Escuela Politécnica nacional

Tarea N°5

Erick Carcelén

1. Sea $f(x) = -x^3 - cosx \;\; y \;\; p_0 = -1$. Utilice el método de Newton y de la Secante para encontrar p_2. ¿Se podría usar $p_0 = 0$?

```
import numpy as np

def newton_method(f, df, p0, tol=1e-6, max_iter=100):
    for i in range(max_iter):
        p = p0 - f(p0)/df(p0)
        if abs(p - p0) < tol:
            return p, i + 1
        p0 = p
    return p, max_iter
```

```
f1 = lambda x: -x**3 - np.cos(x)
df1 = lambda x: -3*x**2 + np.sin(x)

p0_newton = -1
p2_newton, iterations_newton = newton_method(f1, df1, p0_newton)
print(f"Método de Newton: p2 = {p2_newton}, iteraciones = {iterations_newton}")
```

```
Método de Newton: p2 = -0.8654740331016162, iteraciones = 4
```

```
def secant_method(f, p0, p1, tol=1e-6, max_iter=100):
    for i in range(max_iter):
        q0, q1 = f(p0), f(p1)
        p = p1 - q1 * (p1 - p0) / (q1 - q0)
```

```
        if abs(p - p1) < tol:
            return p, i + 1
        p0, p1 = p1, p
    return p, max_iter
```

```
p0_secant, p1_secant = -1, -0.5
p2_secant, iterations_secant = secant_method(f1, p0_secant, p1_secant)
print(f"Método de la secenate: p2 = {p2_secant}, iteraciones = {iterations_secant}")
```

```
Método de la secenate: p2 = -0.8654740330997166, iteraciones = 6
```

```
# Evaluamos la función y su derivada en x = 0
f_at_0 = f1(0)
df_at_0 = df1(0)

print(f"f(0) = {f_at_0}")
print(f"f'(0) = {df_at_0}")

# Intentamos aplicar el Método de Newton desde p0 = 0
try:
    p0_newton = 0
    p2_newton, iterations_newton = newton_method(f1, df1, p0_newton)
    print(f"Métodod de Newton: p2 = {p2_newton}, iteraciones = {iterations_newton}")
except ZeroDivisionError:
    print("No es posible ejecutar el método debido a una división por 0 en p0 = 0.")
```

```
f(0) = -1.0
f'(0) = 0.0
Métodod de Newton: p2 = nan, iteraciones = 100
```

```
C:\Users\Erick\AppData\Local\Temp\ipykernel_7580\2482269434.py:5: RuntimeWarning: divide by z
  p = p0 - f(p0)/df(p0)
C:\Users\Erick\AppData\Local\Temp\ipykernel_7580\1908360107.py:1: RuntimeWarning: invalid val
  f1 = lambda x: -x**3 - np.cos(x)
C:\Users\Erick\AppData\Local\Temp\ipykernel_7580\1908360107.py:2: RuntimeWarning: invalid val
  df1 = lambda x: -3*x**2 + np.sin(x)
```

El resultado obetnido confirma que no se puede usar $p_0 = 0$ en el método de Newton debido a la división por cero en la derivada

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

a. $x^3 - 2x^2 - 5 = 0,\ [1, 4]$

```
f2a = lambda x: x**3 - 2*x**2 - 5
df2a = lambda x: 3*x**2 - 4*x

p0_newton_a = 1
root_newton_a, iterations_newton_a = newton_method(f2a, df2a, p0_newton_a, tol=1e-4)
print(f"Métodod de Newton (a): raíz = {root_newton_a}, iteraciones = {iterations_newton_a}")

p0_secant_a, p1_secant_a = 1, 4
root_secant_a, iterations_secant_a = secant_method(f2a, p0_secant_a, p1_secant_a, tol=1e-4)
print(f"Método de la Secante (a): raíz = {root_secant_a}, iteraciones = {iterations_secant_a
```

```
Métodod de Newton (a): raíz = 2.6906474480286287, iteraciones = 22
Método de la Secante (a): raíz = 2.690647447883773, iteraciones = 10
```

b. $x^3 + 3x^2 - 1 = 0,\ [-3, -2]$

```
f2b = lambda x: x**3 + 3*x**2 - 1
df2b = lambda x: 3*x**2 + 6*x

p0_newton_b = -3
root_newton_b, iterations_newton_b = newton_method(f2b, df2b, p0_newton_b, tol=1e-4)
print(f"Método de Newton (b): raíz = {root_newton_b}, iteraciones = {iterations_newton_b}")

p0_secant_b, p1_secant_b = -3, -2
root_secant_b, iterations_secant_b = secant_method(f2b, p0_secant_b, p1_secant_b, tol=1e-4)
print(f"Método de la Secante (b): raíz = {root_secant_b}, iteraciones = {iterations_secant_b
```

```
Método de Newton (b): raíz = -2.8793852448366706, iteraciones = 3
Método de la Secante (b): raíz = -2.879385194736809, iteraciones = 6
```

c. $x - cosx = 0,\ [0, \pi/2]$

```
f2c = lambda x: x - np.cos(x)
df2c = lambda x: 1 + np.sin(x)

p0_newton_c = 0
root_newton_c, iterations_newton_c = newton_method(f2c, df2c, p0_newton_c, tol=1e-4)
print(f"Método de Newton (c): raíz = {root_newton_c}, iteraciones = {iterations_newton_c}")
```

```
p0_secant_c, p1_secant_c = 0, np.pi/2
root_secant_c, iterations_secant_c = secant_method(f2c, p0_secant_c, p1_secant_c, tol=1e-4)
print(f"Método de la Secante (c): raíz = {root_secant_c}, iteraciones = {iterations_secant_c}
```

```
Método de Newton (c): raíz = 0.739085133385284, iteraciones = 4
Método de la Secante (c): raíz = 0.739085133034638, iteraciones = 5
```

d. $x - 0.8 - 0.2 sen x = 0,\ [0, \pi/2]$

```
f2d = lambda x: x - 0.8 - 0.2 * np.sin(x)
df2d = lambda x: 1 - 0.2 * np.cos(x)

p0_newton_d = 0
root_newton_d, iterations_newton_d = newton_method(f2d, df2d, p0_newton_d, tol=1e-4)
print(f"Método de Newton (d): raíz = {root_newton_d}, iteraciones = {iterations_newton_d}")

p0_secant_d, p1_secant_d = 0, np.pi/2
root_secant_d, iterations_secant_d = secant_method(f2d, p0_secant_d, p1_secant_d, tol=1e-4)
print(f"Método de la Secante (d): raíz = {root_secant_d}, iteraciones = {iterations_secant_d}
```

```
Método de Newton (d): raíz = 0.9643338876952227, iteraciones = 4
Método de la Secante (d): raíz = 0.964333884548886, iteraciones = 4
```

3. Use los 2 métodos en esta sección para encontrar las soluciones dentro de 10^{-5} para los siguientes problemas:

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

```
f3a = lambda x: 3*x - np.exp(x)
df3a = lambda x: 3 - np.exp(x)

p0_newton_3a = 1
root_newton_3a, iterations_newton_3a = newton_method(f3a, df3a, p0_newton_3a, tol=1e-5)
print(f"Método de Newton (3a): raíz = {root_newton_3a}, iteraciones = {iterations_newton_3a}"

p0_secant_3a, p1_secant_3a = 1, 2
root_secant_3a, iterations_secant_3a = secant_method(f3a, p0_secant_3a, p1_secant_3a, tol=1e-
print(f"Método de la Secante (3a): raíz = {root_secant_3a}, iteraciones = {iterations_secant_
```

```
Método de Newton (3a): raíz = 0.6190612867359452, iteraciones = 6
Método de la Secante (3a): raíz = 1.5121345517620621, iteraciones = 9
```

b. $2+3cosx-e^{x}=0 $ para $1 \leq x \leq 2$

```
f3b = lambda x: 2*x + 3*np.cos(x) - np.exp(x)
df3b = lambda x: 2 - 3*np.sin(x) - np.exp(x)

p0_newton_3b = 1
root_newton_3b, iterations_newton_3b = newton_method(f3b, df3b, p0_newton_3b, tol=1e-5)
print(f"Método de Newton (3b): raíz = {root_newton_3b}, iteraciones = {iterations_newton_3b}"

p0_secant_3b, p1_secant_3b = 1, 2
root_secant_3b, iterations_secant_3b = secant_method(f3b, p0_secant_3b, p1_secant_3b, tol=1e
print(f"Método de la Secante (3b): raíz = {root_secant_3b}, iteraciones = {iterations_secant_
```

```
Método de Newton (3b): raíz = 1.2397146979752596, iteraciones = 4
Método de la Secante (3b): raíz = 1.2397146979752531, iteraciones = 6
```

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 $[0,]$. Intente aproximar estos ceros dentro de 10^{-6} con:

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

```
f4 = lambda x: 230*x**4 + 18*x**3 + 9*x**2 - 221*x - 9

p0_secant_4a, p1_secant_4a = -1, 0
root_secant_4a, iterations_secant_4a = secant_method(f4, p0_secant_4a, p1_secant_4a, tol=1e-
print(f"Método de la Secante (4a): raíz = {root_secant_4a}, iteraciones = {iterations_secant_

p0_secant_4b, p1_secant_4b = 0, 1
root_secant_4b, iterations_secant_4b = secant_method(f4, p0_secant_4b, p1_secant_4b, tol=1e-
print(f"Método de la Secante (4b): raíz = {root_secant_4b}, iteraciones = {iterations_secant_
```

```
Método de la Secante (4a): raíz = -0.040659288315725135, iteraciones = 4
Método de la Secante (4b): raíz = -0.04065928831557162, iteraciones = 11
```

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

```
df4 = lambda x: 920*x**3 + 54*x**2 + 18*x - 221

p0_newton_4a = -0.5
root_newton_4a, iterations_newton_4a = newton_method(f4, df4, p0_newton_4a, tol=1e-6)
print(f"Método de Newton (4a): raíz = {root_newton_4a}, iteraciones = {iterations_newton_4a}"

p0_newton_4b = 0.5
root_newton_4b, iterations_newton_4b = newton_method(f4, df4, p0_newton_4b, tol=1e-6)
print(f"Método de Newton (4b): raíz = {root_newton_4b}, iteraciones = {iterations_newton_4b}"
```

```
Método de Newton (4a): raíz = -0.04065928831575899, iteraciones = 4
Método de Newton (4b): raíz = -0.040659288315758865, iteraciones = 6
```

5. La función $f(x) = tan\pi x - 6$ tiene cero en $(1/\pi)$ arcotangente $6 \approx 0.447431543$. Sea $p_0 = 0$ y $p_1 = 0.48$ 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

```
import numpy as np

# Función f(x)
f5 = lambda x: np.tan(np.pi * x) - 6

# Derivada f'(x)
df5 = lambda x: np.pi / (np.cos(np.pi * x))**2
```

```
def bisection_method(f, a, b, tol=1e-6, max_iter=10):
    if f(a) * f(b) > 0:
        raise ValueError("f(a) and f(b) must have opposite signs")
    for i in range(max_iter):
        c = (a + b) / 2
        if abs(f(c)) < tol or (b - a) / 2 < tol:
            return c, i + 1
        if f(c) * f(a) > 0:
            a = c
        else:
            b = c
```

```
    return c, max_iter

a, b = 0, 0.48
root_bisection, iterations_bisection = bisection_method(f5, a, b, tol=1e-6, max_iter=10)
print(f"Método de la bisección: raíz = {root_bisection}, iteraciones = {iterations_bisection}
```

```
Método de la bisección: raíz = 0.44765625, iteraciones = 10
```

b. Método de Newton

```
p0_newton_5 = 0
root_newton, iterations_newton = newton_method(f5, df5, p0_newton_5, tol=1e-6, max_iter=10)
print(f"Método de Newton: raíz = {root_newton}, iteraciones = {iterations_newton}")
```

```
Método de Newton: raíz = 13.655012218663435, iteraciones = 10
```

c. Método de la Secante

```
p0_secant_5, p1_secant_5 = 0, 0.48
root_secant_5, iterations_secant_5 = secant_method(f5, p0_secant_5, p1_secant_5, tol=1e-6, ma
print(f"Método de la Secante: raíz = {root_secant_5}, iteracions = {iterations_secant_5}")
```

```
Método de la Secante: raíz = -2989.9400375314453, iteracions = 10
```

El método de la bisección ha encontrado una raíz cercana a 0.44765625 después de 10 iteraciones, lo cual es bastante preciso considerando la raíz verdadera aproximada 0.447431543.

El método de Newton parece diverger, resultando en una raíz no razonable 13.655012218663435. Esto puede suceder si el punto inicial está demasiado lejos de la raíz o si la derivada en algún punto es muy grande o cero.

El método de la secante también parece diverger, resultando en una raíz no razonable -2989.9400375314453. Esto puede suceder si las iteraciones iniciales están lejos de la raíz y la función cambia rápidamente.

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

a. Determine, dentro de 10^{-6}, el único cero negativo.

b. Determinar, dentro de 10^{-6}, los cuatro ceros positivos más pequeños

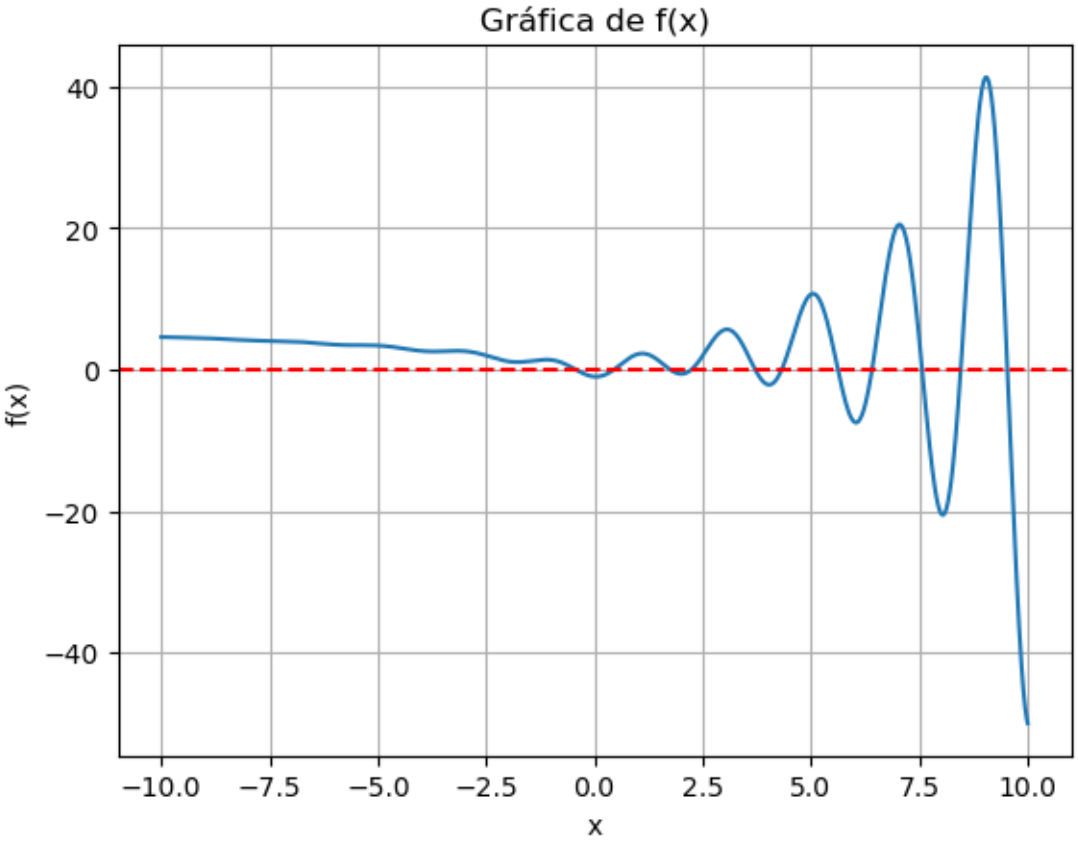
c. Determinar una aproximación inicial razonable para encontrar el enésimo cero positivo más pequeño de f.

[Sugerencia: Dibuje una gráfica aproximada de f.]

d. Use la parte c) para determinar, dentro de 10^{-6}, el vigesimoquinto cero positivo más pequeño de f.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import newton, bisect

# Definimos la función y su derivada
def f(x):
    return np.log(x**2 + 1) - np.exp(0.4*x) * np.cos(np.pi * x)

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

# Encontramos el único cero negativo
cero_negativo = newton(f, -1, fprime=df, tol=1e-6)

# Encontramos los cuatro ceros positivos más pequeños
ceros_positivos = []
for i in range(1, 5):
    cero = newton(f, i, fprime=df, tol=1e-6)
    ceros_positivos.append(cero)

# Aproximación inicial para el enésimo cero positivo más pequeño
def aproximacion_inicial(n):
    return n

# Vigesimoquinto cero positivo más pequeño
vigesimoquinto_cero = newton(f, aproximacion_inicial(25), fprime=df, tol=1e-6)

cero_negativo, ceros_positivos, vigesimoquinto_ceroimport matplotlib.pyplot as plt

x = np.linspace(-10, 10, 1000)
y = f6(x)
```

```
plt.plot(x, y)
plt.axhline(0, color='red', linestyle='--')
plt.title("Gráfica de f(x)")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.grid(True)
plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import newton, bisect

# Definimos la función y su derivada
def f(x):
    return np.log(x**2 + 1) - np.exp(0.4*x) * np.cos(np.pi * x)
```

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

# Encontramos el único cero negativo
cero_negativo = newton(f, -1, fprime=df, tol=1e-6)

# Encontramos los cuatro ceros positivos más pequeños
ceros_positivos = []
for i in range(1, 5):
    cero = newton(f, i, fprime=df, tol=1e-6)
    ceros_positivos.append(cero)

# Aproximación inicial para el enésimo cero positivo más pequeño
def aproximacion_inicial(n):
    return n

# Vigesimoquinto cero positivo más pequeño
vigesimoquinto_cero = newton(f, aproximacion_inicial(25), fprime=df, tol=1e-6)

cero_negativo, ceros_positivos, vigesimoquinto_cero
```

```
(0.4506567478899403,
 [-0.4341430472857333,
  0.4506567478899357,
  1.744738053368827,
  2.2383197950741383],
 22.49975528564899)
```

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.

```
f7 = lambda x: x**(1/3)
df7 = lambda x: (1/3) * x**(-2/3)

p0_newton_7 = 5
root_newton_7, iterations_newton_7 = newton_method(f7, df7, p0_newton_7, tol=1e-6)
print(f"Método de Newton (7): raíz = {root_newton_7}, iteraciones = {iterations_newton_7}")

p0_secant_7, p1_secant_7 = 5, 0.5
```

```
root_secant_7, iterations_secant_7 = secant_method(f7, p0_secant_7, p1_secant_7, tol=1e-6)
print(f"Método de la Secante (7): raíz = {root_secant_7}, iteraciones = {iterations_secant_7
```

```
Método de Newton (7): raíz = (5119.999999999987-1.2612400128951961e-11j), iteraciones = 10
Método de la Secante (7): raíz = (1.2916757952824988e+16+7.467594290408362e+16j), iteraciones
```