Tarea N°4 Método de la Bisección

Tamara Benavidez

Tabla de Contenidos

ESCUELA POLITÉCNICA NACIONAL

TAREA N°4 Ejercicios Unidad 02 A

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

Método de la Bisección: Ejercicios

1. Use el método de bisección para encontrar soluciones precisas dentro de $ 10^{-2}$ para $ x^{3} -7x^{2} + 14x-6=0$ en cada intervalo.

    a. [0, 1]

    b. [1, 3.2]

    c. [3.2, 4]

```python
def biseccion(f, a, b, tol=1e-2, max_iter=100):

    if f(a)*f(b) > 0:
        return None, 0  # No hay cambio de signo

    for i in range(max_iter):
        c = (a + b)/2
        if f(c) == 0 or (b - a)/2 < tol:
            return c, i+1
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a+b)/2, max_iter

# Función
f = lambda x: x**3 - 7*x**2 + 14*x - 6

# Intervalos
intervalos = {
    "a": (0, 1),
    "b": (1, 3.2),
    "c": (3.2, 4)
}

# Se aplica el método de Bisección a cada intervalo
for key, (a, b) in intervalos.items():
    raiz, iteraciones = biseccion(f, a, b)
    print(f"Raíz {key}: {raiz:.4f} encontrada en {iteraciones} iteraciones")
```

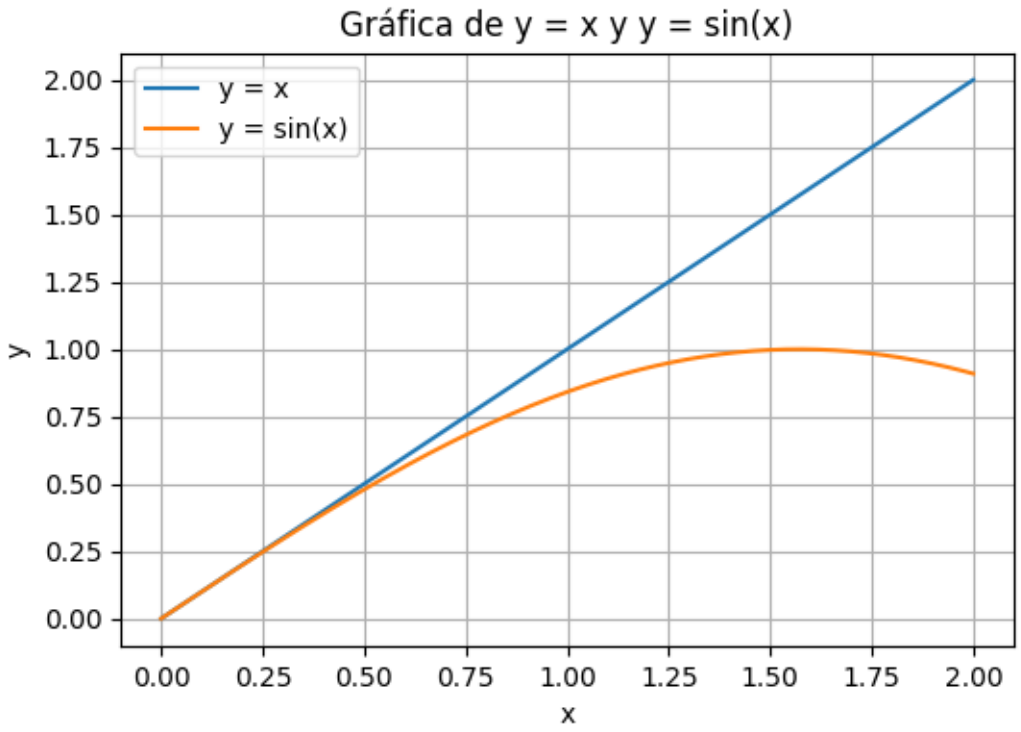
```
Raíz a: 0.5859 encontrada en 7 iteraciones
Raíz b: 3.0023 encontrada en 8 iteraciones
Raíz c: 3.4188 encontrada en 7 iteraciones
```

2. a. Dibuje las gráficas para = y = sin x.

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

x = np.linspace(0, 2, 400)
y1 = x
y2 = np.sin(x)

plt.figure(figsize=(6,4))
plt.plot(x, y1, label='y = x')
plt.plot(x, y2, label='y = sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gráfica de y = x y y = sin(x)')
plt.grid(True)
plt.legend()
plt.show()
```



b. Use el método de bisección para encontrar soluciones precisas dentro de $10^{-5} $ para el

```python
def biseccion(f, a, b, tol=1e-5, max_iter=100):
    if f(a)*f(b) > 0:
        return None, 0  # No hay cambio de signo
    for i in range(max_iter):
        c = (a + b)/2
        if f(c) == 0 or (b - a)/2 < tol:
            return c, i+1
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a+b)/2, max_iter

# Función f(x) = x - 2*sin(x)
f = lambda x: x - 2*np.sin(x)

# Primer cero positivo
raiz, iteraciones = biseccion(f, 0, 2)
print(f"Primer cero positivo: x = {raiz:.5f} encontrado en {iteraciones} iteraciones")
```

```
Primer cero positivo: x = 1.89550 encontrado en 18 iteraciones
```

3. a. Dibuje las gráficas para $ y = x $ y $ y = \tan x $

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

x = np.linspace(0, 1.4, 400)  # evitamos pi/2 porque tan(x) → ∞
y1 = x
y2 = np.tan(x)

plt.figure(figsize=(6,4))
plt.plot(x, y1, label='y = x')
plt.plot(x, y2, label='y = tan(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gráfica de y = x y y = tan(x)')
plt.grid(True)
plt.legend()
plt.ylim(-5, 5)  # para que la curva de tan(x) se vea mejor
plt.show()
```

b. Use el método de bisección para encontrar una aproximación dentro de 10^{-5} para el pr

```
def biseccion(f, a, b, tol=1e-5, max_iter=100):
    if f(a)*f(b) > 0:
        return None, 0  # No hay cambio de signo
    for i in range(max_iter):
        c = (a + b)/2
        if f(c) == 0 or (b - a)/2 < tol:
            return c, i+1
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a+b)/2, max_iter

# Función f(x) = x - tan(x)
f = lambda x: x - np.tan(x)

# Primer cero positivo (buscado antes de pi/2)
```

```
raiz, iteraciones = biseccion(f, 0, 1.4)
print(f"Primer cero positivo: x = {raiz:.5f} encontrado en {iteraciones} iteraciones")
```

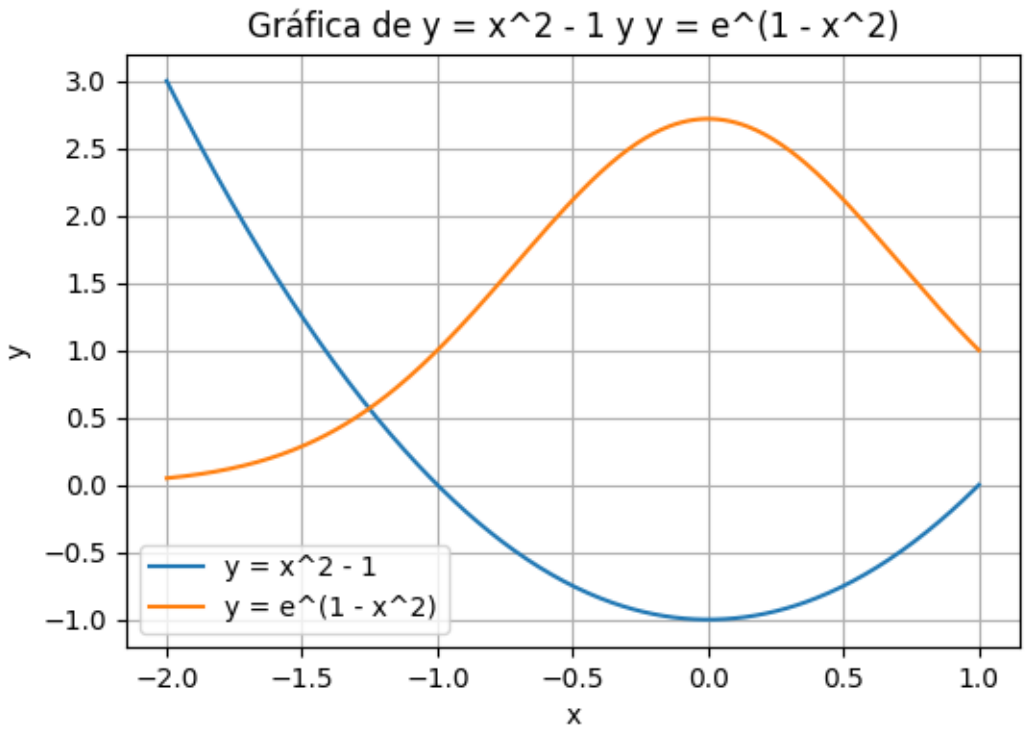
```
Primer cero positivo: x = 1.39999 encontrado en 18 iteraciones
```

4. a. Dibuje las gráficas para $ y = x^{2} -1$ y $ y = e^{1-x}{2}}$

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

x = np.linspace(-2, 1, 400)
y1 = x**2 - 1
y2 = np.exp(1 - x**2)

plt.figure(figsize=(6,4))
plt.plot(x, y1, label='y = x^2 - 1')
plt.plot(x, y2, label='y = e^(1 - x^2)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gráfica de y = x^2 - 1 y y = e^(1 - x^2)')
plt.grid(True)
plt.legend()
plt.show()
```



b. Use el método de bisección para encontrar una aproximación dentro de $10 ^{-3} $ para un

```python
def biseccion(f, a, b, tol=1e-3, max_iter=100):
    if f(a)*f(b) > 0:
        return None, 0  # No hay cambio de signo
    for i in range(max_iter):
        c = (a + b)/2
        if f(c) == 0 or (b - a)/2 < tol:
            return c, i+1
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a+b)/2, max_iter

# Función f(x) = x^2 - 1 - exp(1 - x^2)
f = lambda x: x**2 - 1 - np.exp(1 - x**2)

# Raíz en [-2, 0]
```

```
raiz, iteraciones = biseccion(f, -2, 0)
print(f"Raíz en [-2,0]: x = {raiz:.3f} encontrada en {iteraciones} iteraciones")
```

```
Raíz en [-2,0]: x = -1.251 encontrada en 11 iteraciones
```

5. Sea $(\ ) = (x+3)(x+1)^2x(x-1)^3(x-3)$ ¿En qué cero de converge el método de bisección cuando se aplica en los siguientes intervalos?

   a. [-1.5, 2.5]

   b. [-0.5, 2.4]

   c. [-0.5, 3]

```
def biseccion(f, a, b, tol=1e-5, max_iter=100):
    if f(a)*f(b) > 0:
        return None, 0  # No hay cambio de signo
    for i in range(max_iter):
        c = (a + b)/2
        if f(c) == 0 or (b - a)/2 < tol:
            return c, i+1
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a+b)/2, max_iter

# Función
f = lambda x: (x + 3)*(x + 1)**2 * x * (x - 1)**3 * (x - 3)

# Intervalos
intervalos = {
    "a": (-1.5, 2.5),
    "b": (-0.5, 2.4),
    "c": (-0.5, 3),
    "d": (-3, -0.5)
}

# Se aplica el método de Bisección a cada intervalo
for key, (a, b) in intervalos.items():
    raiz, iteraciones = biseccion(f, a, b)
    if raiz is not None:
        print(f"Intervalo {key} [{a},{b}] → raíz aproximada: {raiz:.5f} en {iteraciones} ite
```

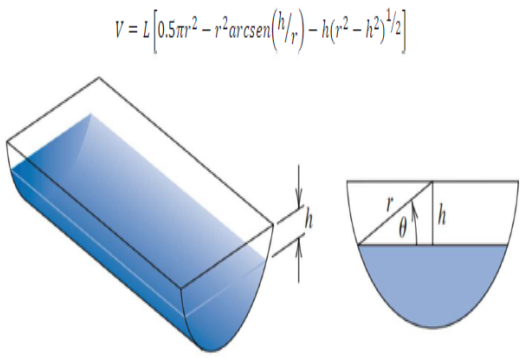
```
    else:
        print(f"Intervalo {key} [{a},{b}] → No hay cambio de signo, no se encuentra raíz por
```

```
Intervalo a [-1.5,2.5] → No hay cambio de signo, no se encuentra raíz por bisección
Intervalo b [-0.5,2.4] → No hay cambio de signo, no se encuentra raíz por bisección
Intervalo c [-0.5,3] → raíz aproximada: 2.99999 en 19 iteraciones
Intervalo d [-3,-0.5] → raíz aproximada: -0.50001 en 18 iteraciones
```

EJERCICIOS APLICADOS

1. Un abrevadero de longitud tiene una sección transversal en forma de semicírculo con radio . (Consulte la figura adjunta.) Cuando se llena con agua hasta una distancia a partir de la parte superior, el volumen de agua es

$$V = L\left[\tfrac{1}{2}\pi r^2 - r^2 \arcsin\left(\tfrac{h}{r}\right) - h\sqrt{r^2 - h^2}\right]$$



Suponga que = 10 , = 1 y = 12.4 $cm^{3} $. Encuentre la profundidad del agua en el abrevadero dentro de 0.01 .

```
import numpy as np

# Datos
L = 10  # cm
r = 1   # cm
V_obj = 12.4  # cm^3

# Función f(h) = V(h) - V_obj
def f(h):
```

```
    return L*(0.5*np.pi*r**2 - r**2*np.arcsin(h/r) - h*np.sqrt(r**2 - h**2)) - V_obj

# Método de Bisección
def biseccion(f, a, b, tol=0.01, max_iter=100):
    if f(a)*f(b) > 0:
        return None, 0  # No hay cambio de signo
    for i in range(max_iter):
        c = (a + b)/2
        if f(c) == 0 or (b - a)/2 < tol:
            return c, i+1
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a+b)/2, max_iter

# Intervalo posible [0, r]
raiz, iteraciones = biseccion(f, 0, r)
print(f"Profundidad del agua: h = {raiz:.2f} cm encontrada en {iteraciones} iteraciones")
```

```
Profundidad del agua: h = 0.16 cm encontrada en 7 iteraciones
```

2. Un objeto que cae verticalmente a través del aire está sujeto a una resistencia viscosa, así como a la fuerza de gravedad. Suponga que un objeto con masa cae desde una altura S_0 y que la altura del objeto después de segundos es

$$S(t) = S_0 - \frac{mg}{k}t + \frac{m^2g}{k^2}\left(1 - e^{-\frac{k}{m}t}\right)$$

donde g = 9.81 m/s^2 y k representa el coeficiente de la resistencia del aire en Ns/m. Suponga S_0 = 300m, m = 0.25 kg y k = 0.1 Ns/m. Encuentre dentro de 0.01 , el tiempo que tarda un cuarto de kg en golpear el piso.

```
import numpy as np

# Datos
s0 = 300   # m
m = 0.25   # kg
k = 0.1    # N·s/m
g = 9.81   # m/s^2

# Función f(t) = s(t)
def f(t):
```

```
    return s0 - (m*g/k)*t + (m**2 * g / k**2)*(1 - np.exp(-k*t/m))

# Método de bisección
def biseccion(f, a, b, tol=0.01, max_iter=100):
    if f(a)*f(b) > 0:
        return None, 0  # No hay cambio de signo
    for i in range(max_iter):
        c = (a + b)/2
        if f(c) == 0 or (b - a)/2 < tol:
            return c, i+1
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a+b)/2, max_iter

# Intervalo aproximado [0, 100] s
raiz, iteraciones = biseccion(f, 0, 100)
print(f"Tiempo para caer al piso: t = {raiz:.2f} s en {iteraciones} iteraciones")
```

```
Tiempo para caer al piso: t = 14.73 s en 14 iteraciones
```

EJERCICIOS TEÓRICOS

1. Use el teorema 2.1 para encontrar una cota para el número de iteraciones necesarias para lograr una aproximación con precisión de 10^{-4} para la solución de $x^3 - x - 1 = 0$ que se encuentra dentro del intervalo [1, 2]. Encuentre una aproximación para la raíz con este grado de precisión

```
import numpy as np

# Datos
a = 1
b = 2
tol = 1e-4

N_max = np.ceil((np.log(b - a) - np.log(tol)) / np.log(2))
print(f"Cota máxima de iteraciones: N  {N_max:.0f}")

# Método de bisección
f = lambda x: x**3 - x - 1
```

```python
def biseccion(f, a, b, tol=1e-4, max_iter=100):
    if f(a)*f(b) > 0:
        return None, 0
    for i in range(max_iter):
        c = (a + b)/2
        if f(c) == 0 or (b - a)/2 < tol:
            return c, i+1
        if f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a+b)/2, max_iter

raiz, iteraciones = biseccion(f, a, b, tol=tol, max_iter=int(N_max))
print(f"Raíz aproximada: x  {raiz:.4f} encontrada en {iteraciones} iteraciones")
```

```
Cota máxima de iteraciones: N  14
Raíz aproximada: x  1.3248 encontrada en 14 iteraciones
```

2. La función definida por $f(x) = sin\pi x$ tiene ceros en cada entero. Muestre cuando $-1 < a < 0$ y $2 < b < 3$, el método de bisección converge a

   a. 0, si $a + b < 2$

   El método de bisección encuentra el cero más cercano a la izquierda, que es 0, por lo tanto converge a la izquierda

   b. 2, si $a + b > 2$

   El método de bisección converge a la derecha porque encuentra el cero más cercano a la derecha, que es 2

   c. 1, si $a + b = 2$

   El método de bisección, también converge a la derecha y empieza exactamente en el punto medio, por lo que converge al cero = 1

Link del repositorio GitHub

Tarea N°4 en GitHub de Tamy Benavidez