Tarea 4 BISECCIÓN

Pasquel Johann

Tabla de Contenidos

GITHUB

https://github.com/Vladimirjon/MetodosNumericos_PasquelJohann

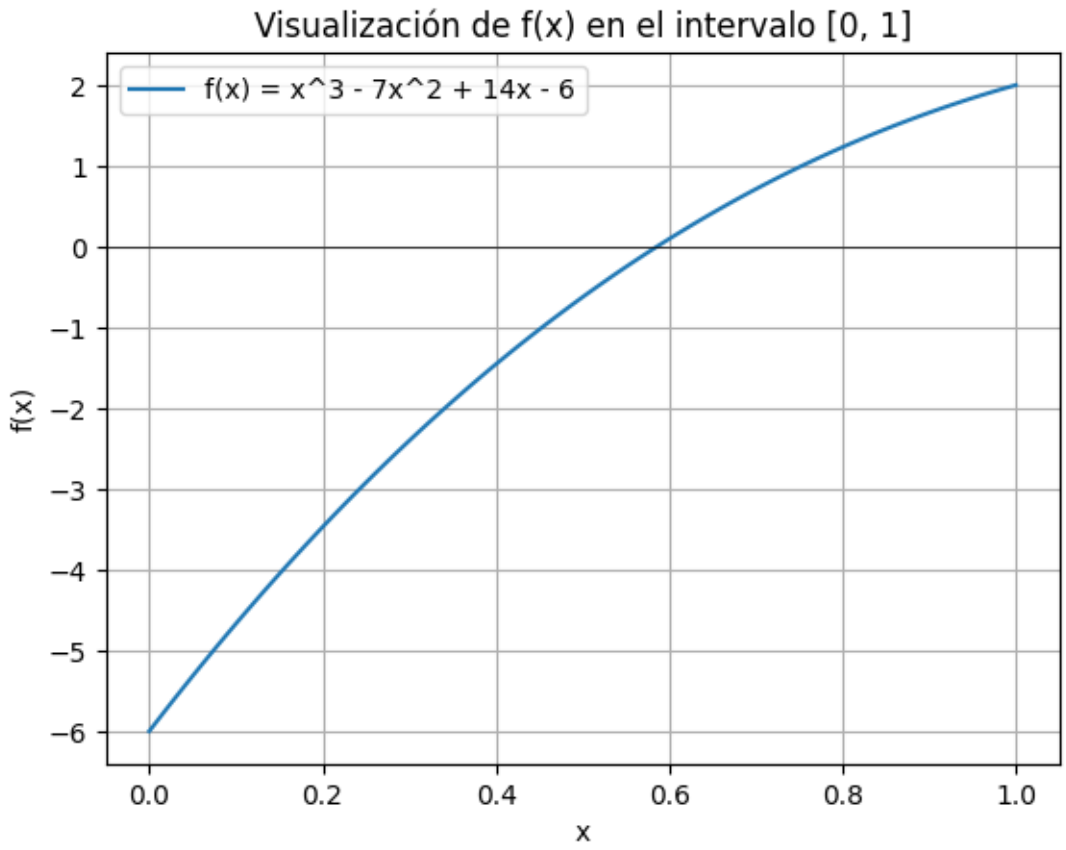
CONJUNTO DE EJERCICIOS

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]

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

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

a = 0
b = 1
tolerancia = 10**-2
max_iter = 100

x_vals = np.linspace(a, b, 100)
y_vals = f(x_vals)

plt.plot(x_vals, y_vals, label="f(x) = x^3 - 7x^2 + 14x - 6")
plt.axhline(0, color="black", linewidth=0.5)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Visualización de f(x) en el intervalo [0, 1]")
plt.legend()
plt.grid(True)
plt.show()

def biseccion(a, b, tolerancia, max_iter):
    if f(a) * f(b) >= 0:
```

```
        print("No hay raíz en el intervalo dado.")
        return None

    iteraciones = 0
    while (b - a) / 2 > tolerancia and iteraciones < max_iter:
        c = (a + b) / 2
        if f(c) == 0:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        iteraciones += 1
        print(f"Iteración {iteraciones}: a = {a}, b = {b}, c = {c}, f(c) = {f(c)}")


    return (a + b) / 2


raiz = biseccion(a, b, tolerancia, max_iter)
print(f"La raíz aproximada en el intervalo [0, 1] es: {raiz}")
```

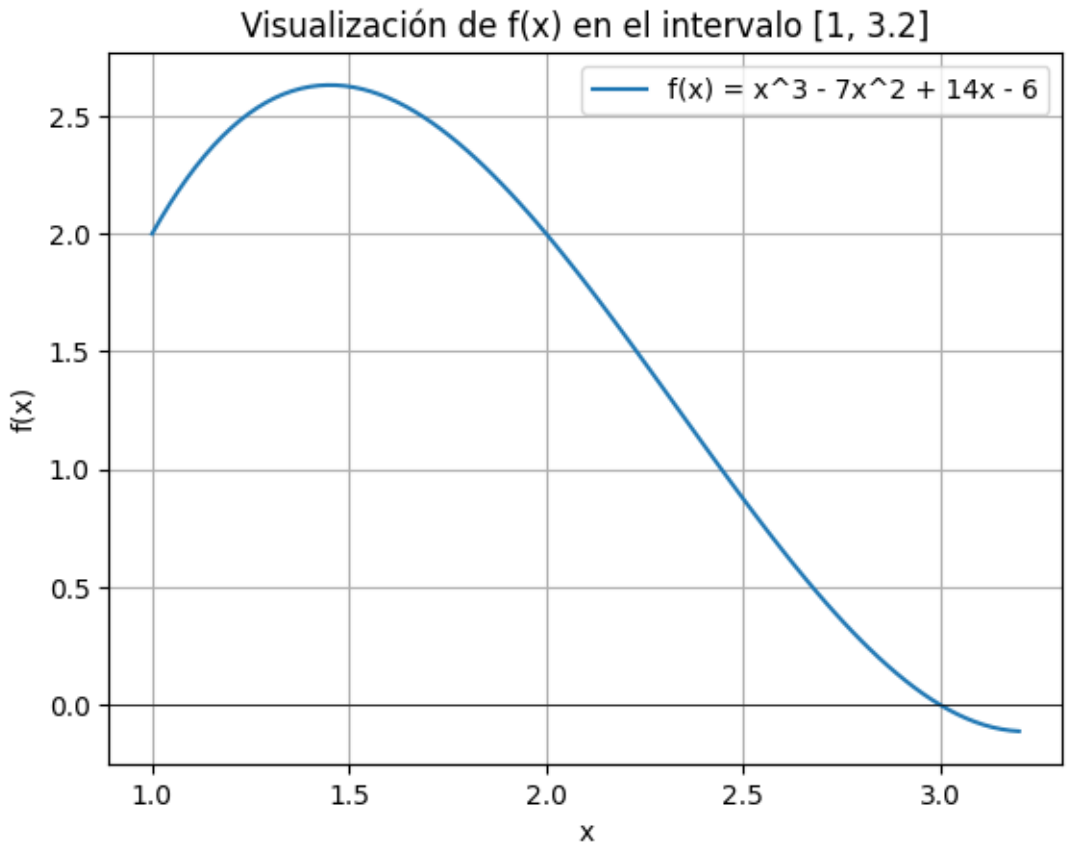



```
Iteración 1: a = 0.5, b = 1, c = 0.5, f(c) = -0.625
Iteración 2: a = 0.5, b = 0.75, c = 0.75, f(c) = 0.984375
Iteración 3: a = 0.5, b = 0.625, c = 0.625, f(c) = 0.259765625
Iteración 4: a = 0.5625, b = 0.625, c = 0.5625, f(c) = -0.161865234375
Iteración 5: a = 0.5625, b = 0.59375, c = 0.59375, f(c) = 0.054046630859375
Iteración 6: a = 0.578125, b = 0.59375, c = 0.578125, f(c) = -0.052623748779296875
La raíz aproximada en el intervalo [0, 1] es: 0.5859375
```

b. [1,3.2]

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

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

```
a = 1
b = 3.2
tolerancia = 10**-2
max_iter = 100

x_vals = np.linspace(a, b, 100)
y_vals = f(x_vals)

plt.plot(x_vals, y_vals, label="f(x) = x^3 - 7x^2 + 14x - 6")
plt.axhline(0, color="black", linewidth=0.5)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Visualización de f(x) en el intervalo [1, 3.2]")
plt.legend()
plt.grid(True)
plt.show()

def biseccion(a, b, tolerancia, max_iter):
    if f(a) * f(b) >= 0:
        print("No hay raíz en el intervalo dado.")
        return None

    iteraciones = 0
    while (b - a) / 2 > tolerancia and iteraciones < max_iter:
        c = (a + b) / 2
        if f(c) == 0:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        iteraciones += 1
        print(f"Iteración {iteraciones}: a = {a}, b = {b}, c = {c}, f(c) = {f(c)}")


    return (a + b) / 2


raiz = biseccion(a, b, tolerancia, max_iter)
print(f"La raíz aproximada en el intervalo [1, 3.2] es: {raiz}")
```

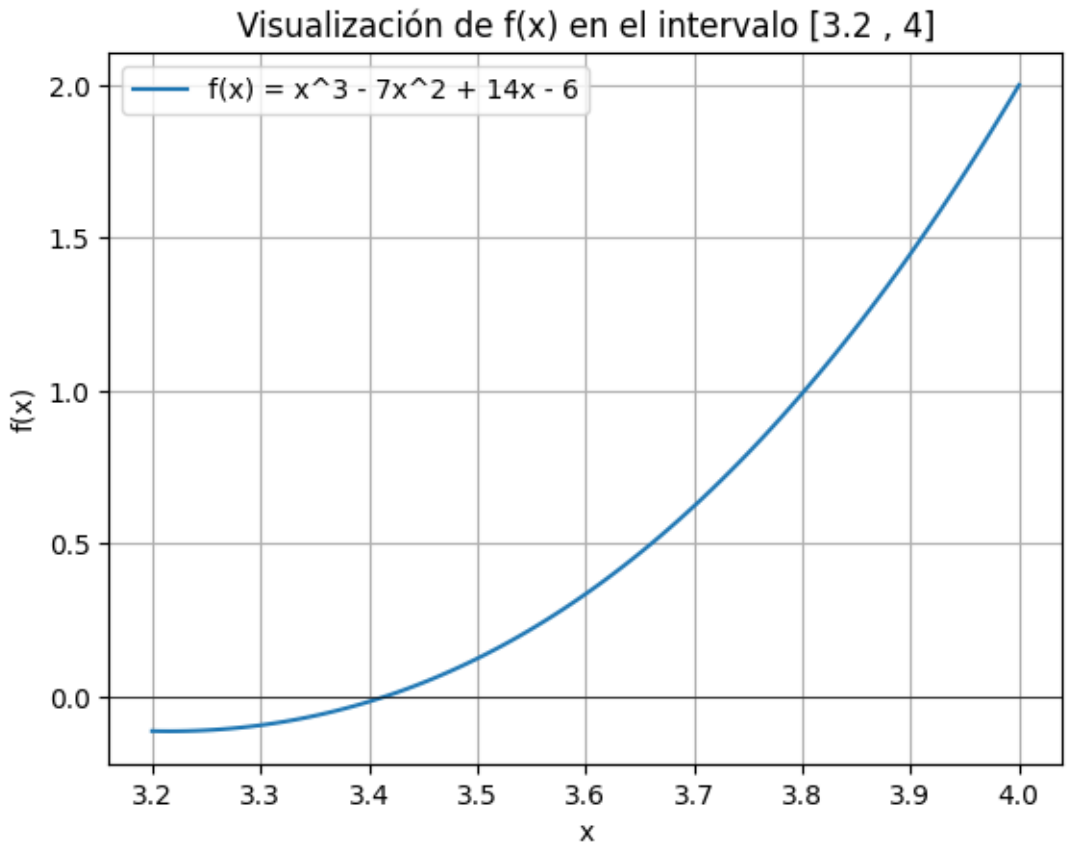



```
Iteración 1: a = 2.1, b = 3.2, c = 2.1, f(c) = 1.7910000000000004
Iteración 2: a = 2.6500000000000004, b = 3.2, c = 2.6500000000000004, f(c) = 0.5521250000000
Iteración 3: a = 2.9250000000000003, b = 3.2, c = 2.9250000000000003, f(c) = 0.0858281250000
Iteración 4: a = 2.9250000000000003, b = 3.0625, c = 3.0625, f(c) = -0.054443359375
Iteración 5: a = 2.9937500000000004, b = 3.0625, c = 2.9937500000000004, f(c) = 0.0063278808
Iteración 6: a = 2.9937500000000004, b = 3.028125, c = 3.028125, f(c) = -0.02652072143553852
Iteración 7: a = 2.9937500000000004, b = 3.0109375000000003, c = 3.0109375000000003, f(c) = -
La raíz aproximada en el intervalo [1, 3.2] es: 3.0023437500000005
```

c. [3.2,4]

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

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

```
a = 3.2
b = 4
tolerancia = 10**-2
max_iter = 100

x_vals = np.linspace(a, b, 100)
y_vals = f(x_vals)

plt.plot(x_vals, y_vals, label="f(x) = x^3 - 7x^2 + 14x - 6")
plt.axhline(0, color="black", linewidth=0.5)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Visualización de f(x) en el intervalo [3.2 , 4]")
plt.legend()
plt.grid(True)
plt.show()

def biseccion(a, b, tolerancia, max_iter):
    if f(a) * f(b) >= 0:
        print("No hay raíz en el intervalo dado.")
        return None

    iteraciones = 0
    while (b - a) / 2 > tolerancia and iteraciones < max_iter:
        c = (a + b) / 2
        if f(c) == 0:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        iteraciones += 1
        print(f"Iteración {iteraciones}: a = {a}, b = {b}, c = {c}, f(c) = {f(c)}")


    return (a + b) / 2


raiz = biseccion(a, b, tolerancia, max_iter)
print(f"La raíz aproximada en el intervalo [3.2 , 4] es: {raiz}")
```



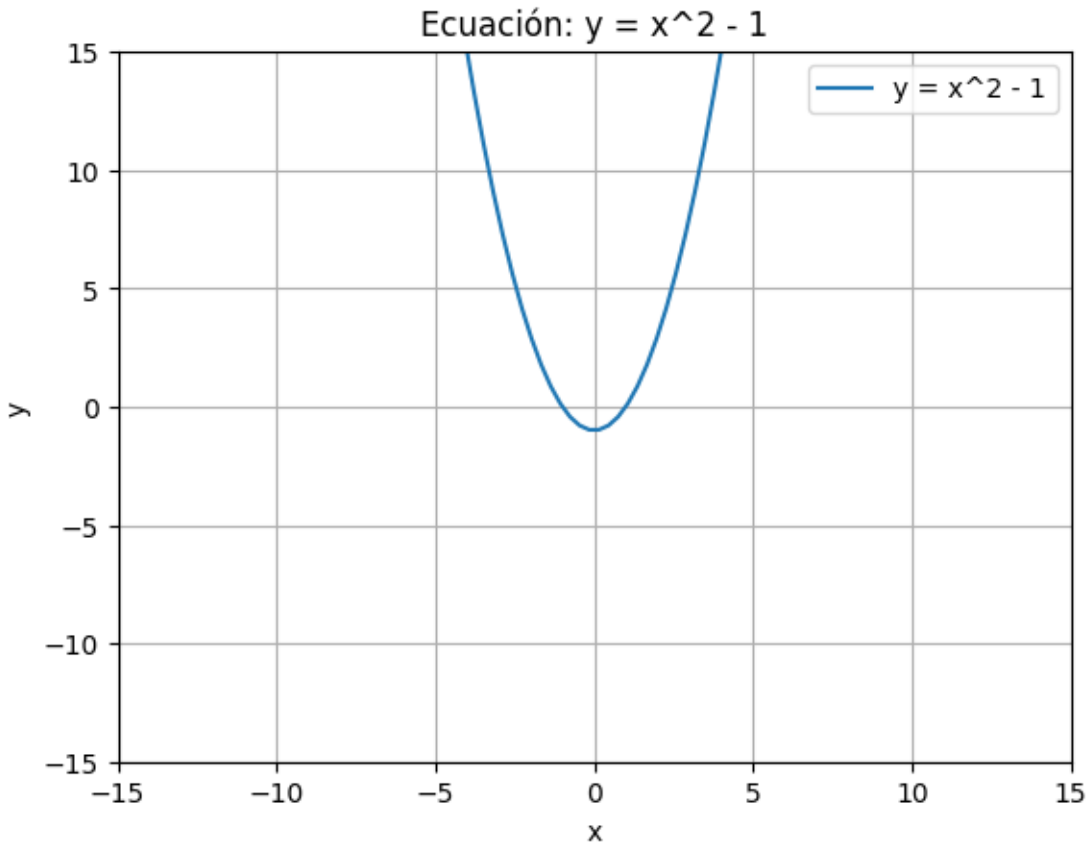

```
Iteración 1: a = 3.2, b = 3.6, c = 3.6, f(c) = 0.3360000000000056
Iteración 2: a = 3.4000000000000004, b = 3.6, c = 3.4000000000000004, f(c) = -0.015999999999
Iteración 3: a = 3.4000000000000004, b = 3.5, c = 3.5, f(c) = 0.125
Iteración 4: a = 3.4000000000000004, b = 3.45, c = 3.45, f(c) = 0.046125000000003524
Iteración 5: a = 3.4000000000000004, b = 3.4250000000000003, c = 3.4250000000000003, f(c) = 
Iteración 6: a = 3.4125000000000005, b = 3.4250000000000003, c = 3.4125000000000005, f(c) = 
La raíz aproximada en el intervalo [3.2 , 4] es: 3.41875
```

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

$y = x^2 - 1$

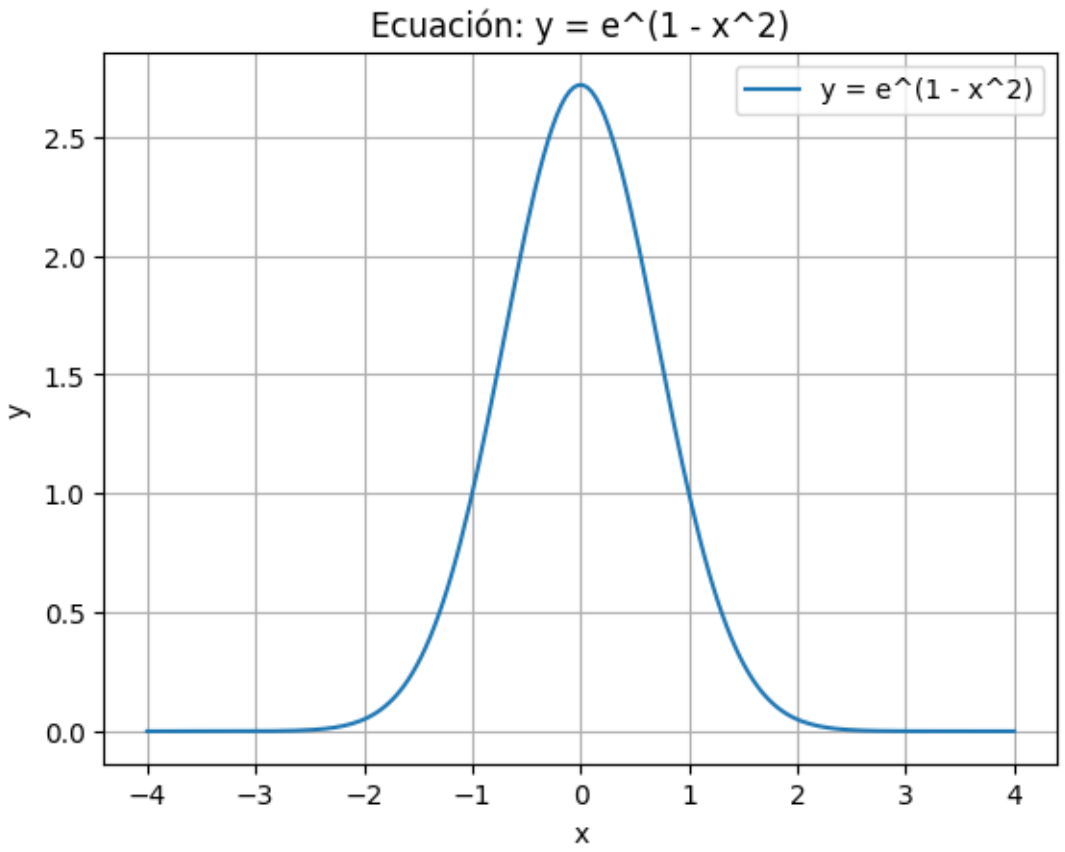
```
import numpy as np
import matplotlib.pyplot as plt

def equation(x):
```

```
    return x**2 - 1

x = np.linspace(-15, 15, 100)
y = equation(x)

plt.xlabel('x')
plt.ylabel('y')
plt.title('Ecuación: y = x^2 - 1')
ax = plt.gca()
ax.set_ylim([-15, 15])
ax.set_xlim([-15, 15])
plt.plot(x, y, label="y = x^2 - 1")
plt.legend()
plt.grid(True)
plt.show()
```



$y = e^{1-x^2}$

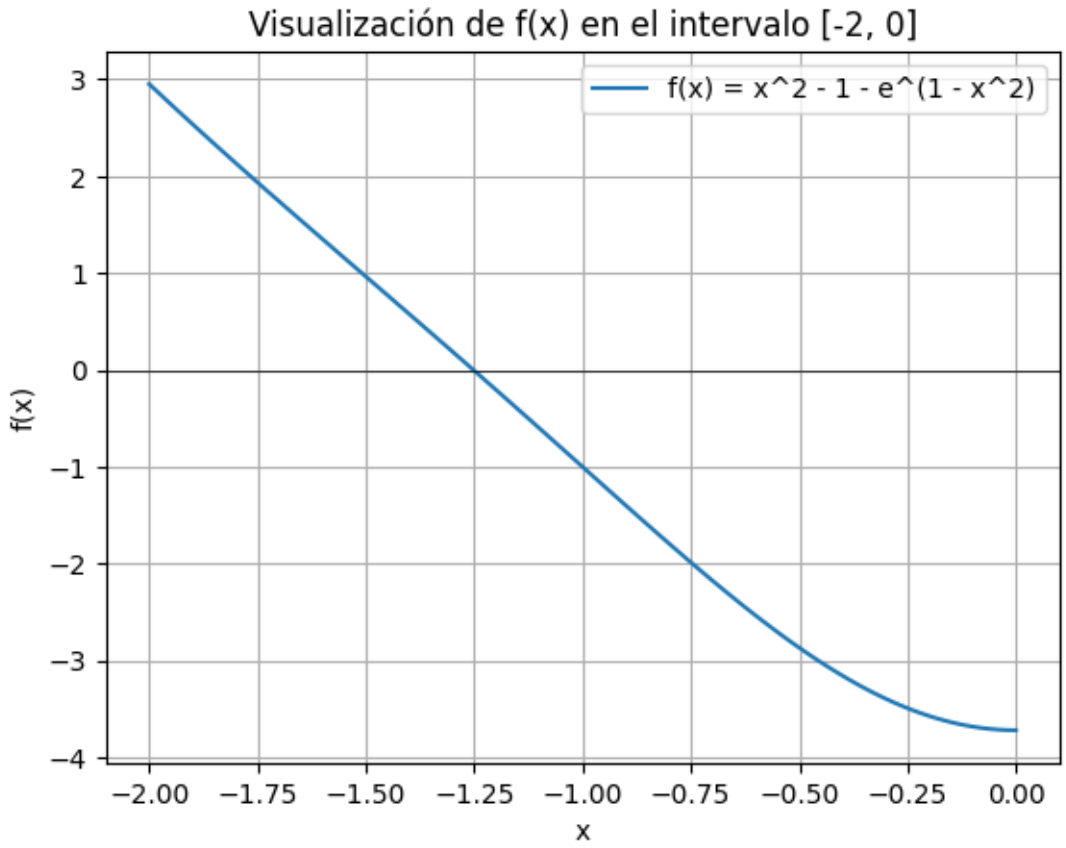
```python
import numpy as np
import matplotlib.pyplot as plt

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

x = np.linspace(-4, 4, 200)
y = equation(x)

plt.xlabel('x')
plt.ylabel('y')
plt.title('Ecuación: y = e^(1 - x^2)')
plt.plot(x, y, label="y = e^(1 - x^2)")
plt.legend()
plt.grid(True)
plt.show()
```



4.b Use el método de bisección para encontrar una aproximación dentro de 10^{-3} para un valor en [-2,0] con $x^2 - 1 = e^{1-x^2}$.

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

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

a = -2
b = 0
tolerancia = 10**-3
max_iter = 100

x_vals = np.linspace(a, b, 100)
y_vals = f(x_vals)

plt.plot(x_vals, y_vals, label="f(x) = x^2 - 1 - e^(1 - x^2)")
plt.axhline(0, color="black", linewidth=0.5)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Visualización de f(x) en el intervalo [-2, 0]")
plt.legend()
plt.grid(True)
plt.show()

def biseccion(a, b, tolerancia, max_iter):
    if f(a) * f(b) >= 0:
        print("No hay raíz en el intervalo dado.")
        return None

    iteraciones = 0
    while (b - a) / 2 > tolerancia and iteraciones < max_iter:
        c = (a + b) / 2
        if f(c) == 0:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        iteraciones += 1
        print(f"Iteración {iteraciones}: a = {a}, b = {b}, c = {c}, f(c) = {f(c)}")
```

```
    return (a + b) / 2

raiz = biseccion(a, b, tolerancia, max_iter)
print(f"La raíz aproximada en el intervalo [-2, 0] es: {raiz}")
```




```
Iteración 1: a = -2, b = -1.0, c = -1.0, f(c) = -1.0
Iteración 2: a = -1.5, b = -1.0, c = -1.5, f(c) = 0.9634952031398099
Iteración 3: a = -1.5, b = -1.25, c = -1.25, f(c) = -0.00728282473092301
Iteración 4: a = -1.375, b = -1.25, c = -1.375, f(c) = 0.48022582690363
Iteración 5: a = -1.3125, b = -1.25, c = -1.3125, f(c) = 0.23719521405913302
Iteración 6: a = -1.28125, b = -1.25, c = -1.28125, f(c) = 0.11515295433753148
Iteración 7: a = -1.265625, b = -1.25, c = -1.265625, f(c) = 0.05398561482037567
Iteración 8: a = -1.2578125, b = -1.25, c = -1.2578125, f(c) = 0.02336416096493721
Iteración 9: a = -1.25390625, b = -1.25, c = -1.25390625, f(c) = 0.008043872959257237
Iteración 10: a = -1.251953125, b = -1.25, c = -1.251953125, f(c) = 0.0003813268241048551
La raíz aproximada en el intervalo [-2, 0] es: -1.2509765625
```

EJERCICIOS APLICADOS

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

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

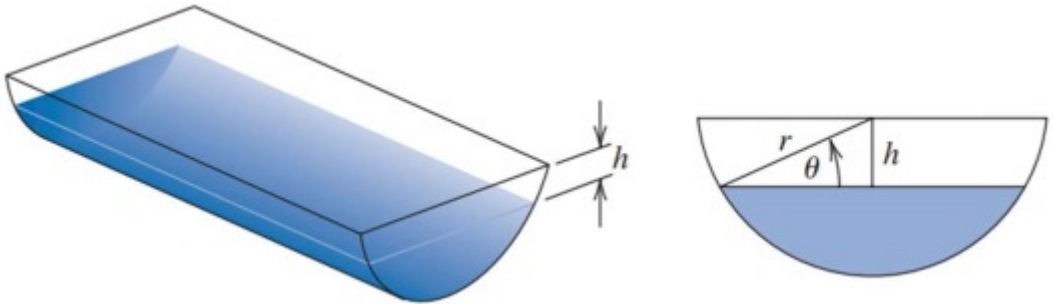
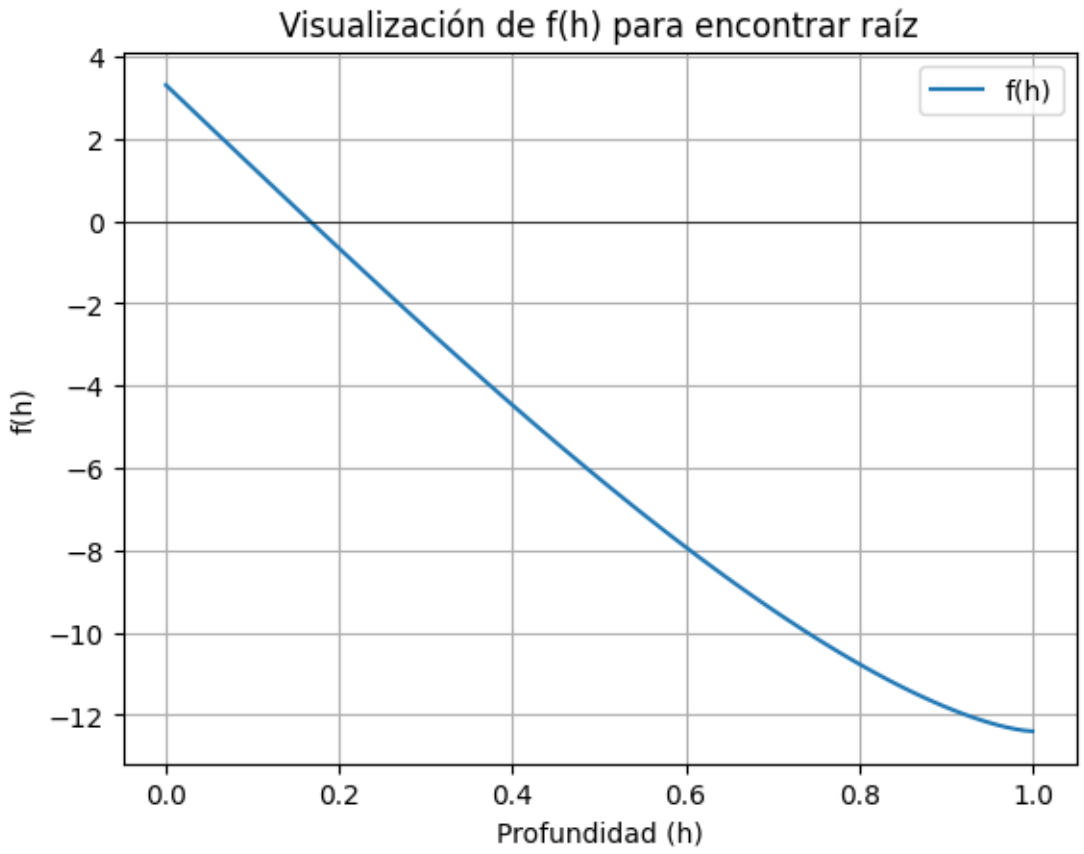


Figura 1: Ejercicio Aplicado

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

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

L = 10
r = 1
V_deseado = 12.4

def f(h):
    term1 = 0.5 * np.pi * r**2
    term2 = np.where((h >= 0) & (h <= r), r**2 * np.arcsin(h / r), np.nan)
    term3 = np.where((h >= 0) & (h <= r), h * np.sqrt(r**2 - h**2), np.nan)
    V_calculado = L * (term1 - term2 - term3)
    return V_calculado - V_deseado

h_vals = np.linspace(0, r, 100)
f_vals = f(h_vals)

plt.plot(h_vals, f_vals, label="f(h)")
```

```
plt.axhline(0, color="black", linewidth=0.5)
plt.xlabel("Profundidad (h)")
plt.ylabel("f(h)")
plt.title("Visualización de f(h) para encontrar raíz")
plt.legend()
plt.grid(True)
plt.show()

def biseccion(a, b, tolerancia=0.01, max_iter=100):
    if f(a) * f(b) >= 0:
        print("No hay raíz en el intervalo dado.")
        return None

    iteraciones = 0
    while (b - a) / 2 > tolerancia and iteraciones < max_iter:
        c = (a + b) / 2
        if np.isnan(f(c)):
            break
        if f(c) == 0:
            return c
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        iteraciones += 1
        print(f"Iteración {iteraciones}: a = {a}, b = {b}, c = {c}, f(c) = {f(c)}")

    return (a + b) / 2

a = 0
b = r
tolerancia = 0.01

h_aproximada = biseccion(a, b, tolerancia)
print(f"La profundidad aproximada del agua en el abrevadero es: {h_aproximada} cm")
```




```
Iteración 1: a = 0, b = 0.5, c = 0.5, f(c) = -6.258151506956217
Iteración 2: a = 0, b = 0.25, c = 0.25, f(c) = -1.6394538748514567
Iteración 3: a = 0.125, b = 0.25, c = 0.125, f(c) = 0.8144890292067846
Iteración 4: a = 0.125, b = 0.1875, c = 0.1875, f(c) = -0.4199467241373078
Iteración 5: a = 0.15625, b = 0.1875, c = 0.15625, f(c) = 0.1957259025413176
Iteración 6: a = 0.15625, b = 0.171875, c = 0.171875, f(c) = -0.11253639384839786
La profundidad aproximada del agua en el abrevadero es: 0.1640625 cm
```

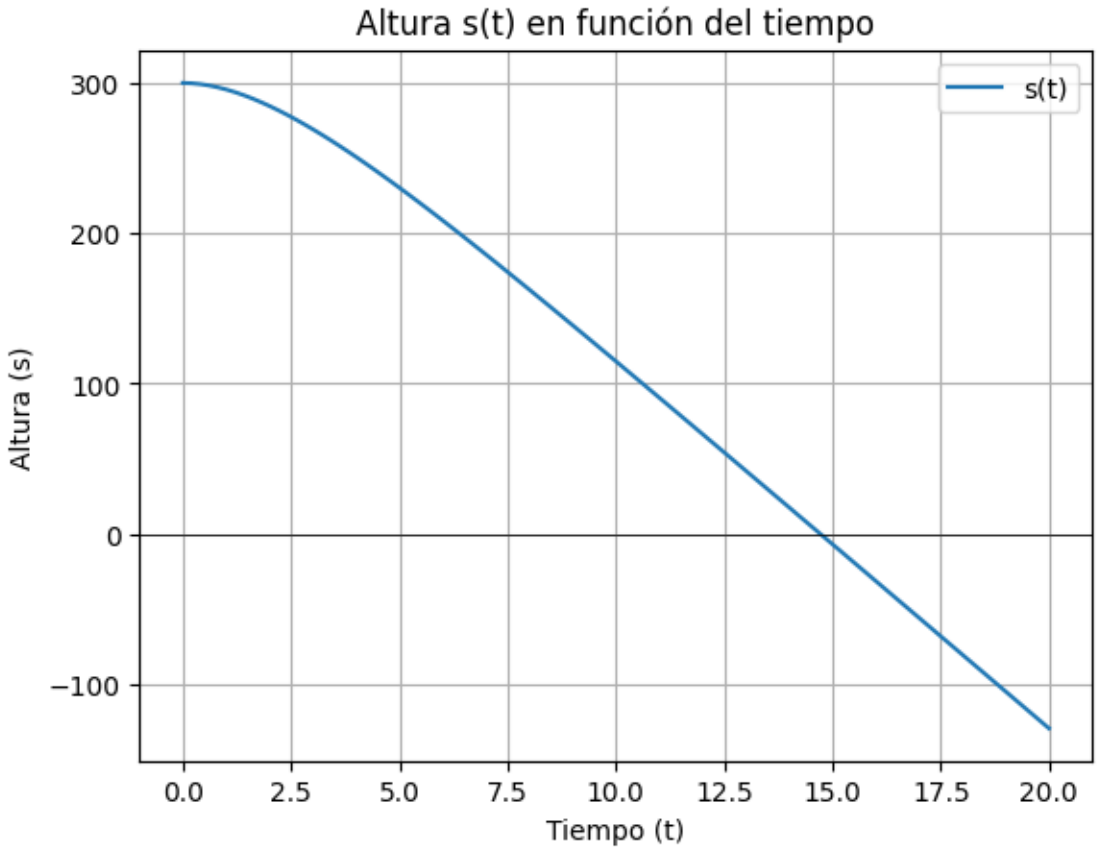
2. Un objeto que cae verticalmente a través del aire está sujeto a una resistencia viscosa, así como a la fuerza de gravedad. Suponga que un objeto con masa m cae desde una altura s_0 y que la altura del objeto después de t segundos es

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

donde $g = 9.81m/s^2$ y k representa el coeficiente de la resistencia del aire en Ns/m. Suponga $s_0 = 300m$, $m = 0.25kg$ y $k = 0.1Ns/m$. Encuentre, dentro de 0.01 segundos, el tiempo que tarda un cuarto de kg en golpear el piso.

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

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

def s(t):
    term1 = s0
    term2 = (m * g / k) * t
    term3 = (m**2 * g / k**2) * (1 - np.exp(-k * t / m))
    return term1 - term2 + term3

t_vals = np.linspace(0, 20, 100)
s_vals = s(t_vals)

plt.plot(t_vals, s_vals, label="s(t)")
plt.axhline(0, color="black", linewidth=0.5)
plt.xlabel("Tiempo (t)")
plt.ylabel("Altura (s)")
plt.title("Altura s(t) en función del tiempo")
plt.legend()
plt.grid(True)
plt.show()

def biseccion(a, b, tolerancia=0.01, max_iter=100):
    if s(a) * s(b) >= 0:
        print("No hay raíz en el intervalo dado.")
        return None

    iteraciones = 0
    while (b - a) / 2 > tolerancia and iteraciones < max_iter:
        c = (a + b) / 2
        if s(c) == 0:
            return c
        elif s(a) * s(c) < 0:
            b = c
        else:
```

```
            a = c
        iteraciones += 1
        print(f"Iteración {iteraciones}: a = {a}, b = {b}, c = {c}, s(c) = {s(c)}")

    return (a + b) / 2

a = 0
b = 20
tolerancia = 0.01
tiempo_impacto = biseccion(a, b, tolerancia)
print(f"El tiempo aproximado de impacto es: {tiempo_impacto} segundos")
```




```
Iteración 1: a = 10.0, b = 20, c = 10.0, s(c) = 114.93952239063448
Iteración 2: a = 10.0, b = 15.0, c = 15.0, s(c) = -6.714478492831866
Iteración 3: a = 12.5, b = 15.0, c = 12.5, s(c) = 54.33687962461857
Iteración 4: a = 13.75, b = 15.0, c = 13.75, s(c) = 23.843179826179167
Iteración 5: a = 14.375, b = 15.0, c = 14.375, s(c) = 8.570480752413992
```

```
Iteración 6: a = 14.6875, b = 15.0, c = 14.6875, s(c) = 0.929348305948146
Iteración 7: a = 14.6875, b = 14.84375, c = 14.84375, s(c) = -2.892249013494208
Iteración 8: a = 14.6875, b = 14.765625, c = 14.765625, s(c) = -0.9813688453236153
Iteración 9: a = 14.6875, b = 14.7265625, c = 14.7265625, s(c) = -0.025989572945810835
Iteración 10: a = 14.70703125, b = 14.7265625, c = 14.70703125, s(c) = 0.45168458118872223
El tiempo aproximado de impacto es: 14.716796875 segundos
```

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.

TEOREMA 2.1: BISECCIÓN

Para encontrar una solución (a f(x) = 0) dada la función continua determinada (f) en el intervalo ([a, b]), donde (f(a)) y (f(b)) tienen signos opuestos:

ENTRADA
- Puntos finales (a), (b); tolerancia (TOL); número máximo de iteraciones (N_0).

SALIDA
- Solución aproximada (p) o mensaje de falla.

Algoritmo

1. **Paso 1**
   Sea (i = 1);
   (FA = f(a)).
2. **Paso 2**
   Mientras (i N_0) haga los pasos 3-6.
3. **Paso 3**
   Sea (p = a + (b - a)/2); (Calcule (p_i))
   (FP = f(p)).

4. **Paso 4**
   Si (FP = 0) o ((b - a)/2 < TOL) entonces
   SALIDA ((p)); (Procedimiento completado exitosamente)
   PARE.

5. **Paso 5**
   Sea (i = i + 1).

6. **Paso 6**
   Si (FA FP > 0) entonces determine (a = p); (Calcule (a_i, b_i))
   (FA = FP)
   también determine (b = p). (FA no cambia).

7. **Paso 7**
   SALIDA ("El método fracasó después de (N_0) iteraciones, (N_0 =)", (N_0));
   (El procedimiento no fue exitoso)
   PARE.

```python
def bisection_method(f, a, b, tol, max_iter):
    # Paso 1
    i = 1
    FA = f(a)

    while i <= max_iter:
        # Paso 3
        p = (a + b) / 2
        FP = f(p)

        # Paso 4
        if FP == 0 or (b - a) / 2 < tol:
            return p, i

        # Paso 5
        i += 1

        # Paso 6
        if FA * FP > 0:
            a = p
            FA = FP
        else:
            b = p

    # Paso 7
    return None, i
```

```
# Function
f = lambda x: x**3 - x - 1

# Parámetros de input
a = 1
b = 2
tol = 1e-4
max_iter = 100

root, iterations = bisection_method(f, a, b, tol, max_iter)
root, iterations
```

```
(1.32476806640625, 14)
```

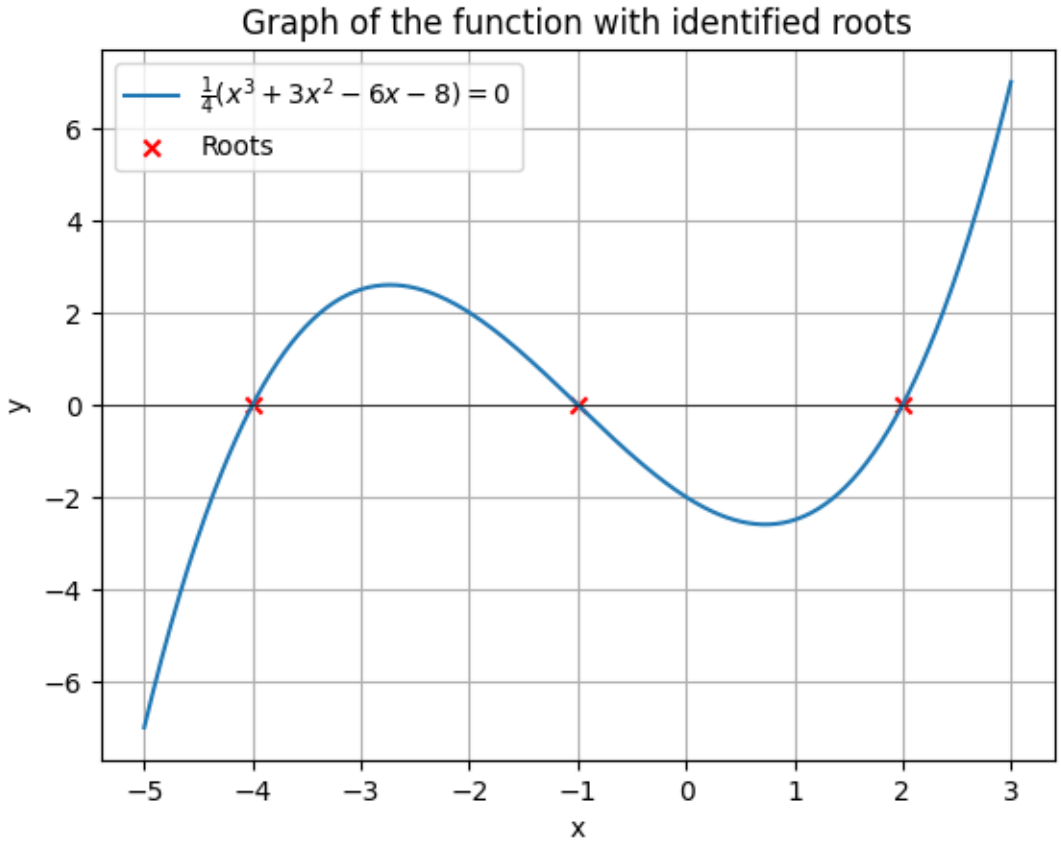
PRÁCTICA

Implementar el algoritmo y resolver $\frac{1}{4}(x^3 + 3x^2 - 6x - 8) = 0$

¿Cuántas raíces obtiene el algoritmo en el rango [-5 3]?

```
step = 0.1
roots = []
a = a_range

while a < b_range:
    b = a + step
    if equation(a) * equation(b) < 0:
        root_info = bisection(a, b, equation=equation, tol=tol, N=max_iter)
        if root_info:
            root, *_ = root_info
            if not any(abs(root - r) < tol for r in roots):
                roots.append(root)
    a = b

x = np.linspace(a_range, b_range, 500)
y = equation(x)

plt.plot(x, y, label=r"$\frac{1}{4}(x^3 + 3x^2 - 6x - 8) = 0$")
plt.xlabel('x')
```

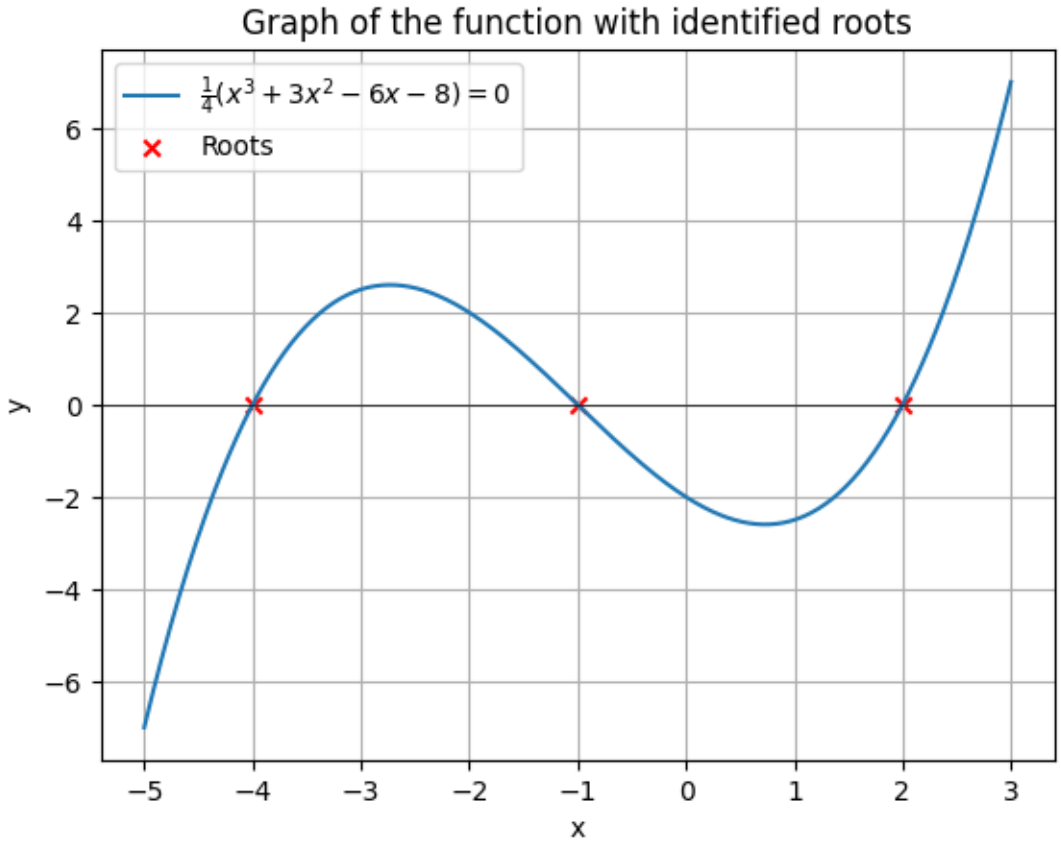
```
plt.ylabel('y')
plt.title('Graph of the function with identified roots')
plt.axhline(0, color='black', linewidth=0.5)
plt.scatter(roots, [0] * len(roots), color='red', marker='x', label='Roots')
plt.grid(True)
plt.legend()
plt.show()

len(roots), roots
```



¿A qué solución converge el algoritmo en el rango [-4.7 2.5]?

```
x = np.linspace(a_range, b_range, 500)
y = equation(x)
```

```
plt.plot(x, y, label=r"$\frac{1}{4}(x^3 + 3x^2 - 6x - 8) = 0$")
plt.xlabel('x')
plt.ylabel('y')
plt.title('Graph of the function with identified roots')
plt.axhline(0, color='black', linewidth=0.5)
plt.scatter(roots, [0] * len(roots), color='red', marker='x', label='Roots')
plt.grid(True)
plt.legend()
plt.show()
```




```
a_new_range, b_new_range = -4.7, 2.5

# Ejecutar el algoritmo de bisección en este intervalo para encontrar una única raíz
root_info = bisection(a_new_range, b_new_range, equation=equation, tol=tol, N=max_iter)

# Resultado
if root_info:
```

```
    root = root_info[0]
    print(f"En el intervalo [{a_new_range}, {b_new_range}], existe únicamente una raíz en x
else:
    print(f"No se encontró una raíz en el intervalo [{a_new_range}, {b_new_range}].")
```

```
En el intervalo [-4.7, 2.5], existe únicamente una raíz en x   -3.999999237060547.
```

¿Cuál es la respuesta en [-3 -2]?

```
# Verificar si hay raíces en el intervalo [-3, -2]
a_new_range, b_new_range = -3, -2

# Ejecutar el algoritmo de bisección en este intervalo
root_info = bisection(a_new_range, b_new_range, equation=equation, tol=tol, N=max_iter)

# Resultado
if root_info and equation(root_info[0]) == 0:
    root = root_info[0]
    print(f"En el intervalo [{a_new_range}, {b_new_range}], existe una raíz en x   {root}.")
else:
    print(f"No se encontró ninguna raíz en el intervalo [{a_new_range}, {b_new_range}].")
```

```
No se encontró ninguna raíz en el intervalo [-3, -2].
```

¿Qué sucede al ingresar rangos inválidos (e.g. [3 2])?

Si se ingresan rangos inválidos, el algoritmo de bisección no encuentra una raíz y devuelve un mensaje indicando que el intervalo es incorrecto o que no hay un cambio de signo en el rango especificado. Esto evita intentar encontrar raíces en intervalos inapropiados.