Laboratorio di Metodi Numerici per Equazioni Differenziali Ordinarie

Giuseppe Lombardi

November 9, 2018

Esercitazione 1
Metodo di Eulero esplicito

1. Dato l' IVP generico

$$\begin{cases} x'(t) = f(t, x(t)), \quad t \in [t_0, T] \\ x(t_0) = x_0 \end{cases}$$

con $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$,
implemento su griglia uniforme il **metodo di Eulero esplicito**

```
%Dati di INPUT:
%odefun    numero stime dell' ordine di convergenza
%tspan     intervallo di integrazione
%x0        condizione iniziale
%N         numero di sottointervalli
%varargin  parametri addizionali della funzione odefun
%
%Dati OUTPUT:
%t         punti ecquispaziati interni all' intervallo
%          di integrazione
function [t, x] = euler_forw(odefun, tspan, x0, N, varargin)
t0 = tspan(1);
T = tspan(2);
t = linspace(t0, T, N + 1);
h = t(2) - t(1);
x = zeros(1, N + 1);
x(1) = x0;
for (i = 1 : N)
x(i+1) = x(i) +  h * feval(odefun, t(i), x(i));
endfor
endfunction
```

2. Considerato ora l' IVP

$$\begin{cases} x'(t) = -x(t) + 2cos(t), \quad t \in (0, 5] \\ x(0) = 1 \end{cases}$$

la cui soluzione esatta è $x(t) = sin(t) + cos(t)$, il file di tipo *function* implementa $f(t, x(t))$ di tale problema.

```
function f = funz(t, x)
f = -x + 2 * cos(t);
endfunction
```

Qui sotto il file di tipo *script* **eser2_1.m** calcola la soluzione approssimata con il metodo di Eulero esplicito sulla griglia ecquispaziata di 501 punti. Infine viene calcolato **e** l'errore assoluto in norma infinito.

```
%Dati OUTPUT:
%e         errore assoluto in norma infinito
tspan = [0, 5];
x0 = 1;
[t, x] = euler_forw("funz", tspan, x0, 500);
t = linspace(0, 5, 501);
%x_ex   soluzione esatta
x_ex = sin(t) + cos(t);
e = norm(x_ex - x, inf)
```

L'errore commesso è $\mathbf{e = 0.0050126}$.

3. Dato l' IVP

$$\begin{cases} x'(t) = cos(2x(t)), \quad t \in (0, 1] \\ x(0) = 0 \end{cases}$$

la cui soluzione esatta è

$$x(t) = \frac{1}{2} \arcsin\left(\frac{e^{4t} - 1}{e^{4t} + 1}\right),$$

il file di tipo *function* **ord_euler_forw.m** fornisce, nella variabile **p**, **nstep** stime dell' ordine di convergenza del metodo di Eulero esplicito determinate sfruttando l' errore commesso in $t = 1$.

```
%Dati di INPUT:
%nstep     numero stime dell' ordine di convergenza
%
%Dati OUTPUT:
%p         vettore contenente nstep stime
function p = ord_euler_forw(nstep)
p = zeros(1, nstep);
tspan = [0, 1];
x0 = 0;
%x_ex   sol. esatta
x_ex = 0.5 * asin((exp(4) - 1) / exp(4) + 1);
N = 2;
[t, x] = euler_forw("funz1", tspan, x0, N);
%err_aux   errore commesso in t=1 con passo h
%err   errore commesso in t=1 con passo 2*h
err_aux = norm(x_ex - x(N + 1), inf);
for i = 1 : nstep
N = 2 * N;
[t, x] = euler_forw("funz1", tspan, x0, N);
err = abs(x_ex - x(end));
p(i) = abs(log2(err/err_aux));
err_aux = err;
endfor
endfunction
```

```
function f = funz1(t, x)
f = cos(2 * x);
```

Ponendo nstep = 5:

$$p = (\ 0.0119301,\ 0.0083824,\ 0.0045385,\ 0.0023396,\ 0.0011857\ )$$

Ci ci porta a dire che l' ordine di convergenza del metodo di Eulero è 1.

4. Dato il problema test

$$\begin{cases} x'(t) = \lambda x(t), & t \in (0, T] \\ x(0) = 1 \end{cases}$$

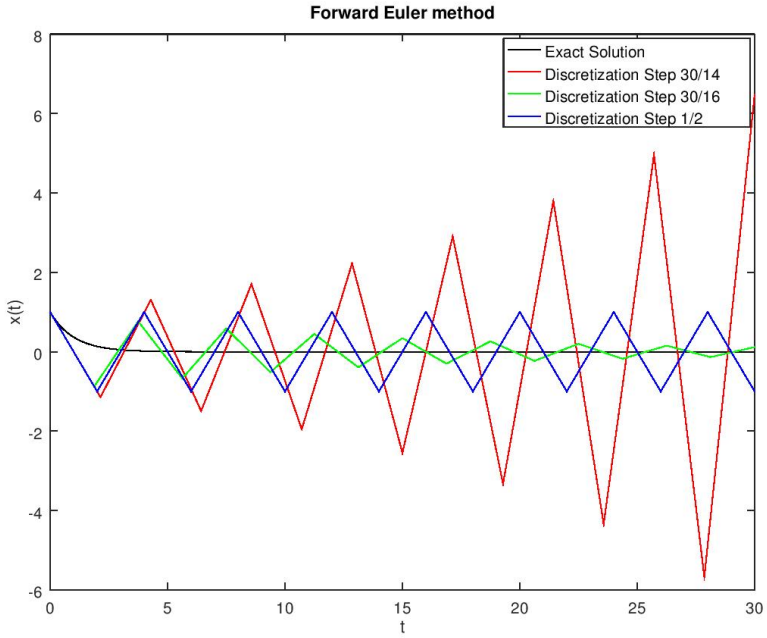
la cui soluzione esatta è

$$x(t) = e^{\lambda t},$$

risolvo il problema con $\lambda = -1$ utilizzando il metodo di Eulero esplicito.

```
tspan = [0 30];
t0 = tspan(1);
T = tspan(2);
x0 = 1;
t = linspace(t0, T, 300);
x_ex = exp(-t);
[t1, x1] = euler_forw("funz3", tspan, x0, 14);
[t2, x2] = euler_forw("funz3", tspan, x0, 16);
[t3, x3] = euler_forw("funz3", tspan, x0, 15);
plot(t, x_ex, "k;Exact Solution;", t1, x1, "r;Discretization...
 Step 30/14;", t2, x2, "g;Discretization Step 30/16;",...
 t3, x3, "b;Discretization Step 1/2;");
xlabel("t");
ylabel("x(t)");
title("Forward Euler method");
function f = funz3(t, x);
f = (-1) * x;
```

Per garantire l' assoluta stabilità dobbiamo avere $h < 2$. Sulla figura qui sotto vengono riportati il grafico della soluzione esatta e le soluzioni ottenute sull' intervallo $[0, 30]$ per tre diversi valori di h:


Forward Euler method
Exact Solution
Discretization Step 30/14
Discretization Step 30/16
Discretization Step 1/2
x(t)
t
8
6
4
2
0
-2
-4
-6
0
5
10
15
20
25
30

Esercitazione 2
Metodo di Eulero implicito

1. Dato l' IVP generico

$$\begin{cases} x'(t) = f(t, x(t)), \quad t \in [t_0, T] \\ x(t_0) = x_0 \end{cases}$$

con $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$,
il file di tipo function implementa su griglia uniforme il **metodo di Eulero implicito**

```
function [t, x] = euler_back(odefun, tspan, x0, N, varargin)
t0 = tspan(1);
T = tspan(2);
t = linspace(t0, T, N + 1);
h = t(2) - t(1);
x = zeros(1,N + 1);
x(1) = x0;
%F funzione da cui calcolare lo zero
%fzero mi approssima lo zero di una funzione
%@(s) definisce una funzione nella variabile s
for i = 1 : N
F = @(s) x(i) - s + h * feval(odefun, t(i + 1), s);
x(i + 1) = fzero(F, x(i));
endfor
endfunction
```

2. Dato l' IVP

$$\begin{cases} x'(t) = 1 - x^2(t), \quad t > 0 \\ x(0) = (e-1)/(e+1) \end{cases}$$

la cui soluzione esatta è

$$x(t) = (e^{2t+1} - 1)/(e^{2t+1} + 1),$$

il file di tipo *function* **ord_euler_back.m** fornisce, nella variabile **p**, **nstep** stime dell' ordine di convergenza del metodo di Eulero implicito determinate sfruttando l' errore commesso in $t = 1$.

```
function p = ord_euler_back(nstep)
tspan = [0 1];
x0 = (e - 1) / (e + 1); %e = exp(1)
x_ex = (exp(3) -1) / (exp(3) + 1);
N = 2;
[t, x] = euler_back("funz2", tspan, x0, N);
%errore commesso in t=1
err_aux = norm(x_ex - x(N + 1), inf);
for i = 1 : nstep
N = 2 * N;
```

```
[t, x] = euler_back("funz2", tspan, x0, N);
err = abs(x_ex - x(end));
p(i) = abs(log2(err/err_aux));
err_aux = err;
endfor
endfunction
function f = funz2(t, x)
f = 1 - x^2;
```

Con nstep = 5,

$$p = (\ 0.88113,\ 0.93559,\ 0.96633,\ 0.98277,\ 0.99128\ )$$

Perci l' ordine di convergenza del metodo di Eulero implicito sembra essere 2.

3. Dato il problema test

$$\begin{cases} x'(t) = \lambda x(t), \quad t \in (0, T] \\ x(0) = 1 \end{cases}$$

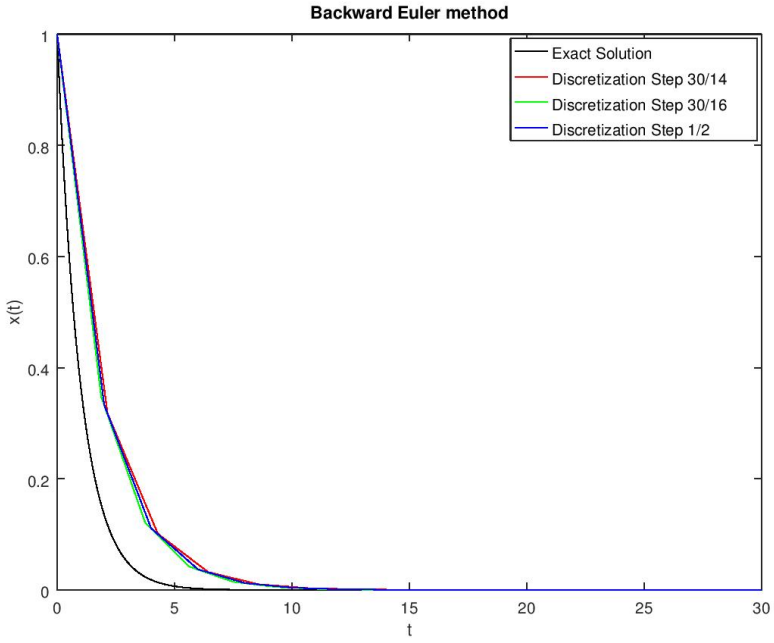
la cui soluzione esatta è

$$x(t) = e^{\lambda t},$$

risolvo il problema con $\lambda = -1$ utilizzando il metodo di Eulero implicito.

```
tspan = [0 30];
t0 = tspan(1);
T = tspan(2);
x0 = 1;
t = linspace(t0, T, 300);
x_ex = exp(-t);
[t1, x1] = euler_back("funz3", tspan, x0, 14);
[t2, x2] = euler_back("funz3", tspan, x0, 16);
[t3, x3] = euler_back("funz3", tspan, x0, 15);
plot(t, x_ex, "k;Exact Solution;", t1, x1, "r;Discretization...
 Step 30/14;", t2, x2, "g;Discretization Step 30/16;",...
 t3, x3, "b;Discretization Step 1/2;");
xlabel("t");
ylabel("x(t)");
title("Backward Euler method");
print -djpg image2_3.jpg
function f = funz3(t, x);
f = (-1) * x;
```

Sulla figura qui sotto vengono riportati il grafico della soluzione esatta e le soluzioni ottenute sull' intervallo $[0, 30]$ per tre diversi valori di h:



Le soluzioni trovate con i 3 valori di h non sono stabili come si puó osservare dalle figura di sopra.

4. Dato l' IVP

$$\begin{cases} x'(t) = \lambda(x(t) - e^{-t}), & t \in (0, T] \\ x(0) = \gamma/(\lambda + 1) \end{cases}$$

la cui soluzione esatta è

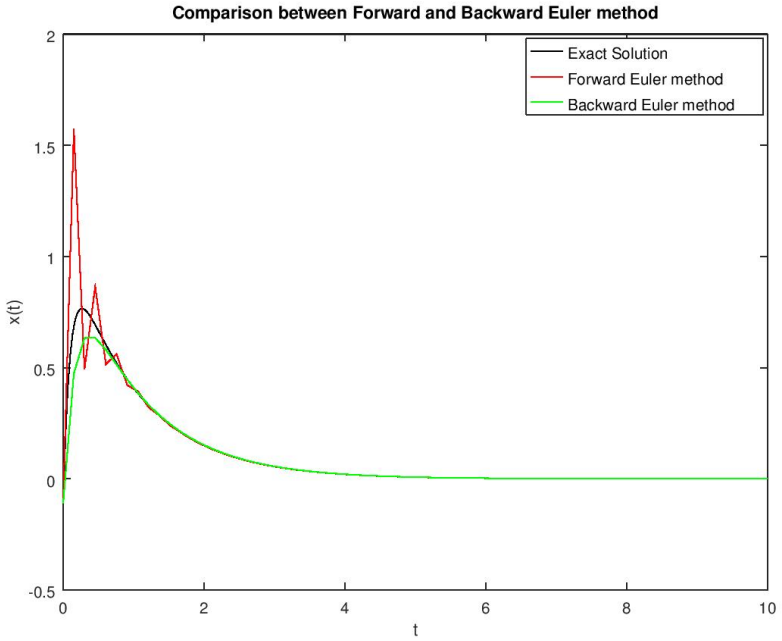
$$x(t) = \frac{\gamma - \lambda}{\lambda + 1} e^{\lambda t} + \frac{\lambda}{\lambda + 1} e^{-t},$$

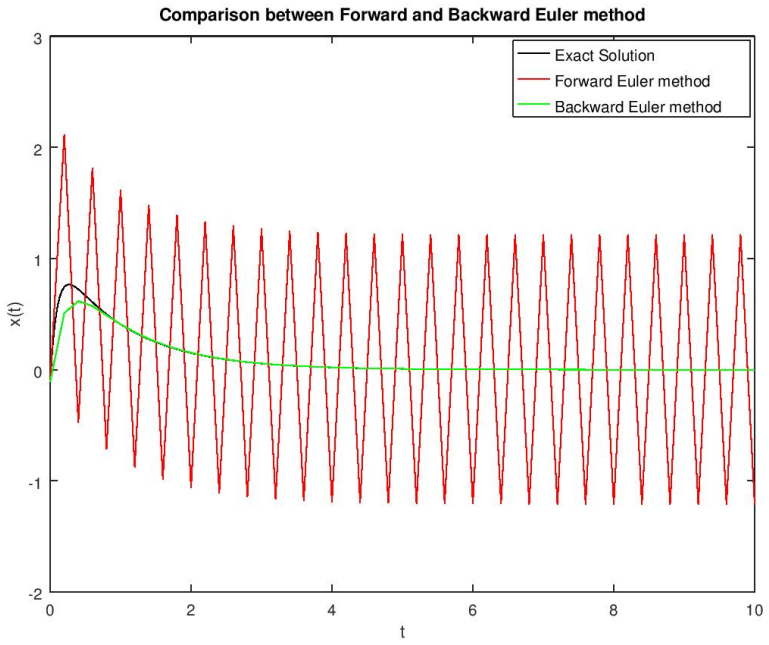
il file di tipo *function* fornisce in output le soluzioni ottenute utlizzando sia il metodo di Eulero esplicito sia il metodo di Eulero implicito con $h = 0.15, 0.2$.

```
function ivp1(gamma)
tspan = [0 10];
t0 = tspan(1);
T = tspan(2);
lambda = -10;
x0 = gamma / (lambda + 1);
h1 = 0.15;
h2 = 0.2;
N1 = 10 / h1;
N2 = 10 / h2;
t = linspace(t0, T, 300);
x_ex = ((gamma - lambda) / (lambda + 1)) * exp((lambda) * t) +...
```

```
(lambda / (lambda + 1)) * exp(-t);
[t1, x1_1] = euler_forw("funz4", tspan, x0, N1);
[t1, x2_1] = euler_back("funz4", tspan, x0, N1);
[t2, x1_2] = euler_forw("funz4", tspan, x0, N2);
[t2, x2_2] = euler_back("funz4", tspan, x0, N2);
figure(1);
plot(t, x_ex, "k;Exact Solution;", t1, x1_1, "r;Forward Euler method;",...
t1, x2_1, "g;Backward Euler method;");
xlabel("t");
ylabel("x(t)");
title("Comparison between Forward and Backward Euler method");
print -djpg image2_4_1.jpg
figure(2);
plot(t, x_ex, "k;Exact Solution;", t2, x1_2, "r;Forward Euler method;",...
t2, x2_2, "g;Backward Euler method;");
xlabel("t");
ylabel("x(t)");
title("Comparison between Forward and Backward Euler method");
print -djpg image2_4_2.jpg
endfunction
```

Qui sotto i grafici della soluzione esatta e delle soluzioni ottenute prima con $h = 0.15$ e poi con $h = 0.2$:





5. Dato l' IVP

$$\begin{cases} x'(t) = cos(t)x(t), & t \in (0, b] \\ x(0) = 1 \end{cases}$$

la cui soluzione esatta è

$$x(t) = e^{sin(t)},$$

il file di tipo *function* fornisce in output gli errori assoluti in norma infinito commessi utlizzando il metodo di Eulero e il metodo di Eulero implicito

```
function ivp2(b, N)
%errore se b = N > 5 in fzero di euler_back
tspan = [0 b];
t0 = tspan(1);
T = tspan(2);
x0 = 1;
t = linspace(t0, T, N + 1);
x_ex = exp(sin(t));
[t1, x1] = euler_forw("funz5", tspan, x0, N);
[t2, x2] = euler_back("funz5", tspan, x0, N);
e_f = norm(x_ex - x1, inf);
e_b = norm(x_ex - x2, inf);
printf("Forward Error %1.4e\n", e_f);
printf("Backward Error %1.4e\n", e_b);
endfunction
```

I dati sono forniti nella tabella sottostante.

b	N	**Eulero esplicito**	**Eulero implicito**
1	10	3.4983e-02	3.5984e-02
	20	1.7625e-02	1.7878e-02
10	100	3.8952e-01	4.5152e-01
	200	2.0148e-01	2.1705e-01
100	1000	2.4603e+00	2.5898e+01
	2000	1.8781e+00	6.0743e+00
1000	1000	2.7183e+00	error
	10000	2.7183e+00	1.7897e+11

Osservando la tabella si nota che gli errori commessi dai metodi aumentano con l'aumentare di b, in modo circa lineare per Eulero esplicito e circa esponenziale per Eulero implicito.

Esercitazione 3
Metodi di Runge-Kutta espliciti

1. Dato l' IVP generico

$$\begin{cases} x'(t) = f(t, x(t)), \quad t \in [t_0, T] \\ x(t_0) = x_0 \end{cases}$$

con $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$,
il file di tipo *function* implementa su griglia uniforme un **metodo di Runge-Kutta esplicito** a m-stadi

```
function [t, x] = RK(odefun, tspan, x0, h, G, beta)
beta = beta(:);
m = size(G, 1);
rho = G * ones(m,1);
t0 = tspan(1);
T = tspan(2);
t = t0 : h : T;
n = length(t);
x = zeros(1, n);
x(1) = x0;
for i = 1 : (n - 1)
k = zeros(m, 1);
for j = 1 : m
k(j) = feval(odefun, t(i) + rho(j) * h, x(i) + h * (G(j, :) * k));
endfor
x(i + 1) = x(i) + h * ((k') * beta);
endfor
endfunction
function f = funz5(t, x)
f = cos(t) * x;
```

2. Dato l' IVP

$$\begin{cases} x'(t) = -x(t) + 2x^2(t)e{-}t, \quad t \in (0, 1] \\ x(0) = 1/3 \end{cases}$$

la cui soluzione esatta è

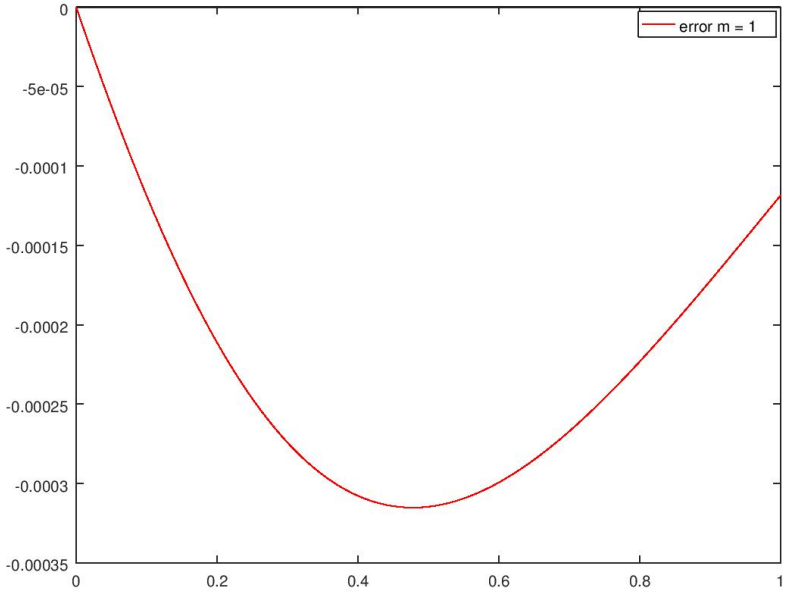
$$x(t) = 1/(2e^t + e^{-t}),$$

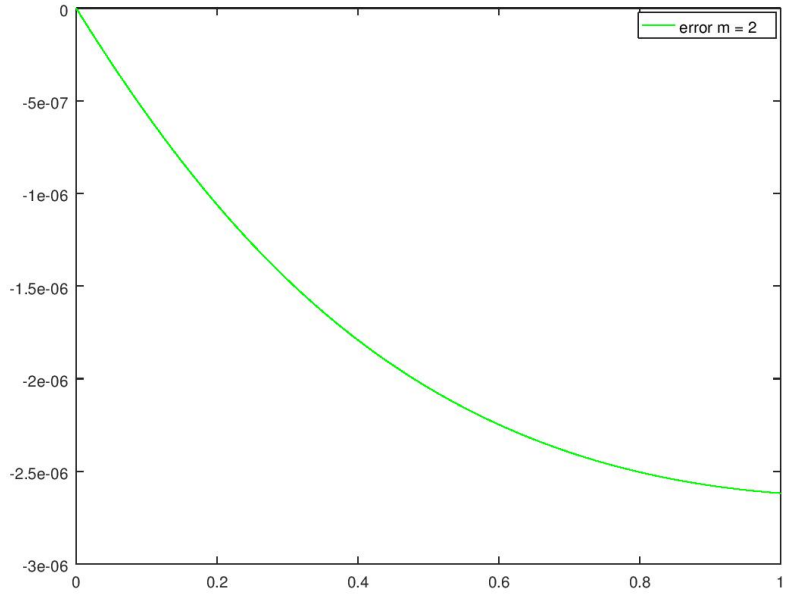
fissato $h = 10^{-2}$, discretizzo il problema utilizzando i metodi di Runge-Kutta espliciti a m-stadi, $m = 1, 2, 3, 4$. Calcolo quindi l' errore assoluto derivato dall' uso di ciascun metodo.

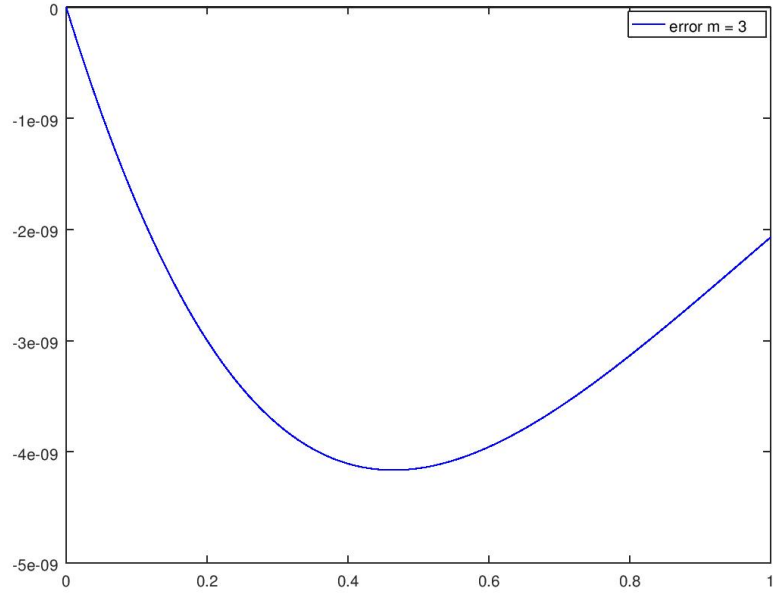
```
tspan = [0 1];
t0 = tspan(1);
T = tspan(2);
x0 = 1/3;
h = 10^(-2);
```

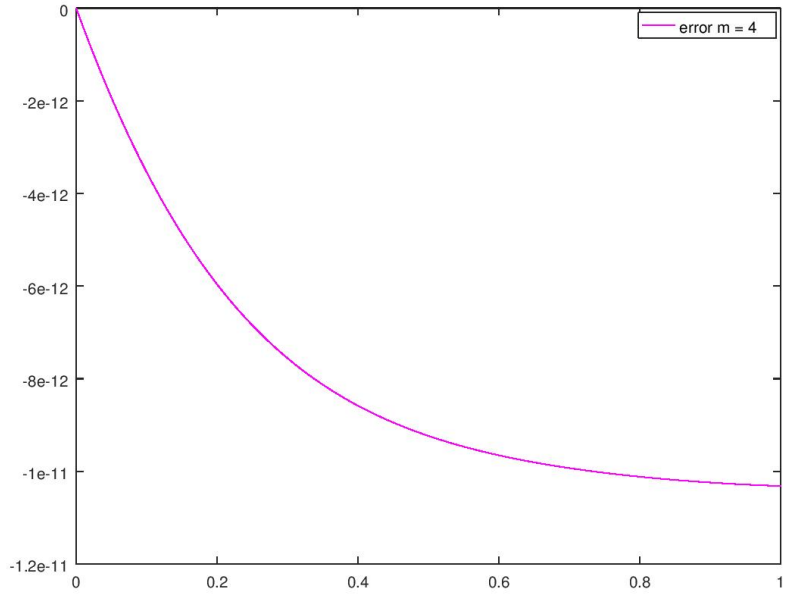
```
t = t0 : h : T;
x_ex = (2 * exp(t) + exp(-t)).^(-1);
G1 = [0];
beta1 = 1;
G2 = [0 0; 1 0];
beta2 = [1/2 1/2];
G3 = [0 0 0; 1/2 0 0; -1 2 0];
beta3 = [1/6 2/3 1/6];
G4 = [0 0 0 0; 1/2 0 0 0; 0 1/2 0 0; 0 0 1 0];
beta4 = [1/6 1/3 1/3 1/6];
[t1, x1] = RK("funz6", tspan, x0, h, G1, beta1);
[t2, x2] = RK("funz6", tspan, x0, h, G2, beta2);
[t3, x3] = RK("funz6", tspan, x0, h, G3, beta3);
[t4, x4] = RK("funz6", tspan, x0, h, G4, beta4);
e_1 = x_ex - x1;
e_2 = x_ex - x2;
e_3 = x_ex - x3;
e_4 = x_ex - x4;
figure(1);
plot(t1, e_1, "r;error m = 1;");
print -djpg image3_2_1.jpg
figure(2); plot(t2, e_2, "g;error m = 2;");
print -djpg image3_2_2.jpg
figure(3); plot(t3, e_3, "b;error m = 3;");
print -djpg image3_2_3.jpg
figure(4); plot(t4, e_4, "m;error m = 4;");
print -djpg image3_2_4.jpg
xlabel("t");
ylabel("e(t)");
title("RK method");
function f = funz6(t, x)
f = -x + 2 * ((x.^2) * exp(-t));
```

I grafici sono riportati qui sotto:


error m = 1
0
-5e-05
-0.0001
-0.00015
-0.0002
-0.00025
-0.0003
-0.00035
0
0.2
0.4
0.6
0.8
1


error m = 2
0
-5e-07
-1e-06
-1.5e-06
-2e-06
-2.5e-06
-3e-06
0
0.2
0.4
0.6
0.8
1





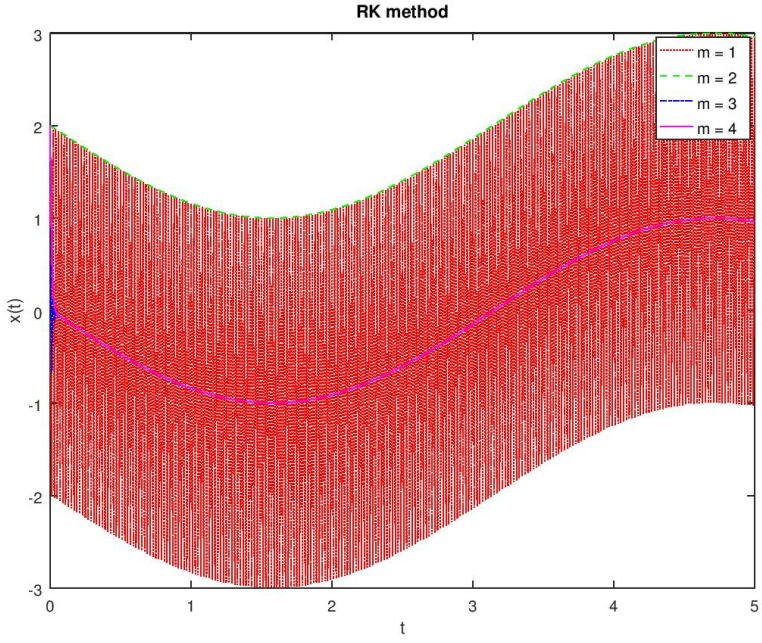
3. Dato l' IVP

$$\begin{cases} x'(t) = -200(x(t) + sin(t)), \quad t \in (0, 5] \\ x(0) = 2 \end{cases}$$

il file di tipo *script*, fissato $h = 10^{-2}$, determina la soluzione approssimata con i metodi di Runge-Kutta espliciti a m-stadi, $m = 1, 2, 3, 4$.

```
tspan = [0 5];
x0 = 2;
%provare diversi passi
h = 1 / 100;
G1 = [0];
beta1 = 1;
G2 = [0 0; 1 0];
beta2 = [1/2 1/2];
G3 = [0 0 0; 1/2 0 0; -1 2 0];
beta3 = [1/6 2/3 1/6];
G4 = [0 0 0 0; 1/2 0 0 0; 0 1/2 0 0; 0 0 1 0];
beta4 = [1/6 1/3 1/3 1/6];
[t1, x1] = RK("funz7", tspan, x0, h, G1, beta1);
[t2, x2] = RK("funz7", tspan, x0, h, G2, beta2);
[t3, x3] = RK("funz7", tspan, x0, h, G3, beta3);
[t4, x4] = RK("funz7", tspan, x0, h, G4, beta4);
plot(t1, x1, ":r;m = 1;", t2, x2, "--g;m = 2;", t3, x3,...
 "-.b;m = 3;", t4, x4, "m;m = 4;");
xlabel("t");
ylabel("x(t)");
title("RK method");
print -djpg image3_3.jpg
function f = funz7(t, x)
f = -200 * (x + sin(t));
```

Il grafico riportato qui sotto:



Fissando $h = 10^{-2}$ il metodo per $m = 1$ è stabile ma non convergente alla soluzione. Fissando $h = 10^{-3}$ i metodi restituiscono soluzioni molto vicine alla soluzione esatta.

4. Considero il moto di un corpo di massa trascurabile soggetto all' attrazione di altri due corpi, uno di massa $\mu = 0.012277471$ e uno di massa $\hat{\mu} = 1 - \mu$, che si muovono in un' orbita planare. Tale moto è governato dalle seguenti equazioni:

$$\begin{cases} u_1''(t) = u_1 + 2u_2' - \hat{\mu}\frac{u_1+\mu}{D_1} - \mu\frac{u_1-\hat{\mu}}{D_2} \\ u_2''(t) = u_2 - 2u_1' - \hat{\mu}\frac{u_2}{D_1} - \mu\frac{u_1-\hat{\mu}}{D_2} \\ u_1(0) = 0.994, u_2 = 0, u_1' = 0, u_2'(0) = -2.00158511, \end{cases}$$

essendo

$$D_1 = ((u_1 + \mu)^2 + u_2^2)^{\frac{3}{2}}, \quad D_2 = ((u_1 - \hat{\mu})^2 + u_2^2)^{\frac{3}{2}}.$$

calcolo la soluzione approssmimata con il metodo (in versione vettoriale) di Runge-Kutta esplicito a 4-stadi, sull' intervallo $[0, 17.1]$ usando $10^2, 10^3, 10^4, 2 * 10^4 passidiintegrazione$. (il file di tipo *function* **moto.m** implementa tali equazioni):

```
function es3_4
tspan = [0 17.1];
u0 = [0.994; 0; 0; -2.00158511];
G4 = [0 0 0 0; 1/2 0 0 0; 0 1/2 0 0; 0 0 1 0];
beta4 = [1/6 1/3 1/3 1/6];
h1 = 17.1 / 1e2;
h2 = 17.1 / 1e3;
h3 = 17.1 / 1e4;
h4 = 17.1 / (2*1e4);

[t, u_h1] = RKvett(@moto, tspan, u0, h1, G4, beta4);
l = length(u_h1);
u1_h1(1) = u_h1(1);
for i = 2 : (l/4)
u1_h1(i) = u_h1(4*(i-1));
endfor
u2_h1(1) = u_h1(2);
for i = 2 : (l/4)
u2_h1(i) = u_h1(4*(i-1) + 1);
endfor
figure(1); plot(u1_h1, u2_h1);
xlabel("u1");
ylabel("u2");
title("RK method");
print -djpg image3_4_1.jpg

[t, u_h2] = RKvett(@moto, tspan, u0, h2, G4, beta4);
l = length(u_h2);
u1_h2(1) = u_h2(1);
```

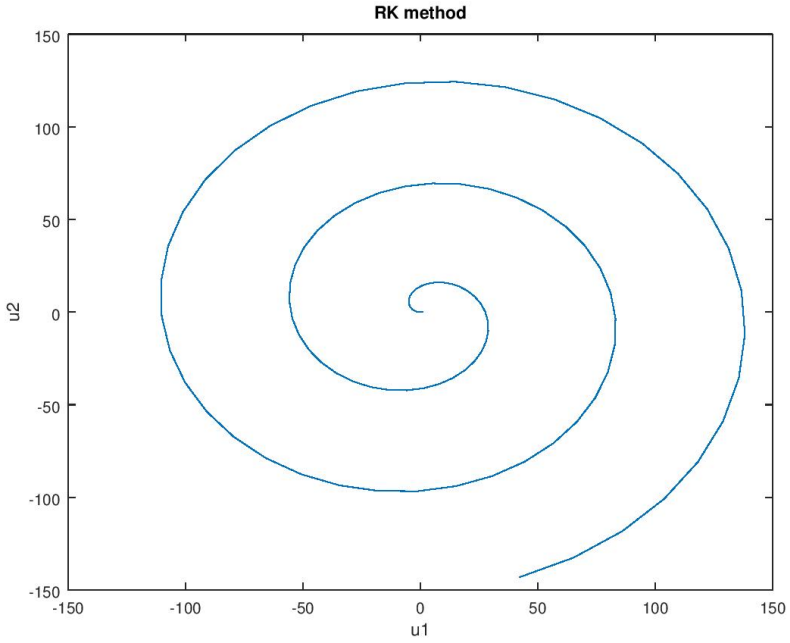
```
for i = 2 : (l/4)
u1_h2(i) = u_h2(4*(i-1));
endfor
u2_h2(1) = u_h2(2);
for i = 2 : (l/4)
u2_h2(i) = u_h2(4*(i-1) + 1);
endfor
figure(2); plot(u1_h2, u2_h2);
xlabel("u1");
ylabel("u2");
title("RK method");
print -djpg image3_4_2.jpg

[t, u_h3] = RKvett(@moto, tspan, u0, h3, G4, beta4);
l = length(u_h3);
u1_h3(1) = u_h3(1);
for i = 2 : (l/4)
u1_h3(i) = u_h3(4*(i-1));
endfor
u2_h3(1) = u_h3(2);
for i = 2 : (l/4)
u2_h3(i) = u_h3(4*(i-1) + 1);
endfor
figure(3); plot(u1_h3, u2_h3);
xlabel("u1");
ylabel("u2");
title("RK method");
print -djpg image3_4_3.jpg

[t, u_h4] = RKvett(@moto, tspan, u0, h4, G4, beta4);
l = length(u_h4);
u1_h4(1) = u_h4(1);
for i = 2 : (l/4)
u1_h4(i) = u_h4(4*(i-1));
endfor
u2_h4(1) = u_h4(2);
for i = 2 : (l/4)
u2_h4(i) = u_h4(4*(i-1) + 1);
endfor
figure(4); plot(u1_h4, u2_h4);
xlabel("u1");
ylabel("u2");
title("RK method");
print -djpg image3_4_4.jpg

function f = moto(t, u)
m1 = 0.012277471;
m2 = 1 - m1;
D1 = ((u(1) + m1).^2 + u(2).^2).^(3/2);
D2 = ((u(1) - m2).^2 + u(2).^2).^(3/2);
```

```
f = zeros(4, 1);
f(1) = u(3);
f(2) = u(4);
f(3) = u(1) + 2 * u(4) - (m2 * ((u(1) + m1) ./ D1)) -...
(m1 * ((u(1) - m2) ./ D2));
f(4) = u(2) - 2 * u(3) - (m2 * (u(2) ./ D1)) -...
(m1 * (u(2) ./ D2));
```

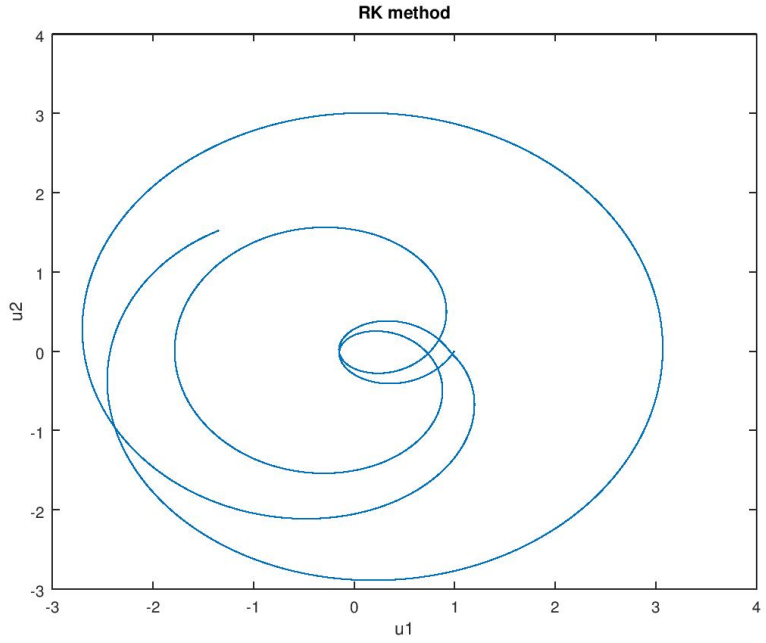
Qui riporto la versione vettroriale del Runge-Kutta esplicito:

```
function [t, u] = RKvett(odefun, tspan, u0, h, G, beta)
%beta = beta(:);
m = size(G, 1);
rho = G * ones(m,1);
t = [tspan(1) : h : tspan(2)]';
n = length(u0);
u = zeros(n * length(t), 1);
u(1:n) = u0;
%reshape(k, [n m]) riscrive il vettore k di lunghezza n*m...
%in una matrice di n righe e m colonne
for i = 1 : (length(t) - 1)
k = zeros(n * m, 1);
for j = 1 : m
k(((j - 1) * n + 1) : (j * n)) = feval(odefun,...
t(i) + rho(j) * h, u(((i-1) * n + 1) : (i * n)) +...
h * (G(j, :) * reshape(k, [n m])')');
endfor
u((i * n + 1) : ((i + 1) * n)) = u(((i - 1) * n + 1) :...
(i * n)) + h * (beta * reshape(k, [n m])')';
endfor
endfunction
```

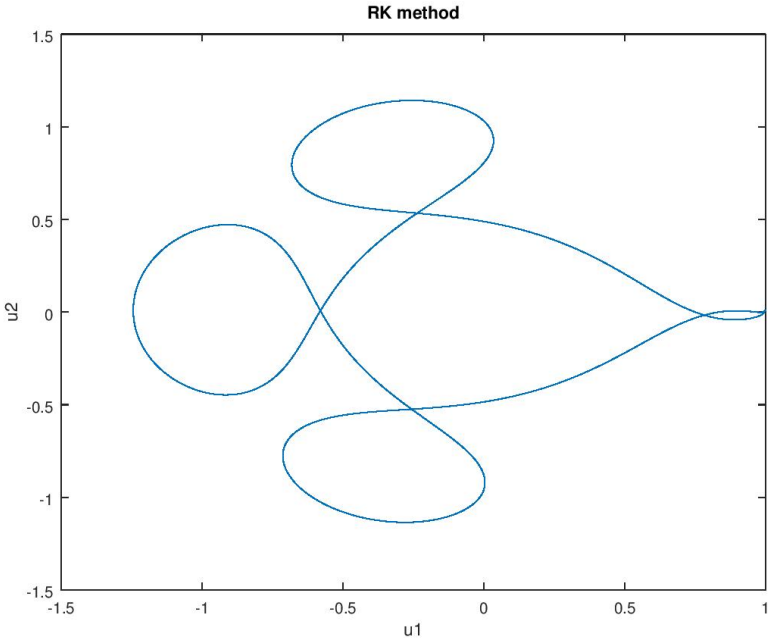
Sotto i grafici della traettoria del corpo: usando 10^2 passi di integrazione:



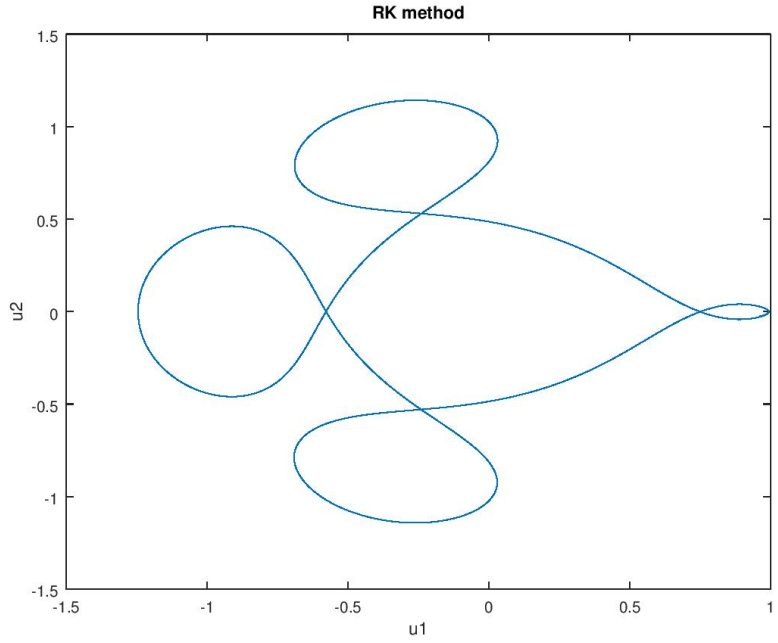
usando 10^3 passi di integrazione:



usando 10^4 passi di integrazione:



usando $2 * 10^4$ passi di integrazione:



Esercitazione 4
Metodi di Runge-Kutta espliciti implementati in Matlab

1. Uso i codici **ode23** e **ode45** per risolvere i problemi a valori iniziali della ESERCITAZIONE 3.

L'esercizio 2 dell' esercitazione 3 usando **ode23** con tolleranza dell' errore assoluto **AbsTol** $= 10^{-7}$ e tolleranza dell' errore relativo **RelTol**$= 10^{-4}$:

```
tspan = [0 1];
x0 = 1/3;
opzioni = odeset('AbsTol', 1e-7, 'RelTol', 1e-4, 'Stats', 'on');
S = ode23(@funz6, tspan, x0, opzioni);
t = S.x;
x = S.y;
S.stats
```

L' intervallo discretizzato:
$[0.00000, 0.01000, 0.02500, 0.04750, 0.08125, 0.13188, 0.20781, 0.30294, 0.40294, 0.50294, 0.60294, 0.70294, 0.80294, 0.90294, 1.00000]$
Le statistiche:
nsteps = 14
nfailed = 0
nfevals = 43
npds = 0
ndecomps = 0
nlinsols = 0

L'esercizio 2 dell' esercitazione 3 usando **ode45**:

```
tspan = [0 1];
x0 = 1/3;
opzioni = odeset('AbsTol', 1e-7, 'RelTol', 1e-4, 'Stats', 'on');
S = ode45(@funz6, tspan, x0, opzioni);
t = S.x;
x = S.y;
S.stats
```

L' intervallo discretizzato:
$[0.00000, 0.04642, 0.11604, 0.21604, 0.31604, 0.41604, 0.51604, 0.61604, 0.71604, 0.81604, 0.91604, 1.00000]$
Le statistiche:
nsteps = 11
nfailed = 0
nfevals = 67
npds = 0
ndecomps = 0
nlinsols = 0

L'esercizio 3 dell' esercitazione 3 usando **ode23**:

```
tspan = [0 5];
x0 = 2;
opzioni = odeset('AbsTol', 1e-7, 'RelTol', 1e-4, 'Stats', 'on');
S = ode23(@funz7, tspan, x0, opzioni);
t = S.x;
x = S.y;
S.stats
```

L' intervallo discretizzato:
[0.00000, 0.00086, 0.00156, 0.00221, 0.00286, 0.00350, 0.00414, 0.00478, 0.00542, 0.00606, 0.00670, 0.00734, 0.00798, 0.00861, 0.00925, 0.00989, 0.01053, 0.01116, ... 4.79673, 4.80841, 4.82159, 4.83327, 4.84644, 4.85812, 4.87129, 4.88297, 4.89614, 4.90781, 4.92098, 4.93266, 4.94583, 4.95751, 4.97067, 4.98235, 4.99551, 5.00000]
Le statistiche:
nsteps = 460
nfailed = 20
nfevals = 1441
npds = 0
ndecomps = 0
nlinsols = 0

L'esercizio 3 dell' esercitazione 3 usando **ode45**:

```
tspan = [0 5];
x0 = 2;
opzioni = odeset('AbsTol', 1e-7, 'RelTol', 1e-4, 'Stats', 'on');
S = ode23(@funz7, tspan, x0, opzioni);
t = S.x;
x = S.y;
S.stats
```

L' intervallo discretizzato:
[0.00000, 0.00268, 0.00529, 0.00790, 0.01051, 0.01311, 0.01570, 0.01828, 0.02083, 0.02330, 0.02560, 0.02784, 0.03050, 0.03360, 0.03718, 0.04133, 0.04621, 0.05205, ... 4.74444, 4.75964, 4.77484, 4.79005, 4.80525, 4.82045, 4.83565, 4.85085, 4.86605, 4.88124, 4.89644, 4.91164, 4.92684, 4.94204, 4.95723, 4.97243, 4.98763, 5.00000]
Le statistiche:
nsteps = 343
nfailed = 14
nfevals = 2143
npds = 0
ndecomps = 0
nlinsols = 0

2. Sia

$$f(t, x(t)) = \lambda(x(t) - sin(t)) + cos(t), \quad t \geq 0$$

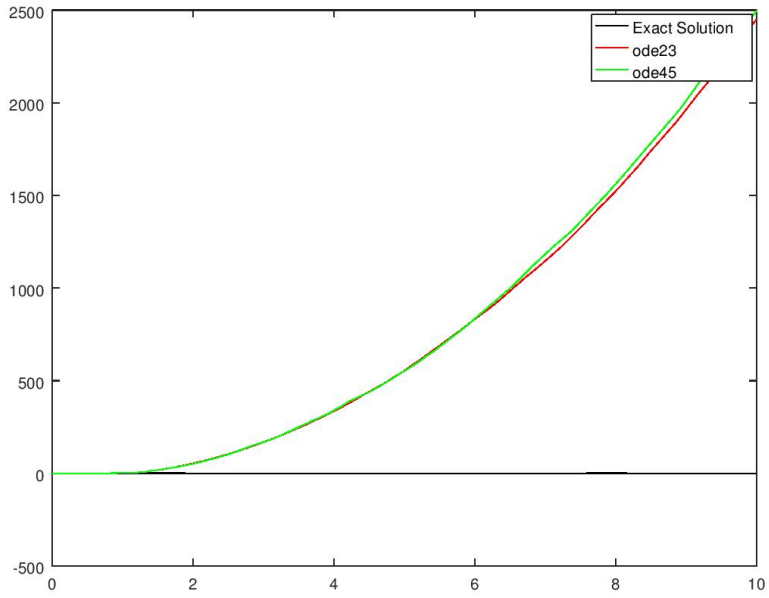
la funzione che definisce una equazione differenziale su cui i metodi **odeXY** falliscono in quanto l' errore globale molto piú grande dell' errore locale. La *function* **funz2_4.m** implementa tale funzione:

```
function f = funz2_4(x, t, lambda)
%In OCTAVE odeXY vuole f definita in (x, t) \in R^n x R
f = lambda * (x - sin(t)) + cos(t);
endfunction
```

Determino sull' intervallo $[0, 10]$ con i codici **odeXY** la soluzione approssimata dell' IVP con $x(0) = 0$:

```
opzioni = odeset('Stats', 'on');
tspan = [0 10];
x0 = 0;
lambda = 50;
S = ode45(@funz2_4, tspan, x0, opzioni, lambda);
t = S.x;
sol = S.y;
S.stats
x_ex = sin(t);
S.stats
plot(t, x_ex, "k;Exact Solution;", t, sol, "r;ode23;");
print -djpg image4_2.jpg
```

Sullo stesso grafico vengono riportate sia la soluzione cosí sia la soluzione esatta $x(t) = sin(t)$:



Dalle statistiche si evince che per il risolutore l' approssimazione fornita è buona.

Statistiche di **ode23**:
nsteps = 284
nfailed = 45
nfevals = 988
npds = 0
ndecomps = 0
nlinsols = 0

Statistiche di **ode45**:
nsteps = 145
nfailed = 37
nfevals = 1093
npds = 0
ndecomps = 0
nlinsols = 0

3. Si consideri il modello di una volpe che insegue un coniglio. Il coniglio segue una traettoria nel piano

$$r(t) = \begin{pmatrix} r_1(t) \\ r_2(t) \end{pmatrix} = \sqrt{1+t} \begin{pmatrix} cos(t) \\ sin(t) \end{pmatrix},$$

mentre la volpe insegue il coniglio (con una velocità k volte quella del coniglio) seguendo un traettoria $\mathbf{y}(t) = (\ y_1(t),\ y_2(t)\ )$ definita dalla seguente equazione differenziale:

$$\begin{cases} y_1'(t) = s(t)(r_1(t) - y_1(t)) \\ y_2'(t) = s(t)(r_2(t) - y_2(t)) \end{cases},$$

dove

$$s(t) = \frac{k\|\mathbf{r}'(t)\|_2}{\|\mathbf{r}(t) - \mathbf{y}(t)\|_2}.$$

Il fle di tipo *function* **fox1.m** implementa la funzione che definisce il problema differenziale.

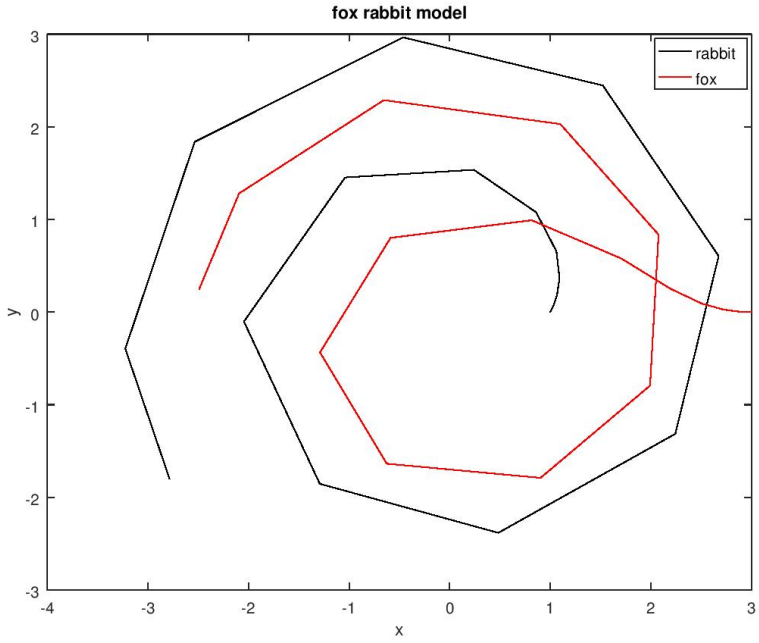
```
function f = fox1(t, y, k)
r = zeros(2,1);
r = (1 + t)^(1/2) * [cos(t); sin(t)];
%r_p derivata di r
r_p = zeros(2,1);
r_p = (1 / (2 * (t+1)^(1/2))) * [cos(t) - 2*(t+1)*sin(t); sin(t) +...
2*(t+1)*cos(t)];
dist = norm(r - y);
%if(dist > 1e-4)
s = k * norm(r_p) / dist;
f = s * (r - y);
%else
```

```
% printf("error\n");
%endif
endfunction
```

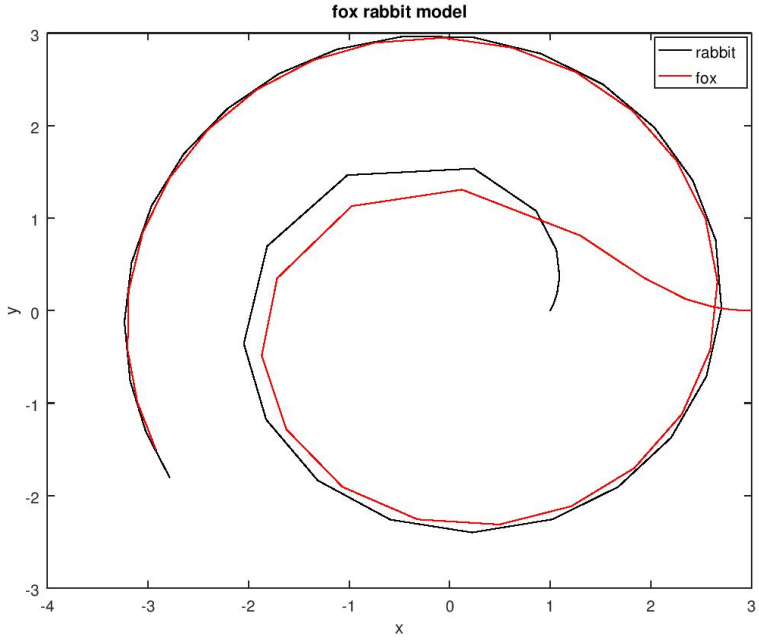
I file di tipo *function* **fox_rabbit1.m** integra usando **ode45** il problema con condizione iniziale $\mathbf{y}(t) = (3, 0)^T$, sull' intervallo [0, 10], e disegna sullo stesso grafico le due traettorie.

```
function [t_fox, y_fox] = fox_rabbit1(k)
y0 = [3; 0];
tspan =[0 10];
S = ode45(@fox1, tspan, y0, k);
t = S.x;
r = zeros(2,length(t));
r(1,:) = (1 .+ t).^(1/2) .* cos(t);
r(2,:) = (1 .+ t).^(1/2) .* sin(t);
y = S.y;
%i = 1;
%while(i <= length(t) & norm(y(:,i) - r(:,i) > 1e-4)
% i++;
%endwhile
%if(i < length(t))
% error
%else
plot(r(1,:), r(2,:), "k;rabbit", y(1,:), y(2,:), "r;fox");
xlabel("x");
ylabel("y");
title("fox rabbit model");
%endif
endfunction
%endif
endfunction
```

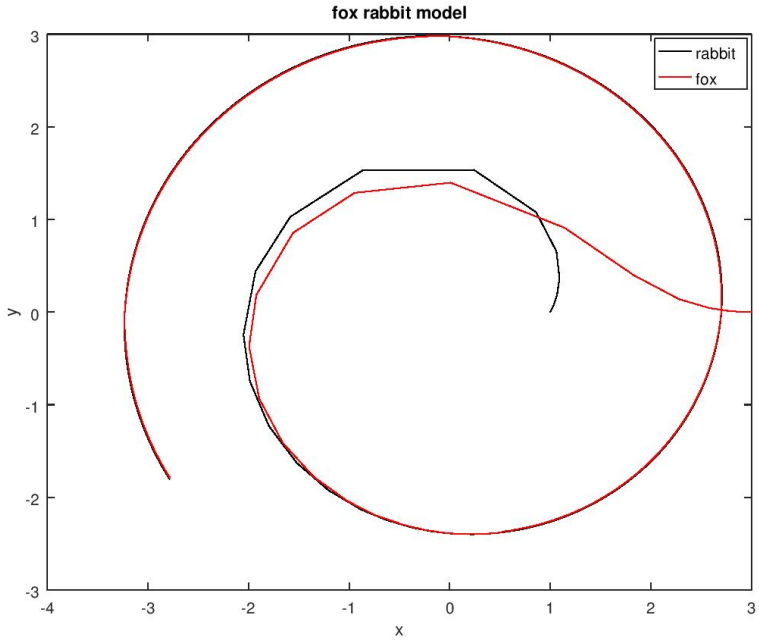
Qui sotto i grafici delle traettorie della volpe e del coniglio:
Con $k = 0.75$



Con $k = 1$



Con $k = 1.1$

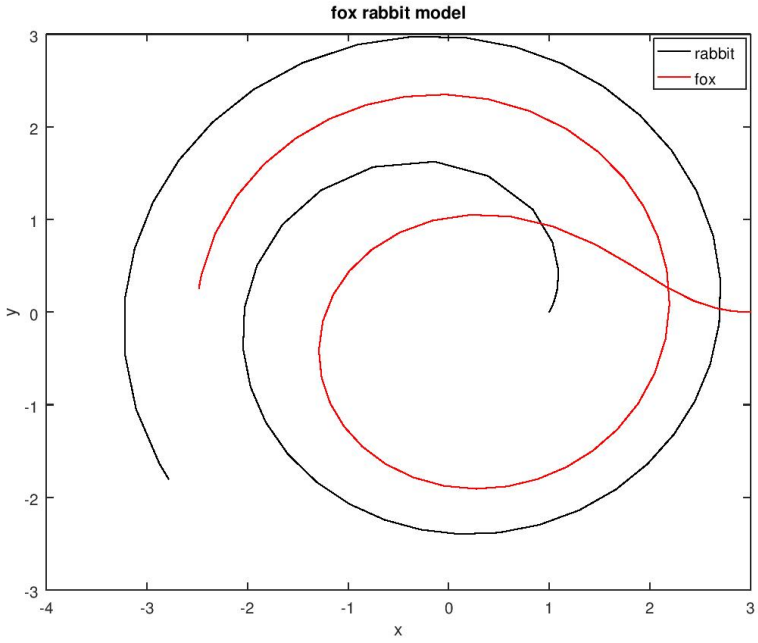


Modifico ora i file costruiti modificando le opzioni, e caricando in queste la nuova function events:

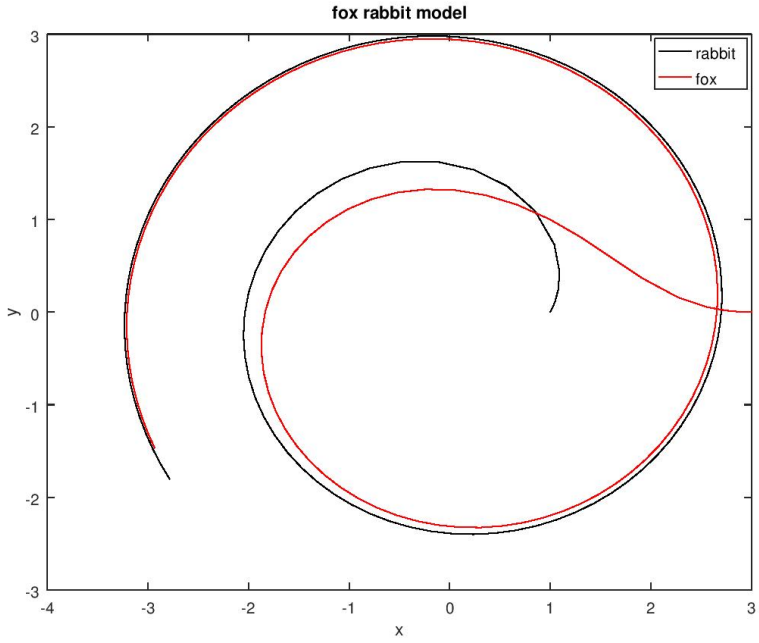
```
function [t_fox, y_fox] = fox_rabbit2(k)

y0 = [3; 0];
tspan =[0 10];
options = odeset('RelTol', 1e-6, 'AbsTol', 1e-6, 'Events', @events);
S = ode45(@fox1, tspan, y0, options, k);
t = S.x;
r = zeros(2,length(t));
r(1,:) = (1 .+ t).^(1/2) .* cos(t);
r(2,:) = (1 .+ t).^(1/2) .* sin(t);
y = S.y;

%i = 1;
%while(i <= length(t) & norm(y(:,i) - r(:,i) > 1e-4)
% i++;
%endwhile
%if(i < length(t))
% error
%else
plot(r(1,:), r(2,:), "k;rabbit;", y(1,:), y(2,:), "r;fox;");
xlabel("x");
ylabel("y");
title("fox rabbit model");
%endif
print -djpg fox_rabbit2_3.jpg
```

```
function [value, isterminal, direction] = events(t,y,k)
r = sqrt(1+t)*[cos(t); sin(t)];
value = norm(r - y) - 1e-4;
isterminal = 1;
direction = -1;
```

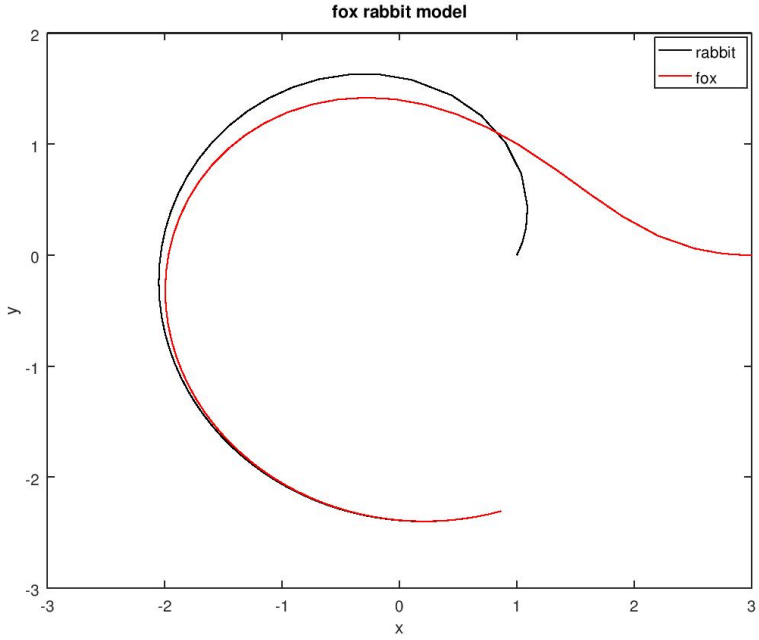
Qui sotto i grafici delle traettorie della volpe e del coniglio usando la nuova *function*:
Con $k = 0.75$



Con $k = 1$


fox rabbit model
rabbit
fox
x
y

Con $k = 1.1$


fox rabbit model
rabbit
fox
x
y

Esercitazione 5
Metodi di Adams-Bashforth

1. Dato l' IVP generico

$$\begin{cases} x'(t) = f(t, x(t)), \quad t \in [t_0, T] \\ x(t_0) = x_0 \end{cases}$$

con $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$,
il file di tipo function implementa su griglia uniforme il **metodo di Adams-Bashforth a k passi**

```
function [t, x] = adamsbash(odefun, tspan, x0, xinit, h, k)
AB = zeros(4, 4);
AB(1,1) = 1;
AB(2,1:2) = [3/2 -1/2];
AB(3, 1:3) = [23/12 -16/12 5/12];
AB(4, 1:4) = [55/24 -59/24 37/24 -9/24];
t = tspan(1) : h : tspan(2);
n = length(t);
xinit = xinit(:);
f = odefun(t(1 : k)', [x0; xinit]);
f = f(:);
x = [x0 xinit' zeros(1, n - k)];
%flipud return a copy of array with the order of the rows reversed
for i = 1 : n-k
x(k+i) = x(k+i-1) + h * AB(k, 1 : k) * flipud(f);
f = [f(2 : end); odefun(t(k+i), x(k+i))];
endfor
endfunction
```

2. Dato l' IVP

$$\begin{cases} x'(t) = -5tx^2(t) + \frac{5}{t} - \frac{1}{t^2}, \quad t \in (1, 25] \\ x(1) = 1, \end{cases}$$

la cui soluzione esatta è

$$x(t) = 1/t$$

Il file di tipo *script* **es5_2.m**, calcolati i valori iniziali con un metodo di Runge-Kutta a k-passi, deriva la soluzione approssimata ottenuta con il metodo di Adams-Bashfoth a k-passi, con k parametro fissato con l' impostazione **input**. (**funz5_2.m** implementa $f(t, x(t))$ di tale problema.)
Fissato $h = 0.2$ e dimezzando il passo per cinque volte, calcolo la norma infinito dell' errore momorizzando i risultati, al variare di h e k, nella successiva tabella.

```
k = input("Scegli il passo\n");
if(k == 1)
G = [0];
beta = 1;
```

```
elseif(k == 2)
G = [0 0; 1 0];
beta = [1/2 1/2];
elseif(k == 3)
G = [0 0 0; 1/2 0 0; -1 2 0];
beta = [1/6 2/3 1/6];
elseif(k == 4)
G = [0 0 0 0; 1/2 0 0 0; 0 1/2 0 0; 0 0 1 0];
beta = [1/6 1/3 1/3 1/6];
endif

tspan = [1, 25];
x0 = 1;
h = 0.2;
[s, xinit] = RK(@funz, tspan, x0, h, G, beta);
xinit = xinit(:);
xinit = xinit(2:k);
[t, x] = adamsbash(@funz, tspan, x0, xinit, h, k);
x_ex = t.^(-1);
e(1) = norm(x_ex - x, inf);

h = 0.1;
[s, xinit] = RK(@funz, tspan, x0, h, G, beta);
xinit = xinit(:);
xinit = xinit(2:k);
[t, x] = adamsbash(@funz, tspan, x0, xinit, h, k);
x_ex = t.^(-1);
e(2) = norm(x_ex - x, inf);

h = 0.05;
[s, xinit] = RK(@funz, tspan, x0, h, G, beta);
xinit = xinit(:);
xinit = xinit(2:k);
[t, x] = adamsbash(@funz, tspan, x0, xinit, h, k);
x_ex = t.^(-1);
e(3) = norm(x_ex - x, inf);

h = 0.025;
[s, xinit] = RK(@funz, tspan, x0, h, G, beta);
xinit = xinit(:);
xinit = xinit(2:k);
[t, x] = adamsbash(@funz, tspan, x0, xinit, h, k);
x_ex = t.^(-1);
e(4) = norm(x_ex - x, inf);

h = 0.0125;
[s, xinit] = RK(@funz, tspan, x0, h, G, beta);
xinit = xinit(:);
xinit = xinit(2:k);
[t, x] = adamsbash(@funz, tspan, x0, xinit, h, k);
```

```
x_ex = t.^(-1);
e(5) = norm(x_ex - x, inf);

h = 0.00625;
[s, xinit] = RK(@funz, tspan, x0, h, G, beta);
xinit = xinit(:);
xinit = xinit(2:k);
[t, x] = adamsbash(@funz, tspan, x0, xinit, h, k);
x_ex = t.^(-1);
e(6) = norm(x_ex - x, inf);
e

function f = funz(t, x)
f = - (5 * t.* (x.^2)) + (5 * t.^(-1)) - (t.^(-2));
```

I dati sono forniti nella tabella sottostante.

	k = 1	k = 2	k = 3	k = 4
h = 0.2	1.7580	Inf	Inf	Inf
h = 0.1	1.7257	4.1095e-03	7.6670e-01	Inf
h = 0.05	1.7082	5.3780e-04	7.7351e-05	5.1995e-01
h = 0.025	1.7001	9.8104e-05	8.1423e-06	7.6434e-07
h = 0.0125	1.6960	2.3038e-05	9.4826e-07	4.7965e-08
h = 0.00626	1.6940	5.5773e-06	1.1505e-07	3.0104e-09

Man mano che il passo si dimezza le soluzione diventano piú accurate. I valori sono piú accurati per k piú grandi. Per h grande ($h = 0.2$ e $h = 0.1$) i metodi a piú stadi restituiscono soluzioni numeriche lontane da quella esatta.

Esercitazione 6
Problemi stiff

1. Sia dato il problema a valori iniziali

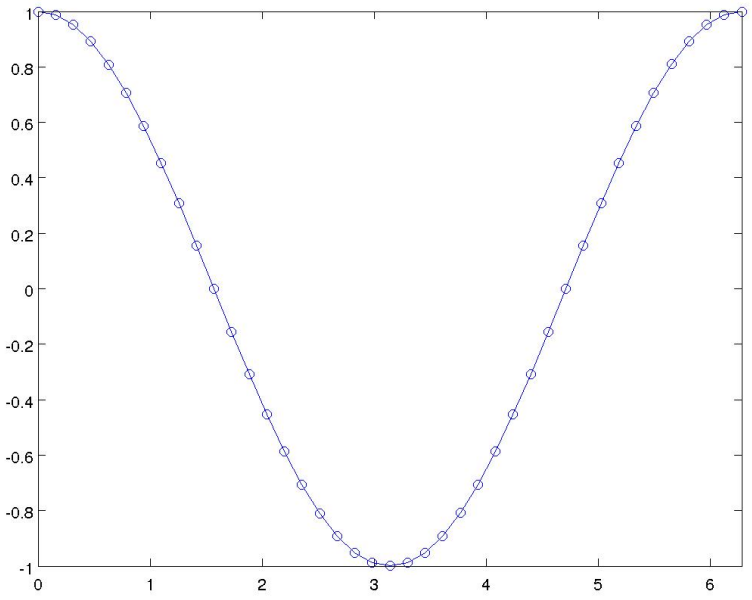
$$\begin{cases} x'(t) = \lambda(x(t) - cos(t)) - sin(t), & t \in (1, 2\pi] \\ x(0) = x_0, \end{cases}$$

la cui soluzione esatta é:

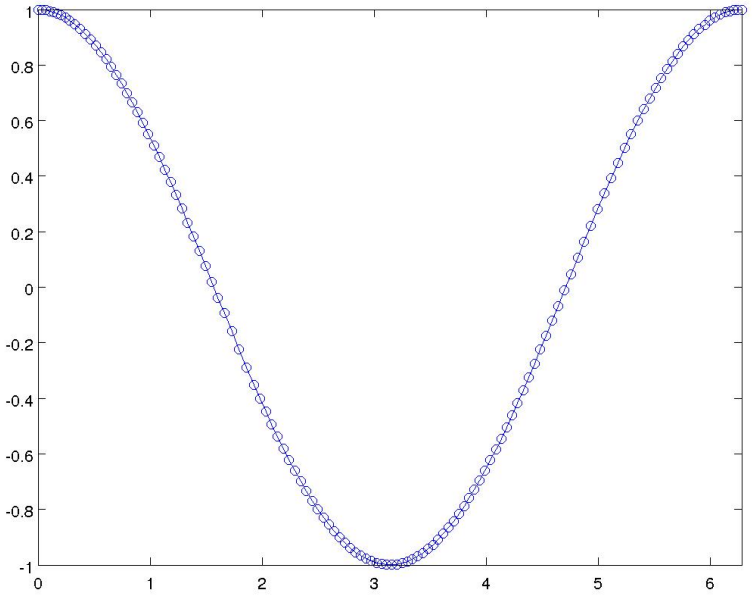
$$x(t) = (x_0 - 1)e^{\lambda t} + cos(t)$$

Uso le function **ode45_test.m** e **ode15s_test.m** per la sua risoluzione numerica che utilizzano le routine **ode45** e **ode15s**, rispettivamente. Entrambe richiedono in input i seguenti parametri: $\lambda, x_0, rtol, atol$ (tolleranze per l' errore relativo ed assoluto rispettivamente).

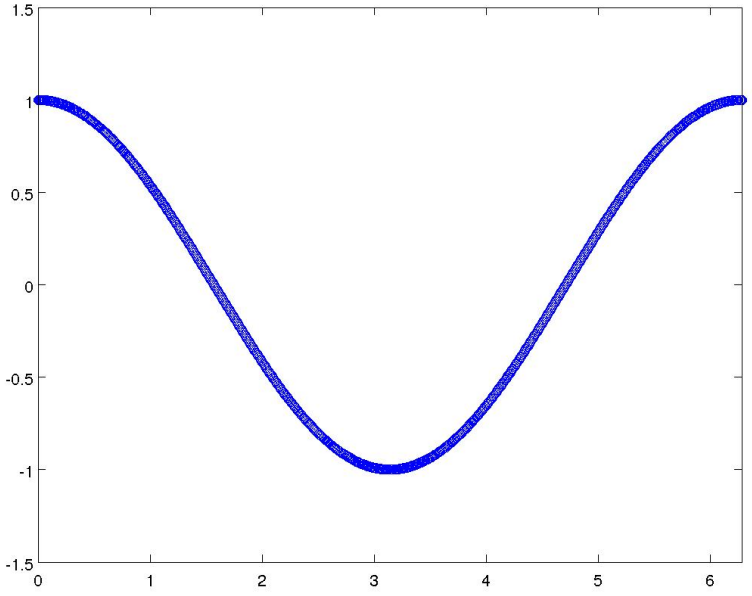
1. Fisso $x_0 = 1, rtol = atol = 1e-3$.
   Applico la function **ode45_test.m** con $\lambda = -1$:



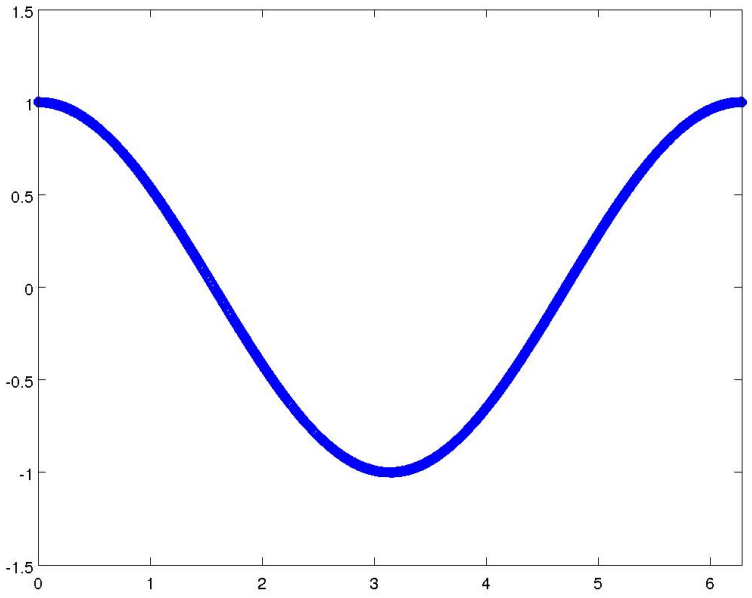
   con $\lambda = -10$:



con $\lambda = -100$:

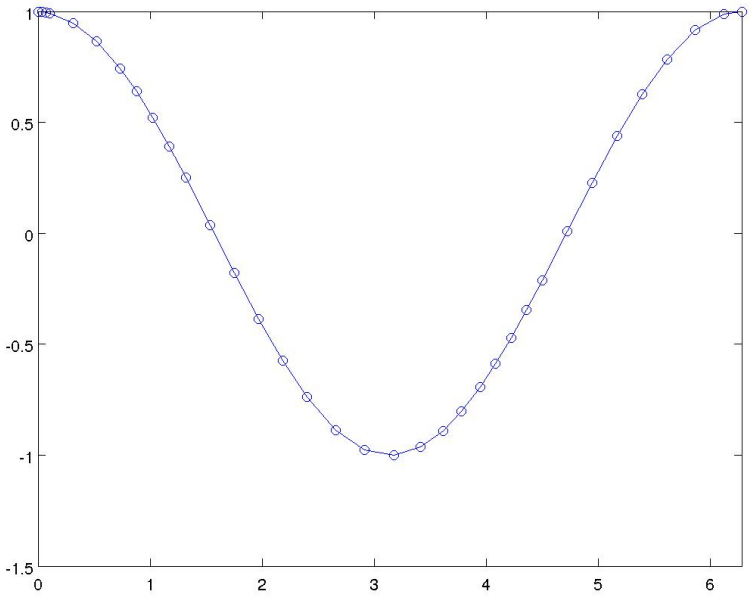


con $\lambda = -1000$:

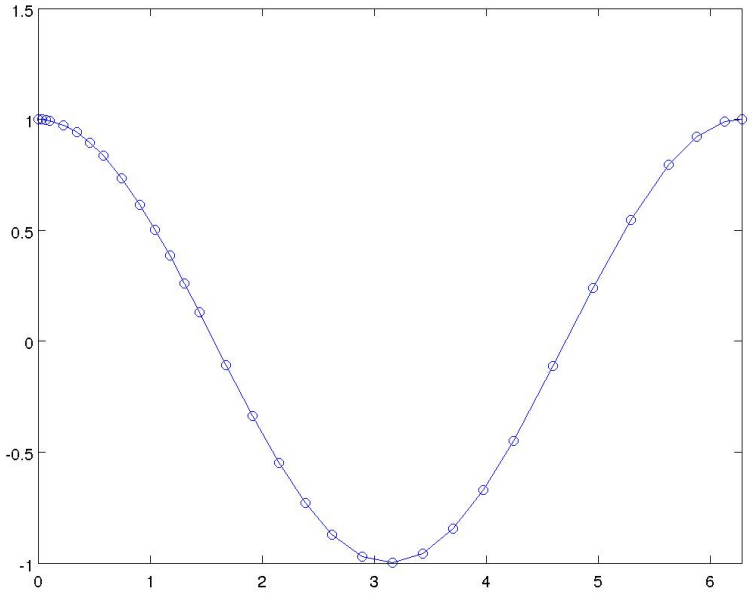


In tal caso la soluzione esatta non dipende da lambda, ma il numero di passi di integrazione cresce all' aumentare di $|\lambda|$.

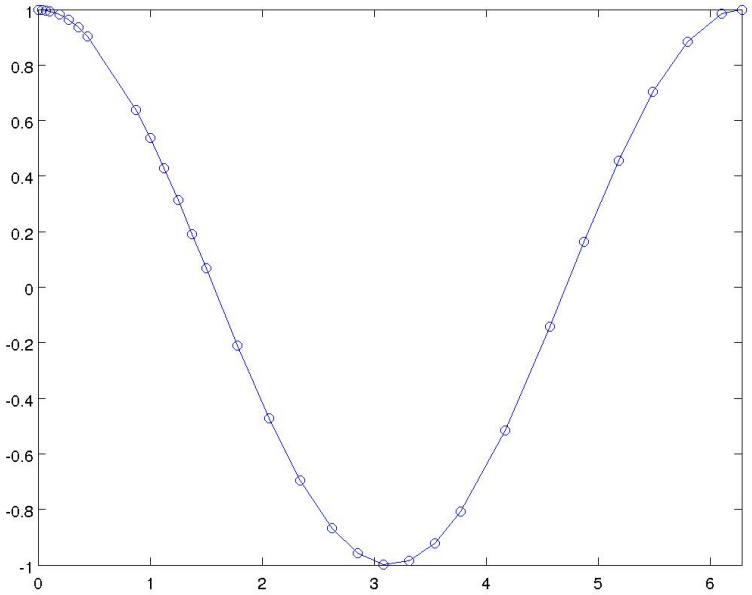
2. Con i stessi valori di sopra per $x_0, rtol$ e $atol$ utilizzo ora la function **ode15s_test.m** con $\lambda = -1$:



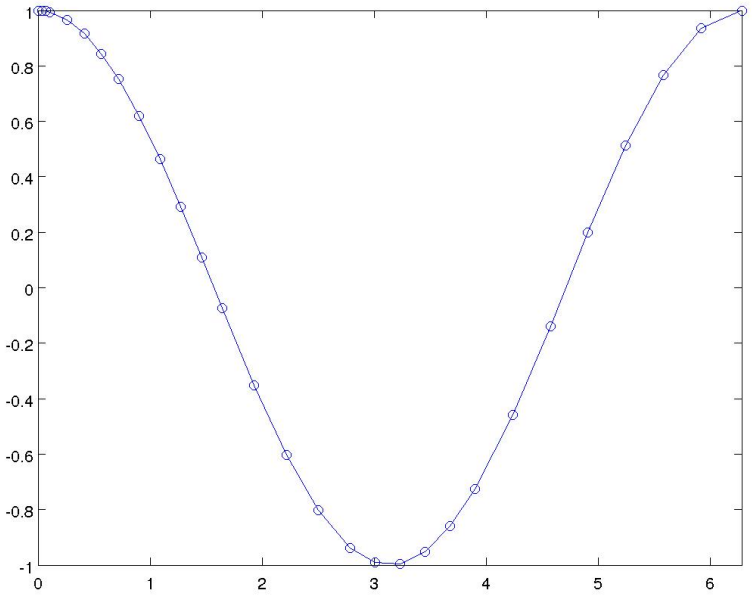
Con $\lambda = -10$:



Con $\lambda = -100$:

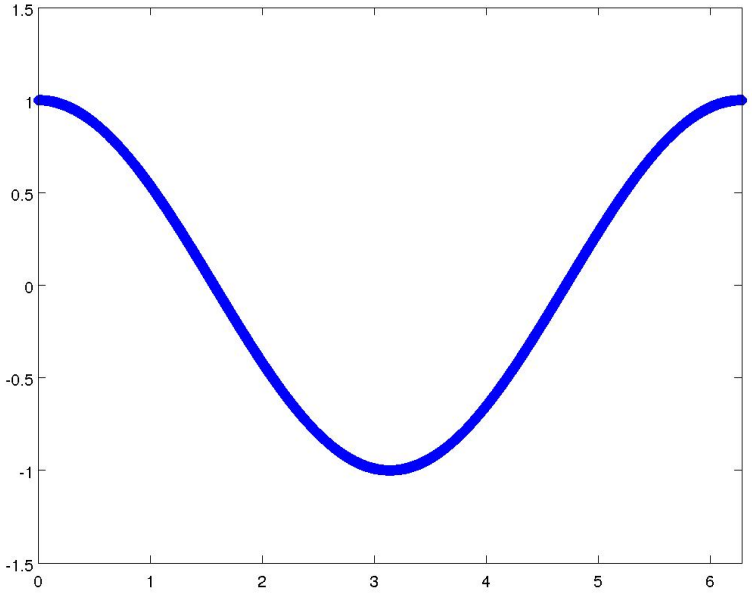


Con $\lambda = -1000$:



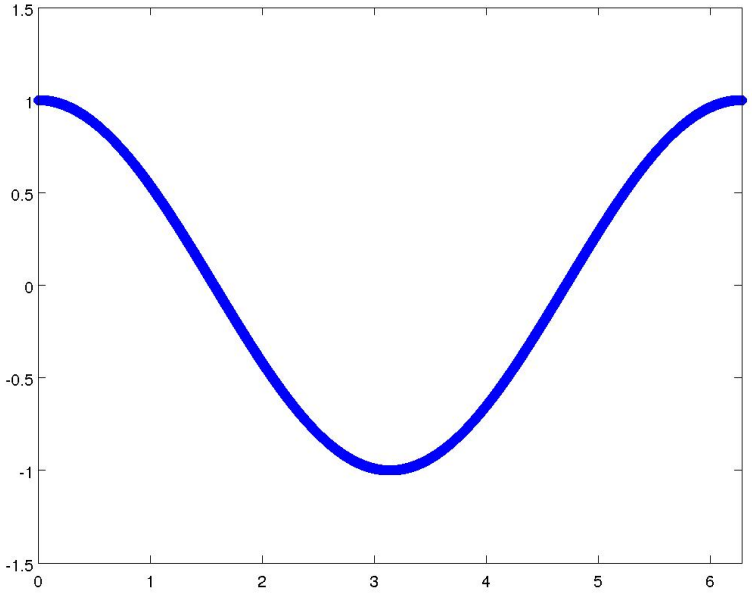
Le statistiche dei metodi con $RTol = ATol = 10^3$ e con $RTol = ATol = 10^{-6}$ sono diverse (ad esempio il numero di passi di integrazione).

3. Fisso $x_0 = 1, \lambda - 10^4$, e applico **ode45_test.m** con $rtol = atol = 1e - 3$.



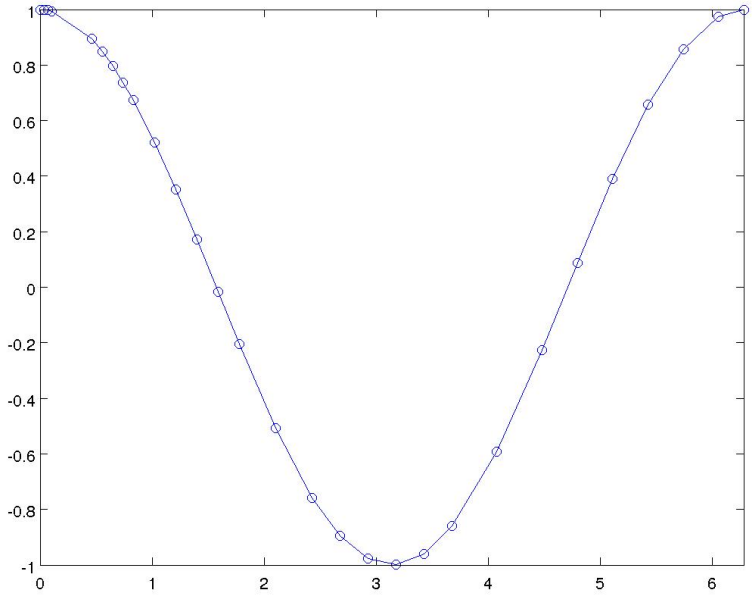
Le statistiche di integrazione ode45 stats:
18930 successful steps
1266 failed attempts
121177 function evaluations

Con $rtol = atol = 1e - 6$.



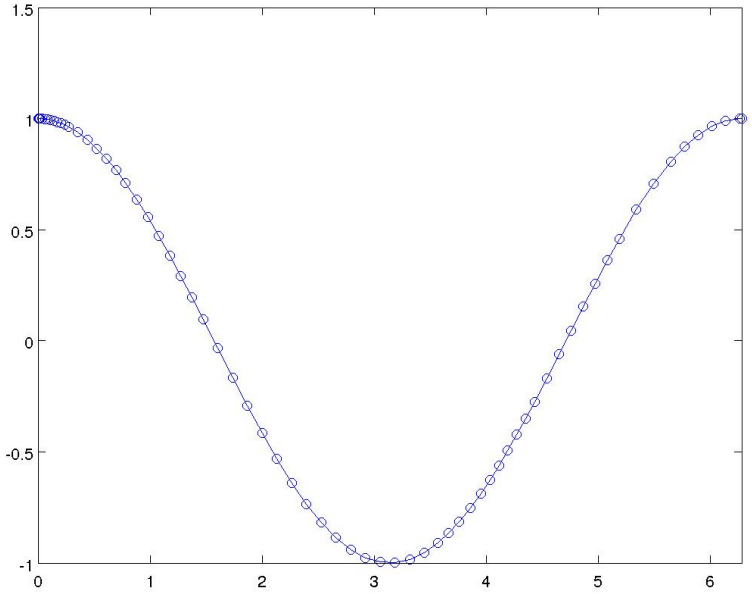
Le statistiche di integrazione ode45 stats:
18926 successful steps
1057 failed attempts
119899 function evaluations

Con i stessi valori di sopra per x_0 e λ applico **ode15s_test.m** con $rtol = atol = 1e-3$.



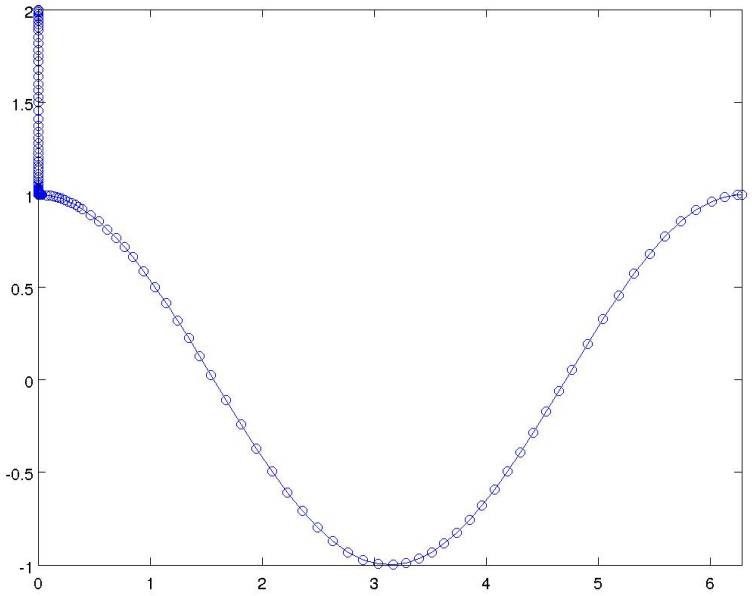
Le statistiche di integrazione
ode15s stats:
28 successful steps
4 failed attempts
66 function evaluations
0 partial derivatives
10 LU decompositions
64 solutions of linear systems

Ora con $rtol = atol = 1e-6$.



Le statistiche di integrazione
ode15s stats:
71 successful steps
4 failed attempts
152 function evaluations
0 partial derivatives
15 LU decompositions
150 solutions of linear systems
Le statistiche di integrazione sono simili.

4. Risolvo ora il problema con $x_0 = 2, \lambda = -10^4$ e $rtol = atol = 1e-6$ applicando **ode15s_test.m**



Si utilizzano passi di integrazione "piccoli" soltanto durante la fase transiente.

2. Il seguente IVP

$$\begin{cases} x_1'(t) = -0.04x_1 + 10^4 x_2 x_3 \\ x_2'(t) = 0.04x_1 - 10^4 x_2 x_3 - 3 \cdot 10^7 x_2 \quad t \in (0, 10^9] \\ x_3'(t) = 3 \cdot 10^7 x_2 \\ x_1(0) = 1, \quad x_2(0) = 0, \quad x_3(0) = 0, \end{cases}$$

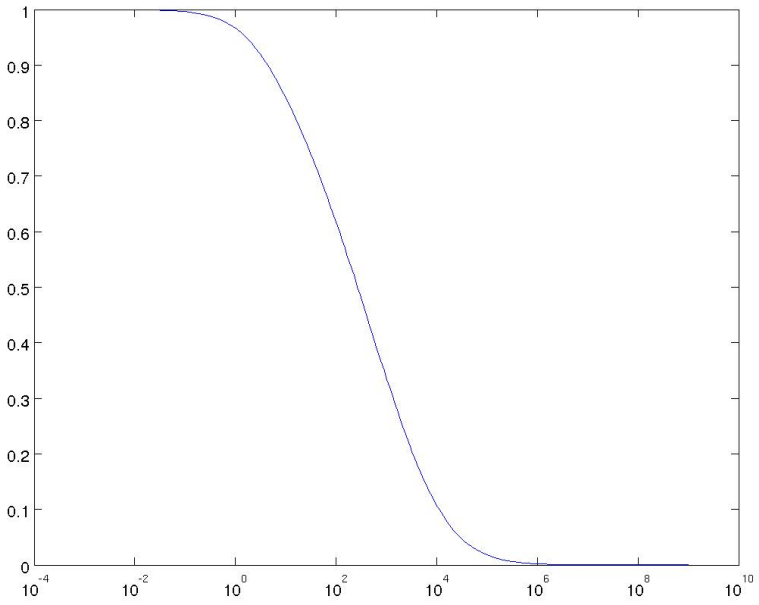
modellizza un sistema di reazioni chimiche tra tre sostanze. Le tre componenti della soluzione rappresentano la concentrazione di ciascuna di esse. Pertanto, esse hanno significato fisico solamente se non negative. La stiffness del problema dovuta alle diverse velocità con cui avvengono tali reazioni.

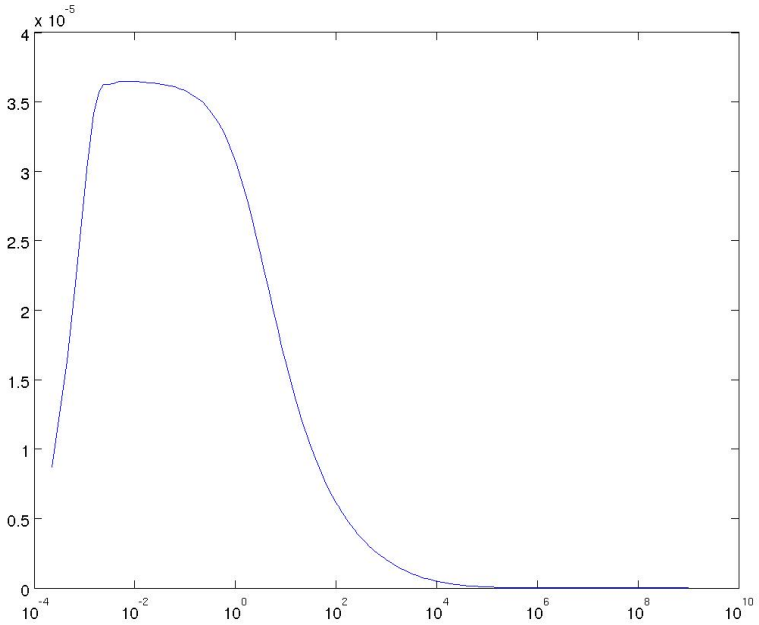
Applico la function **ode15s_rober.m**, che usa il codice **ode15s**, per risolvere tale problema ai valori iniziali specificando in input **rtol=1e-3** e **atol=1e-6**.

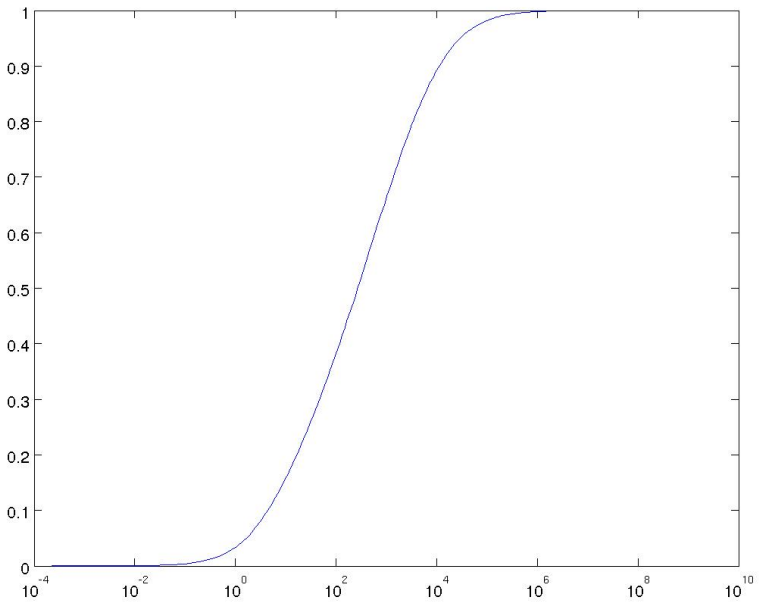
```
function [t,x] = ode15s_rober(rtol, atol)
x0 = [1
    0
    0];
tspan = [0, 1e9];
opzioni = odeset('RelTol',rtol,'AbsTol',atol,'Stats','on');
%opzioni = odeset(opzioni,'BDF','on');
```

```
disp('ode15s stats:')
[t,x] = ode15s(@fun,tspan,x0,opzioni);
x = x';

%grafici con scala logaritmica sull' asse delle ascisse
figure(1); semilogx(t, x(1,:)); print('es6_2_a1', '-djpeg')
figure(2); semilogx(t, x(2,:)); print('es6_2_a2', '-djpeg')
figure(3); semilogx(t, x(3,:)); print('es6_2_a3', '-djpeg')

function f = fun(t, x)
f = [-0.04*x(1) + 1e4*x(2)*x(3)
    0.04*x(1) - 1e4*x(2)*x(3) - 3*1e7*(x(2)^2)
    3*1e7*(x(2)^2)];
```

Utilizzando la scala logaritmica sull' asse delle ascisse, traccio, su tre figure distinte, il grafico di ciascuna componente della soluzione numerica calcolata:





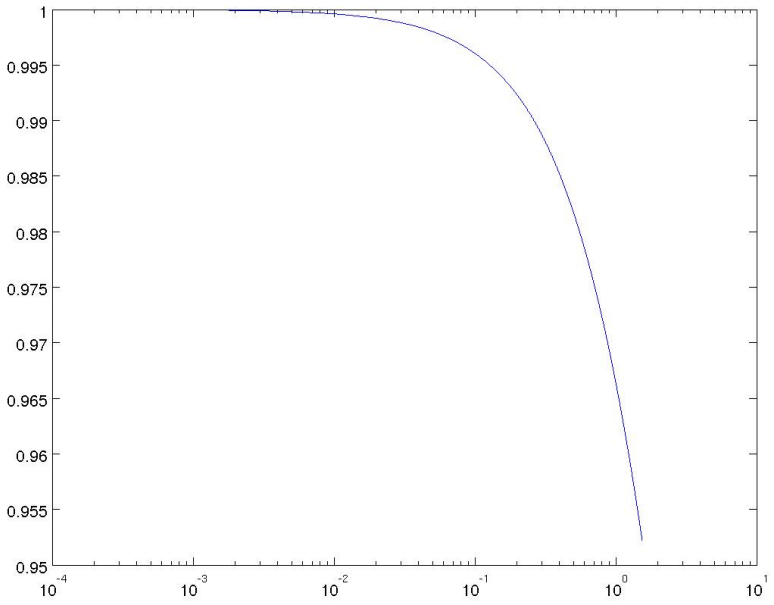


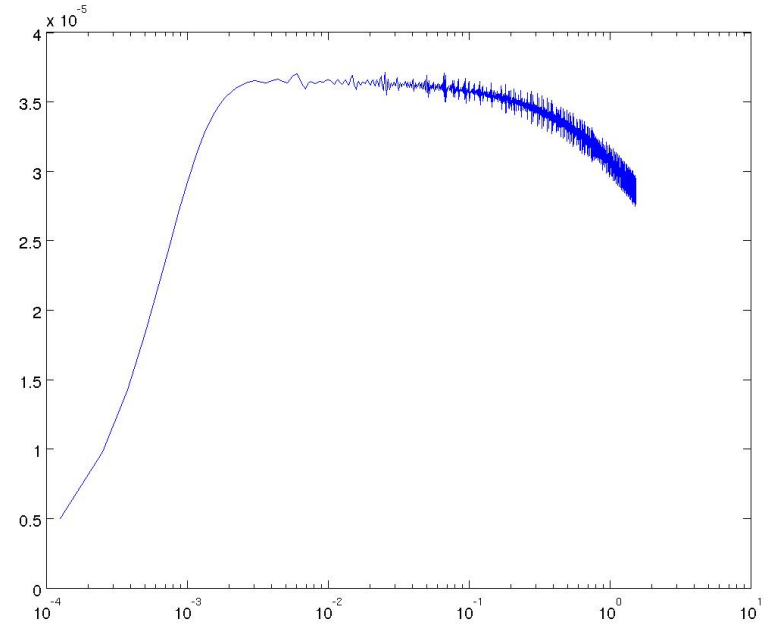
Ora, invece, applico la function **ode45_rober.m**, che usa il codice **ode45**, per risolvere tale problema ai valori iniziali specificando in input **rtol=1e-3** e **atol=1e-6**.

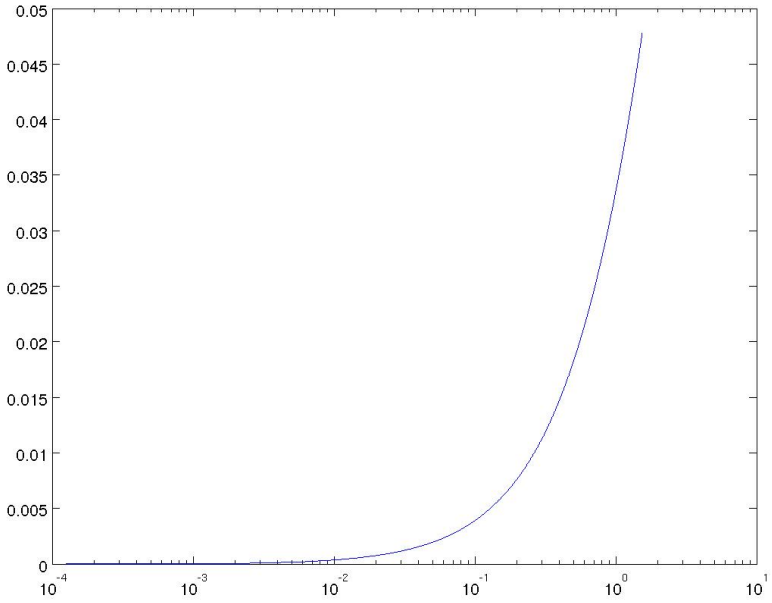
```
function [t,x] = ode45_rober(rtol, atol)
```

```
x0 = [1
    0
    0];
tspan = [0, 1e9];
opzioni = odeset('RelTol',rtol,'AbsTol',atol,'Stats','on');
%opzioni = odeset(opzioni,'BDF','on');
disp('ode45 stats:')
[t,x] = ode45(@fun, tspan, x0, opzioni);
x = x';

%grafici con scala logaritmica sull' asse delle ascisse
figure(1); semilogx(t, x(1,:)); print('es6_2_b1', '-djpeg')
figure(2); semilogx(t, x(2,:)); print('es6_2_b2', '-djpeg')
figure(3); semilogx(t, x(3,:)); print('es6_2_b3', '-djpeg')

function f = fun(t, x)
f = [-0.04*x(1) + 1e4*x(2)*x(3)
    0.04*x(1) - 1e4*x(2)*x(3) - 3*1e7*(x(2)^2)
    3*1e7*(x(2)^2)];
```

Utilizzando la scala logaritmica sull' asse delle ascisse, traccio, su tre figure distinte, il grafico di ciascuna componente della soluzione numerica calcolata:




x 10^-5
4
3.5
3
2.5
2
1.5
1
0.5
0
10^-4
10^-3
10^-2
10^-1
10^0
10^1


0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
10^-4
10^-3
10^-2
10^-1
10^0
10^1

Esercitazione 7
Metodi lineari a più passi

1. Siano dati i seguenti metodi lineari a 2-passi:

$$(a) \quad x_i = 4x_{i-1} - 3x_{i-2} - 2hf_{i-2}$$

$$(b) \quad x_i = x_{i-2} + 2hf_{i-1}$$

$$(c) \quad x_i = x_{i-1} - \frac{h}{2}(3f_{i-1} - f_{i-2})$$

$$(d) \quad x_i = \frac{4x_{i-1} - x_{i-2}}{3} - \frac{2h}{3}f_i.$$

Il file di tipo *function* **vari_lmm.m**, utilizzando tali metodi, calcoli su griglia uniforme la soluzione approssimata del problema a valori iniziali

$$\begin{cases} x'(t) = \lambda(x(t) - sin(t)) + cos(t), & t \in (0, 1] \\ x(0) = 1, \end{cases}$$

la cui soluzione esatta è

$$x(t) = e^{\lambda t} + sin(t).$$

```
function [t, x1, x2, x3, x4] = vari_lmm(h, lambda)
tspan = [0 1];
x0 = 1;
t = tspan(1) : h : tspan(2);
n = length(t);
xinit = exp(lambda*h) + sin(h);
f = funz7_1(t(1 : 2)', [x0; xinit], lambda);
f = f(:);
x_ex = exp(lambda*t) + sin(t)
%metodo a
x1 = [x0 xinit zeros(1, n - 2)];
alfa1 = [-3; 4];
beta1 = [0, -2];
%flipud return a copy of array with the order of the rows reversed
for i = 1 : n-2
x1(2+i) = x1(2+i-2 : 2+i-1) * alfa1  + h * beta1 * flipud(f);
f = [f(2 : end); funz7_1(t(2+i), x1(2+i), lambda)];
endfor
e(1) = norm(x_ex - x1, inf);
x1
%metodo b
x2 = [x0 xinit zeros(1, n - 2)];
alfa2 = [1; 0];
beta2 = [2, 0];
%flipud return a copy of array with the order of the rows reversed
for i = 1 : n-2
x2(2+i) = x2(2+i-2 : 2+i-1) * alfa2  + h * beta2 * flipud(f);
f = [f(2 : end); funz7_1(t(2+i), x2(2+i), lambda)];
```

```
endfor
e(2) = norm(x_ex - x2, inf);
%metodo c
x3 = [x0 xinit zeros(1, n - 2)];
alfa3 = [0; 1];
beta3 = [3/2, -1/2];
%flipud return a copy of array with the order of the rows reversed
for i = 1 : n-2
x3(2+i) = x3(2+i-2 : 2+i-1) * alfa3  + h * beta3 * flipud(f);
f = [f(2 : end); funz7_1(t(2+i), x3(2+i), lambda)];
endfor
e(3) = norm(x_ex - x3, inf);
%metodo d
x4 = [x0 xinit zeros(1, n - 2)];
alfa4 = [-1/3; 4/3];
beta4 = 2/3;
for i = 1 : n-2
F = @(s) x4(2+i-2 : 2+i-1) * alfa4 - s  +...
h * beta4 * funz7_1(t(2+i), s, lambda);
x4(2+i) = fzero(F, x4(2+i-1));
endfor
e(4) = norm(x_ex - x4, inf);
e
roots([1; -flipud(alfa1)])
roots([1; -flipud(alfa2)])
roots([1; -flipud(alfa3)])
roots([1; -flipud(alfa4)])

plot(t, x1, ":r;metodo a;", t, x2, "--g;metodo b;",...
t, x3, "-.b;metodo c;", t, x4, "m;metodo d;");
xlabel("t");
ylabel("x(t)");
title("lmm methods");
print -djpg image7_1.jpg

endfunction
```

Le radici del primo polinomio caratteristico dei vari metodi sono:
metodo a: $\{3, 1\}$
metodo b: $\{-1, 1\}$
metodo c: $\{1, 0\}$
metodo d: $\{1, 0.\overline{3}\}$
I metodi a,b,c sono A-stabili, poiché hanno radici di modulo minore o uguale a 1 e quelle di modulo unitario sono semplici.
Per questi metodi stimo l' ordine di consistenza tramite la seguente *function*:

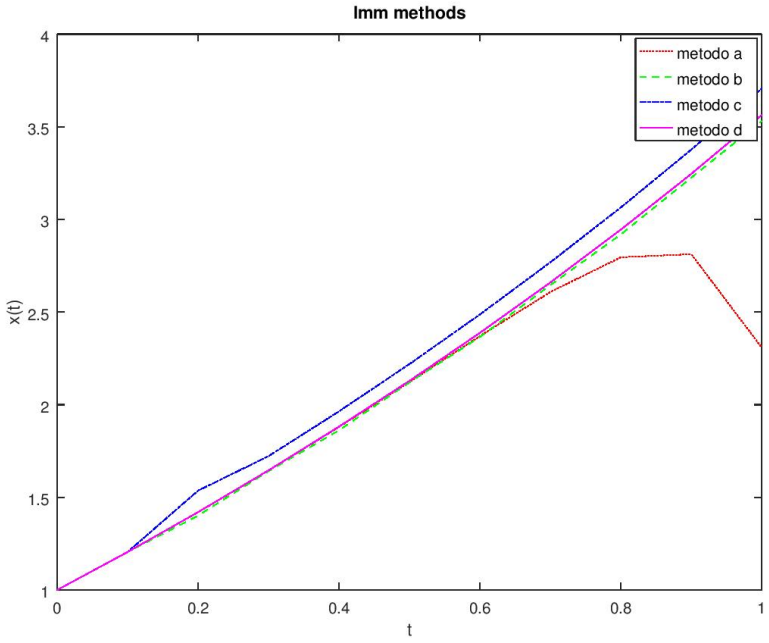
```
function P = ord_lmm(nstep)
tspan = [0 1];
x0 = 1;
x_ex = e + sin(1);
N = 2;
```

```
[t, x1, x2, x3, x4] = vari_lmm(1/N, 1);
err_ass = norm(x2(end) - x_ex, inf);
for(i = 1 : nstep)
N = 2*N;
[t, x1, x2, x3, x4] = vari_lmm(1/N, 1);
err = abs(x_ex - x2(end));
p(i) = abs(log2(err/err_ass));
err_ass = err;
endfor
P = [p];
N = 2;
[t, x1, x2, x3, x4] = vari_lmm(1/N, 1);
err_ass = norm(x3(end) - x_ex, inf);
for(j = 1 : nstep)
N = 2*N;
[t, x1, x2, x3, x4] = vari_lmm(1/N, 1);
err = abs(x_ex - x3(end));
p(i) = abs(log2(err/err_ass));
err_ass = err;
endfor
P = [P; p];
N = 2;
[t, x1, x2, x3, x4] = vari_lmm(1/N, 1);
err_ass = norm(x4(end) - x_ex, inf);
for(j = 1 : nstep)
N = 2*N;
[t, x1, x2, x3, x4] = vari_lmm(1/N, 1);
err = abs(x_ex - x4(end));
p(i) = abs(log2(err/err_ass));
err_ass = err;
endfor
P = [P; p];
endfunction
```

Dalla stima dell' ordine di convergenza sembra che i metodi siano di ordine di convergenza 2.
Per $\lambda = 1$ e $h = 0.1$ la norma infinito degli errori globali di discretizzazione sono registrati nel vettore e:

$$e = (\ 5.5185e+04,\ 4.7451e+04,\ 5.1406e+03,\ 3.6783e+00\ )$$

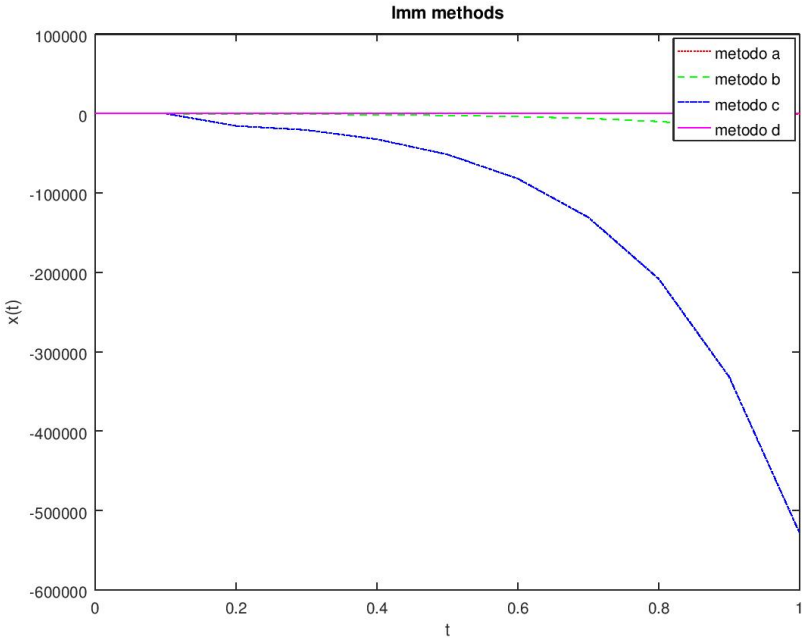
L' approssimazione della soluzione, per $\lambda = 1$, è disegnata nel sottostante grafico:



Per $\lambda = 5$ e $h = 0.1$ la norma infinito degli errori globali di discretizzazione sono registrati nel vettore e:

$$e = (\ 9.2028e+02,\ 2.6326e+04,\ 5.2849e+05,\ 5.3031e+01\ )$$

L' approssimazione della soluzione, per $\lambda = 5$, è disegnata nel sottostante grafico:



2. Il file di tipo *function* **BDF4.m** implementa il metodo BDF a 4-passi:

$$x_{i+i} = \frac{12}{25}hf_{i+1} + \frac{48}{25}x_i - \frac{36}{25}x_{i-1} + \frac{16}{25}x_{i-2}\frac{3}{25}x_{i-3}$$

```
function [t, x] = BDF4(odefun, x0, xinit, tspan, h)
t = tspan(1) : h : tspan(2);
n = length(t);
x = [x0 xinit zeros(1, n - 4)];
alfa = (1/25)*[-3; 16; -36; 48];
beta = 12/25;
for i = 1 : n-4
F = @(s) x(4+i-4 : 4+i-1) * alfa - s + h * beta *...
odefun(t(4+i), s);
x(4+i) = fzero(F, x(4+i-1));
endfor
endfunction
```

3. Sia dato l' IVP:

$$\begin{cases} x'(t) = -tx(t) + \frac{4t}{x(t)}, & t \in (0,1] \\ x(0) = 1 \end{cases}$$

la soluzione esatta è:

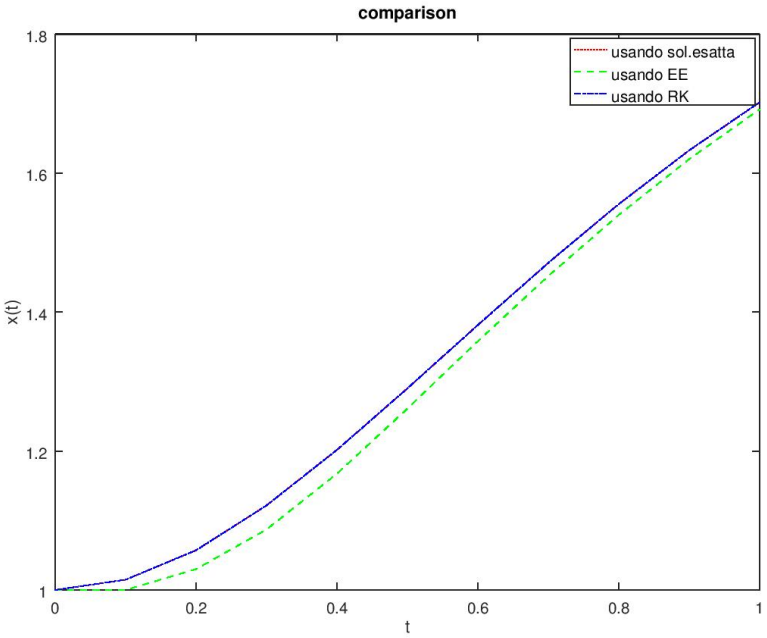
$$x(t) = \sqrt{4 - 3e^{-t^2}}.$$

Per calcolare l' approssimazione numerica della soluzione, il file *es7_3.m* usa la *function* **BDF4.m** con valori iniziali calcolati usando:

1. la soluzione esatta;
2. il metodo di Eulero esplicito;
3. il metodo di Runge Kutta esplicito a 4-stadi.

```
function [t, x1, x2, x3] = es7_3(h)

tspan =[0, 1];
x0 = 1;
x_ex = @(s) (4 - 3 * exp(- s^2))^(1/2);

%xinit calolato usando la soluzione esatta
xinit =  [x_ex(h), x_ex(2*h), x_ex(3*h)];
[t, x1] = BDF4(@funz7_3, x0, xinit, tspan, h);

%xinit calolato usando EE
N = (tspan(2) - tspan(1)) / h;
[r, xinit] =  euler_forw(@funz7_3, tspan, x0, N);
xinit = xinit(2:4);
[t, x2] = BDF4(@funz7_3, x0, xinit, tspan, h);

%xinit calolato usando RK a 4-stadi
G = [0 0 0 0; 1/2 0 0 0; 0 1/2 0 0; 0 0 1 0];
beta = [1/6 1/3 1/3 1/6];
[r, xinit] =  RK(@funz7_3, tspan, x0, h, G, beta);
xinit = xinit(2:4);
[t, x3] = BDF4(@funz7_3, x0, xinit, tspan, h);

plot(t, x1, ":r;usando sol.esatta;", t, x2, "--g;usando EE;",...
t, x3, "-.b;usando RK;");
xlabel("t");
ylabel("x(t)");
title("comparison");
print -djpg image7_3.jpg

function f = funz7_3(t, x)
f = -(t.*x) + 4*(t.*(x^(-1)));
```

Qui sotto il grafico con le approssimazioni numeriche:


comparison
usando sol.esatta
usando EE
usando RK
x(t)
t
1.8
1.6
1.4
1.2
1
0
0.2
0.4
0.6
0.8
1

Esercitazione 8
Metodi predittori-correttori

Il file di tipo *function* **abm4.m** implementa un metodo predittore correttore del tipo PECE basato sul metodo a 4-passi di Adams-Bashforth e sul metodo a 3 passi di Adams-Moulton (entrambe di ordine 4). Per il calcolo dei valori iniziali necessari, utilizzo un metodo di Runge-Kutta esplicito di ordine 4.

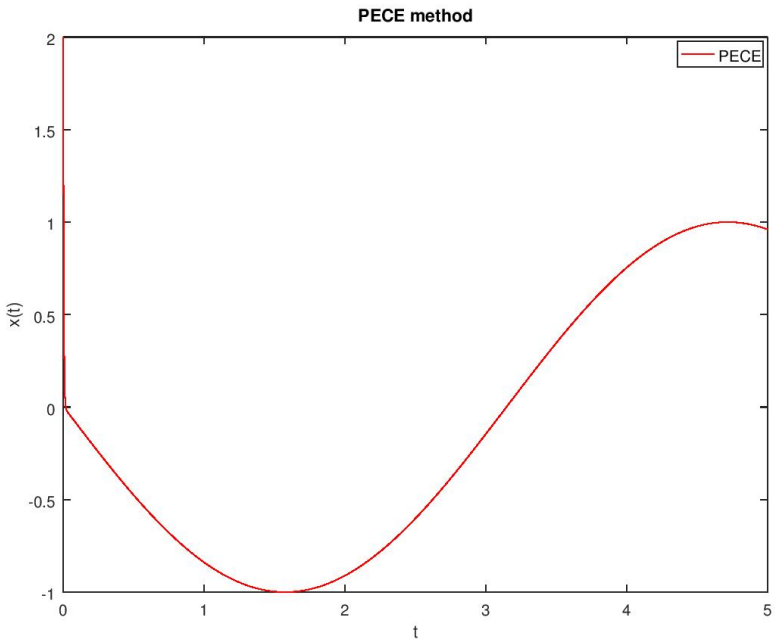
```
function [t, x] = abm4(f, a, b, xa, n)
t = linspace(a, b, n + 1);
x(1) = xa;
h = t(2) - t(1);
alfa_mou = [1 -1 0 0];
beta_mou = [9/24 19/24 -5/24 1/24];
alfa_bash = [1 -1 0 0];
beta_bash = [55/24 -59/24 37/24 -9/24];
Gamma = [0 0 0 0; 1/2 0 0 0; 0 1/2 0 0; 0 0 1 0];
Beta = [1/6 1/3 1/3 1/6];
[s, x] = RK(f, [t(1) t(4)], xa, h, Gamma, Beta);
aux = feval(f, t(1 : 4), x(1 : 4));
aux = aux(:)
x = x(:);
for i = 4 : n
%predictor Adams-Bashforth
x(i+1) = - alfa_bash(2 : end) * flipud(x(i-2 : i)) -...
h * beta_bash * flipud(aux);
%evaluate
ev_pre = feval(f, t(i+1), x(i+1));
%corrector Adams-Moulton
x(i+1) = - alfa_mou(2 : end) * flipud(x(i-2 : i)) +...
h * beta_mou(2 : end) * flipud(aux(1 : (end - 1))) +...
h * beta_mou(1) * ev_pre;
%evaluate
ev_corr = feval(f, t(i+1), x(i+1));
aux = [aux(2 : 4); ev_corr];
endfor
endfunction
```

Applico il metodo ad alcuni problemi visti nelle lezioni precedenti:

Esercizio 3 della esercitazione 3:

```
a = 0;
b = 5;
xa = 2;
n = (b - a)/(10^(-3));
[t, x] = abm4(@funz7, a, b, xa, n);
plot(t, x, "r;PECE;");%diverge con 10^(-2)
xlabel("t");
ylabel("x(t)");
title("PECE method");
```

```
print -djpg image8_3_3.jpg

function f = funz7(t, x)
f = -200 * (x + sin(t));
```



Con la *function* **ord_pece** stimo l' ordine di convergenza del metodo PECE applicato all' esercizio 3 della esercitazione 3

```
function p = ord_pece(nstep)
tspan = [0, 2*pi];
xa = 1;
x_ex = cos(2*pi);
N = 4;
[t, x] = abm4(@funz, tspan(1), tspan(2), xa, N);
err_ass = norm(x(end) - x_ex, inf);
for(i = 1 : nstep)
N = 2*N;
[t, x] = abm4(@funz, tspan(1), tspan(2), xa, N);
err = abs(x_ex - x(end));
p(i) = abs(log2(err/err_ass));
err_ass = err;
endfor

function f = funz(t, x)
f = -x + cos(t) - sin(t);
```

Le *nstep* stime sono memorizzate nel seguente vettore:

$p = (\ 0.34957,\ 0.84962,\ 0.94539,\ 0.97940,\ 0.99124,\ 0.99600,\ 0.99809,\ 0.99907\ )$

L' ordine di convergenza, dunque, sembra essere 1. Problema Stiff (esercizio 1

della esercitazione 6)

```
a = 0;
b = 2*pi;
xa = 1;
h = 10^(-3);
n = (b - a) / h;
[t, x] = abm4(@funz, a, b, xa, n);
x_ex = cos(t);
plot(t, x, t, x_ex);
print -djpg image8_6_1.jpg

function f = funz(t, x)
f = -x + cos(t) - sin(t);
%stimare l' ordine di convergenza
```

Il PECE, rendendo esplicito il metodo, non è applicabile al problema Stiff.

Esercitazione 9
Metodo di shooting

1. Sia dato il problema ai limiti BVP

$$\begin{cases} u''(t) + e^{u(t)+1} = 0 \\ u(0) = u(1) = 0. \end{cases}$$

Tale problema ha due soluzioni

$$u(t) = -2 \ln \left\{ \frac{\cosh((t - 1/2)\theta/2)}{\cosh(\theta/4)} \right\},$$

in quanto θ soddisfa l' equazione:

$$\theta = \sqrt{2e} \cosh(\theta/4).$$

Posto

$$\mathbf{x}(t) = (u(t), u'(t))^T \equiv (x_1(t), x_2(t))^T,$$

il problema può essere riscritto come segue:

$$\begin{cases} \mathbf{x}'(t) = f(t, \mathbf{x}(t)) \\ g(\mathbf{x}(0), \mathbf{x}(1)) = \mathbf{0}. \end{cases} \tag{1}$$

Sia

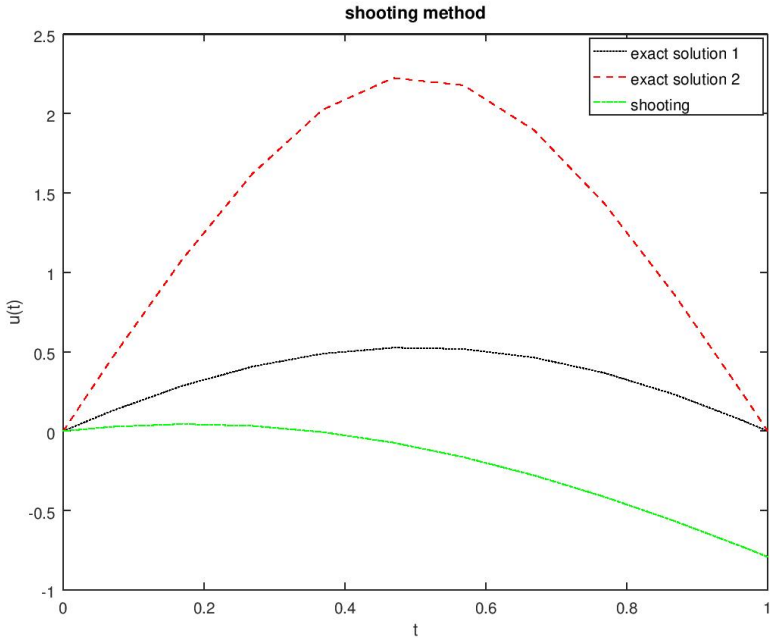
$$\begin{cases} \mathbf{y_s}'(t) = f(t, \mathbf{y_s}) \\ \mathbf{y_s}(0) = (0, s_2^{(0)})^T \equiv \mathbf{s^{(0)}} \end{cases}$$

il primo problema a valori iniziali associato al problema ai limiti assegnato. Il file di tipo *function* **es9_1.m** implementa il metodo di shooting per la risoluzione di (2).

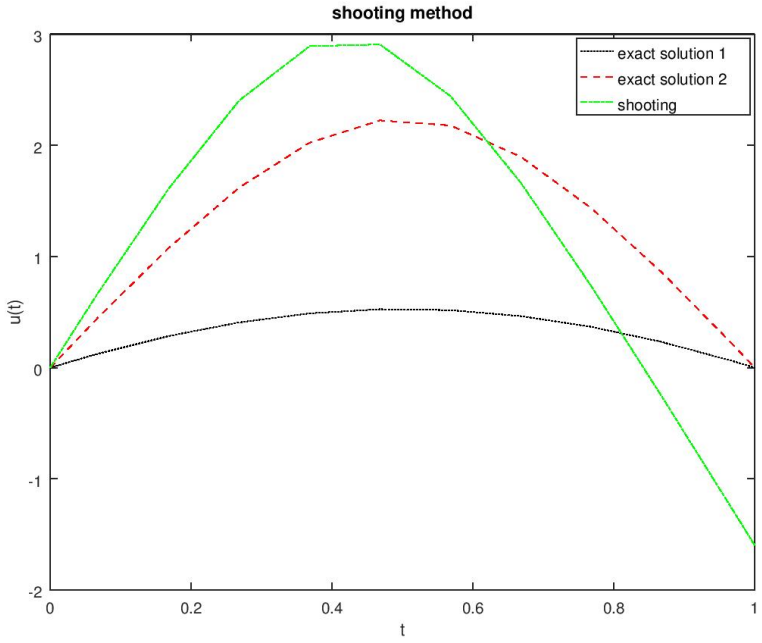
```
%s2_0 = 0.5 oppure 10,maxit = 1000,tol = 10^(-6)
function [t_sol, X] = es9_1(s2_0, tol, maxit)
f1 = @(t,x) [x(2); -exp(x(1) + 1)];
B1 = [1 0; 0 0];
B2 = [0 0; 1 0];
i = 0;
%initial value
s = [0; s2_0];
s_new = [inf; inf];
while  i < maxit && norm(s_new - s, Inf) > tol
%solve IVP1
S1 = ode45(f1, [0 1], s);
t_sol = S1.x;
X = S1.y;
%solve IVP2 with ode45
A = [0 1; -exp(X(end,1) + 1) 0];
f2 = @(t,y) reshape((A * reshape(y, 2,2)), 4, 1);
S2 = ode45(f2, [0 1], [1 0 0 1]');
```

```
t = S2.x;
y = S2.y;
Y = reshape(y', 2, 2*length(t));
%IVP2 exact solution
%Y = @(t,x) expm([0 1; -exp(x + 1) 0] * t);
%exact Jacobian
%J = B0 + B1 * Y(1, X(end,1));
J = B0 + B1 * Y(:, end-1:end);
xi = (J^(-1)) * (-B0 * s - B1 * X(:,end));
s_new = s + xi;
s = s_new;
i = i + 1;
endwhile

F = @(theta) theta - ((2*e)^(1/2)) * cosh(theta/4);
%F(3) * F(5) < 0
theta1 = fzero(F, [3, 5]);
%sol. esatta 1
u_ex1 = -2 * (log(cosh((t_sol.- 1/2) * (theta1/2)) / cosh(theta1/4)));
%F(5) * F(7.5) < 0
theta2 = fzero(F, [5 ,7.5]);
%sol. esatta 2
u_ex2 = -2 * (log(cosh((t_sol.- 1/2) * (theta2/2)) / cosh(theta2/4)));

plot(t_sol, u_ex1, ":k;exact solution 1;", t_sol, u_ex2,...
"--r;exact solution 2;", t_sol, X(1,:), "-.g;shooting;");
xlabel("t");
ylabel("u(t)");
title("shooting method");
print -djpg image9_1.jpg
endfunction
```

A seconda del valore assegnato ad $s_2^{(0)}$, il metodo di shooting convergerà ad una o all' altra soluzione continua aventi come parametro rispettivamente θ_1 e θ_2, con $\theta_1 < \theta_2$.
Per $s_2^{(0)} = 0.5$ converge alla soluzione con θ_1 come possiamo vedere dall' immagine:



Per $s_2^{(0)} = 10$ converge alla soluzione con θ_2 come possiamo vedere dall' immagine:


shooting method
exact solution 1
exact solution 2
shooting
u(t)
t
3
2
1
0
-1
-2
0
0.2
0.4
0.6
0.8
1

2. Sia λ un numero reale e positivo,

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2\lambda^3 & \lambda^2 & 2\lambda \end{pmatrix}, \quad \mathbf{r}(t) = \begin{pmatrix} 0 \\ 0 \\ (\pi^2+\lambda^2)(\pi sin(\pi t) + 2\lambda cos(\pi t) \end{pmatrix},$$

$$B_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad B_1 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \frac{3+2e^{-\lambda}+e^{-2\lambda}}{2+e^{-\lambda}} \\ 0 \\ \frac{3\lambda-\lambda e^{-\lambda}}{2+e^{-\lambda}} \end{pmatrix},$$

Posto $\mathbf{x}(t) = (u(t), u'(t), u''(t))^T$, il file di tipo *function* **es9_2.m** implementa il metodo di shooting per la risoluzione del problema ai limiti

$$\begin{cases} \mathbf{x}'(t) = A\mathbf{x}'(t) + \mathbf{r}(t) \\ B_0\mathbf{x}(0) + B_1\mathbf{x}(1) = \mathbf{b}, \end{cases}$$

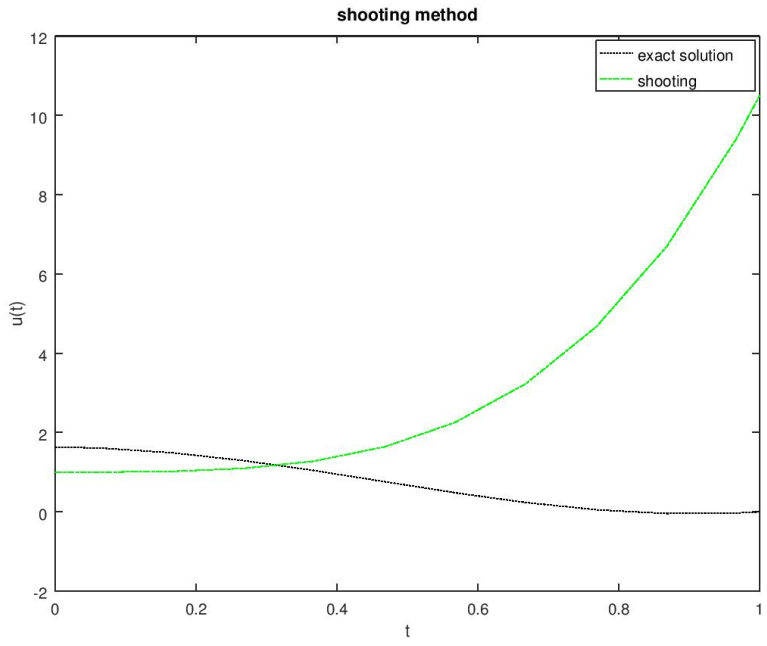
La cui soluzione é

$$u(t) = \frac{e^{\lambda(t-1)} + e^{2\lambda(t-1)} + e^{-\lambda t}}{2 + e^{-\lambda}} + cos(\pi t)$$

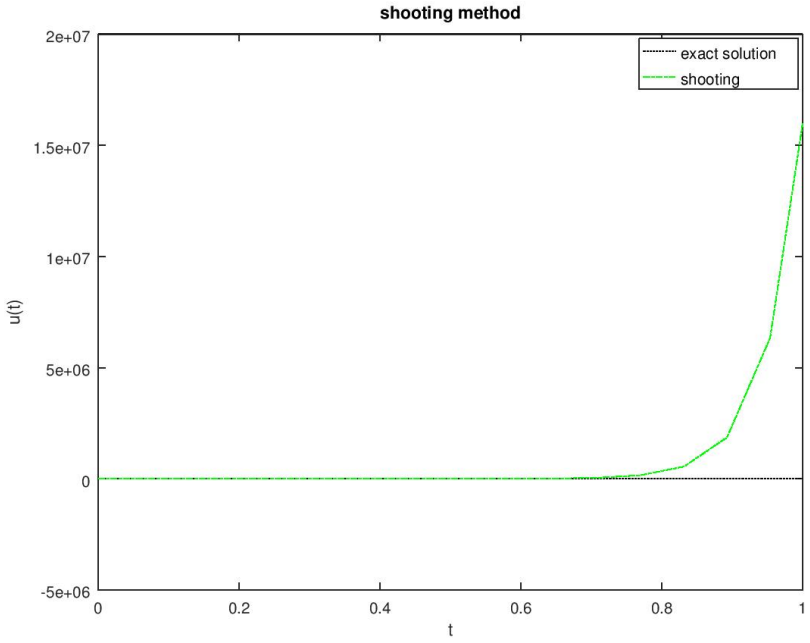
```
function [t_sol, x] = es9_2(lambda, s, maxit, tol)
%maxit = 1000,tol = 10^(-6)
s = s(:);
A = [0, 1, 0; 0, 0, 1; -2*(lambda^3), lambda^2, 2*lambda];
r = @(t) [0; 0; (pi^2 + lambda^2) * (pi*sin(pi*t) + 2*lambda*cos(pi*t))];
B0 = [1, 0, 0; 0, 0, 0; 0, 0, 0];
B1 = [0, 0, 0; 1, 0, 0; 0 1 0];
b = (1/(2+exp(-lambda)))*[3 + 2*exp(-lambda) + exp(-2*lambda);...
0; 3*lambda - lambda*exp(-lambda)];
f1 = @(t,x) A*x + r(t);
i = 0;
%initial value s
s_new = [inf, inf];
while  i < maxit && norm(s_new - s, Inf) > tol
%solve IVP1
S1 = ode45(f1, [0 1], s);
t_sol = S1.x;
X = S1.y;
%solve IVP2 iwth ode45
%D/Dx(A*x + r(t)) = A
f2 = @(t,y) reshape((A * reshape(y, 3,3)), 9, 1);
S2 = ode45(f2, [0 1], [1 0 0 0 1 0 0 0 1]');
t = S2.x;
y = S2.y;
Y = reshape(y', 3, 3*length(t));
%IVP2 exact solution
%Y = @(t,x) expm(A * t);
%exact Jacobian
%J = B1 + B0 * Y(1, X(end,1));
```

```
J = B0 + B1 * Y(:, end-2:end);
xi = (J^(-1)) * (-B0 * s - B1 * X(:,end));
s_new = s + xi;
s = s_new;
i = i + 1;
endwhile
x = X(1,:);

%sol. esatta
u_ex = ((exp(lambda*(t_sol.-1)) + exp(2*lambda*(t_sol.-1)) +...
exp(-lambda*(t_sol))) / (2 + exp(-lambda))) + cos(pi*t_sol);
plot(t_sol, u_ex, ":k;exact solution;", t_sol, x, "-.g;shooting;");
xlabel("t");
ylabel("u(t)");
title("shooting method");
print -djpg image9_2.jpg
endfunction
```

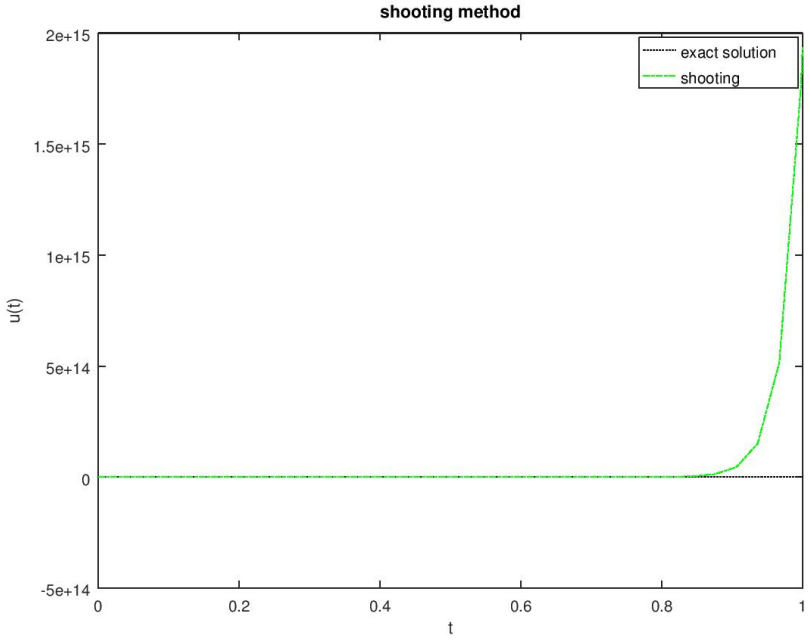
Qui sotto i grafici con la soluzione esatta e quella approssimata per vari valori di del parametro λ, compresi tra 1 e 50: Per $\lambda = 1$:



Per $\lambda = 10$:



Per $\lambda = 20$:



Osserviamo che man mano che λ cresce la matrice Jacobiana è sempre più mal condizionata e l' algoritmo di shooting non converge alla soluzione del problema.

Esercitazione 10
Metodi alle differenze finite e codici Matlab bvp4c e tom

Esempio. Sia dato il problema ai limiti BVP

$$\begin{cases} y''(t) + \frac{3py}{(p+t^2)^2} = 0, \\ y(-0.1) = -\frac{0.1}{\sqrt{p+0.01}}, \quad y(0.1) = \frac{0.1}{\sqrt{p+0.01}}, \end{cases}$$

con p parametro reale. La sua soluzione analitica è

$$y(t) = \frac{t}{\sqrt{p+t^2}}.$$

Per poter risolvere numericamente tale problema, utilizzando i codici **bvp4c** e **tom**, inanzitutto implemento una *function* relativa alla funzione che definisce l' equazione differenziale:

```
function dydt = exode( t,y,p )
dydt = [y(2); -3*p*y(1)/(p+t^2)^2];
endfunction
```

Devo inoltre costruire una *function* relativa alle condizioni ai limiti:

```
function res = exbc(ya, yb, p)
yatb = 0.1/sqrt(p+0.01);
yata = -yatb;
res = [ya(1)-yata; -yb(1) - yatb];
endfunction
```

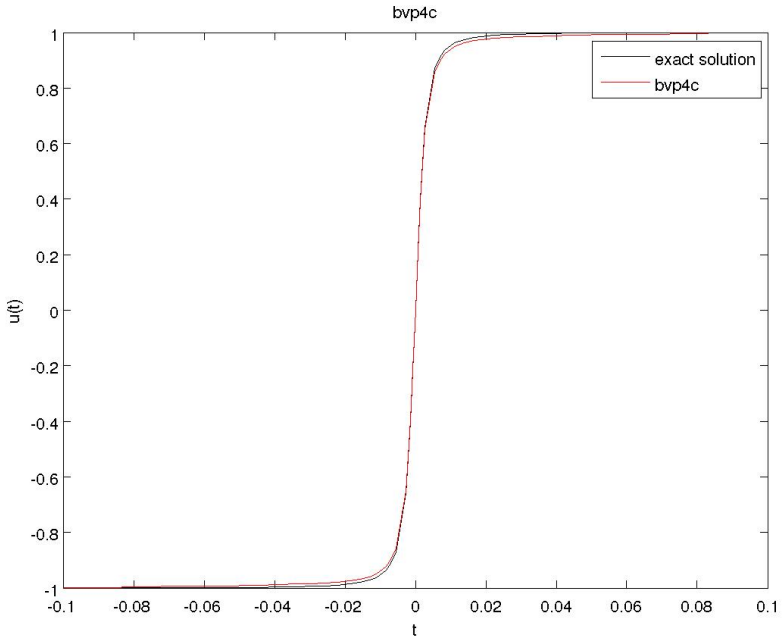
Infine assegno un profilo iniziale alla soluzione, definendo a tal fine:

```
function v = exinit(t)
v =[0; 10];
endfunction
```

Il problema dato puó dunque essere risolto numericamente dando le seguenti istruzioni:

```
solinit = bvpinit(linspace(-0.1, 0.1, 10), @exinit);
par = 1e-5;
sol = bvp4c(@exode, @exbc, solinit, [], par);
```

Il grafico dell' approssimazione numerica e della soluzione esatta:



1. Sia λ un numero reale e positivo,

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2\lambda^3 & \lambda^2 & 2\lambda \end{pmatrix}, \quad \mathbf{r}(t) = \begin{pmatrix} 0 \\ 0 \\ (\pi^2 + \lambda^2)(\pi sin(\pi t) + 2\lambda cos(\pi t) \end{pmatrix},$$

$$B_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad B_1 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \frac{3+2e^{-\lambda}+e^{-2\lambda}}{2+e^{-\lambda}} \\ 0 \\ \frac{3\lambda-\lambda e^{-\lambda}}{2+e^{-\lambda}} \end{pmatrix},$$

Posto $\mathbf{x}(t) = (u(t), u'(t), u''(t))^T$, il file di tipo *function* **es10_1.m** implementa il metodo alle differenze finite, basato sul metodo midpoint, per la risoluzione del problema ai limiti

$$\begin{cases} \mathbf{x}'(t) = A\mathbf{x}'(t) + \mathbf{r}(t) \\ B_0\mathbf{x}(0) + B_1\mathbf{x}(1) = \mathbf{b}, \end{cases}$$

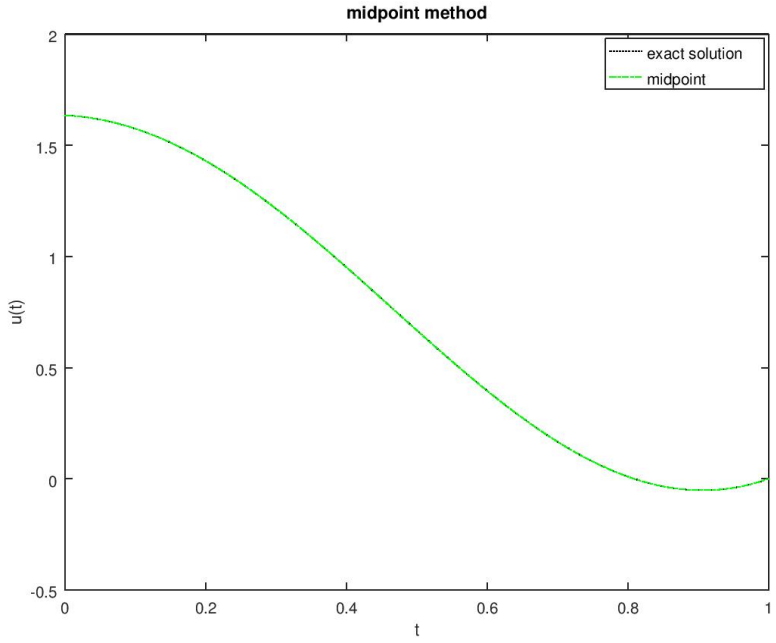
La cui soluzione é

$$u(t) = \frac{e^{\lambda(t-1)} + e^{2\lambda(t-1)} + e^{-\lambda t}}{2 + e^{-\lambda}} + cos(\pi t).$$

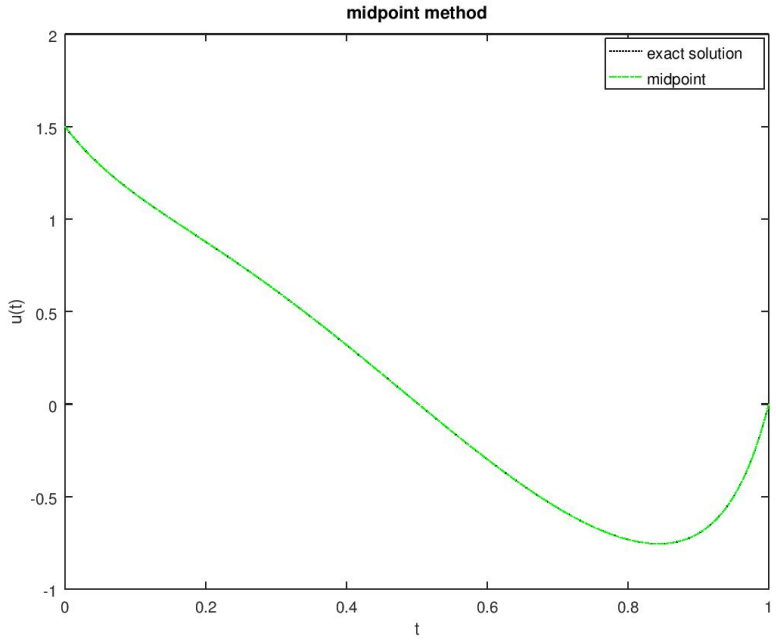
```
function [t, x] = es10_1midpoint(lambda, N)
%N numero di sottointervalli
tspan = [0, 1];
A = [0, 1, 0; 0, 0, 1; -2*(lambda^3), lambda^2, 2*lambda];
r = @(t) [0; 0; (pi^2 + lambda^2) * (pi*sin(pi*t) +...
```

```
2*lambda*cos(pi*t))];
B0 = [1, 0, 0; 0, 0, 0; 0, 0, 0];
B1 = [0, 0, 0; 1, 0, 0; 0 1 0];
b = (1/(2+exp(-lambda)))*[3 + 2*exp(-lambda) + exp(-2*lambda);...
0; 3*lambda - lambda*exp(-lambda)];
t = linspace(tspan(1), tspan(2), N+1);
h = t(2) - t(1);
n = size(A)(1);
%costruzione della matrice del sistema lineare
W = zeros(0);
for  i = 1 : N
T1 = zeros(n, n * (i-1));
S = -(eye(n) + (h/2)*A) / h;
R = (eye(n) - (h/2)*A) / h;
T2 = zeros(n, n * (N-1-(i-1)));
W = [W; T1, S, R, T2];
endfor
W = [W; B0, zeros(n, n * (N-1)), B1];
%costruzione del termine noto del sistema lineare
c = zeros(0);
for  i = 1 : N
tau = (t(i+1) + t(i)) / 2;
c = [c; r(tau)];
endfor
c = [c; b];
%soluzione del sistema lineare
X = inv(W) * c;
X = reshape(X, n, length(t));
%soluzione numerica
x = X(1,:);

%sol. esatta
u_ex = ((exp(lambda*(t.-1)) + exp(2*lambda*(t.-1)) +...
exp(-lambda*(t))) / (2 + exp(-lambda))) + cos(pi*t);
plot(t, u_ex, ":k;exact solution;", t, x, "-.g;midpoint;");
xlabel("t");
ylabel("u(t)");
title("midpoint method");
print -djpg image10_1midpoint.jpg
endfunction
```

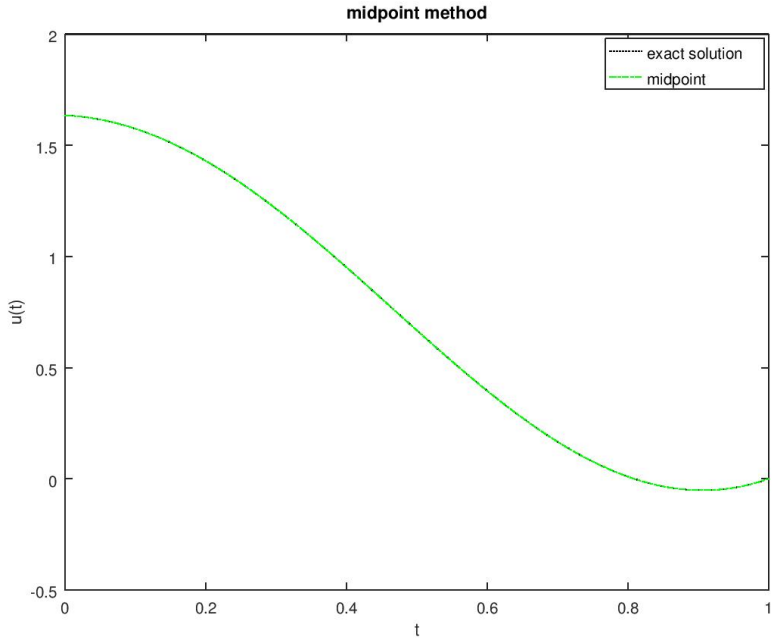
1. Qui sotto i grafici con la soluzione esatta e quella approssimata per vari valori di del parametro λ, compresi tra 1 e 50.
   Per $\lambda = 1$:



Per $\lambda = 10$:



Per $\lambda = 20$:



Ora con il metodo alle differenze finite si riescono a risolvere anche i problemi in cui il metodo di shooting falliva.

2. Ora determino sperimentalmente l' ordine di convergenza del metodo:

```
function p = ord_midpoint(lambda, nstep)
tspan = [0, 1];
x_ex = 1 + cos(pi);
N = 2;
[t, x] = es10_1midpoint(lambda, N);
err_ass = norm(x(end) - x_ex, inf)
for(i = 1 : nstep)
N = 2*N;
[t, x] = es10_1midpoint(lambda, N);
err = abs(x_ex - x(end));
p(i) = abs(log2(err/err_ass));
err_ass = err;
endfor
endfunction
```

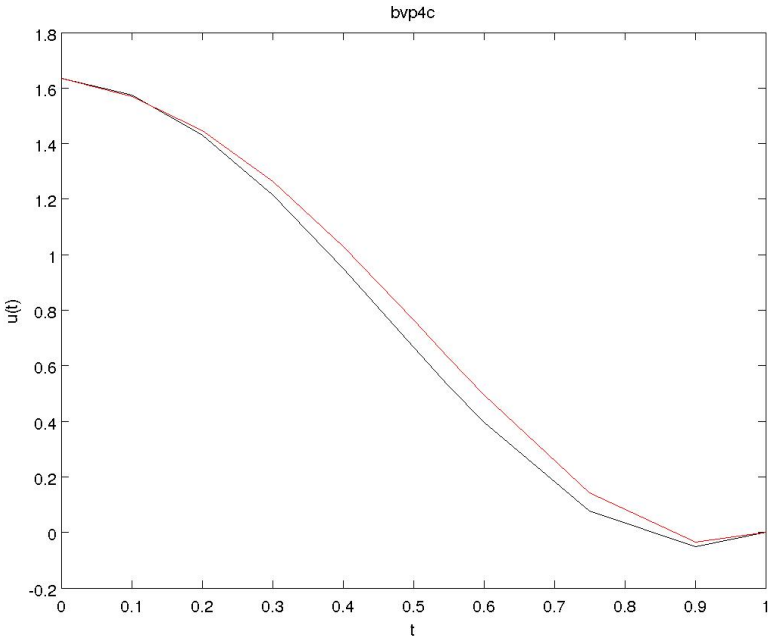
Il metodo midpoint sembra convergere alla soluzione esatta con ordine 2.

2. Risolvo ora il problema precedente con i codici **bvp4c** e **tom**:

Usando **bvp4c**:

```
function [t, u] = es10_2bvp4c(lambda, N)
%A = [0, 1, 0; 0, 0, 1; -2*(lambda^3), lambda^2, 2*lambda];
%r = @(t) [0; 0; (pi^2 + lambda^2) * (pi*sin(pi*t)) +...
%2*lambda*cos(pi*t)];
%B0 = [1, 0, 0; 0, 0, 0; 0, 0, 0];
%B1 = [0, 0, 0; 1, 0, 0; 0, 1, 0];
%b = (1/(2+exp(-lambda)))*[3 + 2*exp(-lambda) + exp(-2*lambda);...
%0; 3*lambda - lambda*exp(-lambda)];
%f1 =@(t,x) A*x +r(t);
tspan = [0, 1];
solinit = bvpinit(linspace(tspan(1), tspan(2), N+1), @exinit);
par = lambda;
sol = bvp4c(@exode, @exbc, solinit, [], par);
t = sol.x;
X = sol.y;
u = X(1,:);
%sol. esatta
u_ex = (1/(2+exp(-lambda))) * (exp(lambda*(t-1)) + ....
exp(2*lambda*(t-1)) + exp(-lambda*t)) + cos(pi*t);

plot(t, u_ex, 'k', t, u, 'r');
xlabel('t');
ylabel('u(t)');
title('bvp4c');
print('es10_2bvp4c','-djpeg')

function dxdt = exode(t, x, lambda)
dxdt = [x(2); x(3); -2*(lambda^3)*x(1) + (lambda^2)*x(2) +...
2*lambda*x(3) + (pi^2 + lambda^2) * (pi*sin(pi*t)) +...
2*lambda*cos(pi*t)];
function res = exbc(xa, xb, lambda)
res = [xa(1) - (1/(2+exp(-lambda)))*(3 + 2*exp(-lambda) +...
exp(-2*lambda)); xb(1);...
xb(2) - (1/(2+exp(-lambda)))*(3*lambda - lambda*exp(-lambda))];
function v = exinit(t)
v = [0; 0; 1];
```

Qui sotto il grafico della soluzione esatta e di quella approssimata

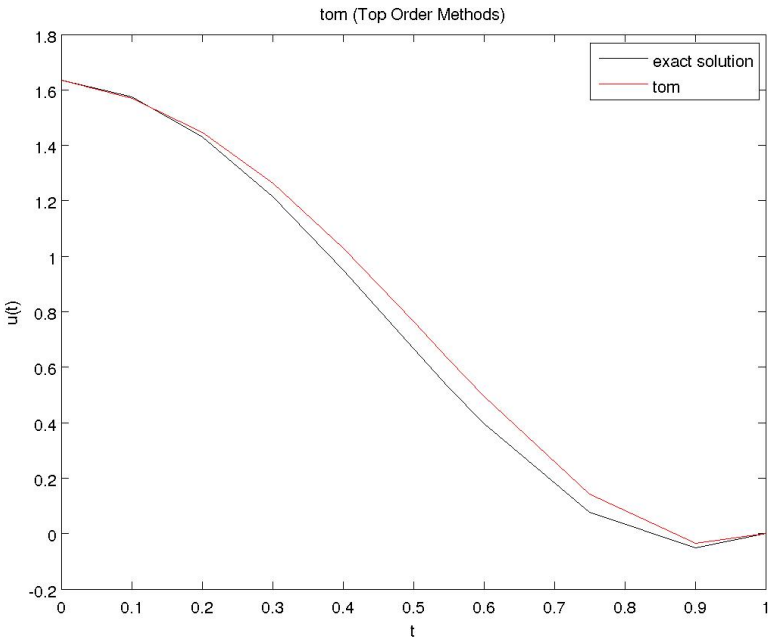


Usando invece **tom**:

```
function [t, u] = es10_2tom(lambda, N)
%A = [0, 1, 0; 0, 0, 1; -2*(lambda^3), lambda^2, 2*lambda];
%r = @(t) [0; 0; (pi^2 + lambda^2) * (pi*sin(pi*t)) +...
%2*lambda*cos(pi*t)];
%B0 = [1, 0, 0; 0, 0, 0; 0, 0, 0];
%B1 = [0, 0, 0; 1, 0, 0; 0, 1, 0];
%b = (1/(2+exp(-lambda)))*[3 + 2*exp(-lambda) + exp(-2*lambda);...
%0; 3*lambda - lambda*exp(-lambda)];
%f1 =@(t,x) A*x +r(t);
tspan = [0, 1];
solinit = tominit(linspace(tspan(1), tspan(2), N+1), @exinit);
par = lambda;
sol = bvp4c(@exode, @exbc, solinit, [], par);
t = sol.x;
X = sol.y;
u = X(1,:);
%sol. esatta
u_ex = (1/(2+exp(-lambda))) * (exp(lambda*(t-1)) +...
exp(2*lambda*(t-1)) + exp(-lambda*t)) + cos(pi*t);

plot(t, u_ex, 'k', t, u, 'r');
xlabel('t');
ylabel('u(t)');
title('tom (Top Order Methods)');
print('es10_2tom','-djpeg')
```

```
function dxdt = exode(t, x, lambda)
dxdt = [x(2); x(3);...
-2*(lambda^3)*x(1) + (lambda^2)*x(2) + 2*lambda*x(3) +...
(pi^2 + lambda^2) * (pi*sin(pi*t)) + 2*lambda*cos(pi*t)];
function res = exbc(xa, xb, lambda)
res = [xa(1) - (1/(2+exp(-lambda)))*(3 + 2*exp(-lambda) + exp(-2*lambda));...
xb(1); xb(2) - (1/(2+exp(-lambda)))*(3*lambda - lambda*exp(-lambda))];
function v = exinit(t)
v = [0; 0; 1];
```

Qui sotto il grafico della soluzione esatta e di quella approssimata



3. Dato l' IVP

$$\begin{cases} u''(t) + e^{u(t)+1} = 0 \\ u(0) = u(1) = 0. \end{cases}$$

Tale problema ha due soluzioni

$$u(t) = -2\ln\left\{\frac{\cosh((t-1/2)\theta/2)}{\cosh(\theta/4)}\right\},$$

in quanto θ soddisfa l' equazione:

$$\theta = \sqrt{2e}\cosh(\theta/4).$$

Posto

$$\mathbf{x}(t) = (u(t), u'(t))^T \equiv (x_1(t), x_2(t))^T,$$

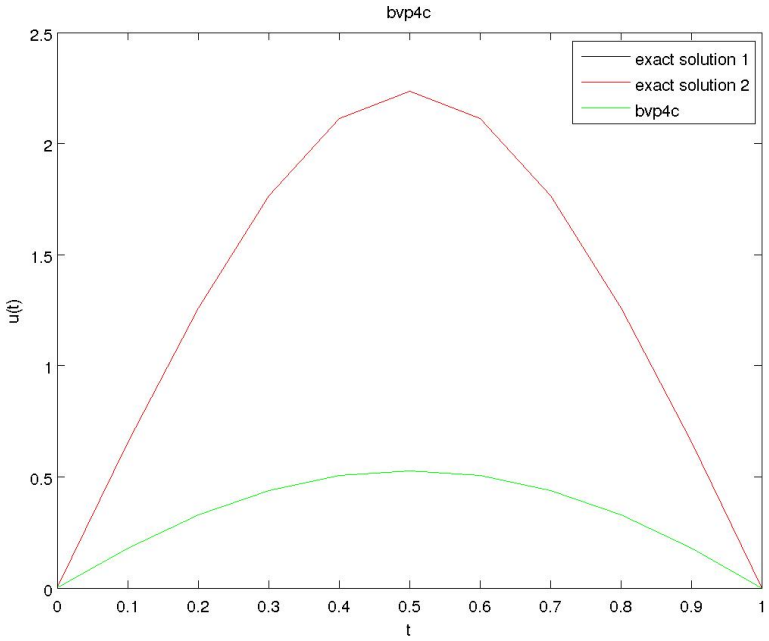
il problema può essere riscritto come segue:

$$\begin{cases} \mathbf{x}'(t) = f(t, \mathbf{x}(t)) \\ g(\mathbf{x}(0), \mathbf{x}(1)) = \mathbf{0}. \end{cases} \tag{2}$$

Utilizzo ora il codice **bvp4c** per la sua risoluzione:

```
function [t, u] = es10_3bvp4c(N)
tspan = [0, 1];
solinit = bvpinit(linspace(tspan(1), tspan(2), N+1), @exinit);
sol = bvp4c(@exode, @exbc, solinit);
t = sol.x;
u = sol.y(1,:);
F = @(theta) theta - ((2*exp(1))^(1/2))* cosh(theta/4);
%F(3) * F(5) < 0
theta1 = fzero(F, [3, 5]);
%sol. esatta 1
u_ex1 = -2 * (log(cosh((t-(1/2)*ones(1,length(t))) *...
(theta1/2)) / cosh(theta1/4)));
%F(5) * F(7.5) < 0
theta2 = fzero(F, [5, 7.5]);
%sol. esatta 2
u_ex2 = -2 * (log(cosh((t-(1/2)*ones(1,length(t))) *...
(theta2/2)) / cosh(theta2/4)));
%la soluzione approssimata u e' "uguale" a u_ex1

plot(t, u_ex1, 'k', t, u_ex2, 'r', t, u, 'g');
xlabel('t');
ylabel('u(t)');
title('bvp4c');
print('es10_3bvp4c','-djpeg')

function dudt = exode(t, u)
dudt = [u(2); -exp(u(1) + 1)];
function res = exbc(ua, ub)
uata = 0;
uatb = 0;
res = [ua(1) - uata; ub(1) - uatb];
function v = exinit(t)
v = [0; 10];
```

Qui sotto il grafico:



Ora invece utilizzo il codice **tom**

```
function [t, u] = es10_3tom(N)
tspan = [0, 1];
solinit = tominit(linspace(tspan(1), tspan(2), N+1), @exinit);
sol = tom(@exode, @exbc, solinit);
t = sol.x;
u = sol.y(1,:);
F = @(theta) theta - ((2*exp(1))^(1/2))* cosh(theta/4);
%F(3) * F(5) < 0
theta1 = fzero(F, [3, 5]);
%sol. esatta 1
u_ex1 = -2 * (log(cosh((t-(1/2)*ones(1,length(t))) *...
(theta1/2)) / cosh(theta1/4)));
%F(5) * F(7.5) < 0
theta2 = fzero(F, [5, 7.5]);
%sol. esatta 2
u_ex2 = -2 * (log(cosh((t-(1/2)*ones(1,length(t))) *...
(theta2/2)) / cosh(theta2/4)));

%la soluzione approssimata u e' "uguale" a u_ex1
plot(t, u_ex1, 'k', t, u_ex2, 'r', t, u, 'g');
xlabel('t');
ylabel('u(t)');
title('tom (Top Order Methods)');
print('es10_3tom','-djpeg')
```

```
function dudt = exode(t, u)
dudt = [u(2); -exp(u(1) + 1)];
function res = exbc(ua, ub)
uata = 0;
uatb = 0;
res = [ua(1) - uata; ub(1) - uatb];
function v = exinit(t)
v = [0; 10];
```

Qui sotto il grafico: