

Equazioni differenziali

Gabriella Puppo

Equazioni differenziali

- Metodi Runge-Kutta
- Sistemi di equazioni differenziali
- Equazioni differenziali in Matlab

Metodi Runge-Kutta

Una function che implementa un metodo di Runge-Kutta deve avere le seguenti caratteristiche:

- In input deve avere: la funzione f , i dati iniziali, t_0 e y_0 , l'istante t_f in cui si desidera la soluzione, il numero di passi.
- In output, bisogna fornire la soluzione $y(t_f)$. Le functions che seguono danno qualcosa in più: in output c'è il vettore t che contiene tutti i nodi usati ed il vettore y , che contiene la soluzione in tutti gli istanti presenti nel vettore t : cioè $y(k)$ contiene la soluzione numerica all'istante $t(k)$.

Function per il metodo Runge-Kutta 1 (Eulero esplicito)

```
function [t,y] = rk1(fun,t0,tf,y0,n)
%RK1 Risolve sistemi di equazioni differenziali con il metodo di Eulero
% [T,Y] = RK1('FUN',T0,TF,Y0,N) integra il sistema di equazioni
% differenziali definito da y'=fun(t,y(t)) dall'istante T0 a TF con
% N passi.
% FUN deve ritornare un vettore della stessa lunghezza di Y0
%
dt=(tf-t0)/n;
t(1) = t0; y(1,:)=y0;
for k=1:n
    t(k+1) = t0+k*dt;
    k1=feval(fun,t(k),y(k,:));
    y(k+1,:)=y(k,:)+dt*k1';
end
```

Attenzione!

Se $y'(t) = f(t, y(t))$ definisce un'unica equazione, allora il vettore y ha una sola colonna.

Se invece $y'(t) = f(t, y(t))$ è un'equazione vettoriale, allora dobbiamo risolvere un sistema differenziale. In questo caso, y è una matrice, che ha un numero di colonne p uguale alle dimensioni del sistema. Notare che in questo caso, la condizione iniziale y_0 è un vettore riga, con p colonne.

Quindi la function `rk1.m` può essere usata sia per risolvere una singola equazione differenziale, che per risolvere un sistema differenziale

Esempio

Considero l'equazione differenziale

$$y' = -y - 5 * \exp(-t) * \sin(5t),$$

con condizione iniziale $y(0)=1$.

La soluzione esatta è $y(t) = \exp(-t) * \cos(5t)$.

Costruisco una function funz.m che contiene il calcolo di:

$$f(t,y) = -y - 5 * \exp(-t) * \sin(5t)$$

```
function f=funz(t,y)
f=-y-5*exp(-t).*sin(5*t);
```

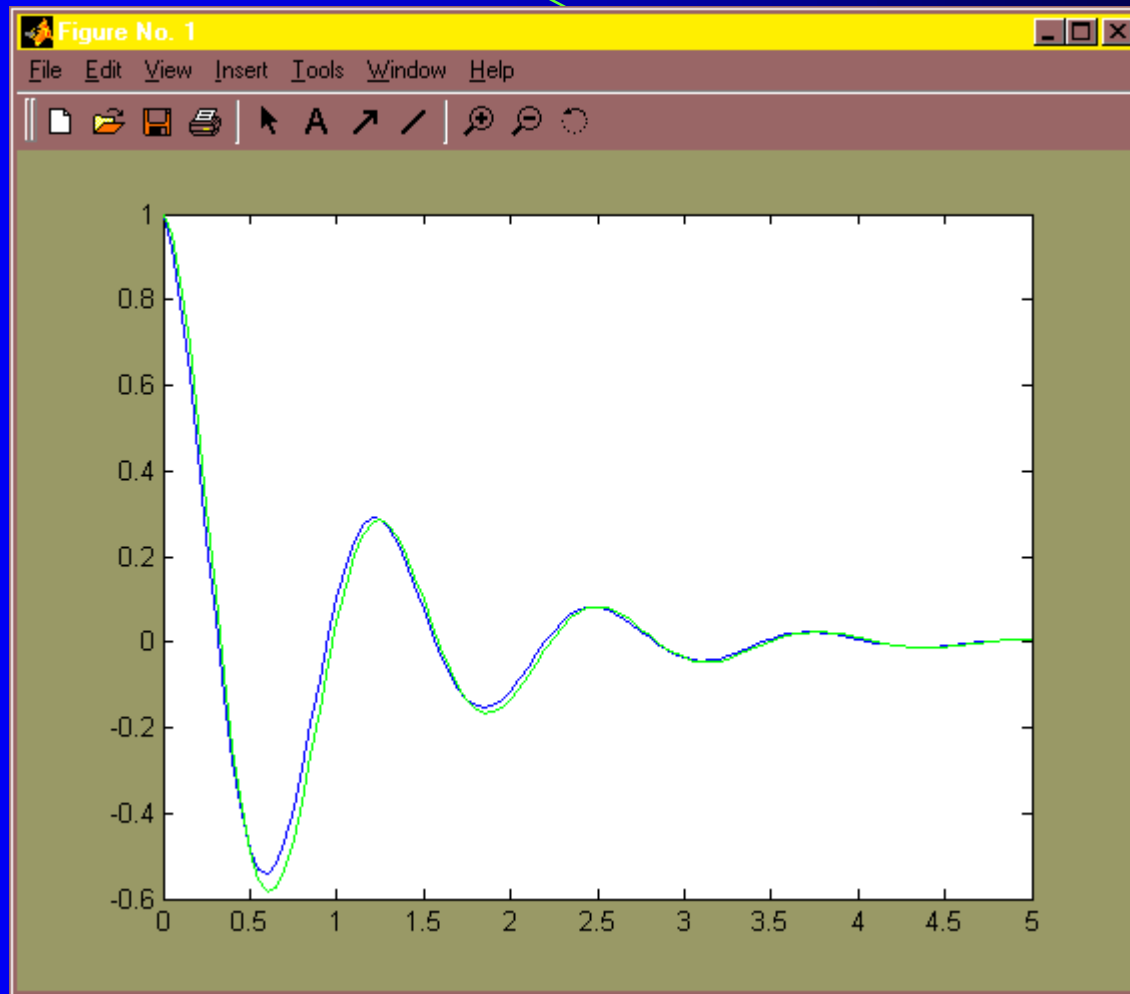
Per risolvere l'equazione differenziale assegnata fino a $t_f=5$, devo dare i seguenti comandi:

```
>>y0=1; t0=0; tf=5;  
>> [t,y1] = rk1('funz',t0,tf,y0,100);
```

Se voglio stampare il grafico della funzione trovata, y_1 , insieme alla soluzione esatta, devo dare i comandi:

```
>> y0=1; t0=0; tf=5;  
>> [t,y1] = rk1('funz',t0,tf,y0,100);  
>> exa=inline('exp(-t).*cos(5*t)');  
>> ye=exa(t);  
>> plot(t,ye)  
>> hold on  
>> plot(t,y1,'g')
```

Ottingo la seguente figura:



Osserviamo che la soluzione numerica ha un'ampiezza leggermente maggiore della soluzione esatta

Aumentiamo l'ordine

I metodi Runge Kutta più accurati del metodo Runge-Kutta 1 che abbiamo appena descritto possono essere implementati in modo molto simile.

In particolare, l'input e l'output sono gli stessi.

All'aumentare dell'ordine, aumenta il numero di valutazioni funzionali che è necessario fare ad ogni passo

Function *rk2.m*

```
function [t,y] = rk2(fun,t0,tf,y0,n)
%RK2 Risolve sistemi di equazioni differenziali con il metodo di Heun
% [T,Y] = RK2('FUN',T0,TF,Y0,N) integra il sistema di equazioni
% differenziali definito da y'=fun(t,y(t)) dall'istante T0 a TF
% N passi. FUN deve ritornare un vettore della stessa lunghezza di Y0
%
dt=(tf-t0)/n;
t(1) = t0; y(1,:)=y0;
for k=1:n
    t(k+1) = t0+k*dt;
    k1 = feval(fun,t(k),y(k,:));
    k2 = feval(fun,t(k)+dt,y(k,:)+dt*k1');
    y(k+1,:)=y(k,:)+dt/2*(k1'+k2');
end
```

Function rk3.m

```
function [t,y] = rk3(fun,t0,tf,y0,n)
%RK3 Risolve sistemi di equazioni differenziali con il metodo di Heun
% [T,Y] = RK3('FUN',T0,TF,Y0,N) integra il sistema di equazioni
% differenziali definito da y'=fun(t,y(t)) dall'istante T0 a TF
% N passi. FUN deve ritornare un vettore della stessa lunghezza di Y0
%
dt=(tf-t0)/n;
t(1) = t0; y(1,:)=y0;
for k=1:n
    t(k+1) = t0+k*dt;
    k1 = feval(fun,t(k),y(k,:));
    k2 = feval(fun,t(k)+dt/3,y(k,:)+dt/3*k1');
    k3 = feval(fun,t(k)+dt*2/3,y(k,:)+dt*2/3*k2');
    y(k+1,:)=y(k,:)+dt/4*(k1'+3*k3');
end
```

Function *rk4.m*

```
function [t,y] = rk4(fun,t0,tf,y0,n)
%RK4 Risolve sistemi di equazioni differenziali con il metodo RK4
% [T,Y] = RK4('FUN',T0,TF,Y0,N) integra il sistema di equazioni
% differenziali definito da y'=fun(t,y(t)) dall'istante T0 a TF
% N passi. FUN deve ritornare un vettore della stessa lunghezza di Y0
%
dt=(tf-t0)/n;
t(1) = t0; y(1,:)=y0;
for k=1:n
    t(k+1) = t0+k*dt;
    k1 = feval(fun,t(k),y(k,:));
    k2 = feval(fun,t(k)+dt/2,y(k,:)+dt/2*k1');
    k3 = feval(fun,t(k)+dt/2,y(k,:)+dt/2*k2');
    k4 = feval(fun,t(k)+dt,y(k,:)+dt*k3');
    y(k+1,:)=y(k,:)+dt/6*(k1'+2*k2' +2*k3' +k4');
end
```

Esempio

Considero lo stesso problema ai valori iniziali di prima:

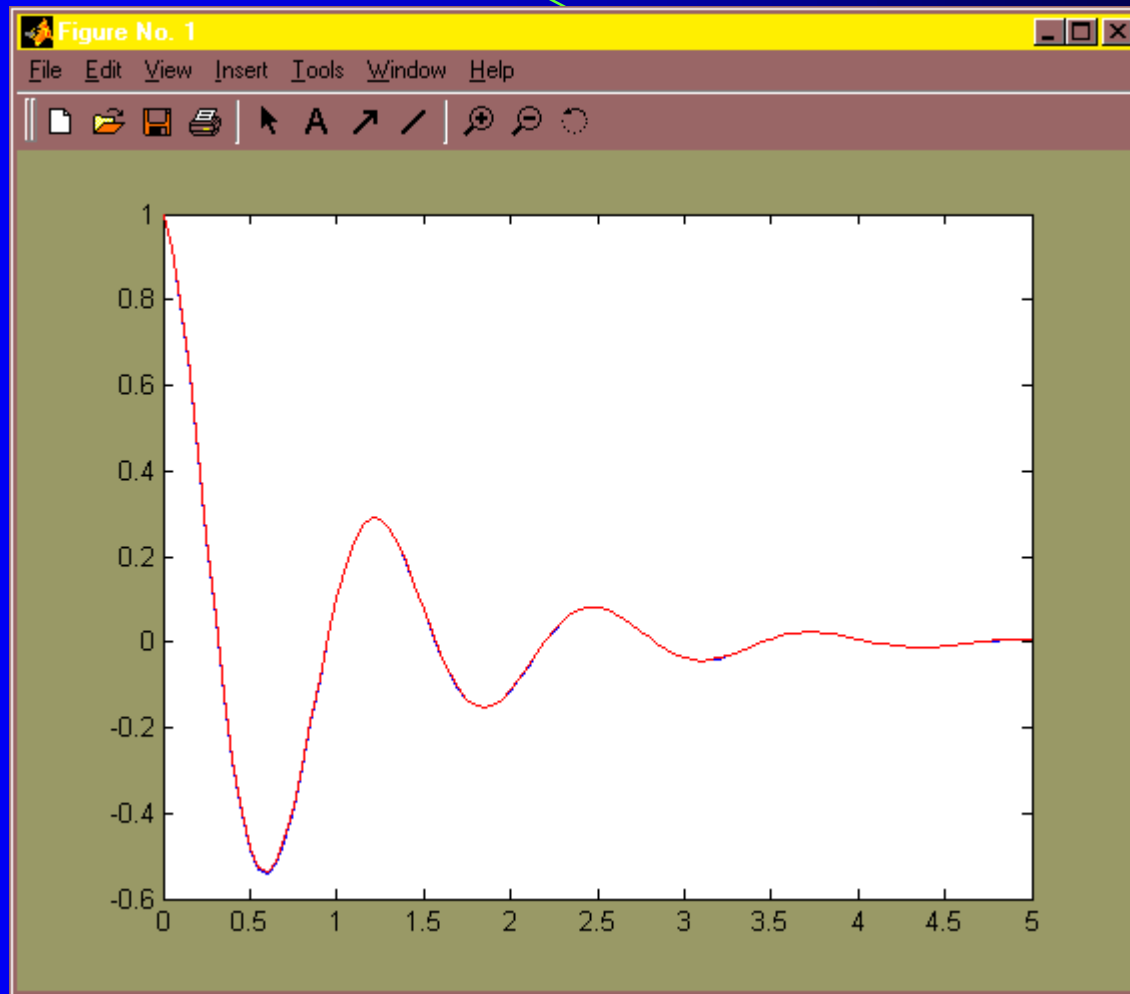
$$y' = -y - 5 * \exp(-t) * \sin(5t),$$

con condizione iniziale $y(0)=1$.

La soluzione esatta è $y(t) = \exp(-t) * \cos(5t)$.

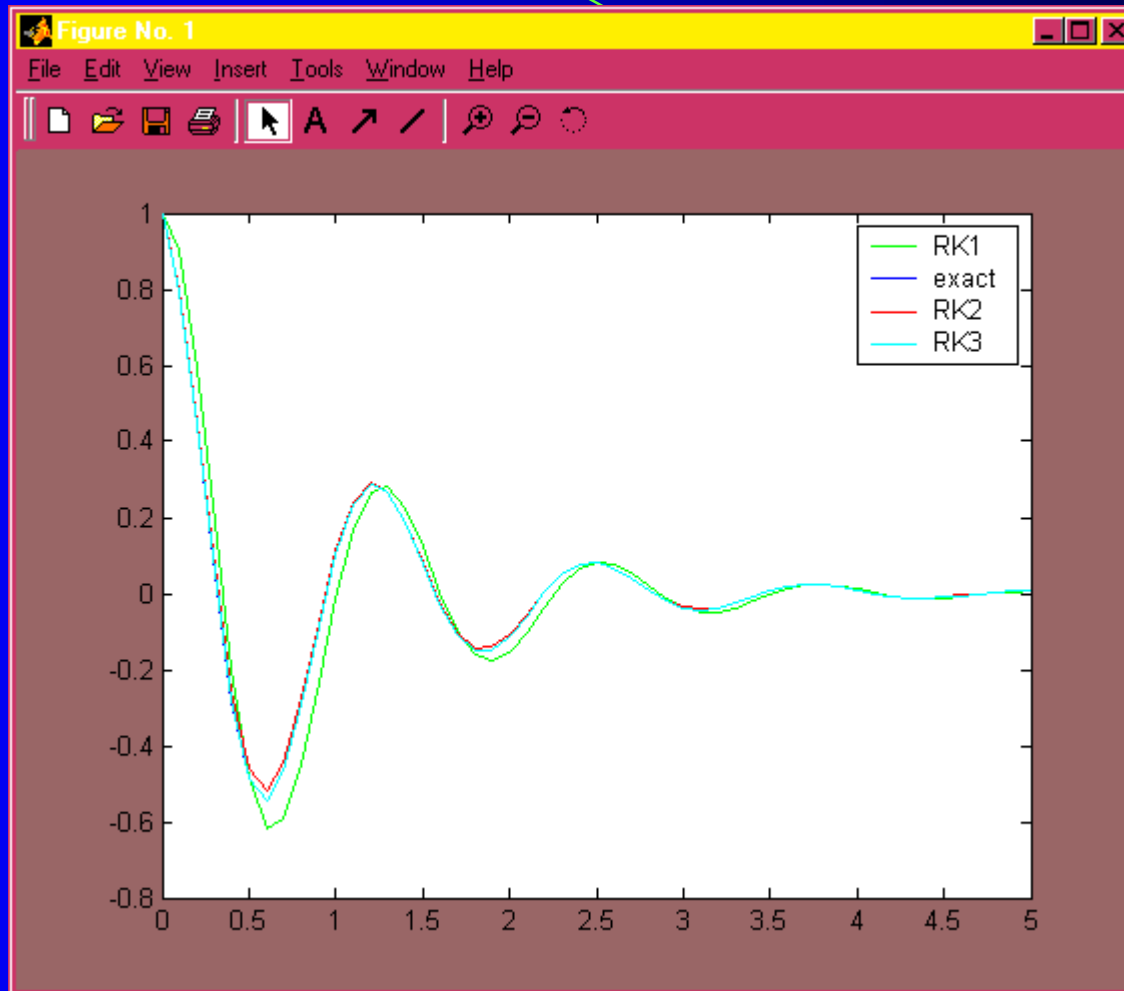
```
>> [t,y2]=rk2('funz',t0,tf,y0,100);  
>> ye=exa(t);  
>> plot(t,ye)  
>> hold on  
>> plot(t,y2,'r')
```

Questa volta ottengo questa figura:



Con 100 punti ed il metodo Runge-Kutta 2, la soluzione esatta e quella numerica sono quasi indistinguibili.

Confronto i metodi RK1, RK2, RK3 sullo stesso grafico, usando lo stesso numero di punti



La soluzione esatta è nascosta dalla soluzione prodotta da RK3.

Accuratezza

- Studio l'andamento dell'errore al diminuire del passo di integrazione dt
- Calcolo l'accuratezza per i metodi RK

Andamento dell'errore

Studio l'andamento dell'errore, usando l'equazione
 $y' = -y - 5 \exp(-t) \sin(5t)$, $y(0) = 1$
con soluzione esatta $y(t) = \exp(-t) \cos(5t)$

Confronto la soluzione esatta e la soluzione numerica ad un istante fissato, per esempio in questo caso $t_f = 2$.

Dimezzo il passo di integrazione e ripeto il calcolo.

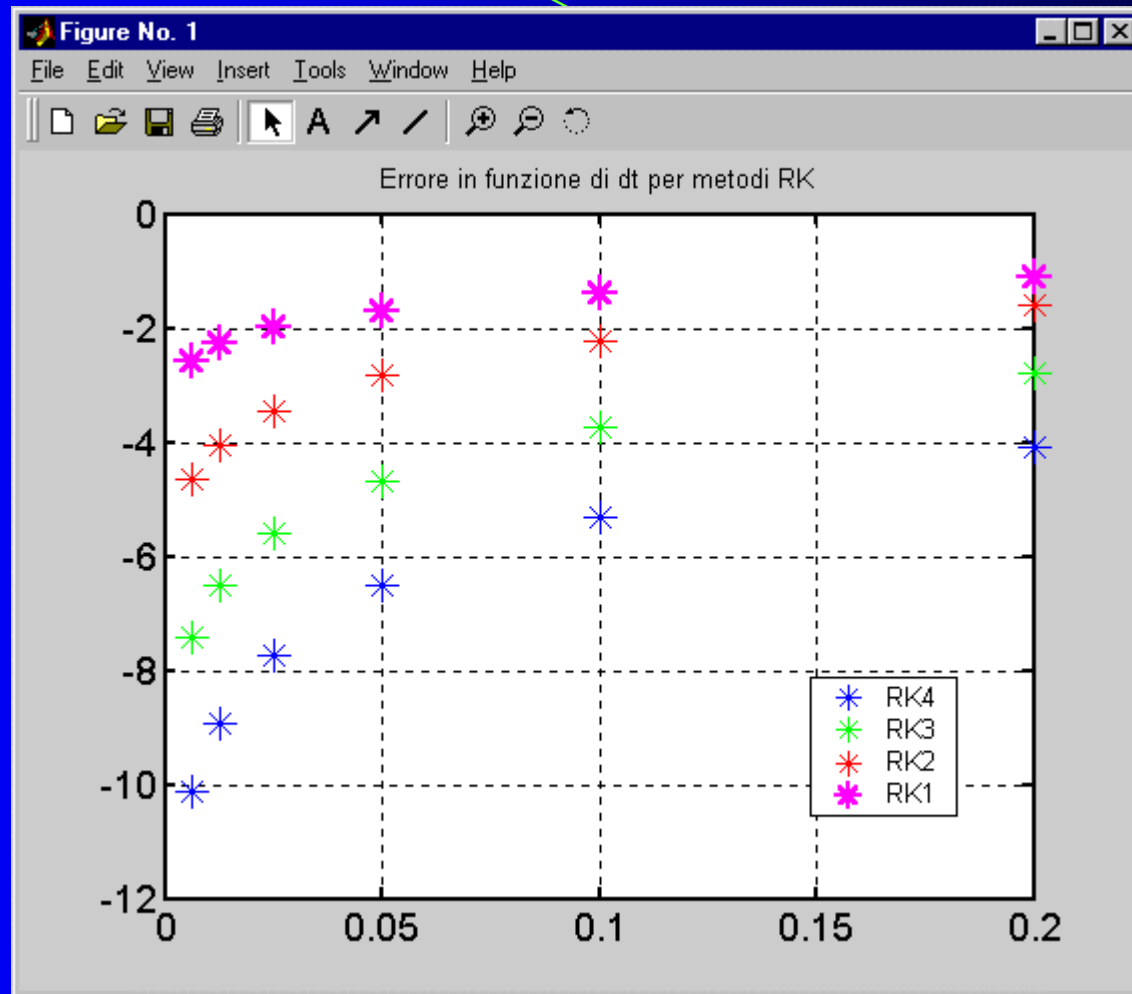
Otengo l'errore in funzione di dt

Calcolo dell'errore in funzione di dt per il metodo RK1

```
% Calcola l'errore in funzione di delta t per metodi RK
% usando l'equazione  $y' = -y - 5 \exp(-t) \sin(5t)$ ,  $y(0) = 1$ 
% con soluzione esatta  $y(t) = \exp(-t) \cos(5t)$ .
% NMAX deve essere impostato dall'esterno
%
exa=inline('exp(-t)*cos(5*t)');
f=inline('-y-5*exp(-t).*sin(5*t)','t','y');
exact=exa(2);          % Calcola l'errore in t=2
n=10; t0=0; tf=2; y0=1;
for k=1:nmax
    [t,y] = rk1(f,t0,tf,y0,n); % Sceglie il metodo RK
    y_rk=y(length(y));
    err_rk(k)=abs(y_rk-exact);
    deltat(k) = t(2)-t(1);
    n=n*2;
end
```

Per calcolare l'errore per un altro schema, cambiare l'istruzione **rossa**

Risultati ottenuti con i 4 schemi Runge-Kutta studiati



Calcolo dell'accuratezza

Uso lo stesso esempio di prima,
 $y' = -y - 5 \exp(-t) \sin(5t)$, $y(0) = 1$
con soluzione esatta $y(t) = \exp(-t) \cos(5t)$.

Calcolo l'errore diverse volte, dimezzando dt ogni volta.

Ottengo l'errore in funzione di dt e calcolo l'accuratezza di ogni schema, in modo analogo a quanto visto per le formule di quadratura.

Script acc.m per stimare numericamente l'accuratezza degli schemi Runge Kutta.

1) Impostazione del problema e dei dati iniziali

```
%  
% calcola l'accuratezza dei metodi RK  
% usando l'equazione  $y' = -y - 5 \exp(-t) \sin(5t)$ ,  $y(0) = 1$   
% con soluzione esatta  $y(t) = \exp(-t) \cos(5t)$   
%  
exa=inline('exp(-t)*cos(5*t)');  
f=inline('-y-5*exp(-t).*sin(5*t)', 't', 'y');  
exact=exa(2);  
n=10; t0=0; tf=2; y0=1;  
fprintf(' n      RK1      RK2      RK3      RK4\n')  
nmax=11;
```

Calcolo dell'errore

```
for k=1:nmax
    [t,y] = rk1(f,t0,tf,y0,n);
    y_rk1=y(length(y));
    [t,y] = rk2(f,t0,tf,y0,n);
    y_rk2=y(length(y));
    [t,y] = rk3(f,t0,tf,y0,n);
    y_rk3=y(length(y));
    [t,y] = rk4(f,t0,tf,y0,n);
    y_rk4=y(length(y));
    err_rk1(k)=abs(y_rk1-exact);
    err_rk2(k)=abs(y_rk2-exact);
    err_rk3(k)=abs(y_rk3-exact);
    err_rk4(k)=abs(y_rk4-exact);
    fprintf('%5.0f %12.6e %12.6e %12.6e %12.6e \n',n, err_rk1(k), ...
           err_rk2(k), err_rk3(k), err_rk4(k))
    n=n*2;
end
```

Calcola l'accuratezza

```
%stampa l'accuratezza
fprintf('\n  accuratezza  \n')
for k=2:nmax
    acc_rk1=log(err_rk1(k-1)/err_rk1(k))/log(2);
    acc_rk2=log(err_rk2(k-1)/err_rk2(k))/log(2);
    acc_rk3=log(err_rk3(k-1)/err_rk3(k))/log(2);
    acc_rk4=log(err_rk4(k-1)/err_rk4(k))/log(2);
    fprintf('%4.0f  %12.6e  %12.6e  %12.6e  %12.6e \n',k, acc_rk1, acc_rk2, ...
           acc_rk3, acc_rk4)
end
```

N.B. Un'istruzione troppo lunga può essere continuata in una riga successiva. Per segnalare a Matlab che la riga non è terminata, ma continua sulla riga seguente, devo inserire i caratteri ... prima di andare a capo

Lanciando il programma acc.m, con nmax=11, ottengo:

```
>> acc
```

n	RK1	RK2	RK3	RK4
10	7.537914e-002	2.440271e-002	1.632834e-003	8.061451e-005
20	4.018615e-002	5.915579e-003	1.792026e-004	4.850098e-006
40	2.068962e-002	1.460674e-003	2.112258e-005	2.998153e-007
80	1.049248e-002	3.631979e-004	2.568070e-006	1.867181e-008
160	5.283077e-003	9.057239e-005	3.167163e-007	1.165466e-009
320	2.650744e-003	2.261591e-005	3.932803e-008	7.280263e-011
640	1.327673e-003	5.650656e-006	4.899868e-009	4.549056e-012
1280	6.644120e-004	1.412253e-006	6.114816e-010	2.841338e-013
2560	3.323498e-004	3.530123e-007	7.637246e-011	1.817990e-014
5120	1.662109e-004	8.824672e-008	9.542825e-012	1.276756e-015
10240	8.311442e-005	2.206089e-008	1.192643e-012	8.049117e-016

... continua ...

accuratezza

2	9.074671e-001	2.044450e+000	3.187715e+000	4.054954e+000
3	9.577908e-001	2.017885e+000	3.084734e+000	4.015868e+000
4	9.795518e-001	2.007807e+000	3.040030e+000	4.005140e+000
5	9.899053e-001	2.003613e+000	3.019422e+000	4.001883e+000
6	9.949811e-001	2.001733e+000	3.009561e+000	4.000772e+000
7	9.974972e-001	2.000848e+000	3.004743e+000	4.000352e+000
8	9.987502e-001	2.000419e+000	3.002362e+000	4.000925e+000
9	9.993755e-001	2.000209e+000	3.001185e+000	3.966154e+000
10	9.996878e-001	2.000104e+000	3.000564e+000	3.831789e+000
11	9.998439e-001	2.000052e+000	3.000254e+000	6.655810e-001

>>

Notare che l'accuratezza dello schema RK4 si deteriora su griglie troppo fini, perché l'errore è paragonabile all'errore di macchina

Sistemi di equazioni differenziali

Per integrare sistemi di equazioni differenziali, devo scrivere una function che:

- riceva in input l'istante t ed il vettore y , contenente le incognite al tempo t .
- dia in output il vettore f contenente i valori $f(t,y)$. **Attenzione:** per poter essere usato dalle functions per ODE di Matlab, f deve essere un vettore colonna

Esempio: pendolo semplice

Il file *pend_lin.m* contiene le equazioni per definire il sistema differenziale del pendolo semplice smorzato

```
function f=pend_lin(t,y)
%PENDOLO Costruisce la funzione F per il sistema Y'=F(T,Y),
% dove F definisce il sistema differenziale che descrive il
% moto di un pendolo lineare di frequenza BETA e
% smorzamento ALPHA
alpha=-1;
beta=20;
f(1) = y(2);
f(2) = -beta^2*y(1) +alpha*y(2);
f=f; % Trasformo f in vettore colonna
```

Per integrare le equazioni del pendolo semplice con il metodo di Runge-Kutta 4, dobbiamo dare i seguenti comandi:

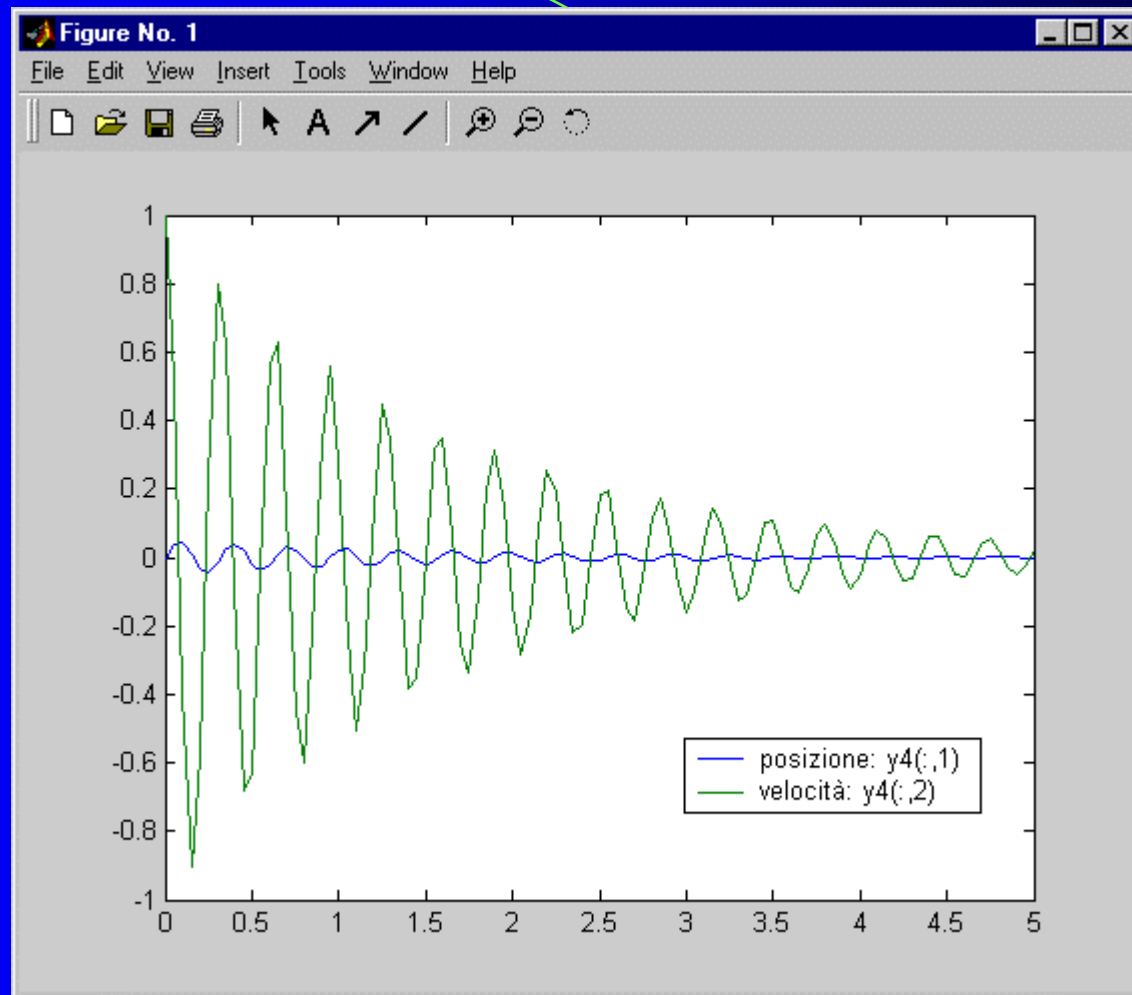
```
>> [t,y4]=rk4('pend_lin',0,5,[0,1],100);  
>> plot(t,y4)
```

Con questo comando, integriamo le equazioni del pendolo da $t_0 = 0$ a $t_f = 5$, con condizione iniziale $y(t_0)=[0,1]$, usando 100 punti di integrazione

La matrice y_4 contiene due colonne, la prima colonna è $y_4(:,1)$ e contiene la posizione del pendolo all'istante t ; la seconda colonna è $y_4(:,2)$ e contiene la velocità del pendolo all'istante t .

L'istruzione $\text{plot}(t,y_4)$ fa un grafico contenente tutte e due le componenti di y_4 :

Posizione e velocità per un pendolo lineare



Equazioni differenziali in Matlab

Matlab dispone di diverse functions per risolvere sistemi di equazioni differenziali:

- Metodi espliciti: *ode23* (basato su schemi Runge-Kutta di ordine 2 e 3); *ode45* (basato su schemi Runge-Kutta di ordine 4 e 5); *ode113* (basato su schemi multistep di ordine variabile da 1 a 13)
- Metodi impliciti: *ode15s* (schemi multistep di ordine variabile da 1 a 5)

Sintassi

Tutte le functions per ODE di Matlab sono basate sulla stessa sintassi. I primi esempi saranno per la function `ode45`, che è la più usata, ma possono essere applicati a tutte le altre ODE functions.

La chiamata più semplice è:

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

Qui:

- `fun` è una stringa di caratteri che contiene il nome della funzione $f(t,y)$ che definisce il sistema differenziale.
- `tspan` è un vettore che contiene l'istante iniziale e l'istante finale: `tspan=[t0, tf]`.
- `y0` contiene le condizioni iniziali: è uno scalare se c'è un' unica equazione differenziale; è un vettore con m componenti se dobbiamo integrare un sistema con m equazioni.

Esempio 1

Considero ancora l'equazione differenziale
 $y' = -y - 5 \cdot \exp(-t) \cdot \sin(5t)$,

con condizione iniziale $y(0)=1$.

La soluzione esatta è $y(t) = \exp(-t) \cdot \cos(5t)$.

Uso la function `funz.m` che contiene il calcolo di:

$$f(t,y) = -y - 5 \cdot \exp(-t) \cdot \sin(5t)$$

Per integrare l'equazione differenziale da $t_0=0$ a $t_f=5$, con condizione iniziale $y_0=1$, devo dare il comando:

```
>> [t,y]=ode45('funz',[0,5],1);
```

Notare che non devo inserire il numero di passi, perché le functions di Matlab sono adattative

Esempio 2

Per integrare il sistema differenziale con le equazioni del pendolo, devo dare il comando:

```
>> [t,y]=ode45('pend_lin',[0,5],[0,1]);
```

In questo caso, le equazioni del pendolo vengono integrate da $t_0=0$ a $t_f=5$, con condizioni iniziali $y_0(0) = [0,1]$, cioè con posizione iniziale uguale a zero, e velocità iniziale uguale a 1

Impostare le tolleranze

Le functions di Matlab per ODE sono adattative, come la function quad per l'integrazione numerica.

Il passo di integrazione viene calcolato dalla function stessa, in modo da assicurare che l'errore locale sia dell'ordine di una tolleranza prefissata.

Anche per le functions per ODE, ci sono dei valori di default per la tolleranza. E' però possibile anche impostare in input la tolleranza desiderata.

Le functions per ODE usano due tolleranze, RelTol e AbsTol. L'errore stimato ad ogni passo per la componente i della soluzione y soddisfa la stima:

$$e(i) \leq \max(\text{RelTol} * \text{abs}(y(i)), \text{AbsTol}(i))$$

I valori di default sono:

RelTol = 1e-3

AbsTol = 1e-6

Per cambiare i valori di default, devo dare due comandi:

Supponiamo di voler impostare RelTol = 1e-4 e AbsTol = 1e-7:

```
>> options=odeset('AbsTol',1e-7,'RelTol',1e-4);  
>> [t,y]=ode45('pend_lin',[0,5],[0,1],options);
```

Il primo comando introduce i nuovi valori nella struttura *options*.

Il secondo comando chiama la function *ode45* con i valori fissati in *options*

Ci aspettiamo che modificando le tolleranze in questo modo, aumenterà il numero di passi usato dalla function *ode45*. Per verificarlo stampiamo il numero di passi:

Diamo quindi i comandi:

```
>> [t,y]=ode45('pend_lin',[0,5],[0,1]);  
>> length(t)  
ans =  
    521  
>> options=odeset('AbsTol',1e-7,'RelTol',1e-4);  
>> [t,y]=ode45('pend_lin',[0,5],[0,1],options);  
>> length(t)  
ans =  
    865
```

Quindi il numero di passi è aumentato da 521 a 865.

Problemi stiff

- Un'equazione (o un sistema di equazioni) si dice stiff, quando uno schema esplicito è costretto ad usare un passo di integrazione molto piccolo, altrimenti la soluzione diventa instabile.
- I problemi stiff si incontrano quando si vuole simulare un fenomeno caratterizzato da un transitorio molto veloce, dopo il quale il sistema si stabilizza su una soluzione che varia più lentamente nel tempo. Esempi tipici di problemi stiff si hanno nella simulazione dei circuiti elettrici e nei fenomeni con reazioni chimiche
- Le functions implicite, come la ode15s, permettono di trattare in modo efficiente anche problemi stiff.