Serie de Taylor y Polinomios de Lagrange

Anthony Contreras

Tabla de Contenidos

Conjunto de Ejercicios **1**
Ejercicio 1 1
Ejercicio 2 3
Ejercicio 3 6
Ejericio 4 8
Ejercicio 5 9
Ejercicio 6 11
Ejercicio 7 13

Conjunto de Ejercicios

Link Repositorio Github: Método de Newton, Secante y Bisección

Ejercicio 1

Sea f(x) $= -\ {}^3 - \cos$ y $= -1$. Use el **método de Newton** y de la **Secante** para encontrar . ¿Se podría usar $= 0$?

```
import numpy as np

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

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

def metodo_newton(f,f_prime, p0, tol=1e-10,max_iter=100):
    for i in range(max_iter):
        f_p0 = f(p0)
        f_prime_p0 = f_prime(p0)
        if abs(f_prime_p0) < tol:
            print('La derivada es muy cercana a cero, el método no converge.')
            return None
        p1 = p0 - f_p0 / f_prime_p0
        print(f'Iteración {i+1}: p{i} = {p0}, p{i+1} = {p1}')
        if abs(p1 - p0) < tol:  # Converge si la diferencia es menor a la tolerancia
            return p1
        p0 = p1
    print('No se alcanzó la tolerancia en el número máximo de iteraciones.')
    return p0

def metodo_secante(f, p0, p1, tol=1e-10, max_iter=100):
    for i in range(max_iter):
        f_p0 = f(p0)
        f_p1 = f(p1)
        if abs(f_p1 - f_p0) < tol:  # Evitar división por cero
            print('La diferencia entre f(p0) y f(p1) es muy pequeña, el método no converge.'
            return None
        p2 = p1 - f_p1 * (p1 - p0) / (f_p1 - f_p0)
        print(f'Iteración {i+1}: p{i} = {p0}, p{i+1} = {p1}, p{i+2} = {p2}')
        if abs(p2 - p1) < tol:  # Converge si la diferencia es menor a la tolerancia
            return p2
        p0, p1 = p1, p2
    print('No se alcanzó la tolerancia en el número máximo de iteraciones.')
    return p1

# Parámetros iniciales
p0_newton = -1
p0_secant = -1
p1_secant = -0.8804  # A partir de Newton

# Resolución con el método de Newton
print('\nMétodo de Newton:')
p2_newton = metodo_newton(f, f_prime, p0_newton)

# Resolución con el método de la Secante
print('\nMétodo de la Secante:')
p2_secant = metodo_secante(f, p0_secant, p1_secant)
```

```python
# Resultados finales
print('\nResultado final:')
print(f'Con el método de Newton: p2   {p2_newton}')
print(f'Con el método de la Secante: p2   {p2_secant}')
```

```
Método de Newton:
Iteración 1: p0 = -1, p1 = -0.880332899571582
Iteración 2: p1 = -0.880332899571582, p2 = -0.8656841631760818
Iteración 3: p2 = -0.8656841631760818, p3 = -0.865474075952977
Iteración 4: p3 = -0.865474075952977, p4 = -0.8654740331016162
Iteración 5: p4 = -0.8654740331016162, p5 = -0.8654740331016144

Método de la Secante:
Iteración 1: p0 = -1, p1 = -0.8804, p2 = -0.8672429493882272
Iteración 2: p1 = -0.8804, p2 = -0.8672429493882272, p3 = -0.8654993790397474
Iteración 3: p2 = -0.8672429493882272, p3 = -0.8654993790397474, p4 = -0.865474076572966
Iteración 4: p3 = -0.8654993790397474, p4 = -0.865474076572966, p5 = -0.865474033102684
Iteración 5: p4 = -0.865474076572966, p5 = -0.865474033102684, p6 = -0.8654740331016144

Resultado final:
Con el método de Newton: p2   -0.8654740331016144
Con el método de la Secante: p2   -0.8654740331016144
```

Ejercicio 2

Encuentre soluciones precisas dentro de 10—4 para los siguientes problemas. a. 3 − 2 2 − 5 = 0, [1,4]

b. 3 + 3 2 − 1 = 0, [−3, −2]

c. − cos = 0, [0, 2/]

d. − 0.8 − 0.2 sen = 0, [0, 2]

```python
intervalos = {
    "a": (1, 4),
    "b": (-3, -2),
    "c": (0, np.pi / 2),
    "d": (0, np.pi / 2),
}
```

```python
def metodo_biseccion(f,a,b, tol =1e-4, max_iter = 100):
    if f(a) * f(b) > 0:
        print('NO tiene al menos una raiz')
    print(f'Resolviendo el intervalo [{a},{b}] con tolerancia {tol}')
    iteracion = 0
    while (b - a) > tol and iteracion < max_iter:
        p = (a+b)/2 # El primer paso es calcular el punto medio del intervalo
        f_p = f(p)
        print(f'Iteracion {iteracion +1}: p = {p}, p{iteracion +1} = {f_p}')
        if abs(f_p) < tol or (b-a)/2 < tol: # Verificamos la tolerancia
            return p
        # Actualizamos los intervalos
        if f(a) * f_p < 0:
            b = p
        else:
            a = p
        iteracion += 1
    if iteracion == 100:
        print('No se alcanzo la tolerancia en el numero maximo de iteraciones.')


def f_a(x):  # a. x^3 - 2x^2 - 5
    return x**3 - 2*x**2 - 5

def f_b(x):  # b. x^3 + 3x^2 - 1
    return x**3 + 3*x**2 - 1

def f_c(x):  # c. x - cos(x)
    return x - np.cos(x)

def f_d(x):  # d. x - 0.8 - 0.2*sin(x)
    return x - 0.8 - 0.2 * np.sin(x)

# Resolución de las funciones usando el metodo de Bisección
print("Soluciones del ejercicio 2:\n")

# Función a
sol_a = metodo_biseccion(f_a, *intervalos["a"])
print(f"\nSolución para (a): x   {sol_a}")

# Función b
sol_b = metodo_biseccion(f_b, *intervalos["b"])
```

```
print(f"\nSolución para (b): x   {sol_b}")

# Función c
sol_c = metodo_biseccion(f_c, *intervalos["c"])
print(f"\nSolución para (c): x   {sol_c}")

# Función d
sol_d = metodo_biseccion(f_d, *intervalos["d"])
print(f"\nSolución para (d): x   {sol_d}")
```

```
Soluciones del ejercicio 2:

Resolviendo el intervalo [1,4] con tolerancia 0.0001
Iteracion 1: p = 2.5, p1 = -1.875
Iteracion 2: p = 3.25, p2 = 8.203125
Iteracion 3: p = 2.875, p3 = 2.232421875
Iteracion 4: p = 2.6875, p4 = -0.034423828125
Iteracion 5: p = 2.78125, p5 = 1.043243408203125
Iteracion 6: p = 2.734375, p6 = 0.4907798767089844
Iteracion 7: p = 2.7109375, p7 = 0.2248091697692871
Iteracion 8: p = 2.69921875, p8 = 0.09435528516769409
Iteracion 9: p = 2.693359375, p9 = 0.02975698560476303
Iteracion 10: p = 2.6904296875, p10 = -0.0023855315521359444
Iteracion 11: p = 2.69189453125, p11 = 0.01367269002366811
Iteracion 12: p = 2.691162109375, p12 = 0.005640321163809858
Iteracion 13: p = 2.6907958984375, p13 = 0.0016265804351860425
Iteracion 14: p = 2.69061279296875, p14 = -0.0003796791327204187
Iteracion 15: p = 2.690704345703125, p15 = 0.0006233997553692916

Solución para (a): x   2.690704345703125
Resolviendo el intervalo [-3,-2] con tolerancia 0.0001
Iteracion 1: p = -2.5, p1 = 2.125
Iteracion 2: p = -2.75, p2 = 0.890625
Iteracion 3: p = -2.875, p3 = 0.033203125
Iteracion 4: p = -2.9375, p4 = -0.460693359375
Iteracion 5: p = -2.90625, p5 = -0.208160400390625
Iteracion 6: p = -2.890625, p6 = -0.08609390258789062
Iteracion 7: p = -2.8828125, p7 = -0.026100635528564453
Iteracion 8: p = -2.87890625, p8 = 0.003637254238128662
Iteracion 9: p = -2.880859375, p9 = -0.011210165917873383
Iteracion 10: p = -2.8798828125, p10 = -0.003781077452003956
Iteracion 11: p = -2.87939453125, p11 = -7.056735921651125e-05
```

```
Solución para (b): x   -2.87939453125
Resolviendo el intervalo [0,1.5707963267948966] con tolerancia 0.0001
Iteracion 1: p = 0.7853981633974483, p1 = 0.0782913822109007
Iteracion 2: p = 0.39269908169872414, p2 = -0.5311804508125626
Iteracion 3: p = 0.5890486225480862, p3 = -0.24242098975445903
Iteracion 4: p = 0.6872233929727672, p4 = -0.08578706038996975
Iteracion 5: p = 0.7363107781851077, p5 = -0.004640347169851511
Iteracion 6: p = 0.760854470791278, p6 = 0.03660738783981099
Iteracion 7: p = 0.7485826244881928, p7 = 0.015928352815779867
Iteracion 8: p = 0.7424467013366502, p8 = 0.005630132459280346
Iteracion 9: p = 0.739378739760879, p9 = 0.0004914153002637534
Iteracion 10: p = 0.7378447589729933, p10 = -0.0020753364865229162
Iteracion 11: p = 0.7386117493669362, p11 = -0.0007921780792695676
Iteracion 12: p = 0.7389952445639076, p12 = -0.0001504357420498703
Iteracion 13: p = 0.7391869921623933, p13 = 0.00017047619334453756
Iteracion 14: p = 0.7390911183631504, p14 = 1.0016828909886755e-05

Solución para (c): x   0.7390911183631504
Resolviendo el intervalo [0,1.5707963267948966] con tolerancia 0.0001
Iteracion 1: p = 0.7853981633974483, p1 = -0.1560231928398613
Iteracion 2: p = 1.1780972450961724, p2 = 0.193321338593915
Iteracion 3: p = 0.9817477042468103, p3 = 0.015453781786301246
Iteracion 4: p = 0.8835729338221293, p4 = -0.07102915685041811
Iteracion 5: p = 0.9326603190344698, p5 = -0.02798118726165924
Iteracion 6: p = 0.9572040116406401, p6 = -0.006312950989676741
Iteracion 7: p = 0.9694758579437253, p7 = 0.004557997386720136
Iteracion 8: p = 0.9633399347921827, p8 = -0.000880568206038268
Iteracion 9: p = 0.966407896367954, p9 = 0.0018379400927886758
Iteracion 10: p = 0.9648739155800683, p10 = 0.00047849252421994226
Iteracion 11: p = 0.9641069251861255, p11 = -0.0002010861699940636
Iteracion 12: p = 0.9644904203830968, p12 = 0.00013869109162645277
Iteracion 13: p = 0.9642986727846112, p13 = -3.120056015359918e-05

Solución para (d): x   0.9642986727846112
```

Ejercicio 3

Use los 2 métodos en esta sección para encontrar las soluciones dentro de 10−5 para los siguientes problemas. a. 3 − = 0 para 1 2 b. 2 + 3 cos − = 0 para 1 2

```
# Vamos a utilizar las funciones que hemos utilizado anteriormente para resolver los ejercic
# Metodo de Newton
def f_a(x):
    return 3 * x - np.exp(x)

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

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

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

newton_a = metodo_newton(f_a,f_prime_a,p0=1.5,tol=1e-5)
print('\n')
newton_b = metodo_newton(f_b,f_prime_b,p0=1.5,tol=1e-5)
```

```
Iteración 1: p0 = 1.5, p1 = 1.5123581458677815
Iteración 2: p1 = 1.5123581458677815, p2 = 1.512134625427124
Iteración 3: p2 = 1.512134625427124, p3 = 1.5121345516578504


Iteración 1: p0 = 1.5, p1 = 1.268096984432167
Iteración 2: p1 = 1.268096984432167, p2 = 1.240119693460797
Iteración 3: p2 = 1.240119693460797, p3 = 1.2397147825931407
Iteración 4: p3 = 1.2397147825931407, p4 = 1.2397146979752212
```

```
# Metodo de la Secante

secante_a = metodo_secante(f_a,1,2,tol=1e-5)
print('\n')
secante_b = metodo_secante(f_b,1,2,tol=1e-5)
```

```
Iteración 1: p0 = 1, p1 = 2, p2 = 1.1686153399174835
Iteración 2: p1 = 2, p2 = 1.1686153399174835, p3 = 1.3115165547175733
Iteración 3: p2 = 1.1686153399174835, p3 = 1.3115165547175733, p4 = 1.7970430096312444
Iteración 4: p3 = 1.3115165547175733, p4 = 1.7970430096312444, p5 = 1.4367778925334904
Iteración 5: p4 = 1.7970430096312444, p5 = 1.4367778925334904, p6 = 1.4867662868726117
Iteración 6: p5 = 1.4367778925334904, p6 = 1.4867662868726117, p7 = 1.5153257605230879
Iteración 7: p6 = 1.4867662868726117, p7 = 1.5153257605230879, p8 = 1.512011934333299
```

```
Iteración 8: p7 = 1.5153257605230879, p8 = 1.512011934333299, p9 = 1.5121339760022816
Iteración 9: p8 = 1.512011934333299, p9 = 1.5121339760022816, p10 = 1.5121345517620621
```

```
Iteración 1: p0 = 1, p1 = 2, p2 = 1.1629251374599332
Iteración 2: p1 = 2, p2 = 1.1629251374599332, p3 = 1.216410423288127
Iteración 3: p2 = 1.1629251374599332, p3 = 1.216410423288127, p4 = 1.2406842080722846
Iteración 4: p3 = 1.216410423288127, p4 = 1.2406842080722846, p5 = 1.2397029136618418
Iteración 5: p4 = 1.2406842080722846, p5 = 1.2397029136618418, p6 = 1.239714692081511
Iteración 6: p5 = 1.2397029136618418, p6 = 1.239714692081511, p7 = 1.2397146979752531
```

Ejericio 4

El polinomio de cuarto grado

\[( ) = 230 + 18 ^{3} + 9 ^{2} - 221 - 9\]

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

a. El **método de la secante** (use los extremos como las estimaciones iniciales).
b. El **método de Newton** (use el punto medio como estimación inicial).

```python
# Definimos la funcion y su derivada para el metodo de newton

# Función f(x) y su derivada f'(x)
def f_4(x):
    return 230 * x**4 + 18 * x**3 + 9 * x**2 - 221 * x - 9

def f_prime_4(x):
    return 920 * x**3 + 54 * x**2 + 18 * x - 221


newton_4_a = metodo_newton(f_4,f_prime_4,-0.5)
print('\n')
newton_4_b = metodo_newton(f_4,f_prime_4,0.5)
```

```
Iteración 1: p0 = -0.5, p1 = -0.15045248868778283
Iteración 2: p1 = -0.15045248868778283, p2 = -0.04181681394887035
Iteración 3: p2 = -0.04181681394887035, p3 = -0.04065934349732934
```

```
Iteración 4: p3 = -0.04065934349732934, p4 = -0.04065928831575899
Iteración 5: p4 = -0.04065928831575899, p5 = -0.040659288315758865


Iteración 1: p0 = 0.5, p1 = -0.7050898203592815
Iteración 2: p1 = -0.7050898203592815, p2 = -0.3237911142304748
Iteración 3: p2 = -0.3237911142304748, p3 = -0.06460313103057486
Iteración 4: p3 = -0.06460313103057486, p4 = -0.04068615115195558
Iteración 5: p4 = -0.04068615115195558, p5 = -0.04065928834533494
Iteración 6: p5 = -0.04065928834533494, p6 = -0.040659288315758865
```

```
# Ahora con el metodo de la secante

secate_4_a = metodo_secante(f_4,-1,0)
print('\n')
secate_4_b = metodo_secante(f_4,0,1)
```

```
Iteración 1: p0 = -1, p1 = 0, p2 = -0.020361990950226245
Iteración 2: p1 = 0, p2 = -0.020361990950226245, p3 = -0.04069125643524189
Iteración 3: p2 = -0.020361990950226245, p3 = -0.04069125643524189, p4 = -0.0406592625776910
Iteración 4: p3 = -0.04069125643524189, p4 = -0.04065926257769109, p5 = -0.04065928831572513
Iteración 5: p4 = -0.04065926257769109, p5 = -0.040659288315725135, p6 = -0.04065928831575886


Iteración 1: p0 = 0, p1 = 1, p2 = 0.25
Iteración 2: p1 = 1, p2 = 0.25, p3 = 0.7737627651217597
Iteración 3: p2 = 0.25, p3 = 0.7737627651217597, p4 = -1.2854177835209222
Iteración 4: p3 = 0.7737627651217597, p4 = -1.2854177835209222, p5 = 0.5945955204028852
Iteración 5: p4 = -1.2854177835209222, p5 = 0.5945955204028852, p6 = 0.3946411046833955
Iteración 6: p5 = 0.5945955204028852, p6 = 0.3946411046833955, p7 = -0.6693181355515856
Iteración 7: p6 = 0.3946411046833955, p7 = -0.6693181355515856, p8 = 0.04971439761607255
Iteración 8: p7 = -0.6693181355515856, p8 = 0.04971439761607255, p9 = -0.020754150823816236
Iteración 9: p8 = 0.04971439761607255, p9 = -0.020754150823816236, p10 = -0.0407353328963781
Iteración 10: p9 = -0.020754150823816236, p10 = -0.04073533289637811, p11 = -0.0406592282432
Iteración 11: p10 = -0.04073533289637811, p11 = -0.04065922824320605, p12 = -0.0406592883155
Iteración 12: p11 = -0.04065922824320605, p12 = -0.04065928831557162, p13 = -0.0406592883157
```

Ejercicio 5

La función ( ) = tan − 6 tiene cero en (1 /) arcotangente 6 0.447431543. Sea 0 = 0 y 1 = 0.48 y use 10 iteraciones en cada uno de los siguientes métodos para aproximar esta raíz.

¿Cuál método es más eficaz y por qué? a. método de bisección b. método de Newton c. método de la secante

```python
# Definimos sus funciones

# Función f(x) y su derivada f'(x)
def f_5(x):
    return np.tan(np.pi * x) - 6

def f_prime_5(x):
    return np.pi * (1 / np.cos(np.pi * x))**2


newton_5 = metodo_newton(f_5,f_prime_5,p0=0.48,max_iter=10)
```

```
Iteración 1: p0 = 0.48, p1 = 0.4675825019258912
Iteración 2: p1 = 0.4675825019258912, p2 = 0.4551291915177739
Iteración 3: p2 = 0.4551291915177739, p3 = 0.4485512339384831
Iteración 4: p3 = 0.4485512339384831, p4 = 0.4474551842507058
Iteración 5: p4 = 0.4474551842507058, p5 = 0.4474315538237576
Iteración 6: p5 = 0.4474315538237576, p6 = 0.4474315432887487
Iteración 7: p6 = 0.4474315432887487, p7 = 0.4474315432887466
```

```python
secante_5 = metodo_secante(f_5,0,0.48,max_iter=10)
```

```
Iteración 1: p0 = 0, p1 = 0.48, p2 = 0.18119424169051174
Iteración 2: p1 = 0.48, p2 = 0.18119424169051174, p3 = 0.2861871658222898
Iteración 3: p2 = 0.18119424169051174, p3 = 0.2861871658222898, p4 = 1.0919861065027499
Iteración 4: p3 = 0.2861871658222898, p4 = 1.0919861065027499, p5 = -3.6922966654011073
Iteración 5: p4 = 1.0919861065027499, p5 = -3.6922966654011073, p6 = -22.60064985474053
Iteración 6: p5 = -3.6922966654011073, p6 = -22.60064985474053, p7 = -57.22283247260205
Iteración 7: p6 = -22.60064985474053, p7 = -57.22283247260205, p8 = 3.5387581457345476
Iteración 8: p7 = -57.22283247260205, p8 = 3.5387581457345476, p9 = -113.94440504807905
Iteración 9: p8 = 3.5387581457345476, p9 = -113.94440504807905, p10 = -195.89499482451663
Iteración 10: p9 = -113.94440504807905, p10 = -195.89499482451663, p11 = -2989.9400375314453
No se alcanzó la tolerancia en el número máximo de iteraciones.
```

```python
biseccion_5 = metodo_biseccion(f_5,0,0.48,max_iter=10)
```

```
Resolviendo el intervalo [0,0.48] con tolerancia 0.0001
```

```
Iteracion 1: p = 0.24, p1 = -5.060937494182507
Iteracion 2: p = 0.36, p2 = -3.874891826842797
Iteracion 3: p = 0.42, p3 = -2.105257145070144
Iteracion 4: p = 0.44999999999999996, p4 = 0.3137515146750314
Iteracion 5: p = 0.43499999999999994, p5 = -1.171182647807246
Iteracion 6: p = 0.44249999999999995, p6 = -0.5245211508218706
Iteracion 7: p = 0.4462499999999999, p7 = -0.13434976287736955
Iteracion 8: p = 0.44812499999999994, p8 = 0.08167438867622234
Iteracion 9: p = 0.44718749999999996, p9 = -0.028237440581283302
Iteracion 10: p = 0.44765625, p10 = 0.0262307752703812
```

El método de Newton es generalmente considerado el más eficiente debido a su rapidez y eficiencia en la reducción del error. Este método converge rápidamente hacia la solución, lo que lo convierte en una opción ideal cuando se dispone de un buen valor inicial. Sin embargo, si no contamos con un valor inicial adecuado, el método de la secante puede ser más apropiado. Aunque este método puede requerir más iteraciones, tiene la ventaja de no necesitar el cálculo de la primera derivada de la función. Finalmente, el método de la bisección, aunque fiable, es comparativamente más lento que los otros dos métodos.

Ejercicio 6

La función descrita por $( ) = \ln(2 + 1) − 0.4 \cos$ tiene un número infinito de ceros.

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

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

c. Determine una aproximación inicial razonable para encontrar el enésimo cero positivo más pequeño de . [Sugerencia: Dibuje una gráfica aproximada de .]

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

```python
# Literal A

def f_6(x):
    return np.log(x**2 + 1) - np.exp(0.4 * x) * np.cos(np.pi * x)

def f_prime_6(x):
    return (2 * x / (x**2 + 1)) - 0.4 * np.exp(0.4 * x) * np.cos(np.pi * x) + np.pi * np.exp

unico_cero = metodo_newton(f_6,f_prime_6,p0=-0.5)
```

```
Iteración 1: p0 = -0.5, p1 = -0.43382689545228015
Iteración 2: p1 = -0.43382689545228015, p2 = -0.4341430585731075
Iteración 3: p2 = -0.4341430585731075, p3 = -0.43414304728572883
Iteración 4: p3 = -0.43414304728572883, p4 = -0.4341430472857288
```

```python
# Literal B
# Lista para almacenar los ceros
ceros_positivos = []

# Aproximaciones iniciales para los primeros cuatro ceros positivos
p_iniciales = [0.5, 1.5, 2.5, 3.5]

for p in p_iniciales:
    cero = metodo_newton(f_6, f_prime_6, p0=p)
    print('\n')
    ceros_positivos.append(cero)

print(f'Cuatro ceros positivos más pequeños: {ceros_positivos}')
```

```
Iteración 1: p0 = 0.5, p1 = 0.451879159703456
Iteración 2: p1 = 0.451879159703456, p2 = 0.4506577398553849
Iteración 3: p2 = 0.4506577398553849, p3 = 0.45065674789059323
Iteración 4: p3 = 0.45065674789059323, p4 = 0.45065674788993576


Iteración 1: p0 = 1.5, p1 = 1.745487757282019
Iteración 2: p1 = 1.745487757282019, p2 = 1.7447374072120456
Iteración 3: p2 = 1.7447374072120456, p3 = 1.7447380533683496
Iteración 4: p3 = 1.7447380533683496, p4 = 1.7447380533688273


Iteración 1: p0 = 2.5, p1 = 2.2853594225939764
Iteración 2: p1 = 2.2853594225939764, p2 = 2.2419356088744236
Iteración 3: p2 = 2.2419356088744236, p3 = 2.2383458848583833
Iteración 4: p3 = 2.2383458848583833, p4 = 2.238319796456584
Iteración 5: p4 = 2.238319796456584, p5 = 2.2383197950741383
Iteración 6: p5 = 2.2383197950741383, p6 = 2.2383197950741383


Iteración 1: p0 = 3.5, p1 = 3.7116038835749867
Iteración 2: p1 = 3.7116038835749867, p2 = 3.709036211970711
Iteración 3: p2 = 3.709036211970711, p3 = 3.709041201357361
```

```
Iteración 4: p3 = 3.709041201357361, p4 = 3.709041201375952


Cuatro ceros positivos más pequeños: [np.float64(0.45065674788993576), np.float64(1.74473805
```

```
# Literal D
# Encontrar el vigesimoquinto cero positivo
cero_25 = metodo_newton(f_6, f_prime_6,25.5)
print(f'Vigesimoquinto cero positivo: {cero_25}')
```

```
Iteración 1: p0 = 25.5, p1 = 25.50007665626619
Iteración 2: p1 = 25.50007665626619, p2 = 25.50007665391656
Iteración 3: p2 = 25.50007665391656, p3 = 25.50007665391656
Vigesimoquinto cero positivo: 25.50007665391656
```

Ejercicio 7

La función ( ) = (1 3/) tiene raíz en = 0.Usando el punto de inicio de = 1 y 0 = 5, 1 = 0.5 para el método de secante, compare los resultados de los métodos de la secante y de Newton.

```
def f7(x):
    return x**(1/3)

def f_prime7(x):
    return (1/3) * x**(-2/3)


raiz_newton = metodo_newton(f7, f_prime7, 1)
print('\n')
raiz_secante = metodo_secante(f7, 5, 0.5)

print(f'Resultado del Método de Newton: {raiz_newton}')
print('\n')
print(f'Resultado del Método de la Secante: {raiz_secante}')
```

```
Iteración 1: p0 = 1, p1 = -2.0
Iteración 2: p1 = -2.0, p2 = (4-2.2893847434456487e-15j)
Iteración 3: p2 = (4-2.2893847434456487e-15j), p3 = (-7.999999999999998+4.5787694868912965e-1
Iteración 4: p3 = (-7.999999999999998+4.5787694868912965e-15j), p4 = (15.999999999999991-2.1
Iteración 5: p4 = (15.999999999999991-2.195811558520189e-14j), p5 = (-31.99999999999998+4.39
Iteración 6: p5 = (-31.99999999999998+4.391623117040377e-14j), p6 = (63.999999999999936-1.18
Iteración 7: p6 = (63.999999999999936-1.1889466323432573e-13j), p7 = (-127.99999999999983+2.
Iteración 8: p7 = (-127.99999999999983+2.377893264686514e-13j), p8 = (255.9999999999996-6.41
Iteración 9: p8 = (255.9999999999996-6.413344150144774e-13j), p9 = (-511.99999999999903+1.28
Iteración 10: p9 = (-511.99999999999903+1.282668830028954e-12j), p10 = (1023.9999999999977-2
Iteración 11: p10 = (1023.9999999999977-2.1878942263561007e-12j), p11 = (-2047.9999999999945
Iteración 12: p11 = (-2047.9999999999945+4.3757884527122e-12j), p12 = (4095.999999999988-1.0
Iteración 13: p12 = (4095.999999999988-1.0137639848509385e-11j), p13 = (-8191.99999999972+2
Iteración 14: p13 = (-8191.99999999972+2.0275279697018756e-11j), p14 = (16383.9999999993-3
Iteración 15: p14 = (16383.9999999993-3.59886397579014e-11j), p15 = (-32767.999999999844+7.
Iteración 16: p15 = (-32767.999999999844+7.197727951580276e-11j), p16 = (65535.999999999636-
Iteración 17: p16 = (65535.999999999636-1.548646291908965e-10j), p17 = (-131071.99999999914+
Iteración 18: p17 = (-131071.99999999914+3.097292583817926e-10j), p18 = (262143.99999999808-
Iteración 19: p18 = (262143.99999999808-5.953132449148943e-10j), p19 = (-524287.99999999563+
Iteración 20: p19 = (-524287.99999999563+1.1906264898297877e-09j), p20 = (1048575.9999999905
Iteración 21: p20 = (1048575.9999999905-2.5302940501222528e-09j), p21 = (-2097151.9999999783
Iteración 22: p21 = (-2097151.9999999783+5.0605881002445005e-09j), p22 = (4194303.9999999534
Iteración 23: p22 = (4194303.9999999534-8.852382300433e-09j), p23 = (-8388607.999999896+1.77
Iteración 24: p23 = (-8388607.999999896+1.770476460086598e-08j), p24 = (16777215.999999776-4
Iteración 25: p24 = (16777215.999999776-4.0217955610121755e-08j), p25 = (-33554431.99999951+
Iteración 26: p25 = (-33554431.99999951+8.043591122024339e-08j), p26 = (67108863.99999891-1.
Iteración 27: p26 = (67108863.99999891-1.577030033366861e-07j), p27 = (-134217727.99999763+3
Iteración 28: p27 = (-134217727.99999763+3.1540600667337177e-07j), p28 = (268435455.999995-5
Iteración 29: p28 = (268435455.999995-5.94381275179622e-07j), p29 = (-536870911.99998915+1.1
Iteración 30: p29 = (-536870911.99998915+1.1887625503592426e-06j), p30 = (1073741823.9999768
Iteración 31: p30 = (1073741823.9999768-2.518291543143952e-06j), p31 = (-2147483647.9999504+
Iteración 32: p31 = (-2147483647.9999504+5.036583086287896e-06j), p32 = (4294967295.999893-9
Iteración 33: p32 = (4294967295.999893-9.712665815302494e-06j), p33 = (-8589934591.999771+1.
Iteración 34: p33 = (-8589934591.999771+1.9425331630604957e-05j), p34 = (17179869183.999512-
Iteración 35: p34 = (17179869183.999512-4.003991368540312e-05j), p35 = (-34359738367.99897+8
Iteración 36: p35 = (-34359738367.99897+8.007982737080612e-05j), p36 = (68719476735.997826-0
Iteración 37: p36 = (68719476735.997826-0.00016341509413929687j), p37 = (-137438953471.99539
Iteración 38: p37 = (-137438953471.99539+0.00032683018827859303j), p38 = (274877906943.9902-
Iteración 39: p38 = (274877906943.9902-0.0006171217414231852j), p39 = (-549755813887.97925+0
Iteración 40: p39 = (-549755813887.97925+0.0012342434828463682j), p40 = (1099511627775.9563-
Iteración 41: p40 = (1099511627775.9563-0.0024922452236234433j), p41 = (-2199023255551.908+0
Iteración 42: p41 = (-2199023255551.908+0.004984490447246874j), p42 = (4398046511103.8047-0.
Iteración 43: p42 = (4398046511103.8047-0.010199603119741985j), p43 = (-8796093022207.59+0.0
Iteración 44: p43 = (-8796093022207.59+0.02039920623948392j), p44 = (17592186044415.137-0.04
Iteración 45: p44 = (17592186044415.137-0.04202748137812645j), p45 = (-35184372088830.2+0.08
Iteración 46: p45 = (-35184372088830.2+0.08405496275625271j), p46 = (70368744177660.22-0.159
Iteración 47: p46 = (70368744177660.22-0.15951468205811228j), p47 = (-140737488355320.06+0.3
Iteración 48: p47 = (-140737488355320.06+0.3190293641162236j), p48 = (281474976710639.3-0.678
La derivada es muy cercana a cero, el método no converge.


Iteración 1: p0 = 5, p1 = 0.5, p2 = -3.3980117618223327
Iteración 2: p1 = 0.5, p2 = -3.3980117618223327, p3 = (0.42342548212957754-2.373791283892393j
Iteración 3: p2 = -3.3980117618223327, p3 = (0.42342548212957754-2.373791283892393j), p4 = (-
Iteración 4: p3 = (0.42342548212957754-2.373791283892393j), p4 = (-2.506434572816832-3.343001
Iteración 5: p4 = (-2.506434572816832-3.343001585580911j), p5 = (1.537946395028479+6.1258226
Iteración 6: p5 = (1.537946395028479+6.125822678839629j), p6 = (-7.895622939891771+3.3258892
Iteración 7: p6 = (-7.895622939891771+3.325889225481498j), p7 = (6.554151215536118-12.735088
Iteración 8: p7 = (6.554151215536118-12.735088076137382j), p8 = (-12.95378376757566-12.317401
Iteración 9: p8 = (-12.95378376757566-12.317401136903326j), p9 = (5.468753662426424+30.80985
Iteración 10: p9 = (5.468753662426424+30.80985385003664j), p10 = (-35.05958741642519+15.8869
Iteración 11: p10 = (-35.05958741642519+15.886948700273436j), p11 = (32.050165663512885-59.72
Iteración 12: p11 = (32.050165663512885-59.72033599449141j), p12 = (-60.69294161558166-56.558
Iteración 13: p12 = (-60.69294161558166-56.55878015192357j), p13 = (25.003712830447+144.3311
Iteración 14: p13 = (25.003712830447+144.33113207100942j), p14 = (-163.0538389578072+74.2005
Iteración 15: p14 = (-163.0538389578072+74.20051941998942j), p15 = (149.90944093589647-278.65
Iteración 16: p15 = (149.90944093589647-278.65013651493945j), p16 = (-283.17526560153624-263
Iteración 17: p16 = (-283.17526560153624-263.571162582952j), p17 = (116.48968158137973+673.40
Iteración 18: p17 = (116.48968158137973+673.4027429444614j), p18 = (-760.4311803892363+346.13
Iteración 19: p18 = (-760.4311803892363+346.13437169303984j), p19 = (699.3656185288155-1299.7
Iteración 20: p19 = (699.3656185288155-1299.7857828780218j), p20 = (-1320.8900563245384-1229
Iteración 21: p20 = (-1320.8900563245384-1229.3584115710375j), p21 = (543.326975583032+3141.1
Iteración 22: p21 = (543.326975583032+3141.131427360435j), p22 = (-3546.9914314657844+1614.54
Iteración 23: p22 = (-3546.9914314657844+1614.5496089745213j), p23 = (3262.219386939554-6062
Iteración 24: p23 = (3262.219386939554-6062.851826393732j), p24 = (-6161.291746869603-5734.31
Iteración 25: p24 = (-6161.291746869603-5734.318375773134j), p25 = (2534.3357456777176+14651
Iteración 26: p25 = (2534.3357456777176+14651.80747770748j), p26 = (-16544.916653716667+7531
Iteración 27: p26 = (-16544.916653716667+7531.061937836286j), p27 = (15216.617409777293-28280
Iteración 28: p27 = (15216.617409777293-28280.14888843895j), p28 = (-28739.32124619665-26747
Iteración 29: p28 = (-28739.32124619665-26747.699000810822j), p29 = (11821.395591438351+68343
Iteración 30: p29 = (11821.395591438351+68343.29858220498j), p30 = (-77173.69207132887+35128
Iteración 31: p30 = (-77173.69207132887+35128.60761493815j), p31 = (70977.8509583179-131912.6
Iteración 32: p31 = (70977.8509583179-131912.63726519077j), p32 = (-134054.44474997328-124764
Iteración 33: p32 = (-134054.44474997328-124764.52921812683j), p33 = (55140.84971670003+31878
Iteración 34: p33 = (55140.84971670003+318787.0327479243j), p34 = (-359976.3646597736+163857
Iteración 35: p34 = (-359976.3646597736+163857.2441495246j), p35 = (331075.8910084634-615305
Iteración 36: p35 = (331075.8910084634-615305.947571132j), p36 = (-625296.3996501989-581963.6
Iteración 37: p36 = (-625296.3996501989-581963.6268083638j), p37 = (257204.26375471032+148698
Iteración 38: p37 = (257204.26375471032+1486980.7875711839j), p38 = (-1679108.2543366943+7643
Iteración 39: p38 = (-1679108.2543366943+764311.4333198144j), p39 = (1544302.115683448-287009
Iteración 40: p39 = (1544302.115683448-2870092.0312642083j), p40 = (-2916692.4533965536-2714
Iteración 41: p40 = (-2916692.4533965536-2714566.91470746j), p41 = (1199728.2176059298+69360
Iteración 42: p41 = (1199728.2176059298+6936015.694124965j), p42 = (-7832193.462985488+35651
Iteración 43: p42 = (-7832193.462985488+3565127.4992759265j), p43 = (7203390.790478623-13387
Iteración 44: p43 = (7203390.790478623-13387532.33304789j), p44 = (-13604899.808107963-12662
Iteración 45: p44 = (-13604899.808107963-12662086.074203722j), p45 = (5596127.277121918+3235
Iteración 46: p45 = (5596127.277121918+32353016.334013768j), p46 = (-36533233.806475356+1662
Iteración 47: p46 = (-36533233.806475356+16629522.38048752j), p47 = (33600186.357996166-6244
Iteración 48: p47 = (33600186.357996166-62446088.84173228j), p48 = (-63459998.52435477-59062
Iteración 49: p48 = (-63459998.52435477-59062247.78702162j), p49 = (26103112.390179977+15091
Iteración 50: p49 = (26103112.390179977+150910510.02017367j), p50 = (-170409117.02025253+775
Iteración 51: p50 = (-170409117.02025253+77568337.92318794j), p51 = (156727929.40567666-2912
Iteración 52: p51 = (156727929.40567666-291279521.4696274j), p52 = (-296008898.963808-275495
Iteración 53: p52 = (-296008898.963808-275495608.9552074j), p53 = (121757859.08945823+703921
Iteración 54: p53 = (121757859.08945823+703921445.8221972j), p54 = (-794872617.0108638+36181
Iteración 55: p54 = (-794872617.0108638+361817189.36352414j), p55 = (731056774.3307936-13586
Iteración 56: p55 = (731056774.3307936-1358672115.4403884j), p56 = (-1380732277.0759008-1285
Iteración 57: p56 = (-1380732277.0759008-1285048121.217611j), p57 = (567939026.9041507+32834
Iteración 58: p57 = (567939026.9041507+3283438653.9557257j), p58 = (-3707680013.379861+16876
Iteración 59: p58 = (-3707680013.379861+1687694773.7175364j), p59 = (3410011280.832924-63375
Iteración 60: p59 = (3410011280.832924-6337520426.981846j), p60 = (-6440419959.104992-599410
Iteración 61: p60 = (-6440419959.104992-5994101612.390502j), p61 = (2649149226.9410477+15315
Iteración 62: p61 = (2649149226.9410477+15315585934.02161j), p62 = (-17294457989.145466+7872
Iteración 63: p62 = (-17294457989.145466+7872245246.9546385j), p63 = (15905983425.230137-295
Iteración 64: p63 = (15905983425.230137-29561337651.64799j), p64 = (-30041312090.915806-2795
Iteración 65: p64 = (-30041312090.915806-27959462020.4718j), p65 = (12356945541.949398+71439
Iteración 66: p65 = (12356945541.949398+71439486837.92389j), p66 = (-80669927301.96915+36720
Iteración 67: p66 = (-80669927301.96915+36720055186.099655j), p67 = (74193393478.12894-13788
Iteración 68: p67 = (74193393478.12894-137888736426.6706j), p68 = (-140127575200.73572-13041
Iteración 69: p68 = (-140127575200.73572-130416794212.87677j), p69 = (57638921044.48114+3332
Iteración 70: p69 = (57638921044.48114+333229188987.7292j), p70 = (-376284540110.44336+17128
Iteración 71: p70 = (-376284540110.44336+171280544567.86832j), p71 = (346074775047.7942-6431
Iteración 72: p71 = (346074775047.7942-643181437098.5977j), p72 = (-653624491241.0298-608328
Iteración 73: p72 = (-653624491241.0298-608328593744.4097j), p73 = (268856507289.24646+15543
Iteración 74: p73 = (268856507289.24646+1554346164962.554j), p74 = (-1755177671056.0112+7989
Iteración 75: p74 = (-1755177671056.0112+798937387179.3917j), p75 = (1614264347669.812-30001
Iteración 76: p75 = (1614264347669.812-3000117136095.5522j), p76 = (-3048828718673.5825-2837
Iteración 77: p76 = (-3048828718673.5825-2837546193345.3423j), p77 = (1254080059132.083+7250
Iteración 78: p77 = (1254080059132.083+7250241216482.281j), p78 = (-8187018940691.53+372664
Iteración 79: p78 = (-8187018940691.53+3726640116911.3105j), p79 = (7529729330310.091-139940
Iteración 80: p79 = (7529729330310.091-13994033893292.209j), p80 = (-14221248867464.852-1323
Iteración 81: p80 = (-14221248867464.852-13235722407537.46j), p81 = (5849651215697.48+338187
Iteración 82: p81 = (5849651215697.48+33818719975060.93j), p82 = (-38188315770285.86+173828
Iteración 83: p82 = (-38188315770285.86+17382897313146.191j), p83 = (35122391118637.97-65275
Iteración 84: p83 = (35122391118637.97-65275112844918.66j), p84 = (-66334956146161.72-617379
Iteración 85: p84 = (-66334956146161.72-61737972076096.74j), p85 = (27285673746373.875+15774
Iteración 86: p85 = (27285673746373.875+157747278552822.44j), p86 = (-178129239951151.47+810
Iteración 87: p86 = (-178129239951151.47+81082452160640.78j), p87 = (163828247175489.78-3044
Iteración 88: p87 = (163828247175489.78-304475492156640.9j), p88 = (-309419126823678.1-28797
Iteración 89: p88 = (-309419126823678.1-287976513763864.3j), p89 = (127273911612999.31+73581
Iteración 90: p89 = (127273911612999.31+735811524184600j), p90 = (-830883098286933.5+378208
Iteración 91: p90 = (-830883098286933.5+378208760596577.4j), p91 = (764176177012924-14202246
Iteración 92: p91 = (764176177012924-1420224665781568j), p92 = (-1443284229108021.8-13432652
Iteración 93: p92 = (-1443284229108021.8-1343265249745667.8j), p93 = (593668630939568.8+3432
Iteración 94: p93 = (593668630939568.8+3432189791734300j), p94 = (-3875650753398008+17641531
Iteración 95: p94 = (-3875650753398008+1764153189503988.8j), p95 = (3564496596783791-6624632
Iteración 96: p95 = (3564496596783791-6624632041835007j), p96 = (-6732193278985523-626565516
Iteración 97: p96 = (-6732193278985523-6265655166080094j), p97 = (2769164857864443+1.6009434
Iteración 98: p97 = (2769164857864443+1.6009434997011262e+16j), p98 = (-1.8077956806779764e+
Iteración 99: p98 = (-1.8077956806779764e+16+8228885209131412j), p99 = (1.6626579538435876e+
Iteración 100: p99 = (1.6626579538435876e+16-3.0900568584024804e+16j), p100 = (-3.140228752
No se alcanzó la tolerancia en el número máximo de iteraciones.
Resultado del Método de Newton: None


Resultado del Método de la Secante: (1.2916757952824988e+16+7.467594290408362e+16j)
```