```
>> help ode45
 ODE45  Solve non-stiff differential equations, medium order method.
    [TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates
    the system of differential equations y' = f(t,y) from time T0 to TFINAL
    with initial conditions Y0. ODEFUN is a function handle. For a scalar T
    and a vector Y, ODEFUN(T,Y) must return a column vector corresponding
    to f(t,y). Each row in the solution array YOUT corresponds to a time
    returned in the column vector TOUT.  To obtain solutions at specific
    times T0,T1,...,TFINAL (all increasing or all decreasing), use TSPAN =
    [T0 T1 ... TFINAL].

% Tutti i solutori predefiniti in MatLab hanno la stessa sintassi
% Quelli più comunemente usati sono
%    ode45 : metodo esplicito
%    ode15s: metodo implicito

% Prima si definisce la funzione f(t,y) che compare nell'equazione
% differenziale
>> f = @(t,y) -3*y +sin(t)
f =
    @(t,y)-3*y+sin(t)

% Poi si chiama uno dei solutori

>> [t,y]=ode45(f,[0,2] , 0.5);
>> clf
>> figure
>> plot(t,y,'o-')

% Per usarne un altro, basta chiamare un'altra function, ma la sintassi
% è la stessa

>> hold on
>> [t,y]=ode15s(f,[0,2] , 0.5);
>> plot(t,y,'rx-')

% Consideriamo ora l'esempio
%  y1' = -2*y1 + y2 + 2*sin(t)
%  y2' =  y1 - 2*y2 + 2*(cos(t)-sin(t))
% Poiché f(t,y) deve essere funzione di due argomenti, dobbiamo
% raggruppare le due variabili in una unica variabile vettoriale
% a due componenti

>> f1 = @(t,y) [-2*y(1)+y(2)+2*sin(t);y(1)-2*y(2)+2*(cos(t)-sin(t))]
f1 =
    @(t,y)[-2*y(1)+y(2)+2*sin(t);y(1)-2*y(2)+2*(cos(t)-sin(t))]
>> [t,y] = ode45(f1 , [0,2] , [2;3] );
>> whos t y
  Name      Size            Bytes  Class     Attributes
```

```
  t        41x1              328  double
  y        41x2              656  double
>> plot(t,y,'o-')
>> hold on
>> [t,y] = ode15s(f1 , [0,2] , [2;3] );
>> whos t y
  Name      Size            Bytes  Class     Attributes
  t         15x1              120  double
  y         15x2              240  double
>> plot(t,y,'x-')

% Il metodo esplicito ha impiegato 40 passi a raggiungere il tempo finale
% t=2, quello implicito ne ha usati 14. Dato che i passi del metodo
% implicito sono più costosi, non è detto che ode15s sia più veloce.
% Infatti:

>> tic; [t,y] = ode45(f1 , [0,2] , [2;3] ); toc
Elapsed time is 0.014103 seconds.
>> tic; [t,y] = ode15s(f1 , [0,2] , [2;3] ); toc
Elapsed time is 0.030642 seconds.

% Consideriamo ora un secondo esempio:
%  y1' = -2*y1 + y2 + 2*sin(t)
%  y2' =  998*y1 - 999*y2 + 999*(cos(t)-sin(t))
% Questa volta programmiamo la f(t,y) in un M-file

%%%%%%%%%%%%%  f2.m %%%%%%%%%%%%%%%%%%%%%%%%%%%
function dy = f2(t,y)
dy=zeros(size(y));
s=sin(t);
dy(1) = -2*y(1) + y(2) +2*s;
dy(2) = 998*y(1) - 999*y(2) + 999*(cos(t)-s);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

% In questo modo possiamo salvare risultati intermedi e fare calcoli
% più complessi di quelli permessi lavorando direttamente sul prompt.
% Per usare il file f2.m, si usa la sintassi

>> [t,y] = ode15s(@f2 , [0,2] , [2;3] );
>> whos t y
  Name      Size            Bytes  Class     Attributes
  t         15x1              120  double
  y         15x2              240  double
>> plot(t,y,'*-')
>> [t,y] = ode45(@f2 , [0,2] , [2;3] );
>> whos t y
  Name        Size            Bytes  Class     Attributes
  t         2405x1            19240  double
```

```
  y        2405x2           38480  double
>> plot(t,y,'.-')

% I grafici mostrano che la soluzione esatta è la stessa dell'equazione
% precedente, ma questa volta il comportamento dei metodi numerici è
% profondamente differente. Sebbene l'accuratezza della soluzione sia
% comparabile, il metodo implicito ha impiagato ancora 14 passi, ma
% quello esplicito ne ha dovuti usare 2404. Questa volta la differenza
% è notevole anche sul tempo di calcolo

>> tic; [t,y] = ode15s(@f2 , [0,2] , [2;3] ); toc
Elapsed time is 0.024778 seconds.
>> tic; [t,y] = ode45(@f2 , [0,2] , [2;3] ); toc
Elapsed time is 0.239784 seconds.

% Il motivo è la stabilità dei metodi. Le due funzioni f1(t,y) e f2(t,y)
% hanno matrici delle derivate parziali rispettivamente
% A1 = [-2,1;1,-2]
% A2 = [-2,1;998,-999]
% i cui autovalotri sono
>> eig([-2,1;1,-2])
ans =
   -3
   -1
>> eig([-2,1;998,-999])
ans =
         -1
      -1000
% e quindi il metodo esplicito, nel secondo caso, deve scegliere un
% passo temporale che tenga conto di lambda=1000, il che provoca la
% necessità di usare dt molto piccoli.

% Consideriamo ora un altro esempio:
% y1' = 10*( y1 -1/3*y1^3 -y2
% y2' = 1/10 * y1

>> vdp=@(t,y)[10*(y(1)-1/3*(y(1)^3)-y(2));y(1)/10]
vdp =
   @(t,y)[10*(y(1)-1/3*(y(1)^3)-y(2));y(1)/10]
>> [t45,y45]=ode45(vdp,[0,20],[1;1]);
>> [t15,y15]=ode15s(vdp,[0,20],[1;1]);
>> whos
  Name       Size           Bytes  Class             Attributes
  t15      105x1              840  double
  t45      529x1             4232  double
  vdp        1x1               16  function_handle
  y15      105x2             1680  double
  y45      529x2             8464  double
```

```
% Per visualizzare il dt scelto dai matodi suddividiamo la finestra
% grafica in due porzioni

>> subplot(2,1,1);
>> plot(t15,y15,'-')
>> hold on
>> plot(t45,y45,'-')
>> subplot(2,1,2);
>> semilogy( t15(1:end-1) , t15(2:end)-t15(1:end-1) )
>> hold on
>> semilogy( t45(1:end-1) , t45(2:end)-t45(1:end-1) ,'r')
>> legend('ode15','ode45')

% Analizziamo ora la seguente function che approssiama la soluzione
% di una equazione differenziale non dipendente dal tempo, usando
% un metodo Runge-Kutta esplicito scelto dall'utente.

%%%%%%%%%%%%%%%%%%%%% RKstepper.m %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
function [T,Y]=RKstepper(f,y0,tfin,dt,RK)
% Approssima la soluzione di
%   y' = f(y)    <--- senza la dipendenza da t!
% con il metodo Runge-Kutta (esplicito!) specificato
% a partire da y0 (colonna) e usando passo dt
% f(y) deve restituire un vettore colonna
% Se non passo RK, usa Heun come default
if nargin<5
        % RK è una struct che contiene tutti i dati del metodo
   RK.A = [0,0 ; 1,0]; %coefficienti A
   RK.b = [1/2 ; 1/2]; %coefficienti b (in colonna!)
end
if nnz(triu(RK.A,1)>0)
   error 'Il Runge-Kutta deve essere esplicito!';
end
s = length(RK.b); %numero degli stadi
if (size(RK.A)~=[s,s])
   error 'Il numero dei coefficienti del Runge-Kutta è errato!';
end
K = zeros(length(y0),s); %matrice per salvare gli stadi
% uno stadio per colonna, una componente di y per ciascuna riga
t=0; %tempo corrente
T=0; Y=y0; % per l'output
while t<tfin
   if (t+dt>tfin)
      dt=tfin-t;
   end
   for i=1:s
      y1 = y0 + dt * K* (RK.A(i,:)');
      K(:,i) = f(y1); %stadio i-esimo
```

```
   end
   y1 = y0 + dt* (K*RK.b); %valore a t+dt
   t=t+dt;
   T(end+1) = t; %salvo per l'output
   Y(:,end+1) = y1;
   y0 = y1; %preparo per il nuovo passo
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Proviamo ad applicarlo


>> mia=@(y)[10*(y(1)-1/3*(y(1)^3)-y(2));y(1)/10]
mia =
   @(y)[10*(y(1)-1/3*(y(1)^3)-y(2));y(1)/10]
>> [t,y]=RKstepper(mia,[1;1],20,0.1);
>> plot(t,y)
>> hold on
>> [t,y]=RKstepper(mia,[1;1],20,0.01);
>> plot(t,y,'--')
>> [t,y]=RKstepper(mia,[1;1],20,0.001);
>> plot(t,y,'-.')
>> legend('dt=0.1','dt=0.1','dt=0.01','dt=0.01','dt=0.001','dt=0.001')

% Ovviamente dt deve essere sufficientemente piccolo perché il metodo
% risulti stabile!
```