

# Lab 6 - Approssimazione di integrali e derivate (soluzione completa)

March 12, 2024

## 1 Lab 6 - Approssimazione numerica di integrali e derivate

### 2 Formule di quadratura (approssimazione di integrali)

Le formule di quadratura sono tecniche numeriche volte ad approssimare integrali del tipo

$$\int_a^b f(s)ds$$

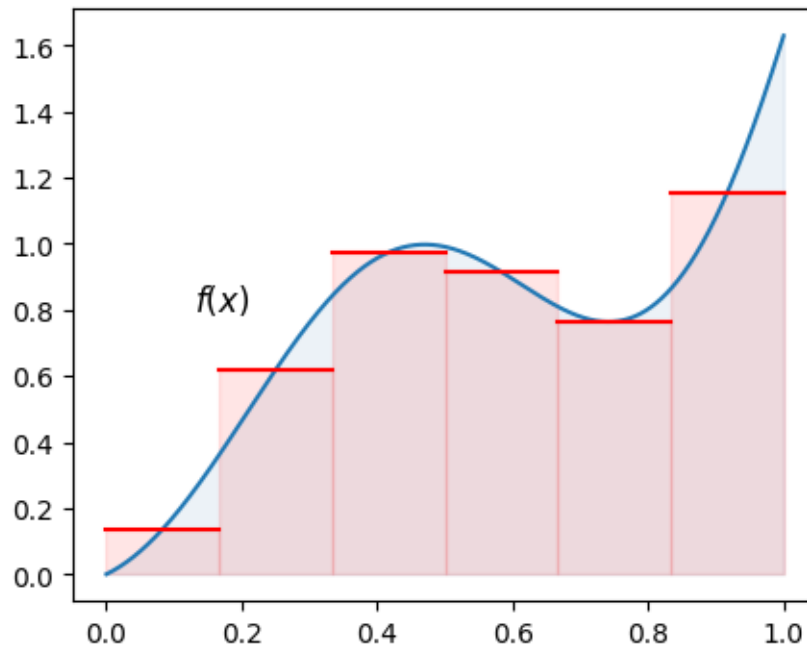
per mezzo di valutazioni *puntuali* dell'integranda  $f$ . In altre parole, una volta fissata un opportuna griglia  $\{x_0 < x_1 < \dots < x_n\} \subset [a, b]$ , l'idea è quella di combinare in maniera opportuna i valori  $\{(x_i, f(x_i))\}$  per approssimare l'integrale in questione. Molto spesso, queste formule vengono costruire rimpiazzando  $f$  con opportune interpolanti/approssimanti il cui integrale sia facile da calcolare. In quanto segue, assumiamo di utilizzare una griglia uniforme di passo  $h > 0$ .

#### 2.0.1 Metodo del punto medio (composito)

$$\int_a^b f(s)ds \approx \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right)(x_{i+1} - x_i) = h \sum_{i=0}^{n-1} f\left(x_i + \frac{h}{2}\right).$$

Geometricamente parlando, si sta approssimando l'area sottesa da  $f$  con una serie di rettangoli (equivalentemente, stiamo sostituendo  $f$  con una sua approssimazione **costante a tratti**).

```
[ ]: from niceplots import show
show("Punto medio")
```

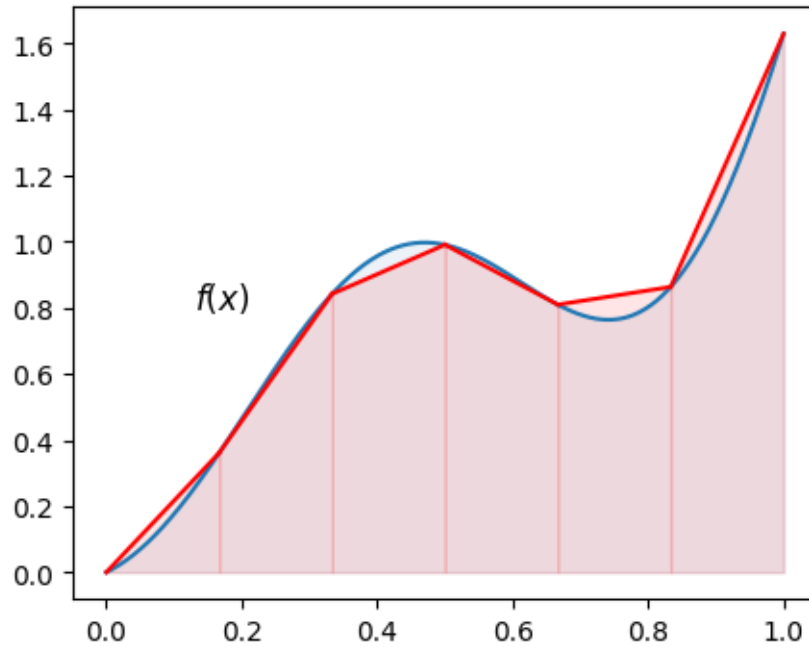


### 2.0.2 Metodo dei trapezi (composito)

$$\int_a^b f(s)ds \approx \sum_{i=0}^{n-1} \frac{1}{2} [f(x_i) + f(x_{i+1})] (x_{i+1} - x_i) = \frac{h}{2} \sum_{i=0}^{n-1} [f(x_i) + f(x_{i+1})].$$

Geometricamente parlando, si sta approssimando l'area sottesa da  $f$  con una serie di trapezi (equivalentemente, stiamo sostituendo  $f$  con una sua approssimazione **lineare a tratti**).

```
[ ]: show("Trapezi")
```

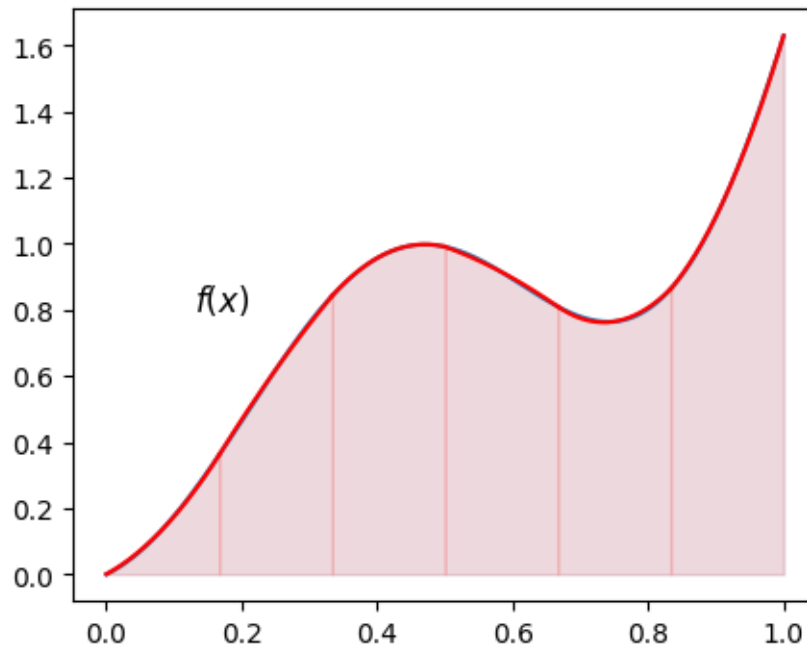


### 2.0.3 Metodo di Cavalieri-Simpson (composito)

$$\int_a^b f(s)ds \approx \frac{h}{6} \sum_{i=0}^{n-1} \left[ f(x_i) + 4f\left(\frac{x_i + x_{i+1}}{2}\right) + f(x_{i+1}) \right].$$

Geometricamente parlando, si sta approssimando l'area sottesa da  $f$  con una serie di “sotto parabole” (equivalentemente, stiamo sostituendo  $f$  con una sua approssimazione **quadratica a tratti**).

```
[ ]: show("Simpson")
```



**Esercizio 1** Scrivere tre funzioni, chiamate *pmedcomp*, *trapcomp* e *simpcomp*, che implementano rispettivamente le formule di quadratura composta del punto medio, del trapezio e di Simpson su intervalli equispaziati. Tali funzioni dovranno ricevere in ingresso gli estremi di integrazione  $a$  e  $b$ , il numero di sottointervalli  $N$  in cui si vuole suddividere il dominio di integrazione, e la funzione  $f$  da integrare; in uscita, dovranno restituire il valore approssimato dell'integrale.

```
[ ]: import numpy as np

[ ]: def pmedcomp(f, a, b, N):
    """ Formula del punto medio composta
    Input:
        f:   funzione da integrare
        a:   estremo inferiore intervallo di integrazione
        b:   estremo superiore intervallo di integrazione
        N:   numero di sottointervalli (N = 1 formula di integrazione semplice)
    Output:
        I:   integrale approssimato """

    h = (b-a)/N                # ampiezza sottointervalli
    x = np.linspace(a, b, N+1) # griglia spaziale
    xL, xR = x[:-1], x[1:]     # liste dei nodi "sinistri" e "destri"
    xM = 0.5*(xL + xR)         # punti medi
    I = h*f(xM).sum()          # integrale approssimato

    return I
```

```
[ ]: def trapcomp(f, a, b, N):
    """ Formula dei trapezi composta
    Input:
        f: funzione da integrare
        a: estremo inferiore intervallo di integrazione
        b: estremo superiore intervallo di integrazione
        N: numero di sottointervalli (N = 1 formula di integrazione semplice)
    Output:
        I: integrale approssimato """

    h = (b-a)/N # ampiezza sottointervalli
    x = np.linspace(a, b, N+1) # griglia spaziale
    xL, xR = x[:-1], x[1:] # liste dei nodi "sinistri" e "destri"
    xM = 0.5*(xL + xR) # punti medi

    I = 0.5*h*(f(xL)+f(xR)).sum() # integrale approssimato

    return I
```

```
[ ]: def simpcomp(f, a, b, N):
    """ Formula di Cavalieri-Simpson composta
    Input:
        f: funzione da integrare
        a: estremo inferiore intervallo di integrazione
        b: estremo superiore intervallo di integrazione
        N: numero di sottointervalli (N = 1 formula di integrazione semplice)
    Output:
        I: integrale approssimato """

    h = (b-a)/N # ampiezza sottointervalli
    x = np.linspace(a, b, N+1) # griglia spaziale
    xL, xR = x[:-1], x[1:] # liste dei nodi "sinistri" e
    ↪ "destri"
    xM = 0.5*(xL + xR) # punti medi
    I = (h/6.0)*(f(xL)+4*f(xM)+f(xR)).sum() # integrale approssimato

    return I
```

**Esercizio 2** Utilizzando le funzioni definite all'Es. 1, approssimate il valore del seguente integrale,

$$I = \int_0^1 x^2 dx.$$

usando  $N = 20$  sottointervalli. Calcolate quindi -a mano- il vero valore del suddetto integrale e confrontatelo con le approssimazioni ottenute. Cosa osservate? Commentate opportunamente il risultato ottenuto.

```
[ ]: import numpy as np

f = lambda x: x**2
a, b = 0, 1
N = 20

# Integrali approssimati
I_pmc = pmedcomp(f, a, b, N)
I_tc = trapcomp(f, a, b, N)
I_sc = simpcomp(f, a, b, N)

# Integrale esatto
I_ex = 1/3

err_pmc = abs(I_ex - I_pmc)
err_tc = abs(I_ex - I_tc)
err_sc = abs(I_ex - I_sc)

print("Errori assoluti\n"+"-"*28)
print("Punto medio\t\t%e" % err_pmc)
print("Trapezi\t\t\t%e" % err_tc)
print("Simpson\t\t\t%e" % err_sc)
```

Errori assoluti

```
-----
Punto medio      2.083333e-04
Trapezi          4.166667e-04
Simpson          0.000000e+00
```

### 3 Formule alle differenze finite (approssimazione di derivate)

Data una funzione  $f$  ed un punto  $x_0$ , la sua derivata nel punto  $x_0$  si può approssimare attraverso le seguenti formule:

- $f'(x_0) \approx \frac{1}{h} [f(x_0 + h) - f(x_0)]$  (differenza in avanti)
- $f'(x_0) \approx \frac{1}{h} [f(x_0) - f(x_0 - h)]$  (differenza all'indietro)
- $f'(x_0) \approx \frac{1}{2h} [f(x_0 + h) - f(x_0 - h)]$  (differenza centrata)

dove  $h > 0$  è il cosiddetto *passo*.

**Esercizio 3** Sia  $f(x) = e^{-x}$ . Si calcoli -a mano- il valore della derivata di  $f$  nel punto  $x_0 = 0.25$ , quindi lo si confronti con la sua approssimazione alle differenze finite per  $h = 0.05$ . In particolare, si calcoli l'errore ottenuto con tre metodi, evidenziando quello che restituisce la miglior approssimazione.

```
[ ]: import numpy as np

x0 = 0.25
```

```

f = lambda x: np.exp(-x)
df = lambda x: -np.exp(-x)

dfEx = df(x0)

h = 0.05
dfIndietro = (f(x0)-f(x0-h))/h
dfAvanti = (f(x0+h)-f(x0))/h
dfCentrata = (f(x0+h)-f(x0-h))/(2*h)

print("Errori approssimazione derivata\n" + "-"*31)
print("Indietro\t%.4e" % abs(dfEx-dfIndietro))
print("Avanti\t\t%.4e" % abs(dfEx-dfAvanti))
print("Centrata\t%.4e" % abs(dfEx-dfCentrata))

```

Errori approssimazione derivata

```

-----
Indietro      1.9799e-02
Avanti        1.9150e-02
Centrata      3.2454e-04

```

**Esercizio 4** Ripetere l'analisi proposta all'Es. 3 facendo variare l'ampiezza del passo  $h$ . In particolare, si consideri

$$h \in \{0.2, 0.1, 0.05, 0.025, 0.0125\}.$$

Verificare graficamente l'ordine di convergenza delle singole formule come previsto dalla teoria (per farlo, utilizzare un opportuno grafico in scala logaritmica).

```

[ ]: import matplotlib.pyplot as plt

dfErrori = {'Avanti':[], 'Indietro':[], 'Centrata':[]}

hlist = np.array([0.2,0.1,0.05,0.025,0.0125])

for h in hlist:
    dfIndietro = (f(x0)-f(x0-h))/h
    dfAvanti = (f(x0+h)-f(x0))/h
    dfCentrata = (f(x0+h)-f(x0-h))/(2*h)

    dfErrori['Indietro'].append(abs(dfEx - dfIndietro))
    dfErrori['Avanti'].append(abs(dfEx - dfAvanti))
    dfErrori['Centrata'].append(abs(dfEx - dfCentrata))

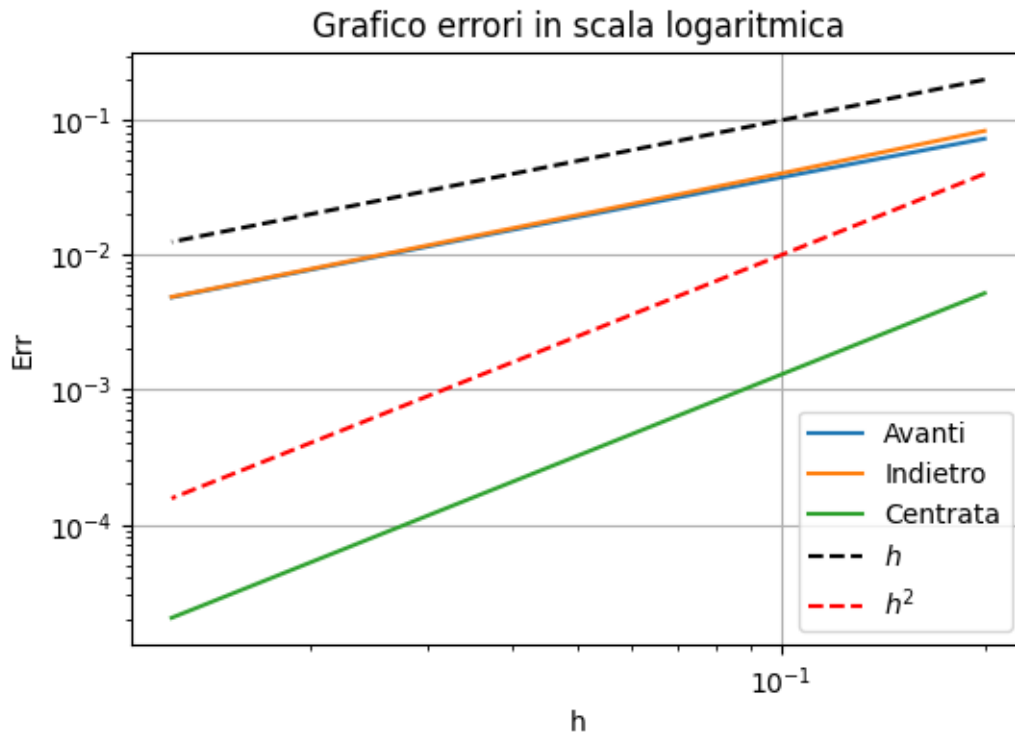
plt.figure(figsize = (6, 4))
for metodo in dfErrori.keys():

```

```

plt.loglog(hlist, dfErrori[metodo], label = metodo)
plt.loglog(hlist, hlist, 'k--', label = '$h$')
plt.loglog(hlist, hlist**2, 'r--', label = '$h^2$')
plt.legend()
plt.title('Grafico errori in scala logaritmica')
plt.xlabel('h')
plt.ylabel('Err')
plt.grid()
plt.show()

```



**Commento** (Perché usare una scala logaritmica?)

Il motivo di questa scelta è legato alla forma in cui si presentano le stime dell'errore viste a teoria. Infatti, dalla teoria sappiamo che l'errore delle formule alle differenze finite è della forma:

$$|f'(x_0) - (D_h f)(x_0)| = Ch^p,$$

dove  $h$  è il passo di discretizzazione della formula,  $D_h f$  è la discretizzazione alle differenze finite (in avanti, all'indietro o centrate), mentre  $C$  è una costante indipendente da  $h$ . Applicando il logaritmo ad ambo i membri dell'uguaglianza, possiamo riscrivere la stima come

$$\log |f'(x_0) - (D_h f)(x_0)| = \log C + p \log h.$$



Ne deduciamo quindi che, il logaritmo dell'errore dipende dal logaritmo del passo in maniera affine. In particolare, il grafico del logaritmo dell'errore, inteso come funzione di  $\log h$ , è costituito da una retta con intercetta  $\log C$  e pendenza  $p$ . Questo ci dice che per confrontare agevolmente gli ordini di convergenza delle diverse formule è sufficiente confrontare le pendenze delle rette che si ottengono in un grafico a scala logaritmica (su entrambi gli assi).

**Commento** (Dizionari in Python)

La soluzione proposta fa uso dei cosiddetti *dizionari*, oggetti simili alle liste ma che si distinguono per l'utilizzo di etichette (dette *chiavi*), al posto dei classici indici posizionali. Ovviamente, non è necessario utilizzare i dizionari per risolvere l'esercizio, si tratta solo di una piccola aggiunta volta ad espandere la vostra conoscenza del linguaggio.

La chiamata

```
carrello = {'banane':5, 'mele':4, 'pere':8}
```

definisce un dizionario, le cui *chiavi* sono le stringhe 'banane', 'mele' e 'pere'. Queste etichette, si possono utilizzare per accedere agli elementi corrispondenti. Per esempio,

```
carrello['banane']
```

restituisce 5, che è il valore corrispondente. In generale, il contenuto di un dizionario può essere qualsiasi: qui, gli elementi sono numeri interi, nell'esercizio, invece, gli elementi sono liste (liste di errori, una per ogni metodo).

L'elenco delle chiavi in un dizionario si può estrarre con il metodo “.keys()”

```
carrello.keys()
```

che qui restituisce un iterabile (sostanzialmente una lista) contenente 'banane', 'mele' e 'pere'.

**Esercizio 5** Si consideri ora una partizione uniforme dell'intervallo  $[1, 3]$  e si immagini di approssimare la derivata prima di  $f$  in ciascun punto della partizione, secondo le formule proposte sopra. Cosa si può dire delle tre diverse formule per l'approssimazione della derivata negli estremi dell'intervallo?

**Soluzione**

Con le differenze finite in avanti non si può calcolare il valore approssimato della derivata nell'estremo destro dell'intervallo, mentre con le differenze finite all'indietro non si può calcolare il valore della derivata nell'estremo sinistro. Per quanto riguarda invece le differenze finite centrate, non si possono calcolare entrambi i valori.

## 4 Esercizi per casa

**Esercizio 6** Ripetere l'analisi proposta all'Es. 4, ma avendo posto  $f(x) = 4(x - 0.25)|x - 0.25|$ , sapendo che tale funzione è derivabile in  $x_0 = 0.25$ , e che  $f'(x_0) = 0.0$ . Cosa si osserva? Perché?

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

f = lambda x: 4*(x-0.25)*abs(x-0.25)
dfEx = 0.0
```

```

dfErrori = {'Avanti':[], 'Indietro':[], 'Centrata':[]}

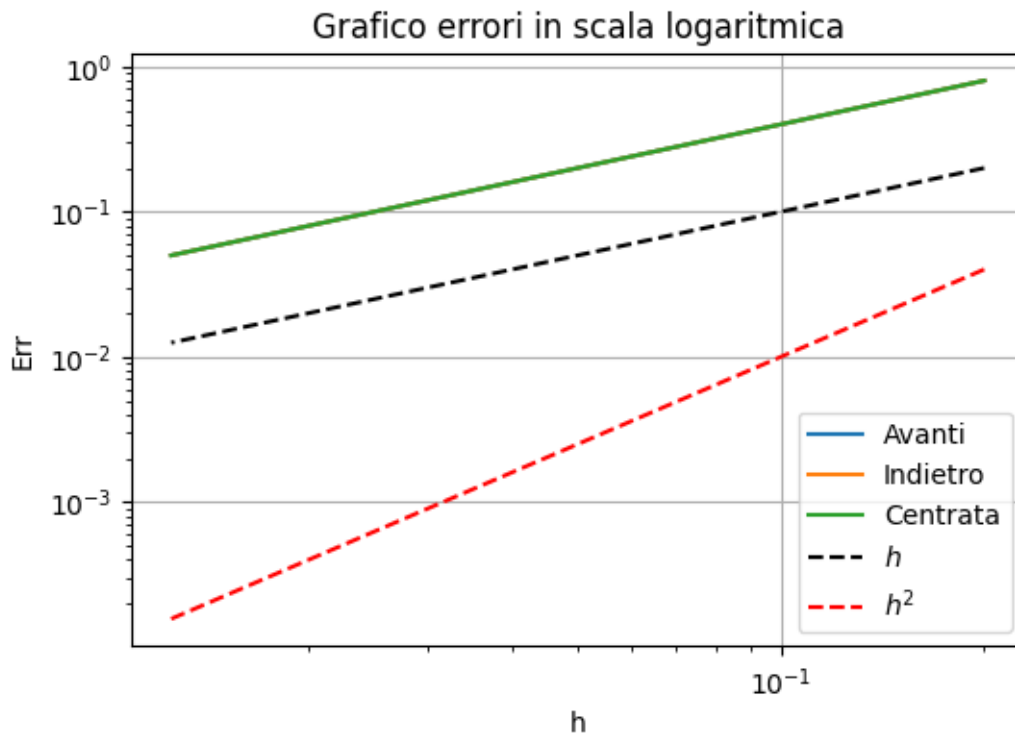
hlist = np.array([0.2,0.1,0.05,0.025,0.0125])

for h in hlist:
    dfIndietro = (f(x0)-f(x0-h))/h
    dfAvanti    = (f(x0+h)-f(x0))/h
    dfCentrata  = (f(x0+h)-f(x0-h))/(2*h)

    dfErrori['Indietro'].append(abs(dfEx - dfIndietro))
    dfErrori['Avanti'].append(abs(dfEx - dfAvanti))
    dfErrori['Centrata'].append(abs(dfEx - dfCentrata))

plt.figure(figsize = (6, 4))
for metodo in dfErrori.keys():
    plt.loglog(hlist, dfErrori[metodo], label = metodo)
plt.loglog(hlist, hlist,'k--', label = '$h$')
plt.loglog(hlist, hlist**2,'r--', label = '$h^2$')
plt.legend()
plt.title('Grafico errori in scala logaritmica')
plt.xlabel('h')
plt.ylabel('Err')
plt.grid()
plt.show()

```



## Commento

Questa volta, tutti metodi presentano ordine 1. Come mai? Perché la teoria prevede un determinato ordine di convergenza, solo se la funzione è abbastanza regolare. In particolare, affinché il metodo alle differenze finite centrate sia di ordine 2,  $f$  deve essere almeno di classe  $\mathcal{C}^2$  (nella dimostrazione, si usa lo sviluppo di Taylor!). La funzione dell'esercizio, invece, è sì derivabile, ma la sua derivata seconda è discontinua.

**Esercizio 7** Si consideri il seguente integrale,

$$I = \int_0^1 x^5 \sin(\pi x) dx,$$

il cui valore esatto è  $(120 - 20\pi^2 + \pi^4)/\pi^5$ . Si calcolino gli errori associati alle tre formule di quadratura (punto medio, trapezi, Cavalieri-Simpson) per il calcolo dell'integrale  $I$ , al variare dell'ampiezza dei sottointervalli  $h = 2^{-k}$ ,  $k = 2, \dots, 10$ .

Si verifichino quindi gli ordini di convergenza previsti dalla teoria impostando un opportuno grafico in scala logaritmica.

```
[ ]: import matplotlib.pyplot as plt
from numpy import pi
f = lambda x: (x**5)*np.sin(pi*x)

I = (120-20*pi**2 + pi**4)/(pi**5)

errori = {'Punto medio':[], 'Trapezi': [], 'Cavalieri-Simpson':[]}

for k in range(2, 11):
    h = 2.0**(-k)
    N = int(1.0/h)
    errori['Punto medio'].append(abs(I - pmedcomp(f, 0, 1, N)))
    errori['Trapezi'].append(abs(I - trapcomp(f, 0, 1, N)))
    errori['Cavalieri-Simpson'].append(abs(I - simpcomp(f, 0, 1, N)))

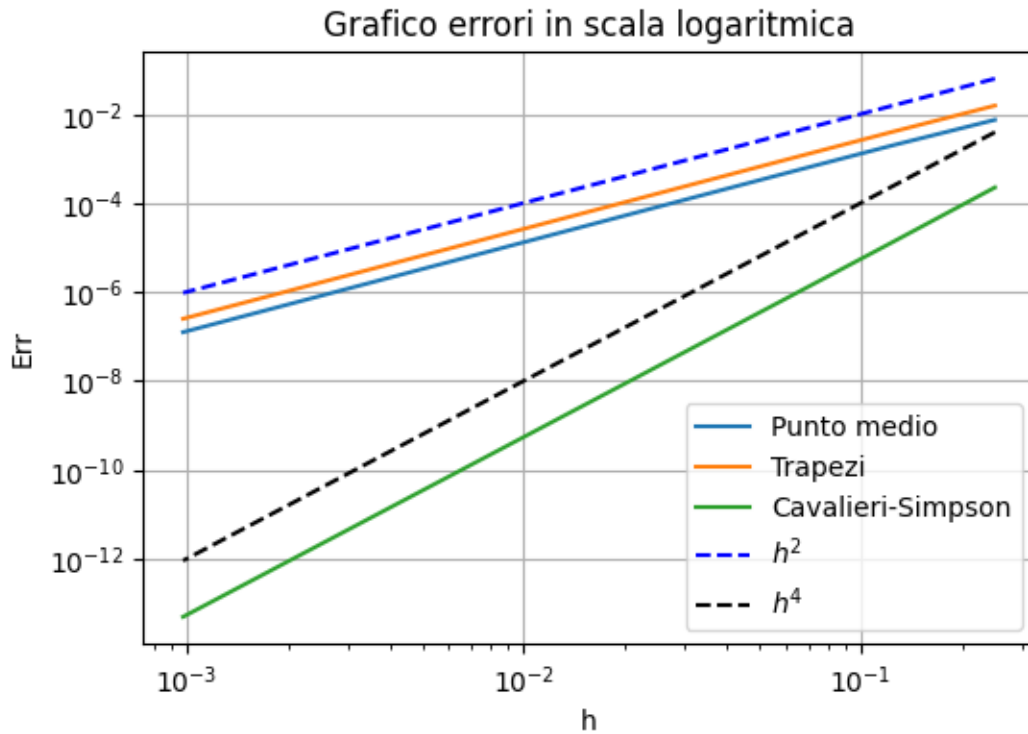
h = 2.0**(-np.arange(2, 11)) # elenco dei passi h utilizzati

plt.figure(figsize = (6, 4))

# Il ciclo for sottostante è equivalente a fare
# plt.loglog(h, errori['Punto medio'], label = 'Punto medio')
# plt.loglog(h, errori['Trapezi'], label = 'Trapezi')
# plt.loglog(h, errori['Cavalieri-Simpson'], label = 'Cavalieri-Simpson')
for metodo in errori.keys():
    plt.loglog(h, errori[metodo], label = metodo)

plt.loglog(h, h**2, '--b', label = '$h^2$')
plt.loglog(h, h**4, '--k', label = '$h^4$')
```

```
plt.title('Grafico errori in scala logaritmica')
plt.legend()
plt.grid()
plt.xlabel("h")
plt.ylabel("Err")
plt.show()
```



### Commento

Osserviamo che i metodi del Punto medio e dei Trapezi sono di ordine 2. Invece, il metodo di Cavalieri-Simpson è di ordine 4.

[ ]: