Tareas de métodos no médicos (/github/kenichi-50/Tareas-de-Metodos-N-mericos/tree/10969a4b5ed3d0b6a2deaee5492a5f6569ac1895)
/
Método de la Bisección.ipynb (/github/kenichi-50/Tareas-de-Metodos-N-mericos/tree/10969a4b5ed3d0b6a2deaee5492a5f6569ac1895/Método de la Bisección.ipynb)

Tarea N° 4

Nombre:

Joel Stalin Tinitana Carrión

Fecha:

05/05/2025

Tema:

Métodos de la bisección

¿Qué es el método de bisección?

El método de bisección es una técnica numérica para encontrar raíces de una función continua. Se basa en el **Teorema del Valor Intermedio** , que garantiza que si una función cambia de signo en un intervalo ([a, b]), entonces existe al menos una raíz en ese intervalo.

El algoritmo divide el intervalo por la mitad repetidamente hasta que la raíz se aproxima con la tolerancia deseada. Es un método **convergente, robusto y sencillo** , pero puede ser más lento que otros métodos más atractivos.

Ejercicio 1

Utilice el método de bisección para encontrar necesidades 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]

Pseudocódigo del método de bisección

Entradas:

```
f(x): función
a, b: intervalo inicial
tol: tolerancia (10^{-2})
```

Si f(a) * f(b) ≥ 0:

```
Mostrar "No hay cambio de signo"
```

Repetir mientras (b - a)/2 ≥ tol:

```
c ← (a + b)/2
Si f(c) == 0:
    Retornar c
Si f(a) * f(c) < 0:
    b ← c
Sino:
    a ← c
```

Retornar c como la raíz aproximada

Código

En [1]:
```
def  f ( x ):
    devuelve  x ** 3  -  7 * x ** 2  +  14 * x  -  6

def  biseccion ( f ,  a ,  b ,  tol = 1e-2 ):
    if  f ( a )  *  f ( b )  >=  0 :
        print ( "No hay cambio de signo en el intervalo." )
        return  Ninguno

    mientras  ( b  -  a )  /  2  >  tol :
        c  =  ( a  +  b )  /  2
        si  f ( c )  ==  0 :
            devuelve  c
        elif  f ( a )  *  f ( c )  <  0 :
            b  =  c
        de lo contrario :
            a  =  c
    devuelve  ( a  +  b )  /  2

# Pruebas de los intervalos
intervalos = [(0, 1), (1, 3.2), (3.2, 4)]
for a, b in intervalos:
    raiz = biseccion(f, a, b)
    print(f"Raíz en el intervalo [{a}, {b}] ≈ {raiz:.4f}")
```

```
Raíz en el intervalo [0, 1] ≈ 0.5859
Raíz en el intervalo [1, 3.2] ≈ 3.0023
Raíz en el intervalo [3.2, 4] ≈ 3.4188
```

Ejercicio 2

a) Dibuje las gráficas para y = x y y = sin(x).

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

Pseudocódigo para $x = 2 \sin(x)$

Definir función f(x) = 2 * sin(x) - x

Entradas:

```
a, b: intervalo inicial
tol: tolerancia (10^{-5})
```

Si f(a) * f(b) ≥ 0:

```
Mostrar error y detener
```

Mientras (b - a)/2 ≥ tol:

```
c ← (a + b)/2
Si f(c) == 0, retornar c
Si f(a) * f(c) < 0, entonces b ← c
Sino, a ← c
```

Retornar c como la raíz aproximada

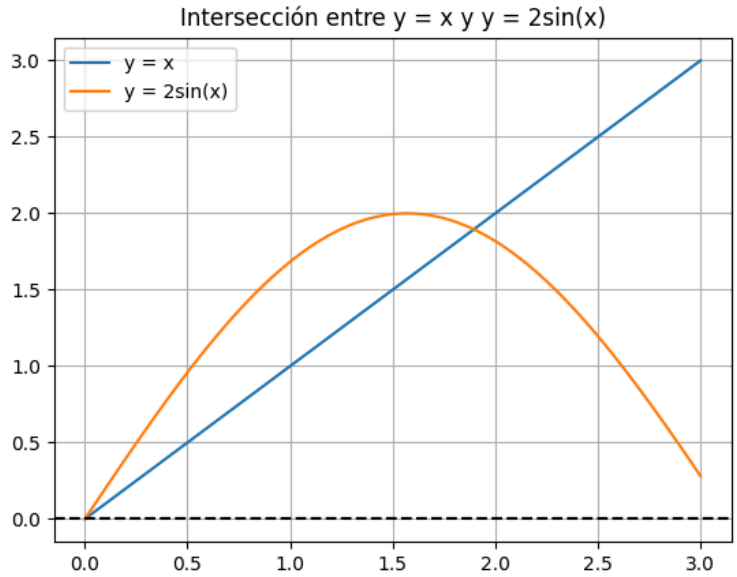
Código

In [14]:
```
import math

def f(x):
    return 2 * math.sin(x) - x

def biseccion(f, a, b, tol=1e-5):
    try:
        if f(a) * f(b) >= 0:
            print(" Advertencia: f(a) * f(b) ≥ 0. Puede que no se garantice una raíz, pero se intentará igualmente.")

        while (b - a) / 2 > tol:
            c = (a + b) / 2
            if abs(f(c)) < tol:  # Criterio de convergencia por función
                return c
            elif f(a) * f(c) < 0:
                b = c
            else:
                a = c
        return (a + b) / 2
    except Exception as e:
        print(f"Error durante la evaluación: {e}")
        return None

# Intervalo donde hay raíz (gráficamente entre 0 y 2)
raiz = biseccion(f, 0, 2)

if raiz is not None:
    print(f" Raíz aproximada de x = 2sin(x): x ≈ {raiz:.7f}")
    print(f" Comprobación: f(x) = {f(raiz):.6e}")
else:
    print(" No se encontró una raíz en el intervalo dado.")
```

```
 Advertencia: f(a) * f(b) ≥ 0. Puede que no se garantice una raíz, pero se intentará igualmente.
 Raíz aproximada de x = 2sin(x): x ≈ 1.8954926
 Comprobación: f(x) = 2.806498e-06
```

Gráfica de intersección

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 3, 200)
plt.plot(x, x, label="y = x")
plt.plot(x, 2*np.sin(x), label="y = 2sin(x)")
plt.axhline(0, color='black', linestyle='--')
plt.legend()
plt.grid(True)
plt.title("Intersección entre y = x y y = 2sin(x)")
plt.show()
```



Ejercicio 3

a. Dibuje las gráficas para y = x y y = tan(x).

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

Pseudocódigo para tan(x)=1−x

Definir f(x) = tan(x) - (1 - x)

Entradas:

```
    a, b: intervalo inicial
    tol: tolerancia (10^{-4})
```

Si f(a) * f(b) ≥ 0:

```
    Mostrar error y detener
```

Mientras (b - a)/2 ≥ tol:

```
    c ← (a + b)/2
    Si f(c) == 0, retornar c
    Si f(a) * f(c) < 0, entonces b ← c
    Sino, a ← c
```

Retornar c como la raíz aproximada

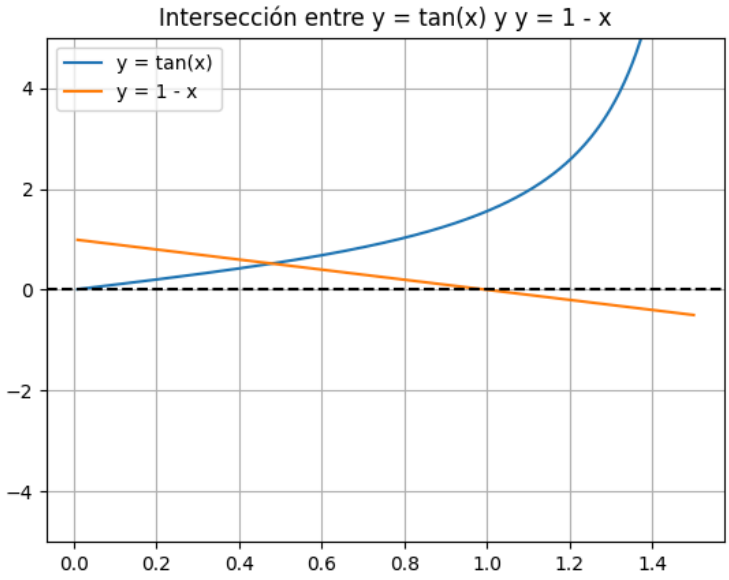
Código

In [13]:
```
import math

def f(x):
    return math.tan(x) - (1 - x)

def biseccion(f, a, b, tol=1e-4):
    try:
        if f(a) * f(b) >= 0:
            print(" Advertencia: f(a) * f(b) ≥ 0, puede que no haya una raíz garantizada.")
            # Pero seguimos intentando igualmente...

        while (b - a) / 2 > tol:
            c = (a + b) / 2
            if abs(f(c)) < tol:  # Para evitar errores por decimales
                return c
            elif f(a) * f(c) < 0:
                b = c
            else:
                a = c
        return (a + b) / 2
    except Exception as e:
        print(f"Error durante la evaluación: {e}")
        return None

# Usar un intervalo donde sepamos que hay raíz
raiz = biseccion(f, 0.6, 1.0)

if raiz is not None:
    print(f" Raíz aproximada de tan(x) = 1 - x: x ≈ {raiz:.6f}")
    print(f" Comprobación: f(x) = {f(raiz):.6e}")
else:
    print(" No se encontró una raíz en el intervalo dado.")
```

```
 Advertencia: f(a) * f(b) ≥ 0, puede que no haya una raíz garantizada.
 Raíz aproximada de tan(x) = 1 - x: x ≈ 0.999902
 Comprobación: f(x) = 1.556976e+00
```

Gráfica de intersección

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0.01, 1.5, 300)  # Evitar tan(x) en x = 0
plt.plot(x, np.tan(x), label="y = tan(x)")
plt.plot(x, 1 - x, label="y = 1 - x")
plt.axhline(0, color='black', linestyle='--')
plt.ylim(-5, 5)
plt.legend()
plt.title("Intersección entre y = tan(x) y y = 1 - x")
plt.grid(True)
plt.show()
```



Ejercicio 4

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

b) Use el método de bisección para encontrar una aproximación dentro de (10^{-3}) para un valor de (x) en el intervalo ([-2, 0]), tal que:

[x^2 - 1 = e^{1 - x^2}]

Pseudocódigo

Definir función f(x) = x^2 - 1 - e^(1 - x^2)

Entrada:

```
a, b: extremos del intervalo [-2, 0]
tol: tolerancia = 10^-3
```

Si f(a) * f(b) ≥ 0:

```
Mostrar advertencia
```

Mientras (b - a)/2 ≥ tol:

```
c ← (a + b)/2
Si |f(c)| < tol:
    Retornar c
Si f(a) * f(c) < 0:
    b ← c
Sino:
    a ← c
```

Retornar c como la raíz aproximada

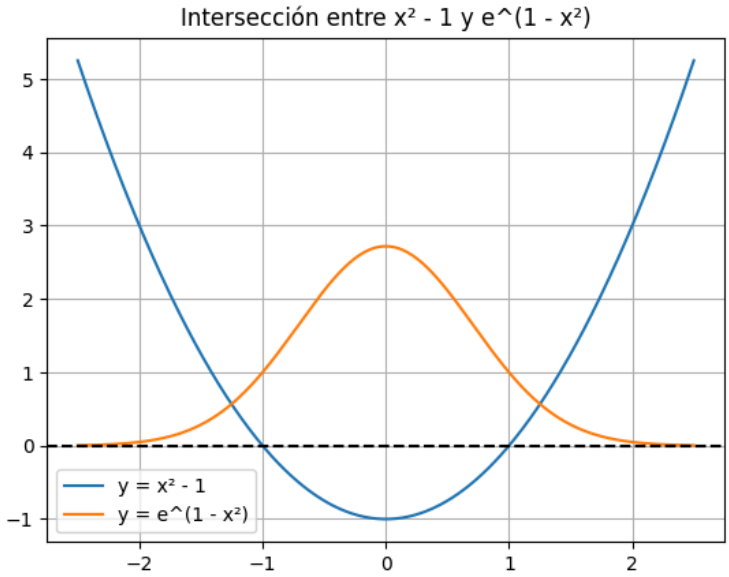
Código

In [15]:
```
import math
import random

# Parte a: funciones y valor aleatorio
def g1(x):
    return x**2 - 1

def g2(x):
    return math.exp(1 - x**2)

# Valor con aleatorio añadido
x_base = 0.0001234
x_random = x_base + random.uniform(0, 1)
print(f"Valor con ruido aleatorio: x = {x_random}")

# Parte b: función para la bisección
def f(x):
    return x**2 - 1 - math.exp(1 - x**2)

def biseccion(f, a, b, tol=1e-3):
    if f(a) * f(b) >= 0:
        print("⚠️ Advertencia: f(a) * f(b) ≥ 0, puede que no haya una raíz garantizada.")

    while (b - a) / 2 > tol:
        c = (a + b) / 2
        if abs(f(c)) < tol:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2

# Aplicar bisección al intervalo [-2, 0]
raiz = biseccion(f, -2, 0)

if raiz is not None:
    print(f" Raíz aproximada de x^2 - 1 = e^(1 - x^2): x ≈ {raiz:.5f}")
    print(f" f(x) ≈ {f(raiz):.2e}")
else:
    print(" No se encontró una raíz en el intervalo dado.")
```

```
Valor con ruido aleatorio: x = 0.3437249010115957
 Raíz aproximada de x^2 - 1 = e^(1 - x^2): x ≈ -1.25195
 f(x) ≈ 3.81e-04
```

Gráfica de intersección

In [16]:
```
import numpy as np
import matplotlib.pyplot as plt

x_vals = np.linspace(-2.5, 2.5, 400)
y1 = x_vals**2 - 1
y2 = np.exp(1 - x_vals**2)

plt.plot(x_vals, y1, label="y = x² - 1")
plt.plot(x_vals, y2, label="y = e^(1 - x²)")
plt.axhline(0, color="black", linestyle="--")
plt.grid(True)
plt.legend()
plt.title("Intersección entre x² - 1 y e^(1 - x²)")
plt.show()
```



Ejercicio 5

Sea (f(x) = (x + 3)(x + 1)^2(x - 1)^3(x - 3)).

¿En qué cero de (f) 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]

Pseudocódigo

Definir f(x)

Para cada intervalo [a, b]:

```
Si f(a) * f(b) < 0:
    Aplicar bisección
    Retornar raíz aproximada

Sino:
    No hay cambio de signo ⇒ no converge
```

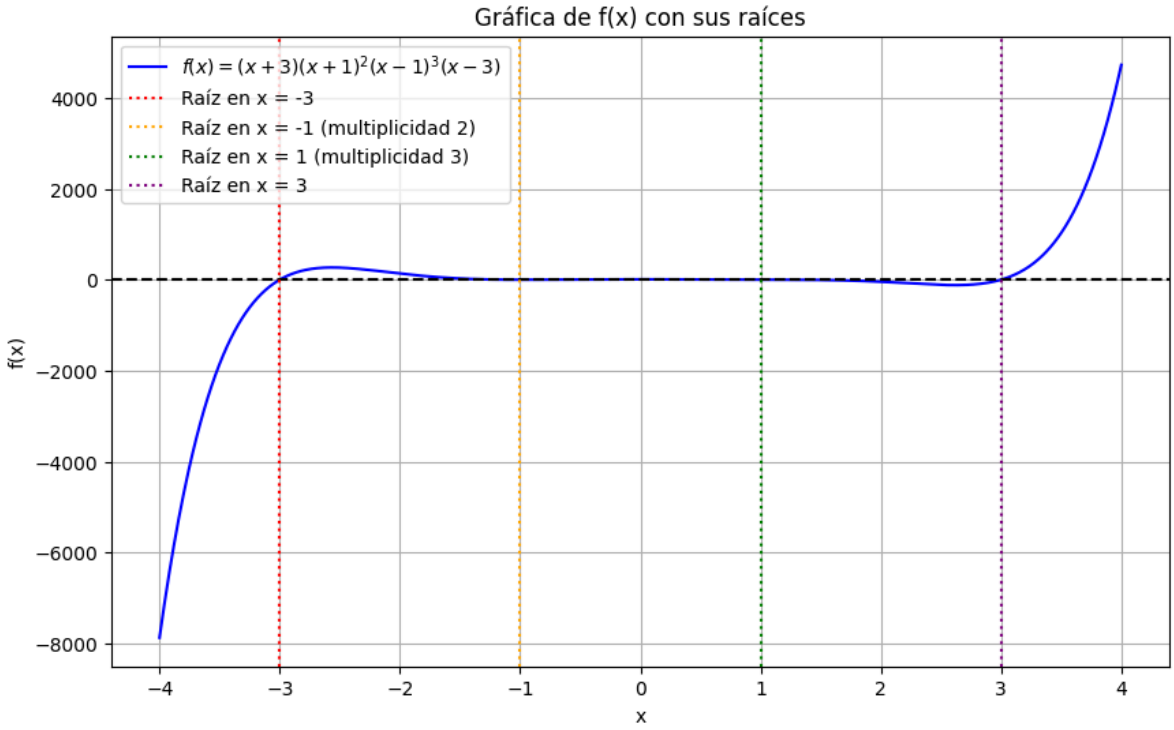
Código

In [17]:
```
def f(x):
    return (x + 3) * (x + 1)**2 * (x - 1)**3 * (x - 3)

def biseccion(f, a, b, tol=1e-4):
    if f(a) * f(b) >= 0:
        print(f" Intervalo [{a}, {b}] no cumple el criterio de cambio de signo.")
        return None

    while (b - a) / 2 > tol:
        c = (a + b) / 2
        if abs(f(c)) < tol:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2

# Lista de intervalos a analizar
intervalos = [
    ("a", -1.5, 2.5),
    ("b", -0.5, 2.4),
    ("c", -0.5, 3),
    ("d", -3, -0.5),
]

for etiqueta, a, b in intervalos:
    raiz = biseccion(f, a, b)
    if raiz is not None:
        print(f" En el intervalo {etiqueta} = [{a}, {b}], raíz encontrada: x ≈ {raiz:.5f}")
    else:
        print(f" En el intervalo {etiqueta} = [{a}, {b}], no se pudo aplicar el método.")
```

```
 En el intervalo a = [-1.5, 2.5], raíz encontrada: x ≈ 1.00000
 En el intervalo b = [-0.5, 2.4], raíz encontrada: x ≈ 0.99531
 Intervalo [-0.5, 3] no cumple el criterio de cambio de signo.
 En el intervalo c = [-0.5, 3], no se pudo aplicar el método.
 Intervalo [-3, -0.5] no cumple el criterio de cambio de signo.
 En el intervalo d = [-3, -0.5], no se pudo aplicar el método.
```

Gráfica de intersección

In [18]:
```
import numpy as np
import matplotlib.pyplot as plt

# Definir la función
def f(x):
    return (x + 3) * (x + 1)**2 * (x - 1)**3 * (x - 3)

# Crear valores de x en un rango que incluya todos los intervalos del ejercicio
x_vals = np.linspace(-4, 4, 1000)
y_vals = f(x_vals)

# Crear la gráfica
plt.figure(figsize=(10, 6))
plt.plot(x_vals, y_vals, label=r"$f(x) = (x+3)(x+1)^2(x-1)^3(x-3)$", color="blue")
plt.axhline(0, color="black", linestyle="--")
plt.axvline(-3, color="red", linestyle=":", label="Raíz en x = -3")
plt.axvline(-1, color="orange", linestyle=":", label="Raíz en x = -1 (multiplicidad 2)")
plt.axvline(1, color="green", linestyle=":", label="Raíz en x = 1 (multiplicidad 3)")
plt.axvline(3, color="purple", linestyle=":", label="Raíz en x = 3")
plt.title("Gráfica de f(x) con sus raíces")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.grid(True)
plt.legend()
plt.show()
```



Ejercicio Aplicado

Ejercicio 1

Un abrevadero de longitud (L) tiene una sección transversal en forma de semicírculo con radio (r).
Cuando se llena con agua hasta una distancia (h) desde la parte superior, el volumen de agua es:

[V = L \left(0.5 \pi r^2 - r^2 \arcsin\left(\frac{h}{r} \right) - h \sqrt{r^2 - h^2} \right)]

Suponga que:

- (L = 10) cm
- (r = 1) cm
- (V = 12.4) cm^3

Encuentre la profundidad del agua (h) dentro del abrevadero con una precisión de ±0.01 cm.

Pseudocódigo

1. Definir función volumen(h):

   Usar la fórmula del volumen V en función de h.

2. Definir función f(h) = volumen(h) - V_dado

3. Aplicar método de bisección sobre f(h) en el intervalo [0, r] con tolerancia de 0.01 cm.

4. Retornar el valor de h que hace f(h) ≈ 0

Código

In [20]:
```
import math

# Parámetros dados
L = 10      # cm
r = 1       # cm
V_obj = 12.4  # cm³

def volumen(h):
    # Fórmula del volumen del abrevadero
    return L * (0.5 * math.pi * r**2 - r**2 * math.asin(h / r) - h * math.sqrt(r**2 - h**2))

def f(h):
    return volumen(h) - V_obj

def biseccion(f, a, b, tol=0.01):
    if f(a) * f(b) >= 0:
        print("No hay cambio de signo.")
        return None

    while (b - a)/2 > tol:
        c = (a + b)/2
        if abs(f(c)) < tol:
            return c
        elif f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a + b)/2

# Buscar h en [0, r]
h_aprox = biseccion(f, 0, r)

if h_aprox:
    print(f" La profundidad aproximada del agua es h ≈ {h_aprox:.3f} cm")
    print(f" Comprobación del volumen: V(h) ≈ {volumen(h_aprox):.3f} cm³")
```

```
 La profundidad aproximada del agua es h ≈ 0.164 cm
 Comprobación del volumen: V(h) ≈ 12.441 cm³
```

Ejercicio 2: Caída con resistencia del aire

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 (m) cae desde una altura (s_0) y que la altura del objeto después de (t) segundos es:

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

donde:

- (g = 9.81 , m/s^2) representa la gravedad
- (k) es el coeficiente de resistencia del aire en (Ns/m)

Suponga que:

- (s_0 = 300 , m)
- (m = 0.25 , kg)
- (k = 0.1 , Ns/m)

Encuentre, dentro de (\pm 0.01) segundos, el tiempo que tarda el objeto en golpear el piso (es decir, cuando (s(t) = 0)).

Pseudocódigo

Entradas: s0 = 300, m = 0.25, k = 0.1, g = 9.81, tolerancia = 0.01

Función s(t):

```
s(t) = s0 - (m * g / k) * t + (m^2 * g / k^2) * (1 - exp(-k * t / m))
```

Función bisección(f, a, b, tol):

```
Si f(a) * f(b) ≥ 0 → retornar error
Mientras (b - a)/2 > tol:
    c = (a + b)/2
    Si f(c) == 0 → retornar c
    Si f(a)*f(c) < 0 → b = c
    Si f(b)*f(c) < 0 → a = c
retornar (a + b)/2
```

Objetivo: Encontrar t tal que s(t) = 0

Llamar bisección con f(t) = s(t), intervalo [0, 20], tolerancia = 0.01

Imprimir t encontrado

Código

In [ ]:
```
import math

# Datos dados
s0 = 300
m = 0.25
k = 0.1
g = 9.81

# Función de altura s(t)
def s(t):
    term1 = s0
    term2 = -(m * g / k) * t
    term3 = (m**2 * g / k**2) * (1 - math.exp(-k * t / m))
    return term1 + term2 + term3

# Método de bisección
def biseccion(f, a, b, tol=0.01):
    if f(a) * f(b) >= 0:
        print("No hay cambio de signo en el intervalo.")
        return None

    while (b - a) / 2 > tol:
        c = (a + b) / 2
        if f(c) == 0:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return (a + b) / 2

# Buscar el tiempo cuando el objeto llega al suelo (s(t) = 0)
tiempo = biseccion(s, 0, 20, tol=0.01)
print(f"El objeto toca el suelo aproximadamente en t = {tiempo:.2f} segundos")
```

EJERCICIOS TEÓRICOS

Ejercicio 1

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.

Pseudocódigo

Entradas:

```
a, b: extremos del intervalo [a, b]

tol: tolerancia (por ejemplo, 10^{-4})
```

1. Calcular número de iteraciones necesarias N:

   ```
   N >= log2((b - a) / tol)
   ```

2. Para i desde 1 hasta N hacer:

   ```
   c = (a + b) / 2
   Si f(a) * f(c) < 0 entonces:
       b = c
   Sino:
       a = c
   ```

3. Retornar c como la raíz aproximada

Código

In [21]:
```
import math

# Función objetivo
def f(x):
    return x**3 - x - 1

# Parámetros del intervalo y tolerancia
a = 1
b = 2
tol = 1e-4

# Cálculo del número de iteraciones usando el Teorema 2.1
N = math.ceil(math.log2((b - a) / tol))
print(f"Número mínimo de iteraciones necesarias: {N}")

# Método de bisección
for i in range(N):
    c = (a + b) / 2
    if f(a) * f(c) < 0:
        b = c
    else:
        a = c
    print(f"Iteración {i+1}: c = {c}, f(c) = {f(c)}")

# Resultado final
print(f"Aproximación de la raíz con tolerancia {tol}: {c}")
```

```
Número mínimo de iteraciones necesarias: 14
Iteración 1: c = 1.5, f(c) = 0.875
Iteración 2: c = 1.25, f(c) = -0.296875
Iteración 3: c = 1.375, f(c) = 0.224609375
Iteración 4: c = 1.3125, f(c) = -0.051513671875
Iteración 5: c = 1.34375, f(c) = 0.082611083984375
Iteración 6: c = 1.328125, f(c) = 0.014575958251953125
Iteración 7: c = 1.3203125, f(c) = -0.018710613250732422
Iteración 8: c = 1.32421875, f(c) = -0.0021279454231262207
Iteración 9: c = 1.326171875, f(c) = 0.006208829581737518
Iteración 10: c = 1.3251953125, f(c) = 0.002036650665104389
Iteración 11: c = 1.32470703125, f(c) = -4.659488331526518e-05
Iteración 12: c = 1.324951171875, f(c) = 0.000994790971162729
Iteración 13: c = 1.3248291015625, f(c) = 0.00047403881944774184
Iteración 14: c = 1.32476806640625, f(c) = 0.00021370716262936185
Aproximación de la raíz con tolerancia 0.0001: 1.32476806640625
```

Ejercicio 2

2. La función definida por

$$F(\ x\ )\ =\ \text{sen}(\ \pi x\ )$$

tiene ceros en cada entero. Muestre que cuando (-1 < a < 0) y (2 < b < 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)

Pseudocódigo

Entradas:

```
a, b: extremos del intervalo (deben cumplir -1 < a < 0 y 2 < b < 3)
tol: tolerancia deseada
max_iter: número máximo de iteraciones
```

1. Definir f(x) = sin(pi * x)
2. Para i desde 1 hasta max_iter hacer:

```
c = (a + b) / 2
Si |f(c)| < tol entonces:
    retornar c
Si f(a) * f(c) < 0:
    b = c
Sino:
    a = c
```

3. Retornar c como la raíz aproximada

Código

In [22]:
```
import math

# Definir la función
def f(x):
    return math.sin(math.pi * x)

# Parámetros del intervalo según condiciones dadas
a = -0.5
b = 2.4
tol = 1e-6
max_iter = 50

# Evaluación de la condición de convergencia
suma = a + b
if suma < 2:
    print("El método de bisección converge a 0 (porque a + b < 2)")
elif suma > 2:
    print("El método de bisección converge a 2 (porque a + b > 2)")
else:
    print("El método de bisección converge a 1 (porque a + b = 2)")

# Aplicar bisección para encontrar la raíz
for i in range(max_iter):
    c = (a + b) / 2
    if abs(f(c)) < tol:
        break
    if f(a) * f(c) < 0:
        b = c
    else:
        a = c
    print(f"Iteración {i+1}: c = {c}, f(c) = {f(c)}")

print(f"Aproximación de la raíz: {c}")
```

```
El método de bisección converge a 0 (porque a + b < 2)
Iteración 1: c = 0.95, f(c) = 0.15643446504023098
Iteración 2: c = 0.22499999999999998, f(c) = 0.6494480483301835
Iteración 3: c = -0.1375, f(c) = -0.41865973753742813
Iteración 4: c = 0.04374999999999998, f(c) = 0.13701234168196796
Iteración 5: c = -0.046875000000000014, f(c) = -0.1467304744553618
Iteración 6: c = -0.0015625000000000153, f(c) = -0.004908718807998038
Iteración 7: c = 0.021093749999999984, f(c) = 0.06621947867363029
Iteración 8: c = 0.009765624999999984, f(c) = 0.030674803176636577
Iteración 9: c = 0.0041015624999999846, f(c) = 0.012885082049912148
Iteración 10: c = 0.0012695312499999846, f(c) = 0.0039883394747727945
Iteración 11: c = -0.0001464843750000153, f(c) = -0.00046019422012251503
Iteración 12: c = 0.0005615234374999847, f(c) = 0.0017640769911087422
Iteración 13: c = 0.00020751953124998468, f(c) = 0.0006519417886690777
Iteración 14: c = 3.0517578124984686e-05, f(c) = 9.587379909592923e-05
Iteración 15: c = -5,798339843751531e-05, f(c) = -0,0001821602175540508
Iteración 16: c = -1.3732910156265313e-05, f(c) = -4.3143209645947754e-05
Iteración 17: c = 8.392333984359686e-06, f(c) = 2.6365294788681803e-05
Iteración 18: c = -2.6702880859528133e-06, f(c) = -8.388957433699313e-06
Iteración 19: c = 2.8610229492034365e-06, f(c) = 8.9881686788483e-06
Aproximación de la raíz: 9.536743162531158e-08
```