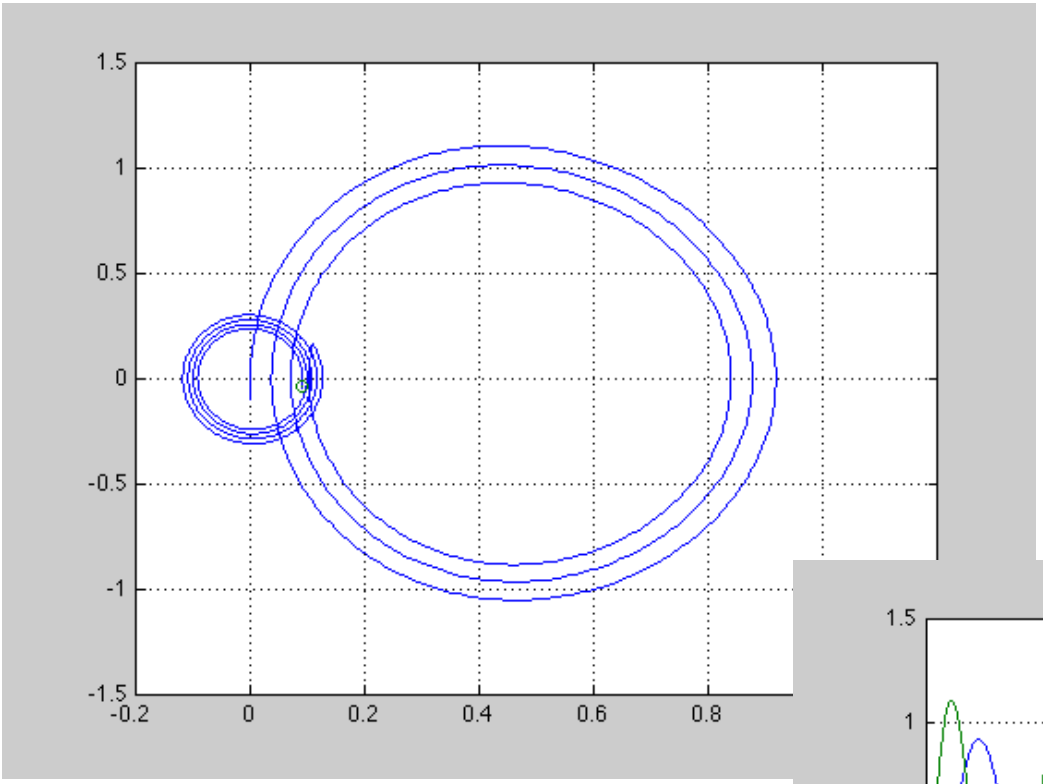
$$f - C_a - mgl\sin\vartheta = ml^2\ddot{\vartheta}$$

$$f - \beta\dot{\vartheta} - mgl\sin\vartheta = ml^2\ddot{\vartheta}$$

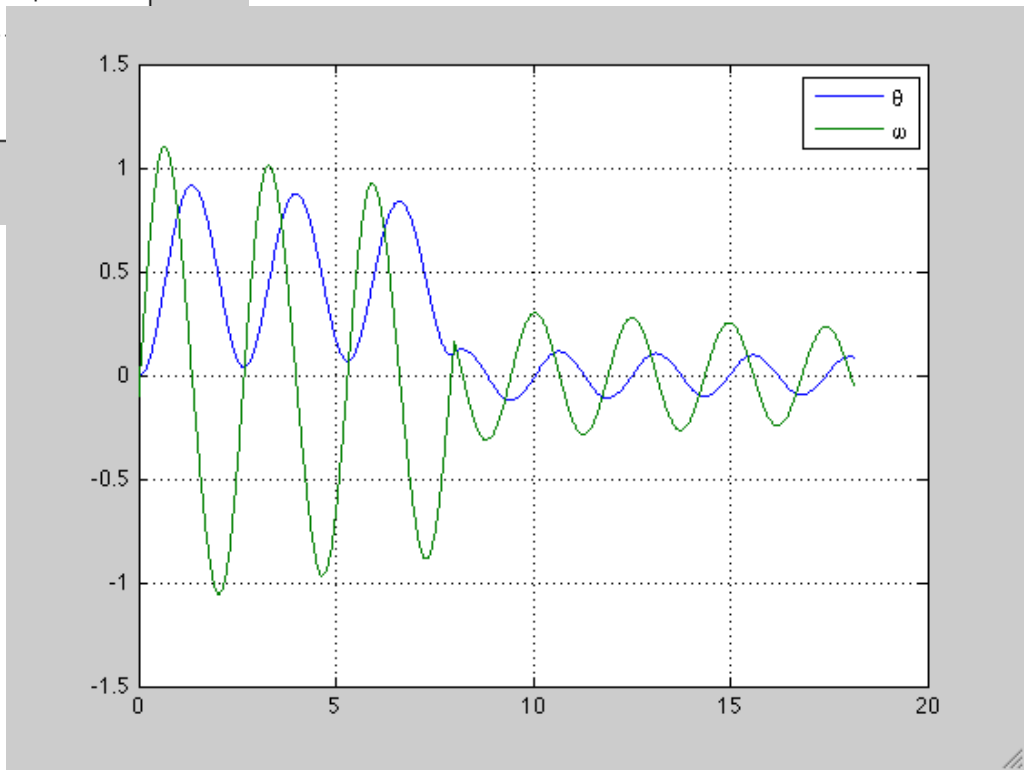
```
function dydt = TDSC_ode_esempio3(t, y, param1)
% [… omissis – testo completo disponibile online]
 g = 9.80665; % accelerazione di gravita'
 dydt = [y(2);...
         -(g/l)*sin(y(1))-(b/m/l^2)*y(2)+f/m/l^2];
%
-------------------------------------------------
```

```
% script TDSC_ode_esempio 3
%
% corso TDSC - a.a. 2010/2011
%
% script di soluzione della ODE di esempio3

% configurazione del problema
 m = 2;  b = 0.3; l = 1.5;
 F = 12.8; t_f = 8;

param1 = [m l b F t_f];

ode_time = [0 45];
ode_y0 = [0;-.1]; %[3; -2];
ode_options = odeset('MaxStep', 1e-2,'OutputFcn',@odephas2);

[t,Y] = ode23(@(t,y)TDSC_ode_esempio3(t, y, param1), ode_time,...
        ode_y0, ode_options);

figure;plot(t,Y);legend('\theta','\omega');grid
```

In questo modo la soluzione viene visualizzata nel "piano delle fasi" del sistema (piano i cui punti rappresentano univocamente tutti e soli i possibili stati [posizione e velocità angolari] del sistema).

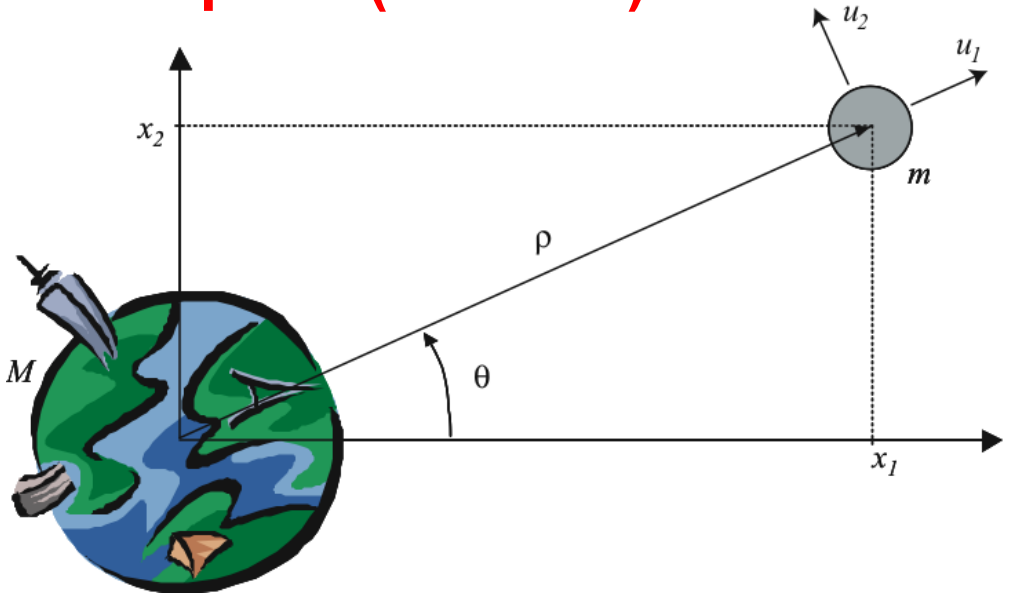


Nel piano delle fasi



Nel dominio del tempo

Esempio 4 (satellite)



$$f_g \simeq G\frac{Mm}{\rho^2}, \qquad G = 6.67 \cdot 10^{-11}$$

$M >> m$ ⇒ f_g agisce solo su m

$$m\ddot{r} = -G\frac{Mm}{\|r\|^2}\frac{r}{\|r\|} \quad \Rightarrow \quad \ddot{r} = -\frac{\mu}{\|r\|^3}r \quad \text{con} \quad \mu := GM$$

$$\rho = \|r\|$$

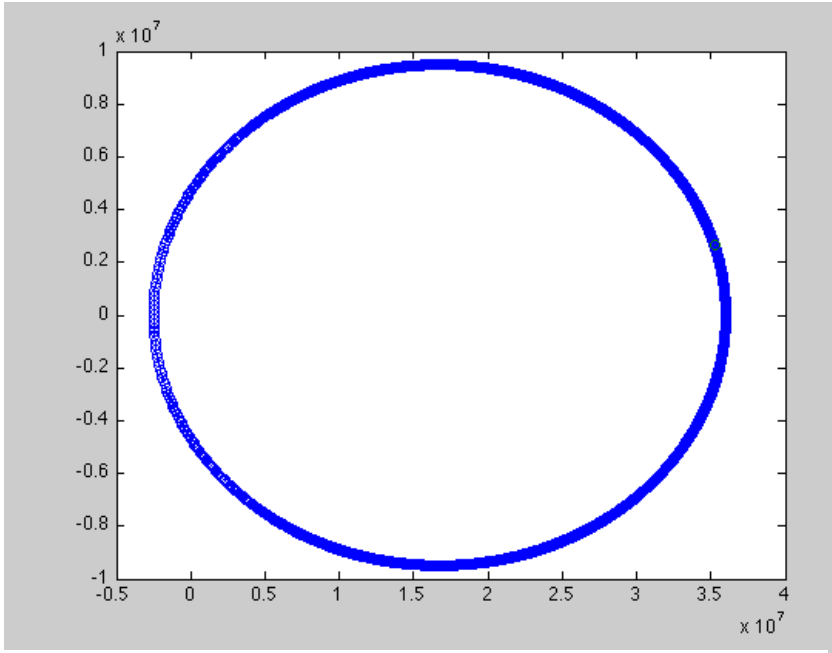
$$r = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \rho\cos\vartheta \\ \rho\sin\vartheta \end{bmatrix}$$

$$\ddot{x}_1 = -\mu\frac{x_1}{(x_1^2 + x_2^2)^{3/2}}$$

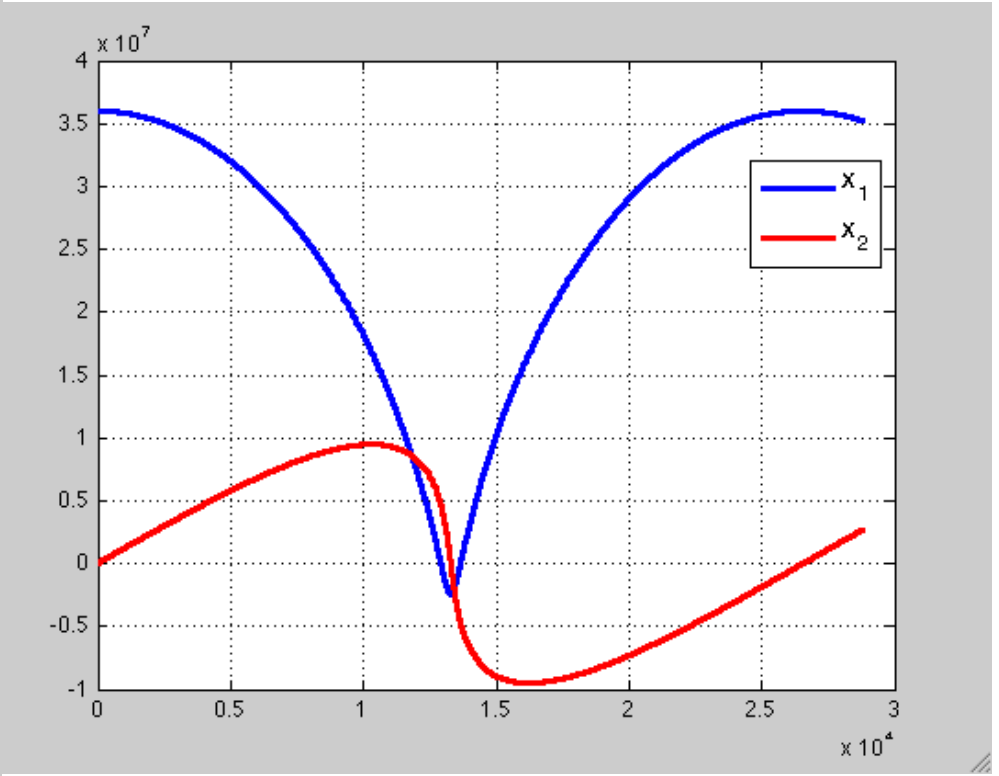
$$\ddot{x}_2 = -\mu\frac{x_2}{(x_1^2 + x_2^2)^{3/2}}$$

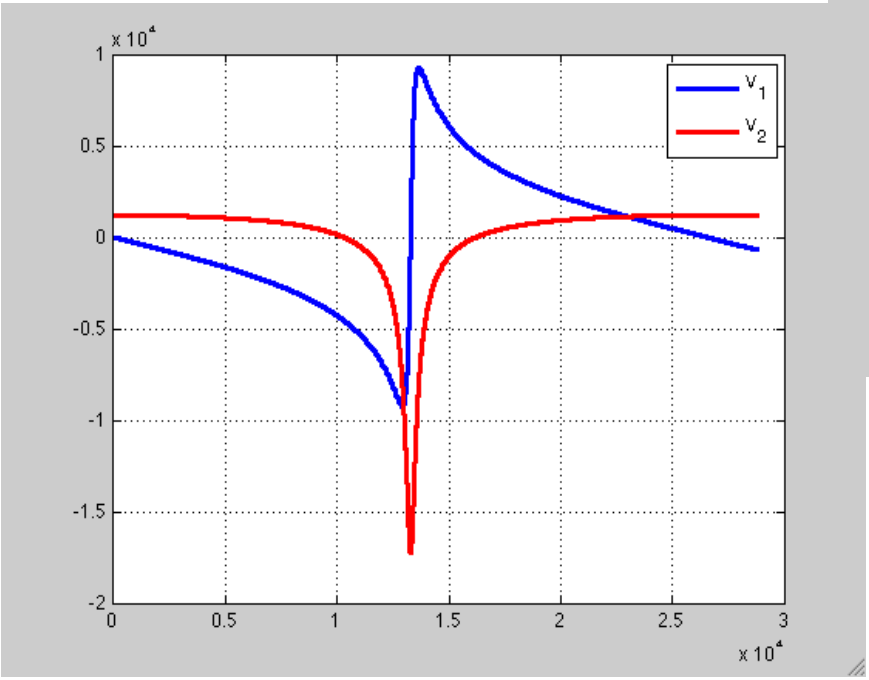
```
function dydt = TDSC_ode_esempio4(t, y)
% ODE dell'esempio 4 - satellite in orbita
% corso TDSC a.a. 2010/2011
%
% modello in equazioni di stato di un satellite in
orbita
M = 5.98e24; % kg - massa della Terra
G = 6.67e-11; % costante di gravitazione
mu = M*G;
% il sistema di equazioni differenziali che descrive
il sistema
 dydt = [y(2); -mu*y(1)/((y(1)^2+y(3)^2)^(3/2));...
                y(4); -mu*y(1)/((y(1)^2+y(3)^2)^(3/2))];
```

```
% script TDSC_ode_esempio 4
% corso TDSC - a.a. 2010/2011
% script di soluzione della ODE di esempio4
% configurazione del problema
ode_time = [0 28800];
ode_y0 = [36e6;0;0;1.2e3]; %[3; -2];
ode_options = odeset('MaxStep',
1e1,'OutputFcn',@odephas2,'OutputSel',[1 3]);
 [t,Y] = ode113(@(t,y)TDSC_ode_esempio4(t, y), ode_time, ode_y0,
ode_options);
figure;plot(t,Y(:,1),'b',t,Y(:,3),'r');grid
legend('x_1','x_2');
figure;plot(t,Y(:,2),'b',t,Y(:,4),'r');grid
legend('v_1','v_2');
```



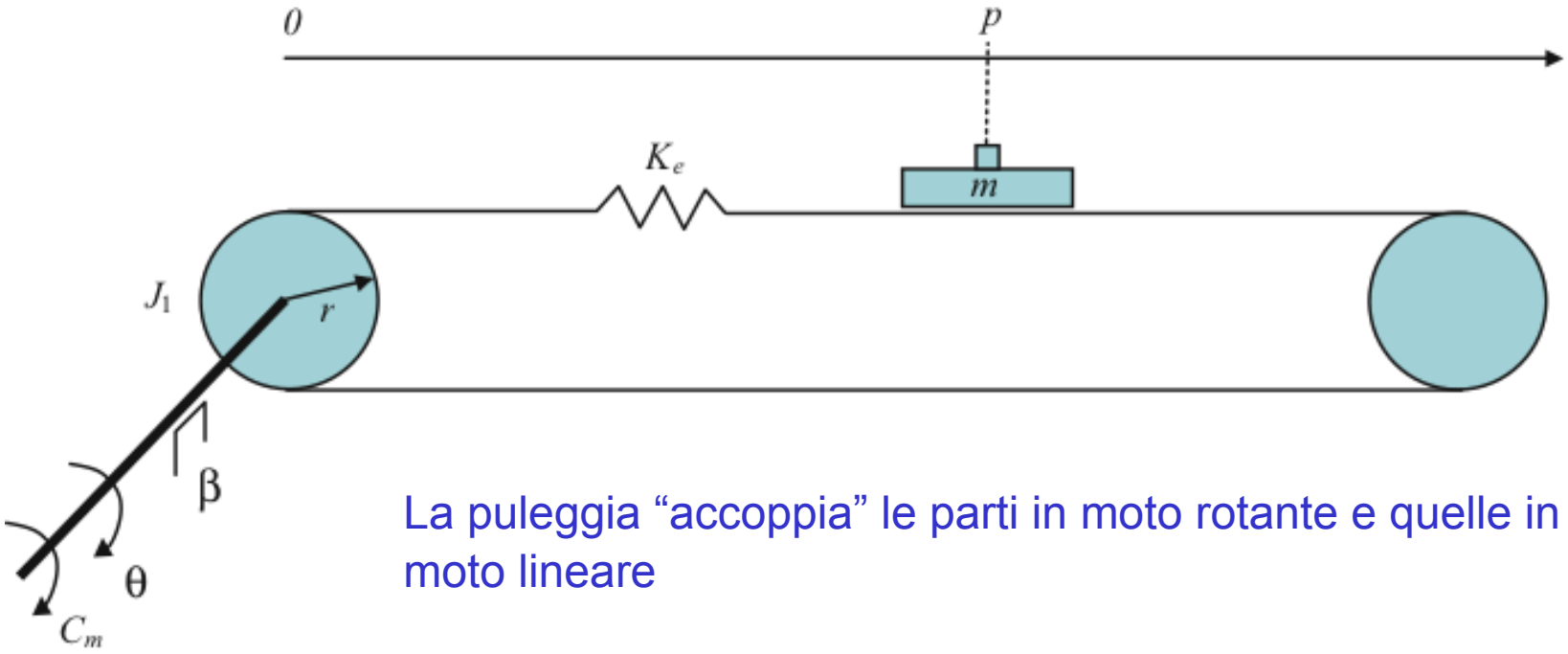
Nel piano delle fasi





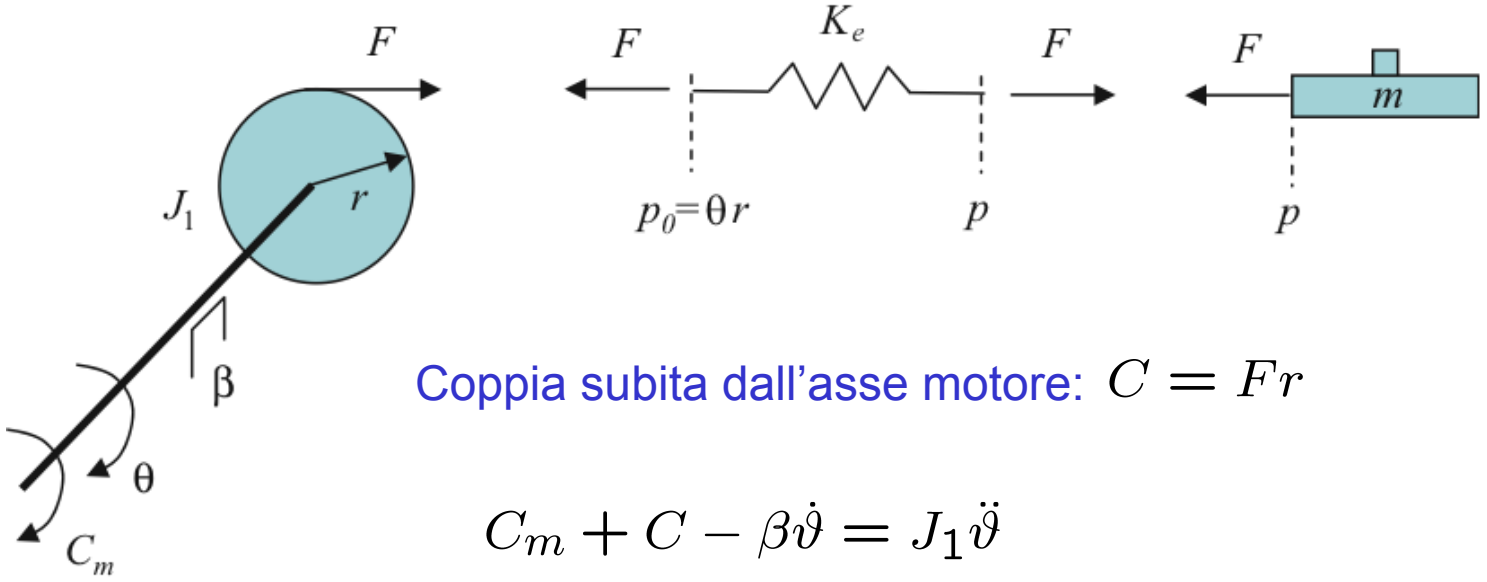
Nel dominio del tempo

Esempio 5 (testina stampante)



La puleggia “accoppia” le parti in moto rotante e quelle in moto lineare

Conviene passare al diagramma a corpi liberi



Coppia subita dall'asse motore: $C = Fr$

$$C_m + C - \beta\dot{\vartheta} = J_1\ddot{\vartheta}$$

$$-F = m\ddot{p}$$

$$F = K_e(p - r\vartheta)$$

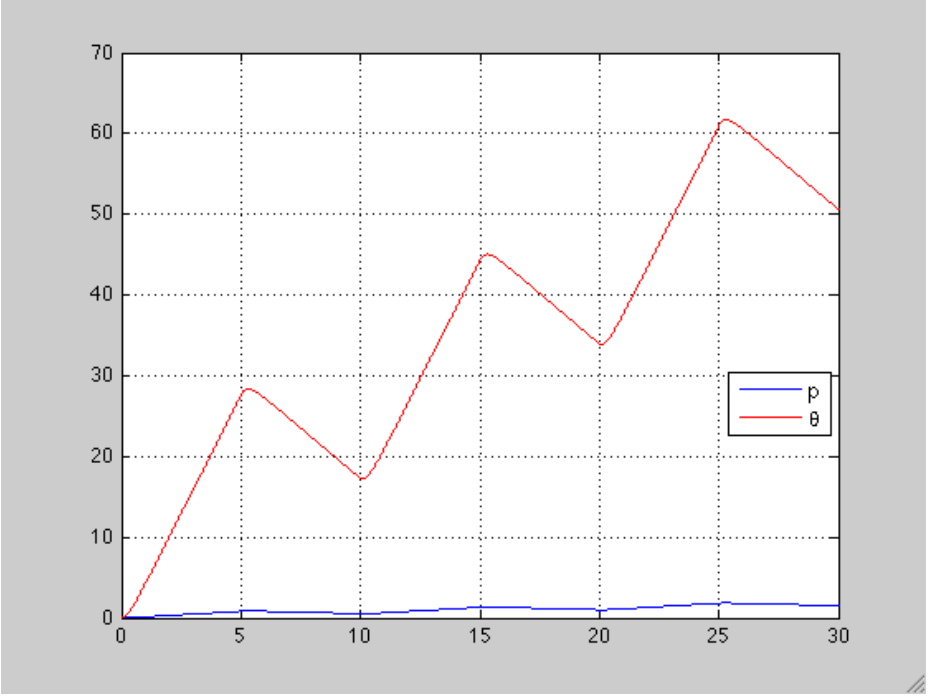
$$m\ddot{p} = -K_e p + rK_e\vartheta$$

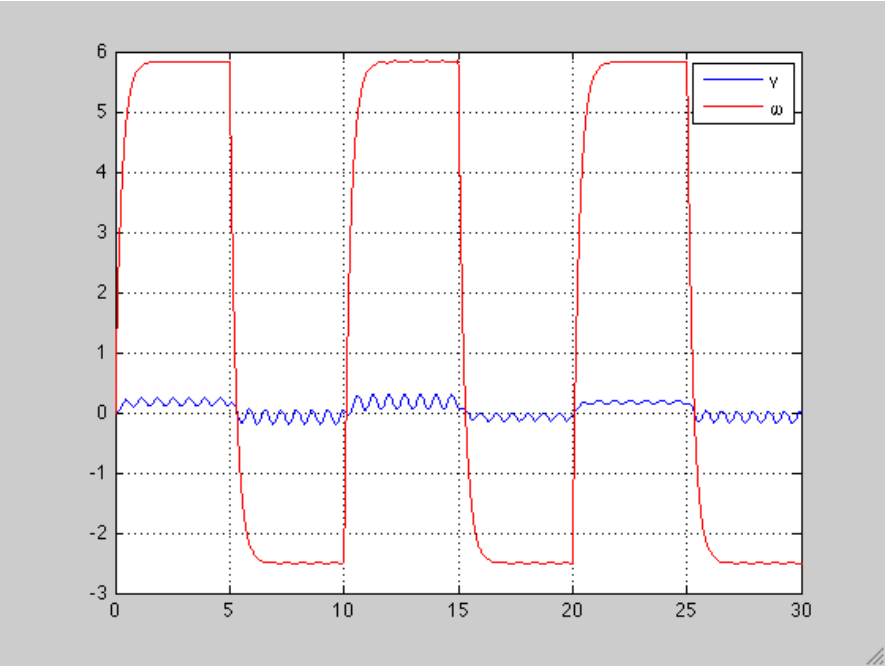
$$J_1\ddot{\vartheta} = rK_e p - r^2K_e\vartheta - \beta\dot{\vartheta} + C_m$$

```
function dydt = TDSC_ode_esempio5(t, y)
% ODE dell'esempio 5
% corso TDSC a.a. 2010/2011
K_e = 1e1; % costante elastica del nastro
m = 1.2e-1; % massa della testina [in kg]
J = 3.2e-2; % [kg m^2]
r = 3e-2; % [m]
b = 0.12;
u_step =0.2;
u_pulse = 0.5*sign(sin(0.1*2*pi*t));
u = u_step + u_pulse; % coppia pulsante in ingresso
% il sistema di equazioni differenziali che descrive il
sistema
 dydt = [y(2); (-K_e * y(1) + r* K_e * y(3) ) /m;...
         y(4); (r* K_e * y(1) - r^2*K_e * y(3) - b * y(4)
+ u )/J];
```

```
% script TDSC_ode_esempio 5
% [… omissis – script completo disponibile online]
ode_time = [0 30];
ode_y0 = [0;0;0;0];
ode_options = odeset('MaxStep', 1e-2);
[t,Y] = ode23(@(t,y)TDSC_ode_esempio5(t, y), ode_time, ode_y0,
ode_options);
figure;plot(t,Y(:,1),'b',t,Y(:,3),'r');grid
legend('p','\theta');
figure;plot(t,Y(:,2),'b',t,Y(:,4),'r');grid
legend('v','\omega');
```

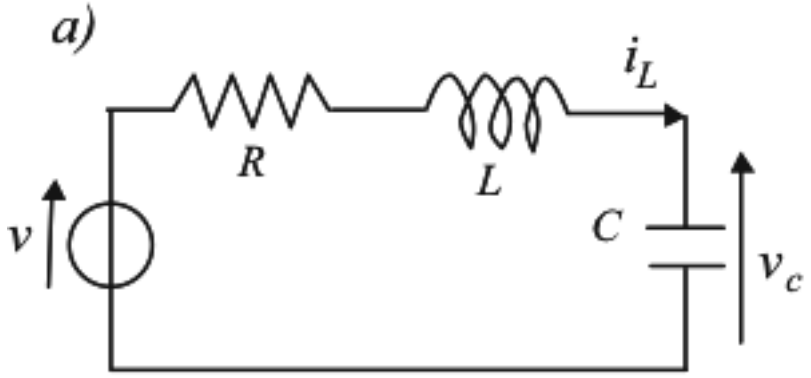
Posizione della testina ed angolo di rotazione della puleggia

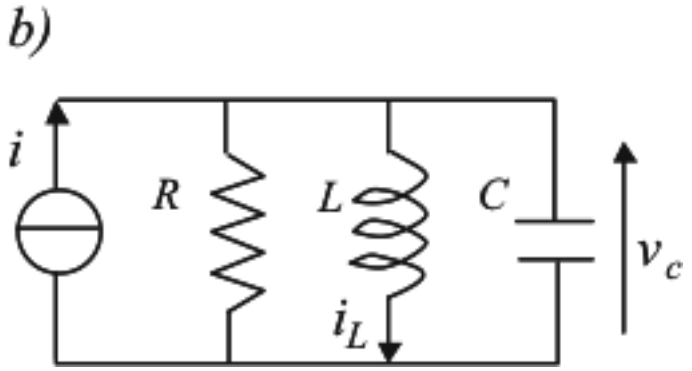




Velocità della testina e velocità di rotazione della puleggia

Esempio 6 (circuiti RLC risonanti)





Caso a): risonatore RLC in serie

$$L\frac{di_L}{dt} = v - Ri_L - v_C$$

$$C\frac{dv_C}{dt} = i_L$$

Caso b): risonatore RLC in parallelo

$$C\frac{dv_C}{dt} = i - \frac{1}{R}v_C - i_L$$

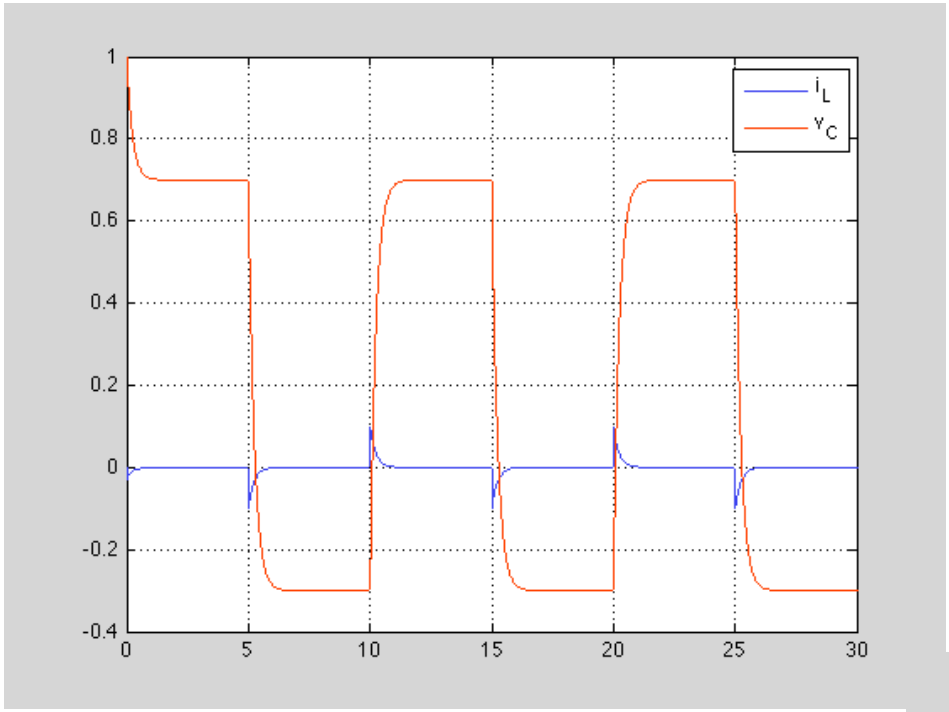
$$L\frac{di_L}{dt} = v_C$$

```
function dydt = TDSC_ode_esempio6a(t, y)
% ODE dell'esempio 6 - circuito RLC serie
R = 10; % Ohm
L = 1.2e-2; % H
C = 2.34e-2; % F

u_step =0.2;
u_pulse = 0.5*sign(sin(0.1*2*pi*t));
u = u_step + u_pulse; % tensione pulsante in
ingresso
%
-----------------------------------------------
-----
% il sistema di equazioni differenziali
% y(1) i_L; y(2) v_C
 dydt = [-R/L*y(1)-y(2)/L+u/L; y(1)/C];
```

```
% script TDSC_ode_esempio6a
% script di soluzione della ODE di esempio6a
% configurazione del problema
ode_time = [0 30];
ode_y0 = [0;1];
ode_options = odeset('MaxStep', 1e-2,'OutputFcn',@odephas2);
[t,Y] = ode15s(@(t,y)TDSC_ode_esempio6a(t, y), ode_time, ode_y0,
ode_options);
figure;plot(t,Y(:,1),'b',t,Y(:,2),'r');grid
legend('i_L','v_C');
```

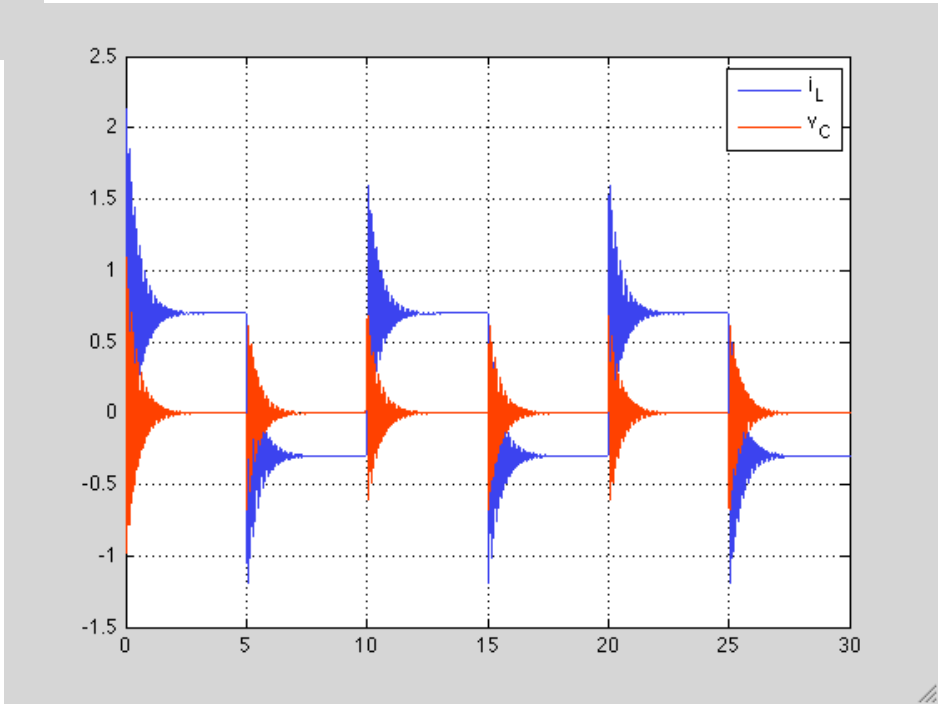
```
function dydt = TDSC_ode_esempio6b(t, y)
% ODE dell'esempio 6 - circuito RLC parallelo
R = 10; % Ohm
L = 1.2e-2; % H
C = 2.34e-2; % F

u_step =0.2;
u_pulse = 0.5*sign(sin(0.1*2*pi*t));
u = u_step + u_pulse; % corrente pulsante in
ingresso
%
------------------------------------------------
------
% il sistema di equazioni differenziali che
descrive il sistema
% y(1) i_L; y(2) v_C
 dydt = [y(2)/L;  (u-y(2)/R-y(1))/C];
```

```
% script TDSC_ode_esempio6b
ode_time = [0 30];
ode_y0 = [0;1];
ode_options = odeset('MaxStep', 1e-2,'OutputFcn',@odephas2);
[t,Y] = ode15s(@(t,y)TDSC_ode_esempio6b(t, y), ode_time, ode_y0,
ode_options);
figure;plot(t,Y(:,1),'b',t,Y(:,2),'r');grid
legend('i_L','v_C');
```



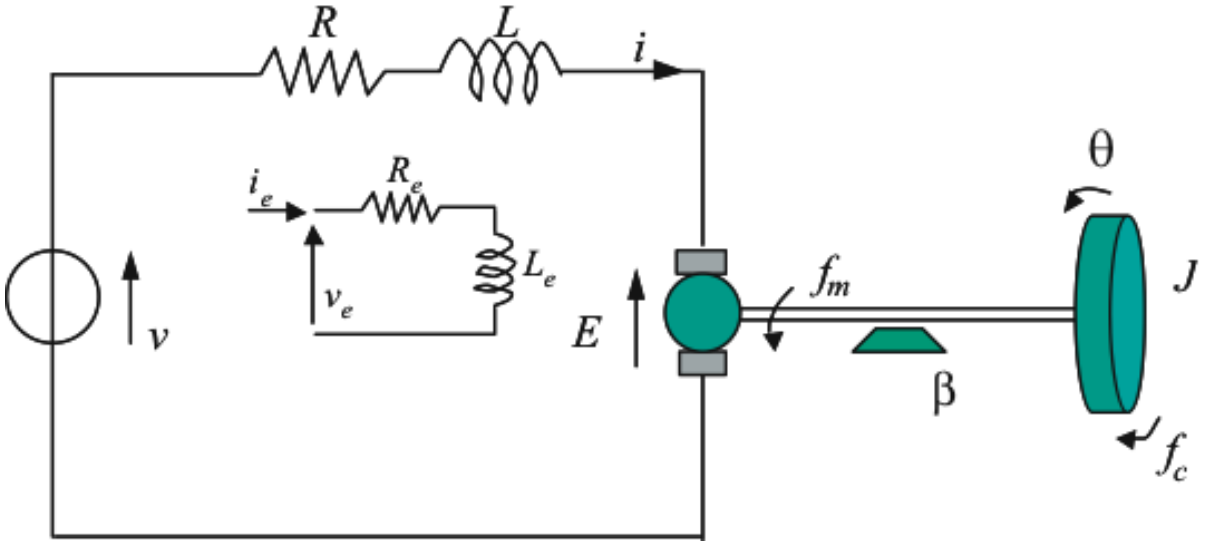
Il risuonatore - serie

Il risuonatore - parallelo



Esempio 7 (sistema elettromeccanico)

Motore C.C. con comando in armatura ed eccitazione costante



$$f_m(t) = K_f i(t)$$

$$\omega(t) = \dot{\vartheta}(t)$$

$$E(t) = K_e \omega(t)$$

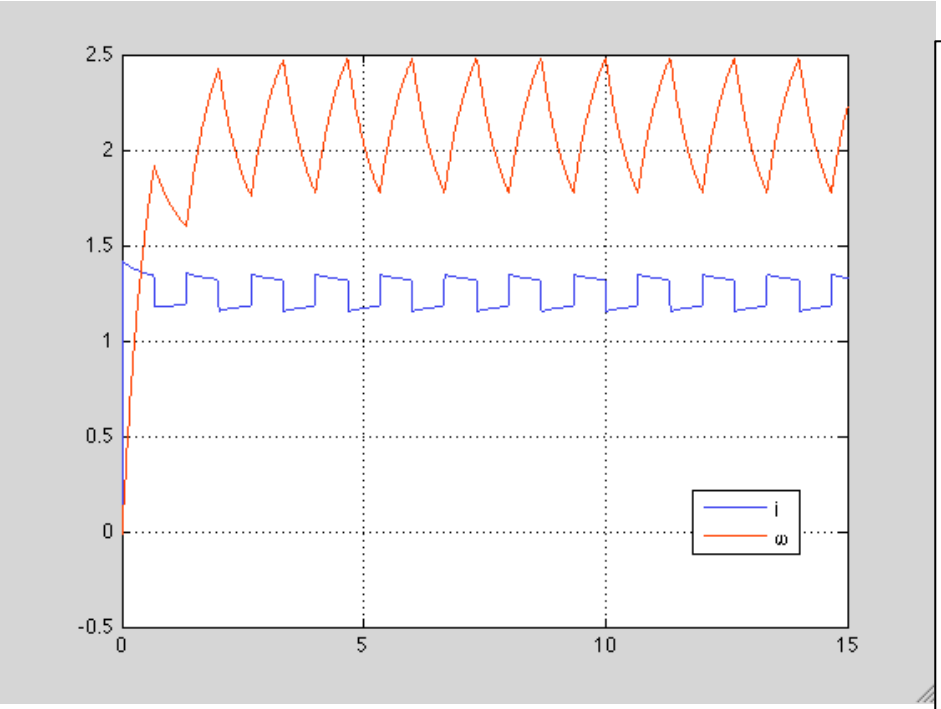
$$K := K_e \simeq K_f$$

$$v(t) - Ri(t) - L\frac{di(t)}{dt} - E(t) = 0$$

$$f_m(t) - f_c(t) - \beta\omega(t) = J\frac{d\omega(t)}{dt}$$

$$L\frac{di(t)}{dt} = -Ri(t) - K\omega(t) + v(t)$$

$$J\frac{d\omega(t)}{dt} = Ki(t) - \beta\omega(t) - f_c(t)$$

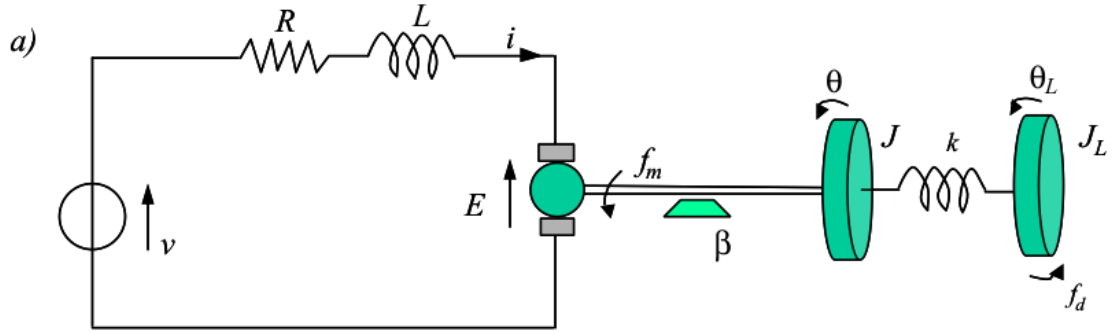



```
function dydt = TDSC_ode_esempio7(t, y)
% ODE dell'esempio 7 - motore c.c.
R = 6; % Ohm
L = 1.2e-2; % H
K = 2.34e-1; % V s /rad
b = 0.02; % N m/rad
J = 1.67e-2; % kg m^2
u_step = 8;
u_pulse = 0.5*sign(sin(0.75*2*pi*t));
u = u_step + u_pulse; % tensione
pulsante
f_c = 0.25; %  coppia sul carico [N m]
% y(1) i; y(2) omega
 dydt = [(-R*y(1)-K*y(2)+u)/L;...
          (K*y(1)-b*y(2)-f_c)/J];
```

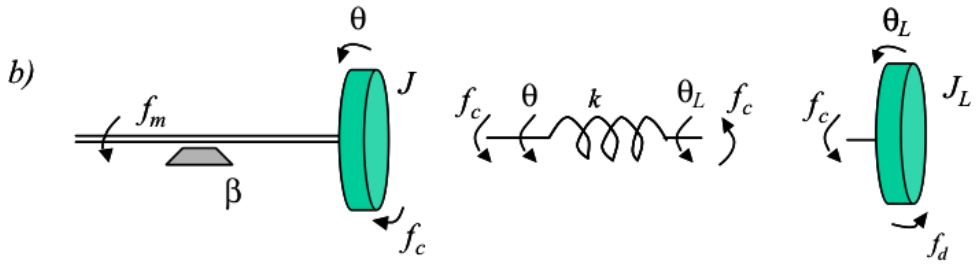
```
% script TDSC_ode_esempio7
% configurazione del problema
ode_time = [0 15];
ode_y0 = [0;0];
ode_options = odeset('MaxStep', 1e-2,'OutputFcn',@odephas2);

[t,Y] = ode15s(@(t,y)TDSC_ode_esempio7(t, y), ode_time, ode_y0,
ode_options);
figure;plot(t,Y(:,1),'b',t,Y(:,2),'r');grid; legend('i','\omega');
```

Esempio 8 (sistema elettromeccanico)

Motore C.C. con comando in armatura ed eccitazione costante con accoppiamento elastico al carico



$$f_c(t) = k(\vartheta - \vartheta_L)$$



$$L\frac{di(t)}{dt} = -Ri(t) - K\dot{\vartheta}(t) + v(t)$$

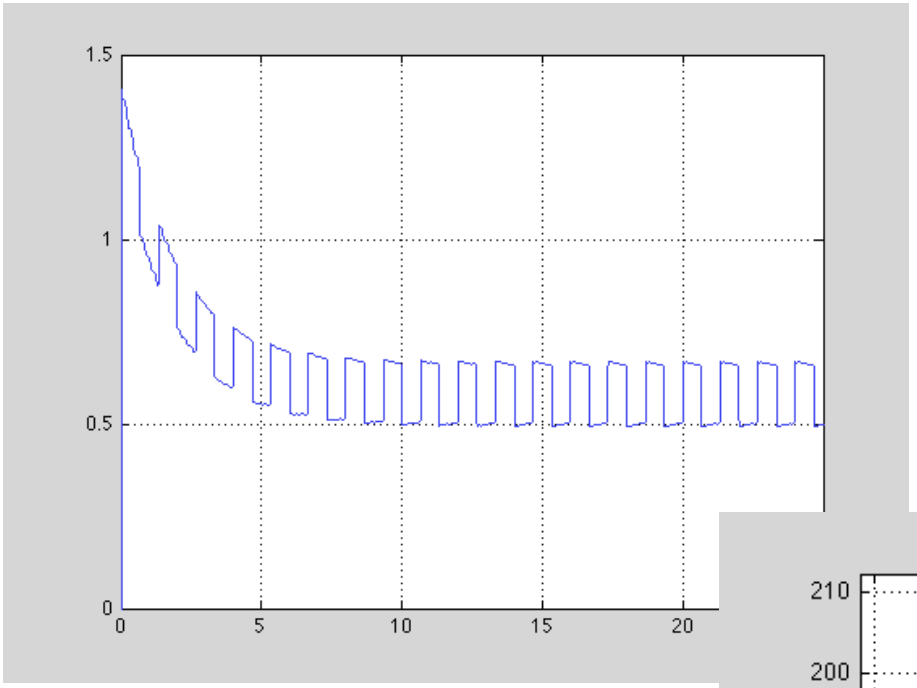
$$J\ddot{\vartheta}(t) = Ki(t) - \beta\dot{\vartheta}(t) - f_c(t)$$

$$J_L\ddot{\vartheta}_L(t) = f_c(t) + f_d(t)$$

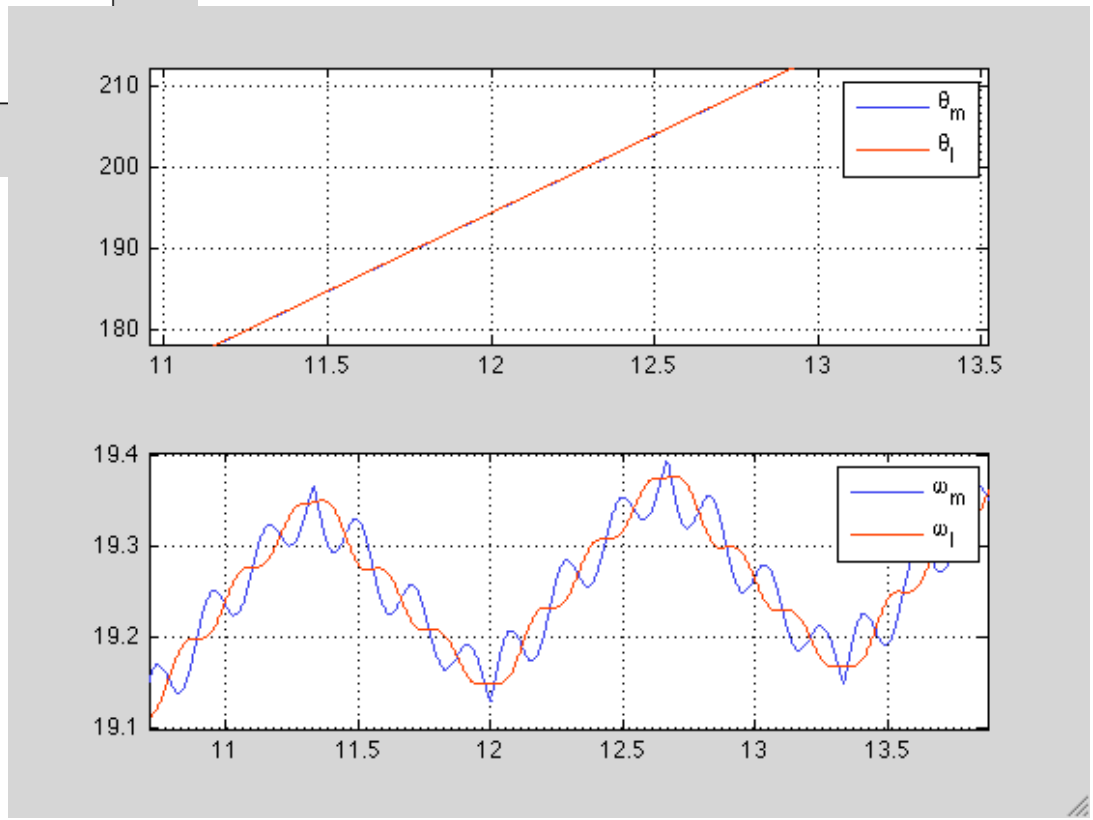
$$L\frac{di(t)}{dt} = -Ri(t) - K\dot{\vartheta}(t) + v(t)$$

$$J\ddot{\vartheta}(t) = Ki(t) - \beta\dot{\vartheta}(t) - k(\vartheta(t) - \vartheta_L(t))$$

$$J_L\ddot{\vartheta}_L(t) = k(\vartheta(t) - \vartheta_L(t)) + f_d(t)$$



Andamento temporale della corrente di armatura

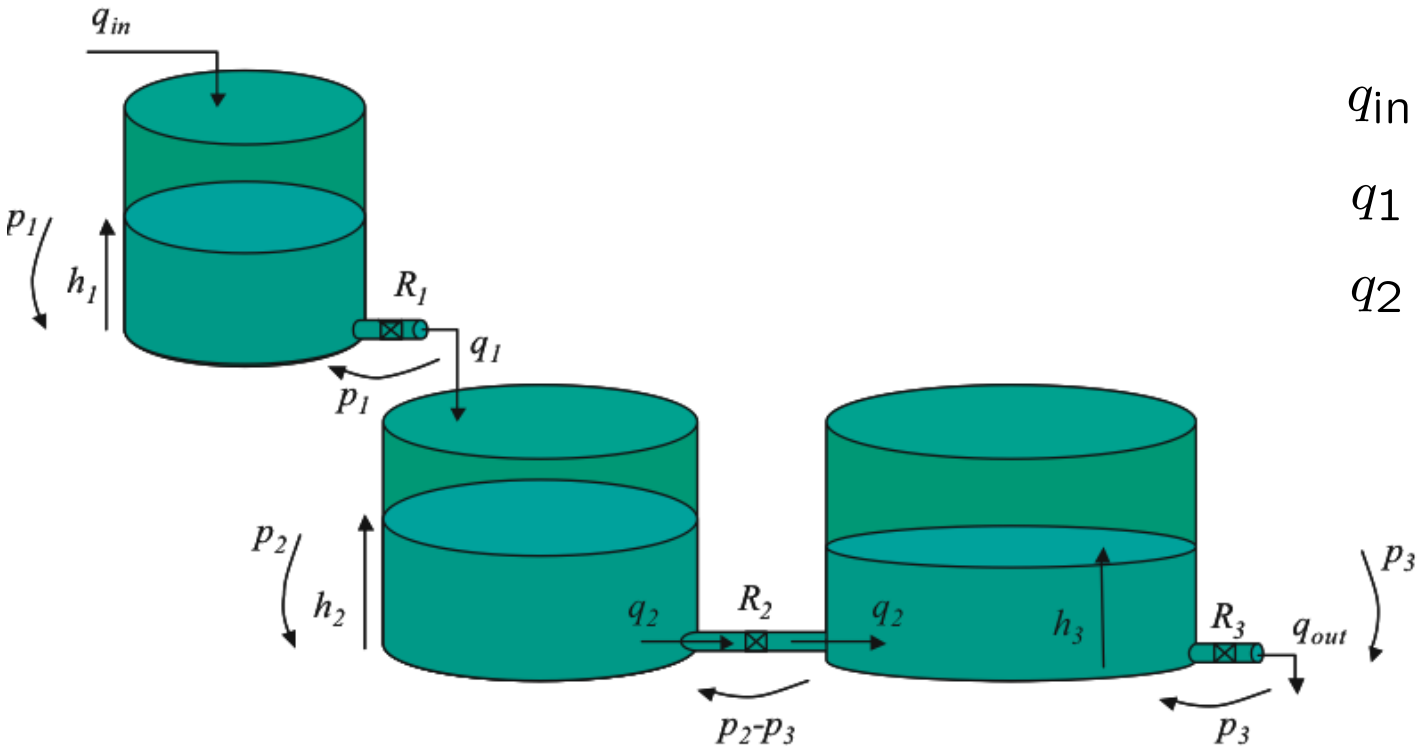


Andamento temporale degli angoli di rotazione e delle velocità angolari lato motore e del carico.

```
function dydt = TDSC_ode_esempio8(t, y)
% ODE dell'esempio 8 %
R = 6; % Ohm
L = 1.2e-2; % H
K = 2.34e-1; % V s /rad
b = 0.02; % Nm/rad
J = 1.67e-2; % kg m^2
kk = 10.45; % asse elastico
J_l = 3.98e-2; % inerzia del carico
u_step = 8; u_pulse = 0.5*sign(sin(0.75*2*pi*t));
u = u_step + u_pulse; % tensione pulsante in ingresso
f_d = 0.25; %  coppia sul carico
% ------------------------------------------------------
dydt = [(-R*y(1)-K*y(3)+u)/L; y(3);...
          (K*y(1)-b*y(3)-kk*(y(2)-y(4)))/J; y(5);...
          (kk*(y(2)-y(4))+f_d)/J_l];
```

```
% script TDSC_ode_esempio8
% configurazione del problema
ode_time = [0 25]; ode_y0 = [0;0;0;0;0];
ode_options = odeset('MaxStep', 1e-4);
[t,Y] = ode15s(@(t,y)TDSC_ode_esempio8(t, y), ode_time, ode_y0, ode_options);
figure;plot(t,Y(:,1));grid
figure;subplot(2,1,1); plot(t,Y(:,2),'b',t,Y(:,4),'r');grid;legend
('\theta_m','\theta_l');
subplot(2,1,2); plot(t,Y(:,3),'b',t,Y(:,5),'r');grid;legend
('\omega_m','\omega_l');
```

Esempio 9 (sistema idraulico tre serbatoi):



$$q_{\text{in}} - q_1 = C_1\dot{p}_1$$

$$q_1 - q_2 = C_2\dot{p}_2$$

$$q_2 - q_{\text{out}} = C_3\dot{p}_3$$

$$\sqrt{p_1} = R_1 q_1$$

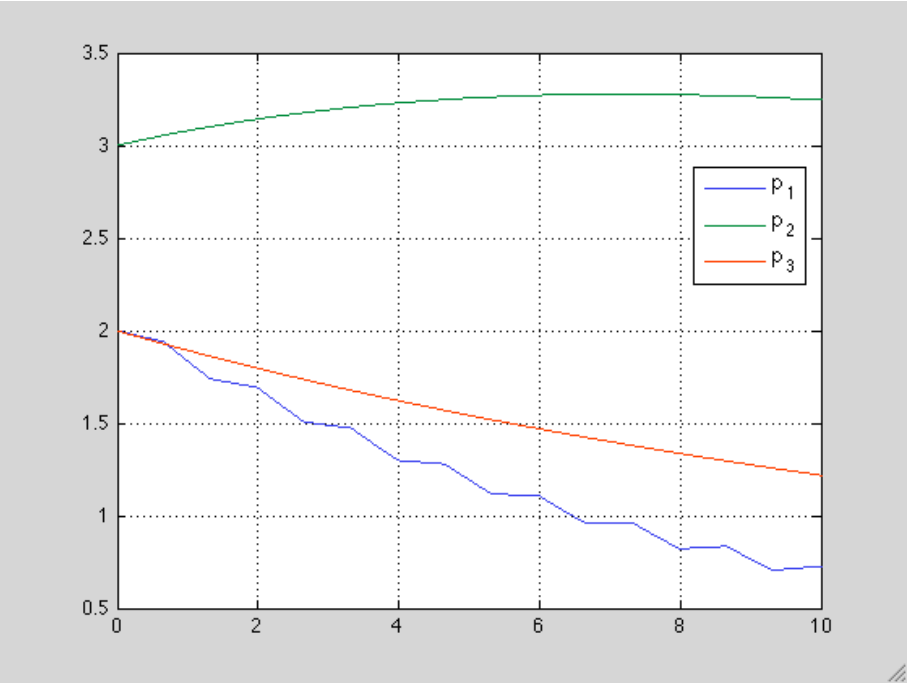
$$\frac{p_2 - p_3}{|p_2 - p_3|}\sqrt{|p_2 - p_3|} = R_2 q_2$$

$$\sqrt{p_3} = R_3 q_{\text{out}}$$

$$C_1\dot{p}_1 = q_{\text{in}} - \frac{\sqrt{p_1}}{R_1}$$

$$C_2\dot{p}_2 = \frac{\sqrt{p_1}}{R_1} - \frac{p_2 - p_3}{|p_2 - p_3|}\frac{\sqrt{|p_2 - p_3|}}{R_2}$$

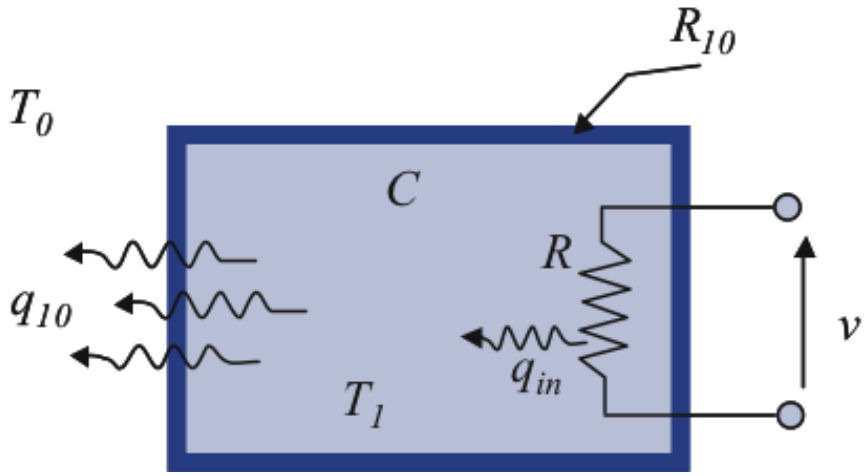
$$C_3\dot{p}_3 = \frac{p_2 - p_3}{|p_2 - p_3|}\frac{\sqrt{|p_2 - p_3|}}{R_2} - \frac{\sqrt{p_3}}{R_3}$$




```
% script TDSC_ode_esempio9
ode_time = [0 10];
ode_y0 = [2;3;2];
ode_options = odeset('MaxStep', 1e-3);
[t,Y] = ode45(@(t,y)TDSC_ode_esempio9
(t, y), ode_time, ode_y0, ode_options);
figure;plot(t,Y);grid on;
legend('p_1','p_2','p_3');
```

```
function dydt = TDSC_ode_esempio9(t, y)
% ODE dell'esempio 9 - sistema idraulico
C1 = 4.5; C2 = 3.45; C3 = 6.21; % sezioni dei serbatoi
R1 = 2.4; R2 = 3.4; R3 = 1.45; % perdite di carico
u_step = 0.5;
u_pulse = 0.5*sign(sin(0.75*2*pi*t));
u = u_step + u_pulse; % portata pulsante in ingresso
% y(1): p_1; y(2): p_2; y(3): p_3
 dydt = [(u-sqrt(y(1)))/C1;...
          (sqrt(y(1))/R1-(y(2)-y(3))*sqrt(abs(y(2)-y(3)))/abs(y
(2)-y(3))/R2)/C2;...
          ((y(2)-y(3))*sqrt(abs(y(2)-y(3)))/abs(y(2)-y(3))/R2-
sqrt(y(3))/R3)/C3];
```

Esempio 10 (boiler elettrico):

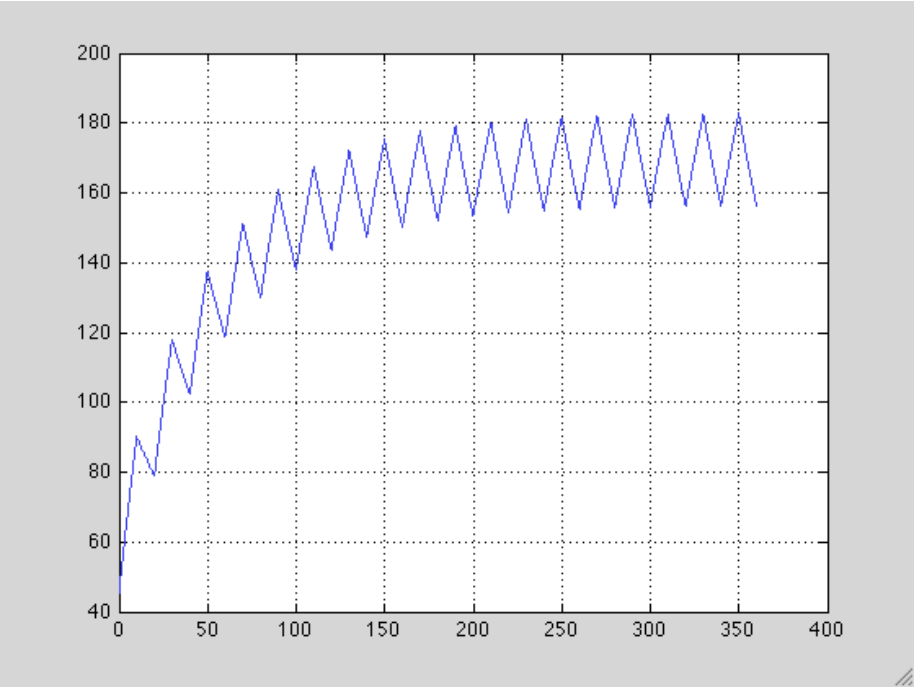


$$q_{\text{in}} = \frac{1}{R}v^2$$

$$q_{10} = \frac{1}{R_{10}}(T_1 - T_0)$$

$$q_{\text{net}} = q_{\text{in}} - q_{10} = \frac{1}{R}v^2 - \frac{1}{R_{10}}(T_1 - T_0)$$

$$\dot{T}_1(t) = \frac{1}{C}q_{\text{net}} = -\frac{T_1(t)}{R_{10}C} + \frac{T_0}{R_{10}C} + \frac{v^2(t)}{RC}$$




```
function dydt = TDSC_ode_esempio10(t,
y)
C = 4.5; % capacita' termica
R10 = 12.4; R = 3.4; % resistenze
termiche
u_step = 4.5;
u_pulse = 4.5*sign(sin(0.05*2*pi*t));
u = u_step + u_pulse; % portata
pulsante in ingresso
T0 = 22; % temperatura ambiente
%
-------------------------------------
-------------------
dydt = [ -y(1)/R10/C+T0/R10/C+u^2/R/
C];
```

```
% script TDSC_ode_esempio10
%
ode_time = [0 360];
ode_y0 = [45.6];
ode_options = odeset('MaxStep', 1e-2);

[t,Y] = ode23(@(t,y)TDSC_ode_esempio10(t, y), ode_time, ode_y0, ode_options);
figure;plot(t,Y);grid on;
```