Metodi numerici
Corso di LSMC, a.a. 2017-2018

Davide Gori
550282

October 22, 2018

1 Prima sperimentazione: un classico problema

Considero il seguente problema:

$$\begin{cases} y' = -y, \quad x \in (0, 10] \\ y(0) = 1 \end{cases} \tag{1}$$

la cui soluzione esatta è: $y(x) = e^{-x}$. Risolvo numericamente il problema mediante il metodo di Eulero. e utilizzando il comando subplot, suddivido un figura in quattro parti. In ciascuna di esse disegno la soluzione esatta e la soluzione numerica ottenuta per i valori del passo di integrazione $h = 0.5, 1, 2, 2.5$, rispettivamente. In ciascun grafico identifico le soluzioni disegnate utilizzando il comando legend.

1.1 Il codice

Questo è lo script che realizza la sperimentazione:

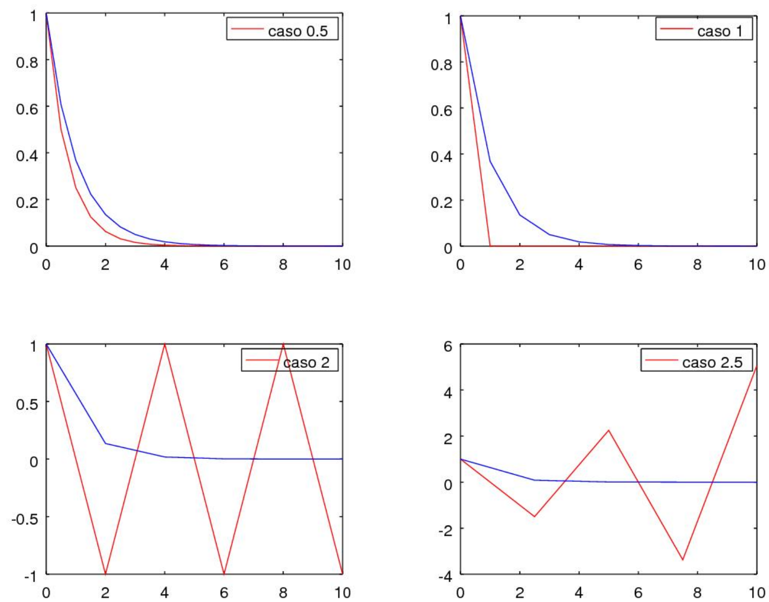
```
slot=[0, 10];
y0=1;
B=[0.5, 1, 2, 2.5]
for i=1:4
    h=B(i);
    [x, u]=eulero(@fun2,slot,y0,h);
    sol_esatta=e.^(-x);
    subplot(2, 2, i);
    plot(x, u, 'r');
    hold on
    subplot(2, 2, i);
    plot(x, sol_esatta, 'b');
    legend(sprintf("caso␣%d",B(i)))
    drawnow;
end
```

Dove `fun2.m` è la seguente:

```
function r=fun2(x, y)
r=-y;
end
```

1.2 Risultati

Riportiamo il grafico in output.



Se uso un passo troppo grande ottengo che la funzione esce da [0, 1] e si ha un comportamento a "zig-zag" dovuto al fatto che la derivata cambia segno ad ogni passo

2 Seconda sperimentazione: metodo di Runge-Kutta

Scriviamo un file di tipo function che implementi su una griglia uniforme il metodo di Runge-Kutta classico per la risoluzione del problema ai valori iniziali:

$$\begin{cases} \boldsymbol{y'} = \boldsymbol{f}(x, \boldsymbol{y}), \quad x \in [x_0, T] \\ y(x_0) = y_0 \end{cases} \tag{2}$$

con $\boldsymbol{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$.
Applicheremo poi la function `RK4.m` al problema precedente, con h = 1, 2, 2.5,

3.

2.1 Il codice

Questa è la funzione `RK4.m`:

```
function [x,u] = RK4(odefun,tspan,y0,h)
dim=length(y0);
x=tspan(1):h:tspan(2);
n=length(x);
u=zeros(n, dim);
u(1,:)=y0;
for i=2:n
    F1=odefun(x(i-1), u(i-1,:));
    F2=odefun(x(i-1)+h/2, u(i-1,:)+h*F1/2);
    F3=odefun(x(i-1)+h/2, u(i-1,:)+h*F2/2);
    F4=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
```

Questo è lo script che risolve il problema precedente con il metodo di Runge-Kutta:

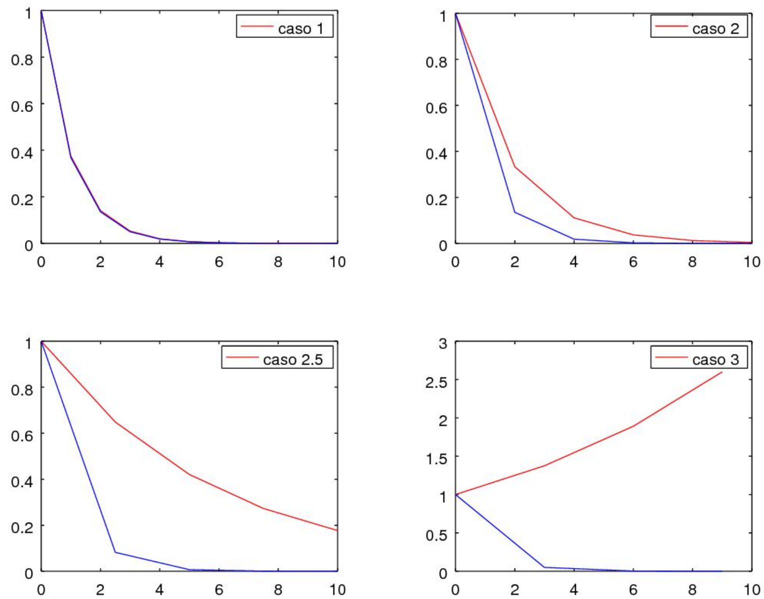
```
slot=[0, 10];
y0=1;
B=[1, 2, 2.5, 3];
for i=1:4
    h=B(i);
    [x, u]=RK4(@fun2,slot,y0,h);
    sol_esatta=e.^(-x);
    subplot(2, 2, i);
    plot(x, u, 'r');
    hold on
    subplot(2, 2, i);
    plot(x, sol_esatta, 'b');
    legend(sprintf("caso %d",B(i)))
    drawnow;
end
```

Dove `fun2.m` è la seguente:

```
function r=fun2(x, y)
r=-y;
end
```

2.2 Risultati

Riportiamo il grafico in output.



Si nota che usando RK4 la soluzione è più precisa, in particolare non si verifica il fenomeno ”zig-zag” che accadeva nella precedente sperimentazione.

3 Terza sperimentazione: applicazione di Runge-Kutta

Risolviamo con il metodo di Runge-Kutta classico il seguente problema ai valori iniziali:

$$\begin{cases} y'' = -\frac{4x+1}{2(x+1)}y' + \frac{2x-1}{4x^2} \cdot \frac{3y^3+y}{y^2+1}, \quad x \in [1,2] \\ y'(1) = 1 \\ y(1) = 0 \end{cases} \tag{3}$$

Applicheremo poi la function `RK4.m` con $h = 0.01$.

3.1 Il codice

Questo è lo script che risolve il problema con il metodo di Runge-Kutta:

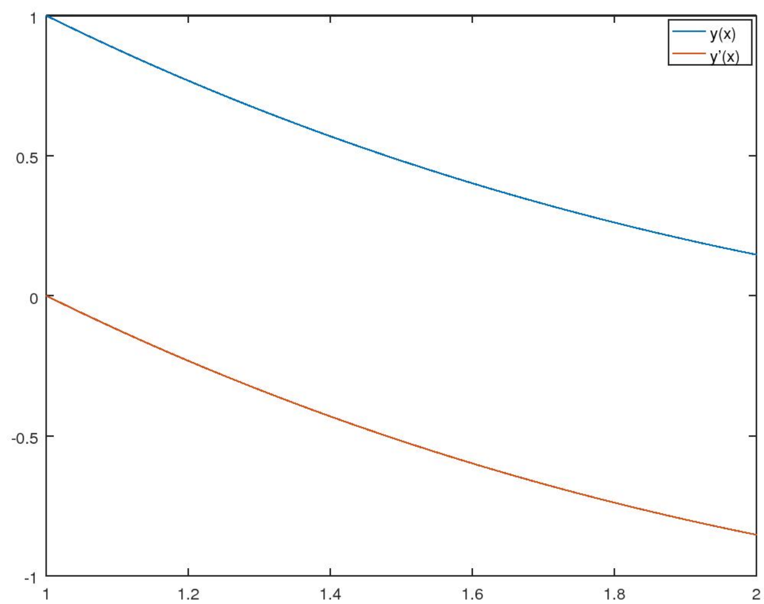
```
slot=[1, 2];
y0=[1, 0];
h=0.01;
[x, u]=RK4(@fun3,slot,y0,h);
plot(x, u)
legend("y(x)","y'(x)")
```

Dove `fun3.m` è la seguente:

```
function [a, b]=fun3(x, y)
b=y(1);
a=-(4*x+1)/(2*x+2)*y(1)+(2*x-1)/(4*x*x) *(3*y(2)^3+y(2))/(1+y(2)^2);
end
%funzione f(t, x1, x2)
```

3.2 Risultati

Riportiamo il grafico in output.



4 Quarta sperimentazione: confronto tra Runge-Kutta e Eulero

Consideriamo il seguente problema ai valori iniziali:

$$\begin{cases} y' = -y - 5e^{-t}\sin(5t), & x \in [0, 5] \\ y(0) = 1 \end{cases} \tag{4}$$

la cui soluzione esatta è $y = e^{-t}\cos(5t)$. Stimeremo l'ordine di convergenza sia del metodo di Eulero che del metodo di RungeKutta classico, calcolando l'errore commesso per $h = \frac{1}{10k}$ con $k = 1, \ldots, 10$. Rappresentando gli errori in un grafico opportuno.

4.1 Il codice

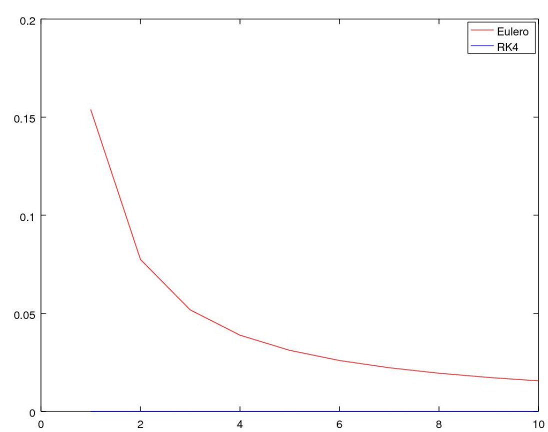
Questo è lo script che risolve il problema precedente con il metodo di Runge-Kutta:

```
slot=[0, 5];
y0=1;
err=zeros(10, 2);
for i=1:10
    h=1/(10*i);
    [x, u]=eulero(@fun4,slot,y0,h);
    [x, v]=RK4(@fun4,slot,y0,h);
    sol_esatta=e.^(-x) .* cos(5.*x);
    err(i, 1)=max(abs(u-sol_esatta));
    err(i, 2)=max(abs(v'-sol_esatta));
end
err
hold on
plot(1:10, err(:,1), 'r');
plot(1:10, err(:,2));
legend("Eulero","RK4")
```

Dove `fun4.m` è la seguente:

```
function r=fun4(x, y)
r=-y-5*e^(-x)*sin(5*x);
end
```

4.2 Risultati



Riportiamo il grafico in output. Notiamo che il metodo di Eulero è molto meno preciso di RK4.

i	eulero	RK4
i=1	1.5386e-01	2.3929e-05
i=2	7.7457e-02	1.4690e-06
i=3	5.1750e-02	2.8876e-07
i=4	3.8870e-02	9.1275e-08
i=5	3.1132e-02	3.7356e-08
i=6	2.5960e-02	1.8002e-08
i=7	2.2259e-02	9.7114e-09
i=8	1.9481e-02	5.6899e-09
i=9	1.7319e-02	3.5508e-09
i=10	1.5589e-02	2.3294e-09

5 Sesta sperimentazione: confronto tra metodi numerici

Fissato $h = 0.1, 0.01, 0.001$, calcolo l'errore relativo e l'errore assoluto che si commettono nell'approssimare con il metodo di Eulero i due problemi ai valori iniziali

$$\begin{cases} y'(x) = -\alpha y + 2x, \quad x \in [0, 6] \\ y(0) = 1 \end{cases} \tag{5}$$

ottenuti per $1\alpha = 1$ e $\alpha = 100$, sapendo che la soluzione esatta è

$$y = \left(1 + \frac{2}{\alpha^2}\right) e^{-\alpha x} + \frac{2}{\alpha} x - \frac{2}{\alpha^2}$$

Confronterò poi gli errori ottenuti con quelli applicando la routine `ode45` con `RelTol=10-7`. Inoltre, per entrambi i valori di α, rappresenterò in un grafico il passo di integrazione adattivo determinato da `ode45`.

5.1 Il codice

Questo è lo script usato:

```
%format long
slot=[0, 6];
y0=1;
err=zeros(3, 2);
errabs=zeros(4, 2);
for j=1:2
    if j==1, a=1; b=0; else a=100; b=3; end
    fun4 = @(x, y) -a*y+2*x;
    for i=1:3
        h=10^(-i);
        [x, u]=eulero(fun4,slot,y0,h);
        sol_esatta=(1+2/(a^2))*(e.^(-a.*x)).+2/a*x -2/(a^2);
        err(i, j)=max(abs(u-sol_esatta));
```

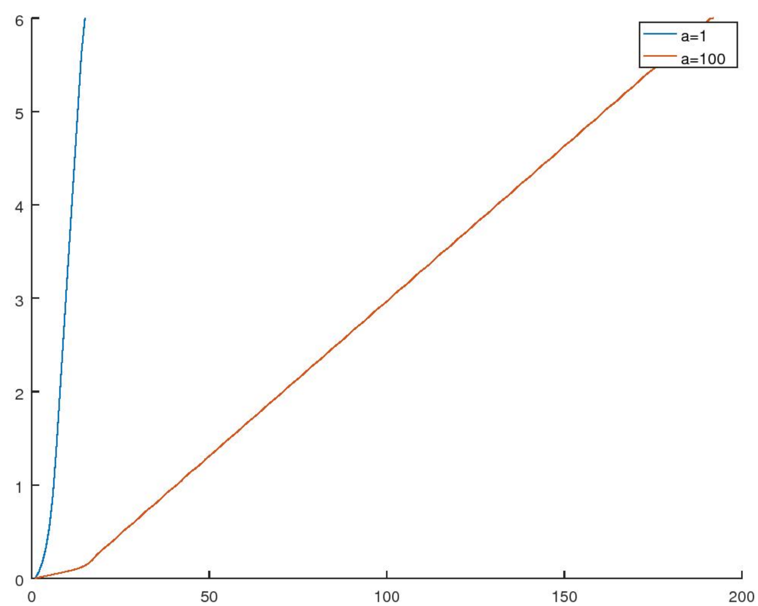
```
        errabs(i, j)=max(abs((u-sol_esatta)./sol_esatta));
    end
    %confronto con ode45 (errori ultima riga matrice)
    [xc,yc]=ode45(@(x,y) (-a*y+2*x),slot,y0,"RelTol",0.0000001);
    sol_esatta=(1+2/(a^2))*(e.^(-a.*xc)).+2/a*xc -2/(a^2);
    err(4, j)=max(abs(yc-sol_esatta));
    errabs(4, j)=max(abs((yc-sol_esatta)./sol_esatta));
    hold on
    plot(1:length(xc), xc);
    length(xc)
    xc
    legend("a=1","a=100");
end
err
errabs
```

Dove `fun4.m` è la seguente:

```
function r=fun4(x, y)
r=-y-5*e^(-x)*sin(5*x);
end
```

5.2 Risultati

Riportiamo il grafico in output.



Confronto con ode45: notiamo che il metodo di Eulero è molto meno preciso di

ode45.
I valori sono i seguenti:
Errore assoluto:

h	a=1	a=100
h=E-1	5.7603e-02	1.7974e+57
h=E-2	5.5413e-03	3.6795e-01
h=E-3	5.5205e-04	1.9205e-02
ode45	1.8360e-05	2.5751e-04

Errore relativo:

h	a=1	a=100
h=E-1	6.1663e-02	1.5003e+58
h=E-2	5.9073e-03	1.0000e+00
h=E-3	5.8806e-04	2.1113e-01
ode45	7.5516e-06	1.8043e-03

Passo ode45: Notiamo che nel caso a=100, escluso il momento iniziale, si ha che il passo è costante. Nel caso a=1 vengono eseguiti meno passi (15) ma più lunghi