1 Calcolo zero funzione

Exercise 1.1. Scrivere una function che implementi il metodo delle approssimazioni successive per il calcolo dello zero di una funzione $f(x)$ prendendo come input una delle seguenti funzioni per l'aggiornamento:

- $g(x) = x - f(x)e^{x/2}$
- $g(x) = x - f(x)e^{-x/2}$

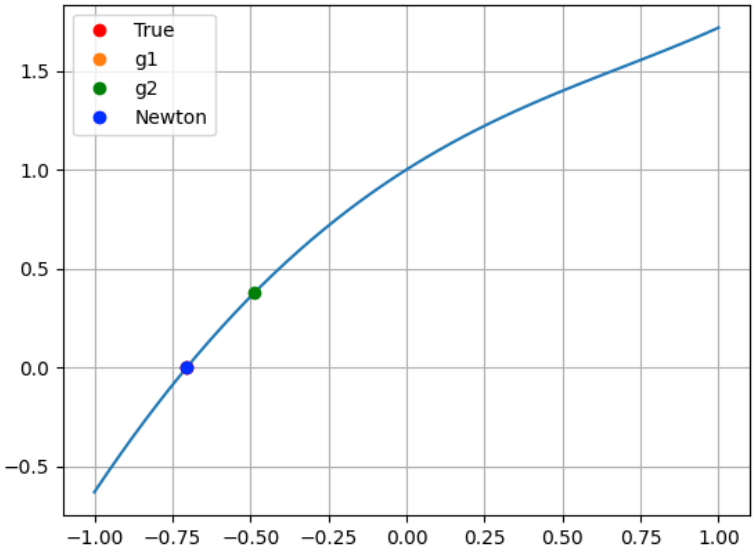
Testare il la function per trovare lo zero della funzione $f(x) = e^x - x^2$, la cui soluzione è $x^* = -0.703467$. Scrivere una function che implementi il metodo di Newton, ricordando che il metodo di Newton può essere considerato come un caso particolare del metodo delle approssimazioni successive dove la funzione di aggiornamento è $g(x) = x - f(x)/f'(x)$.

```
f = lambda x: np.exp(x) - x**2
df = lambda x: np.exp(x) - 2*x
g1 = lambda x: x - f(x) * np.exp(x / 2)
g2 = lambda x: x - f(x) * np.exp(-x / 2)

# Function approssimazioni successive
def succ_app(f, g, tolf, tolx, maxit, xTrue, x0=0):
    i = 0
    err = np.zeros(maxit + 1, dtype=np.float64)
    err[0] = tolx + 1
    vecErrore = np.zeros(maxit + 1, dtype=np.float64)
    vecErrore[0] = np.abs(xTrue- x0)
    x = x0

    while i<maxit and ((np.abs(f(x))>tolf) or err[i]>tolx):  # scarto assoluto tra iterati
        x_new = g(x)
        err[i + 1] = np.abs(x_new-x)
        vecErrore[i + 1] = np.abs(xTrue-x_new)
        i = i + 1
        x = x_new

    err = err[0:i]
    vecErrore = vecErrore[0:i]
    return (x, i, err, vecErrore)

#Metodo di Newton
def newton(f, df, tolf, tolx, maxit, xTrue, x0=0):
    g = lambda x: x- f(x)/df(x)
    (x, i, err, vecErrore) = succ_app(f, g, tolf, tolx, maxit, xTrue, x0)
    return (x, i, err, vecErrore)
```

Disegnare il grafico della funzione f nell'intervallo I = [−1, 1] e verificare che x∗ sia lo zero di f in [−1, 1], calcolare lo zero della funzione utilizzando i metodi precedentemente descritti e l'accuratezza delle soluzioni trovate e il numero di iterazioni effettuate dai solutori.

```
xTrue = -0.703467
fTrue = f(xTrue)
print("fTrue = ", fTrue)
xplot = np.linspace(-1, 1)
fplot = f(xplot)
plt.plot(xplot, fplot)

tolx = 10 ** (-10)
tolf = 10 ** (-6)
maxit = 100
x0 = 0

[sol_g1, iter_g1, err_g1, vecErrore_g1] = succ_app(f, g1, tolf, tolx, maxit, xTrue, x0)
print("Metodo approssimazioni successive g1 \n x =", sol_g1, "\n iter_new=", iter_g1)

plt.plot(sol_g1, f(sol_g1), "o", label="g1")

[sol_g2, iter_g2, err_g2, vecErrore_g2] = succ_app(f, g2, tolf, tolx, maxit, xTrue, x0)
print("Metodo approssimazioni successive g2 \n x =", sol_g2, "\n iter_new=", iter_g2)

plt.plot(sol_g2, f(sol_g2), "og", label="g2")

[sol_newton, iter_newton, err_newton, vecErrore_newton] = newton(
    f, df, tolf, tolx, maxit, xTrue, x0
)
print("Metodo Newton \n x =", sol_newton, "\n iter_new=", iter_newton)

plt.plot(sol_newton, f(sol_newton), "ob", label="Newton")
plt.legend()
plt.grid()
plt.show()
```

Output:
fTrue = 8.035078391532835e-07
Metodo approssimazioni successive g1
x = -0.7034674225096886
iter_new= 23
Metodo approssimazioni successive g2
x = -0.48775859005033334
iter_new= 100
Metodo Newton
x = -0.7034674224983917
Iter_new= 6



Si nota che il metodo delle approssimazioni non converge in 100 iterazioni, mentre gli altri 2 metodi si; inoltre, si nota un netto distacco fra il metodo di newton che usa solo 6 iterate e il metodo di approssimazioni successive g1.

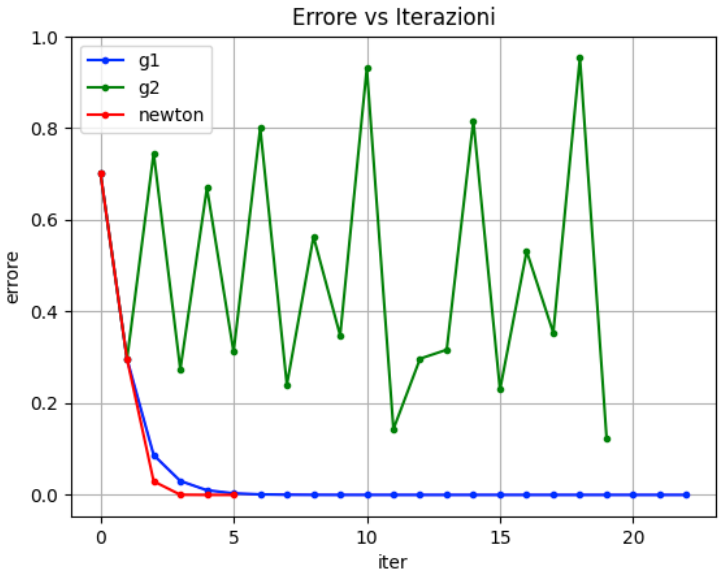
Metodo approssimazioni successive g2
x = -0.9677243597663951
iter_new= 101

Metodo approssimazioni successive g2
x = -0.48777349250060825
iter_new= 150

Metodo approssimazioni successive g2
x = -0.48777349250060825
iter_new= 5000

Metodo approssimazioni successive g2
x = -0.48777349250060825
iter_new= 100000000

Modificare le due funzioni in modo da calcolare l'errore $\|x_k - x_*\|_2$ ad ogni iterazione k-esima e graficare



È palese che con la funzione g2 la funzione risulta essere molto instabile, mentre newton approssima meglio.

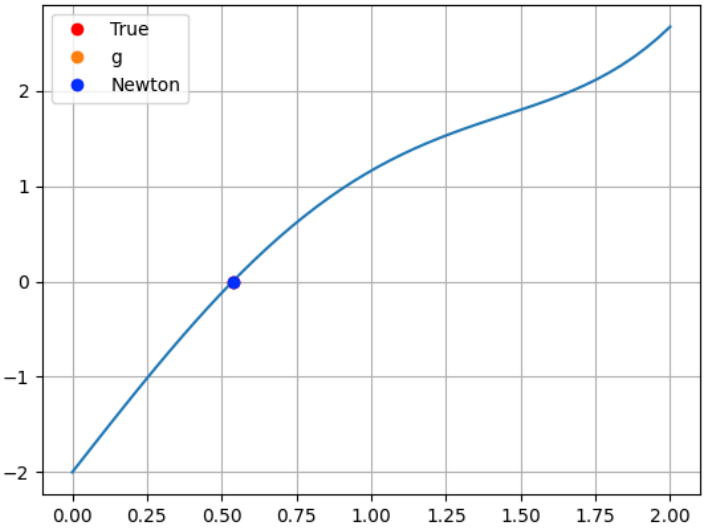
Exercise 1.2. Applicare il metodo delle approssimazioni successive e il metodo di Newton a:

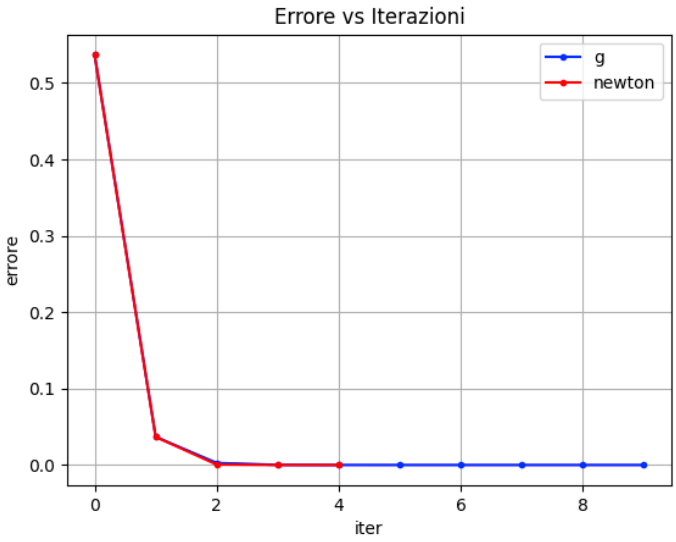
- $f(x) = x^3 + 4xcos(x) - 2$ nell'intervallo $[0, 2]$, $g(x) = \frac{2-x^3}{4cos(x)}$, con $x^* \approx 0.5369$
- $f(x) = x - x^{1/3} - 2$ nell'intervallo $[3, 5]$, $g(x) = x^{1/3} + 2$, con $x^* \approx 3.5213$

Suggerimento per l'analisi dei risultati. Confronta l'accuratezza e il numero di iterazioni dei metodi al variare del punto iniziale e dei parametri per i criteri di arresto. Spiegare il comportamento dei metodi nei diversi casi

```
f = lambda x: x**3 + 4*x*np.cos(x) - 2
df = lambda x: 3*x**2 + 4*np.cos(x) - 4*x*np.sin(x)
g = lambda x: (2-x**3)/(4*np.cos(x))
xTrue = 0.5369
```

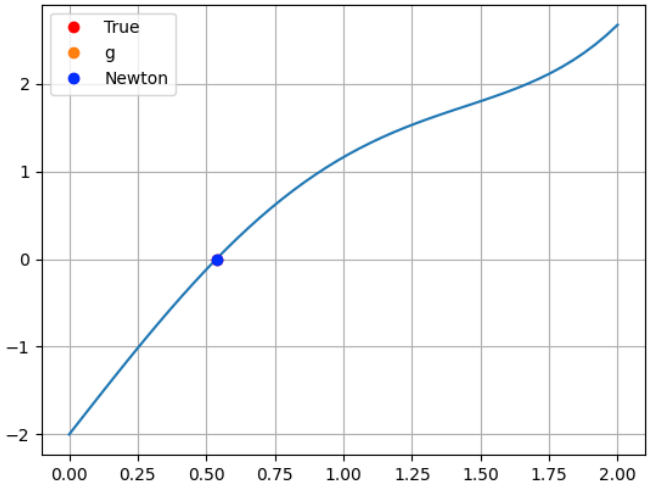
Output con
x0 = 0:





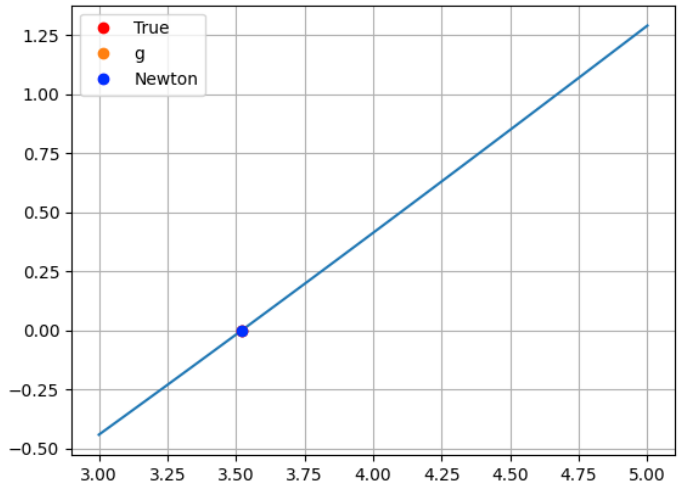
fTrue = 0.00019685736043539848
Metodo approssimazioni successive g
x = 0.5368385515655376
iter_new= 10
Metodo Newton
x = 0.5368385515667755
iter_new= 5

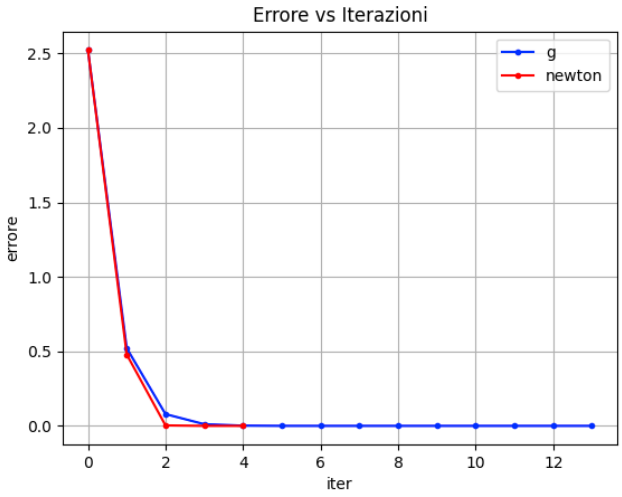
x0 = 2:




```
fTrue =  0.00019685736043539848
/Users/andrea/Unibo/Calcolo numerico/Python_Script/Es5/es1 copy 2.py:36: RuntimeWarning:
overflow encountered in scalar power
  f = lambda x: x**3 + 4*x*np.cos(x) - 2
/Users/andrea/Unibo/Calcolo numerico/Python_Script/Es5/es1 copy 2.py:38: RuntimeWarning:
overflow encountered in scalar power
  g = lambda x: (2-x**3)/(4*np.cos(x))
/Users/andrea/Unibo/Calcolo numerico/Python_Script/Es5/es1 copy 2.py:36: RuntimeWarning:
invalid value encountered in cos
  f = lambda x: x**3 + 4*x*np.cos(x) - 2
/Users/andrea/Unibo/Calcolo numerico/Python_Script/Es5/es1 copy 2.py:38: RuntimeWarning:
invalid value encountered in cos
  g = lambda x: (2-x**3)/(4*np.cos(x))
Metodo approssimazioni successive g
 x = nan
 iter_new= 9
Metodo Newton
 x = 0.5368385515667756
 iter_new= 7
```

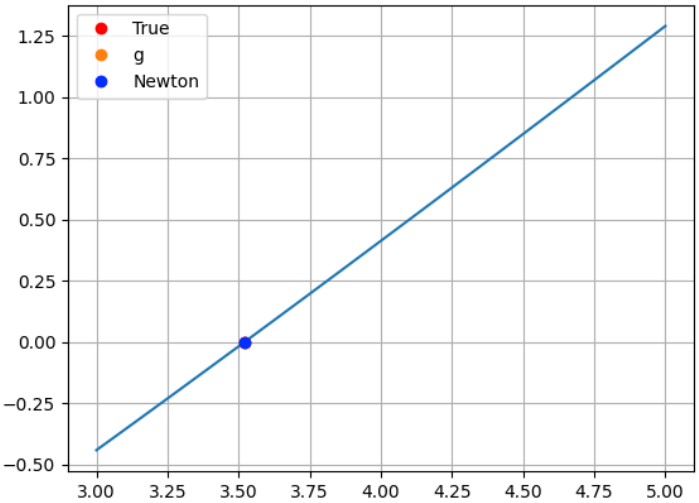
```
f = lambda x: x - x**(1/3) - 2
df = lambda x: 1 - (1/(3*np.cbrt(x**2)))
g = lambda x: x**(1/3) + 2
xTrue = 3.5213
x0 = 1
```

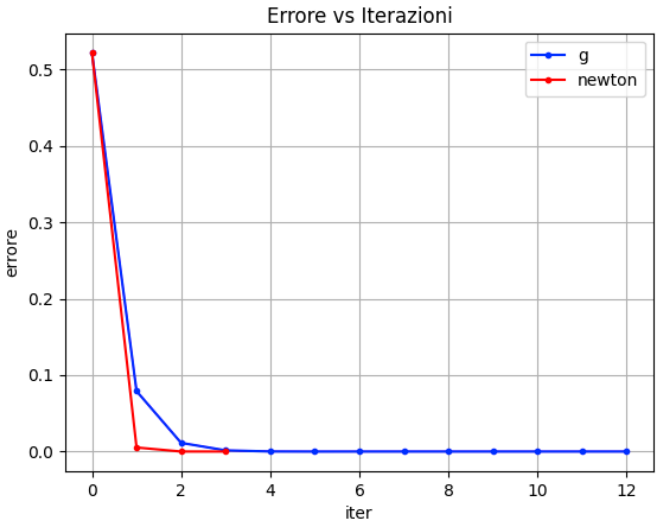




fTrue = -6.822785407889498e-05
Metodo approssimazioni successive g
x = 3.521379706798214
iter_new= 14
Metodo Newton
x = 3.521379706804568
iter_new= 5

Output con x0 = 3:





fTrue = -6.822785407889498e-05
Metodo approssimazioni successive g
x = 3.521379706798214
iter_new= 13
Metodo Newton
x = 3.521379706804568
iter_new= 4