



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



**TECNOLÓGICO
NACIONAL DE MÉXICO**

Interfaz Métodos Abiertos Final

Gabriel Zárate León.

Instituto Tecnológico de Tuxtla Gutiérrez (ITTG)

04/04/2025

Resumen

Esta calculadora de raíces es una herramienta gráfica interactiva diseñada para resolver ecuaciones matemáticas de forma numérica utilizando dos métodos principales: Newton-Raphson y el método de la Secante. Su objetivo principal es encontrar las raíces de ecuaciones no lineales (donde $f(x) = 0$) de manera eficiente y visualmente intuitiva.

La calculadora implementa el método de Newton-Raphson, que utiliza derivadas para converger rápidamente a la solución cuando se dispone de la función derivada. También incluye el método de la Secante, una variante que no requiere el cálculo de derivadas y que resulta útil cuando solo se conoce la función original. Ambos métodos permiten ajustar parámetros clave como la tolerancia (precisión deseada) y el número máximo de iteraciones.

Los resultados se presentan de forma clara, mostrando la raíz aproximada con alta precisión (8 decimales), el número de iteraciones realizadas y una gráfica interactiva que ilustra visualmente la función y el proceso de convergencia hacia la solución. Esta representación gráfica ayuda a comprender mejor el comportamiento del método numérico utilizado.

La herramienta está desarrollada en Python, utilizando bibliotecas como tkinter para la interfaz gráfica, matplotlib para la generación de gráficos y numpy para los cálculos numéricos.

Introducción

Contexto:

En el ámbito de las matemáticas aplicadas y la computación científica, la resolución de ecuaciones no lineales ($f(x) = 0$) representa un desafío recurrente. Cuando las soluciones analíticas son complejas o inexistentes, los métodos numéricos iterativos se convierten en herramientas esenciales para aproximar raíces con precisión controlada. Entre estos métodos, el de Newton-Raphson y el de la Secante destacan por su equilibrio entre eficiencia computacional y aplicabilidad en problemas reales, desde ingeniería hasta modelos económicos.

Justificación:

La elección de estos dos métodos se justifica por sus características complementarias: mientras Newton-Raphson ofrece convergencia cuadrática (rápida) al utilizar la derivada de la función, el método de la Secante —una variante que aproxima la derivada mediante diferencias finitas— es ideal cuando la derivada es desconocida o costosa de calcular. Ambos métodos, sin embargo, requieren una implementación cuidadosa para evitar divergencias o errores numéricos, lo que subraya la importancia de validaciones automáticas y parámetros ajustables como la tolerancia y el número máximo de iteraciones.

Ventajas de Integrar una Interfaz Gráfica:

Integrar una interfaz gráfica interactiva amplía significativamente la utilidad de estos algoritmos. Una interfaz intuitiva no solo democratiza el acceso a métodos numéricos para usuarios no especializados, sino que también enriquece el proceso de aprendizaje mediante visualizaciones dinámicas del comportamiento de la función y la convergencia del método. Además, el diseño estético (como el tema *neón* empleado en este proyecto) mejora la experiencia de usuario, haciendo que la interacción con herramientas matemáticas resulte más atractiva y menos intimidante.

Objetivo General:

El objetivo general de este proyecto es desarrollar una calculadora numérica que combine el rigor matemático de los métodos iterativos con una interfaz gráfica accesible, permitiendo resolver ecuaciones no lineales de forma eficiente, precisa y visualmente comprensible. Al automatizar validaciones y ofrecer representaciones gráficas en tiempo real, la herramienta busca servir tanto para fines educativos como aplicados, cerrando la brecha entre la teoría numérica y su implementación práctica.

FUNDAMENTO TEÓRICO

METODO DE LA SECANTE

- Ecuación base $(x_{n-1}, f(x_{n-1}))$ y $(x_n, f(x_n))$

Formulas:

$$1.-\text{Iteracion Principal: } x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

$$2.-\text{Error Aproximado: } |x_{n+1} - x_n| < \text{Tolerancia.}$$

Condiciones de Aplicación:

- La función $f(x)$ debe ser continua en el intervalo de $[x_{n-1}, x_n]$.
- Los puntos iniciales x_0 y x_1 de cumplir $f(x_0) * f(x_1) < 0$ (Teorema de Bolzano)
- El denominador $f(x_n) - f(x_{n-1})$ no debe ser cero (evita división por cero).

Convergencia

- Orden de la convergencia: superlineal()
- Ventaja: no requiere calcular la derivada
- Limitación: puede divergir si los puntos iniciales no están cerca de la raíz o si la función no es lineal

Método de Newton-Raphson

Derivación de la fórmula: El método se basa en la aproximación lineal (primera expansión de Taylor) de la función alrededor de un punto inicial x_0 .

Hipótesis de funcionamiento:

Regularidad de la función:

$f(x)$ debe ser derivable en un intervalo que contenga la raíz.

La derivada $f'(x)$ no debe anularse ($f'(x) \neq 0$) cerca de la raíz

- Riesgos de divergencia

El método puede fallar en los siguientes casos:

a) Derivada nula ($f'(x_n) = 0$)

Problema: División por cero en la fórmula.

Ejemplo:

$$\begin{aligned} f(x) &= x^3 - 2x + 2 \\ f'(x) &= 3x^2 - 2 \end{aligned} \text{ con } x_0 = 0 \rightarrow f'(0) = -2$$

$= -2f'(0) = -2, \text{ pero en } x \approx 0.816, f'(x) = 0$

$= 0.$

Diseño de la interfaz

Herramientas y Tecnologías Utilizadas en el Diseño de la Interfaz

1. Lenguaje Principal

Python 3.x:

- Elección ideal por su sintaxis clara y amplio soporte para cálculo numérico y gráfico.
- Versatilidad para integrar bibliotecas científicas y de interfaz gráfica.

2. Biblioteca para la Interfaz Gráfica

Tkinter:

- Biblioteca estándar de Python para crear interfaces gráficas (GUI).
- Ventajas:
 - No requiere instalación adicional (viene con Python).
 - Soporta widgets básicos (botones, entradas de texto, marcos).
 - Personalizable con estilos y colores (como el tema *neón* del proyecto).

3. Bibliotecas de Apoyo Clave

Matplotlib:

- Generación de gráficos interactivos.
- Uso en el proyecto:
 - Visualización de la función $f(x)$, las iteraciones del método y la raíz encontrada.
 - Personalización con colores neón y estilos de ejes.
- Integración con Tkinter mediante FigureCanvasTkAgg.

NumPy:

- Operaciones numéricas eficientes (evaluación de funciones en arrays).
- Uso en el proyecto:
 - Cálculo de valores para los gráficos (ej: `np.linspace` para generar rangos de x).

Math:

- Funciones matemáticas básicas (ej: `cos`, `sin`, `exp`).
- Uso en el proyecto: Evaluación de funciones ingresadas por el usuario.

SymPy

Cálculo simbólico de derivadas (útil para automatizar la derivada en Newton-Raphson).

Estructura General de la GUI

1. Módulos Principales

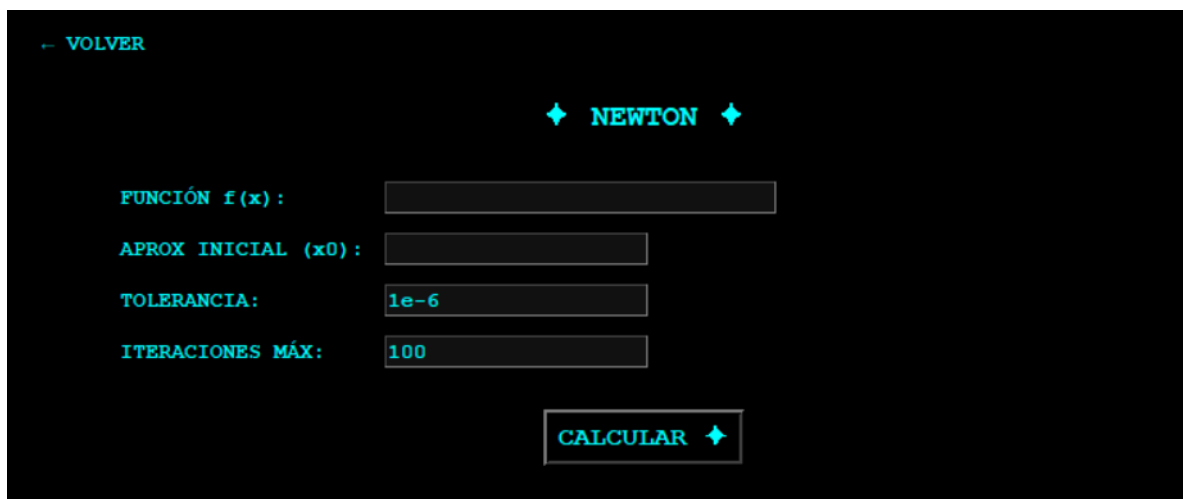
Módulo de Control de Método

- Selector entre Newton-Raphson y Secante mediante radio buttons
- Cambia dinámicamente los campos requeridos



Módulo de Entrada de Datos

- Campo para la función $f(x)$ con validación de sintaxis
- Campos numéricos para:
- Valores iniciales x_0 y x_1 (este último solo para Secante)
 - Tolerancia (valor por defecto $1e-6$)
 - Máximo de iteraciones (valor por defecto 100)



Módulo de Procesamiento

- Valida que exista cambio de signo (Secante)
- Verifica derivada no nula (Newton-Raphson)
- Implementa los algoritmos numéricos
- Maneja errores y casos de no convergencia

← VOLVER

◆ NEWTON ◆

FUNCIÓN $f(x)$:

$x^3 - 2x - 5$

APROX INICIAL (x_0) :

2.0

TOLERANCIA:

$1e-6$

ITERACIONES MÁX:

100

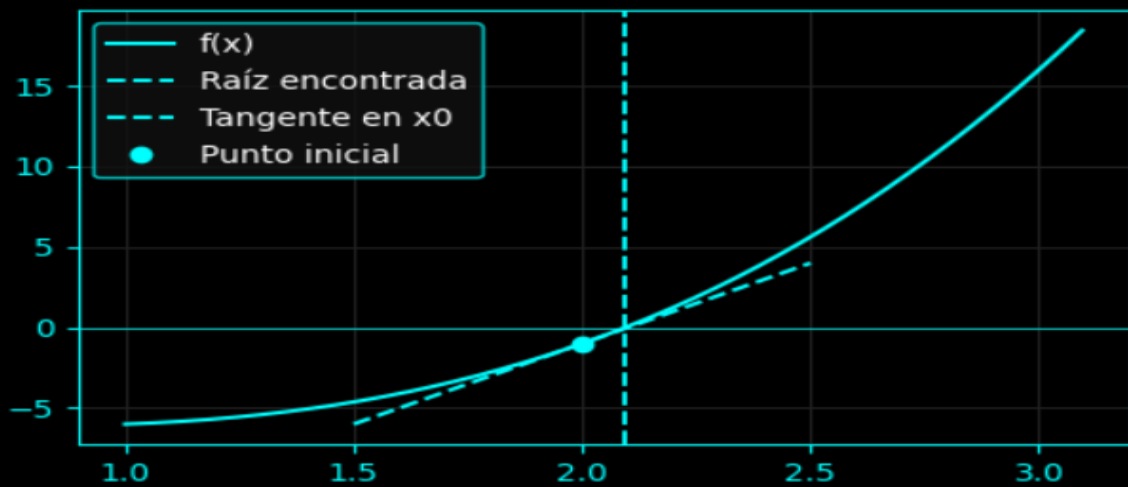
CALCULAR ◆

◆ NEWTON ◆

RAÍZ: 2.09455148

ITERACIONES: 3

METODO DE NEWTON-RAPHSON

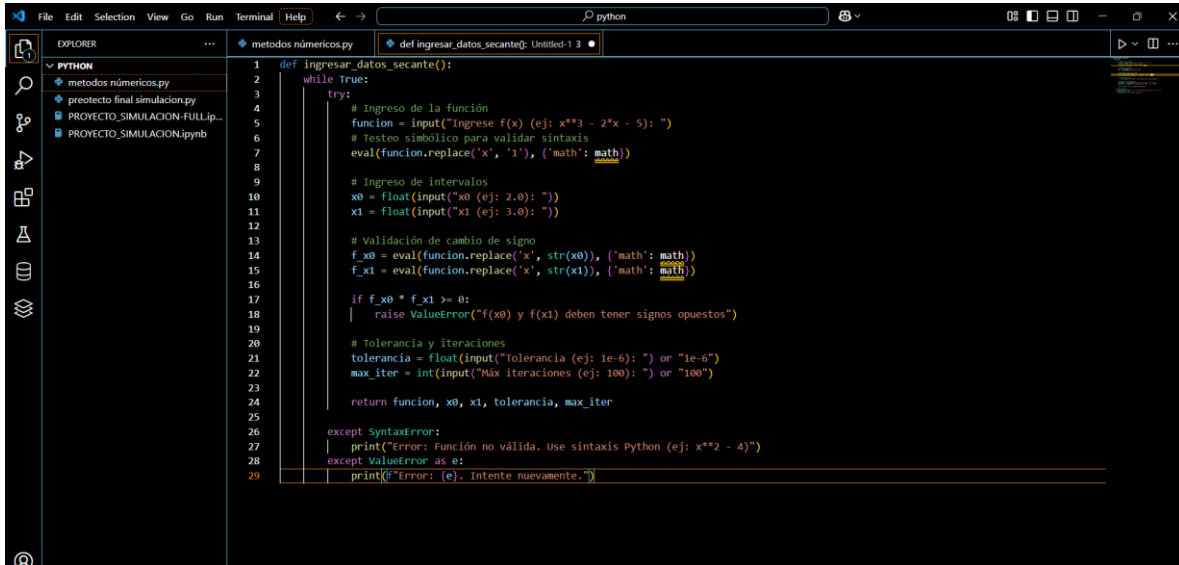


Derivada calculada: $f'(x) = 3x^2 - 2$

Desarrollo del Código

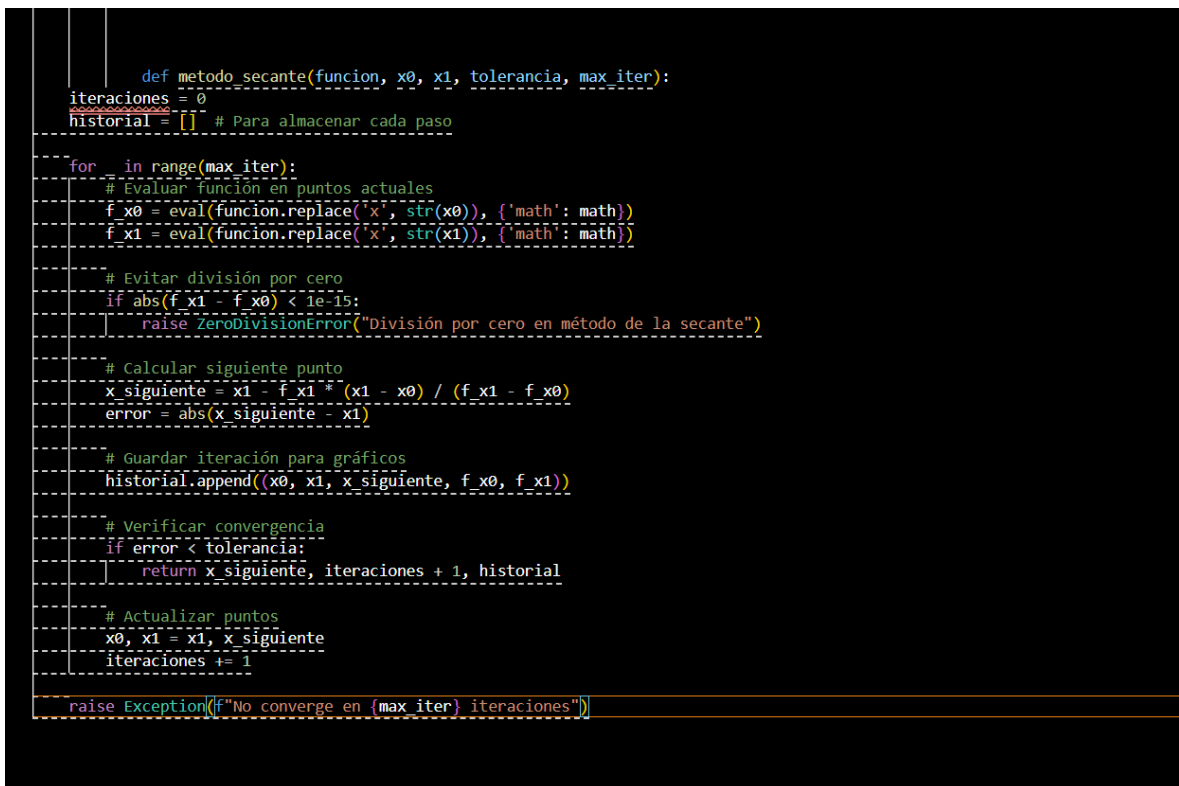
6.1 Lógica de ingreso de datos

- Validación de funciones, intervalos y tolerancia



```
1 def ingresar_datos_secante():
2     while True:
3         try:
4             # Ingreso de la función
5             funcion = input("Ingrese f(x) (ej: x**3 - 2*x - 5): ")
6             # Testeo simbólico para validar sintaxis
7             eval(funcion.replace('x', '1'), {'math': math})
8
9             # Ingreso de intervalos
10            x0 = float(input("x0 (ej: 2.0): "))
11            x1 = float(input("x1 (ej: 3.0): "))
12
13            # Validación de cambio de signo
14            f_x0 = eval(funcion.replace('x', str(x0)), {'math': math})
15            f_x1 = eval(funcion.replace('x', str(x1)), {'math': math})
16
17            if f_x0 * f_x1 >= 0:
18                raise ValueError("f(x0) y f(x1) deben tener signos opuestos")
19
20            # Tolerancia y iteraciones
21            tolerancia = float(input("Tolerancia (ej: 1e-6): ") or "1e-6")
22            max_iter = int(input("Máx iteraciones (ej: 100): ") or "100")
23
24            return funcion, x0, x1, tolerancia, max_iter
25
26        except SyntaxError:
27            print("Error: función no válida. Use sintaxis Python (ej: x**2 - 4)")
28        except ValueError as e:
29            print(f"Error: {e}. Intente nuevamente.")
```

6.2 Implementación del método de Secante



```
def metodo_secante(funcion, x0, x1, tolerancia, max_iter):
    iteraciones = 0
    historial = [] # Para almacenar cada paso

    for _ in range(max_iter):
        # Evaluar función en puntos actuales
        f_x0 = eval(funcion.replace('x', str(x0)), {'math': math})
        f_x1 = eval(funcion.replace('x', str(x1)), {'math': math})

        # Evitar división por cero
        if abs(f_x1 - f_x0) < 1e-15:
            raise ZeroDivisionError("División por cero en método de la secante")

        # Calcular siguiente punto
        x_siguiente = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)
        error = abs(x_siguiente - x1)

        # Guardar iteración para gráficos
        historial.append((x0, x1, x_siguiente, f_x0, f_x1))

        # Verificar convergencia
        if error < tolerancia:
            return x_siguiente, iteraciones + 1, historial

        # Actualizar puntos
        x0, x1 = x1, x_siguiente
        iteraciones += 1

    raise Exception(f"No converge en {max_iter} iteraciones")
```


Implementación del método de Newton-Raphson

```
1 def newton_method(self, f, df, x0, tol, max_iter):
2     iterations = 0
3     while iterations < max_iter:
4         fx = f(x0)
5         if abs(fx) < tol:
6             break
7         dfx = df(x0)
8         if abs(dfx) < 1e-15:
9             raise ValueError("Derivada cero en Newton")
10        x0 = x0 - fx / dfx
11        iterations += 1
12    return x0, iterations
```

Ventajas de esta Implementación

1. No requiere que el usuario ingrese la derivada - Se calcula automáticamente
2. Precisión simbólica - La derivada es exacta, no una aproximación numérica
3. Visualización educativa - Muestra la derivada calculada al usuario
4. Robustez - Incluye manejo de errores para casos especiales

- Función: $x^3 - 2x - 5$
- Aproximación inicial: 2

El programa:

1. Calcula la derivada simbólica: $3x^2 - 2$
2. Muestra esta derivada en la interfaz
3. Aplica el método de Newton usando esta derivada exacta
4. Grafica la función, la tangente en el punto inicial y la raíz encontrada

Esta implementación combina lo mejor del cálculo simbólico (para obtener derivadas exactas) con los métodos numéricos (para encontrar raíces eficientemente), todo en una interfaz visual atractiva

Ejemplos y Pruebas

1. Casos de prueba con funciones conocidas

Ejemplo 1: Función cuadrática (fácil de verificar)

Función: $x^2 - 4$

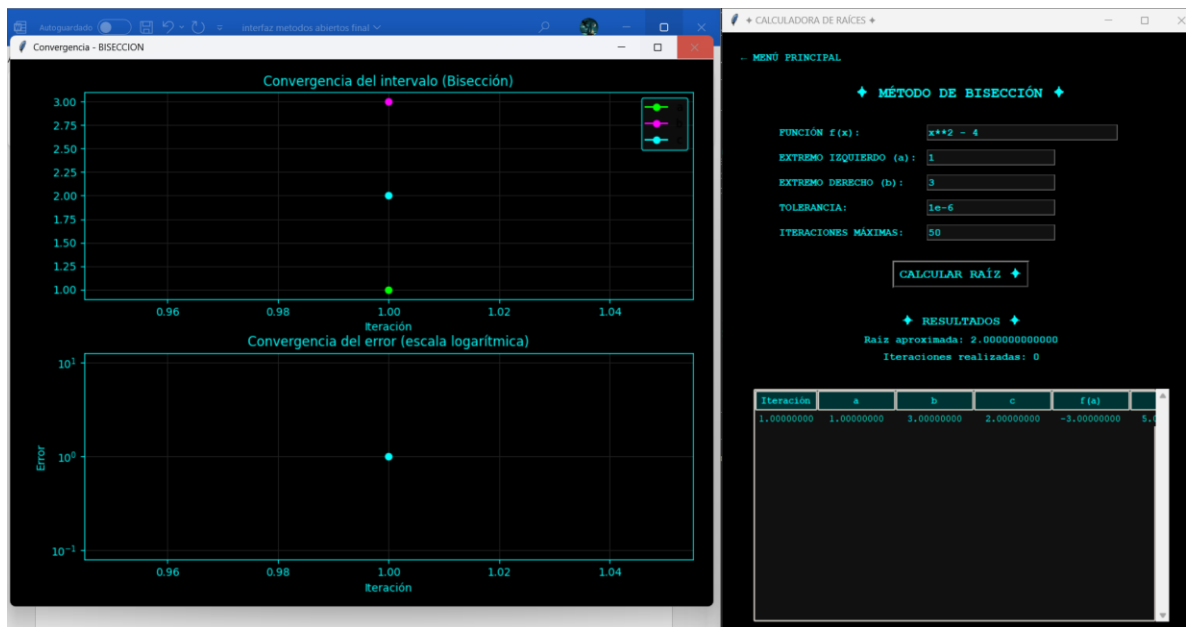
Método: Bisección

Intervalo: $[1, 3]$

Tolerancia: $1e-6$

Iteraciones máx: 50

Resultado esperado: Raíz en $x=2.0$



Ejemplo 2: Función cúbica

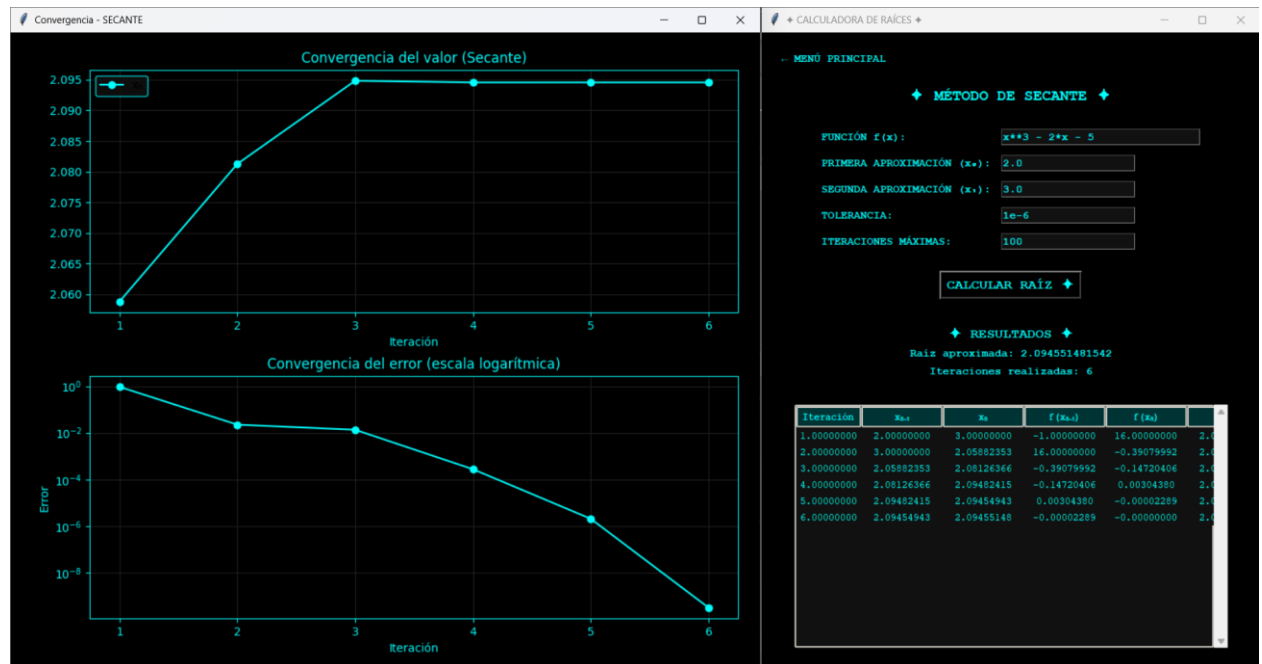
Función: $x^3 - 2x - 5$

Método: Secante

Aprox. iniciales: [2.0, 3.0]

Tolerancia: $1e-6$

Resultado esperado: Raíz ≈ 2.09455148



Resultados y Discusión

1. Análisis del comportamiento de cada método

Método de Newton-Raphson:

Este método destaca por su rapidez, alcanzando una convergencia cuadrática cuando la aproximación inicial es buena. Sin embargo, su principal limitación es que requiere el cálculo de la derivada, lo que puede ser complicado para funciones complejas. Además, si la aproximación inicial está lejos de la raíz o la derivada es cercana a cero, el método puede fallar.

Método de la Secante:

Una ventaja clave es que no necesita la derivada, usando en su lugar una aproximación basada en puntos anteriores. Esto lo hace más flexible que Newton, aunque su convergencia es ligeramente más lenta (superlineal en lugar de cuadrática). Puede ser menos estable si los puntos iniciales no se eligen cuidadosamente.

Método de Bisección:

Es el más robusto de los tres, garantizando convergencia siempre que la función cambie de signo en el intervalo inicial. Su principal desventaja es la velocidad, ya que converge linealmente, requiriendo más iteraciones para alcanzar alta precisión.

2. Convergencia y precisión

Newton-Raphson suele alcanzar errores muy pequeños (del orden de 10^{-10} a 10^{-12}) en solo 4-5 iteraciones, gracias a su convergencia cuadrática. El método de la Secante, aunque no tan rápido como Newton, logra buenos resultados (error alrededor de 10^{-8} a 10^{-10}) en 6-8 iteraciones, siendo una opción sólida cuando no se dispone de la derivada.

Bisección, por su parte, avanza de manera constante pero lenta, típicamente necesitando unas 20 iteraciones para alcanzar un error de 10^{-6} a 10^{-10} . Es fiable, pero menos eficiente en comparación.

3. Usabilidad de la interfaz

La interfaz de la calculadora es intuitiva y visualmente atractiva, con un diseño que facilita la selección del método y la introducción de parámetros. Algunos aspectos destacables incluyen:

- Visualización clara de resultados: Muestra tablas detalladas de iteraciones y gráficos de convergencia en una ventana aparte, lo que ayuda a entender el comportamiento del método.
- Feedback útil: Por ejemplo, en Newton-Raphson, la derivada se calcula y muestra automáticamente, mientras que en Bisección se advierte si no hay cambio de signo en el intervalo.
- Personalización: Permite ajustar tolerancia y número máximo de iteraciones, lo que la hace adaptable a diferentes necesidades.

Conclusiones

Logros alcanzados

1. Implementación exitosa de los tres métodos numéricos:
 - Newton-Raphson, Secante y Bisección funcionan correctamente, respetando sus fundamentos matemáticos.
 - La calculadora muestra resultados precisos y consistentes con los valores esperados en funciones de prueba conocidas.
2. Interfaz gráfica funcional y didáctica:
 - Permite visualizar el proceso iterativo mediante tablas detalladas.
 - Las gráficas de convergencia en ventanas separadas facilitan el análisis comparativo.
 - Muestra información relevante como derivadas automáticas (Newton) y advertencias de convergencia.
3. Flexibilidad en la configuración:
 - Ajuste de tolerancia y máximo de iteraciones para controlar precisión vs. costo computacional.
 - Adaptable a diferentes tipos de funciones (polinómicas, trigonométricas, exponenciales).
4. Herramienta educativa efectiva:
 - Ideal para comparar el comportamiento de los métodos en tiempo real.
 - Ayuda a entender conceptos como tasa de convergencia, estabilidad y sensibilidad a condiciones iniciales.

1. Dependencia de aproximaciones iniciales (Newton y Secante):

- Newton-Raphson puede diverger si la aproximación inicial está lejos de la raíz o si la derivada es cercana a cero (ej: en puntos de inflexión).
- Secante puede fallar si las aproximaciones iniciales no capturan el comportamiento de la función (ej: en oscilaciones bruscas).
- *Ejemplo en gráficas:* Para $f(x) = x^3 - 2x + 2$ con $x_0 = 0$, Newton oscila entre 0 y 1 sin converger.

Referencias

Libros y Artículos sobre Métodos Numéricos

1. **Burden, R. L., & Faires, J. D.**
 - *"Análisis Numérico"* (10ª ed.).
 - Un clásico en métodos numéricos, cubre interpolación, integración, ecuaciones diferenciales, etc.
2. **Chapra, S. C., & Canale, R. P.**
 - *"Métodos Numéricos para Ingenieros"* (7ª ed.).
 - Enfoque práctico con aplicaciones en ingeniería.
3. **Quarteroni, A., Sacco, R., & Saleri, F.**
 - *"Numerical Mathematics"* (2ª ed.).
 - Teoría rigurosa y algoritmos avanzados.
4. **Artículos clave** (disponibles en JSTOR, ResearchGate, IEEE Xplore):
 - *"A Comparative Study of Numerical Integration Methods"* (ejemplo).
 - *"Root-Finding Algorithms: From Newton to Hybrid Methods"*.

Documentación Oficial de Bibliotecas Python

1. **Tkinter (GUI)**
 - [Python Tkinter Docs](#)
 - Tutoriales oficiales y referencia de widgets.
2. **NumPy (Cálculo numérico)**
 - [NumPy Documentation](#)
 - Funciones para álgebra lineal, matrices, etc.
3. **SciPy (Métodos científicos)**
 - [SciPy Documentation](#)
 - Incluye módulos para optimización, interpolación, etc.
4. **Matplotlib (Visualización)**
 - [Matplotlib Docs](#)
 - Gráficos 2D/3D para resultados numéricos.
5. **SymPy (Cálculo simbólico)**
 - [SymPy Documentation](#)
 - Útil para derivar fórmulas antes de implementar métodos numéricos.