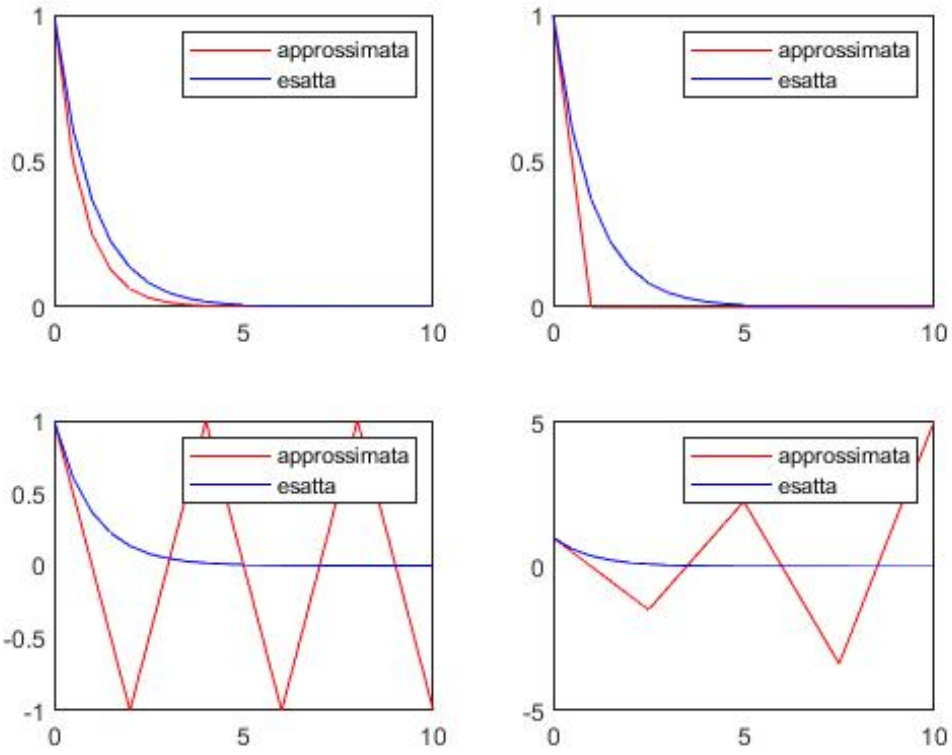
Esercizio 1

Il seguente script risolve l'equazione $y' = -y$ nell'intervallo $(0, 10]$ usando la funzione `eulero` fatta alla prima lezione, e confronta la soluzione numerica con la soluzione reale $y = e^{-x}$. I passi di integrazione sono pari a $0.5, 1, 2, 2.5$. nelle varie sezioni del grafico.

```
odefun=@(x,y) -y;
[x1,u1]=eulero(odefun, [0,10],1,0.5);
[x2,u2]=eulero(odefun, [0,10],1,1);
[x3,u3]=eulero(odefun, [0,10],1,2);
[x4,u4]=eulero(odefun, [0,10],1,2.5);
y=exp(-x1);
subplot(2,2,1)
plot(x1,u1,'r',x1,y,'b')
legend('approssimata','esatta')
subplot(2,2,2)
plot(x2,u2,'r',x1,y,'b')
legend('approssimata','esatta')
subplot(2,2,3)
plot(x3,u3,'r',x1,y,'b')
legend('approssimata','esatta')
subplot(2,2,4)
plot(x4,u4,'r',x1,y,'b')
legend('approssimata','esatta')
```



Notiamo che solo il primo valore di h approssima in modo soddisfacente la soluzione. Nel secondo caso, il primo passo di integrazione porta la funzione a essere pari a zero e quindi poi costante, mentre negli ultimi due i risultati sono ancora peggiori, avendo soluzioni oscillanti fra valori positivi e negativi (assurdo per un'esponenziale).

Esercizio 2

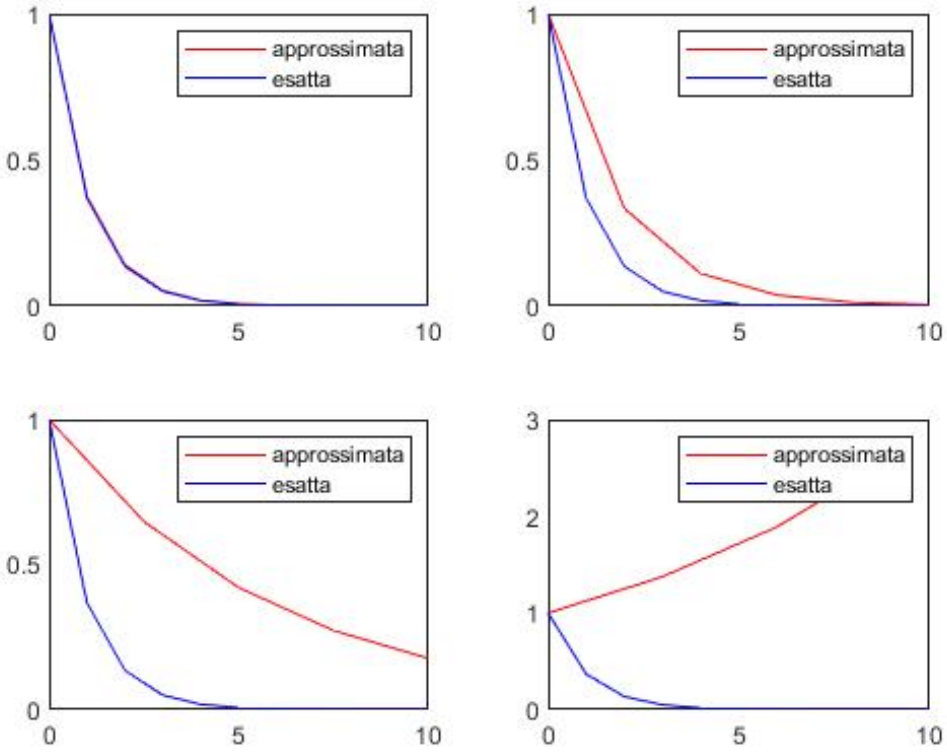
Questa e' la function che prende in input, come la function `eulero`, la funzione da integrare, l'intervallo di integrazione, il valore iniziale e il passo di integrazione.

```
function [x,u]=RK4(odefun,tspan,y0,h)
N=floor((tspan(2)-tspan(1))/h)+1;
x=zeros(1,N);
x(1)=tspan(1);
for i=2:N
    x(i)=x(i-1)+h;
end
u(:,1)=y0;
for i=2:N
    F1=feval(odefun,x(i-1), u(:,i-1));
    F2=feval(odefun,x(i-1)+0.5*h,u(:,i-1)+0.5*h*F1);
    F3=feval(odefun,x(i-1)+0.5*h,u(:,i-1)+0.5*h*F2);
    F4=feval(odefun,x(i-1)+h,u(:,i-1)+h*F3);
    u(:,i)= u(:,i-1)+(h/6)*(F1+2*F2+2*F3+F4);
end
end
```

Il seguente script invece utilizza la function RK4 ponendo $h = 1, 2, 2.5, 3$ per risolvere l'equazione del punto precedente:

```
odefun=@(x,y) -y;
[x1,u1]=RK4(odefun, [0,10],1,1);
[x2,u2]=RK4(odefun, [0,10],1,2);
[x3,u3]=RK4(odefun, [0,10],1,2.5);
[x4,u4]=RK4(odefun, [0,10],1,3);
y=exp(-x1);
subplot(2,2,1)
plot(x1,u1,'r',x1,y,'b')
legend('approssimata','esatta')
subplot(2,2,2)
plot(x2,u2,'r',x1,y,'b')
legend('approssimata','esatta')
subplot(2,2,3)
plot(x3,u3,'r',x1,y,'b')
legend('approssimata','esatta')
subplot(2,2,4)
plot(x4,u4,'r',x1,y,'b')
legend('approssimata','esatta')
```

Questo è il grafico:



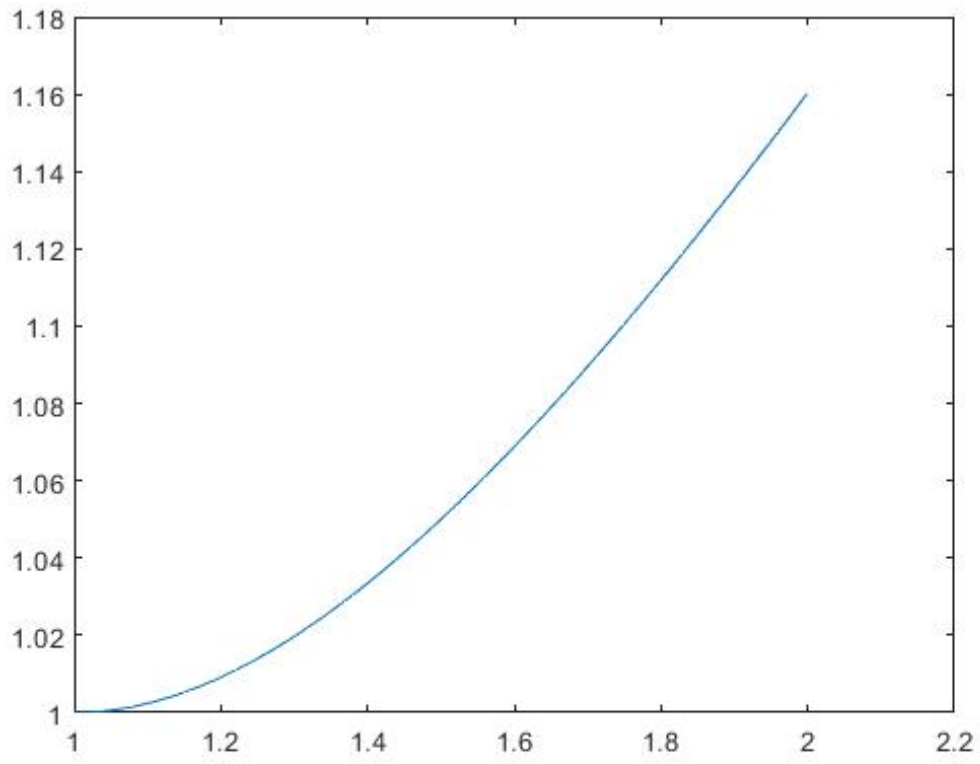
Dal grafico notiamo immediatamente come $h = 1$ porti una magnifica approssimazione della soluzione, e poi la precisione vada via degradandosi per h che cresce, fino al arrivare a $h = 3$ che non conserva nemmeno il segno della derivata della soluzione. Ad ogni modo i risultati sono molto migliori di quelli del metodo di Eulero: ad esempio il segno della soluzione è perlomeno sempre quello che ci aspettiamo.

Esercizio 3

Il seguente script invece utilizza la function RK4 (ponendo $h = 0.01$) per risolvere un'equazione del secondo ordine, trasformandola nella risoluzione di un sistema, e poi plotta la soluzione.

```
fun=@(x,y) [y(2); -(4*x+1)/(2*(x+1))*y(2)+(2*x-1)/(4*x^2)*(3*y(1)
    ^3+y(1))/(1+y(1)^2)];
tspan=[1,2];
y0=[1,0];
h=0.01;
[x,u]=RK4(fun,tspan,y0,h);
plot(x,u(1,:));
```

Il seguente è il grafico ottenuto



Esercizio 4

La function seguente risolve $y' = -y - 5e^{-x}\sin(5x)$ usando prima la function `eulero` e in seguito la function `RK4` per vari valori di h, e ne confronta i valori in $x = 2$ con la soluzione corretta $y = e^{-x}\cos(5x)$, riportandoli in una matrice.

```
function E=errori
fun=@(x,y) -y-5*exp(-x)*sin(5*x);
tspan=[0,5];
init=1;
sol=@(x) exp(-x)*cos(5*x);
s=sol(2);
h=zeros(1,10);
for i=1:10
    h(i)=1/(10*i);
end
e1=zeros(1,10);
e2=zeros(1,10);
for i=1:10
    [~,y]=eulero(fun,tspan,init,h(i));
    [~,u]=RK4(fun,tspan,init,h(i));
    e1(i)=abs(y(2/h(i)+1)-s);
    e2(i)=abs(u(2/h(i)+1)-s);
end
E=[h;e1;e2]';
end
```

Valore di h	Errore Eulero	Errore Runge-Kutta
0.100000000000000	0.040186145338622	0.000004850098076
0.050000000000000	0.020689623587931	0.000000299815263
0.033333333333333	0.013924423699184	0.000000059073893
0.025000000000000	0.010492479090989	0.000000018671810
0.020000000000000	0.008417565026720	0.000000007643690
0.016666666666667	0.007027734523799	0.000000003684923
0.014285714285714	0.006031789745835	0.000000001988568
0.012500000000000	0.005283076898811	0.000000001165468
0.011111111111111	0.004699705297912	0.000000000727505
0.010000000000000	0.004232353274852	0.000000000477267

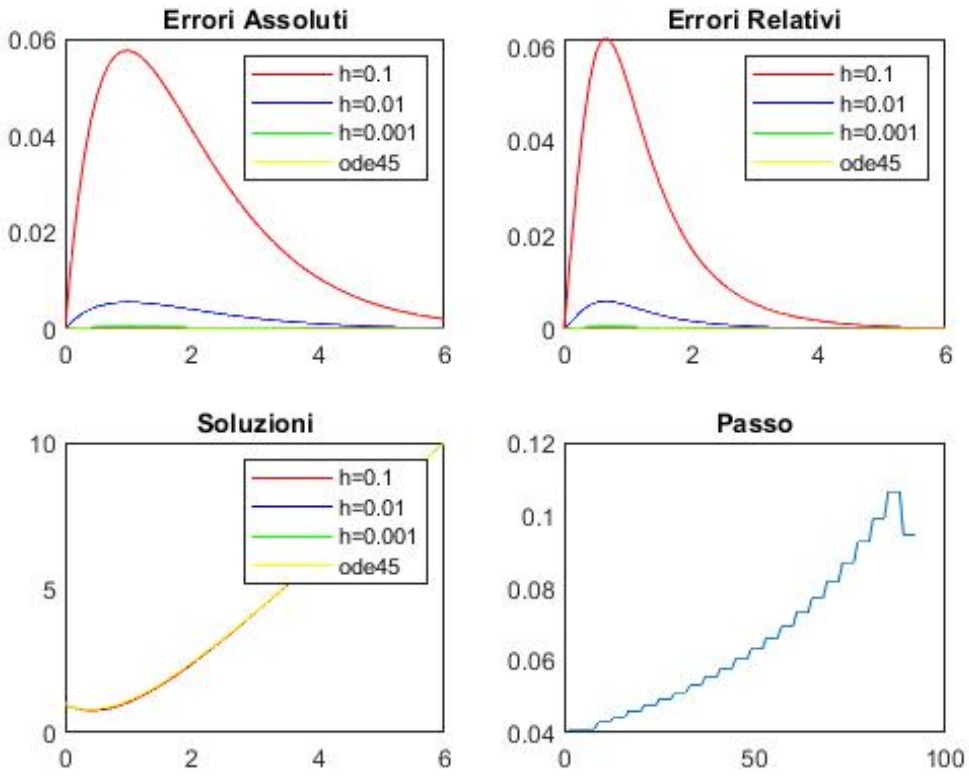
Notiamo che, fin dal valore iniziale $h = 1$, l'errore portato dal metodo di Eulero risulta molto maggiore rispetto a quello del metodo di Runge-Kutta, in particolare di un fattore 10^4. Notiamo che l'errore del metodo di Eulero dipende, in accordo con le teoria, in modo approssimativamente lineare da h: invece l'errore del metodo di Runge Kutta segue per i primi valori l'andamento, anche questo in accordo con la teoria, della quarta potenza di h. Per i valori più elevati dell'indice di h questo andamento scompare, anche se gli errori in questione sono abbastanza piccoli da farci pensare che altri fattori li stiano influenzando (come ad esempio la precisione di macchina, oppure l'implementazione delle function precostruite in Matlab che stiamo utilizzando).

Esercizio 6

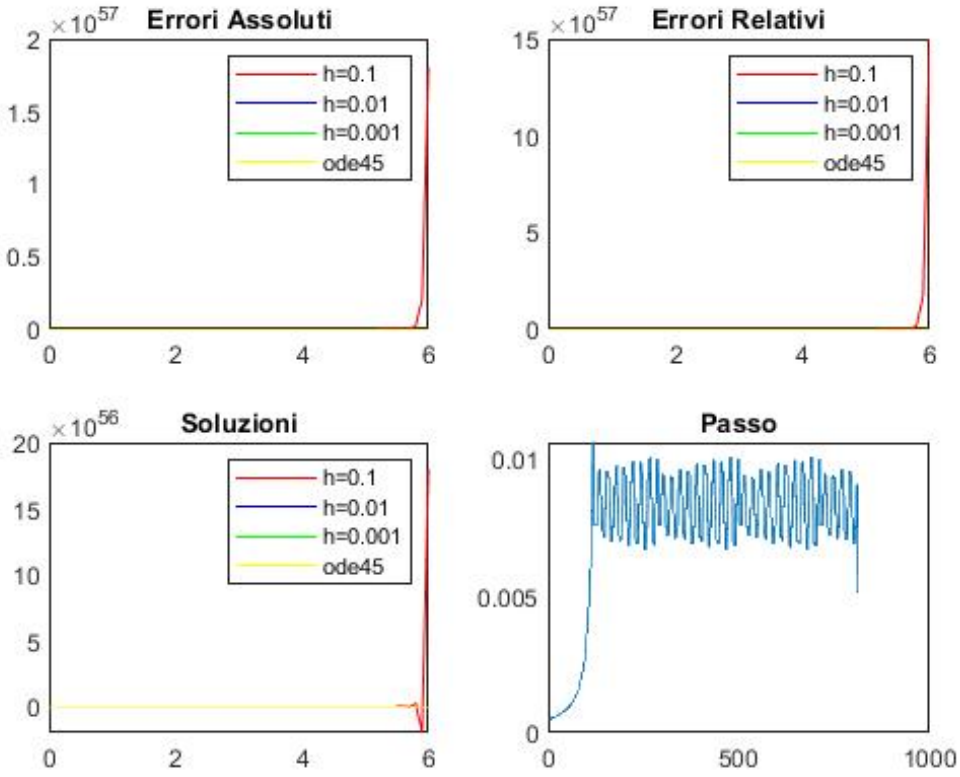
La function seguente confronta quattro metodi di risoluzione di equazioni differenziali: metodo di Eulero con $h = 0.1, 0.01, 0.001$ e `ode45` con il parametro `RelTol` $= 10^{-7}$. Li applicheremo ad un'equazione dipendente da un parametro a, che setteremo prima a 1 e poi a 100.

```
function esercizio6(a)
f=@(x,y) -a*y+2*x;
sol=@(x) (1+2/a^2)*exp(-a*x)-2/a^2+2/a*x;
tspan=[0,6];
init=1;
[x1,u1]=eulero(f,tspan,init,0.1);
[x2,u2]=eulero(f,tspan,init,0.01);
[x3,u3]=eulero(f,tspan,init,0.001);
options=odeset('RelTol',1e-7);
[x,u]=ode45(f,tspan,init,options);
passo=x(2:end)-x(1:end-1);
er_abs1=abs(u1-sol(x1));
er_rel1=er_abs1./sol(x1);
er_abs2=abs(u2-sol(x2));
er_rel2=er_abs2./sol(x2);
er_abs3=abs(u3-sol(x3));
er_rel3=er_abs3./sol(x3);
er_absode=abs(u-sol(x));
er_relode=er_absode./sol(x);
subplot(2,2,1)
plot (x1,er_abs1,'r',x2,er_abs2,'b',x3,er_abs3,'g',x,er_absode,'y');
legend('h=0.1','h=0.01','h=0.001','ode45');
title('Errori Assoluti');

subplot(2,2,2)
plot (x1,er_rel1,'r',x2,er_rel2,'b',x3,er_rel3,'g',x,er_relode,'y');
legend('h=0.1','h=0.01','h=0.001','ode45');

title('Errori Relativi');

subplot(2,2,3)
plot (x1,u1,'r',x2,u2,'b',x3,u3,'g',x,u,'y');
legend('h=0.1','h=0.01','h=0.001','ode45');
title('Soluzioni');

subplot(2,2,4)
plot (passo);
title('Passo');
end
```

Nel grafico abbiamo mostrato, oltre agli errori relativi e agli errori assoluti richiesti, i grafici delle soluzioni ottenute e il variare del passo del metodo usato (ultimo grafico, in blu). Per $a = 1$ l'errore del metodo di Eulero diminuisce con il diminuire di h, mentre l'errore di `ode45` sembra trascurabile. Per $a = 100$ la soluzione per $h = 0.1$ ha un comportamento già sostanzialmente incontrollato, e tutti gli altri valori di h non sembrano dare informazioni significative.


Errori Assoluti
Errori Relativi
Soluzioni
Passo
h=0.1
h=0.01
h=0.001
ode45

$a = 1$


Errori Assoluti
×10^57
2
1.5
1
0.5
0
0
2
4
6
h=0.1
h=0.01
h=0.001
ode45
Errori Relativi
×10^57
15
10
5
0
0
2
4
6
h=0.1
h=0.01
h=0.001
ode45
Soluzioni
×10^56
20
15
10
5
0
0
2
4
6
h=0.1
h=0.01
h=0.001
ode45
Passo
0.01
0.005
0
0
500
1000

$a = 100$