Métodos Numéricos - DEBER 05 - Método de Newton y la Secante

Alicia Pereira

Tabla de Contenidos

0.1 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_1. ¿Se podría usar $p_0 = 0$?

Para la resolución del ejercicio se plantea el uso de la librería scipy, con el objetivo de utilizar el algoritmo de **método de Newton**.

```
from scipy.optimize import newton
import math
```

Como primer punto, se necesita la derivada de la función propuesta:

$$f'(x) = -3x^2 + \sin(x)$$

Con esto, se llama a la función Newton con los parámetros obtenidos:

```
def func(x):
    return -x**3 - math.cos(x)
def fprime(x):
    return -3*x**2 + math.sin(x)

p1 = newton(func = func, x0 = -1 , fprime=fprime)
print("Método de Newton: " + str(p1))
```

```
Método de Newton: -0.8654740331016144
```

El **método de la secante** en cambio, no utiliza la derivada de la función.

```
p1 = newton(func = func, x0 = -1 , fprime=None)
print("Método de la Secante: " + str(p1))
```

```
Método de la Secante: -0.8654740331016144
```

¿Se puede utilizar el cero como p_0?

```
try:
    p1 = newton(func = func, x0 = 0, fprime=fprime)
except RuntimeError as e:
    print(e)
```

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

No se puede utilizar al cero como p_0 dado que la función nunca converge en ese punto.

0.2 EJERCICIO 2

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

0.2.1 a)

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

Se elige el punto inicial arbitrario dentro del intervalo. En este caso, se escoge $p_0 = 1$.

Posteriormente, haciendo uso del método de la secante se obtiene:

```
def func2(x):
    return x**3-2*x**2-5
p1 = newton(func = func2,x0=1,fprime=None,tol=1e-4)
print("La solución precisa es: " + str(p1))
```

```
La solución precisa es: 2.6906474472020077
```

0.2.2 b)

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

Se elige el punto inicial arbitrario dentro del intervalo. En este caso, se escoge $p_0 = -3$.

Posteriormente, haciendo uso del método de la secante se obtiene:

```
def func3(x):
    return x**3-3*x**2-1

p1 = newton(func = func3,x0=-3,fprime=None,tol=1e-4)
print("La solución precisa es: " + str(p1))
```

```
La solución precisa es: 3.1038033822157804
```

0.2.3 c)

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

Se elige el punto inicial arbitrario dentro del intervalo. En este caso, se escoge $p_0 = 0$.

Posteriormente, haciendo uso del método de la secante se obtiene:

```
def func4(x):
    return x-math.cos(x)

p1 = newton(func = func4,x0=0,fprime=None,tol=1e-4)
print("La solución precisa es: " + str(p1))
```

```
La solución precisa es: 0.7390851121452099
```

0.2.4 d)

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

Se elige el punto inicial arbitrario dentro del intervalo. En este caso, se escoge $p_0 = 0$.

Posteriormente, haciendo uso del método de la secante se obtiene:

```
def func5(x):
    return x-0.8-0.2*math.sin(x)

p1 = newton(func = func5,x0=0,fprime=None,tol=1e-4)
print("La solución precisa es: " + str(p1))
```

```
La solución precisa es: 0.9643338895520454
```

0.3 EJERCICIO 3

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

0.3.1 a)

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

```
def func(x):
    return 3 * x - math.e ** x
def fprime(x):
    return 3 -  math.e ** x

p1New = newton(func = func, x0 = 1 , fprime=fprime,tol = 1e-5)
p1Sec = newton(func = func,x0=1,fprime=None,tol=1e-5)
print("Método de Newton: p1 = " + str(p1New))
print("Método de la Secante: p1 = " + str(p1Sec))
```

```
Método de Newton: p1 = 0.619061286735945
Método de la Secante: p1 = 0.6190612855447449
```

0.3.2 b)

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

```
def func(x):
    return 2 * x + 3 * math.cos(x) - math.e ** x
def fprime(x):
    return 2 - 3 * math.sin(x) - math.e ** x

p1New = newton(func = func, x0 = 1.5 , fprime=fprime,tol = 1e-5)
p1Sec = newton(func = func,x0=1.5,fprime=None,tol=1e-5)
print("Método de Newton: p1 = " + str(p1New))
print("Método de la Secante: p1 = " + str(p1Sec))
```

```
Método de Newton: p1 = 1.2397146979752212
Método de la Secante: p1 = 1.2397146979773508
```

0.4 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

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

```
def func(x):
    return 230 * x ** 4 + 18 * x ** 3 + 9 * x ** 2 - 221 * x - 9
x0 = -1
x1 = 0
p1Sec = newton(func = func,x0=-1,fprime=None,tol=1e-5)
print("Método de la Secante: p1 = " + str(p1Sec))
```

```
Método de la Secante: p1 = -0.0406592883157958
```

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

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

x0 = 0.5
x1 = 1

p1New = newton(func = func, x0 = 0.5 , fprime=fprime,tol = 1e-5)
print("Método de Newton: p1 = " + str(p1New))
```

```
Método de Newton: p1 = -0.040659288315758865
```

0.5 EJERCICIO 5

La función $f(x) = \tan \pi x - 6$ tiene cero en: $\frac{1}{\pi} arcotangente 6 \approx 0.447431543$. 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

b. método de Newton

c. método de la secante

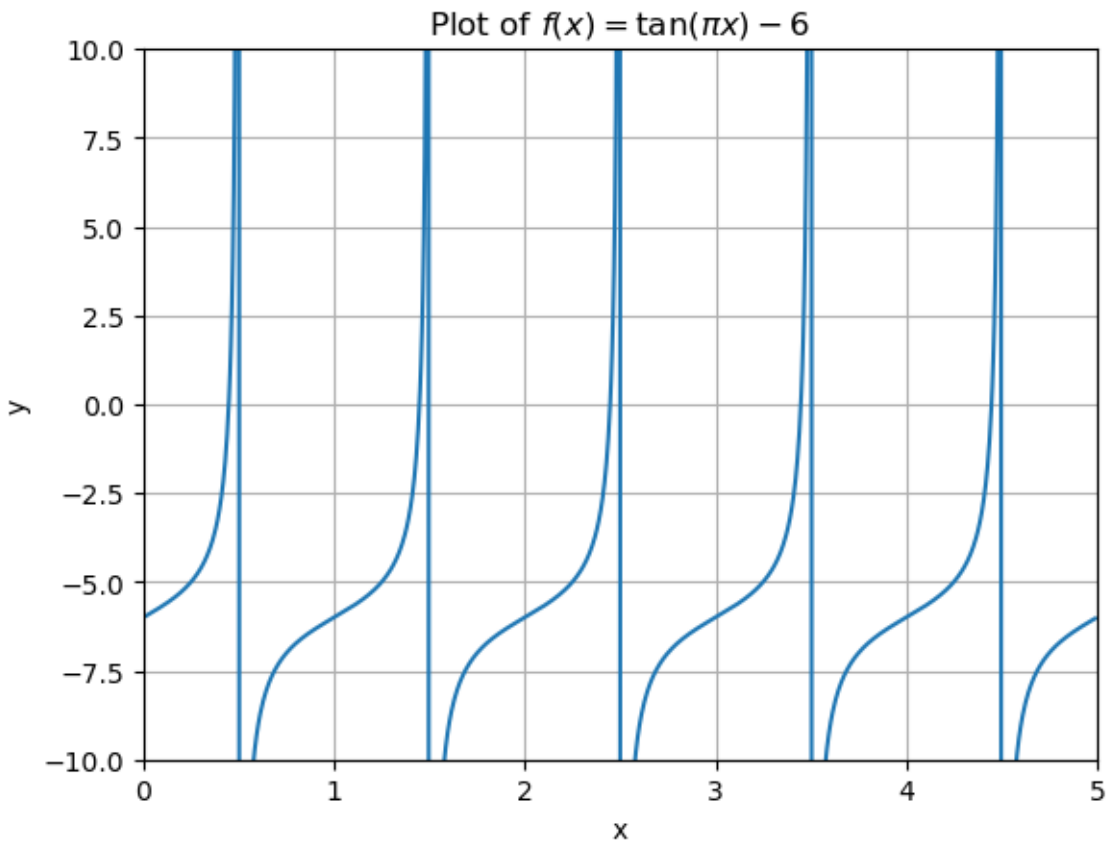
Como primer punto, graficamos y definimos la función correspodiente para aplicar los métodos solicitados:

```python
import numpy as np
import matplotlib.pyplot as plt

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

x = np.linspace(0.01, 4.99, 1000)
y = func(x)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title(r'Plot of $f(x) = \tan(\pi x) - 6$')
ax = plt.gca()
ax.set_ylim([-10, 10])
ax.set_xlim([0, 5])
plt.grid(True)
plt.show()
```



0.5.1 a) Método de Bisección

```
a = 0
b = 0.48

from scipy.optimize import bisect

try:
    result = bisect(func, a, b, maxiter=10, full_output=True)
except RuntimeError as e:
    print('ERROR:', e)
```

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

```
try:
    result2 = bisect(func, a, b, maxiter=100, full_output=True)
except RuntimeError as e:
    print('ERROR:', e)

result2
```

```
(0.44743154328956725,
       converged: True
            flag: converged
  function_calls: 40
      iterations: 38
            root: 0.44743154328956725
          method: bisect)
```

0.5.2 b) Método de Newton

```
def fprime(x):
    return np.pi * (1 / np.cos(np.pi * x))**2
p1New = newton(func = func, x0 = 0.24 , fprime=fprime,maxiter = 10,full_output=True)
p1New
```

```
(5.447431543288746,
       converged: True
            flag: converged
  function_calls: 16
      iterations: 8
            root: 5.447431543288746
          method: newton)
```

```
try:
    p1Sec = newton(func = func,x0=0.24,fprime=None,maxiter = 10,full_output=True)
except RuntimeError as e:
    print('ERROR:', e)
```

```
ERROR: Failed to converge after 10 iterations, value is 3466452.4327915576.
```

0.6 EJERCICIO 6

La función descrita por

$$f(x) = ln(x^2 + 1) - e^{0.4x}cos(\pi x)$$

tiene un número infinito de ceros.

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 f.

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

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

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

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

```
tol = 1e-06
x0 = -0.01

newton(f,x0,fprime=fprime,tol=1e-06,full_output=True)
```

```
(-0.4341424757919693,
       converged: True
            flag: converged
  function_calls: 52
      iterations: 26
            root: -0.4341424757919693
          method: newton)
```

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

```
ceros = []
x0 = 0.5
for _ in range(4):
    root = newton(f,x0,fprime=fprime,tol = 1e-06)
    ceros.append(root)
    x0 += 0.5


for idx, root in enumerate(ceros, start=1):
    print(f'La raiz aproximada es: p{idx} = {root}')
```

```
La raiz aproximada es: p1 = 0.45065732924634816
La raiz aproximada es: p2 = 0.45065585898262517
La raiz aproximada es: p3 = 1.7447378996758833
La raiz aproximada es: p4 = 2.2383199249663526
```

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

```
enesimoCero = []
x0 = 0.75
for _ in range(10):
    root = newton(f,x0,fprime=fprime, maxiter= 100)
    enesimoCero.append(root)
    x0 += 0.2


for idx, root in enumerate(enesimoCero, start=1):
    print(f'La raiz aproximada es: p{idx} = {root}')
```

```
La raiz aproximada es: p1 = 0.4506567582997215
La raiz aproximada es: p2 = 0.4506567593385777
La raiz aproximada es: p3 = 0.45065673558318053
La raiz aproximada es: p4 = 2.2383197956535206
La raiz aproximada es: p5 = 1.7447380509820627
La raiz aproximada es: p6 = 1.74473804949828
La raiz aproximada es: p7 = 8.453480976984618
La raiz aproximada es: p8 = 2.238319794451469
La raiz aproximada es: p9 = 2.238319796225593
La raiz aproximada es: p10 = 2.2383197958116674
```

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

GitHub: git@github.com: dayapt04

GitHub Métodos Númericos - Repositorio

```
vigesimoquinto = []
x0 = 0.75
for _ in range(25):
    root = newton(f,x0,fprime=fprime, maxiter= 100,tol=1e-6)
    vigesimoquinto.append(root)
    x0 += 0.2


for idx, root in enumerate(vigesimoquinto, start=1):
    print(f'La raiz aproximada es: p{idx} = {root}')
```

```
La raiz aproximada es: p1 = 0.45065781194276217
La raiz aproximada es: p2 = 0.4506573521175441
La raiz aproximada es: p3 = 0.4506560983753875
La raiz aproximada es: p4 = 2.2383198958207964
La raiz aproximada es: p5 = 1.7447382064368515
La raiz aproximada es: p6 = 1.7447383015946802
La raiz aproximada es: p7 = 8.45348097738637
La raiz aproximada es: p8 = 2.2383196868003163
La raiz aproximada es: p9 = 2.2383199952962194
La raiz aproximada es: p10 = 2.2383199233203115
La raiz aproximada es: p11 = 2.2383198656962455
La raiz aproximada es: p12 = 1.7447381861857476
La raiz aproximada es: p13 = 5.619935305378205
La raiz aproximada es: p14 = 3.7090411622803936
La raiz aproximada es: p15 = 3.7090412127803596
La raiz aproximada es: p16 = 3.70904125603659
La raiz aproximada es: p17 = 4.322648977690657
La raiz aproximada es: p18 = 4.322648975706081
La raiz aproximada es: p19 = 4.322649013916713
La raiz aproximada es: p20 = 4.322648964397358
La raiz aproximada es: p21 = 4.322648951374646
La raiz aproximada es: p22 = 3.7090411546273176
La raiz aproximada es: p23 = 6.406933621149631
La raiz aproximada es: p24 = 5.619935320337745
La raiz aproximada es: p25 = 5.61993533583606
```

0.7 EJERCICIO 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 secante, compare los resultados de los métodos de la secante y de Newton.

```
import numpy as np

# Función a evaluar
def func(x):
    return np.cbrt(x)  # Raíz cúbica segura para números reales

# Derivada de la función
def fprime(x):
    if x == 0:  # Para evitar división por cero
        return float('inf')
    return 1 / (3 * np.cbrt(x**2))  # Derivada de x^(1/3)

x = 1
p0=5
p1=0.5

try:
    p1New = newton(func = func, x0 = x , fprime=fprime,maxiter=1000)
    p1Sec = newton(func = func,x0=p0,x1=p1,fprime=None,tol=1e-5,maxiter=1000)
except RuntimeError as e:
    print('ERROR:', e)


print("Método de Newton: p1 = " + str(p1New))
print("Método de la Secante: p1 = " + str(p1Sec))
```

```
ERROR: Derivative was zero. Failed to converge after 514 iterations, value is -2.68156158598
Método de Newton: p1 = (5.447431543288746,       converged: True
           flag: converged
 function_calls: 16
     iterations: 8
           root: 5.447431543288746
         method: newton)
Método de la Secante: p1 = -0.0406592883157958
```

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

# Función a evaluar
def func(x):
    return np.cbrt(x)  # Raíz cúbica segura para números reales

# Derivada de la función
def fprime(x):
    return 1 / (3 * (np.cbrt(x)**2))  # Forma segura de calcular la derivada

# Valores iniciales
x = 5
p0 = 5
p1 = 0.5

# Método de Newton
try:
    p1New = newton(func=func, x0=x, fprime=fprime, maxiter=10000, tol=1e-6)
    print("Método de Newton: p1 = " + str(p1New))
except RuntimeError as e:
    print("Error en el método de Newton:", e)

# Método de la Secante (sin derivada)
try:
    p1Sec = newton(func=func, x0=p0, x1=p1, fprime=None, tol=1e-6, maxiter=1000)
    print("Método de la Secante: p1 = " + str(p1Sec))
except RuntimeError as e:
    print("Error en el método de la Secante:", e)
```

```
Error en el método de Newton: Derivative was zero. Failed to converge after 1023 iterations,
Error en el método de la Secante: Failed to converge after 1000 iterations, value is -0.1512
```

GitHub: git@github.com: dayapt04

GitHub Métodos Númericos - Repositorio