

Tarea 5

Boris Garcés

Tabla de Contenidos

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 . ¿Se podría usar $p_0=0$?

- Método de Newton

```
from scipy.optimize import newton
import math
def funcion(x):
    return -x**3 - math.cos(x)
def derivada_funcion(x):
    return -3 * x**2 + math.sin(x)
x0 = -1
raiz = newton(funcion, x0, fprime=derivada_funcion)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: -0.8654740331016144

- Método de la secante

```
from scipy.optimize import newton
import math
def funcion(x):
    return -x**3 - math.cos(x)
x0 = -1
raiz = newton(funcion, x0)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: -0.8654740331016144

Al utilizar $p_0 = 0$ * Método de Newton

Con el método de newton el siguiente punto a evaluar se determina a través de la fórmula: $x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$, $n \geq 1$ por lo tanto: $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$ A continuación determinaremos la derivada de la función para evaluarla en cero: $f'(x) = \sin(x) - 3x^2$ Al evaluar la derivada en el punto 0 obtendremos que el resultado es 0, como el método de newton intentará dividir para cero al determinar el siguiente valor del método resultará en un error. * Método de la secante

Al utilizar el método de la secante el siguiente punto a evaluar se determina a través de la ecuación

$x_n = x_{n-1} - f(x_{n-1}) * \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$ dado que los puntos x_{n-1} y x_{n-2} son cercanos pero nunca iguales el programa seguirá su ejecución hasta determinar un punto que se determine como raíz de la función tomando en cuenta la tolerancia dada.

```
from scipy.optimize import newton
import math
def funcion(x):
    return -x**3 - math.cos(x)
x0 = 0
raiz = newton(funcion, x0)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: -4.998000183473029e-09

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

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

```
from scipy.optimize import newton
import math
def funcion(x):
    return 3*x-math.e**x
def derivada_funcion(x):
    return 3-math.e**x
x0 = 0.5
raiz = newton(funcion, x0, fprime=derivada_funcion, tol=1e-4)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: 0.6190612833553125

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

```
from scipy.optimize import newton
import math
def funcion(x):
    return x**3+3*x**2-1
def derivada_funcion(x):
    return 3*x**2+6*x
x0 = -3.5
raiz = newton(funcion, x0, fprime=derivada_funcion, tol=1e-4)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: -2.8793852415718177

c. $x - \cos x = 0$ $[0, \frac{\pi}{2}]$

```
from scipy.optimize import newton
import math
def funcion(x):
    return x-math.cos(x)
def derivada_funcion(x):
    return math.sin(x)+1
x0 = 1.2
raiz = newton(funcion, x0, fprime=derivada_funcion, tol=1e-4)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: 0.7390851332151607

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

```
from scipy.optimize import newton
import math
def funcion(x):
    return x-0.8-0.2*math.sin(x)
def derivada_funcion(x):
    return 1-(math.sin(x)/5)+x-(4/5)
x0 = 1.2
raiz = newton(funcion, x0, fprime=derivada_funcion, tol=1e-4)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: 0.9643355390637215

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 $1 \leq x \leq 2$ * Método de Newton

```
from scipy.optimize import newton
import math
def funcion(x):
    return 3*x-math.e**x
def derivada_funcion(x):
    return 3-math.e**x
x0 = 1.5
raiz = newton(funcion, x0, fprime=derivada_funcion, tol=1e-4)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: 1.5121345516578504

- Método de la secante

```
from scipy.optimize import newton
import math
def funcion(x):
    return 3*x-math.e**x
x0 = 1.5
raiz = newton(funcion, x0)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: 1.5121345516578502

b. $2x + 3\cos x - e^x$ para $1 \leq x \leq 2$

- Método de Newton

```
from scipy.optimize import newton
import math
def funcion(x):
    return 2 * x + 3 * math.cos(x) - math.exp(x)
def derivada_funcion(x):
    return -3 * math.sin(x) - math.exp(x) + 2
x0 = 1.5
raiz = newton(funcion, x0, fprime=derivada_funcion, tol=1e-4)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: 1.2397146979752212

4. El polinomio de cuatro 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

- El método de la secante (use los extremos como las estimaciones iniciales)
 - El método de Newton (use el punto medio como estimación inicial)
- Método de Newton

```
from scipy.optimize import newton
import math
def funcion(x):
    return 230*x**4+18*x**3+9*x**2-221*x-9
def derivada_funcion(x):
    return 920*x**3+54*x**2+18*x-221
x0 = -0.5
raiz = newton(funcion, x0, fprime=derivada_funcion, tol=1e-6)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: -0.04065928831575899

- Método de la secante

```
from scipy.optimize import newton
import math
def funcion(x):
    return 230 * x**4 + 18 * x**3 + 9 * x**2 - 221 * x - 9
x0 = -1
x1 = 0
raiz = newton(funcion, x0, x1=x1)
print(f"La raíz encontrada es: {raiz}")
```

La raíz encontrada es: -0.04065928831575775

5. La función $f(x) = \tan(\pi x) - 6$ tiene cero en $(\frac{1}{\pi}) \arctan(6) \approx 0.447431543$. 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 la bisección

```
from scipy.optimize import bisect
import math
def f(x):
    return math.tan(math.pi * x) - 6
a = 0
b = 0.48
tol = 1e-6
max_iters = 10
try:
    raiz = bisect(f, a, b, xtol=tol, maxiter=max_iters, full_output=True)
    print(f"La raíz aproximada es: {raiz}")
except RuntimeError as e:
    print(f"Error: {e}")
    print(f"Valor en el que se quedó: {raiz}")
```

Error: Failed to converge after 10 iterations.
Valor en el que se quedó: 0.44743154328874873

b. Método de Newton

```
from scipy.optimize import newton
import math
def f(x):
    return math.tan(math.pi * x) - 6
def df(x):
    return math.pi * (1 / math.cos(math.pi * x))**2
x0 = 0
try:
    raiz = newton(f, x0, fprime=df, tol=1e-6, maxiter=10)
    print(f"La raíz aproximada es: {raiz}")
except RuntimeError as e:
    print(f"Error: {e}")
```

Error: Failed to converge after 10 iterations, value is 13.655012218324751.

c. Método de la secante

```

from scipy.optimize import newton
import math
def f(x):
    return math.tan(math.pi * x) - 6
x0 = 0
x1 = 0.48
try:
    raiz = newton(f, x0, x1=x1, tol=1e-6, maxiter=10)
    print(f"La raíz aproximada es: {raiz}")
except RuntimeError as e:
    print(f"Error: {e}")

```

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

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

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

```

from scipy.optimize import newton
import math
def f(x):
    return math.log(x**2+1)-math.e**(0.4*x)*math.cos(math.pi*x)
x0 = 0
x1 = 0.48
raiz=0.1
while raiz>0:
    x0-=1
    try:
        raiz = newton(f, x0, x1=x1, tol=1e-6)
        print(f"La raíz aproximada es: {raiz}")
    except RuntimeError as e:
        print(f"Error: {e}")

```

La raíz aproximada es: 3.6380349014923588
 La raíz aproximada es: 6.406933614144502
 La raíz aproximada es: 0.45065674788805005
 La raíz aproximada es: 0.45065674788899096
 La raíz aproximada es: 3.709041201375954
 La raíz aproximada es: 0.45065674789008753
 La raíz aproximada es: -0.4341430472855665

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

```
from scipy.optimize import newton
import math
def f(x):
    return math.log(x**2 + 1) - math.e**(0.4 * x) * math.cos(math.pi * x)
x0 = 0
x1 = 0.48
menor = []
while len(menor) < 4:
    x0 += 0.01
    try:
        raiz = newton(f, x0, x1=x1, tol=1e-6)
        if round(raiz,4) not in menor and raiz>0:
            menor.append(round(raiz,4))
            print(f"La raíz aproximada es: {raiz}")
    except RuntimeError as e:
        print(f"Error: {e}")
print(menor)
```

La raíz aproximada es: 0.45065674788992754

La raíz aproximada es: 3.7090412013759475

La raíz aproximada es: 1.744738053388946

La raíz aproximada es: 2.2383197950741427

[0.4507, 3.709, 1.7447, 2.2383]

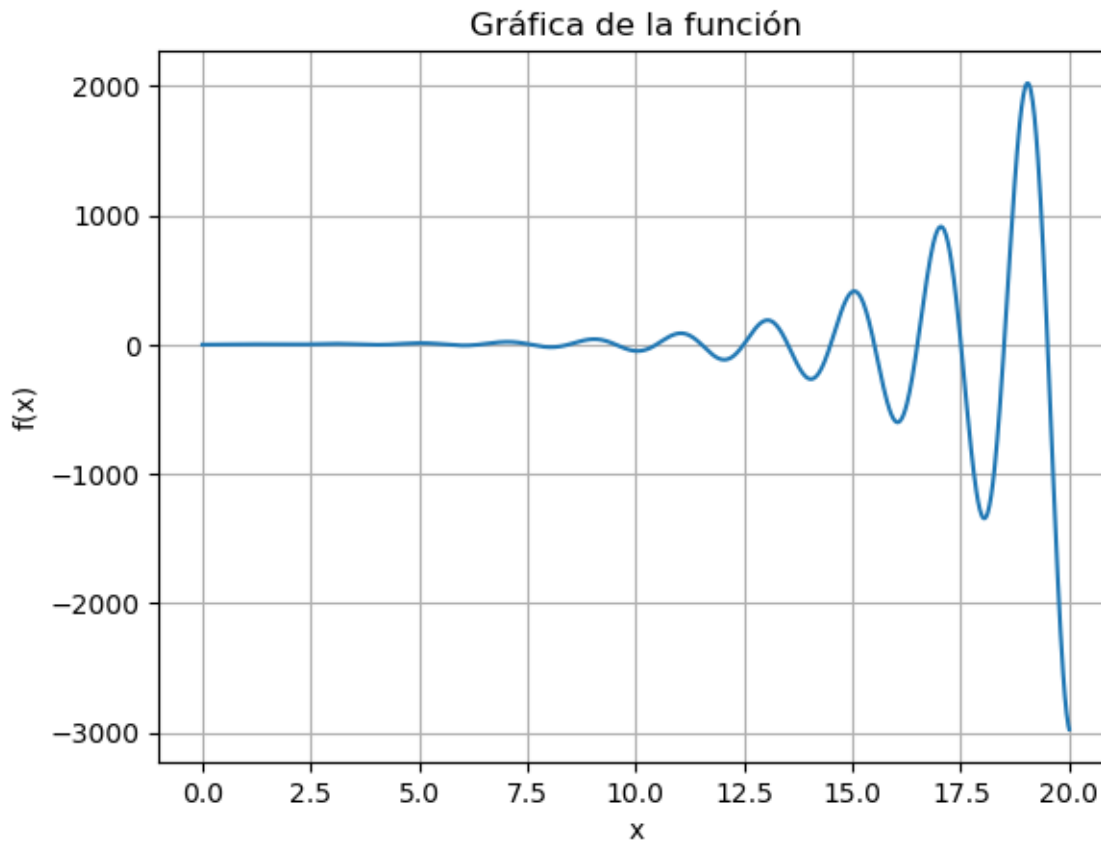
- c. Determine una aproximación inicial razonable para encontrar el n -ésimo cero positivo más pequeño de f .

Para obtener dicha aproximación se puede utilizar la gráfica aproximada de la función con el objetivo de entender el comportamiento de los ceros de la función

```
import numpy as np
import matplotlib.pyplot as plt
import math
def f(x):
    return math.log(x**2 + 1) - math.e**(0.4 * x) * math.cos(math.pi * x)
xv = np.linspace(0, 20, 1000)
y = [f(x) for x in xv]
plt.plot(xv, y)
plt.grid(True)
plt.xlabel("x")
```



```
plt.ylabel("f(x)")
plt.title("Gráfica de la función")
plt.show()
```



A través de la gráfica es posible apreciar como los ceros de la función se encuentran cercanos a múltiplos de 1 debido a que la función tiene una especie de periodicidad dada por el $\cos(\pi * x)$.

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

```
from scipy.optimize import newton
import math
def f(x):
    return math.log(x**2 + 1) - math.e**(0.4 * x) * math.cos(math.pi * x)
x0 = 15
x1 = 15.1
print(f' Se ha encontrado una raíz em :{newton(f, x0, x1=x1, tol=1e-6)}')
```

Se ha encontrado una raíz en :14.49483105099586

7. La función $f(x) = x^{\frac{1}{3}}$ 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 la secante, compare los resultados de los métodos de la secante y de newton

```
import warnings
def f(x):
    return x**(1/3)
def f_prime(x):
    return 1 / (3 * x**(2/3))
x0 = 1
print('El resultado obtenido a través del método de Newton es:')
try:
    with warnings.catch_warnings():
        warnings.simplefilter("ignore", RuntimeWarning)
        resultado, info = newton(f, x0, fprime=f_prime, tol=1e-6, full_output=True, disp=False)
        print(f'Se ha encontrado una raíz en {resultado}')
        print(f'Información adicional: {info}')
except RuntimeError as e:
    print(f"Error: {e}")
print('El resultado obtenido a través del método de la secante es:')
try:
    with warnings.catch_warnings():
        warnings.simplefilter("ignore", RuntimeWarning)
        resultado, info = newton(f, x0=5, x1=0.5, full_output=True, disp=True)
        print(f'Se ha encontrado una raíz en {resultado}')
        print(f'Información adicional: {info}')
except RuntimeError as e:
    print(f"Error: {e}")
```

El resultado obtenido a través del método de Newton es:

Se ha encontrado una raíz en nan

Información adicional: converged: False

flag: convergence error

function_calls: 100

iterations: 50

root: nan

method: newton

El resultado obtenido a través del método de la secante es:

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

En este caso ninguno de los métodos logra converger a la raíz de la función, en el caso del método de Newton esto se debe al comportamiento de la derivada de la función, cuando x se acerca a 0 la derivada tiende al infinito impidiendo que el método converga en la raíz 0, mientras que el método de la secante no logra encontrar la raíz debido a que en esos puntos el comportamiento de la función no es estable evitando que el método logre encontrar el punto 0.