Taller N°2

Tamara Benavidez

Tabla de Contenidos

ESCUELA POLITÉCNICA NACIONAL

Taller 02: Cálculo de raíces

Ejercicio 1

Encuentre todas las raíces del polinomio x4+540∗x3+109124∗x2+9781632∗x+328188672=0

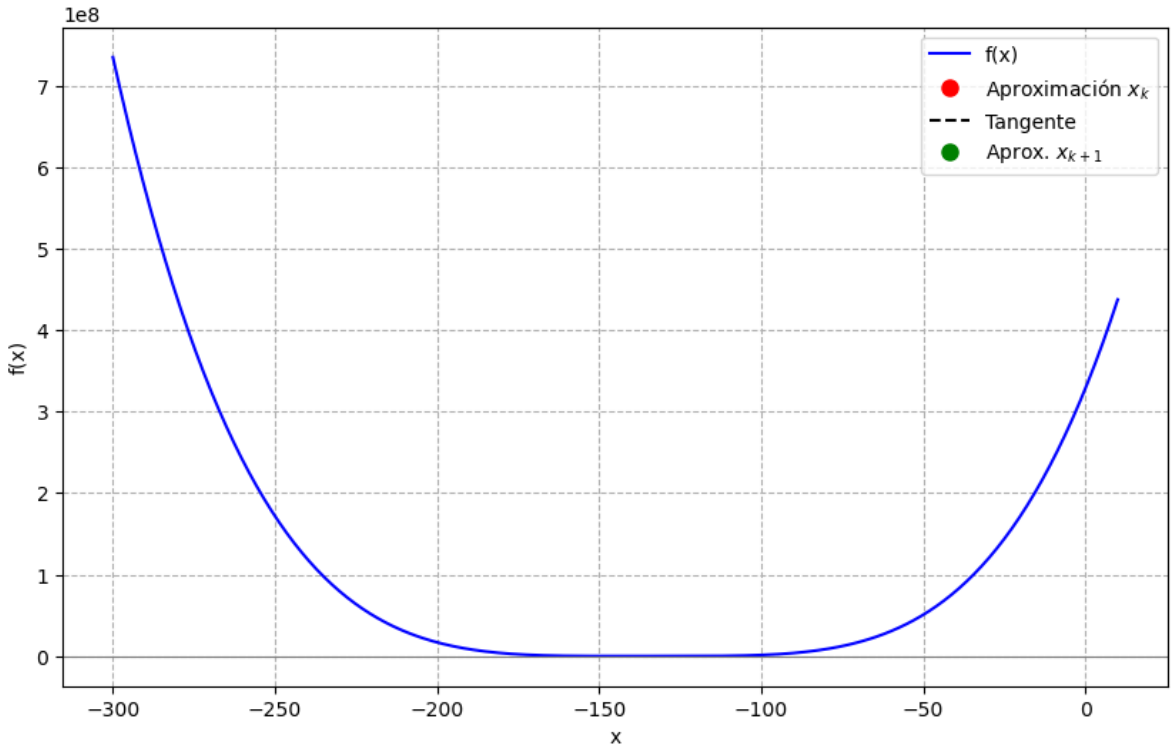
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
```

```
def f(x):
    """Polinomio f(x)"""
    return x**4 + 540.*x**3 + 109124.*x**2 + 9781632.*x + 328188672.

def df(x):
    """Derivada df(x)"""
    return 4.*x**3 + 1620.*x**2 + 218248.*x + 9781632.

def newton(f, df, x0, tol=1e-6, max_iter=100):
    steps = [x0]
    x = x0
    for i in range(max_iter):
        fx, fpx = f(x), df(x)
        if abs(fx) < tol:
            print(f"Convergió en {i+1} iteraciones. Raíz: {x:.6f}")
            break
        if fpx == 0:
            print("Derivada cero. Fallo.")
            break
        x = x - fx / fpx
        steps.append(x)
    return steps

x0 = 0.75
x_steps = newton(f, df, x0) # Llamada a la función 'newton'

x_plot = np.linspace(-300, 10, 400)
y_plot = f(x_plot)
```

```
Convergió en 27 iteraciones. Raíz: -125.999943
```

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(x_plot, y_plot, label="f(x)", color='blue')
ax.axhline(0, color='grey', lw=0.8)
ax.grid(True, linestyle='--')
ax.set_xlabel("x")
ax.set_ylabel("f(x)")

point, = ax.plot([], [], 'ro', markersize=8, label="Aproximación $x_k$")
tangent_line, = ax.plot([], [], 'k--', lw=1.5, label="Tangente")
next_x_intercept, = ax.plot([], [], 'go', markersize=8, label="Aprox. $x_{k+1}$")
```

```
ax.legend()

def init():
    point.set_data([], [])
    tangent_line.set_data([], [])
    next_x_intercept.set_data([], [])
    return point, tangent_line, next_x_intercept

def update(frame):
    x_k = x_steps[frame]
    y_k = f(x_k)
    point.set_data([x_k], [y_k])

    if frame + 1 < len(x_steps):
        x_k_plus_1 = x_steps[frame + 1]
        tangent_x = [x_k, x_k_plus_1]
        tangent_y = [y_k, 0]
        next_x_intercept.set_data([x_k_plus_1], [0])
    else:
        fpx = df(x_k)
        tangent_x = np.linspace(x_k - 1, x_k + 1, 2)
        tangent_y = y_k + fpx * (tangent_x - x_k)
        next_x_intercept.set_data([], [])

    tangent_line.set_data(tangent_x, tangent_y)
    ax.set_title(f"Iteración {frame}: $x_{frame}$ = {x_k:.6f}")
    return point, tangent_line, next_x_intercept

FuncAnimation(
    fig, update, frames=len(x_steps), init_func=init, blit=True, interval=750, repeat=False
)
plt.show()
```

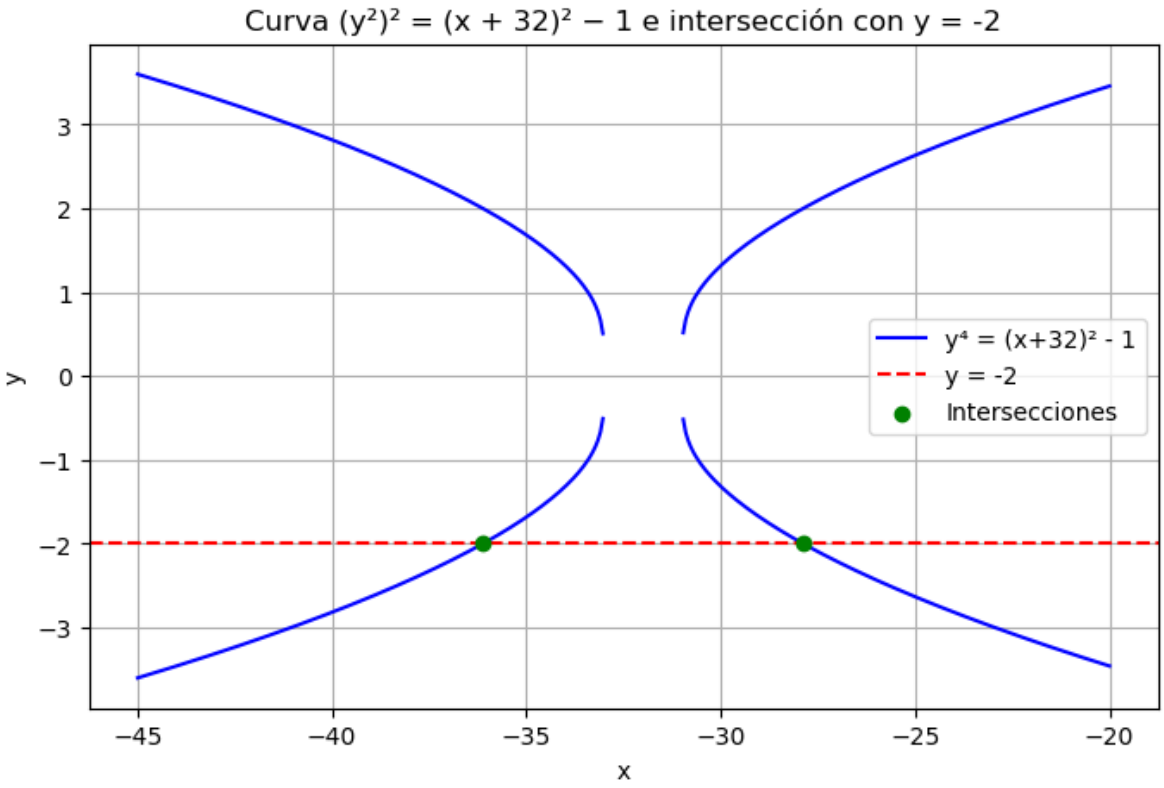


Ejercicio 2

Encuentre todos los puntos en los que la curva (y2)2=(x+32)2−1 interseca el eje y=−2

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

# Ecuación de la curva: (y²)² = (x + 32)² - 1     y  = (x + 32)² - 1
def y_curva(x):
    valor = (x + 32)**2 - 1
    valor[valor < 0] = np.nan  # Evitar valores negativos dentro de la raíz cuarta
    return np.power(valor, 1/4)

# Crear valores de x
x = np.linspace(-45, -20, 400)

# Calcular los valores positivos y negativos de y
y_pos = y_curva(x)
y_neg = -y_pos
```

```
# Gráfico
plt.figure(figsize=(8, 5))
plt.plot(x, y_pos, label='y  = (x+32)² - 1', color='blue')
plt.plot(x, y_neg, color='blue')
plt.axhline(-2, color='red', linestyle='--', label='y = -2')

# Calcular intersecciones exactas (de y=-2)
# Sustituimos en la ecuación: (-2)^4 = (x+32)^2 - 1  → 16 = (x+32)^2 - 1  → (x+32)^2 = 17
x1 = -32 + np.sqrt(17)
x2 = -32 - np.sqrt(17)
plt.scatter([x1, x2], [-2, -2], color='green', zorder=5, label='Intersecciones')

plt.title('Curva (y²)² = (x + 32)² - 1 e intersección con y = -2')

# Detalles del gráfico
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()

print(f"Las intersecciones son aproximadamente:\n x  = {x1:.4f}, y = -2\n x  = {x2:.4f}, y =
```

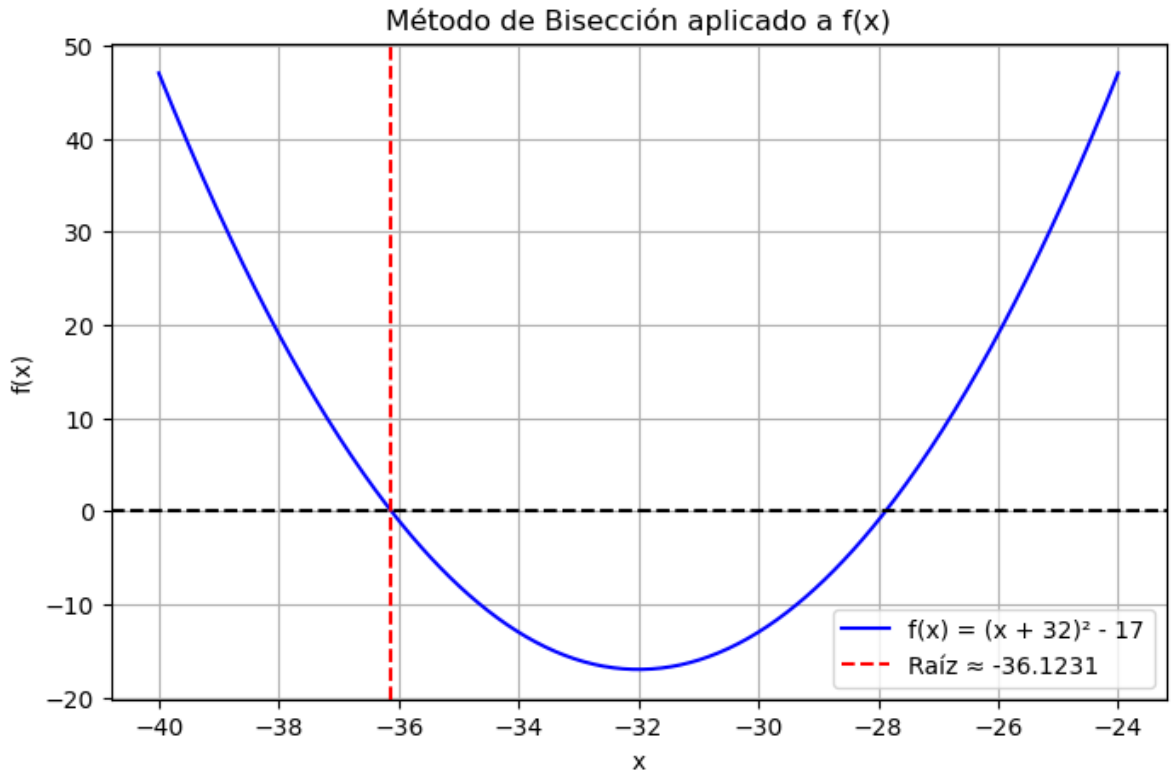



```
Las intersecciones son aproximadamente:
 x  = -27.8769, y = -2
 x  = -36.1231, y = -2
```

```python
def f(x):
    return (x + 32)**2 - 17  # Ecuación equivalente a (y²)² = (x + 32)² - 1, con y = -2

def biseccion(f, a, b, tol=1e-6, max_iter=50):
    pasos = []
    for _ in range(max_iter):
        c = (a + b) / 2
        pasos.append((a, b, c))
        if abs(f(c)) < tol:
            break
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c, pasos
```

```
# Intervalo inicial
a, b = -37, -27
raiz, pasos = biseccion(f, a, b)

print(f" Raíz aproximada (método de Bisección): {raiz:.6f}")

x_vals = np.linspace(-40, -24, 400)
y_vals = f(x_vals)

plt.figure(figsize=(8,5))
plt.plot(x_vals, y_vals, label='f(x) = (x + 32)² - 17', color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(raiz, color='red', linestyle='--', label=f'Raíz  {raiz:.4f}')
plt.title('Método de Bisección aplicado a f(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)
plt.legend()
plt.show()
```

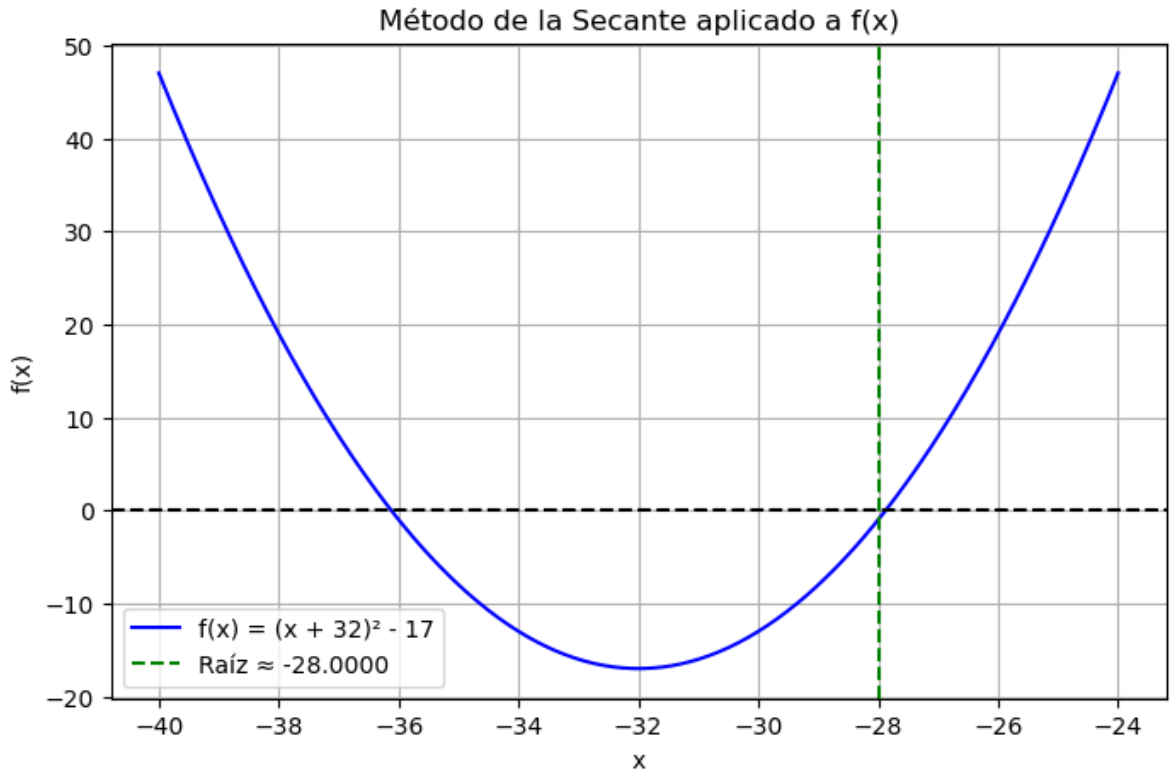
```
 Raíz aproximada (método de Bisección): -36.123106
```




```python
def f(x):
    return (x + 32)**2 - 17

def secante(f, x0, x1, tol=1e-6, max_iter=50):
    pasos = [(x0, x1)]
    x2 = x1  # inicializamos
    for _ in range(max_iter):
        f0, f1 = f(x0), f(x1)
        if f1 - f0 == 0:  # evitar división por cero
            print("División por cero detectada, se detiene la iteración.")
            break
        x2 = x1 - f1 * (x1 - x0) / (f1 - f0)
        pasos.append((x1, x2))
        if abs(x2 - x1) < tol:
            break
        x0, x1 = x1, x2
    return x2, pasos

x0, x1 = -36, -28
```

```
raiz, pasos = secante(f, x0, x1)
print(f" Raíz aproximada (método de la Secante): {raiz:.6f}")

x_vals = np.linspace(-40, -24, 400)
y_vals = f(x_vals)

plt.figure(figsize=(8,5))
plt.plot(x_vals, y_vals, label='f(x) = (x + 32)² - 17', color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(raiz, color='green', linestyle='--', label=f'Raíz   {raiz:.4f}')
plt.title('Método de la Secante aplicado a f(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)
plt.legend()
plt.show()
```

```
División por cero detectada, se detiene la iteración.
 Raíz aproximada (método de la Secante): -28.000000
```

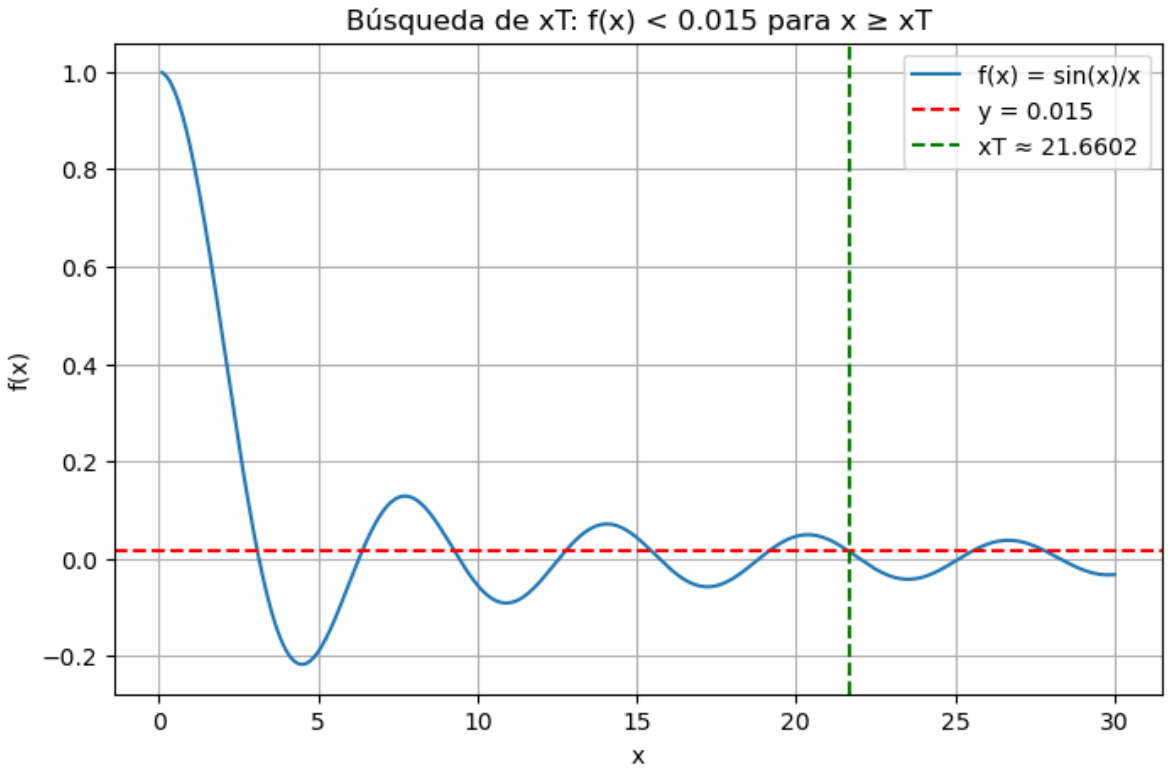


Ejercicio 3

Dada la función f(x)=sin(x)x. ¿A partir de qué valor xT se cumple que f(x)<0.015, x xT?

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import newton, bisect

def f(x):
    return np.sin(x)/x - 0.015

def df(x):
    return (x*np.cos(x) - np.sin(x))/x**2

a, b = 20, 25  # f(x) > 0 al inicio, f(x) < 0 al final
raiz_biseccion = bisect(f, a, b, xtol=1e-6)
print(f" xT por Biseccion  {raiz_biseccion:.6f}")

x0 = 22  # aproximacion inicial
raiz_newton = newton(f, x0, fprime=df, tol=1e-6)
print(f" xT por Newton  {raiz_newton:.6f}")

x_vals = np.linspace(0.1, 30, 1000)  # evitamos x=0
y_vals = np.sin(x_vals)/x_vals

plt.figure(figsize=(8,5))
plt.plot(x_vals, y_vals, label='f(x) = sin(x)/x')
plt.axhline(0.015, color='red', linestyle='--', label='y = 0.015')
plt.axvline(raiz_biseccion, color='green', linestyle='--', label=f'xT  {raiz_biseccion:.4f}'
plt.title('Busqueda de xT: f(x) < 0.015 para x  xT')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.grid(True)
plt.legend()
plt.show()
```

```
 xT por Biseccion  21.660239
 xT por Newton  21.660239
```



Link del repositorio GitHub

Mi repositorio del Taller 2