Tarea Numero 4

Jose Sarango

Tabla de Contenidos

Conjunto de ejercicios **2**

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. 2

2. a. Dibuje las gráficas para = y = sin 3

3. a. Dibuje las gráficas para = y = tan 5

4. a. Dibuje las gráficas $ y = x^2 - 1 $ y $y = e^(1 - x^2)$. 7

5. Sea $f(x) = (x+3)(x+1)^2x(x-1)^3(x-3)$.¿En qué cero de converge el método de bisección cuando se aplica en los siguientes intervalos? 10

Ejercicios Aplicados **11**

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: 11

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: 12

Ejercicios Teoricos **14**

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. . . 14

2.La función definida por ( ) = sin tiene ceros en cada entero. Muestre cuando $-1 < \ < 0$ y $2 < \ < 3$, el método de bisección converge a: 15

Conjunto de 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]**

```
import math

def f(x):
    return x**3 - 7*x**2 + 14*x - 6

def bisection(a, b, tol=1e-2, max_iter=100):
    """
    Función que implementa el método de bisección para encontrar una raíz de f(x) = 0 en el i
    dentro de la tolerancia 'tol' y con un máximo de 'max_iter' iteraciones.
    """
    if f(a) * f(b) >= 0:
        print("La función no cambia de signo en el intervalo dado.")
        return None

    for i in range(max_iter):
        c = (a + b) / 2
        if abs(f(c)) < tol:
            print(f"Se encontró una aproximación de la raíz en {i+1} iteraciones.")
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c

    print(f"El método no converge después de {max_iter} iteraciones.")
    return None

# Definición de los intervalos y parámetros
intervalos = [(0, 1), (1, 3.2), (3.2, 4)]
tol = 1e-2
max_iter = 100
```

```
# Iteración sobre cada intervalo
for a, b in intervalos:
    root = bisection(a, b, tol, max_iter)
    if root is not None:
        print(f"Una aproximación de la raíz en el intervalo [{a}, {b}] con tolerancia {tol}
        print(f"El valor de la función en la raíz aproximada es: {f(root):.4f}")
    print()  # Línea en blanco para separar resultados
```

```
Se encontró una aproximación de la raíz en 7 iteraciones.
Una aproximación de la raíz en el intervalo [0, 1] con tolerancia 0.01 es: 0.5859
El valor de la función en la raíz aproximada es: 0.0010

Se encontró una aproximación de la raíz en 5 iteraciones.
Una aproximación de la raíz en el intervalo [1, 3.2] con tolerancia 0.01 es: 2.9938
El valor de la función en la raíz aproximada es: 0.0063

Se encontró una aproximación de la raíz en 6 iteraciones.
Una aproximación de la raíz en el intervalo [3.2, 4] con tolerancia 0.01 es: 3.4125
El valor de la función en la raíz aproximada es: -0.0020
```

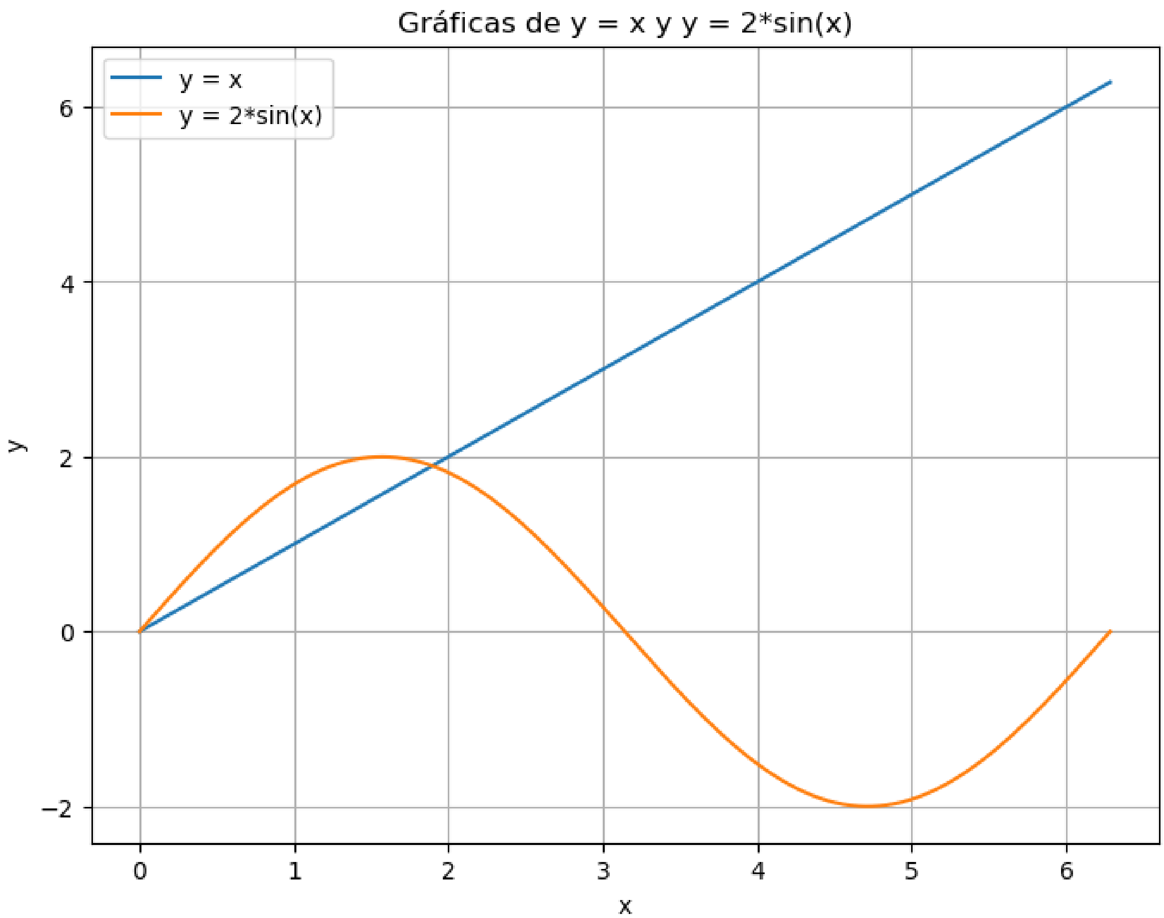
2. a. Dibuje las gráficas para $=$ y $= \sin$.

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

# Función para graficar y = x y y = 2*sin(x)
def graficar_funciones():
    x = np.linspace(0, 2 * np.pi, 1000)
    y1 = x
    y2 = 2 * np.sin(x)

    plt.figure(figsize=(8, 6))
    plt.plot(x, y1, label='y = x')
    plt.plot(x, y2, label='y = 2*sin(x)')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Gráficas de y = x y y = 2*sin(x)')
    plt.legend()
    plt.grid(True)
```

```
    plt.show()
graficar_funciones()
```



b. Use el método de bisección para encontrar soluciones precisas dentro de 10^(-5) para el primer valor positivo de con = 2 sin x.

```
import numpy as np
import math
```

```
def metodo_biseccion(funcion, a, b, tolerancia):
    if funcion(a) * funcion(b) >= 0:
        print("El método de bisección no puede aplicarse en este intervalo.")
        return None, 0  # Retornamos también el número de iteraciones como 0
```

```
    iteraciones = 0  # Inicializamos el contador de iteraciones

    while abs(b - a) > tolerancia:
        iteraciones += 1  # Incrementamos el contador de iteraciones
        c = (a + b) / 2
        if funcion(c) == 0:
            return c, iteraciones
        elif funcion(a) * funcion(c) < 0:
            b = c
        else:
            a = c

    return (a + b) / 2, iteraciones  # Retornamos la solución y el número de iteraciones

# Función para la ecuación x = 2 * sin(x)
def ecuacion(x):
    return x - 2 * np.sin(x)

# Encontrar la solución de x = 2 * sin(x) usando el método de bisección
a = 1  # Límite inferior del intervalo
b = 2  # Límite superior del intervalo (el primer cruce está en [0, 2])
tolerancia = 1e-5  # Tolerancia de 10^-5

solucion, iteraciones = metodo_biseccion(ecuacion, a, b, tolerancia)
if solucion is not None:
    print(f"La solución aproximada es: x = {solucion:.5f}")
    print(f"Número de iteraciones: {iteraciones}")
else:
    print("No se encontró una solución en el intervalo dado.")
```

```
La solución aproximada es: x = 1.89550
Número de iteraciones: 17
```

3. a. Dibuje las gráficas para = y = tan .

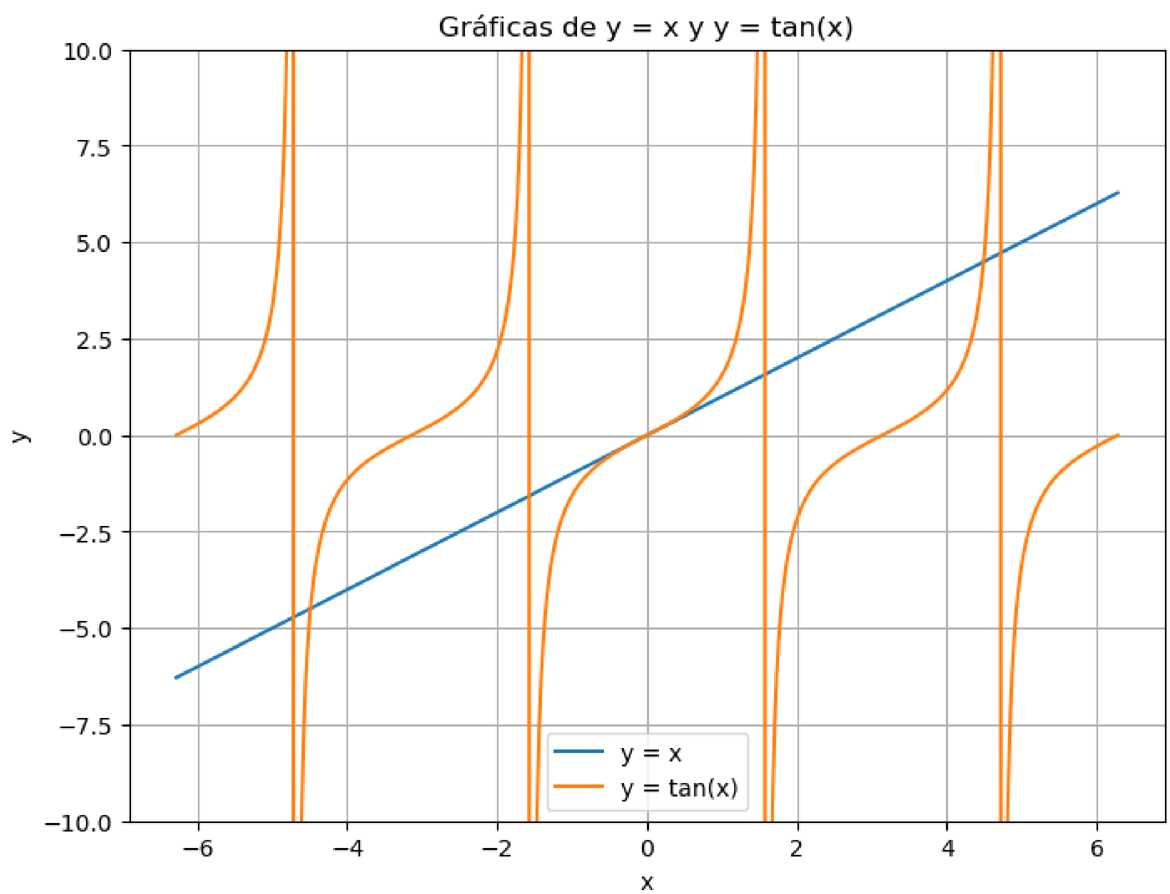
```
import numpy as np
import matplotlib.pyplot as plt

# Función para graficar y = x y y = tan(x)
def graficar_funciones():
```

```
    x = np.linspace(-2 * np.pi, 2 * np.pi, 1000)
    y1 = x
    y2 = np.tan(x)

    plt.figure(figsize=(8, 6))
    plt.plot(x, y1, label='y = x')
    plt.plot(x, y2, label='y = tan(x)')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Gráficas de y = x y y = tan(x)')
    plt.ylim(-10, 10)  # Establecer límites en y para una mejor visualización
    plt.legend()
    plt.grid(True)
    plt.show()

graficar_funciones()
```



b. Use el método de bisección para encontrar una aproximación dentro de 10^-5 para el primer valor positivo de con = tan .

```
import math

def bisection_method(tol):
    a = 1
    b = 2
    iterations = 0

    while (b - a) > tol:
        iterations += 1
        c = (a + b) / 2
        fc = math.tan(c) - c

        if abs(fc) <= tol:
            return c, iterations
        elif math.tan(a) * fc < 0:
            b = c
        else:
            a = c

    return (a + b) / 2, iterations

tolerance = 1e-5
approximation, iterations = bisection_method(tolerance)
print("Aproximación dentro de 10^(-5) para x = tan(x):", approximation)
print("Número de iteraciones:", iterations)
```

```
Aproximación dentro de 10^(-5) para x = tan(x): 1.5707969665527344
Número de iteraciones: 17
```

4. a. Dibuje las gráficas $ y = x^2 - 1 $ y $y = e^(1 - x^2)$.

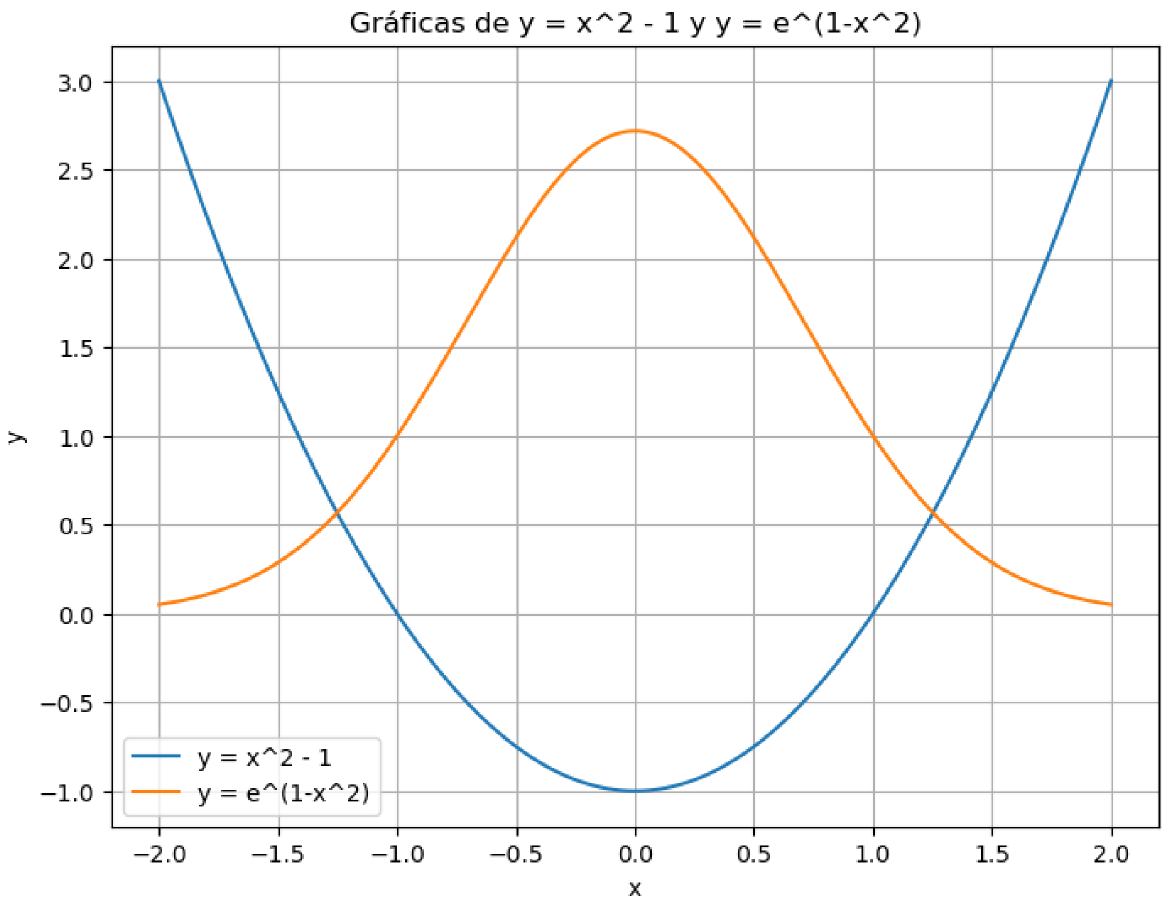
```
import numpy as np
import matplotlib.pyplot as plt

def graficar_funciones():
    x = np.linspace(-2, 2, 1000)
```

```python
    y1 = x**2 - 1
    y2 = np.exp(1 - x**2)

    plt.figure(figsize=(8, 6))
    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('Gráficas de y = x^2 - 1 y y = e^(1-x^2)')
    plt.legend()
    plt.grid(True)
    plt.show()

graficar_funciones()
```



b. Use el método de bisección para encontrar una aproximación dentro de $10^(-5)$

para un valor en [−2, 0] con $x^2 - 1 = e^{(}1 - x^2)$

```
import numpy as np

def f(x):
    return x**2 - 1 - np.exp(1 - x**2)

def bisection(f, a, b, tol):
    if f(a) * f(b) >= 0:
        print("La función no cambia de signo en el intervalo dado.")
        return None, 0  # Agregar un 0 para indicar ninguna iteración

    c = a
    iterations = 0  # Inicializar el contador de iteraciones
    while (b - a) >= tol:
        c = (a + b) / 2
        iterations += 1  # Incrementar el contador de iteraciones
        if f(c) == 0.0:
            break
        if f(c) * f(a) < 0:
            b = c
        else:
            a = c

    return c, iterations  # Devolver la aproximación y el número de iteraciones

# Intervalo [-2, 0]
a = -2
b = 0
tol = 1e-5

root, iterations = bisection(f, a, b, tol)

if root is not None:
    print(f"Una aproximación para la raíz en el intervalo [{a}, {b}] con tolerancia {tol} es
    print(f"Número de iteraciones realizadas: {iterations}")
else:
    print("No se encontró una raíz en el intervalo dado.")
```

```
Una aproximación para la raíz en el intervalo [-2, 0] con tolerancia 1e-05 es: -1.25185
Número de iteraciones realizadas: 18
```

5. Sea $f(x) = (x+3)(x+1)^2 x(x-1)^3(x-3)$.¿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]
d. [-3;-0.5]

```python
def bisection_method(a, b, func, tol=1e-6, max_iter=100):
    if func(a) * func(b) > 0:
        print("El método de la bisección falló.")
        return None, None
    else:
        iter_count = 0
        while ((b - a) / 2.0 > tol) and (iter_count < max_iter):
            c = (a + b) / 2.0
            if func(c) == 0:
                return iter_count, c
            elif func(a) * func(c) < 0:
                b = c
            else:
                a = c
            iter_count += 1
        return iter_count, (a + b) / 2.0

def f(x):
    return (x + 3) * (x + 1)**2 * x * (x - 1)**3 * (x - 3)

intervals = [(-1.5, 2.5), (-0.5, 2.4), (-0.5, 3), (-3, -0.5)]

for interval in intervals:
    iterations, root = bisection_method(interval[0], interval[1], f)
    if iterations is not None:
        print(f"Número de iteraciones en el intervalo {interval}: {iterations}")
        print(f"La raíz de f(x) en el intervalo {interval} es {root:.4f}.")
    else:
        print(f"No se encontró raíz en el intervalo {interval}.")
```

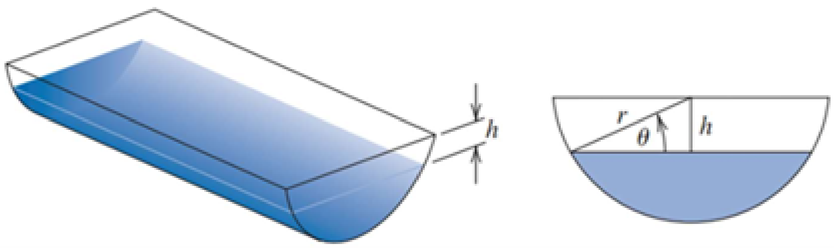
```
El método de la bisección falló.
No se encontró raíz en el intervalo (-1.5, 2.5).
El método de la bisección falló.
No se encontró raíz en el intervalo (-0.5, 2.4).
```

```
Número de iteraciones en el intervalo (-0.5, 3): 21
La raíz de f(x) en el intervalo (-0.5, 3) es 3.0000.
Número de iteraciones en el intervalo (-3, -0.5): 21
La raíz de f(x) en el intervalo (-3, -0.5) es -0.5000.
```

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:

$V = L[0.5\pi r^2 - r^2 arcsen(h/r) - h(r^2 - h^2)^(1/2)]$



Suponga que L=10 cm, r=1cm y V=12.4 cm^3. Encuentre la profundidad del agua en el abrevadero dentro de 0.01 cm.

```
import math

def ecuacion(h, L, r, V):
    return V - L * (0.5*math.pi*r**2 - math.asin(h) - h * math.sqrt(1 - h**2))

L = 10  # longitud del abrevadero en cm
r = 1   # radio del semicírculo en cm
V = 12.4  # volumen de agua en cm³

def biseccion(a, b, tol):
    iteraciones = 0
    while (b - a) / 2 > tol:
        iteraciones += 1
        c = (a + b) / 2
        if ecuacion(c, L, r, V) * ecuacion(a, L, r, V) < 0:
            b = c
        else:
            a = c
```

```
    return (a + b) / 2, iteraciones

a = 0  # límite inferior
b = 1  # límite superior
tol = 0.01  # tolerancia

hSolucion, numIteraciones = biseccion(a, b, tol)
print("RESPUESTA:")
print("La profundidad del agua en el abrevadero es aproximadamente {:.5f} cm".format(hSolucio
print("Número de iteraciones: {}".format(numIteraciones))
```

```
RESPUESTA:
La profundidad del agua en el abrevadero es aproximadamente 0.16406 cm
Número de iteraciones: 6
```

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:

$s(t) = s_0 - \frac{mg}{k}t + \frac{m^2 g}{k^2}\left(1 - e^{-\frac{kt}{m}}\right)$

```
import math

# Constantes
s0 = 300  # m
m = 0.25  # kg
g = 9.81  # m/s^2
k = 0.1   # Ns/m

# Función para calcular s(t)
def s(t):
    return s0 - (m * g / k) * t + (m**2 * g / k**2) * (1 - math.exp(-k * t / m))

# Método de bisección para encontrar la raíz de s(t) = 0
def bisection_method(func, a, b, tol=1e-2):
    if func(a) * func(b) >= 0:
        print("No se puede aplicar el método de bisección.")
        return None

    iter_count = 0  # Contador de iteraciones
```

```python
    c = a
    while (b - a) / 2 > tol:
        iter_count += 1
        c = (a + b) / 2
        print(f"Iteración {iter_count}: a = {a}, b = {b}, c = {c}, f(c) = {func(c)}")

        if func(c) == 0.0:
            break
        elif func(a) * func(c) < 0:
            b = c
        else:
            a = c

    return c

# Intervalo inicial para el método de bisección
a = 0
b = 20  # Tiempo suficientemente grande para que el objeto caiga al suelo

# Encontrar la raíz usando el método de bisección
root = bisection_method(s, a, b)

# Imprimir el resultado
if root is not None:
    print(f"El tiempo que tarda en golpear el piso es aproximadamente {root:.2f} segundos.")
else:
    print("No se encontró una raíz dentro de la tolerancia especificada.")
```

```
Iteración 1: a = 0, b = 20, c = 10.0, f(c) = 114.93952239063448
Iteración 2: a = 10.0, b = 20, c = 15.0, f(c) = -6.714478492831866
Iteración 3: a = 10.0, b = 15.0, c = 12.5, f(c) = 54.33687962461857
Iteración 4: a = 12.5, b = 15.0, c = 13.75, f(c) = 23.843179826179167
Iteración 5: a = 13.75, b = 15.0, c = 14.375, f(c) = 8.570480752413992
Iteración 6: a = 14.375, b = 15.0, c = 14.6875, f(c) = 0.929348305948146
Iteración 7: a = 14.6875, b = 15.0, c = 14.84375, f(c) = -2.892249013494208
Iteración 8: a = 14.6875, b = 14.84375, c = 14.765625, f(c) = -0.9813688453236153
Iteración 9: a = 14.6875, b = 14.765625, c = 14.7265625, f(c) = -0.025989572945810835
Iteración 10: a = 14.6875, b = 14.7265625, c = 14.70703125, f(c) = 0.45168458118872223
El tiempo que tarda en golpear el piso es aproximadamente 14.71 segundos.
```

Ejercicios Teoricos

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.

```python
def f(x):
    return x**3 - x - 1

def bisection_method(a, b, tol):
    n = 0
    while (b - a) / 2 > tol:
        midpoint = (a + b) / 2
        if f(midpoint) == 0:
            return midpoint, n
        elif f(a) * f(midpoint) < 0:
            b = midpoint
        else:
            a = midpoint
        n += 1
    return (a + b) / 2, n

# Parámetros iniciales
a = 1
b = 2
tolerance = 1e-4

# Ejecución del método de bisección
root, iterations = bisection_method(a, b, tolerance)

print(f"La raíz aproximada es: {root}")
print(f"El número de iteraciones necesarias fue: {iterations}")
```

```
La raíz aproximada es: 1.32476806640625
El número de iteraciones necesarias fue: 13
```

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

a.0, si a+b<2
b.2, si a+b>2
c.1, si a+b=2

```
import math

def f(x):
    return math.sin(math.pi * x)

def bisection_method(a, b, tol=1e-6):
    n = 0
    while (b - a) / 2 > tol:
        midpoint = (a + b) / 2
        if f(midpoint) == 0:
            return midpoint, n
        elif f(a) * f(midpoint) < 0:
            b = midpoint
        else:
            a = midpoint
        n += 1
    return (a + b) / 2, n

def find_zero(a, b, tol=1e-6):
    midpoint = (a + b) / 2
    if midpoint < 1:
        return 0, a, b
    elif midpoint > 1:
        return 2, a, b
    else:
        return 1, a, b

# Parámetros iniciales
a_values = [-0.5, -0.5, -0.5]
b_values = [2.5, 2.5, 2.5]
sum_values = [1.5, 3.5, 2]

for a, b, sum_ab in zip(a_values, b_values, sum_values):
    if sum_ab < 2:
        expected_zero = 0
    elif sum_ab > 2:
```

```python
        expected_zero = 2
    else:
        expected_zero = 1

    root, iterations = bisection_method(a, b)
    zero, _, _ = find_zero(a, b)

    print(f"Para a + b = {sum_ab}:")
    print(f"  Cero esperado: {expected_zero}")
    print(f"  Cero encontrado por bisección: {zero}")
    print(f"  Raíz aproximada: {root} después de {iterations} iteraciones")
    print()
```

```
Para a + b = 1.5:
  Cero esperado: 0
  Cero encontrado por bisección: 1
  Raíz aproximada: 2.384185791015625e-07 después de 21 iteraciones

Para a + b = 3.5:
  Cero esperado: 2
  Cero encontrado por bisección: 1
  Raíz aproximada: 2.384185791015625e-07 después de 21 iteraciones

Para a + b = 2:
  Cero esperado: 1
  Cero encontrado por bisección: 1
  Raíz aproximada: 2.384185791015625e-07 después de 21 iteraciones
```