Lab_8_ODE_parte_2_(traccia)

April 16, 2024

1 Lab 8 - Equazioni Differenziali Ordinarie II

1.1 Metodi di Runge-Kutta (RK)

I metodi numerici utilizzati per risolvere il generico problema di Cauchy:

$$\begin{cases} y'(t) = f(t, y), \quad t_0 < t \leq t_{\max}, \\ y(t_0) = y_0, \end{cases}$$

si basano sulla seguente strategia:

1. Stabilire un passo di avanzamento temporale h,
2. Suddividere l'intervallo temporale $[t_0, t_{\max}]$ in un numero N_h di sottointervalli

$$N_h = \frac{t_{\max} - t_0}{h}$$

   di eguale ampiezza h,
3. Per ogni istante temporale discreto t_n, con $t_0 < t_n < t_{\max}$, si calcola il valore incognito u_n che approssima la soluzione $y_n = y(t_n)$.

L'insieme dei valori $\{u_0 = y_0,\ u_1, \dots, u_{N_h}\}$ rappresenta la soluzione numerica del problema di Cauchy.

Il metodo di Runge-Kutta è un metodo a un passo, ma rispetto a Eulero in avanti consente di raggiungere un ordine di accuratezza più elevato. Ad ogni passo, la soluzione numerica è calcolata secondo la seguente formula:

$$\begin{cases} u_0 = y_0, \\ u_{n+1} = u_n + \dfrac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4), \quad n = 0, \dots, N_h, \end{cases}$$

dove

$$\begin{aligned} K_1 &= f(t_n, u_n), \\ K_2 &= f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2}K_1\right), \\ K_3 &= f\left(t_n + \frac{h}{2}, u_n + \frac{h}{2}K_2\right), \\ K_4 &= f(t_{n+1}, u_n + hK_3). \end{aligned}$$

In particolare, questo è un metodo esplicito di ordine 4 rispetto ad h e, ad ogni passo temporale, richiede quattro valutazioni di f. Una variante di questo metodo, che utilizza anche un passo di integrazione variabile, è implementata nella function $\texttt{RK45}$ della libreria Python `scipy.integrate`.

1.2 Esplorazione numerica

Si consideri il problema di Cauchy

$$\begin{cases} y'(t) = -ty(t), \quad t_0 < t \leq t_{\max}, \\ y(t_0) = 1, \end{cases}$$

con $t_0 = 0$ e $t_{\max} = 5$.

Esercizio 1.1

Si risolva il problema di Cauchy con il metodo di Eulero in avanti,utilizzando la function `eulero_avanti` presente nello script `utilities_ODE2.py`, con passo $h = 0.01$ e si calcoli l'errore assoluto

$$e_h = \max_{n=1,\dots,N_h} |y(t_n) - u_n|.$$

NB: a differenza della scorsa implementazione, la function fornita, "eulero_avanti", si aspetta che il dato iniziale y_0 sia un vettore di lunghezza d, dove d è la dimensione del problema (qui, $d = 1$). Eventualmente, aiutatevi anche con l'help di Python.

[ ]:
```
import numpy as np
f = lambda t, y : -t*y
y0 = 1
t0 = 0
t_max = 5

yex = lambda t : np.exp(-t**2/2)
```

[ ]:
```
from utilities_ODE2 import eulero_avanti
help(eulero_avanti)
```

[ ]:
```

```

[ ]:
```
errEA = np.max(np.abs(yEA[:, 0] - yex(tEA)))
nstepsEA = len(tEA)-1

print("Errore globale EA:\t%.2e" % errEA)
print("Passi effettuati:\t%d" % nstepsEA)
```

Esercizio 1.2

Si risolva ora il problema con il meotodo di Runge-Kutta. Per farlo, si sfrutti la function $\texttt{RK23}$ della libreria Python `scipy.integrate` e si calcoli l'errore assoluto. (Si fissino tolleranza relativa ed assoluta a 10^{-8}). Si confrontino quindi gli errori assoluti dei due metodi (RK23 ed EA) rispetto al

numero di istanti temporali utilizzati. NB: la function RK23 implementa un metodo adattivo basato su due metodi di Runge-Kutta: uno di ordine 3 (per effettuare il passo) ed uno di ordine 2 (per scegliere, ad ogni step, il nuovo passo temporale h_n). Inoltre, diversamente dalle implementazioni che abbiamo visto finora, questa function non simula direttamente tutta la traiettoria: invece, ci restituisce un oggetto Python che può essere evoluto in tempo chiamando il metodo `.step()`. In sostanza, RK23 implementa l'iteratore dello schema numerico.

In ultimo: in ogni momento potete accedere agli attributi `.t` e `.y` dell'iteratore per conoscere, rispettivamente, il tempo corrente t_n e lo stato corrente del sistema, y_n.

[ ]:
```
from scipy.integrate import RK23
```

[ ]:
```

```

[ ]:
```
errRK23 = np.max(np.abs(yRK23[:, 0] - yex(tRK23)))
nstepsRK23 = len(tRK23) - 1.0

print("Errore globale RK23:\t%.2e" % errRK23)
print("Passi effettuati:\t%d" % nstepsRK23)
```

Esercizio 1.3

Si ripeta l'Esercizio 1.2 utilizzando un metodo di Runge-Kutta di ordine superiore, implementato nella function `RK45`, della libreria Python `scipy.integrate` e si calcoli l'errore assoluto. Cosa si può concludere? NB: tale function implementa un metodo adattivo basato su due metodi di Runge-Kutta: uno di ordine 5 (per effettuare il passo) ed uno di ordine 4 (per scegliere,ad ogni step, il nuovo passo temporale h_n). Come prima, l'output di RK45 consiste in un oggetto Python, il quale può essere evoluto in tempo chiamando il metodo `.step()`.

[ ]:
```
from scipy.integrate import RK45
```

Esercizio 1.4

Confrontate graficamente soluzione esatta e approssimazioni numeriche (utilizzate quelle ottenute ai punti 1.1, 1.2 ed 1.3).

2 ODE di ordine superiore al primo

Tutto quello che abbiamo fatto finora si generalizza facilmente al contesto vettoriale, dove, in sostanza, non abbiamo più una singola quantità evolvente nel tempo, ma una collezione di d variabili $\mathbf{y}(t) \in \mathbb{R}^d$. Ciò è molto utile anche per modellizzare fenomeni scalari la cui dinamica sia descritta da equazioni di ordine superiore: ad esempio, una ODE del secondo ordine nella variabile x.

$$x''(t) = f(t, x(t), x'(t))$$

si può riscrivere come

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t))$$

avendo posto $\mathbf{y}'(t) := [x(t), \quad x'(t)]^\top$ il vettore di posizione e velocità, mentre $\mathbf{f}(\mathbf{y}) := [y_2, \ f(t, y_1, y_2)]^\top$.

3 Esempio numerico

Si consideri l'equazione che descrive l'oscillatore armonico smorzato e forzato data da:

$$\begin{cases} m\ddot{x} = -kx - \gamma\dot{x} + f_0\cos(\Omega t), \quad t_0 < t \leq t_{\max}, \\ \dot{x}(t_0) = v_0, \\ x(t_0) = x_0, \end{cases}$$

dove m è la massa dell'oggetto attaccato alla molla, k la costante elastica della molla, γ il coefficiente di smorzamento e $f_0\cos(\Omega t)$ è un termine forzante di ampiezza f_0 e frequenza Ω. La pulsazione dell'oscillazione è definita come

$$\omega = \sqrt{\frac{k}{m}}.$$

Esercizio 2.1

Si scriva l'equazione dell'oscillatore armonico come sistema di equazioni differenziali ordinarie del primo ordine.

Soluzione

In questo caso abbiamo

$$\begin{cases} \mathbf{y}'(t) &= \mathbf{f}(t, y), \\ \mathbf{y}(0) &= \mathbf{y}_0. \end{cases}$$

dove $\mathbf{y} = [x(t), \ \dot{x}(t)]^\top$ è un vettore colonna di due componenti, il dato iniziale è $\mathbf{y}_0 = [x_0, \ v_0]^\top$, mentre la funzione $\mathbf{f}$ è data da

$$\mathbf{f}(t, \mathbf{y}) = \left[y_2, \quad -\frac{k}{m}y_1 - \frac{\gamma}{m}y_2 + \frac{f_0}{m}\cos(\Omega t)\right]^\top$$

Equivalentemente,

$$\mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} 0 & 1 \\ -\dfrac{k}{m} & -\dfrac{\gamma}{m} \end{bmatrix} \cdot \mathbf{y} + \begin{bmatrix} 0 \\ \dfrac{f_0}{m}\cos(\Omega t) \end{bmatrix}$$

Esercizio 2.2

Si risolva il problema differenziale ottenuto con Eulero in avanti. A tale scopo, si utilizzi la funzione `eulero_avanti` che è stata opportunamente modificata in modo da gestire correttamente anche il caso vettoriale. Si approssimi la soluzione per $t_0 = 0$, $t_{\max} = 1$, $m = 1$, $k = 100$, $v_0 = 0$, $x_0 = 1$, $\gamma = 0$, $f_0 = 0$.

Si confronti graficamente la soluzione esatta $x(t) = \cos(10t)$ con quella ottenuta ponendo $h = 0.001$, $h = 0.01$ e $h = 0.1$. Si commentino i risultati.

[ ]:
```python
import numpy as np

# Dati del problema
t0 = 0
t_max = 1
x0 = 1
v0 = 0
m = 1
k = 100

# Soluzione esatta
u_ex = lambda t : np.cos(10*t)
```

[ ]:
```python

```

Esercizio 2.3

Si risolva il problema differenziale ottenuto con Eulero all'indietro. A tale scopo, si utilizzi la funzione `eulero_indietro_sis_lineari` che è stata opportunamente modificata in modo da gestire correttamente anche il caso vettoriale, sotto l'ipotesi che $\mathbf{f}$ sia rappresentabile attraverso una matrice. Si approssimi la soluzione per $t_0 = 0$, $t_{\max} = 1$, $m = 1$, $k = 100$, $v_0 = 0$, $x_0 = 1$, $\gamma = 0$, $f_0 = 0$. Si confronti graficamente la soluzione esatta $x(t) = \cos(10t)$ con quella ottenuta ponendo $h = 0.001$, $h = 0.01$ e $h = 0.1$. Si commentino i risultati.

[ ]:
```python
from utilities_ODE2 import eulero_indietro_sis_lineari

# Dati del problema
t0 = 0
t_max = 1
x0 = 1
v0 = 0
m = 1
k = 100
gamma = 0

# Soluzione esatta
u_ex = lambda t : np.cos(10*t)
```

Esercizio 2.4 (Per casa)

Si ponga $t_0 = 0$, $t_{\max} = 10$, $m = 1$, $k = 4$, $v_0 = 1$, $x_0 = 0$. Mediante la funzione RK45 di Python si sperimentino i seguenti casi, visualizzandone la soluzione numerica e discutendo i risultati ottenuti alla luce dei risultati teorici.

1. Oscillatore armonico semplice: $\gamma = f_0 = 0$. La soluzione esatta è

$$x(t) = A\cos(\omega t + \theta_0),$$

dove

$$A = -\frac{v_0}{\sin(\theta_0)\omega}, \quad \theta_0 = \arctan\left(\frac{v_0}{\omega x_0}\right).$$

2. Oscillatore armonico sovra-smorzato: $f_0 = 0$, $\gamma^2 > 4mk$. Si assuma $\gamma = 5$. La soluzione esatta è

$$x(t) = c_0 e^{\lambda_0 t} + c_1 e^{\lambda_1 t},$$

dove

$$\lambda_k = \frac{-\gamma + (-1)^k \sqrt{\gamma^2 - 4mk}}{2m}, \qquad c_0 = x_0 - c_1, \qquad c_1 = \frac{v_0 - x_0\lambda_0}{\lambda_1 - \lambda_0}$$

3. Oscillatore armonico sotto-smorzato: $f_0 = 0$, $\gamma^2 < 4mk$. Si assuma $\gamma = 1$. La soluzione esatta è

$$x(t) = e^{-\frac{\gamma}{2m}t} \left(A\cos(\omega_1 t) + B\sin(\omega_1 t) \right),$$

dove

$$\omega_1 = \frac{\sqrt{4mk - \gamma}}{2m}, \quad A = x_0, \quad B = \frac{v_0}{\omega_1} + \frac{\gamma x_0}{2m\omega_1}.$$

4. Oscillatore armonico forzato: $\gamma = 1$, $f_0 = 1$, $\Omega = 0.5$. In questo caso si ponga $t_{\max} = 30$.

Extra!

Confrontare le soluzioni numeriche ottenute nell'Esercizio 2.4, rappresentandole nel piano delle fasi, cioè nel piano posizione-velocità, $(x, \dot{x})$.