tarea05-ejercicios-unidad-02b

May 19, 2025

ESCUELA POLITÉCNICA NACIONAL FACULTAD DE INGENIERÍA DE SISTEMAS MÉTODOS NUMÉRICOS

Método de Newton y de la Secante

Richard Tipantiza 2025-05-15

1 CONJUNTO DE EJERCICIOS

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

¿Se podría usar $p_0 = 0$?

[2]:
```python
import math

# Definición de la función y su derivada
def f(x):
    return -x**3 - math.cos(x)

def df(x):
    return -3*x**2 + math.sin(x)

# Método de Newton
def newton(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
        p0 = p
    return p0

# Método de la Secante
def secante(p0, p1, tol=1e-6, max_iter=100):
    for i in range(max_iter):
        p = p1 - f(p1) * (p1 - p0) / (f(p1) - f(p0))
        if abs(p - p1) < tol:
            return p
        p0, p1 = p1, p
```

```python
    return p1

# Resultados
p0 = -1
p1_newton = newton(p0)  # p1 usando Newton (1 iteración)
p2_newton = newton(p1_newton)  # p2 usando Newton (2 iteraciones)
p2_secante = secante(p0, p1_newton)  # p2 usando Secante

print(f"Método de Newton: p1 = {p1_newton:.6f}, p2 = {p2_newton:.6f}")
print(f"Método de la Secante: p2 = {p2_secante:.6f}")

# Verificación de p0 = 0
try:
    newton(0)
except ZeroDivisionError:
    print("\n¿p0 = 0 válido en Newton? No (división por cero en f'(0)).")
print("¿p0 = 0 válido en Secante? Sí, si se elige p1   0 y f(p0)   f(p1).")
```

```
Método de Newton: p1 = -0.865474, p2 = -0.865474
Método de la Secante: p2 = -0.865474

¿p0 = 0 válido en Newton? No (división por cero en f'(0)).
¿p0 = 0 válido en Secante? Sí, si se elige p1   0 y f(p0)   f(p1).
```

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

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

[7]:
```python
import math

def f_a(x):
    return x**3 - 2*x**2 - 5

def biseccion(f, a, b, tol=1e-4, max_iter=100):
    if f(a) * f(b) >= 0:
        raise ValueError("No hay cambio de signo en el intervalo.")
    for _ in range(max_iter):
        c = (a + b) / 2
        if abs(f(c)) < tol:
            return c
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c

sol_a = biseccion(f_a, 1, 4)
print(f"Literal a) Solución: {sol_a:.6f}")
```

```
Literal a) Solución: 2.690647
```

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

[8]:
```python
def f_b(x):
    return x**3 + 3*x**2 - 1

sol_b = biseccion(f_b, -3, -2)
print(f"Literal b) Solución: {sol_b:.6f}")
```

```
Literal b) Solución: -2.879395
```

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

[9]:
```python
def f_c(x):
    return x - math.cos(x)

def df_c(x):
    return 1 + math.sin(x)

def newton(f, df, p0, tol=1e-4, max_iter=100):
    for _ in range(max_iter):
        p = p0 - f(p0) / df(p0)
        if abs(p - p0) < tol:
            return p
        p0 = p
    return p0

sol_c = newton(f_c, df_c, 1)  # p0 = 1 es un punto inicial razonable
print(f"Literal c) Solución: {sol_c:.6f}")
```

```
Literal c) Solución: 0.739085
```

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

[10]:
```python
def f_d(x):
    return x - 0.8 - 0.2 * math.sin(x)

def secante(f, p0, p1, tol=1e-4, max_iter=100):
    for _ in range(max_iter):
        p = p1 - f(p1) * (p1 - p0) / (f(p1) - f(p0))
        if abs(p - p1) < tol:
            return p
        p0, p1 = p1, p
    return p1

sol_d = secante(f_d, 0, math.pi/2)
print(f"Literal d) Solución: {sol_d:.6f}")
```

```
Literal d) Solución: 0.964334
```

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

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

[11]:
```
import math

def f_a(x):
    return 3*x - math.exp(x)

def biseccion(f, a, b, tol=1e-5, max_iter=100):
    if f(a) * f(b) >= 0:
        raise ValueError("No hay cambio de signo en el intervalo.")
    for _ in range(max_iter):
        c = (a + b) / 2
        if abs(f(c)) < tol:
            return c
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c

sol_a_biseccion = biseccion(f_a, 1, 2)
print(f"Literal a) Bisección: {sol_a_biseccion:.7f}")
```

```
Literal a) Bisección: 1.5121307
```

[13]:
```
def df_a(x):
    return 3 - math.exp(x)  # Derivada de f_a(x)

def newton(f, df, p0, tol=1e-5, max_iter=100):
    for _ in range(max_iter):
        p = p0 - f(p0) / df(p0)
        if abs(p - p0) < tol:
            return p
        p0 = p
    return p0

sol_a_newton = newton(f_a, df_a, 1.5)  # p0 = 1.5 es un punto inicial adecuado
print(f"Literal a) Newton: {sol_a_newton:.7f}")
```

```
Literal a) Newton: 1.5121346
```

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

[14]:
```
def f_b(x):
    return 2*x + 3*math.cos(x) - math.exp(x)
```

```python
sol_b_biseccion = biseccion(f_b, 1, 2)
print(f"Literal b) Bisección: {sol_b_biseccion:.7f}")
```

```
Literal b) Bisección: 1.2397156
```

[15]:
```python
def df_b(x):
    return 2 - 3*math.sin(x) - math.exp(x)  # Derivada de f_b(x)

sol_b_newton = newton(f_b, df_b, 1.5)  # p0 = 1.5
print(f"Literal b) Newton: {sol_b_newton:.7f}")
```

```
Literal b) Newton: 1.2397147
```

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} usando:

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

[17]:
```python
import math

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

def secante(f, p0, p1, tol=1e-6, max_iter=100):
    for _ in range(max_iter):
        p = p1 - f(p1) * (p1 - p0) / (f(p1) - f(p0))
        if abs(p - p1) < tol:
            return p
        p0, p1 = p1, p
    return p1

# Cero en [-1, 0]
cero_secante_neg = secante(f, -1, 0)
print(f"Cero en [-1, 0] (Secante): {cero_secante_neg:.7f}")

# Cero en [0, 1]
cero_secante_pos = secante(f, 0, 1)
print(f"Cero en [0, 1] (Secante): {cero_secante_pos:.7f}")
```

```
Cero en [-1, 0] (Secante): -0.0406593
Cero en [0, 1] (Secante): -0.0406593
```

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

[18]:
```python
def df(x):
    return 920*x**3 + 54*x**2 + 18*x - 221

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

# Cero en [-1, 0] (p0 = -0.5)
cero_newton_neg = newton(f, df, -0.5)
print(f"Cero en [-1, 0] (Newton): {cero_newton_neg:.7f}")

# Cero en [0, 1] (p0 = 0.5)
cero_newton_pos = newton(f, df, 0.5)
print(f"Cero en [0, 1] (Newton): {cero_newton_pos:.7f}")
```

```
Cero en [-1, 0] (Newton): -0.0406593
Cero en [0, 1] (Newton): -0.0406593
```

5. La función

$$f(x) = \tan(\pi x) - 6$$

tiene un cero en

$$\left(\frac{1}{\pi}\right)\arctan(6) \approx 0.447431543$$

Sea $p_0 = 0$ y $p_1 = 0.48$. Use **10 iteraciones** en cada uno de los siguientes métodos para aproximar esta raíz:

a. Método de bisección

[26]:
```python
import math

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

[27]:
```python
def bisection(f, a, b, n):
    for i in range(n):
        p = (a + b) / 2
        print(f"Iteración {i+1}: p = {p}")
        if f(a) * f(p) < 0:
            b = p
        else:
```

```
            a = p

# Ejecutar 10 iteraciones
bisection(f, 0, 0.48, 10)
```

```
Iteración 1: p = 0.24
Iteración 2: p = 0.36
Iteración 3: p = 0.42
Iteración 4: p = 0.44999999999999996
Iteración 5: p = 0.43499999999999994
Iteración 6: p = 0.44249999999999995
Iteración 7: p = 0.4462499999999999
Iteración 8: p = 0.44812499999999994
Iteración 9: p = 0.44718749999999996
Iteración 10: p = 0.44765625
```

b. Método de Newton

[28]:
```
def df(x):
    return math.pi * (1 / math.cos(math.pi * x))**2

def newton(f, df, p0, n):
    for i in range(n):
        p1 = p0 - f(p0) / df(p0)
        print(f"Iteración {i+1}: p = {p1}")
        p0 = p1

# Ejecutar 10 iteraciones
newton(f, df, 0.48, 10)
```

```
Iteración 1: p = 0.4675825019258912
Iteración 2: p = 0.4551291915177739
Iteración 3: p = 0.4485512339384831
Iteración 4: p = 0.4474551842507058
Iteración 5: p = 0.4474315538237576
Iteración 6: p = 0.4474315432887487
Iteración 7: p = 0.4474315432887466
Iteración 8: p = 0.44743154328874657
Iteración 9: p = 0.4474315432887466
Iteración 10: p = 0.44743154328874657
```

c. Método de la secante

[29]:
```
def secante(f, p0, p1, n):
    for i in range(n):
        p = p1 - f(p1) * (p1 - p0) / (f(p1) - f(p0))
        print(f"Iteración {i+1}: p = {p}")
        p0, p1 = p1, p
```

```python
# Ejecutar 10 iteraciones
secante(f, 0, 0.48, 10)
```

```
Iteración 1: p = 0.18119424169051174
Iteración 2: p = 0.2861871658222898
Iteración 3: p = 1.0919861065027499
Iteración 4: p = -3.6922966654011073
Iteración 5: p = -22.60064985474053
Iteración 6: p = -57.22283247260205
Iteración 7: p = 3.5387581457345476
Iteración 8: p = -113.94440504807905
Iteración 9: p = -195.89499482451663
Iteración 10: p = -2989.9400375314453
```

¿Cuál método es más eficaz y por qué?

- Creo que Newton probablemente convergerá más rápido si se inicia cerca de la raíz por lo que sería la más eficaz aunq bajo una condición.

6. La función descrita por

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

tiene un número infinito de ceros.

[30]:
```python
import math

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

[37]:
```python
def bisection(f, a, b, tol=1e-6, max_iter=100):
    fa, fb = f(a), f(b)
    if fa * fb > 0:
        raise ValueError("El intervalo no contiene un cambio de signo.")

    for i in range(max_iter):
        p = (a + b) / 2
        fp = f(p)
        print(f"Iteración {i+1}: p = {p}")
        if abs(fp) < tol or (b - a) / 2 < tol:
            return p
        if fa * fp < 0:
            b, fb = p, fp
        else:
            a, fa = p, fp
    return (a + b) / 2
```

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

[44]:
```python
def buscar_intervalo_signo(f, a, b, paso=0.1):
    x = a
    while x + paso <= b:
        try:
            if f(x) * f(x + paso) < 0:
                return (x, x + paso)
        except:
            pass  # Evita errores por dominios no válidos
        x += paso
    return None

# Buscar intervalo negativo donde haya cambio de signo
intervalo_neg = buscar_intervalo_signo(f, -5, 0)

if intervalo_neg:
    print(f"Intervalo con cambio de signo encontrado: {intervalo_neg}")
    raiz_neg = bisection(f, intervalo_neg[0], intervalo_neg[1])
    print(f"\nCero negativo encontrado: {raiz_neg}")
else:
    print("No se encontró intervalo negativo con cambio de signo.")
```

```
Intervalo con cambio de signo encontrado: (-0.500000000000001,
-0.400000000000001)
Iteración 1: p = -0.450000000000001
Iteración 2: p = -0.42500000000000104
Iteración 3: p = -0.437500000000001
Iteración 4: p = -0.431250000000001
Iteración 5: p = -0.434375000000001
Iteración 6: p = -0.43281250000000104
Iteración 7: p = -0.433593750000001
Iteración 8: p = -0.433984375000001
Iteración 9: p = -0.434179687500001
Iteración 10: p = -0.434082031250001
Iteración 11: p = -0.434130859375001
Iteración 12: p = -0.43415527343750104
Iteración 13: p = -0.434143066406251

Cero negativo encontrado: -0.434143066406251
```

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

[39]:
```python
ceros = []
x = 0
while len(ceros) < 4:
    if f(x) * f(x + 0.5) < 0:
        cero = bisection(f, x, x + 0.5)
        ceros.append(cero)
        print(f"Cero #{len(ceros)} en: {cero}")
```

```python
    x += 0.1

print("\nCeros positivos encontrados:")
for i, c in enumerate(ceros, 1):
    print(f"Cero #{i}: {c}")
```

```
Iteración 1: p = 0.25
Iteración 2: p = 0.375
Iteración 3: p = 0.4375
Iteración 4: p = 0.46875
Iteración 5: p = 0.453125
Iteración 6: p = 0.4453125
Iteración 7: p = 0.44921875
Iteración 8: p = 0.451171875
Iteración 9: p = 0.4501953125
Iteración 10: p = 0.45068359375
Iteración 11: p = 0.450439453125
Iteración 12: p = 0.4505615234375
Iteración 13: p = 0.45062255859375
Iteración 14: p = 0.450653076171875
Iteración 15: p = 0.4506683349609375
Iteración 16: p = 0.45066070556640625
Iteración 17: p = 0.4506568908691406
Cero #1 en: 0.4506568908691406
Iteración 1: p = 0.35
Iteración 2: p = 0.475
Iteración 3: p = 0.4125
Iteración 4: p = 0.44375
Iteración 5: p = 0.459375
Iteración 6: p = 0.4515625
Iteración 7: p = 0.44765625
Iteración 8: p = 0.449609375
Iteración 9: p = 0.4505859375
Iteración 10: p = 0.45107421875
Iteración 11: p = 0.450830078125
Iteración 12: p = 0.4507080078125
Iteración 13: p = 0.45064697265625
Iteración 14: p = 0.450677490234375
Iteración 15: p = 0.4506622314453125
Iteración 16: p = 0.45065460205078123
Iteración 17: p = 0.45065841674804685
Iteración 18: p = 0.45065650939941404
Iteración 19: p = 0.45065746307373045
Cero #2 en: 0.45065746307373045
Iteración 1: p = 0.44999999999999996
Iteración 2: p = 0.575
Iteración 3: p = 0.5125
```

```
Iteración 4: p = 0.48124999999999996
Iteración 5: p = 0.46562499999999996
Iteración 6: p = 0.45781249999999996
Iteración 7: p = 0.45390624999999996
Iteración 8: p = 0.45195312499999996
Iteración 9: p = 0.45097656249999996
Iteración 10: p = 0.45048828124999996
Iteración 11: p = 0.45073242187499996
Iteración 12: p = 0.45061035156249996
Iteración 13: p = 0.45067138671874996
Iteración 14: p = 0.45064086914062496
Iteración 15: p = 0.45065612792968746
Iteración 16: p = 0.4506637573242187
Iteración 17: p = 0.4506599426269531
Iteración 18: p = 0.45065803527832027
Iteración 19: p = 0.45065708160400386
Cero #3 en: 0.45065708160400386
Iteración 1: p = 0.55
Iteración 2: p = 0.42500000000000004
Iteración 3: p = 0.48750000000000004
Iteración 4: p = 0.45625000000000004
Iteración 5: p = 0.44062500000000004
Iteración 6: p = 0.44843750000000004
Iteración 7: p = 0.45234375000000004
Iteración 8: p = 0.45039062500000004
Iteración 9: p = 0.45136718750000004
Iteración 10: p = 0.45087890625000004
Iteración 11: p = 0.45063476562500004
Iteración 12: p = 0.45075683593750004
Iteración 13: p = 0.45069580078125004
Iteración 14: p = 0.45066528320312504
Iteración 15: p = 0.45065002441406254
Iteración 16: p = 0.4506576538085938
Iteración 17: p = 0.45065383911132817
Iteración 18: p = 0.450655746459961
Iteración 19: p = 0.4506567001342774
Cero #4 en: 0.4506567001342774

Ceros positivos encontrados:
Cero #1: 0.4506568908691406
Cero #2: 0.45065746307373045
Cero #3: 0.45065708160400386
Cero #4: 0.4506567001342774
```

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

[40]:
```
def estimar_cero(n):
    return (2 * n + 1) / 2 - 0.1  # Ajuste hacia la izquierda

# Ejemplo:
for n in range(1, 6):
    print(f"Aproximación inicial para el cero #{n}: {estimar_cero(n)}")
```

```
Aproximación inicial para el cero #1: 1.4
Aproximación inicial para el cero #2: 2.4
Aproximación inicial para el cero #3: 3.4
Aproximación inicial para el cero #4: 4.4
Aproximación inicial para el cero #5: 5.4
```

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

[41]:
```
n = 25
x0 = estimar_cero(n)
x1 = x0 + 0.2  # Tomamos un pequeño intervalo a la derecha
if f(x0) * f(x1) < 0:
    raiz_25 = bisection(f, x0, x1)
    print(f"\nCero positivo número 25: {raiz_25}")
else:
    print("No se encontró cambio de signo en el intervalo. Intenta ajustar x0.")
```

```
Iteración 1: p = 25.5
Iteración 2: p = 25.549999999999997
Iteración 3: p = 25.525
Iteración 4: p = 25.5125
Iteración 5: p = 25.50625
Iteración 6: p = 25.503125
Iteración 7: p = 25.5015625
Iteración 8: p = 25.50078125
Iteración 9: p = 25.500390625
Iteración 10: p = 25.5001953125
Iteración 11: p = 25.50009765625
Iteración 12: p = 25.500048828125
Iteración 13: p = 25.500073242187497
Iteración 14: p = 25.500085449218748
Iteración 15: p = 25.50007934570312
Iteración 16: p = 25.50007629394531
Iteración 17: p = 25.500077819824213
Iteración 18: p = 25.500077056884763

Cero positivo número 25: 25.500077056884763
```

7. La función

$$f(x) = x^{1/3}$$

tiene una raíz en $x = 0$.

Usando un punto de inicio $x = 1$ para el método de **Newton**, y $p_0 = 5$, $p_1 = 0.5$ para el método de la **secante**, compare los resultados de ambos métodos.

¿Cuál método se comporta mejor en este caso? ¿Cuál converge más rápido o más consistentemente hacia la raíz?

[45]:
```
def f(x):
    return x**(1/3)

def df(x):
    # Derivada de x^(1/3) = (1/3) * x^(-2/3)
    if x == 0:
        return float('inf')  # evitar división por cero
    return (1/3) * x**(-2/3)
```

[46]:
```
def newton(f, df, p0, n=10):
    print("Método de Newton:")
    for i in range(n):
        p1 = p0 - f(p0)/df(p0)
        print(f"Iteración {i+1}: p = {p1}")
        p0 = p1
```

[47]:
```
# Ejecutar Newton
newton(f, df, 1, 10)
```

```
Método de Newton:
Iteración 1: p = -2.0
Iteración 2: p = (4-2.2893847434456487e-15j)
Iteración 3: p = (-7.999999999999998+4.5787694868912965e-15j)
Iteración 4: p = (15.999999999999991-2.195811558520189e-14j)
Iteración 5: p = (-31.99999999999998+4.391623117040377e-14j)
Iteración 6: p = (63.999999999999936-1.1889466323432573e-13j)
Iteración 7: p = (-127.99999999999983+2.377893264686514e-13j)
Iteración 8: p = (255.9999999999996-6.413344150144774e-13j)
Iteración 9: p = (-511.99999999999903+1.282668830028954e-12j)
Iteración 10: p = (1023.9999999999977-2.1878942263561007e-12j)
```

[48]:
```
def secante(f, p0, p1, n=10):
    print("\nMétodo de la Secante:")
    for i in range(n):
        denom = f(p1) - f(p0)
        if denom == 0:
            print("División por cero en la iteración", i+1)
            break
```

```python
        p = p1 - f(p1) * (p1 - p0) / denom
        print(f"Iteración {i+1}: p = {p}")
        p0, p1 = p1, p
```

[49]:
```python
# Ejecutar Secante
secante(f, 5, 0.5, 10)
```

```
Método de la Secante:
Iteración 1: p = -3.3980117618223327
Iteración 2: p = (0.42342548212957754-2.373791283892393j)
Iteración 3: p = (-2.506434572816832-3.343001585580911j)
Iteración 4: p = (1.537946395028479+6.125822678839629j)
Iteración 5: p = (-7.895622939891771+3.325889225481498j)
Iteración 6: p = (6.554151215536118-12.735088076137382j)
Iteración 7: p = (-12.95378376757566-12.317401136903326j)
Iteración 8: p = (5.468753662426424+30.80985385003664j)
Iteración 9: p = (-35.05958741642519+15.886948700273436j)
Iteración 10: p = (32.050165663512885-59.72033599449141j)
```

- Newton no converge rápidamente en este caso, porque f'(x) = (1/3)x^{-2/3} se vuelve muy grande cuando x se acerca a 0. Puede incluso divergir.
- La secante puede ser más estable aquí, ya que no requiere derivada y evita los problemas con x^(-2/3).
- En este caso, la **secante converge más consistentemente** hacia la raíz x = 0.