kp2v36hyp

May 18, 2025

ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA DE SISTEMAS

Campo	**Información**
Nombre:	**Michael Yànez**
Asignatura:	**Metodos Nùmericos**
Docente:	**Ing. Alejandro Zea**

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) $[0, 1]$

[1]:
```
def metodo_biseccion(funcion, inicio, fin, tolerancia=1e-5):
    if funcion(inicio) * funcion(fin) >= 0:
        print("La función no cambia de signo en el intervalo dado. Puede que no␣
 ↪haya raíces o haya múltiples raíces.")
        return None
    while abs(fin - inicio) >= tolerancia:
        punto_medio = (inicio + fin) / 2
        valor_medio = funcion(punto_medio)
        if valor_medio == 0:
            return punto_medio
        if funcion(inicio) * valor_medio < 0:
            fin = punto_medio
        else:
            inicio = punto_medio
    return (inicio + fin) / 2
def F(x):
    return x**3 - 7*x**2 + 14*x - 6
```

[2]:
```
intervalo_inicio, intervalo_fin = 0, 1
raiz = metodo_biseccion(F, intervalo_inicio, intervalo_fin)
print(f"La raíz en el intervalo [{intervalo_inicio}, {intervalo_fin}] es:␣
 ↪{raiz}")
```

```
La raíz en el intervalo [0, 1] es: 0.5857887268066406
```

b)$[1, 3.2]$

[3]:
```python
inicio_intervalo, fin_intervalo = 1, 3.2
raiz = metodo_biseccion(F, inicio_intervalo, fin_intervalo)
print(f"La raíz en el intervalo [{inicio_intervalo}, {fin_intervalo}] es: {raiz}")
```

```
La raíz en el intervalo [1, 3.2] es: 2.9999980926513676
```

c)$[3.2, 4]$

[4]:
```python
inicio_intervalo, fin_intervalo = 3, 2.4
raiz = metodo_biseccion(F, inicio_intervalo, fin_intervalo)
print(f"La raíz en el intervalo [{inicio_intervalo}, {fin_intervalo}] es: {raiz}")
```

```
La función no cambia de signo en el intervalo dado. Puede que no haya raíces o
haya múltiples raíces.
La raíz en el intervalo [3, 2.4] es: None
```

0.0.1 Ejercicio 2

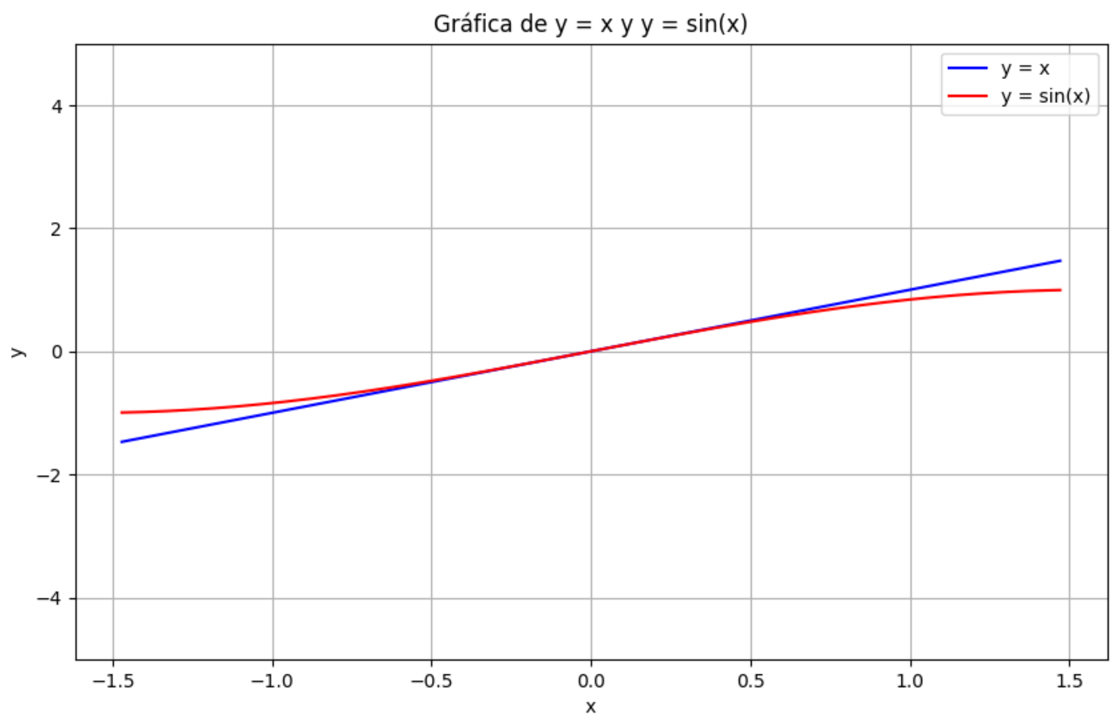
2.1 Dibuje las gráficas para $=$ y $= \sin$.

[30]:
```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-np.pi/2 + 0.1, np.pi/2 - 0.1, 400)
y1 = x          # y = x
y2 = np.sin(x)  # y = sin(x)

# Crear la gráfica
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label="y = x", color='blue')
plt.plot(x, y2, label="y = sin(x)", color='red')
plt.xlabel("x")
plt.ylabel("y")
plt.title("Gráfica de y = x y y = sin(x)")
plt.legend()
plt.grid(True)
plt.ylim(-5, 5)

plt.show()
```



2.2.)Use el método de bisección para encontrar soluciones precisas dentro de 10 elevado -5 para el primer valor positivo de con = 2 sin .

[36]:
```python
import numpy as np

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

def metodo_biseccion(funcion, inicio, fin, tolerancia=1e-5):
    if funcion(inicio) * funcion(fin) >= 0:
        print("La función no cambia de signo en el intervalo dado.")
        return None

    while abs(fin - inicio) >= tolerancia:
        punto_medio = (inicio + fin) / 2
        valor_medio = funcion(punto_medio)

        if abs(valor_medio) < tolerancia:
            return punto_medio
        if funcion(inicio) * valor_medio < 0:
            fin = punto_medio
        else:
            inicio = punto_medio
```

```python
    return (inicio + fin) / 2

# La primera solución positiva está entre 0 y  /2
inicio, fin = 0, np.pi/2
raiz = metodo_biseccion(f, inicio, fin)
print(f"La primera solución positiva de x = 2sin(x) es aproximadamente: {raiz}")
```

```
La función no cambia de signo en el intervalo dado.
La primera solución positiva de x = 2sin(x) es aproximadamente: None
```

0.0.2 Ejercicio 3

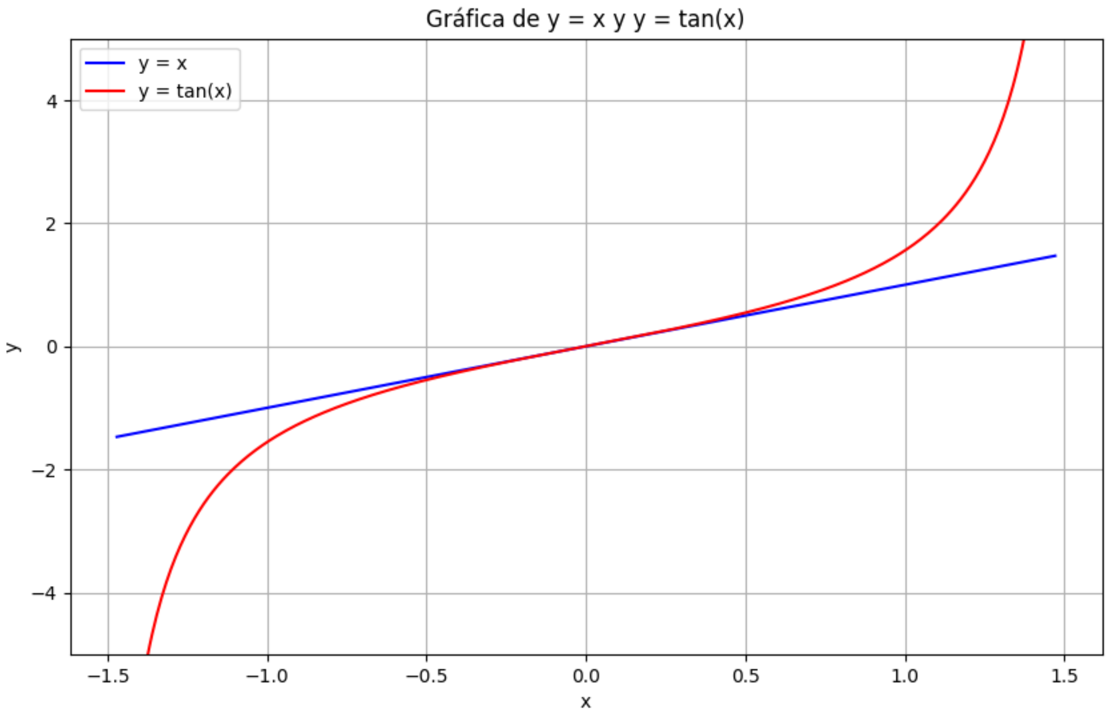
3.1. Dibuje las gráficas para $=$ y $= \tan$.

[37]:
```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-np.pi/2 + 0.1, np.pi/2 - 0.1, 400)
y1 = x           # y = x
y2 = np.tan(x)   # y = tan(x)

# Crear la gráfica
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label="y = x", color='blue')
plt.plot(x, y2, label="y = tan(x)", color='red')
plt.xlabel("x")
plt.ylabel("y")
plt.title("Gráfica de y = x y y = tan(x)")
plt.legend()
plt.grid(True)
plt.ylim(-5, 5)

plt.show()
```



3.2.) Use el método de bisección para encontrar una aproximación dentro de 10 elevado a -5 para el primer valor positivo de   con   = tan .

[38]:
```
import numpy as np

def f(x):
    return x - np.tan(x)

def metodo_biseccion(funcion, inicio, fin, tolerancia=1e-5):
    if funcion(inicio) * funcion(fin) >= 0:
        print("La función no cambia de signo en el intervalo dado.")
        return None

    while abs(fin - inicio) >= tolerancia:
        punto_medio = (inicio + fin) / 2
        valor_medio = funcion(punto_medio)

        if abs(valor_medio) < tolerancia:
            return punto_medio
        if funcion(inicio) * valor_medio < 0:
            fin = punto_medio
        else:
            inicio = punto_medio
```

```
    return (inicio + fin) / 2

# La primera solución positiva está entre 4.0 y 4.5
inicio, fin = 4.0, 4.5
raiz = metodo_biseccion(f, inicio, fin)
print(f"La primera solución positiva de x = tan(x) es aproximadamente: {raiz}")
```

```
La primera solución positiva de x = tan(x) es aproximadamente: 4.493412017822266
```

0.0.3 Ejercico 4

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

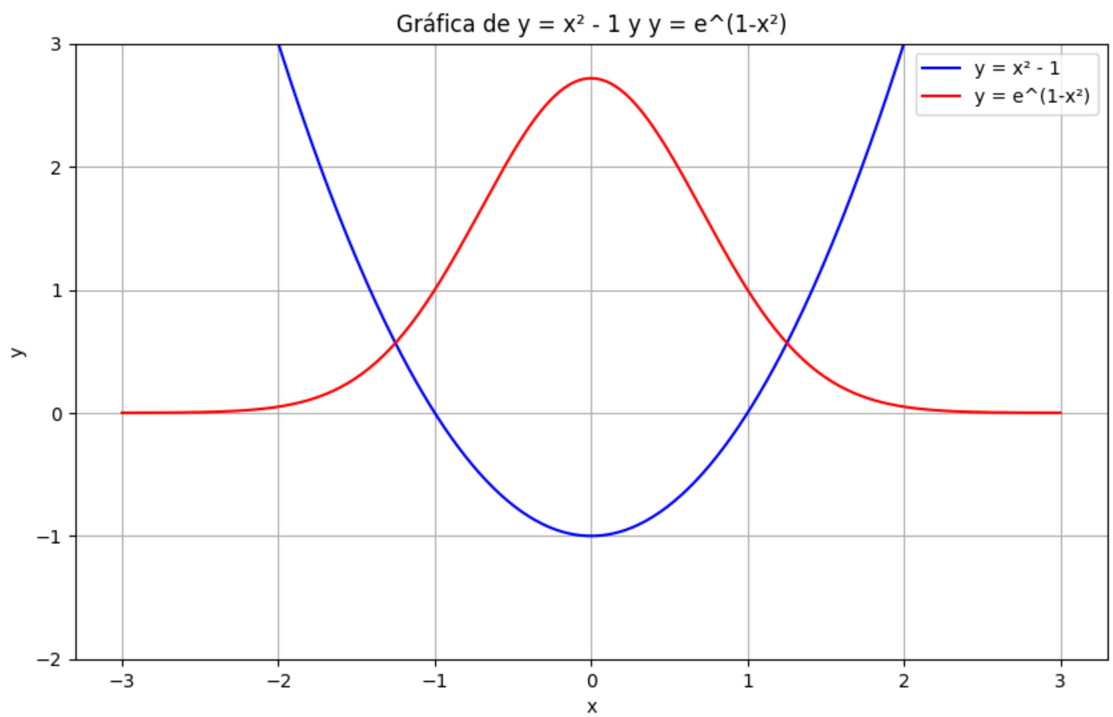
[43]:
```
import numpy as np
import matplotlib.pyplot as plt

# Crear un array de valores x
x = np.linspace(-3, 3, 400)

# Calcular los valores y para cada función
y1 = x**2 - 1        # y = x^2 - 1
y2 = np.exp(1 - x**2)  # y = e^(1-x^2)

# Crear la gráfica
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label="y = x² - 1", color='blue')
plt.plot(x, y2, label="y = e^(1-x²)", color='red')
plt.xlabel("x")
plt.ylabel("y")
plt.title("Gráfica de y = x² - 1 y y = e^(1-x²)")
plt.legend()
plt.grid(True)
plt.ylim(-2, 3)  # Ajustar límites del eje y para mejor visualización

plt.show()
```



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

[47]:
```
import numpy as np

def f(x):
    return x**2 - 1 - np.exp(1 - x**2)  # Reordenando: x^2 - 1 - e^(1-x^2) = 0

def metodo_biseccion(funcion, inicio, fin, tolerancia=1e-3):
    if funcion(inicio) * funcion(fin) >= 0:
        print("La función no cambia de signo en el intervalo dado.")
        return None

    while abs(fin - inicio) >= tolerancia:
        punto_medio = (inicio + fin) / 2
        valor_medio = funcion(punto_medio)

        if abs(valor_medio) < tolerancia:
            return punto_medio
        if funcion(inicio) * valor_medio < 0:
            fin = punto_medio
        else:
            inicio = punto_medio
```

```
    return (inicio + fin) / 2

# Buscamos la raíz en el intervalo [-2, 0]
inicio, fin = -2, 0
raiz = metodo_biseccion(f, inicio, fin, tolerancia=1e-3)
print(f"La raíz en el intervalo [-2, 0] es aproximadamente: {raiz}")
```

```
La raíz en el intervalo [-2, 0] es aproximadamente: -1.251953125
```

0.0.4 Ejercicio 5

5.1. Sea $f(x) = (x+3)(x+1)^2 x(x-1)^3(x-3)$. ¿En qué cero de f converge el método de bisección cuando
se aplica en los siguientes intervalos a. $[-1.5,\ 2.5]$

[51]:
```
import numpy as np

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

def metodo_biseccion(funcion, inicio, fin, tolerancia=1e-5):
    if funcion(inicio) * funcion(fin) >= 0:
        print("La función no cambia de signo en el intervalo dado.")
        return None

    iteraciones = 0
    while abs(fin - inicio) >= tolerancia:
        punto_medio = (inicio + fin) / 2
        valor_medio = funcion(punto_medio)

        if abs(valor_medio) < tolerancia:
            return punto_medio
        if funcion(inicio) * valor_medio < 0:
            fin = punto_medio
        else:
            inicio = punto_medio
        iteraciones += 1

    return (inicio + fin) / 2

# Intervalo a) [-1.5, 2.5]
inicio, fin = -1.5, 2.5
raiz = metodo_biseccion(f, inicio, fin)
print(f"La raíz en el intervalo [-1.5, 2.5] es aproximadamente: {raiz}")
```

```
La función no cambia de signo en el intervalo dado.
La raíz en el intervalo [-1.5, 2.5] es aproximadamente: None
```

b. $[-0.5,\ 2.4]$

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

def metodo_biseccion(funcion, inicio, fin, tolerancia=1e-5):
    if funcion(inicio) * funcion(fin) >= 0:
        print("La función no cambia de signo en el intervalo dado.")
        return None

    while abs(fin - inicio) >= tolerancia:
        punto_medio = (inicio + fin) / 2
        valor_medio = funcion(punto_medio)

        if abs(valor_medio) < tolerancia:
            return punto_medio
        if funcion(inicio) * valor_medio < 0:
            fin = punto_medio
        else:
            inicio = punto_medio

    return (inicio + fin) / 2

# Intervalo b) [-0.5, 2.4]
inicio, fin = -0.5, 2.4
raiz = metodo_biseccion(f, inicio, fin)
print(f"La raíz en el intervalo [-0.5, 2.4] es aproximadamente: {raiz}")
```

c. $[-0.5,\ 3]$

[55]:
```python
def f(x):
    return (x + 3)*(x + 1)**2 * x * (x - 1)**3 * (x - 3)

def metodo_biseccion(funcion, inicio, fin, tolerancia=1e-5):
    if funcion(inicio) * funcion(fin) >= 0:
        print("La función no cambia de signo en el intervalo dado.")
        return None

    while abs(fin - inicio) >= tolerancia:
        punto_medio = (inicio + fin) / 2
        valor_medio = funcion(punto_medio)

        if abs(valor_medio) < tolerancia:
            return punto_medio
        if funcion(inicio) * valor_medio < 0:
            fin = punto_medio
        else:
            inicio = punto_medio
```

```
    return (inicio + fin) / 2

# Intervalo c) [-0.5, 3]
inicio, fin = -0.5, 3
raiz = metodo_biseccion(f, inicio, fin)
print(f"La raíz en el intervalo [-0.5, 3] es aproximadamente: {raiz}")
```

```
La función no cambia de signo en el intervalo dado.
La raíz en el intervalo [-0.5, 3] es aproximadamente: None
```

d. $[-3, \ -0.5]$

[59]:
```
def f(x):
    return (x + 3)*(x + 1)**2 * x * (x - 1)**3 * (x - 3)

def metodo_biseccion(funcion, inicio, fin, tolerancia=1e-5):
    if funcion(inicio) * funcion(fin) >= 0:
        print("La función no cambia de signo en el intervalo dado.")
        return None

    while abs(fin - inicio) >= tolerancia:
        punto_medio = (inicio + fin) / 2
        valor_medio = funcion(punto_medio)

        if abs(valor_medio) < tolerancia:
            return punto_medio
        if funcion(inicio) * valor_medio < 0:
            fin = punto_medio
        else:
            inicio = punto_medio

    return (inicio + fin) / 2

# Intervalo d) [-3, -0.5]
inicio, fin = -3, -0.5
raiz = metodo_biseccion(f, inicio, fin)
print(f"La raíz en el intervalo [-3, -0.5] es aproximadamente: {raiz}")
```

```
La función no cambia de signo en el intervalo dado.
La raíz en el intervalo [-3, -0.5] es aproximadamente: None
```

0.1 Ejercicios Aplicados

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

del agua es:

$$V = L\left[\frac{\pi r^2}{2} - \underbrace{\arcsin\left(\frac{h}{r}\right)}_{\text{ángulo}} - \underbrace{h\sqrt{r^2-h^2}}_{\text{segmento triangular}}\right]$$

Suponga que = 10 , = 1 y = 12.4 Encuentre la profundidad del agua en el abrevadero dentro de 0.01 .

[60]:
```
import numpy as np

def bisection_method(f, a, b, tol=1e-5):
    while abs(b-a) > tol:
        c = (a+b)/2
        if f(c) == 0.0:
            return c
        elif f(a)*f(c) < 0:
            b = c
        else:
            a = c
    return (a+b)/2

def V(h, L=10, r=1):
    return L * (0.5 * np.pi * r**2 - r**2 * np.arcsin(h/r) - h * np.sqrt(r**2 -
 ↪h**2)) - 12.4

a, b = 0, 1  # Initial interval
root = bisection_method(V, a, b, tol=0.01)
print(f"La profundidad del agua es: {root} cm")
```

```
La profundidad del agua es: 0.16796875 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

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

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.

[61]:
```
import numpy as np
altura_inicial = 300
masa = 0.25
gravedad = 9.81
constante_resistencia = 0.1
def altura(t):
    return (
```

```
        altura_inicial
        - (masa * gravedad / constante_resistencia) * t
        + (masa**2 * gravedad / constante_resistencia**2) * (1 - np.
  ↪exp(-constante_resistencia * t / masa))
    )
limite_inferior, limite_superior = 0, 100
tolerancia = 0.01
tiempo_impacto = metodo_biseccion(altura, limite_inferior, limite_superior,␣
  ↪tolerancia)
print(f"El tiempo que tarda en golpear el piso es aproximadamente:␣
  ↪{tiempo_impacto} segundos")
```

```
El tiempo que tarda en golpear el piso es aproximadamente: 14.7247314453125
segundos
```

0.2 Ejercicios teóricos

1. Use el teorrema 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

[62]:
```
import numpy as np
def funcion(x):
    return x**3 - x - 1
limite_inferior, limite_superior = 1, 2
tolerancia = 1e-4
iteraciones_minimas = int(np.ceil((np.log(limite_superior - limite_inferior) -␣
  ↪np.log(tolerancia)) / np.log(2)))
print(f"Iteraciones mínimas necesarias según el Teorema 2.1:␣
  ↪{iteraciones_minimas}")
aproximacion_raiz = metodo_biseccion(funcion, limite_inferior, limite_superior)
print(f"La aproximación de la raíz es: {aproximacion_raiz}")
```

```
Iteraciones mínimas necesarias según el Teorema 2.1: 14
La aproximación de la raíz es: 1.3247184753417969
```

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

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