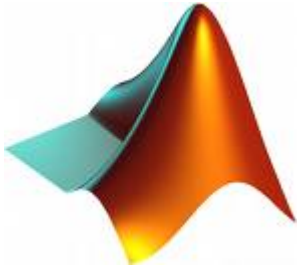


Matlab-Simulink per l'Ingegneria

Slides Lezione 2025
Parte seconda

Prof. Alessandro Pisano
apisano@unica.it

Approfondimenti sulle functions

Accesso di un function file alle variabili del workspace

L'unico modo attraverso il quale una funzione può accedere a variabili del workspace è che tali variabili siano definite come **globali** (`global`)

Le variabili devono essere definite come globali esternamente a tutte le funzioni.

```
global par1 par2 par3
```

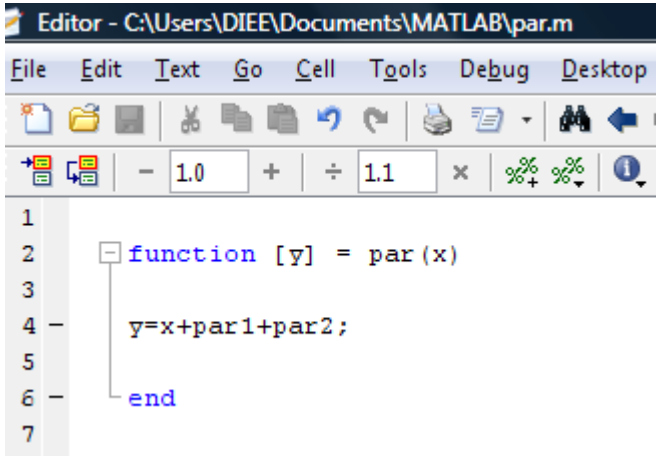
Affinché un function file possa accedere alle variabili `par1 par2 par3` presenti nel workspace allora tali variabili devono essere ridefinite come globali (con la medesima sintassi) **anche all'interno del corpo della funzione**

Esempio

Mediante uno script si caricano nel workspace i valori di determinati parametri

```
clear all
clc

par1=1;
par2=2;
```

Il seguente function file di test (che acquisisce un ingresso e gli somma i due parametri `par1` e `par2`) prova ad accedere ai parametri:




```
function [y] = par(x)

y=x+par1+par2;

end
```

Eseguendolo nel prompt di Matlab si ottiene un messaggio di errore

```
>> a=par(3)
??? Undefined function or variable 'par1'.

Error in ==> par at 4
y=x+par1+par2;
```

Le variabili par1 e par2 devono essere dichiarate globali, sia nello script di definizione dei parametri …

```
clear all
clc
global par1 par2
par1=1;
par2=2;
```

… che nel corpo del function file.

```
function [y] = par(x)
global par1 par2
y=x+par1+par2;
end
```

Ora il function file viene eseguito correttamente

```
>> a=par(3)

a =

     6
```

Anonymous functions (AF)

E' a tutti gli effetti un function file che però non viene salvato in un m-file ma risiede in una variabile del workspace. E' quindi locale alla sessione di lavoro corrente e semplifica progetti complessi riducendo il numero complessivo di files su disco.

Una AF è una espressione che può avere un numero arbitrario di variabili di ingresso ma un'unica uscita. Tale uscita può però essere costituita da un vettore, o da una matrice, quindi è possibile fare in modo che la AF restituisca un numero arbitrario di quantità di interesse.

Una AF sarà definita normalmente all'interno di uno script o di un function file.

La **sintassi** per creare una AF è la seguente

```
f = @(in1,in2,…,inn)espressione
```

Esempio Scrivere una AF che implementa la seguente funzione:

$$g(x) = x^2 + 3x + 4$$

La AF sarà definita dalla seguente istruzione:

```
g = @(x)x^2+3*x+4;
```

E' possibile impiegare la AF, ad esempio in uno script, come segue:

```
a1=g(4)
a2=g(10)
```

ottenendo il seguente output

```
a1 =

    32


a2 =

   134
```

Una AF può avere più di un parametro in ingresso

Esempio Scrivere una AF che implementa la funzione:

$$h(x,n) = x^n$$

```
h = @(x,n)x^n;
```

Esempio Scrivere una AF che implementa la funzione:

$$f(x,y) = x^2 + y^2 + 2xy + y + 5$$

```
f = @(x,y)x^2+y^2+2*x*y+y+5;
```

Una AF puà ricevere in ingresso vettori o matrici

Esempio Scrivere una AF che riceve in input due vettori e restituisce uno scalare che contiene la somma di tutti gli elementi del primo vettore più il primo elemento del secondo vettore:

```
g=@(v,w)(sum(v)+w(1));
```

Una AF puo restituire in uscita vettori o matrici

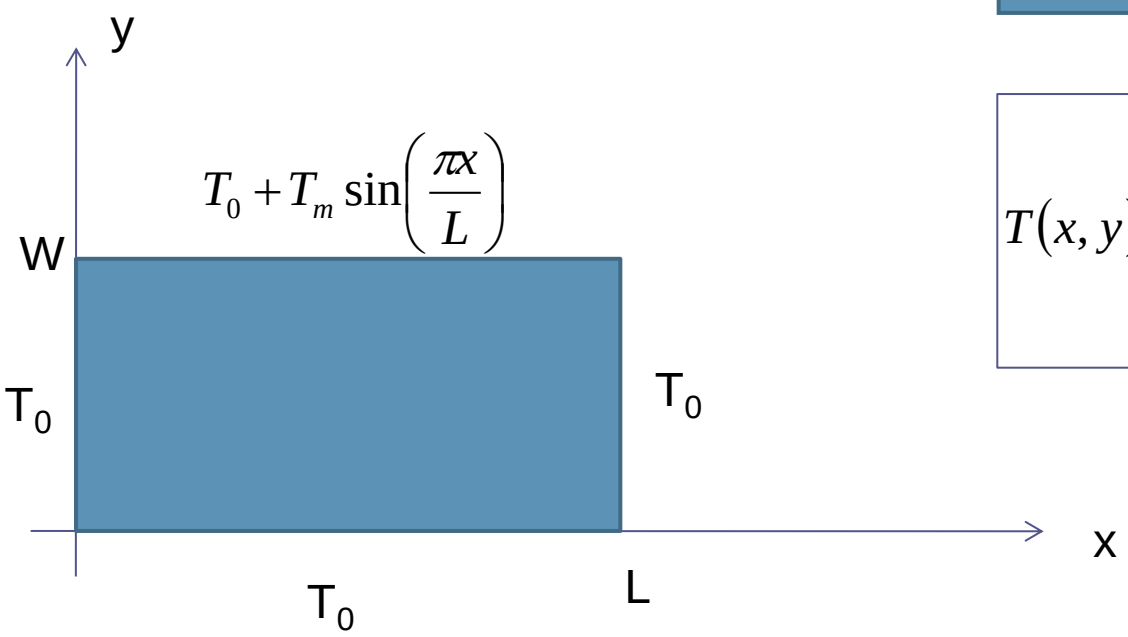
Esempio Scrivere una AF che riceve in input uno scalare x e restituisce in uscita la matrice: $\begin{bmatrix} 1 & x & x^2 \\ 0 & 1 & x \\ 0 & 0 & 1 \end{bmatrix}$

```
g=@(x)[1 x x^2;0 1 x;0 0 1];
```

Una AF può impiegare variabili presenti nel workspace

Esempio

Scrivere una AF che implementi la distribuzione di temperatura a regime in una sezione di una sbarra metallica a sezione rettangolare di lunghezza infinita con temperatura nel bordo imposta dall'esterno.



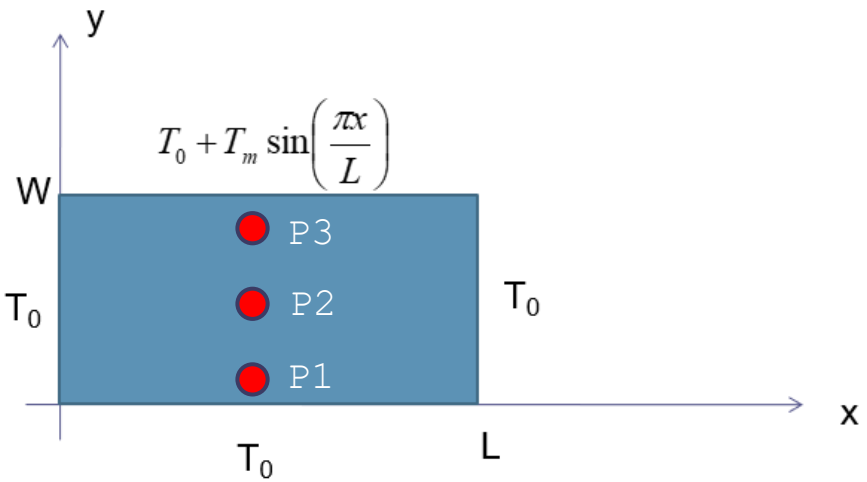
$$T(x,y) = T_0 + T_m \sin\left(\frac{\pi x}{L}\right) \frac{\sinh\left(\frac{\pi y}{L}\right)}{\sinh\left(\frac{\pi W}{L}\right)}$$

$$T(x,y) = T_0 + T_m \sin\left(\frac{\pi x}{L}\right) \frac{\sinh\left(\frac{\pi y}{L}\right)}{\sinh\left(\frac{\pi W}{L}\right)}$$

```
W=0.6;
L=0.6;
T0=25;
Tm=10;

T=@(x,y)(T0+Tm*sin(pi*x/L)*sinh(pi*y/L)/sinh(pi*W/L));
```

```
P1=T(0.3,0.01)
P2=T(0.3,0.3)
P3=T(0.3,0.59)
```

Usiamo la AF per valutare il valore della temperatura in tre punti P1, P2 e P3 della sezione



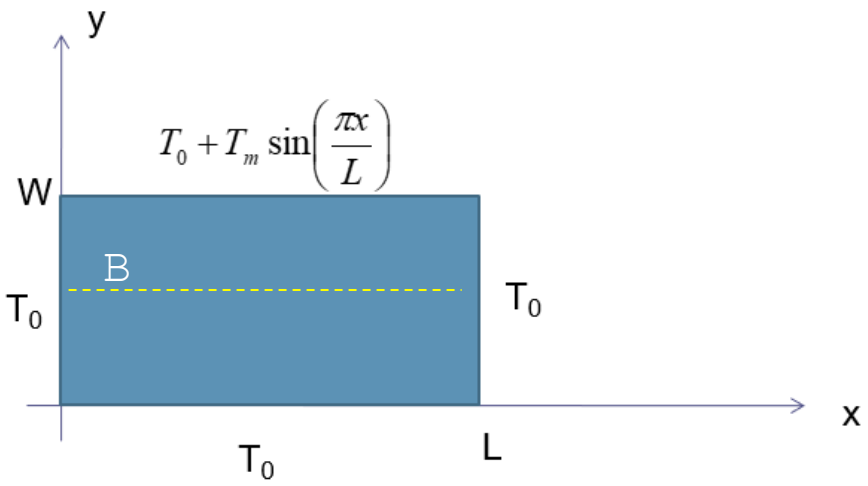

```
P1 =

   25.0454

P2 =

   26.9927

P3 =

   34.4879
```

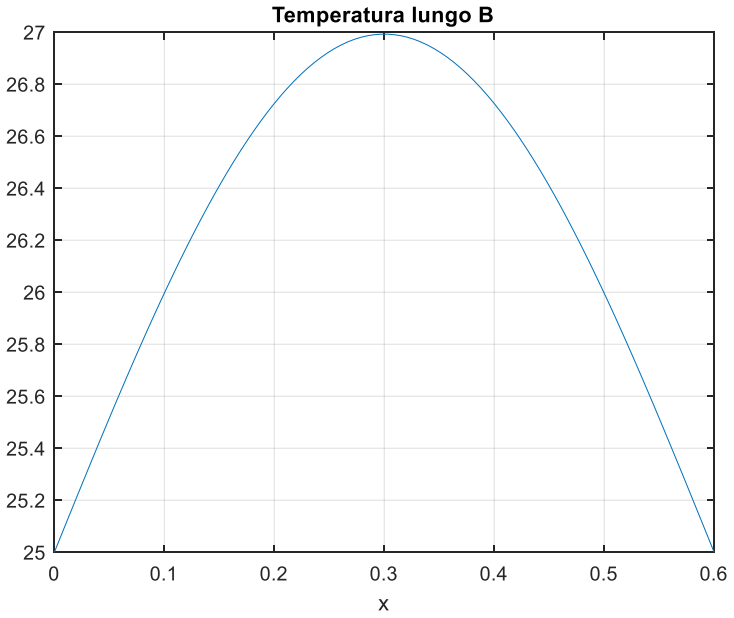
```
W=0.6;
L=0.6;
T0=25;
Tm=10;

T=@(x,y)(T0+Tm*sin(pi*x/L)*sinh(pi*y/L)/sinh(pi*W/L));

x=linspace(0,L,50);
P=T(x,W/2);
```

Utilizzo della AF con input vettoriale. Il vettore x contiene le coordinate orizzontali di 50 punti equispaziati lungo la linea B che divide la sezione a metà

```
plot(x,P),grid
title('Temperatura lungo B')
```

Grafico della temperatura lungo la linea B





Simulazione dinamica in ambiente Matlab

Impariamo a risolvere numericamente equazioni differenziali in ambiente Matlab

Si impiegano un insieme di funzioni dedicate, ciascuna corrispondente ad un particolare solutore numerico.

Un solutore in genere considerato come un ottimo compromesso fra complessità e precisione della soluzione calcolata è quello associato alla funzione Matlab **ode45**

Problema di Cauchy

$$\dot{y} = f(y,t) \qquad y(t) \in R \qquad t \in [t_0 \;\; t_f]$$

$$y(t_0) = y_0 \qquad y_0 \in R$$

Esempio

$$\dot{y} = -y^3 + ty \qquad t \in [0 \quad 5]$$

$$y(0) = 2$$

```
f=@(t,y)-y^3 + t*y;
```

Mediante una anonymous function si implementa il membro destro dell'equazione.
Il tempo t deve essere il primo argomento di ingresso della funzione, la variabile y il secondo.

```
tspan = [0 5];
y0 = 2;
```

Intervallo temporale di interesse e condizione inziale

```
[t,y] = ode45(f,tspan,y0);
```

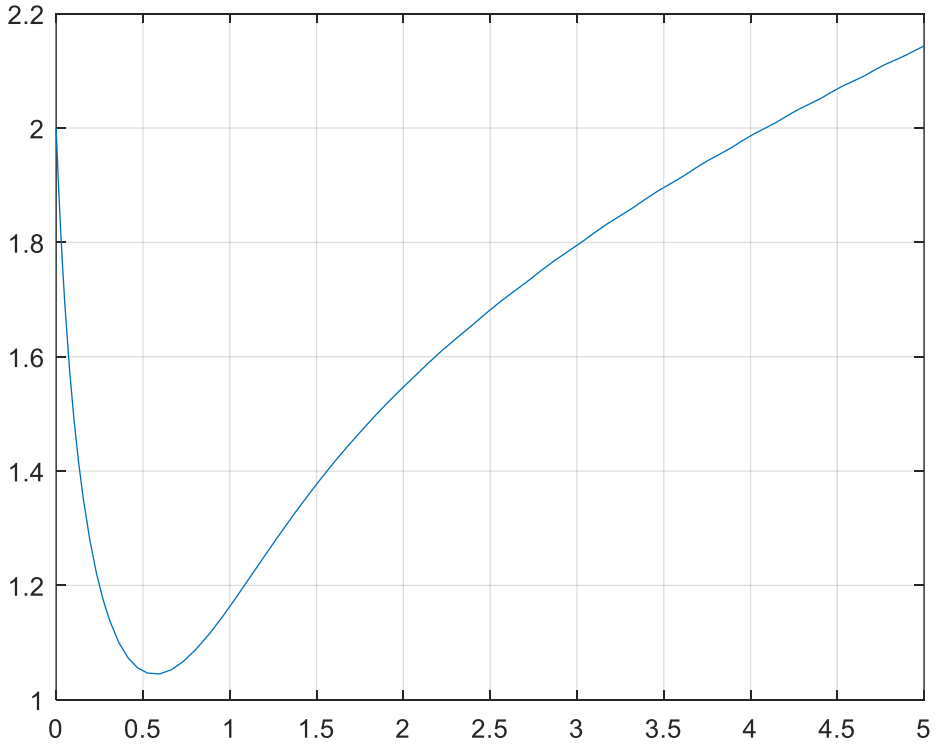
La funzione ode45 riceve in ingresso, nell'ordine, il nome della anonymous function, l'intervallo temporale di interesse per il calcolo della soluzione, e la condizione iniziale.
La funzione ode45 restituisce il vettore degli istanti in cui viene calcolate la soluzione, ed il vettore dei valori della soluzione in tali istanti.
Sono esattamente i due vettori da passare in ingresso alla funzione plot per la creazione del grafico.

```
plot(t,y),grid
```

$$\dot{y} = -y^3 + ty \qquad t \in [0 \quad 5]$$
$$y(0) = 2$$

```
f=@(t,y)-y^3 + t*y;
tspan = [0 5];
y0 = 2;
[t,y] = ode45(f, tspan, y0);
plot(t,y),grid
```

Grafico ottenuto



Il solutore sceglie autonomamente in quali istanti valutare la soluzione.

Nel presente esempio viene restituita una coppia di vettori (t,y) aventi dimensione 81.

La differenza fra gli istanti di campionamento adiacenti non è costante, e viene variata dal solutore con l'obbiettivo di massimizzare la velocita mantenendo una precisione sufficiente.

Vediamo come imporre la scelta degli istanti in cui valutare la soluzione.

```
f=@(t,y)-y^3 + t*y;
tspan = linspace(0,5,500)
y0 = 2;
[t,y] = ode45(f, tspan, y0);
plot(t,y),grid
```

La soluzione viene ora valutata in 500 istanti di tempo equispaziati, coincidenti con gli elementi del vettore tspan

$$\dot{y} = -y^3$$

$$y(0) = 2$$

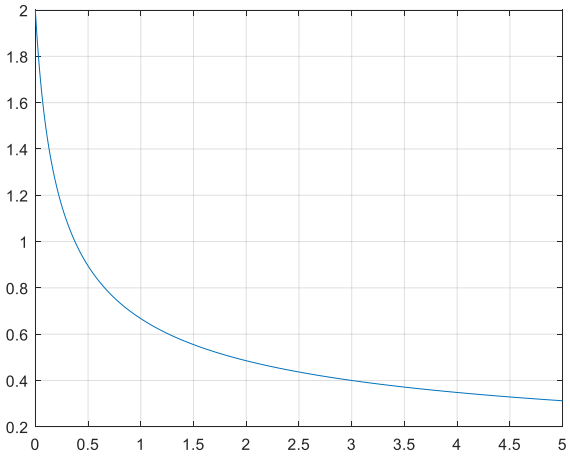
```
f=@(t,y)-y^3;
tspan = linspace(0,5,500)
y0 = 2;
[t,y] = ode45(f, tspan, y0);
plot(t,y),grid
```

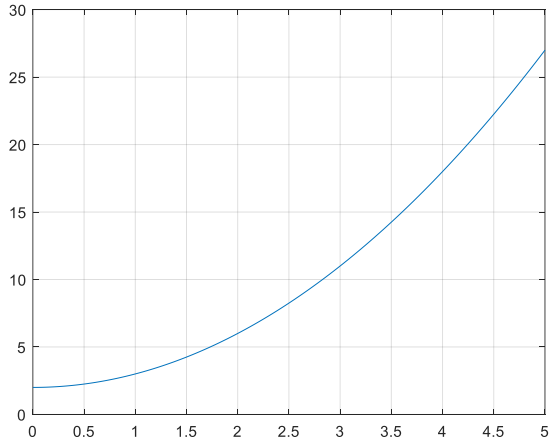
$$\dot{y} = 2t$$

$$y(0) = 2$$

```
f=@(t,y)2*t;
tspan = linspace(0,5,500)
y0 = 2;
[t,y] = ode45(f, tspan, y0);
plot(t,y),grid
```

N.B. Anche se in queste due equazioni il membro destro dipende solo da y o solo da t, **la anonymous function deve sempre essere definita attraverso una funzione con due argomenti di ingresso t ed y.** In caso contrario, si ottiene un messaggio di errore.





Sistemi di equazioni differenziali del primo ordine

$$\dot{y} + y^3 + z = 0$$

$$\dot{z} + \sin(z) \cdot y = 0$$

Forma «esplicita»

$$\dot{y} = -y^3 - z$$

$$\dot{z} = -\sin(z) \cdot y$$

$$y(0) = 1 \qquad z(0) = 2$$

$$y(0) = 1 \qquad t \in [0 \quad 20]$$

$$z(0) = 2$$

E' possibile, con semplici passaggi, riscrivere tale sistema di equazioni in **forma vettoriale**

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$$

$$\boldsymbol{x}(\boldsymbol{0}) = \boldsymbol{x_0}$$

in cui $\boldsymbol{x}$ è un vettore colonna di segnali

$$x_1 = y$$

$$x_2 = z$$

$$\dot{x}_1 = \dot{y} = -x_1^3 - x_2$$

$$\dot{x}_2 = \dot{z} = -\sin(x_2) \cdot x_1$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y \\ z \end{bmatrix}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -x_1^3 - x_2 \\ -\sin(x_2) \cdot x_1 \end{bmatrix} = \boldsymbol{f}(\boldsymbol{x})$$

$$\boldsymbol{x}(\boldsymbol{0}) = \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} y(0) \\ z(0) \end{bmatrix} = \boldsymbol{x_0} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} -x_1^3 - x_2 \\ -\sin(x_2) \cdot x_1 \end{bmatrix}$$

$$\boldsymbol{x}(\mathbf{0}) = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad t \in [0 \quad 20]$$

```
clear all
close all
clc
```

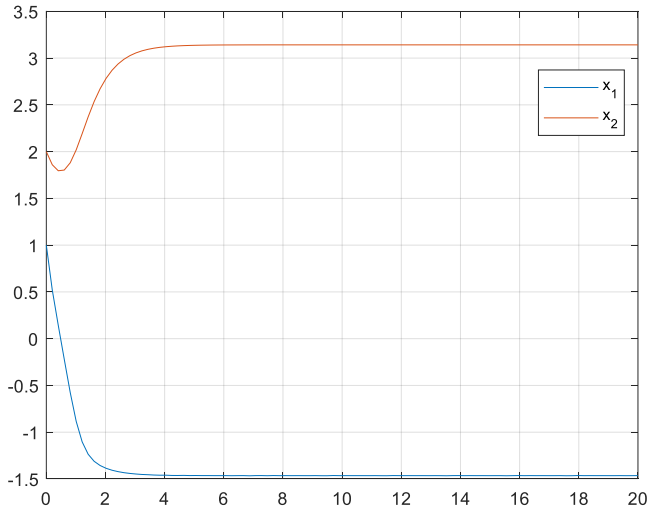
La anonymous function restituisce il vettore colonna f(x), le cui componenti sono delle funzioni di x(1) ed x(2) (in casi piu generali potrebbero anche dipendere dal tempo t)

```
f=@(t,x)[-x(1)^3-x(2); -sin(x(2))*x(1)]
```

```
tspan = linspace(0,20);
x0=[1;2]
```

Anche le condizioni iniziali sono un vettore colonna

```
[t,x] = ode45(f,tspan,x0);

plot(t,x),grid
legend('x_1','x_2')
```



Equazioni differenziali di ordine superiore al primo

$$\ddot{y} - \mu(1 - y^2)\dot{y} + y = 0 \qquad \mu > 0 \qquad t \in [0 \quad 20]$$

$$y(0) = 2$$

$$\dot{y}(0) = 0$$

La chiave per risolvere una equazione di grado superiore al primo è quella di ricondurla, dopo semplici manipolazioni algebriche, ad un sistema di eq. differenziali del primo ordine espresso in forma vettoriale. Fatto ciò, è possibile procedere come nel precedente esempio. Per semplicità di calcolo fissiamo $\mu = 1$.

Riscriviamo l'equazione differenziale in **forma esplicita**, cioè isolando alla sinistra dell'uguale la derivata di ordine più elevato della funzione incognita

$$\ddot{y} = (1 - y^2)\dot{y} - y$$

$$y(0) = 2 \qquad \dot{y}(0) = 0$$

$$\ddot{y} = (1 - y^2)\dot{y} - y$$

$$y(0) = 2 \qquad \dot{y}(0) = 0$$

Scegliamo come elementi x_1 e x_2 del vettore x la funzione incognita y e la sua derivata prima.

$$x_1 = y$$
$$x_2 = \dot{y}$$

$$\dot{x}_1 = \dot{y} = x_2$$
$$\dot{x}_2 = \ddot{y} = (1 - x_1^2)x_2 - x_1$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ (1 - x_1^2)x_2 - x_1 \end{bmatrix} = \boldsymbol{f}(\boldsymbol{x})$$

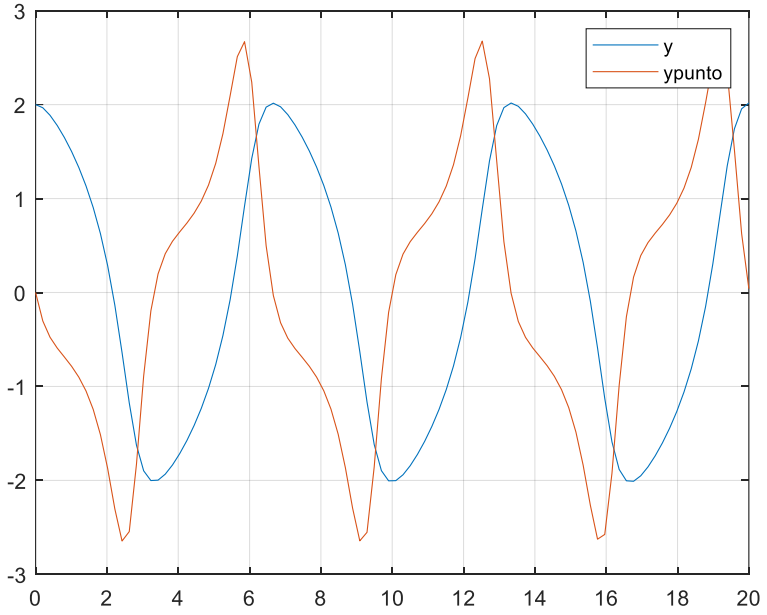
$$\boldsymbol{x}(\boldsymbol{0}) = \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} y(0) \\ \dot{y}(0) \end{bmatrix} = \boldsymbol{x_0} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

Ci siamo ricondotti ad una formulazione analoga a quella ricavata nel precedente esempio.

$$\dot{x} = f(x) = \begin{bmatrix} x_2 \\ (1 - x_1^2)x_2 - x_1 \end{bmatrix}$$

$$x(0) = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \qquad t \in [0 \quad 20]$$

```
clear all
close all
clc

f=@(t,x)[x(2); (1-x(1)^2)*x(2)-x(1)]

tspan = linspace(0,20);
x0=[2;0]

[t,x] = ode45(f,tspan,x0);

plot(tspan,x),grid
legend('y','ypunto')
```



$$\dot{x} = f(x) = \begin{bmatrix} x_2 \\ (1 - x_1^2)x_2 - x_1 \end{bmatrix}$$

$$x(0) = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

Il membro destro dell'equazione può anche essere implementato mediante un function file «tradizionale»

Script

```
clear all, close all, clc

tspan = linspace(0,20);
x0=[2;0]

[t,x] = ode45(@f,tspan,x0);

plot(t,x),grid
legend('y','ypunto')
```

N.B. Se si adotta tale scelta, nel passare il nome del function file alla funzione ode45 si deve anteporre @

Function

```
function xpunto = f(t,x)
xpunto = [x(2); (1-x(1)^2)*x(2)-x(1)];
end
```

Choose an ODE Solver

https://it.mathworks.com/help/matlab/math/choose-an-ode-solver.html

Basic Solver Selection

ode45 performs well with most ODE problems and should generally be your first choice of solver. However, ode23, ode78, ode89 and ode113 can be more efficient than ode45 for problems with looser or tighter accuracy requirements. Some ODE problems exhibit *stiffness*, or difficulty in evaluation. Stiffness is a term that defies a precise definition, but in general, stiffness occurs when there is a difference in scaling somewhere in the problem. For example, if an ODE has two solution components that vary on drastically different time scales, then the equation might be stiff. You can identify a problem as stiff if nonstiff solvers (such as ode45) are unable to solve the problem or are extremely slow. If you observe that a nonstiff solver is very slow, try using a stiff solver such as ode15s instead

Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. ode45 should be the first solver you try.
ode23		Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113		Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to evaluate.
ode78		High	ode78 can be more efficient than ode45 at problems with smooth solutions that have high accuracy requirements.
ode89		High	ode89 can be more efficient than ode78 on very smooth problems, when integrating over long time intervals, or when tolerances are especially tight.
ode15s	Stiff	Low to Medium	Try ode15s when ode45 fails or is inefficient and you suspect that the problem is stiff. Also use ode15s when solving differential algebraic equations (DAEs).
ode23s		Low	ode23s can be more efficient than ode15s at problems with crude error tolerances. It can solve some stiff problems for which ode15s is not effective. ode23s computes the Jacobian in each step, so it is beneficial to provide the Jacobian via odeset to maximize efficiency and accuracy. If there is a mass matrix, it must be constant.
ode23t		Low	Use ode23t if the problem is only moderately stiff and you need a solution without numerical damping. ode23t can solve differential algebraic equations (DAEs).
ode23tb		Low	Like ode23s, the ode23tb solver might be more efficient than ode15s at problems with crude error tolerances.
ode15i	Fully implicit	Low	Use ode15i for fully implicit problems $f(t,y,y') = 0$ and for differential algebraic equations (DAEs) of index 1.

Generazione di numeri casuali reali

rand(n) genera una matrice quadrata di dimensione ***n*** di elementi casuali distribuiti uniformemente nell'intervallo [0,1]

rand(m,n) genera una matrice ***m*** *x* ***n*** di elementi casuali distribuiti uniformemente nell'intervallo [0,1]

Come generare un vettore colonna di N numeri casuali uniformemente distribuiti su un arbitrario intervallo [a,b]?

```
N=8;
a=-5;
b=10;
v = a + (b-a).*rand(1,N)
```

```
v =

   -0.8596
    5.1955
    4.8265
   -2.5608
   -3.2150
    2.4755
    9.3962
    0.1058
    3.7790
   -1.6428
```

randn(n) genera una matrice quadrata di dimensione ***n*** di elementi casuali distribuiti secondo una gaussiana con media nulla e varianza unitaria

randn(m,n) genera una matrice ***m*** *x* ***n*** di elementi casuali distribuiti secondo una gaussiana con media nulla e varianza unitaria

```
M=randn(1,10)
```

```
M =

    1.1174   -1.0891    0.0326    0.5525    1.1006    1.5442    0.0859   -1.4916   -0.7423   -1.0616
```

normnd(mu,sigma,m,n) crea una matrice ***m*** *x* ***n*** di elementi casuali distribuiti secondo una gaussiana con media mu e deviazione standard sigma

Generazione di numeri casuali interi

randi(nmax) Genera uno scalare intero casuale uniformemente distribuito fra 1 ed nmax

randi([nmin nmax]) Genera uno scalare intero casuale uniformemente distribuito fra nmin ed nmax

randi(nmax,n,m) Genera una matrice n x m di interi casuali uniformemente distribuiti fra 1 ed nmax

Esempio: Simulare N lanci di un dado, e valutare la percentuale di volte che si ottiene 1, 2, …, 6

Ci attendiamo che per valori molto grandi di N le percentuali tendano tutte al valore costante 1/6

Possibile soluzione

```
clear all
clc
Numtest=25;
rng(0,'twister'); % rende ripetitivo il risultato

y=randi(6,Numtest,1)

vec1=(y==1);
vec2=(y==2);
vec3=(y==3);
vec4=(y==4);
vec5=(y==5);
vec6=(y==6);

num1=sum(vec1);
num2=sum(vec2);
num3=sum(vec3);
num4=sum(vec4);
num5=sum(vec5);
num6=sum(vec6);

Perc=[num1 num2 num3 num4 num5 num6]*100/Numtest
```

Vettore con elementi unitari in corrispondenza di ciascun lancio del dado che restituisce 1

Vettore con elementi unitari in corrispondenza di ciascun lancio del dado che restituisce 2

Vettore con elementi unitari in corrispondenza di ciascun lancio del dado che restituisce 6

Numero di lanci che restituiscono 1

Numero di lanci che restituiscono 2

Numero di lanci che restituiscono 6

Polinomi

Rappresentazione di polinomi per mezzo di **vettori**

Un polinomio di grado n è rappresentato da un vettore di dimensione n+1 che contiene tutti i coefficienti del polinomio (anche quelli eventualmente nulli) da quello del termine di grado piu elevato fino a giungere al termine noto (ultima componente del vettore)

$$p1(x) = 2x^4 + 7x^3 + x^2 - 4x + 2 \qquad vp1 = [2 \quad 7 \quad 1 \quad -4 \quad 2]$$

$$p2(x) = 3x^3 + 1 \qquad vp2 = [3 \quad 0 \quad 0 \quad 1]$$

Radici di un polinomio

Funzione "**roots**"

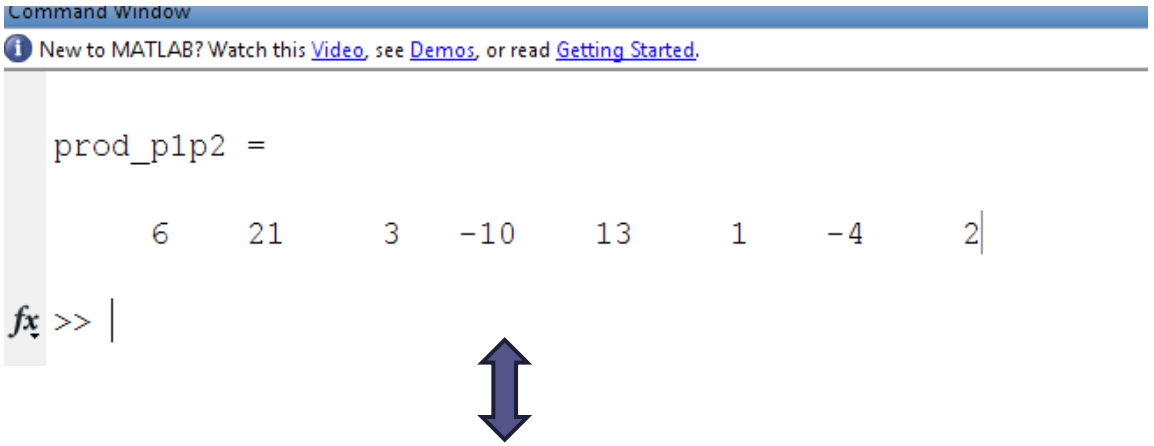
```
clear all
clc
p1=[2 7 1 -4 2];
p2=[3 0 0 1];
rad_p1=roots(p1)
```

```
rad_p1 =

  -3.0962
  -1.2285
   0.4124 + 0.3047i
   0.4124 - 0.3047i
```

Prodotto tra polinomi

Funzione "**conv**"

```
clear all
clc
p1=[2 7 1 -4 2];
p2=[3 0 0 1];
prod_p1p2=conv(p1,p2)
```

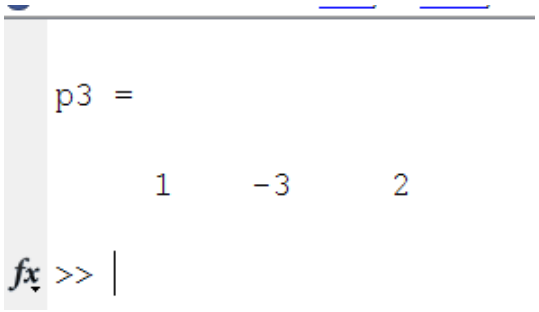


$$P(x) = 6x^7 + 21x^6 + 3x^5 - 10x^4 + 13x^3 + x^2 - 4x + 2$$

Polinomio di radici assegnate (Funzione "**Poly**")

$$p3 = poly([z1 \quad z2 \quad \ldots \quad zn]) \rightarrow p3(x) = (x - z1)(x - z2)\ldots(x - zn)$$

```
clear all
clc
p3=poly([1 2])
```

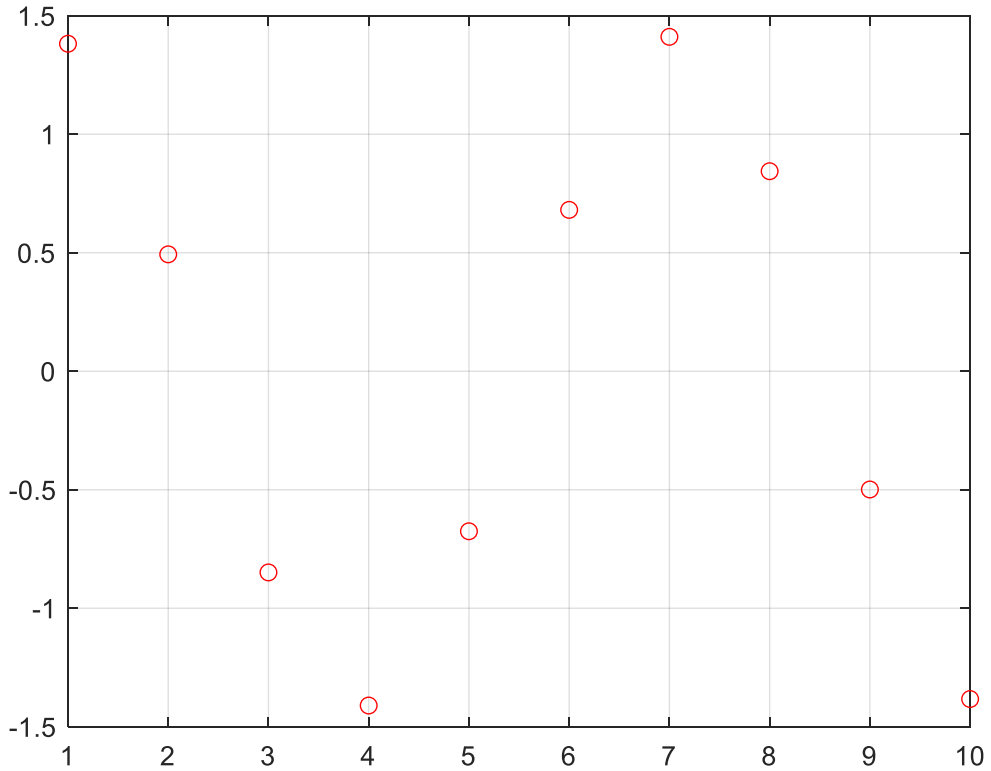


Polynomial fitting

Una esigenza frequente nell'ingegneria è quella di interpolare dei dati acquisiti sperimentalmente mediante una curva che «sposi» al meglio i dati. Vediamo come risolvere tale problema mediante curve approssimanti polinomiali.

Generiamo fittiziamente dei dati da interpolare, campionando una funzione armonica in 10 istanti differenti

```
t = linspace(1,10,10);
y = sin(t)+cos(t);


plot(t,y,'ro')
axis([0 10 -1.2 1.2])
grid
```



Si deve usare la funzione `polyfit` . I primi due parametri da passare in ingresso alla funzione sono i vettori che contengono gli istanti di acquisizione ed i dati acquisiti. Il terzo parametro è il grado del polinomio che si ricerca

```
p = polyfit(t,y,5)
```

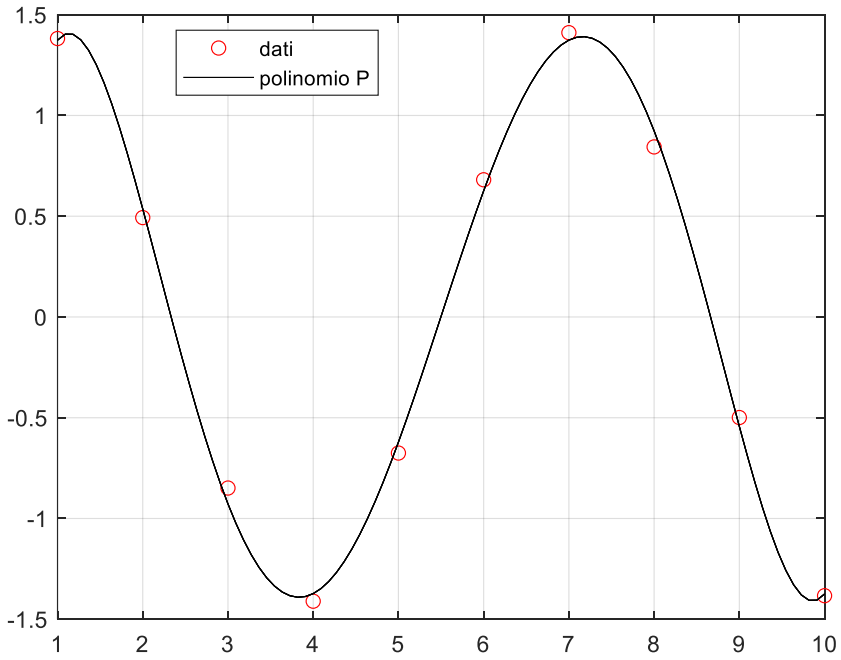
```
p =

    0.0049   -0.1345    1.3013   -5.1982    7.5145   -2.1144
```

Il polinomio interpolante restituito dalla funzione è:

$$P(t) = 0.0047t^5 - 0.1345t^4 + 1.3013t^3 - 5.1982t^2 + 7.5145t - 2.3134$$

Sovrapponiamo al grafico precedente la curva polinomiale calcolata per indagare il grado di sovrapposizione con i dati

```
t1 = linspace(1,10,100);
y1 = polyval(p,t1);
hold on
plot(t1,y1,'k')
hold off
```

Analisi spettrali

Vediamo come realizzare **analisi spettrali** di segnali campionati. Considereremo segnali generati virtualmente all'interno di uno script.

Le procedure descritte saranno ovviamente applicabili anche per effettuare analisi spettrali di segnali campionati acquisiti da file esterni.

Consideriamo il seguente segnale

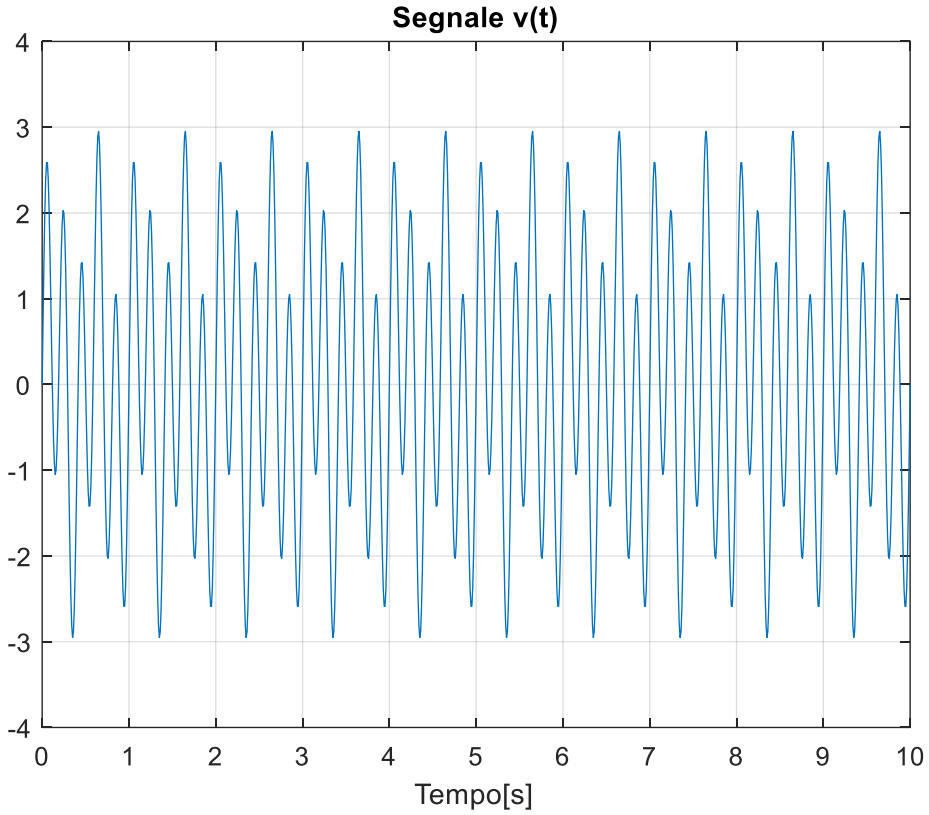
$$V(t) = \sin(4\pi t) + 2\sin(10\pi t)$$

Generiamo mediante il seguente codice un campionamento uniforme di tale segnale per 10 secondi. con passo di campionamento $T_c = 0.01s$

```
Tc=0.01;
t=0:Tc:10;
v=sin(4*pi*t)+2*sin(10*pi*t)
```

Grafichiamo il segnale

```
figure(1),
plot(t,v),grid,
title('Segnale v(t)'),
xlabel('Tempo[s]'),
ylim([-4 4])
```



Per realizzare analisi spettrali, inseriamo in fondo allo Script la seguente funzione, in modo che sia possibile utilizzarla all'interno dello script

```
function [freq data]=spettro(t,x)
% determinazione del periodo di campionamento
% NB il segnale deve essere campionato con periodo uniforme e
% l'istante iniziale deve essere t=0

Tc=t(2)-t(1);

% calcolo del vettore delle frequenze
f=0:1/t(length(t)):1/Tc;
f=f';

% calcolo della Fast Fourier Transform
Y=fft(x);

% calcolo della densità spettrale di potenza normalizzata
P=2*abs(Y)/length(Y);

freq=f(1:ceil(length(f)/2));
data=P(1:ceil(length(P)/2));
end
```

Function file:
spettro.m

La funzione `spettro` riceve in ingresso il vettore equispaziato degli istanti di campionamento ed il vettore che contiene i campioni del segnale (i due vettori che, passati come argomento di ingresso alla funzione `plot`, consentono di crearne il grafico).

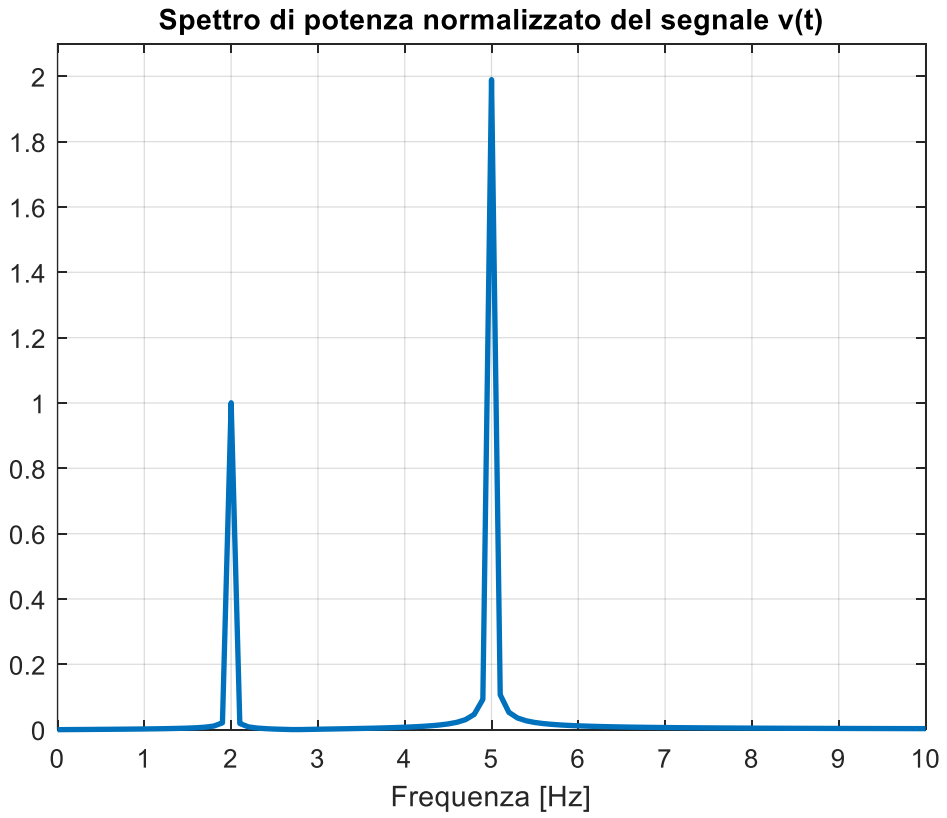
La funzione `spettro` restituisce il vettore delle frequenze (nell'intervallo $[0\,, 1/T_c\,]$, in cui T_c è il periodo di campionamento) ed i relativi valori dello spettro di potenza normalizzato.

Un volta eseguita la funzione con la seguente sintassi

```
[F,X]=spettro(vettore_tempi, vettore_campioni);
```

Si può realizzare il grafico dello spettro normalizzato passando in ingresso i vettori `F` ed `X` alla funzione plot

```
[F,X]=spettro(t,v);

figure(2),
plot(F,X,'LineWidth',2),grid
xlabel('Frequenza [Hz]')
title('Spettro di potenza normalizzato del segnale v(t)')
axis([0 10 0 2.1])
```

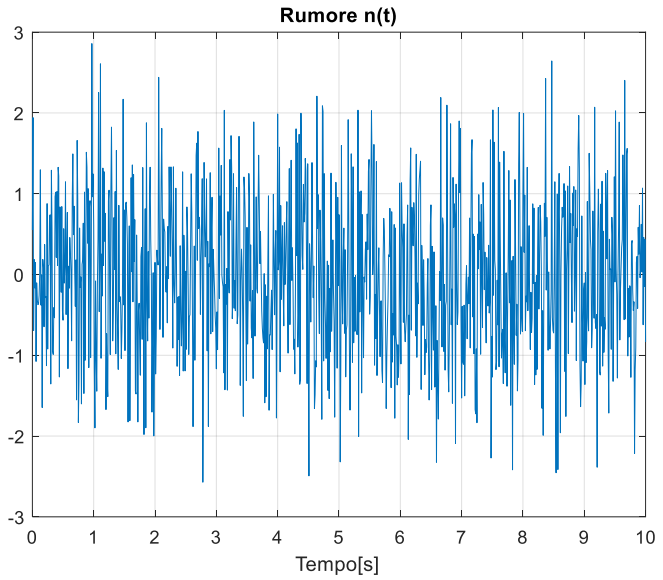


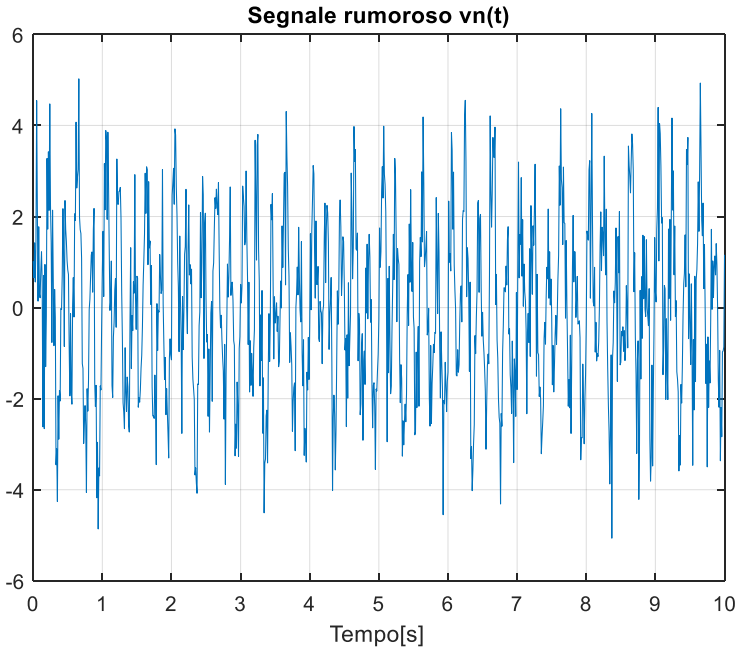
```
Tc=0.01;
t=0:Tc:10;
v=sin(4*pi*t)+2*sin(10*pi*t)

figure(1),
plot(t,v),grid,
title('Segnale v(t)'),
xlabel('Tempo[s]'),
ylim([-4 4])

[F,X]=spettro(t,v);
figure(3),
plot(F,X,'LineWidth',2),grid
xlabel('Frequenza [Hz]')
title('Spettro di potenza normalizzato del segnale v(t)')
axis([0 10 0 2.1])


function [freq data]=spettro(t,x)
% NB il segnale deve essere campionato con periodo uniforme e
% l.istante iniziale deve essere t=0.
% Determinazione del periodo di campionamento
Tc=t(2)-t(1);
% calcolo del vettore delle frequenze
f=0:1/t(length(t)):1/Tc;
f=f';
% calcolo della Fast Fourier Transform
Y=fft(x);
% calcolo della densità spettrale di potenza normalizzata
P=2*abs(Y)/length(Y);
freq=f(1:ceil(length(f)/2));
data=P(1:ceil(length(P)/2));
```

Script completo:
analisispettrale.m

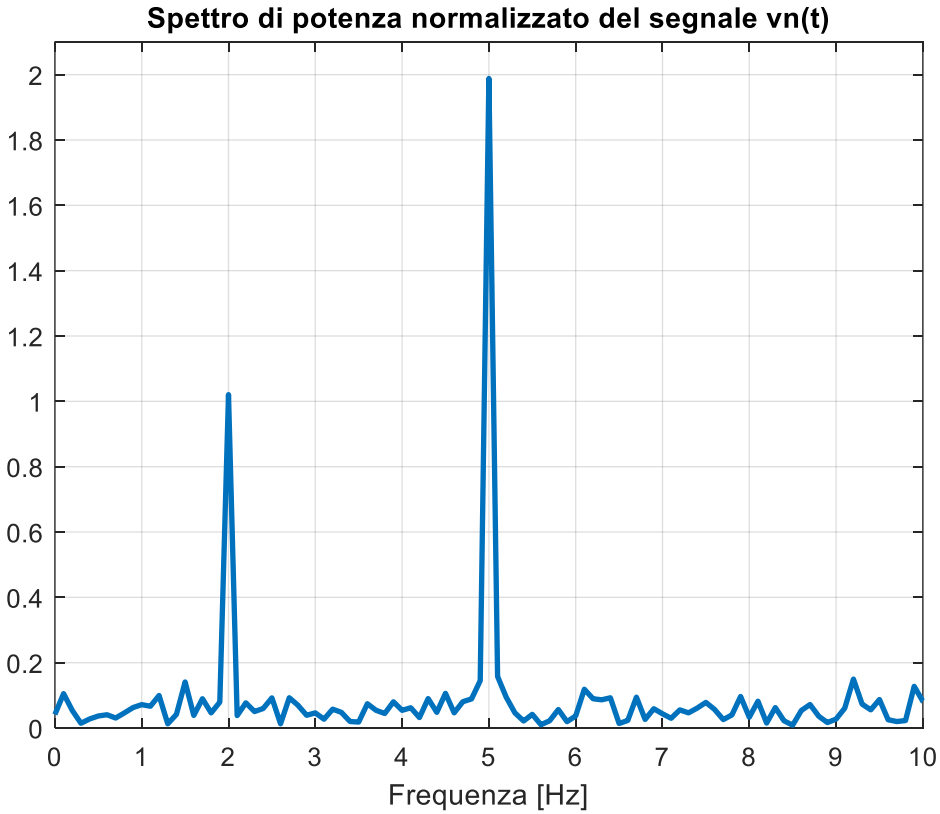
Ora sovrapponiamo al segnale un rumore gaussiano con media nulla e varianza unitaria, e verifichiamo gli effetti sullo spettro.

```
Tc=0.01;
t=0:Tc:10;
v=sin(4*pi*t)+2*sin(10*pi*t);

figure(1),
plot(t,v),grid,
title('Segnale v(t)'),
xlabel('Tempo[s]'),
ylim([-4 4])

Nsample=length(v);
n=randn(1,Nsample);

figure(2),
plot(t,n),grid,
title('Rumore n(t)'),
xlabel('Tempo[s]'),

vn=v+n;

figure(3),
plot(t,vn),grid,
title('Segnale rumoroso vn(t)'),
xlabel('Tempo[s]'),
```





```
[F,X]=spettro(t,vn);

figure(4),
plot(F,X,'LineWidth',2),grid
xlabel('Frequenza [Hz]')
title('Spettro di potenza
normalizzato del segnale vn(t)')
axis([0 10 0 2.1])
```



Script completo:
analisispettralenoise.m