Lab 7 - ODE - parte 1 (completa)

April 4, 2024

1 Equazioni Differenziali Ordinarie (ODE) - Parte 1: Metodi di Eulero

Consideriamo l'evoluzione in tempo di una certa quantità scalare $y = y(t) \in \mathbb{R}$, la cui dinamica sia descritta ad un problema di Cauchy del tipo

$$\begin{cases} y'(t) = f(t, y(t)) \quad t_0 \leq t \leq t_f, \\ y(0) = y_0. \end{cases}$$

Per poter risolvere numericamente il problema suddividiamo l'intervallo $I = [t_0, t_N]$ in N sottointervalli con la stessa dimensione data da

$$h = \frac{t_N - t_0}{N}$$

e definiamo gli istanti discreti $\{t_0, t_1, \dots, t_N\}$ in cui calcoleremo la soluzione approssimata come

$$t_n = t_0 + nh.$$

Lo scopo di un metodo numerico per l'equazioni differenziali ordinarie (ODE) è quello di approssimare la dinamica di y con una sequenza discreta $\{u_0 = y_0, u_1, \dots, u_N\}$ tale per cui

$$u_n \approx y(t_n).$$

1.1 1. Il metodo di Eulero in avanti

L'idea del metodo di Eulero in avanti è quella di costruire una griglia temporale equispaziata di passo $h > 0$ ed approssimare la soluzione del sistema secondo lo schema iterativo sottostante

$$\begin{cases} u_0 = y_0 \\ u_{n+1} = u_n + hf(t_n, u_n) \quad n = 0, \dots, N-1 \end{cases}$$

Tale schema è detto *esplicito* e ad un passo (*one-step*), in quanto, ad ogni passo temporale, la soluzione numerica u_{n+1} dipende soltanto dalla soluzione al passo temporale precedente u_n.

Esercizio 1 Scrivere una funzione chiamata `euleroAvanti` che, dati f, t_0, t_N, y_0 ed h, implementi il metodo di Eulero in avanti, restituendo la griglia temporale $\mathbf{t} = [t_0, \dots, t_N]$ e la sequenza approssimante $\mathbf{u} = [u_0, \dots, u_N]$.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[ ]:
```
def euleroAvanti(f, t0, tN, y0, h):
  """Metodo di Eulero in avanti
  Input:
    f   (lambda function)   Termine di destra dell'ODE, passata come
                            funzione di tempo e spazio, f = f(t, y)
    t0  (float)             Tempo iniziale
    tN  (float)             Tempo finale
    y0  (float)             Dato iniziale
    h   (float)             Passo temporale

  Output:
    t   (numpy.ndarray)     Griglia temporale
    u   (numpy.ndarray)     Approssimazioni della soluzione nei nodi temporali␣
 ↪t_i
  """
  # primi valori della lista
  t = [t0]
  u = [y0]

  # implementazione eulero in avanti
  while t[-1]<tN:
    u.append(u[-1]+h*f(t[-1], u[-1]))
    t.append(t[-1]+h)

  t = np.array(t)
  u = np.array(u)

  return t, u
```

1.2 2. Il metodo di Eulero all'indietro

In alternativa al metodo di Eulero "in avanti", esiste anche una sua versione "all'indietro", la quale si basa invece sul seguente schema iterativo

$$\begin{cases} u_0 = y_0 \\ u_{n+1} = u_n + hf(t_{n+1}, u_{n+1}) \quad n = 0, \dots, N-1 \end{cases}$$

Tale schema è detto *implicito* e ad un passo (*one-step*), in quanto, ad ogni passo temporale, la soluzione numerica u_{n+1} dipende dalla stessa incongita u_{n+1}, oltre che da u_n. Quindi, di volta in volta, bisognerà risolvere un'equazione non lineare

$$z = u_n + hf(t_{n+1}, z)$$

nella variabile incognita z. La soluzione di tale equazione si può approssimare, ad esempio, con il metodo del punto fisso. Si osservi che l'utilizzo di un metodo di punto fisso richiede che il

modulo della derivata della funzione di iterazione utilizzata sia minore di uno; questa condizione è generalmente soddisfatta per h sufficientemente piccolo.

Esercizio 2

Importate la function `euleroIndietro` dallo script `utils_ODE.py`. Aiutandovi con l'help di Python, o ispezionando direttamente il codice, verificate che la function in questione accetta i medesimi input di `euleroAvanti`, e che anch'essa restituisce i vettori **t** e **u**.

[ ]:
```
# importazione Eulero all'indietro
from utilis_ODE import euleroIndietro
```

1.3 3. Applicazione a un caso esempio

Si consideri il problema di Cauchy

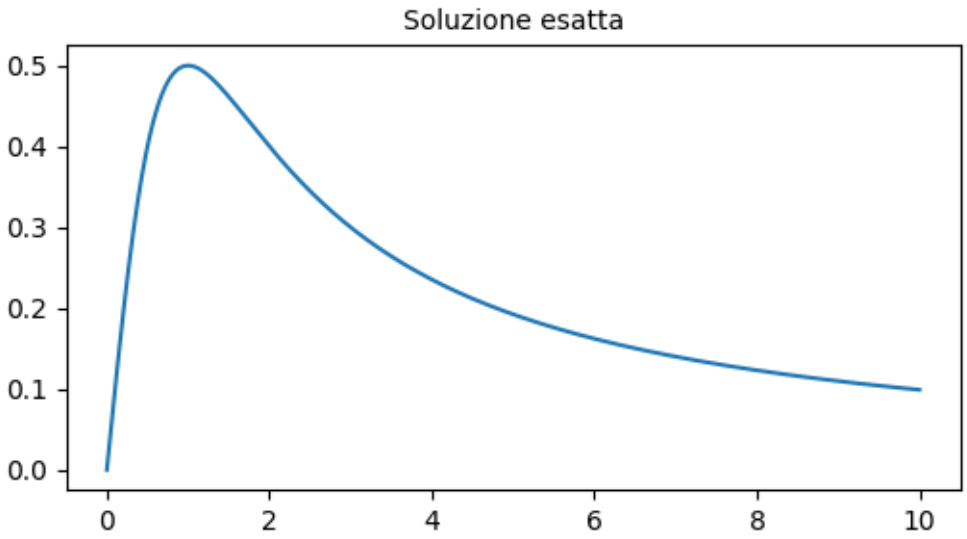
$$\begin{cases} y'(t) = \dfrac{1}{1+t^2} - 2y(t)^2 & 0 \le t \le 10, \\ y(0) = 0, \end{cases}$$

la cui soluzione esatta è

$$y(t) = \frac{t}{1+t^2}.$$

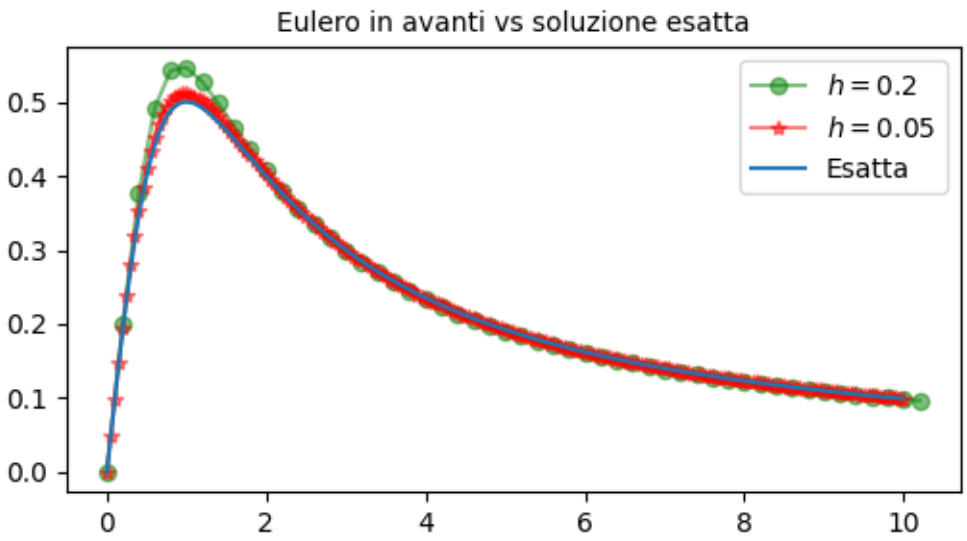
Esercizio 3.1 Rappresentare graficamente la soluzione esatta nell'intervallo considerato.

[ ]:
```
# Es. 3.1 - Plot soluzione esatta
# tempo iniziale e finale
t0, tN = 0, 10
# t per la rappresentazione grafica
tplot = np.linspace(t0, tN, 1000)
# soluzione esatta
yesatta = lambda t: t/(1.0+t**2)

plt.figure(figsize = (6,3))
plt.plot(tplot, yesatta(tplot))
plt.title("Soluzione esatta", fontsize = 10)
plt.show()
```



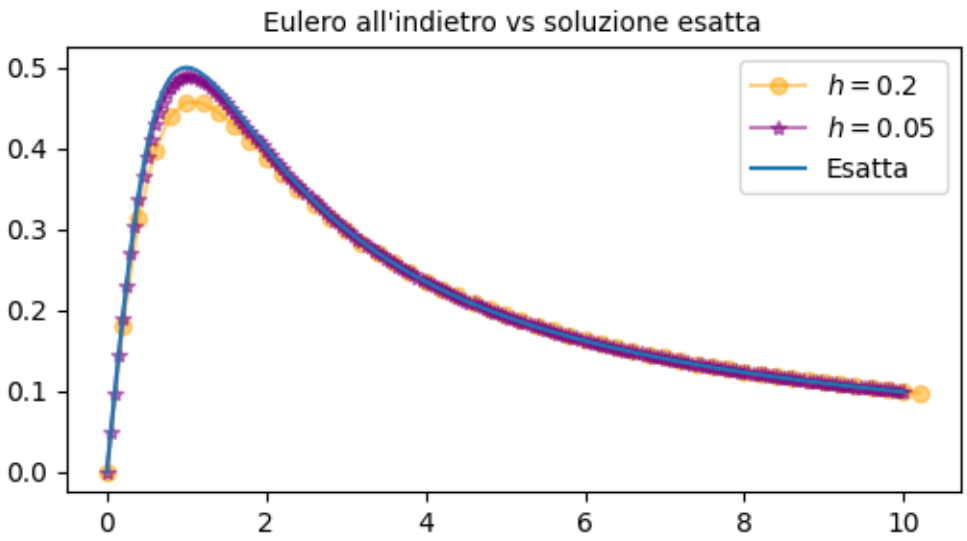
Esercizio 3.2 Approssimare la soluzione del problema utilizzando il metodo di *Eulero in avanti*, prima con $h = 0.2$ e poi con $h = 0.05$. Confrontare graficamente le soluzioni numeriche con la soluzione esatta;

[ ]:
```
# Es. 3.2 - Eulero in avanti per h = 0.2, 0.05
# dato iniziale
y0 = 0
# funzione f del problema di cauchy che dipende dal tempo e dalla y
f = lambda t, y: 1.0/(1+t**2) - 2*y*y

t1, u1 = euleroAvanti(f, t0, tN, y0, h = 0.2)
t2, u2 = euleroAvanti(f, t0, tN, y0, h = 0.05)

plt.figure(figsize = (6,3))
plt.plot(t1, u1, '-o', label = '$h=0.2$', alpha = 0.5, color = 'green')
plt.plot(t2, u2, '-*', label = '$h=0.05$', alpha = 0.5, color = 'red')
plt.plot(tplot, yesatta(tplot), label = 'Esatta')
plt.title("Eulero in avanti vs soluzione esatta", fontsize = 10)
plt.legend()
plt.show()
```



Esercizio 3.3 Ripetere il l'**Esercizio 3.2** utilizzando il metodo di Eulero all'indietro. Come cambiano i risultati?

[ ]:
```
# Es. 3.3 - Eulero all'indietro per h = 0.2, 0.05
# dato iniziale
y0 = 0
# funzione f del problema di cauchy che dipende dal tempo e dalla y
f = lambda t, y: 1.0/(1+t**2) - 2*y*y

t1, u1back = euleroIndietro(f, t0, tN, y0, h = 0.2)
t2, u2back = euleroIndietro(f, t0, tN, y0, h = 0.05)

plt.figure(figsize = (6,3))
plt.plot(t1, u1back, '-o', label = '$h=0.2$', alpha = 0.5, color = 'orange')
plt.plot(t2, u2back, '-*', label = '$h=0.05$', alpha = 0.5, color = 'purple')
plt.plot(tplot, yesatta(tplot), label = 'Esatta')
plt.title("Eulero all'indietro vs soluzione esatta", fontsize = 10)
plt.legend()
plt.show()
```



Esercizio 3.4 I confronti grafici sono molto utili, ma è altrettanto utile avere un riscontro quantitativo. Per valutare la bontà dell'approssimazione $u_n \approx y(t_n)$ sull'intera griglia temporale, si può calcolare l'errore globale

$$e_h = \max_{n=0,\dots,N} |y(t_n) - u_n|,$$

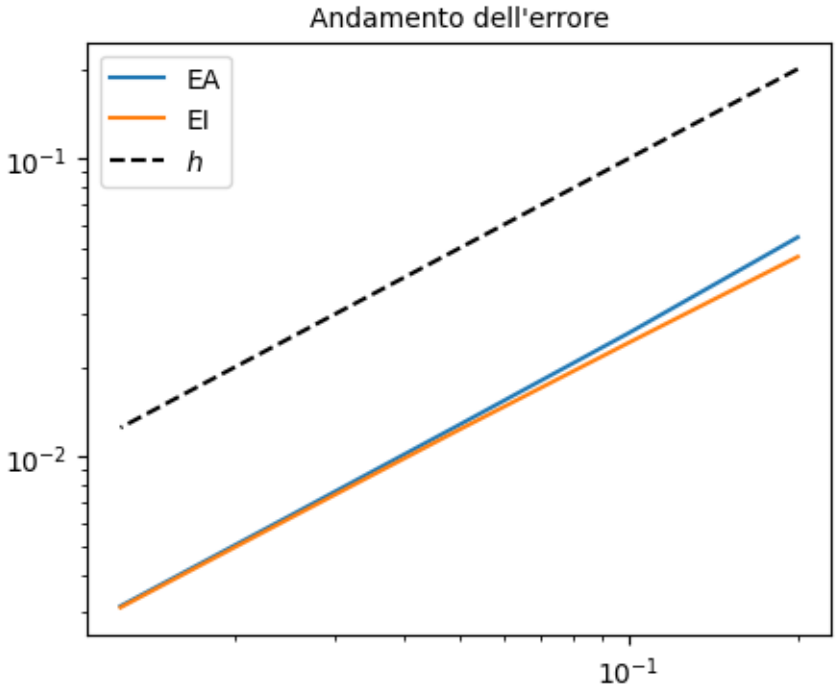
dove $e = e_h$ enfatizza il fatto che l'errore dipende dal passo scelto $h > 0$. Calcolare l'errore globale per i metodi di Eulero in avanti e all'indietro quando $h = 0.2, 0.05$.

[ ]:
```
# Es. 3.4 - Errori globali a confronto

e1 = abs(yesatta(t1)-u1).max() # Eulero in avanti (h = 0.2)
e2 = abs(yesatta(t2)-u2).max() # Eulero in avanti (h = 0.05)

e1back = abs(yesatta(t1)-u1back).max() # Eulero all'indietro (h = 0.2)
e2back = abs(yesatta(t2)-u2back).max() # Eulero all'indietro (h = 0.05)

print("Metodo\t\tErrore\t\tPassi\n" + "-"*37)
print("EA (h=0.2)\t%.2e\t%d" % (e1, len(t1)))
print("EI (h=0.2)\t%.2e\t%d\n" % (e1back, len(t1)))
print("EA (h=0.05)\t%.2e\t%d" % (e2, len(t2)))
print("EI (h=0.05)\t%.2e\t%d" % (e2back, len(t2)))
```

```
Metodo          Errore          Passi
-------------------------------------
EA (h=0.2)      5.45e-02        52
EI (h=0.2)      4.68e-02        52

EA (h=0.05)     1.28e-02        201
EI (h=0.05)     1.23e-02        201
```

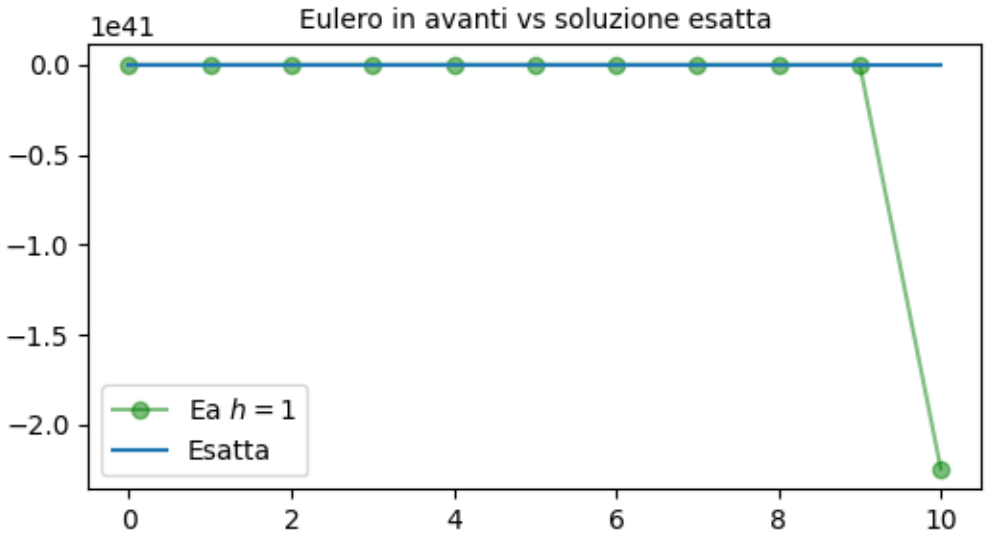
Esercizio 3.5 Calcolare gli errori e_h al variare di $h = 0.2, 0.1, 0.05, 0.025, 0.0125$, quindi rappresentarne l'andamento utilizzando un opportuno plot in scala logaritmica. I risultati sono coerenti con la teoria?

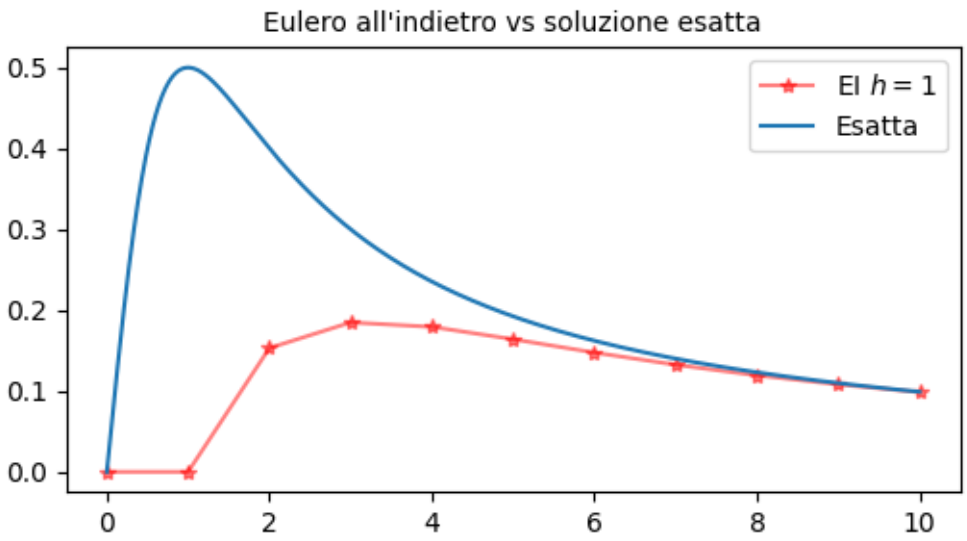
[ ]:
```
# Es. 3.5 - Ordini di convergenza
# passo h ed errori
hs, errEA, errEI = [], [], []

for k in range (5):
  # passo h
  hs.append(0.2/2**k)
  th, uh = euleroAvanti(f, t0, tN, y0, hs[-1])
  errEA.append(abs(yesatta(th)-uh).max())

  th, uh = euleroIndietro(f, t0, tN, y0, hs[-1])
  errEI.append(abs(yesatta(th)-uh).max())

plt.figure(figsize = (5, 4))
plt.loglog(hs, errEA, label = 'EA')
plt.loglog(hs, errEI, label = 'EI')
plt.loglog(hs, hs, '--k', label = '$h$')
plt.legend()
plt.title("Andamento dell'errore", fontsize = 10)
plt.show()
```



Esercizio 3.6 Risolvere il problema con i metodi di Eulero in avanti e di Eulero all'indietro, scegliendo un passo h=1. Riportare su grafici differenti le soluzioni numeriche ottenute, confrontandole con la soluzione esatta e calcolare gli errori ottenuti. Cosa si osserva?

[ ]:
```
# Eulero in avanti
tEA, uEA = euleroAvanti(f, t0, tN, y0, h = 1)

plt.figure(num=1, figsize = (6,3))
plt.plot(tEA, uEA, '-o', label = 'Ea $h=1$', alpha = 0.5, color = 'green')
plt.plot(tplot, yesatta(tplot), label = 'Esatta')
plt.title("Eulero in avanti vs soluzione esatta", fontsize = 10)
plt.legend()

# Eulero all'indietro
tEI, uEI = euleroIndietro(f, t0, tN, y0, h = 1)

plt.figure(num=2, figsize = (6,3))
plt.plot(tEI, uEI, '-*', label = 'EI $h=1$', alpha = 0.5, color = 'red')
plt.plot(tplot, yesatta(tplot), label = 'Esatta')
plt.title("Eulero all'indietro vs soluzione esatta", fontsize = 10)
plt.legend()
plt.show()


print("l'errore dell'Eulero in avanti è: %e " %max(abs(yesatta(tEA)-uEA)))
print("l'errore dell'Eulero all'indietro è: %e " %max(abs(yesatta(tEI)-uEI)))
```






```
l'errore dell'Eulero in avanti è: 2.246814e+41
l'errore dell'Eulero all'indietro è: 5.000000e-01
```

2 Esercizi per casa

Esercizio 4 In generale, il metodo di Eulero all'indietro è più oneroso (computazionalmente parlando) rispetto alla sua variante in avanti. In aggiunta, più h è piccolo, e più la simulazione richiede tempo (bisogna fare più passi!). Per quantificare tutto ciò, provare a misurare il tempo d'esecuzione richiesto dai due metodi al variare di h. *Hint: potete usare la funzione* `perf_counter` *del pacchetto* `time` *per misurare i tempi d'esecuzione del calcolatore.*

[ ]:
```
from time import perf_counter
# passo h
hs, timeEA, timeEI =[], [], []
for k in range (5):
  hs.append(0.2/2**k)
  tEA_start=perf_counter()
  th, uh = euleroAvanti(f, t0, tN, y0, hs[-1])
  tEA_stop=perf_counter()

  timeEA.append(tEA_stop-tEA_start)

  tEI_start=perf_counter()
  th, uh = euleroIndietro(f, t0, tN, y0, hs[-1])
  tEI_stop=perf_counter()
  timeEI.append(tEI_stop-tEI_start)

print("time Eulero in avanti\t time Eulero all'indietro")
```

```
for i in range(5):
  print("   %e \t\t %e" %(timeEA[i], timeEI[i]))
```

```
time Eulero in avanti    time Eulero all'indietro
   1.170900e-04                7.630580e-04
   1.613870e-04                1.110270e-03
   2.833990e-04                1.898894e-03
   5.585686e-03                5.656085e-03
   1.092793e-03                1.052219e-02
```

Esercizio 5 Un ulteriore metodo *implicito* ad un passo (*one-step*) è il metodo di Crank-Nicolson che possiede il seguente schema iterativo

$$\begin{cases} u_0 = y_0 \\ u_{n+1} = u_n + \frac{h}{2}[f(t_n, u_n) + f(t_{n+1}, u_{n+1})] \quad n = 0, \dots, N-1. \end{cases}$$

A differenza dei metodi di Eulero in avanti e di Eulero all'indietro, che hanno convergenza lineare questo metodo ha un andamento quadratico.

Considerando sempre il problema di Cauchy

$$\begin{cases} y'(t) = \dfrac{1}{1+t^2} - 2y(t)^2 \quad 0 \le t \le 10, \\ y(0) = 0, \end{cases}$$

la cui soluzione esatta è

$$y(t) = \frac{t}{1+t^2}.$$

Calcolare gli errori

$$e_h = \max_{n=0,\dots,N} |y(t_n) - u_n|,$$

al variare di $h = 0.2, 0.1, 0.05, 0.025, 0.0125$ e rappresentarne l'andamento utilizzando un opportuno plot in scala logaritmica. I risultati sono coerenti con la teoria?

[ ]:
```
from utilis_ODE import crankNicolson
```

[ ]:
```
# Es. 5 - Ordine di convergenza Crank-Nicolson
# passo h ed errori
hs, hs2, errCN = [], [], []
t0, tN = 0, 10
f = lambda t, y: 1.0/(1+t**2) - 2*y*y
for k in range (5):
  hs.append(0.2/2**k)
  hs2.append((0.2/2**k)**2)
  th, uh = crankNicolson(f, t0, tN, y0, hs[-1])
  errCN.append(abs(yesatta(th)-uh).max())

plt.figure(figsize = (5, 4))
plt.loglog(hs, errCN, label = 'CN')
```

```
plt.loglog(hs, hs, '--k', label = '$h$')
plt.loglog(hs, hs2, ':k', label = '$h^2$')
plt.legend()
plt.title("Andamento dell'errore", fontsize = 10)
plt.show()
```