Método de Newton y de la Secante

Mateo Cumbal

2024-11-16

Tabla de Contenidos

1 Ejercicio 1

Sea $f(x) = -x^3 - \cos(x)$ y $p_0 = -1$. Use el método de Newton y de la Secante para encontrar p_2.

```
from scipy.optimize import newton
import numpy as np

def f(x):
    return - x ** 3 - np.cos(x)

def fder(x):
    return -3 * x**2 + np.sin(x)
```

```
p0 = -1
```

```
n1 = newton(f, p0, fprime=fder)
print(f'La raiz aproximada es: {n1} | Método de Newton')
```

```
La raiz aproximada es: -0.8654740331016144 | Método de Newton
```

```
s1 = newton(f, p0)

print(f'La raiz aproximada es: {s1} | Método de la Secante')
```

```
La raiz aproximada es: -0.8654740331016144 | Método de la Secante
```

¿Se podría usar $p_0 = 0$?

```
p0 = 0
```

Primero probemos con el método de Newton:

```
try:
    n2 = newton(f, p0, fprime=fder)
except RuntimeError as e:
    print('ERROR:', e)
```

```
ERROR: Derivative was zero. Failed to converge after 1 iterations, value is 0.0.
```

> El código retorna un error porque la derivada evaluada en el punto inicial p_0 es cero, por lo que el método no puede continuar.
>
> $p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}$
>
> $p_1 = 0 - \frac{f(0)}{-3(0)^2+\sin(0)}$ -> Error: División por cero.

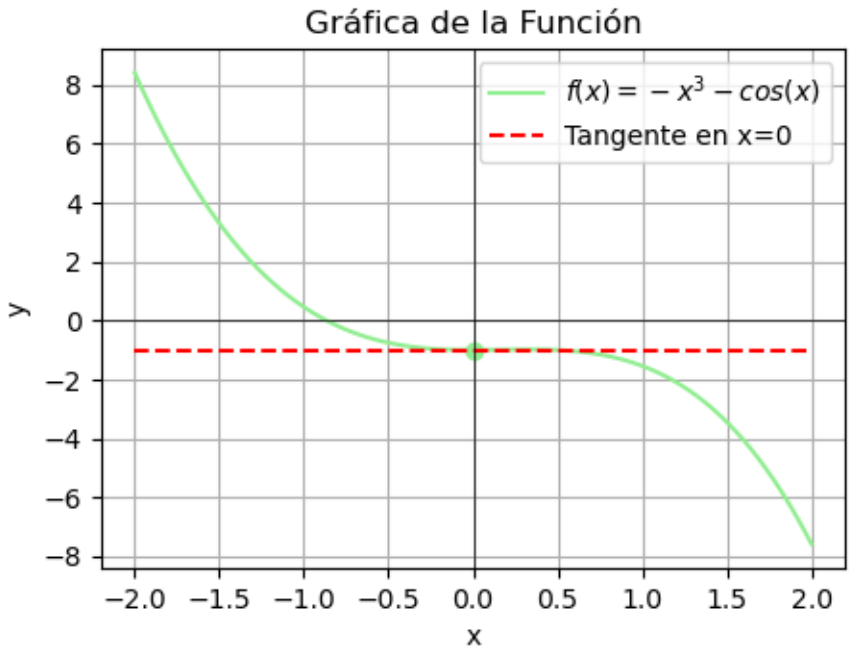
A continuación se puede observar el gráfico de la función y de la recta tangente (derivada) evaluada en ese punto.

```
import matplotlib.pyplot as plt

x = np.linspace(-2, 2, 100)
y = f(x)

f0 = f(0)
fder0 = fder(0)
tangent_line = fder0 * (x - 0) + f0

plt.figure(figsize=(5, 3.5))
plt.plot(x, y, label='$f(x) = -x^3 - cos(x)$', color='lightgreen')
plt.plot(x, tangent_line, label='Tangente en x=0', color='red', linestyle='--')
plt.axhline(0, color='black', lw=0.5)
plt.axvline(0, color='black', lw=0.5)
plt.scatter(0, f(0), color='lightgreen')
plt.title('Gráfica de la Función')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```



Probemos ahora con el método de la secante:

```
s2 = newton(f, p0)

print(f'La raiz aproximada es: {s2} | Método de la Secante')
```

```
La raiz aproximada es: -4.998000183473029e-09 | Método de la Secante
```

El algoritmo retorna un resultado que es incorrecto. Cabe destacar que en este caso, el método calcula el segundo valor inicial, que podría ser un factor para que retorne la respuesta incorrecta.

Probemos esta vez asignando los dos valores iniciales:

```
try:
    s2_1 = newton(f, p0, x1=0.1)
except RuntimeError as e:
    print('ERROR:', e)
```

```
ERROR: Failed to converge after 50 iterations, value is -0.0035371633005660988.
```

En cambio, aquí se produce un error de no convergencia.

```
s2_2 = newton(f, p0, x1 = 2)

print(f'La raiz aproximada es: {s2_2} | Método de la Secante')
```

```
La raiz aproximada es: -0.8654740331017148 | Método de la Secante
```

Para este par de valores iniciales, arroja una respuesta correcta.

2 Ejercicio 2

Encuentre soluciones precisas dentro de 10^{-4} para los siguientes problemas:

Definimos la tolerancia para todos los literales.

```
TOL = 1e-04
```

a. $x^3 - 2x^2 - 5 = 0$**, en el intervalo** $[1, 4]$

```python
def f(x):
    return x ** 3 - 2 * x ** 2 - 5

def fder(x):
    return 3 * x ** 2 - 4 * x

p0 = 2
```

```python
n2a, result = newton(f, p0, fprime=fder, tol=TOL, full_output=True)

print(f'La raiz aproximada es: p{result.iterations} = {n2a:.5f} | Método de Newton')
```

```
La raiz aproximada es: p5 = 2.69065 | Método de Newton
```

b. $x^3 + 3x^2 - 1 = 0$**, en el intervalo** $[-3, -2]$

```python
def f(x):
    return x ** 3 + 3 * x ** 2 - 1

def fder(x):
    return 3 * x ** 2 + 6 * x

p0 = -3
```

```python
n2b, result = newton(f, p0, fprime=fder, tol=TOL, full_output=True)

print(f'La raiz aproximada es: p{result.iterations} = {n2b:.5f} | Método de Newton')
```

```
La raiz aproximada es: p3 = -2.87939 | Método de Newton
```

c. $x - \cos(x) = 0$**, en el intervalo** $[0, \frac{\pi}{2}]$

```python
def f(x):
    return x - np.cos(x)

def fder(x):
    return 1 + np.sin(x)

p0 = 0
```

```
n2c, result = newton(f, p0, fprime=fder, tol=TOL, full_output=True)

print(f'La raiz aproximada es: p{result.iterations} = {n2c:.5f} | Método de Newton')
```

```
La raiz aproximada es: p4 = 0.73909 | Método de Newton
```

d. $x - 0.8 - 0.2\sin(x) = 0$**, en el intervalo** $[0, \frac{\pi}{2}]$

```
def f(x):
    return x - 0.8 - 0.2 * np.sin(x)

def fder(x):
    return 1 - 0.2 * np.cos(x)

p0 = 0
```

```
n2d, result = newton(f, p0, fprime=fder, tol=TOL, full_output=True)

print(f'La raiz aproximada es: p{result.iterations} = {n2d:.5f} | Método de Newton')
```

```
La raiz aproximada es: p4 = 0.96433 | Método de Newton
```

3 Ejercicio 3

Encuentre las soluciones dentro de 10^{-5} **para los siguientes problemas utilizando los métodos de Newton y de la Secante:**

Inicialmente, definimos la toleracia.

```
TOL = 1e-05
```

a. $3x - e^x = 0$ **para** $1 \leq x \leq 2$

```
def f(x):
    return 3 * x - np.exp(x)

def fder(x):
    return 3 - np.exp(x)

p0 = 1.5
```

```
n3a, result = newton(f, p0, fprime=fder, tol=TOL, full_output=True)

print(f'La raiz aproximada es: p{result.iterations} = {n3a:.8f} | Método de Newton')
```

```
La raiz aproximada es: p3 = 1.51213455 | Método de Newton
```

```
p0 = 1
p1 = 2

s3a, result = newton(f, p0, x1=p1, tol=TOL, full_output=True)

print(f'La raiz aproximada es: p{result.iterations} = {s3a:.8f} | Método de la Secante')
```

```
La raiz aproximada es: p9 = 1.51213455 | Método de la Secante
```

b. $2x + 3\cos(x) - e^x = 0$ **para** $1 \leq x \leq 2$

```
def f(x):
    return 2 * x + 3 * np.cos(x) - np.exp(x)

def fder(x):
    return 2 - 3 * np.sin(x) - np.exp(x)

p0 = 1.5
```

```
n3b, result = newton(f, p0, fprime=fder, tol=TOL, full_output=True)

print(f'La raiz aproximada es: p{result.iterations} = {n3b:.8f} | Método de Newton')
```

```
La raiz aproximada es: p4 = 1.23971470 | Método de Newton
```

```
p0 = 1
p1 = 2

s3b, result = newton(f, p0, x1=p1, tol=TOL, full_output=True)

print(f'La raiz aproximada es: p{result.iterations} = {s3b:.8f} | Método de la Secante')
```

```
La raiz aproximada es: p6 = 1.23971470 | Método de la Secante
```

4 Ejercicio 4

El polinomio de cuarto grado

$$f(x) = 230x^4 + 18x^3 + 9x^2 - 221x - 9$$

tiene dos ceros reales, uno en $[-1, 0]$ y el otro en $[0, 1]$. Intente aproximar estos ceros dentro de 10^{-6} con:

```
TOL = 1e-06

def f(x):
    return 230 * x ** 4 + 18 * x ** 3 + 9 * x ** 2 - 221 * x - 9
```

a. El método de la secante (use los extremos como las estimaciones iniciales).

```
p0 = -1
p1 = 0

s4_1 = newton(f, p0, x1=p1, tol=TOL)

print(f'La raiz aproximada es: {s4_1:.8f} | Método de la Secante')
```

```
La raiz aproximada es: -0.04065929 | Método de la Secante
```

```
p0 = 0
p1 = 1

s4_2 = newton(f, p0, x1=p1, tol=TOL)

print(f'La raiz aproximada es: {s4_2:.8f} | Método de la Secante')
```

```
La raiz aproximada es: -0.04065929 | Método de la Secante
```

Debido a la naturaleza del algoritmo, con ambos intevalos específicos se llega a la misma respuesta y no a las distintas.

```
p0 = 0.01
p1 = 1

s4_2_2 = newton(f, p0, x1=p1, tol=TOL)

print(f'La raiz aproximada es: {s4_2_2:.8f} | Método de la Secante')
```

```
La raiz aproximada es: 0.96239842 | Método de la Secante
```

Con una pequeña modificación al intervalo $p_0 = 0.01$, se consigue la otra raíz.

b. El método de Newton (use el punto medio como estimación inicial).

```
def fder(x):
    return 920 * x ** 3 + 54 * x ** 2 + 18 * x - 221
```

```
p0 = -0.5

n4_1 = newton(f, p0, fprime=fder)

print(f'La raiz aproximada es: {n4_1:.8f} | Método de Newton')
```

```
La raiz aproximada es: -0.04065929 | Método de Newton
```

```
p0 = 0.5

n4_2 = newton(f, p0, fprime=fder)

print(f'La raiz aproximada es: {n4_2:.8f} | Método de Newton')
```

```
La raiz aproximada es: -0.04065929 | Método de Newton
```

> Es correcto que hay soluciones diferentes en cada intervalo. Pero, al utilizar los valores medios podemos observar que el método de Newton converge hacia la misma respuesta.
>
> **Esto sucede porque para ambos valores iniciales, se encuentran en una región donde la función es decreciente y continua, sus tangentes apuntan hacia el mismo valor de convergencia en el eje x. Así, independientemente del punto inicial en este intervalo, el método converge a la misma raíz.**

Podemos notar lo antes mencionado en el gráfico:

Pero antes calculamos el mímino relativo

```
min_rel = newton(fder, 1)
print(min_rel)
```

```
0.5925163670059391
```

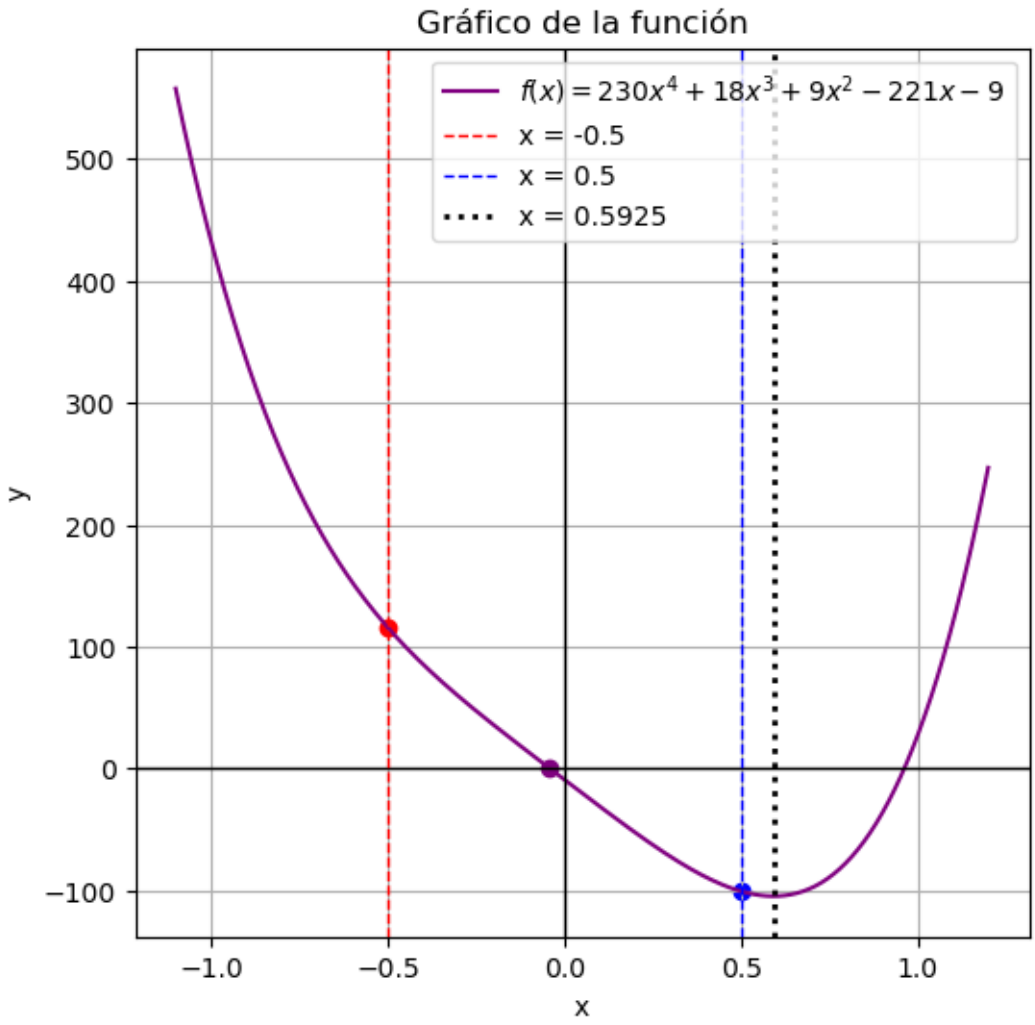
```
x = np.linspace(-1.1, 1.2, 400)
y = f(x)

p0_1 = -0.5
p0_2 = 0.5

plt.figure(figsize=(6, 6))
plt.plot(x, y, label=r"$f(x) = 230x^4 + 18x^3 + 9x^2 - 221x - 9$", color='purple')
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.axvline(p0_1, color='red', linestyle='--', linewidth=1, label="x = -0.5")
plt.axvline(p0_2, color='blue', linestyle='--', linewidth=1, label="x = 0.5")
plt.axvline(min_rel, color='black', linestyle=':', linewidth=2, label="x = 0.5925")

plt.scatter(p0_1, f(p0_1), color='red')
plt.scatter(p0_2, f(p0_2), color='blue')
plt.scatter(n4_1, 0, color='purple')


plt.xlabel("x")
plt.ylabel("y")
plt.title("Gráfico de la función")
plt.legend()
plt.grid(True)
plt.show()
```



5 Ejercicio 5

La función $f(x) = \tan(\pi x) - 6$ tiene un cero en $\frac{1}{\pi}\arctan(6) \approx 0.447431543$. Sea $p_0 = 0$ y $p_1 = 0.48$. Utilice 10 iteraciones en cada uno de los siguientes métodos para aproximar esta raíz. ¿Cuál método es más eficaz y por qué?

```
def f(x):
    return np.tan(np.pi * x) - 6

p0 = 0
p1 = 0.48
```

```
MAXITER = 10
```

- **Método de Bisección**

```
from scipy.optimize import bisect

try:
    b5, result = bisect(f, p0, p1, maxiter=MAXITER, full_output=True)
except RuntimeError as e:
    print('ERROR:', e)
```

```
ERROR: Failed to converge after 10 iterations.
```

Con el método de la bisección converge a las 38 iteraciones.

```
b5_1, result = bisect(f, p0, p1, maxiter=100, full_output=True)

print(f'La raiz aproximada es: p{result.iterations} = {b5_1:.8f} | Método de la Bisección')
```

```
La raiz aproximada es: p38 = 0.44743154 | Método de la Bisección
```

- **Método de Newton**

```
p0 = 0.24

def fder(x):
    return np.pi * (1 / np.cos(np.pi * x))**2

n5, result = newton(f, p0, fprime=fder, maxiter=MAXITER, full_output=True)

print(f'La raiz aproximada es: p{result.iterations - 1} = {n5:.8f} | Método de la Newton')
```

```
La raiz aproximada es: p7 = 5.44743154 | Método de la Newton
```

- **Método de la Secante**

```
p0 = 0
p1 = 0.48
try:
    s5, result = newton(f, p0, x1=p1, maxiter=MAXITER, full_output=True)
except RuntimeError as e:
    print('ERROR:', e)
```

```
ERROR: Failed to converge after 10 iterations, value is -3694.358600967476.
```

Por lo tanto, el método más eficaz es el de **Newton**, debido a las herramientas matemáticas que usa para determinar la aproximación.

6 Ejercicio 6

La función descrita por $f(x) = ln(x^2 + 1) - e^{0.4x}\cos(\pi x)$ tiene un número infinito de ceros.

```python
def f(x):
    return np.log(x ** 2 + 1) - np.exp(0.4  * x) * np.cos(np.pi * x)

def fder(x):
    return ((2 * x) / x ** 2 + 1) - 0.4 * np.exp(0.4 * x) * np.cos(np.pi * x) + np.pi * np.ex

TOL = 1e-06
```

a. Determine, dentro de 10—6, el único cero negativo.

```python
p0 = -0.5

n6a, r = newton(f, p0, fprime=fder, tol=TOL, full_output=True)

print(f'La raiz aproximada es: p{r.iterations - 1} = {n6a:.8f} | Método de la Newton')
```

```
La raiz aproximada es: p14 = -0.43414348 | Método de la Newton
```

b. Determine, dentro de 10—6, los cuatro ceros positivos más pequeños.

```python
ceros = []
p0 = 0.5
for i in range(4):
    ans, r = newton(f, p0, fprime=fder, tol=TOL, full_output=True)
    ceros.append((ans, r.iterations))
    p0 += 1

print('Método de Newton - Raices')
for root, iter in ceros:
    print(f'La raiz aproximada es: p{iter} = {root:.8f}')
```

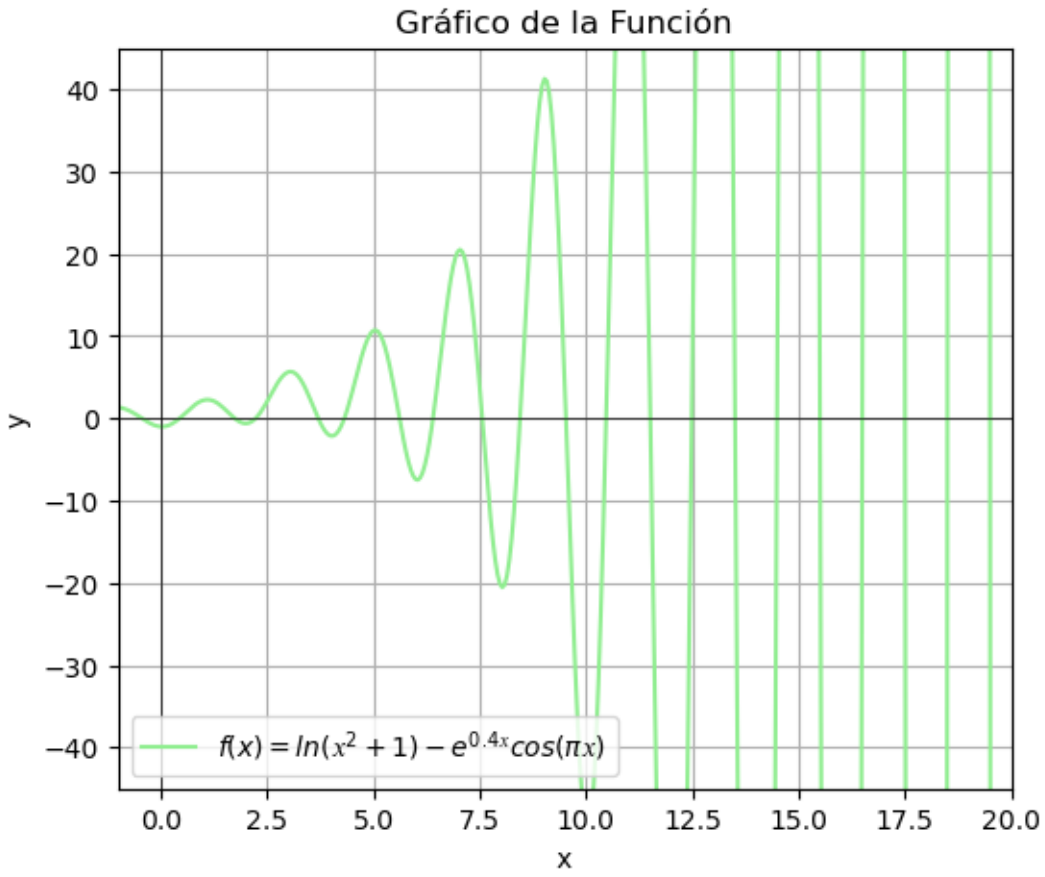
```
Método de Newton - Raices
La raiz aproximada es: p17 = 0.45065733
La raiz aproximada es: p18 = 1.74473790
La raiz aproximada es: p9 = 2.23831990
La raiz aproximada es: p7 = 3.70904123
```

c. Determine una aproximación inicial razonable para encontrar el enésimo cero positivo más pequeño de f.

Primero, dibujamos una gráfica aproximada de f.

```
x = np.linspace(-2, 20, 400)
y = f(x)

plt.figure(figsize=(6, 5))
plt.plot(x, y, label='$f(x) = ln( ^2 + 1) - e^{0.4 } cos(\\pi )$', color='lightgreen')
plt.axhline(0, color='black', lw=0.5)
plt.axvline(0, color='black', lw=0.5)
plt.xlim(-1,20)
plt.ylim(-45, 45)
plt.title('Gráfico de la Función')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```



Con $aprox = 0.5$ se logran obtener todas las raices que aparecen hasta un número razonable de x, tal que $x = 30$

```
roots = []
for n in range(1, 31):
    p0 = n - 0.5
    ans = newton(f, p0, fprime=fder, tol=TOL)
    roots.append((n, ans))

for n, root in roots:
    print(f'Raiz aproximada: p = {root:.8f} con n = {n}')
```

```
Raiz aproximada: p = 0.45065733 con n = 1
Raiz aproximada: p = 1.74473790 con n = 2
Raiz aproximada: p = 2.23831990 con n = 3
Raiz aproximada: p = 3.70904123 con n = 4
Raiz aproximada: p = 4.32264902 con n = 5
```

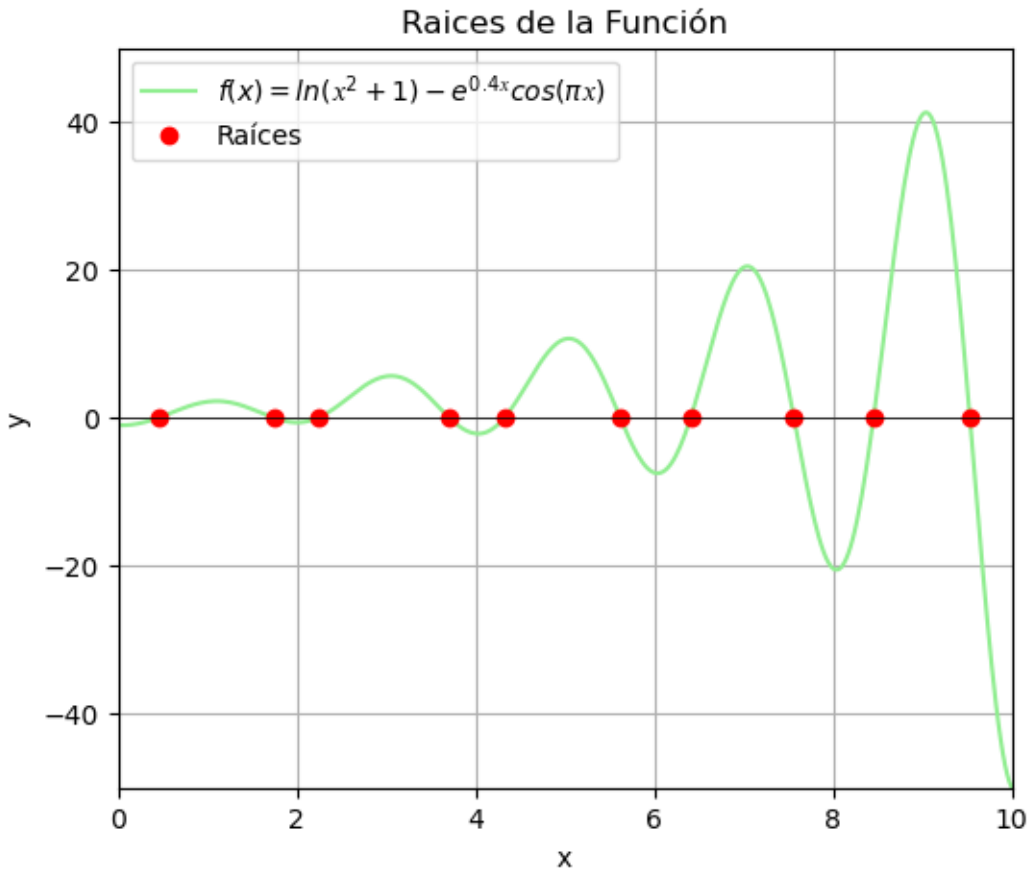
```
Raiz aproximada: p = 5.61993535 con n = 6
Raiz aproximada: p = 6.40693362 con n = 7
Raiz aproximada: p = 7.56321052 con n = 8
Raiz aproximada: p = 8.45348098 con n = 9
Raiz aproximada: p = 9.53183353 con n = 10
Raiz aproximada: p = 10.47730589 con n = 11
Raiz aproximada: p = 11.51557072 con n = 12
Raiz aproximada: p = 12.48910641 con n = 13
Raiz aproximada: p = 13.50747141 con n = 14
Raiz aproximada: p = 14.49483105 con n = 15
Raiz aproximada: p = 15.50353913 con n = 16
Raiz aproximada: p = 16.49756834 con n = 17
Raiz aproximada: p = 17.50166147 con n = 18
Raiz aproximada: p = 18.49886353 con n = 19
Raiz aproximada: p = 19.50077493 con n = 20
Raiz aproximada: p = 20.49947156 con n = 21
Raiz aproximada: p = 21.50035967 con n = 22
Raiz aproximada: p = 22.49975529 con n = 23
Raiz aproximada: p = 23.50016630 con n = 24
Raiz aproximada: p = 24.49988705 con n = 25
Raiz aproximada: p = 25.50007665 con n = 26
Raiz aproximada: p = 26.49994801 con n = 27
Raiz aproximada: p = 27.50003525 con n = 28
Raiz aproximada: p = 28.49997612 con n = 29
Raiz aproximada: p = 29.50001617 con n = 30
```

Como muesta, graficamos para las primeras 20 raices a partir de $x = 0$.

```
x = np.linspace(0, 10, 400)
y = f(x)
raices = [root for _, root in roots]

plt.figure(figsize=(6, 5))
plt.plot(x, y, label='$f(x) = ln( ^2 + 1) - e^{0.4 } cos(\\pi )$', color='lightgreen')
plt.plot(raices, [0] * len(raices), 'o', color='red', label='Raíces')
plt.axhline(0, color='black', lw=0.5)
plt.axvline(0, color='black', lw=0.5)
plt.title('Raices de la Función')
plt.xlabel('x')
plt.ylabel('y')
plt.xlim(0,10)
plt.ylim(-50, 50)
```

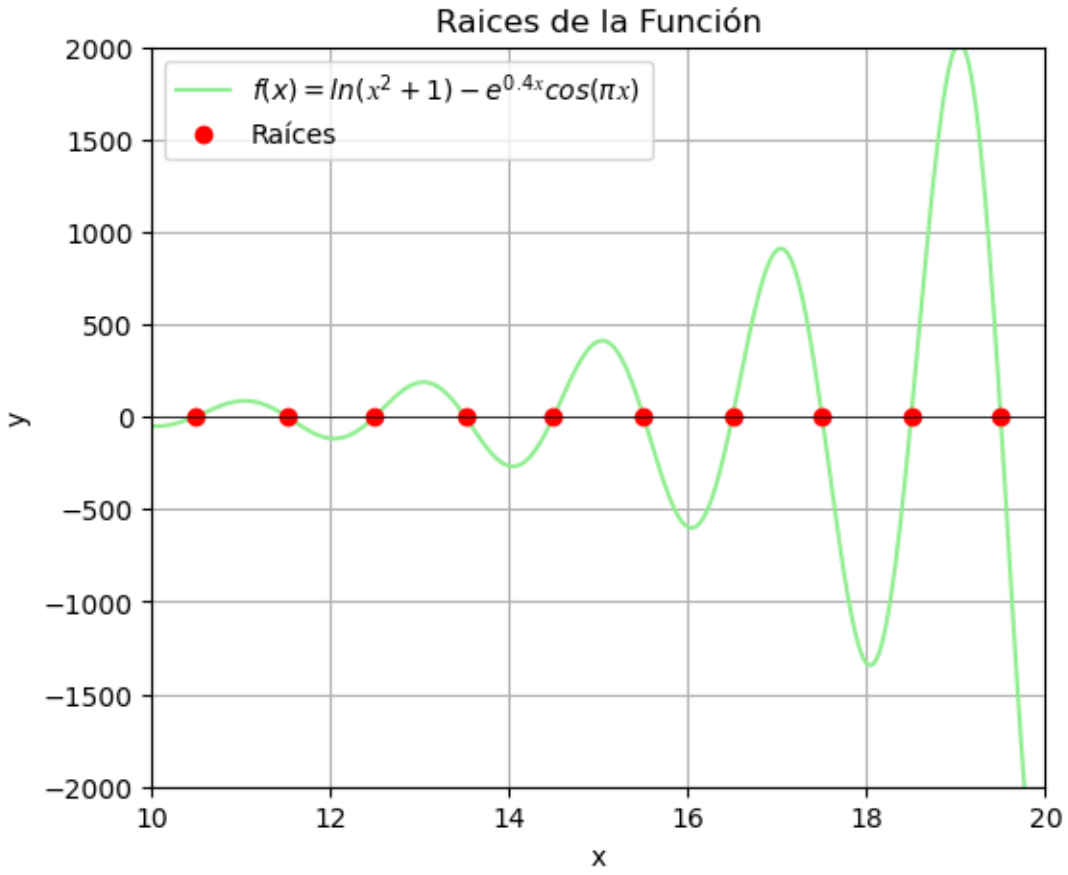
```
plt.legend()
plt.grid()
plt.show()
```




```
x = np.linspace(10, 20, 400)
y = f(x)
raices = [root for _, root in roots]

plt.figure(figsize=(6, 5))
plt.plot(x, y, label='$f(x) = ln( ^2 + 1) - e^{0.4 } cos(\\pi )$', color='lightgreen')
plt.plot(raices, [0] * len(raices), 'o', color='red', label='Raíces')
plt.axhline(0, color='black', lw=0.5)
plt.axvline(0, color='black', lw=0.5)
plt.title('Raices de la Función')
plt.xlabel('x')
plt.ylabel('y')
```

```
plt.xlim(10,20)
plt.ylim(-2000, 2000)
plt.legend()
plt.grid()
plt.show()
```



d. Use la parte c) para determinar, dentro de 10−6, el vigesimoquinto cero positivo más pequeño de .

```
root25 = roots[24]
print(f'Raiz aproximada: p = {root25[1]:.8f} con n = {root25[0]}')
```

```
Raiz aproximada: p = 24.49988705 con n = 25
```

7 Ejercicio 7

La función $f(x) = x^{1/3}$ tiene raíz en $x = 0$. Usando el punto de inicio de $x = 1$ y $p_0 = 5$, $p_1 = 0.5$ para el método de secante, compare los resultados de los métodos de la secante y de Newton.

- **Método de Newton**

```
def f(x):
    return np.cbrt(x)

def fder(x):
    return (1 / 3) * (x ** (-2 / 3)) if x != 0 else float('inf')
```

Usando el método de scipy, nos retorna un error, debido a que el algoritmo no llegó a una respuesta y, de hecho, el valor final es NaN.

```
p0 = 1
try:
    n7 = newton(f, p0, fprime=fder)
except RuntimeError as r:
    print('ERROR:', r)
```

```
ERROR: Failed to converge after 50 iterations, value is nan.
```

Veámoslo con más detalle:

```
def f(x: float, points: list[float] = [], show: bool = False):
    y =  np.cbrt(x)
    points.append((x, y))
    if show:
        print(f"f({x}) = {y}")
    return y


def df(x, *args):
    return (1 / 3) * (x ** (-2 / 3)) if x != 0 else float('inf')

x0 = 1
points = []
try:
```

```
    x_r = newton(f, x0, fprime=df, args=(points, True), maxiter=5)
except RuntimeError as e:
    print('ERROR:', e)
```

```
f(1.0) = 1.0
f(-2.0) = -1.259921049894873
f(nan) = nan
f(nan) = nan
f(nan) = nan
ERROR: Failed to converge after 5 iterations, value is nan.
```

Sesde la tercera iteración, el valor que se obtiene como x_n es un NaN. Según ChatGPT:

> El problema del NaN ocurre en este caso porque la función y/o su derivada genera valores indefinidos o no representables en ciertas condiciones.
>
> Para $x < 0$, el cálculo de $x^{1/3}$ utilizando x^{frac} (potencias fraccionarias) con ciertas implementaciones puede generar NaN debido a cómo se maneja la raíz cúbica de números negativos en floating-point.
>
> Por ejemplo, en Python, escribir directamente x ** (1/3) puede dar un error o un NaN para x<0. Esto ocurre porque, por defecto, las operaciones de potencia no consideran correctamente el manejo de números negativos con exponentes fraccionarios

Pero aún sin eso, de todas maneras el algoritmo no llegaría a una respuesta. Para este caso, el método diverge, y cada vez se aleja más de la respuesta. Se puede observar en las siguentes imagen:

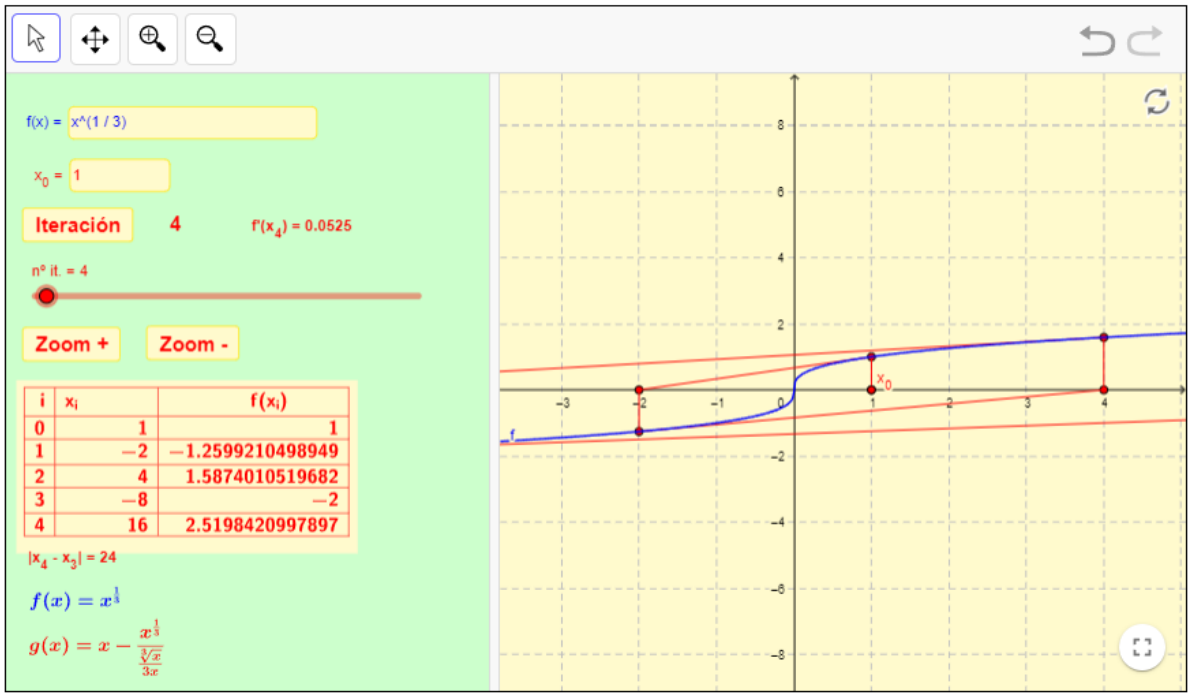
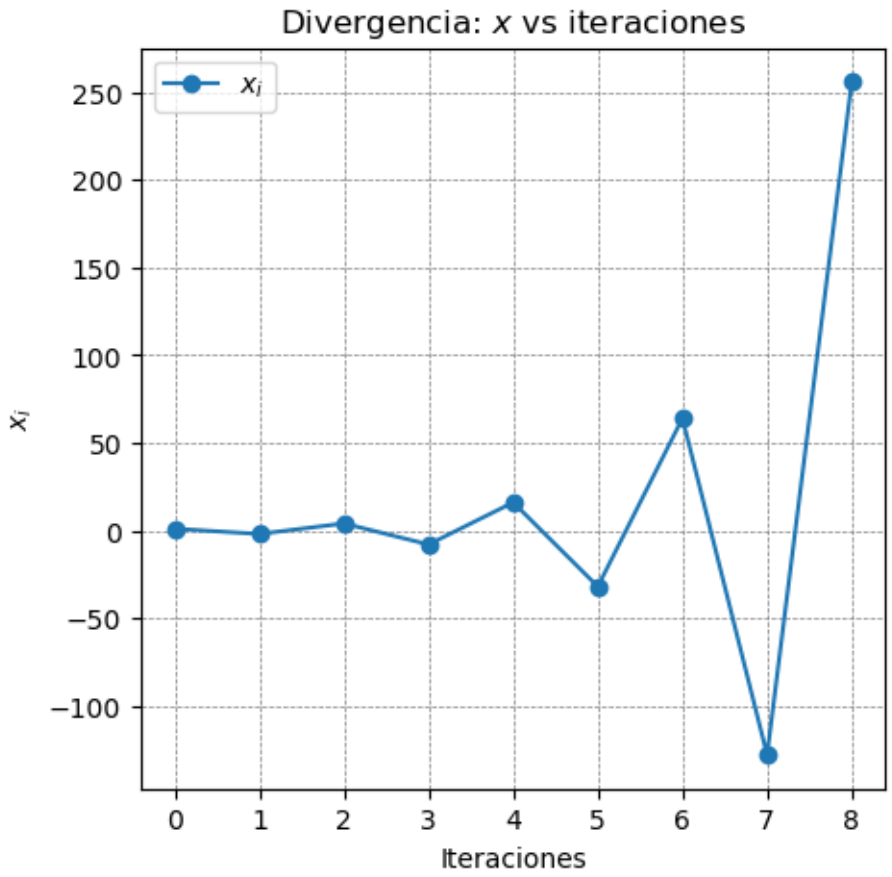


Figura 1: Ejercicio 7: Método de Newton - Divergencia

```
possible = [2 ** x for x in range(9)]

for i in range(9):
    if i%2!=0:
        possible[i] *= -1

# Plot the x points
plt.figure(figsize=(5, 5))
plt.plot(possible, "o-", label="$x_i$")
plt.grid(color="gray", linestyle="--", linewidth=0.5)
plt.title("Divergencia: $x$ vs iteraciones")
plt.xlabel("Iteraciones")
plt.ylabel("$x_i$")
plt.legend()
plt.show()
```



- **Método de la Secante**

No encuentra la respuesta dentro de las iteraciones especificadas:

```
p0 = 5
p1 = 0.5

try:
    s7 = newton(f, p0, x1=p1)
except RuntimeError as e:
    print('ERROR:', e)
```

```
ERROR: Failed to converge after 50 iterations, value is 0.15125956067017968.
```