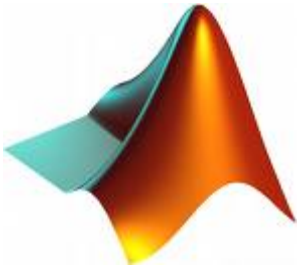


Simulazione dei Sistemi dinamici con Matlab-Simulink

Sistemi di equazioni lineari
Ottimizzazione

Ing. Alessandro Pisano
apisano@unica.it

Sistemi di equazioni lineari

$$\boldsymbol{A\,x = b}$$

$\boldsymbol{A}$ è una matrice $n \times m$

$\boldsymbol{x}$ è il vettore colonna delle incognite (di dimensione m)

$\boldsymbol{b}$ è un vettore di termini noti di dimensione m

$n = m$ **sistema «quadrato»** (tante equazioni quante sono le incognite)

$n > m$ **sistema sovradeterminato** (più equazioni che incognite)

$n < m$ **sistema sottodeterminato** (più incognite che equazioni)

La soluzione esiste se e solo se $rank\,(A) = rank\,([\,A \quad b\,])$

Tipicamente la soluzione non esiste se $n >$ m eccetto il caso fortuito in cui una o piu equazioni risultino essere combinazione lineare fra le altre e possano pertanto essere rimosse fino ad ottenere un sistema quadrato $n > m$

Esistono tuttavia situazioni di interesse pratico in cui anche se la soluzione non esiste si ricava dalla risoluzione approssimata della equazione qualche dato utile.

Sistema «quadrato» con matrice *A* non singolare

La soluzione esiste sempre ed è unica. La si calcola con il comando

x=A\b

Esempio

$$\begin{aligned} x_1 + x_2 + x_3 &= 3 \\ x_1 + 2x_2 + 3x_3 &= 1 \\ x_1 + 3x_2 + 6x_3 &= 4 \end{aligned} \qquad \boldsymbol{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}$$

```
A =[1 1 1;1 2 3;1 3 6];
b = [3;1;4];

x=A\b
```

```
x =

    10
   -12
     5
```

In linea di principio si potrebbe anche utilizzare la sintassi

```
x=inv(A)*b
```

che però risulta numericamente molto meno efficiente rispetto a `x=A\b`

Sistema «quadrato» con matrice *A* singolare

In questo scenario l'istruzione `A\b` (cosi come anche `inv(A)*b`) forniscono errore

Per trattare problemi di questo tipo si utilizza la sintassi

```
x=pinv(A)*b
```

Se la soluzione esiste la precedente sintassi restituisce fra le infinite soluzioni quella che ha la norma minima (restituisce cioè il vettore $\boldsymbol{x}$ che, fra gli infiniti vettori che soddisfano l'equazione, ha la minima norma euclidea).

Se la soluzione non esiste, la precedente sintassi restituisce la soluzione ai minimi quadrati, cioè il vettore $\boldsymbol{x}$ che minimizza la somma fra i quadrati delle componenti del vettore

$$\boldsymbol{A\,x - b}$$

Esempio

```
A = [ 1 3 7; -1 4 4; 1 10  18 ]
```

singolare.

Se si sceglie

```
b =[5;2;12]
```

il sistema ammette soluzioni (in conseguenza del fatto che $rank\ (A) = rank\ ([\ A \quad b\ ])$) e queste sono **infinite**. La soluzione avente norma euclidea minima è:

```
x=pinv(A)*b
```

```
x =

   -1.0892
    1.2512
   -0.5235
```

Si puo verificare che tale vettore soddisfa effettivamente il sistema di equazioni ($\boldsymbol{A\ x = b}$)

Esempio:

```
A = [ 1 3 7; -1 4 4; 1 10  18 ]
```

singolare.

Se si sceglie

```
b =[3;6;0]
```

il sistema **non ammette soluzioni**.

La soluzione ai minimi quadrati è

```
x=pinv(A)*b
```

```
x =

    0.3850
   -0.1103
    0.7066
```

Si può verificare che tale vettore non soddisfa il sistema di equazioni ($\boldsymbol{A\,x \neq b}$) ma in qualche modo è il vettore che più si avvicina a soddisfare il sistema in quanto minimizza la norma di $\boldsymbol{A\,x - b}$

```
A = [ 1 3 7; -1 4 4; 1 10  18 ]
b =[5;2;12]
verificarango= rank(A)==rank([A b])

if verificarango
disp('La soluzione esiste.')
disp('Soluzione a norma minima')
x=pinv(A)*b
else
 disp('Soluzione ai minimi quadrati')
 x=pinv(A)*b
end

b =[3;6;0]
verificarango= rank(A)==rank([A b])

if verificarango
disp('La soluzione esiste.')
disp('Soluzione a norma minima')
x=pinv(A)*b
else
 disp('Soluzione ai minimi quadrati')
 x=pinv(A)*b
end
```

Script: SistEqLinQuadrati.m

Sistema sottodeterminato

Ammette infinite soluzioni.

Una soluzione puo essere calcolata con la sintassi:

x=A\b Soluzione con alcune componenti nulle

La soluzione avente norma euclidea minima è calcolabile con la sintassi:

p = lsqminnorm(A,b)

Esempio

$$6x_1 + 8x_2 + 7x_3 + 3x_4 = 7$$
$$3x_1 + 5x_2 + 4x_3 + x_4 = 8$$

$$\boldsymbol{A} = \begin{bmatrix} 6 & 8 & 7 & 3 \\ 3 & 5 & 4 & 1 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 7 \\ 8 \end{bmatrix}$$

```
A = [6 8 7 3; 3 5 4 1]
b=[7;8]

disp('Soluzione con componenti nulle')
x=A\b
disp('Soluzione a norma minima')
x2=lsqminnorm(A,b)
```

```
x =

         0
    2.4286
         0
   -4.1429
```

```
x2 =

   -1.5109
    2.6642
    0.5766
   -3.0949
```

Script: SistEqLinSottodet.m

Sistema sovradeterminato

In generale la soluzione non esiste, tranne il caso fortuito in cui le equazioni in piu rispetto al numero di incognite siano combinazioni lineari fra le rimanenti equazioni.

Per trattare problemi sovradeterminati si utilizza la sintassi

```
x=A\b
```

Se la soluzione non esiste, la precedente sintassi restituisce la soluzione ai minimi quadrati, cioè il vettore $\boldsymbol{x}$ che minimizza la somma fra i quadrati delle componenti del vettore

$$\boldsymbol{A\,x - b}$$

Esempio

Un ambito ingegneristico in cui si incontrano di frequente problemi lineari sovradeterminati è il fitting di dati sperimentali.

Ipotizziamo che una certa grandezza fisica venga misurata in determinati intervalli temporali in modo da generare lo stream di dati rappresentato dalla seguente matrice M

```
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';
M=[t y]
```

```
M =

         0    0.8200
    0.3000    0.7200
    0.8000    0.6300
    1.1000    0.6000
    1.6000    0.5500
    2.3000    0.5000
```

La prima colonna della matrice contiene gli istanti di campionamento, mentre la seconda colonna le letture dal sensore.

E' noto a priori che la grandezza esibisce un decadimento esponenziale, e si desidera pertanto identificare i coefficienti C_1 e C_2 della seguente curva

$$y(t) = C_1 + C_2 e^{-t}$$

in modo da massimizzare l'aderenza della curva teorica alla curva sperimentale.

Imponendo che la curva teorica passi per i punti individuati dalle misure sperimentali si ottengono le seguenti relazioni

$$y(0) = C_1 + C_2 = 0.82$$
$$y(0.3) = C_1 + C_2 e^{-0.3} = 0.72$$
$$y(0.8) = C_1 + C_2 e^{-0.8} = 0.63$$
$$y(1.1) = C_1 + C_2 e^{-1.1} = 0.6$$
$$y(1.6) = C_1 + C_2 e^{-1.6} = 0.55$$
$$y(2.3) = C_1 + C_2 e^{-2.3} = 0.5$$

$$\boldsymbol{A} \begin{bmatrix} C_2 \\ C_1 \end{bmatrix} = \boldsymbol{b}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 1 & e^{-0.3} \\ 1 & e^{-0.8} \\ 1 & e^{-1.1} \\ 1 & e^{-1.6} \\ 1 & e^{-2.3} \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 0.82 \\ 0.72 \\ 0.63 \\ 0.6 \\ 0.55 \\ 0.5 \end{bmatrix}$$

```
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';
M=[t y]

A = [ones(size(t)) exp(-t)];
b=y;
c = A\b
```

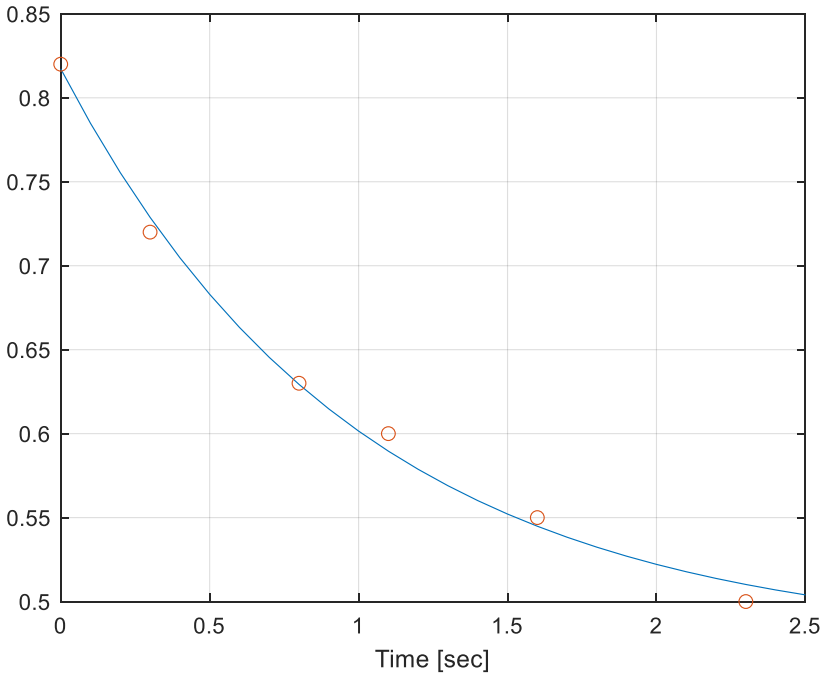
```
c =

    0.4760
    0.3413
```

Curva interpolante ai minimi quadrati

$$y(t) = 0.476 + 0.341e^{-t}$$

```
C1=c(1); C2=c(2);

fit=@(t)(C1+C2*exp(-t))
T = (0:0.1:2.5)';
Y=fit(T);
plot(T,Y,'-',t,y,'o'),grid
xlabel('Time [sec]')
```

Script: SistEqLinSovradet.m



Ottimizzazione multiparametrica

Un problema di ottimizzazione consiste nel determinare il **minimo** di una funzione obiettivo $f(\boldsymbol{x})$ che dipende da più parametri $(\boldsymbol{x} = (x_1, x_2, \dots, x_n))$

La funzione Matlab preposta è la funzione **`fminsearch`**

I parametri da passare in ingresso alla funzione sono la funzione da ottimizzare (descritta mediante una anonymous function oppure mediante un function file) ed una stima iniziale dei parametri ottimi. **Se la stima iniziale del punto di ottimo è molto distante dal punto di ottimo non è detto che l'algoritmo converga alla soluzione corretta.**

La funzione restituisce, in aggiunta ai valori ottimi dei parametri ed al valore ottimale della funzione obiettivo, anche un flag di uscita il cui valore nullo o negativo denota la possibile occorrenza di problemi di convergenza.

Esiste un'altra funzione Matlab, denominata **`fmincon`**, che tratta problemi di **ottimizzazione multiparametrica vincolata** in cui una o più delle variabili di ottimizzazione devono soddisfare particolari vincoli.

N. B. se il problema di ottimizzazione consiste nella massimizzazione di una funzione $f(\boldsymbol{x})$, è sufficiente minimizzare la funzione $\mathrm{g}(\boldsymbol{x}) = -f(\boldsymbol{x})$

Vediamo alcuni esempi

Esempio OPT1 **Minimizzare rispetto alle due variabili d ed α la funzione obiettivo J(d, α)**

$$J(d,\alpha) = \frac{100}{d} - \frac{d}{\tan(\alpha)} + \frac{2d}{sen(\alpha)}$$

```
clc, clear all

J=@(x)100/x(1)-x(1)/tan(x(2))+2*x(1)/sin(x(2));

stima_iniziale=[20 1];

[Xopt functionvalue ExitFlag]=fminsearch(J,stima_iniziale)
```

Stima iniziale dei parametri ottimi.

I valori ottimali dei parametri sono custoditi nel vettore `Xopt`

```
d_ottimo=Xopt(1)
theta_ottimo=Xopt(2)

J_ottimo=J([d_ottimo,theta_ottimo])
```

Script: Opt1.m

Output del codice

```
Xopt =

    7.5983    1.0472


functionvalue =

   26.3215


ExitFlag =

     1
```

valori ottimali dei parametri

valore minimo della funzione obiettivo

Flag di uscita- E' positivo $\rightarrow$ OK

```
d_ottimo =

    7.5983


theta_ottimo =

    1.0472


J_ottimo =

   26.3215
```

Se utilizziamo una diversa stima iniziale dei parametri ottimi l'algoritmo può non convergere

```
clc, clear all
J=@(x)100/x(1)-x(1)/tan(x(2))+2*x(1)/sin(x(2));
stima_iniziale=[20 0.1];

[Xopt functionvalue ExitFlag]=fminsearch(f,stima_iniziale)

d_ottimo=Xopt(1)
theta_ottimo=Xopt(2)
J_ottimo=f([d_ottimo,theta_ottimo])
```

```
Xopt =

 -798.6950    3.1416


functionvalue =

  -1.9566e+19


ExitFlag =

     0
```

La struttura della funzione obiettivo potrebbe non essere compatibile con una anonymous function. Vediamo come utilizzare la funzione `fminsearch` passandogli come argomento di ingresso il nome di un function file.

Esempio OPT1a

```
clc, clear all

stima_iniziale=[20 1];
[Xopt functionvalue
ExitFlag]=fminsearch(@funzione,stima_iniziale)

function out = funzione(x)
out=100/x(1)-x(1)/tan(x(2))+2*x(1)/sin(x(2));
end
```

Istruzioni di ottimizzazione

Definizione Function file.

NB Una funzione può anche essere definita localmente, all'interno (alla fine) di uno script che la utilizza, senza creare un function file dedicato. Si presti attenzione al fatto che anche se la function viene definita localmente, come in questo script di esempio, essa non può accedere alle variabili presenti nel workspace.

Script: Opt1a.m

Ottimizzazione multiparametrica vincolata

Il problema generale di ottimizzazione MP vincolata risolto dalla funzione **`fmincon`** è il seguente:

$$\min_x f(x) \text{ such that } \begin{cases} c(x) \le 0 \\ ceq(x) = 0 \\ A \cdot x \le b \\ Aeq \cdot x = beq \\ lb \le x \le ub, \end{cases}$$

vincolo non lineare di disuguaglianza

vincolo non lineare di eguaglianza

vincolo lineare di diseguaglianza

vincolo lineare di uguaglianza

limiti superiore e inferiore

Ad esempio, un vincolo non lineare di disuguaglianza per un problema di ottimizzazione che coinvolge due variabili x_1 e x_2 potrebbe essere

$$x_1^2 + x_2^2 \le 1$$

che significa imporre che la soluzione debba essere interna alla circonferenza di raggio unitario.

Esempio OPT2

$$f(x_1, x_2) = 1 + \frac{x_1}{1+x_2} - 3x_1x_2 + x_2(1 + x_1)$$

Ricerchiamone il punto di minimo sotto i vincoli:

$$0 \leq x_1 \leq 1 \qquad 0 \leq x_2 \leq 2$$

```
f = @(x)1+x(1)/(1+x(2)) - 3*x(1)*x(2) + x(2)*(1+x(1));
lb = [0,0];
ub = [1,2];
A = [];
b = [];
Aeq = [];
beq = [];
x0 = (lb + ub)/2;
nonlcon=[];
[x,fval,exitflag] = fmincon(f,x0,A,b,Aeq,beq,lb,ub,nonlcon)
```

Script: Opt2.m

```
x =

    1.0000    2.0000
```

```
fval =

   -0.6667
```

```
exitflag =

     1
```

Esempio OPT3

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Funzione di Rosenbrock. Ha un minimo globale che vale 0 nel punto (1,1).

Rappresenta una funzione complicata da minimizzare, e come tale è usata in letteratura come benchmark per porre a confronto fra di loro algoritmi differenti.

Curve di livello

«banana function»

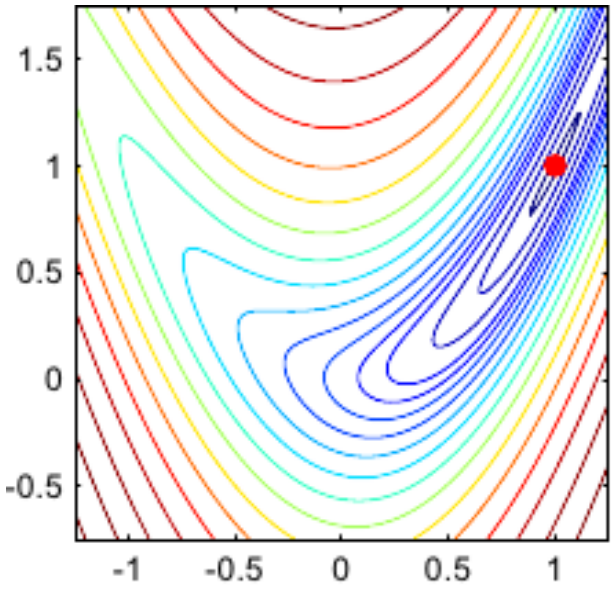
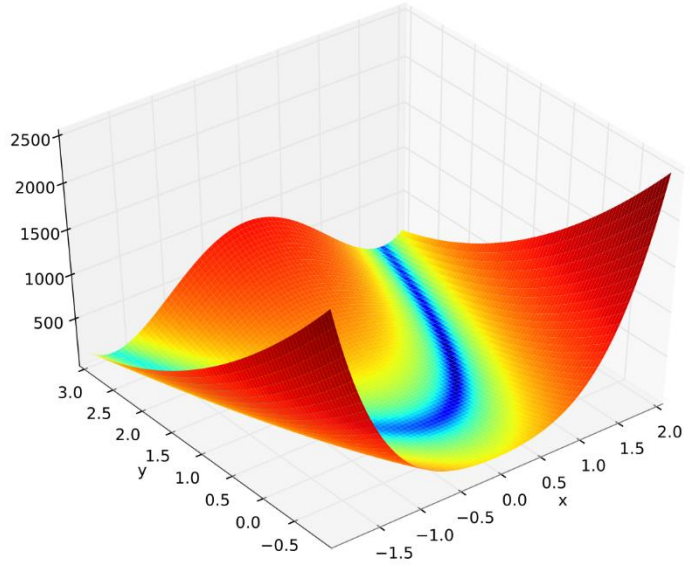


Grafico 3D



Esempio OPT3

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

MINIMIZZAZIONE NON VINCOLATA

Utilizziamo la funzione `fminsearch` con la sintassi illustrata nei precedenti esempi

```
Rosfun = @(x)100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
x0 = [-1.2,1];
[Xopt functionvalue ExitFlag]=fminsearch(Rosfun,x0)
```

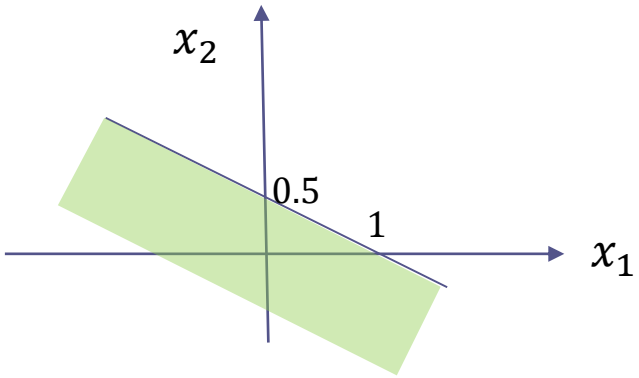
```
Xopt =

    1.0000    1.0000


functionvalue =

   8.1777e-10


ExitFlag =

     1
```

Script: Opt3.m

MINIMIZZAZIONE VINCOLATA - 1

Minimizziamo la Funzione di Rosenbrock sotto il vincolo

$x_1 + 2x_2 \leq 1$



$Ax \leq b$

$A = [1\ 2]$

$b = 1$

```
Rosfun = @(x)100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
x0 = [-1,2];
lb = [];
ub = [];
A = [1 2];
b = 1;
Aeq = [];
beq = [];
nonlcon=[];
[x,fval,exitflag] =
fmincon(Rosfun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
```

```
x =

    0.5022    0.2489


fval =

    0.2489


exitflag =

     1
```

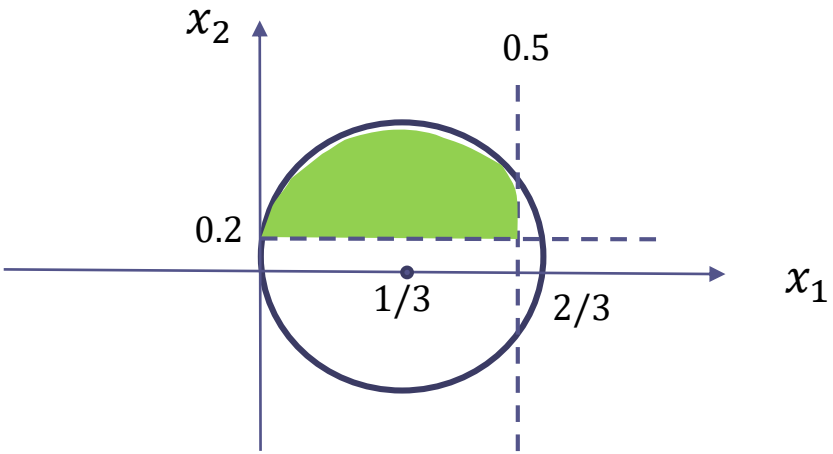
Script: Opt3.m

MINIMIZZAZIONE VINCOLATA - 2

Minimizziamo la Funzione di Rosenbrock sotto i vincoli:

$$0 \leq x_1 \leq 0.5 \qquad 0.2 \leq x_2 \leq 0.8$$

$$\left(x_1 - \frac{1}{3}\right)^2 + x_2^2 < \left(\frac{1}{3}\right)^2 \Rightarrow \left(x_1 - \frac{1}{3}\right)^2 + x_2^2 - \left(\frac{1}{3}\right)^2 < 0 \Rightarrow c(x) = \left(x_1 - \frac{1}{3}\right)^2 x_2^2 - \left(\frac{1}{3}\right)^2$$



I vincoli restringono la ricerca del punto di minimo ad una regione del piano, evidenziata in verde.

Function file che implementa il vincolo non lineare $c(x) \leq 0$

```
function [c,ceq] = vincolocirc(x)
c = (x(1)-1/3)^2 + (x(2)-1/3)^2 - (1/3)^2;
ceq = [];
end
```

Script: Opt3.m

$0 \leq x_1 \leq 0.5$ $\quad 0.2 \leq x_2 \leq 0.8$ $\quad \left(x_1 - \frac{1}{3}\right)^2 + x_2^2 < \left(\frac{1}{3}\right)^2$

Script di ottimizzazione

```
Rosfun = @(x)100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
x0 = [1/4,1/4]
lb = [0,0.2];
ub = [0.5,0.8];
A = [];
b = [];
Aeq = [];
beq = [];
nonlcon=@vincolocirc;
[x,fval,exitflag] = fmincon(Rosfun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
```

```
x =

    0.5000    0.2500
```

Script: Opt3.m

Nonlinear fitting

Riprendiamo l'esempio precedente di fitting di dati sperimentali

Ipotizziamo che una certa grandezza fisica venga misurata in determinati intervalli temporali in modo da generare lo stream di dati rappresentato dalla seguente matrice M

```
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';
M=[t y]
```

```
M =

         0    0.8200
    0.3000    0.7200
    0.8000    0.6300
    1.1000    0.6000
    1.6000    0.5500
    2.3000    0.5000
```

La prima colonna della matrice contiene gli istanti di campionamento, mentre la seconda colonna le letture dal sensore.

Script: OptFitting.m

Ora scegliamo un profilo in cui anche l'esponente della funzione esponenziale sia un parametro da stimare.

$$\hat{y}(C_1, C_2, \alpha, t) = C_1 + C_2 e^{\alpha t}$$

Nella procedura di fitting lineare si era posto a priori $\alpha = -1$

Cerchiamo i valori dei parametri C_1, C_2, α che minimizzano l'indice:

$$J(C_1, C_2, \alpha) = \sum_{i=1}^{N} [y_i - \hat{y}(C_1, C_2, \alpha, t_i)]^2$$

in cui y_i sono le misure sperimentali acquisite agli istanti t_i, per $i = 1,2,..,N$

Funzione per il calcolo dell'indice J

```
function out = Jfit(x)
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';

yhat=@(x,t)(x(1)+x(2)*exp(x(3)*t));

e=y-yhat(x,t);
out=sum(e.*e);
end
```

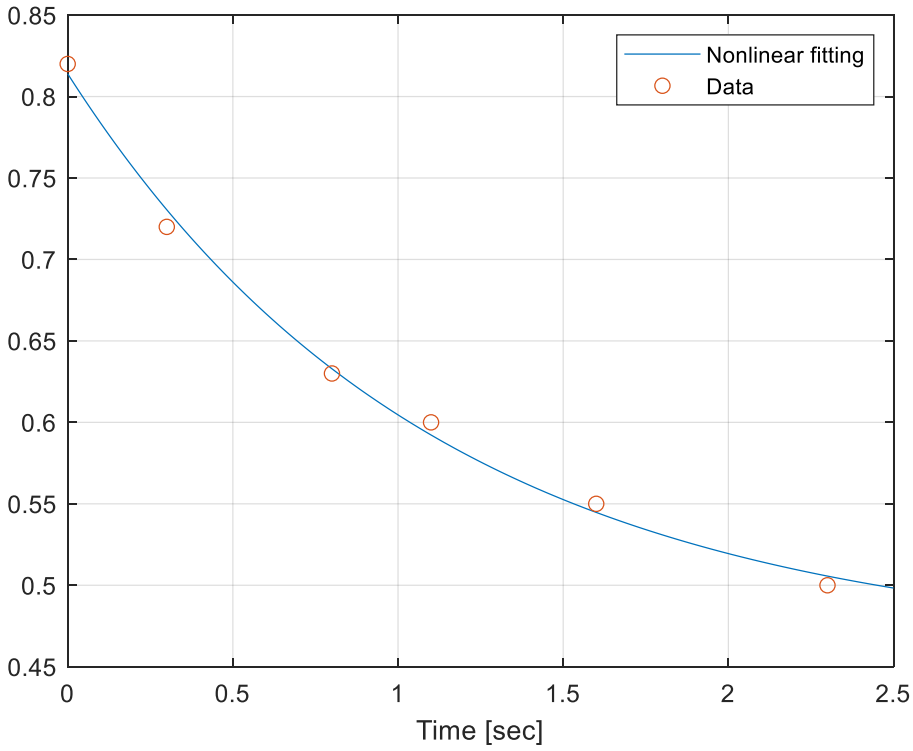
```
stima_iniziale=[0.2 0.1 -0.5];

[Xopt functionvalue ExitFlag]=fminsearch(@Jfit,stima_iniziale)

function out = Jfit(x)
t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';

yhat=@(x,t)(x(1)+x(2)*exp(x(3)*t));

e=y-yhat(x,t);
out=sum(e.*e);
end
```

```
Xopt =

    0.4611    0.3529   -0.8998


functionvalue =

   2.7450e-04


ExitFlag =

     1
```

Script: OptFitting.m

Script: OptFittingGraph.m

```
%VERIFICA GRAFICA DEI RISULTATI

t = [0 .3 .8 1.1 1.6 2.3]';
y = [.82 .72 .63 .60 .55 .50]';
yhat=@(x,t)(x(1)+x(2)*exp(x(3)*t));

T=0:0.01:2.5;
Y=yhat(Xopt,T);

figure(1)
plot(T,Y,'-',t,y,'o'),grid
xlabel('Time [sec]')
legend('Nonlinear fitting','Data')
```



Nella slide successiva confrontiamo i risultati con quanto ottenuto in precedenza impiegando la procedura di fitting lineare

```
Xopt2=[0.476 0.3413 -1];
Y2=yhat(Xopt2,T);

figure(2)
plot(T,Y,'-',T,Y2,'r-',t,y,'o'),grid
xlabel('Time [sec]')
legend('Nonlinear fitting','Linear fitting','Data')

J_NONLFIT=Jfit(Xopt)
J_LINFIT=Jfit(Xopt2)
```

Script: OptFittingGraph.m

```
J_NONLFIT =

   2.7450e-04


J_LINFIT =

   3.2420e-04
```