Equazioni differenziali ordinarie

Matlab possiede diverse funzioni per la risoluzione di equazioni differenziali. Cominciamo a vedere come procedere nel caso di un problema di Cauchy per una singola equazione scalare.

Il problema di Cauchy

Il problema

(*) y'(t)=f(t,y(t)) , y(t0)=y0

dove f(t,y) è una funzione data delle variabili scalari t e y, può essere risolto facilmente mediante diverse funzioni Matlab (*ode45, ode23, ode113*, ecc.) che si distinguono per l'algoritmo implementato, ma hanno sostanzialmente lo stesso utilizzo. Noi useremo *ode45*, che adopera un'opportuna versione del metodo di Runge-Kutta per problemi non stiff, con l'adattamento del passo di integrazione. L'uso di base è il seguente:

>> [t,y]=ode45(FUN,tspan,yzero);

dove in input si assegnano:

- FUN, ovvero una funzione f(t,y) introdotta a parte mediante una function *fun.m*
(come già visto altre volte, sono equivalenti le scritture @fun, @(t,y)fun(t,y), 'fun')
ma la funzione può anche essere introdotta direttamente mediante @(t,y)*espressione*.
Es.: @(t,y)y.*t.^2 . Il primo modo (@fun) è comunque il più efficiente.

- *tspan* che indica l'intervallo temporale di integrazione, e può avere una delle due forme:

>> tspan=[t0 t1]; % vengono forniti solo l'istante iniziale e quello finale
>> tspan=[t0 t1 t2 ... tn]; % array dei tempi dove calcolare la soluzione approssimata

- *yzero* che indica la condizione iniziale (cioè il valore di y(t0)).

L'output è costituito dagli array t e y degli istanti utilizzati dall'algoritmo e dei corrispondenti valori (utilizzabili ad esempio per tracciarne il grafico).

Esempio. Si voglia risolvere il problema di Cauchy:

$$y' = -y-5e^{(-t)}\sin(5t), \quad y(0)=1$$

nell'intervallo [0,3]. Poiché conosciamo la soluzione esatta [y(t)=e^(-t)cos(5t)], possiamo calcolare anche l'errore della differenza in norma infinito e visualizzare soluzione esatta e approssimata in uno stesso grafico. Ecco come potrebbe essere il programma:

```
% soluzione di un'equazione differenziale
close
clear
tspan=[0 3];%array che che contiene estremi intervallo di integrazione
%tspan=linspace(0,3,20);
yzero=1;%condizione iniziale
```

```
[t,y]=ode45(@myfun,tspan,yzero);
%[t,y]=ode23(@myfun,tspan,yzero);
%[t,y]=ode113(@myfun,tspan,yzero);

esatta=exp(-t).*cos(5*t);

%grafico soluzioni approssimata e esatta
plot(t,y,'*',t,esatta)
xlabel('t')
ylabel('y(t)')

% calcolo dell'errore massimo
 Emax=max(abs(y-exp(-t).*cos(5*t)))
```

Osservare la distribuzione non uniforme dei punti t_i (il programma è in grado di variare il passo in base a una data tolleranza). E osservare la differenza se si usa invece per tspan il comando commentato con linspace.

Ricordiamo che anche senza risolvere un'equazione differenziale come la (*) possiamo ottenere delle informazioni sulle soluzioni attraverso il campo di velocità definito in ogni punto del piano (t,y) dal vettore di componenti (1,f(t,y)). Questo campo di velocità può facilmente essere rappresentato mediante la funzione *quiver*. Vediamone il funzionamento:

```
n=20;
tpt=linspace(0,3,n);
ypt=linspace(-1,1,n);
% espando coordinate x,y
[t,y]=meshgrid(tpt,ypt);%genera matrici (nxn) t e y dai vettori tpt e ypt

pt=ones(size(y));%pt matrice quadrata nxn
py=myfun(t,y);   %py matrice quadrata nxn
quiver(t,y,pt,py,1.5) %campo vettoriale mediante array di pendenza (1,f(t,y))
axis([0 3 -1 1])
title('campo vettoriale')
```

In altre parole mediante il già visto comando *meshgrid* si crea una griglia di punti nel piano (t,y) sui quali si calcolano le direzioni indotte dalla funzione f. La funzione *quiver* rappresenta queste direzioni mediante frecce. Il parametro 1.5 serve in questo caso a scalare opportunamente tali frecce (il valore di default è 1; se si usa 0 si ottengono le lunghezze reali).
Come ulteriore parametro si può precisare lo stile grafico usando i simboli consueti (per il colore, la linea e i marker, solo che in quest'ultimo caso non vedremo frecce, ma segmenti uscenti dal marker). Esempi:

>> quiver(x,y,u,v,'--g') % frecce tratteggiate in verde

>> quiver(x,y,u,v,'or') % segmenti rossi che originano da pallini

Sistemi di EDO

Se invece di una singola equazione scalare dobbiamo risolvere un sistema di equazioni differenziali del primo ordine, bastano ovvie modifiche al comando ode45.
Come esempio risolviamo il classico problema dell'oscillatore armonico.

Esempio (**Oscillatore armonico**)
Il problema del secondo ordine: u'' = -w u può essere facilmente riscritto come un sistema di due equazioni del primo ordine mediante la trasformazione y_1=u, y_2=u', cioè:

$$y\_1' = y\_2 \,,\; y\_2' = -w\,y\_1,$$

a cui possiamo aggiungere le condizioni iniziali: y_1(0)=1, y_2(0)=1. Qui w indica un parametro che potremo in seguito far variare.
Ecco allora come risolvere il problema:

```
%Oscillatore armonico
tspan=[0 10];
yzero=[1; 1]; %vettore colonna dei dati iniziali
global w; % parametro da cui dipende l'oscillatore
w=1;

[t,y]=ode45(@fun_osc,tspan,yzero);
%Output: t array di punti ordinati in [0 10], con  t(1)=0,
%        t(end)=10 e t(2:end-1) scelti in maniera adattiva
%
%        y matrice di valori approssimati length(t)x2
%        y(j,1)~y_1(t_j), y(j,2)~y_2(t_j)

plot(y(:,1),y(:,2)) %  disegno traiettoria nel piano delle fasi
hold on
xlabel('u(t)');ylabel('du(t)/dt')

%Grafico campo vettoriale in [-5,5]x[-3,3]
[Y1,Y2]=meshgrid(-5:0.5:5,-3:0.5:3);
DY1Dt=Y2; DY2Dt=-w*(Y1);  %CAMPO VETTORIALE
quiver(Y1,Y2,DY1Dt,DY2Dt)
axis equal
axis([-5 5 -3 3])
hold off
```

La funzione fun_osc avrà la forma:

```
function [y]=fun_osc(t,x)
% funzione per l'oscillatore armonico
global w
  y=[x(2); -w*x(1)];
end
```

Notare l'uso della variabile globale w (basta definirla nel main perché sia nota alla function senza passarla come argomento). Come cambiano le traiettorie in funzione di w?
Notare anche come si estende l'uso di *quiver* al caso del campo di velocità nel piano delle fasi.

Vediamo ora un altro esempio di un sistema di ode che presenta un 'attrattore' per le traiettorie nel piano delle fasi.

Esempio (**Attrattore di Lorenz**)
Consideriamo per un dato parametro *mu* il sistema differenziale:

$$y\_1' = -y\_2 + mu*y\_1*(1-(y\_2)^2), \quad y\_2' = y\_1$$

Vogliamo vedere contemporaneamente nel piano delle fasi le traiettorie di tutte le soluzioni uscenti da una griglia uniforme di punti. Analizziamo il seguente programma e le function che utilizza.

```
% attrattore di Lorenz
close
t0=0; tf=30;
mu=1;
for i=-4:4
    for j=-4:4
    y0=[i;j];
[t,y]=risolveedo(y0,t0,tf,mu);
plot(y(1,1),y(1,2),'or',y(:,1),y(:,2))
axis([-5 5 -5 5])
hold on
xlabel('y_1(t)');ylabel('y_2(t)')
    end
end
hold off
title('Attrattore di Lorenz')


function [t,y]=risolveedo(y0,t0,tf,mu)
tspan=[t0:0.05:tf];
[t,y]=ode45(@(t,y)Lorenz(t,y,mu),tspan,y0);
end


function [ F ] = Lorenz( t,y,mu )
% EDO di Lorenz
F(1,1)=-y(2)+mu*y(1).*(1-y(2).^2);
F(2,1)=y(1);
end
```

Fatelo ora girare e osservate i risultati. Provate a modificare il parametro *mu* e vedete qual è l'effetto.

Esempio. (**Metodi numerici**)
Per finire vogliamo implementare direttamente degli schemi numerici per le ode senza utilizzare la funzione *ode45* di Matlab. In particolare nel seguente programma si permette di scegliere tra i noti metodi espliciti di Eulero e di Heun, si calcolano le soluzioni per un numero crescente di punti nell'intervallo di tempo fissato (riducendo il passo delta) e si determina il valore approssimato dell'ordine del metodo mediante il metodo dei minimi quadrati.

```
% soluzione numerica di y'(t)=f(y(t)), y(0)=yo
% con i metodi di Eulero esplicito e Heun. Nell'esempio f(y)=c*y
% e la soluzione esatta e' y(t)=yo*e^(c*t).
clear
close all
% cominciamo con il fissare y0, c e il tempo finale T.
yo=1; c=1; T=1;

% scelta del metodo
met=input('Scelta metodo (tra apici): E=Eulero, H=Heun ')

for j=1:5
    n=3^(j+1);
    w=linspace(0,T,n+1);
    delta=1/(n);

    z=yo*exp(c*w); % soluzione esatta

    u=zeros(1,n+1);
    u(1)=yo;

    for i=1:n
        if met=='E'
            metodo='Eulero';
            u(i+1)=u(i)+c*u(i)*delta; %    Eulero
        else
             metodo='Heun';
             u(i+1)=u(i)+1/2*(c*u(i)+c*(u(i)+c*u(i)*delta))*delta; %    Heun
        end
    end

     subplot(3,2,j)
     plot(w,u,'.',w,z,'r');
     title(['Metodo di ',metodo,'; numero punti = ',num2str(n+1)])

     num(j)=log10(delta);
     errori(j)=log10(max(abs(u-z)));
 end

% Calcoliamo la retta dei minimi quadrati per ottenere l'ordine con cui va a
% zero l'errore tra la soluzione numerica e quella esatta in funzione di delta

a=(errori-mean(errori))*(num-mean(num))'/((num-mean(num))*(num-mean(num))')
b=mean(errori)-a*mean(num)
z=a*num+b;
%figure
subplot(3,2,6)
plot(num,errori,'-')
hold on
plot(num,z,'+')
title(['Metodo di ',metodo,'; ordine = ',num2str(a)])
```