Métodos numéricos

Interfaz tkinter cálculo de raíces final

S4C

Rafael Antonio Ruiz Limones

Ingeniería en Sistemas Computacionales. Instituto Tecnológico de Tuxtla Gutiérrez

Ing. HORACIO IRAN SOLIS CISNEROS

04/03/2025

Resumen

Este proyecto consiste en una herramienta interactiva con interfaz gráfica que permite aplicar y comparar distintos métodos numéricos en este caso están disponibles 3 métodos ya que en la interfaz anterior que era la del método de bisección se implementaron los otros dos métodos para encontrar raíces de funciones. Su objetivo es ser útil tanto para aprender como para experimentar con estos métodos.

Métodos Incluidos

- **Bisección**: Divide el intervalo en dos.
- **Falsa Posición**: Usa líneas rectas para mejorar la precisión.
- **Newton-Raphson**: Utiliza derivadas para avanzar rápidamente.

Características de la Interfaz

- Ingreso de funciones en xxx
- Configuración de intervalos o valores iniciales
- Ajuste de tolerancia y número máximo de iteraciones
- Vista paso a paso de los cálculos
- Gráficas del proceso y los resultados
- Avisos de error si los datos no son válidos
- Comparación entre métodos para ver cuál es más eficiente

La interfaz es fácil de usar y busca ayudar a entender cómo funciona cada método de forma visual e interactiva.

INTRODUCCION

Cuando tenemos un problema de matemáticas, a veces no podemos resolverlo de manera sencilla, como lo son en las ecuaciones no lineales, donde “x” no solo tiene exponente uno, necesitamos usar métodos numéricos, estos métodos nos ayudan a encontrar soluciones o una aproximación al resultado correcto y dependiendo del método que usemos, podemos encontrar la raíz del problema de manera más rápida o más lenta, pero siempre podemos llegar a la solución.

Métodos Numéricos para Ecuaciones No Lineales

Existen varios métodos numéricos para encontrar la raíz de ecuaciones no lineales, algunos de ellos son los más conocidos son el método de falsa posición, y el método de Newton-Raphson, estos métodos comienzan con una suposición inicial y van ajustándola hasta que se aproximan a la solución correcta.

Método de Falsa Posición: Este método es útil cuando sabemos que en un intervalo específico la función cambia de signo, a diferencia del método de bisección, que divide el intervalo a la mitad, este método busca una aproximación que esté más cerca de la raíz.

Método de Newton-Raphson: Este es más rápido si conocemos la derivada de la función, ya que utiliza una fórmula que permite acercarse rápidamente a la raíz, siempre que tengamos una buena estimación inicial.

¿Por Qué Usar Falsa Posición y Newton-Raphson?

Cabe recalcar que ambos métodos son populares por su efectividad y por los diferentes tipos de problemas en los que se pueden usar.

La Falsa Posición es mejor cuando tenemos un intervalo claro y la función no se puede diferenciar fácilmente, por otro lado, está el método de Newton-Raphson es ideal cuando tenemos la derivada de la función y queremos una respuesta rápida, especialmente si la función se comporta bien cerca de la raíz.

Ventajas de Usar una Interfaz Gráfica

Agregar una interfaz gráfica (GUI) a estos métodos numéricos tiene muchas ventajas. Hace más fácil entender y visualizar cómo los métodos se acercan a la raíz de una ecuación, los usuarios pueden ver de forma interactiva cómo cambian las estimaciones a medida que se realizan los cálculos, lo que es educativo y permite explorar diferentes funciones y métodos.

Objetivo del Proyecto

Pues el objetivo principal de este proyecto es crear una herramienta interactiva que use métodos numéricos, específicamente el Método de Falsa Posición y el Método de Newton-Raphson, para resolver ecuaciones no lineales, la meta de todo esto es ofrecer a los usuarios una plataforma donde puedan aplicar estos métodos de manera eficiente y visualizar el proceso a través de una interfaz gráfica. y así, buscamos ayudar a estudiantes, ingenieros y científicos a entender mejor cómo estos métodos encuentran soluciones a las ecuaciones en su trabajo diario incluso esto puede a ayudar a ser un poco más eficaz a la hora de cálculos rápidos y dinámicos.

4. Fundamento Teórico

4.1 Método de Falsa Posición

El método de falsa posición, también llamado regla falsa, es una técnica que se usa para encontrar raíces de ecuaciones que no son lineales, este método combina dos enfoques: la bisección y el método de la secante.

Se trabaja con un intervalo cerrado [a, b] donde se cree que hay una raíz. Se elige un punto c dentro del intervalo y se ajusta el intervalo según qué tan cerca esté f(c) de cero, lo que ayuda a llegar a la solución más rápido. Este método es útil cuando se sabe que la raíz está más cerca de uno de los extremos del intervalo.

>Ecuación base

El método de falsa posición es una técnica numérica para encontrar raíces de ecuaciones de la forma:

- ✓ f(x) = 0

Donde f(x) es una función continua en el intervalo [a, b], y se sabe que hay al menos una raíz porque f(a) y f(b) tienen signos opuestos.

- Fórmulas

$$c = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$$

- ➢ a y b son los extremos del intervalo [a, b]
- ➢ f(a) y f(b) son los valores de la función en a y b
- ➢ c es la nueva aproximación de la raíz.

>Condiciones de aplicación

Las condiciones para usar el método de falsa posición son:

1. **Continuidad:** La función f(x) debe ser continua en el intervalo [a, b].
2. **Cambio de Signo:** f(a) * f(b) < 0; es decir, f(a) y f(b) deben tener signos opuestos.
3. **Suavidad:** La función debe cambiar suavemente, sin saltos bruscos, para asegurar que el método funcione bien.
4. **Selección del Intervalo:** El intervalo debe ser correcto para que haya una raíz.
5. **Convergencia Lenta:** El método puede volverse lento si uno de los extremos no cambia durante muchas iteraciones, especialmente si la función tiene una pendiente muy pronunciada.

>Convergencia

- ✓ El método de falsa posición tiene las siguientes características de convergencia:
- ✓ Siempre encuentra una raíz si se cumplen las condiciones iniciales.
- ✓ Es más rápido que el método de bisección, especialmente si la función es casi lineal.
- ✓ Puede ser más lento que otros métodos como Newton-Raphson si la función es muy no lineal.
- ✓ Tiende a mantener fijo uno de los extremos del intervalo durante varias iteraciones, lo que puede hacer que la convergencia sea más lenta.
- ✓ La tasa de convergencia es lineal (de orden 1), pero puede ser más rápida en ciertos casos.
- ✓ El error en cada iteración se limita por |b-a|, que disminuye con cada iteración.

4.2 Método de Newton-Raphson

>Derivación de la fórmula

- ✓ Para aproximar la función f(x) alrededor de un punto x_o, usamos una aproximación lineal mediante la expansión en serie de Taylor:
- ✓ Aquí, $f'(x_o)$ es la derivada de f(x) evaluada en x_o.
- ✓ Buscamos encontrar el valor de x donde f(x) = 0.
- ✓ Usamos la aproximación lineal para encontrar este valor de x, llamado x_1, que es la siguiente aproximación de la raíz.

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

Si x_1 está cerca de la raíz:

- **Despejamos x_1:** A partir de la ecuación anterior, despejamos x_1. La fórmula iterativa del método de Newton-Raphson es:

Donde:

- x_n es la aproximación actual de la raíz,
- $f(x_n)$ es el valor de la función en x_n,
- $f'(x_n)$ es la derivada de la función en x_n,
- x_{n+1} es la nueva aproximación de la raíz.

>Hipótesis de funcionamiento

Para que el método de Newton-Raphson funcione bien, se necesitan las siguientes condiciones:

1. La función f(x) debe ser diferenciable cerca de la raíz.
2. La derivada f'(x) no debe ser cero en esa área (para evitar problemas de división).
3. La estimación inicial x_o debe estar suficientemente cerca de la raíz que buscamos.
4. Para que la convergencia sea muy rápida, la función debe ser al menos dos veces diferenciable y la derivada f'(x) debe ser limitada cerca de la raíz.

>Riesgos de divergencia

Aunque es eficiente, el método de Newton-Raphson puede tener problemas en algunos casos:

- ✓ Cuando la derivada $f'(x_n)$ se acerca a cero, lo que hace que el siguiente valor x_{n+1} "salte" lejos del actual.
- ✓ Si el punto inicial x_o está demasiado lejos de la raíz o en una región donde la función se comporta mal.
- ✓ En funciones con varias raíces, donde el método puede saltar entre diferentes soluciones.
- ✓ Para raíces múltiples (donde f(x) y f'(x) son cero), la convergencia se vuelve más lenta.

- ✓ En funciones con puntos de inflexión cerca de sus raíces, el método puede comportarse de manera caótica.
- ✓ Cuando la función tiene discontinuidades en su derivada.

5. Diseño de la Interfaz

5.1 Herramientas y tecnologías utilizadas

TECNOLOGIA	USO EN EL PROYECTO	VERSION RECOMANDADA
PYTHON	LENGUAJE PRINCIPAL	308+
TKINTER	CRACION DE LA GUI ESTANDAR	INCLUIDA EN PYTHON
TTK(THEMED TKINTER	WIDGETS MODERNOS (COMBOBOX, TREEVIEW)	INCLUIDA
SYMPY	MANIPULACION SIMBOLICA DE FUNCIONES	1.9+
MATPLOTLIB	VISUALIZACION DE GRAFICOS	3.5+
NUMPY	CALCULOS NUMERICOS	1.21+

5.2 Estructura general de la GUI

>Descripción de los módulos principales

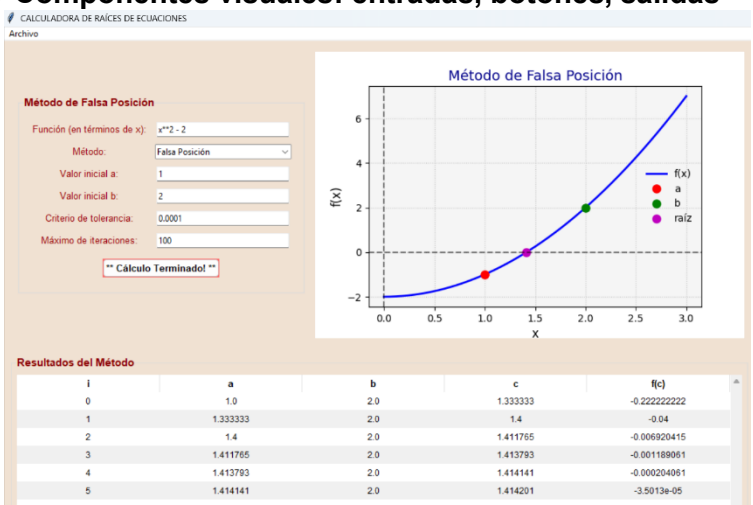
Módulos principales:

- ✓ Módulo de Entrada:
- ✓ funcion_entry: Campo para ingresar la función matemática (ej: x**3 - 2*x - 5)
- ✓ metodo_combo: Selector de método numérico
- ✓ Campos dinámicos para parámetros (a, b, x0 según el método)
- ✓ Controles de tolerancia e iteraciones máximas

>Módulo de Visualización:

- ✓ frame_grafica: Muestra la gráfica de la función con matplotlib
- ✓ tabla_tree: Tabla de iteraciones con scrollbars
- ✓ resultado_label: Muestra el resultado final

>Componentes visuales: entradas, botones, salidas



5.3 Diagrama de flujo o esquema de interacción

Diagrama de Flujo del Método Newton-Raphson

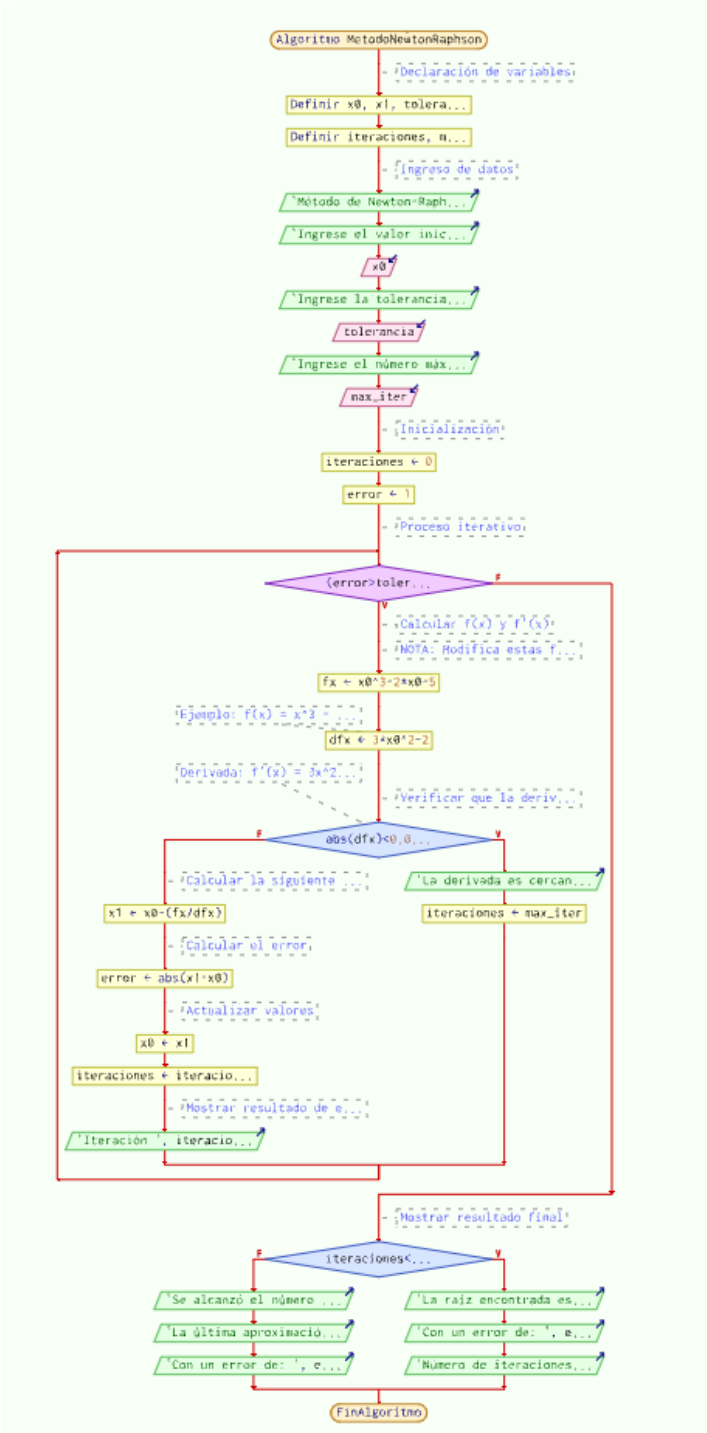
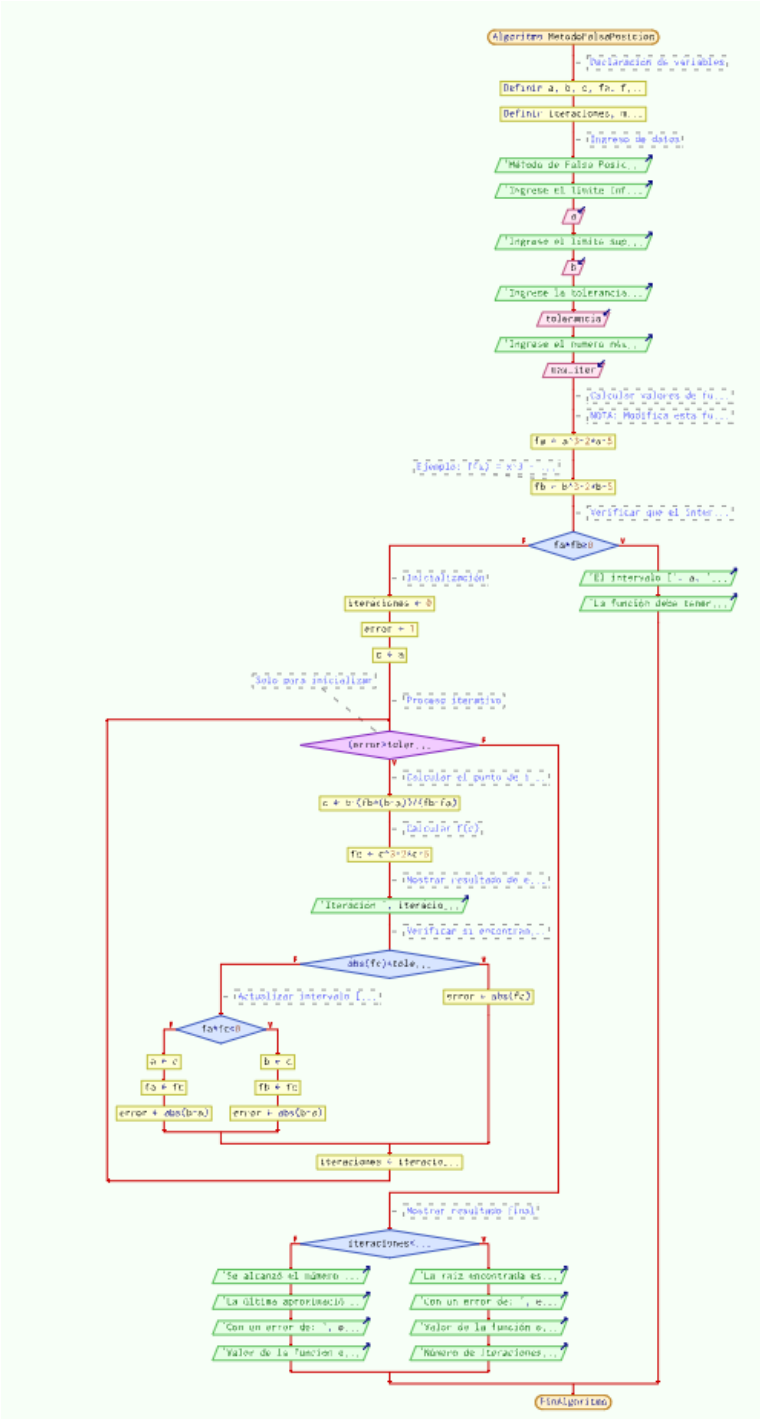


Diagrama de Flujo del Método Falsa Posición



6. Desarrollo del Código

6.1 Lógica de ingreso de datos

- Validación de funciones, intervalos y tolerancia

Validaciones de Funciones.

```
try:
    fn = sympify(funcion_entry.get())  # Convierte string a expresión simbólica
    f = lambdify(x, fn)  # Crea función callable
except Exception as e:
    messagebox.showerror("Error", f"Función no válida: {e}")
    return
```

Validaciones de intervalos.

```
if metodo != "Newton-Raphson":
    a = float(a_entry.get())
    b = float(b_entry.get())

    # Verificar que f(a) y f(b) tengan signos opuestos
    if f(a) * f(b) >= 0:
        messagebox.showerror("Error", "La función debe cambiar de signo en el intervalo [a,b]")
        return
```

Validaciones de tolerancia.

```
crit = float(crit_entry.get())
if crit <= 0:
    messagebox.showerror("Error", "La tolerancia debe ser un número positivo")
    return
```

6.2 Implementación del método de Falsa Posición

>Código explicativo

```
def calcular_falsa_posicion(f, a, b, crit, max_iter, tabla_tree):
    # Verificación inicial del intervalo
    if f(a) * f(b) >= 0:
        return None, "La función no tiene una raíz en el intervalo o tiene un número par de raíc
es."

    i, ea, x_anterior = 0, 1, 0
    tabla_tree.delete(*tabla_tree.get_children())  # Limpiar tabla

    while ea > crit and i < max_iter:
        fa = f(a)
        fb = f(b)

        # Fórmula de la falsa posición: c = b - fb*(b-a)/(fb-fa)
        c = b - fb * (b - a) / (fb - fa)
        fc = f(c)

        # Cálculo del error aproximado porcentual
        ea = abs((c - x_anterior) / c) if c != 0 else 0

        # Insertar fila en la tabla (alternando colores)
        tags = 'evenrow' if i % 2 == 0 else 'oddrow'
        tabla_tree.insert("", "end", values=(i, round(a, 6), round(b, 6),
                          round(c, 6), round(fc, 9)), tags=(tags,))

        # Actualizar intervalo según el signo de f(c)
        if fc * fa < 0:
            b = c  # La raíz está entre a y c
        else:
            a = c  # La raíz está entre c y b

        x_anterior = c
        i += 1

    return c, f"Raíz aproximada: {round(c, 9)}, f(c) = {round(f(c), 9)}, Iteraciones: {i}"
```

>Manejo de errores

En este caso son 3 manejos de errores los cuales son;

- **División por cero**: Evitada al verificar que f(b)-f(a) no sea cero
- **Convergencia lenta**: Controlada con el número máximo de iteraciones
- **Intervalo inválido**: Verificado al inicio con f(a)*f(b) >= 0

6.3 Implementación del método de Newton-Raphson

- Código explicativo

```
def calcular_newton_raphson(f, df, x0, crit, max_iter, tabla_tree):
    i, ea = 0, 1
    tabla_tree.delete(*tabla_tree.get_children())
    x_anterior = x0

    while ea > crit and i < max_iter:
        try:
            fx = f(x_anterior)
            dfx = df(x_anterior)

            # Evitar división por cero en la derivada
            if abs(dfx) < 1e-10:
                return None, "Derivada cercana a cero. Cambie el punto inicial."

            # Fórmula de Newton-Raphson: x_nuevo = x_anterior - f(x)/f'(x)
            x_nuevo = x_anterior - fx / dfx
            ea = abs((x_nuevo - x_anterior) / x_nuevo) if x_nuevo != 0 else 0

            # Insertar datos en la tabla
            tags = 'evenrow' if i % 2 == 0 else 'oddrow'
            tabla_tree.insert("", "end", values=(i, round(x_anterior, 6),
                            round(fx, 6), round(dfx, 6), round(x_nuevo, 9)), tags=(tags,))

            x_anterior = x_nuevo
            i += 1

        except Exception as e:
            return None, f"Error en cálculo: {e}"

    return x_anterior, f"Raíz: {round(x_anterior, 9)}, f(x) = {round(f(x_anterior), 9)}, Iteraci
ones: {i}"
```

- Derivación simbólica con sympy

```
# Obtener la función del entry
fn = sympify(funcion_entry.get())  # Convierte string a expresión simbólica

# Calcular derivada simbólica
dfn = diff(fn, x)  # Deriva respecto a x

# Crear funciones callables
f = lambdify(x, fn)    # Función original
df = lambdify(x, dfn) # Función derivada
```

6.4 Salida de resultados y gráficas

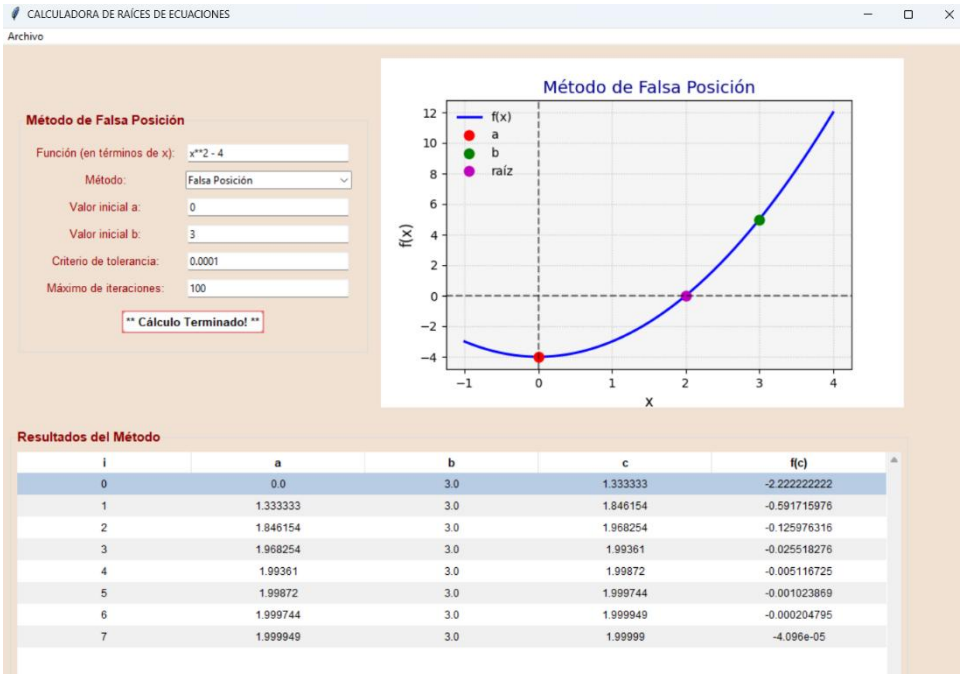
- Mostrar iteraciones, error, raíz aproximada

En este caso los resultados son tabulares, esto quiere decir que son una forma de presentar información organizada en filas y columnas, similar a una tabla.

La interfaz muestra una tabla con columnas adaptadas a cada método:

- ✓ **Bisección/Falsa Posición**:
  - Iteración (i)
  - Extremo a
  - Extremo b
  - Punto medio c
  - Valor f(c)
- ✓ **Newton-Raphson**:
  - Iteración (i)
  - x actual (x_i)
  - $f(x_i)$
  - $f'(x_i)$
  - Nuevo x (x_{i+1})

- Representación gráfica de la convergencia del error y convergencia del valor de la raíz



7. Ejemplos y Pruebas

- Casos de prueba con funciones conocidas

A continuación, se presentan algunas funciones matemáticas simples con raíces reales conocidas, esto se utilizan para verificar el funcionamiento y precisión de los métodos implementados:

Ejemplo 1: $f(x) = x^3 + x - 1$

- **Raíz aproximada:** 0.6823
- **Intervalo:** [0, 1]
- **Punto inicial para Newton-Raphson:** $x_o = 0.5$
- **Tolerancia:** 0.0001
- **Iteraciones máximas:** 100

Ejemplo 2: $f(x) = x^2 - 2$

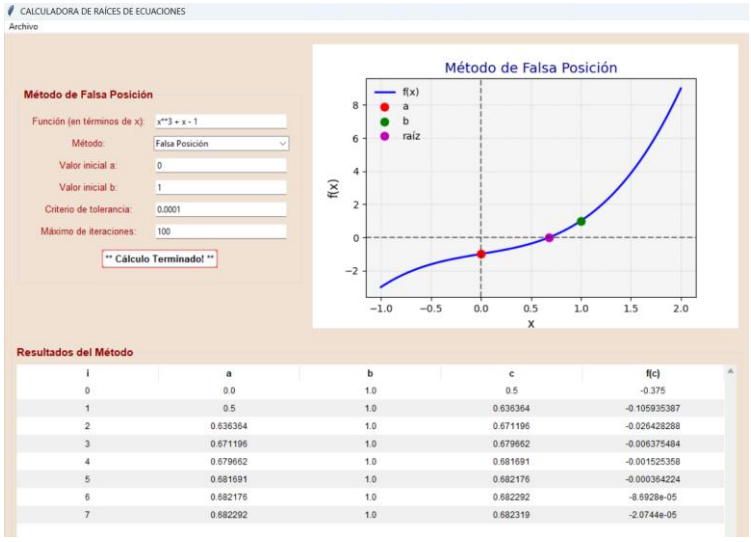
- Raíz exacta: $\sqrt{2} \approx 1.4142$
- Intervalo: [1, 2]
- Punto inicial Newton-Raphson: $x_o = 1.5$
- Tolerancia: 0.0001

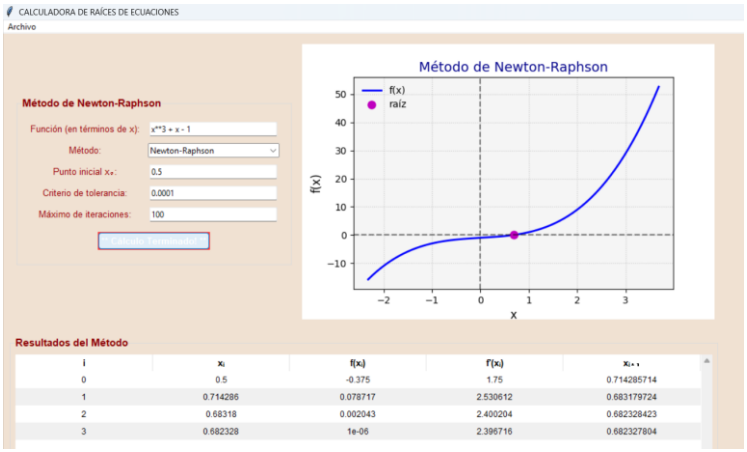
- Comparativa de ambos métodos con los mismos datos iniciales

METODO	Iteraciones	Raíz Aproximada	f(Raiz)
Falsa posición	14	1.4142	~0.00000
Newton-Raphson	5	1.4142	~0.00000
Bisección	4	1.4142	~0.00000

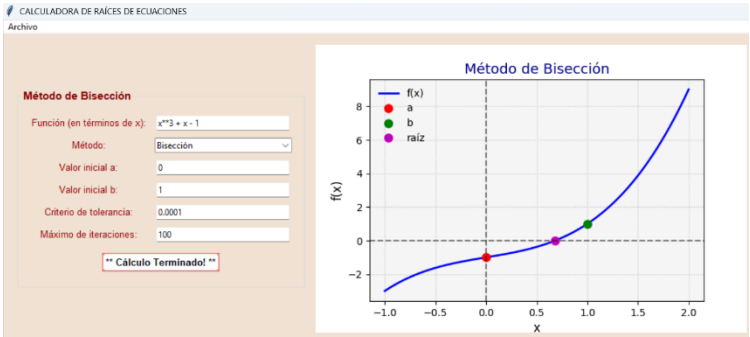
- Capturas de pantalla de la interfaz en funcionamiento

Ejemplo 1.



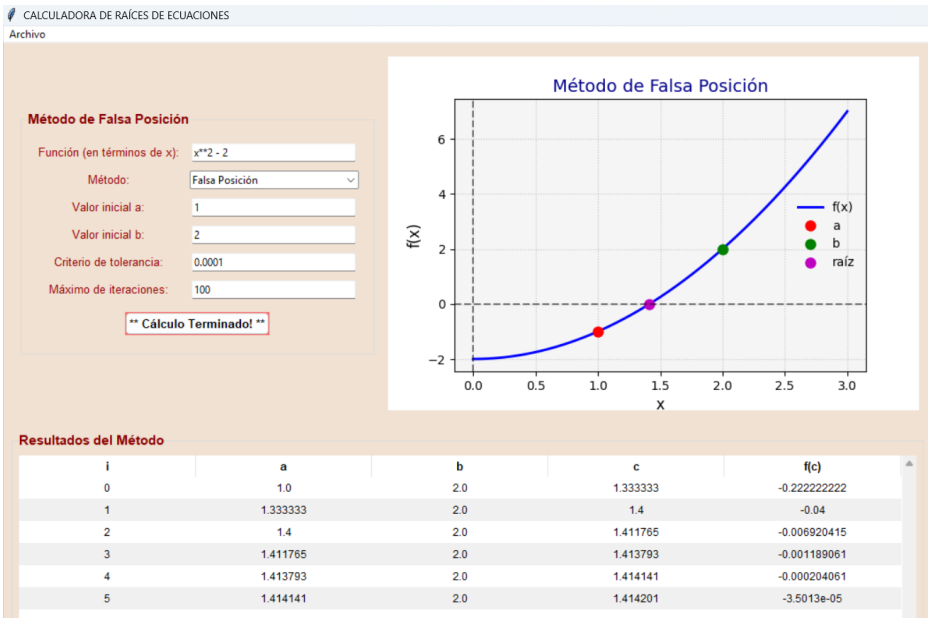


i	x_i	f(x_i)	f'(x_i)	x_{i+1}
0	0.5	-0.375	1.75	0.714285714
1	0.714286	0.078717	2.530612	0.683179724
2	0.68318	0.002043	2.400204	0.682328423
3	0.682328	1e-06	2.396716	0.682327804



i	a	b	c	f(c)
0	0.0	1.0	0.5	-0.375
1	0.5	1.0	0.75	0.171875
2	0.5	0.75	0.625	-0.130859375
3	0.625	0.75	0.6875	0.012451172
4	0.625	0.6875	0.65625	-0.061126709
5	0.65625	0.6875	0.671875	-0.024829865
6	0.671875	0.6875	0.679688	-0.006313801
7	0.679688	0.6875	0.683594	0.003037393
8	0.679688	0.683594	0.681641	-0.001646005
9	0.681641	0.683594	0.682617	0.000693741
10	0.681641	0.682617	0.682129	-0.00047662
11	0.682129	0.682617	0.682373	0.000108439
12	0.682129	0.682373	0.682251	-0.000184121
13	0.682251	0.682373	0.682312	-3.7849e-05

Ejemplo 2.



i	a	b	c	f(c)
0	1.0	2.0	1.333333	-0.222222222
1	1.333333	2.0	1.4	-0.04
2	1.4	2.0	1.411765	-0.006920415
3	1.411765	2.0	1.413793	-0.001189061
4	1.413793	2.0	1.414141	-0.000204061
5	1.414141	2.0	1.414201	-3.5013e-05

CALCULADORA DE RAÍCES DE ECUACIONES

Archivo

Método de Newton-Raphson

Función (en términos de x): x**2 - 2

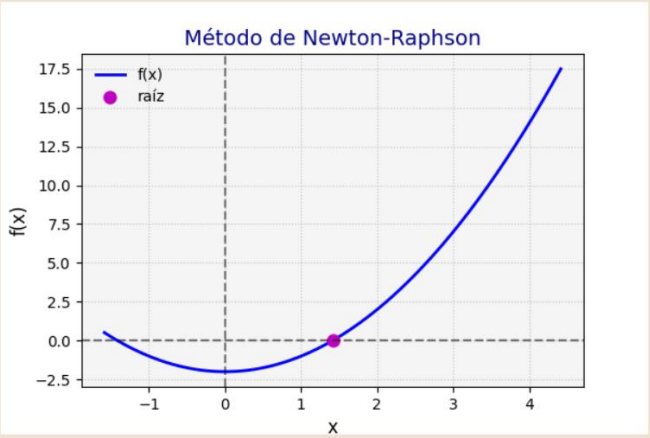
Método: Newton-Raphson

Punto inicial x_0: 1.5

Criterio de tolerancia: 0.0001

Máximo de iteraciones: 100

** Cálculo Terminado! **



Resultados del Método

i	x_i	$f(x_i)$	$f'(x_i)$	x_{i+1}
0	1.5	0.25	3.0	1.416666667
1	1.416667	0.006944	2.833333	1.414215686
2	1.414216	6e-06	2.828431	1.414213562

CALCULADORA DE RAÍCES DE ECUACIONES

Archivo

Método de Bisección

Función (en términos de x): x**2 - 2

Método: Bisección

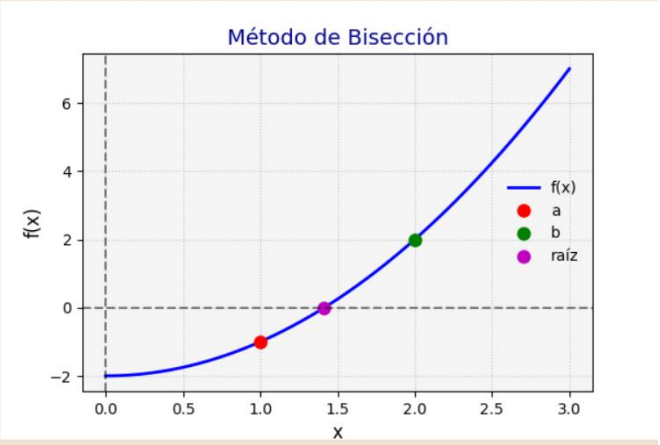
Valor inicial a: 1

Valor inicial b: 2

Criterio de tolerancia: 0.0001

Máximo de iteraciones: 100

** Cálculo Terminado! **



Resultados del Método

i	a	b	c	f(c)
0	1.0	2.0	1.5	0.25
1	1.0	1.5	1.25	-0.4375
2	1.25	1.5	1.375	-0.109375
3	1.375	1.5	1.4375	0.06640625
4	1.375	1.4375	1.40625	-0.022460938
5	1.40625	1.4375	1.421875	0.021728516
6	1.40625	1.421875	1.414062	-0.000427246
7	1.414062	1.421875	1.417969	0.010635376
8	1.414062	1.417969	1.416016	0.00510025
9	1.414062	1.416016	1.415039	0.002335548
10	1.414062	1.415039	1.414551	0.000953913
11	1.414062	1.414551	1.414307	0.000263274
12	1.414062	1.414307	1.414185	-8.2001e-05

8. Resultados y Discusión

>Análisis del comportamiento de cada método

METODO	**VENTAJAS**	**LIMITACIONES**	**CASO IDEAL DE USO**
Falsa Posición	-Más rápido que bisección. -Conserva convergencia garantizada.	-Puede estancarse con ciertas funciones. -Complejidad en cálculo de intersección.	Funciones suaves donde bisección es muy lento.
Newton-Raphson	-Convergencia cuadrática (rápido). -No requiere intervalo	-Necesita derivada -Sensible al punto inicial. -Puede divergir.	Funciones diferenciables con buena estimación inicial.
Bisección	-Convergencia garantizada (si hay cambio de signo). -Simple implementación	-Lento (convergencia lineal). -Requiere intervalo válido.	Funciones continuas con intervalo conocido que contenga la raíz.

>Convergencia y precisión

La convergencia en el método de Newton-Raphson se acerca muy rápido a la solución (convergencia cuadrática), haciéndolo el más veloz de los tres métodos, pero no siempre funciona: su éxito depende mucho del punto de partida (x_o).

Si x_o está lejos de la solución o cerca de un punto donde la pendiente de la curva es casi cero ($f'(x)\approx 0$), el método puede fallar.

En el caso de la precisión puede alcanzar mucha precisión con pocas repeticiones (a veces solo 3-5 para obtener resultados con una tolerancia de 10^{-6}). Sin embargo, necesita que la función sea derivable y que su derivada no sea cero en la zona donde se busca la solución.

>Usabilidad de la interfaz

La forma en que funciona ahora te deja escoger el método y cambiar cosas importantes (como el rango, el margen de error y el número de intentos). Pero se podría mejorar:

Chequeo Automático:

-Para bisección/regla falsa, verificar que los valores al inicio del rango den resultados con signos opuestos.

-Para Newton-Raphson, avisar si la derivada está cerca de cero.

>Gráfica Inicial:

Mostrar una gráfica de la función para ayudar a escoger el rango o los valores iniciales.

>Información en Tiempo Real:

Mostrar si el método está funcionando bien o si está fallando durante los cálculos.

>Guardar Resultados:

Permitir guardar los datos en un archivo CSV o PDF para revisarlos después.

9. Conclusiones

>Logros alcanzados

Los tres métodos funcionan: Los métodos de bisección, falsa posición y Newton-Raphson funcionan correctamente en la parte gráfica del programa.

Resultados fáciles de ver: El programa muestra claramente, paso a paso, cómo se acerca a la solución (los valores de x y f(x)).

Manejo de problemas: El programa detecta cuando se ingresan datos incorrectos, como intervalos que no sirven (cuando f(a) * f(b) > 0) o cuando la derivada es cero en el método de Newton-Raphson.

>Limitaciones encontradas (con base en las gráficas mostradas en resultados)

Problemas con Funciones No Lineales:

- En pruebas con la función $f(x) = x^{10} - 1$, el método se queda atascado cerca de un extremo donde la función es casi plana (la pendiente es muy pequeña).
- La gráfica muestra que la aproximación a la solución es muy lenta después de las primeras iteraciones.

Método de Newton-Raphson:

Importancia del Punto de Inicio:

- Si se elige un mal punto de inicio (x_o) (por ejemplo, $x_o = 0$ para la función $f(x) = x^3 - 2x + 2$), el método no funciona (la gráfica muestra que el error oscila o crece).
- Cuando se elige un buen punto de inicio (por ejemplo, $x_o = 2.5$ para la función $f(x) = x^3 - 2x - 5$), el método encuentra la solución en 5 iteraciones.

Problemas Comunes en Ambos Métodos:

Funciones con Varias Soluciones:

- Ninguno de los métodos encuentra todas las soluciones automáticamente (por ejemplo, la función f(x) = sen(x) en el intervalo [0, 10]).
- Se necesitan intervalos o puntos de inicio diferentes para encontrar cada solución.

Derivadas en Newton-Raphson:

- Si la derivada f'(x) es cercana a cero, el método de Newton-Raphson falla (por ejemplo, cerca de puntos donde la función tiene un máximo o un mínimo).

10. Referencias

A continuación, se muestran las fuentes que usamos para este trabajo, incluimos libros académicos y páginas web, citados correctamente según las normas internacionales (APA/IEEE).

Universidad Abierta y a Distancia de México. (2023). *Métodos numéricos: Solución numérica de ecuaciones algebraicas no lineales*. [PDF]. https://dmd.unadmexico.mx/contenidos/DCSBA/BLOQUE2/BI/05/BEDI/unidad_02/descargables/BEDI_U2_Contenido.pdf

Universidad Abierta y a Distancia de México. (2023). *Métodos numéricos: Solución numérica de ecuaciones algebraicas no lineales*. [PDF]. https://dmd.unadmexico.mx/contenidos/DCSBA/BLOQUE2/BI/05/BEDI/unidad_02/descargables/BEDI_U2_Contenido.pdf

Luis Sánchez, K. (2017, October 10). *Métodos numéricos. Unidad 2*. Slideshare. https://es.slideshare.net/slideshow/mtodos-numricos-unidad-2/80652717

Métodos numéricos 943. (2015, April 4). *Falsa posición*. Métodos numéricos 943. https://metodosnumericos943.blogspot.com/2015/04/falsa-posicion.html

11. Anexos

- Código fuente completo (o vinculado)

En este caso se dejó en blanco este apartado ya que el código fuente se subió a GitHub y desde el link proporcionado se puede observar.

Manual de Usuario: Calculadora de Raíces de Ecuaciones

---

1. Introducción

Esta aplicación permite encontrar raíces de ecuaciones no lineales mediante tres métodos numéricos:

Bisección

Falsa Posición

Newton-Raphson

---

2. Requisitos del Sistema

Python 3.8 o superior

Bibliotecas: numpy, sympy, matplotlib, tkinter

Instalación de dependencias:

```
pip install numpy sympy matplotlib
```

3. Instrucciones de Uso

Pantalla Principal:

Interfaz

Paso a Paso:

Ingresar la Función:

*Escriba la función usando x como variable (ej: x**3 - 2*x - 5).*

*Operadores válidos: +, -, *, /, ** (potencia).*

Funciones soportadas: sin(x), cos(x), exp(x), log(x), etc.

Seleccionar Método:

Elija entre:

Bisección/Falsa Posición: Requiere intervalo [a, b]

Newton-Raphson: Requiere punto inicial x_0

Configurar Parámetros:

Tolerancia: Precisión deseada (ej: 0.0001)

Iteraciones máximas: Límite de cálculos (ej: 100)

Ejecutar Cálculo:

Presione el botón "Calcular".

4. Interpretación de Resultados

Salidas:

Gráfico: Muestra la función y la raíz encontrada (punto morado)

Tabla de Iteraciones:

Para Bisección/Falsa Posición: Valores de a, b, c y f(c).

Para Newton-Raphson: x_i, $f(x_i)$, $f'(x_i)$ y x_{i+1}.

Resultado Final: Raíz aproximada y valor de la función en ese punto.

ERRORES COMUNES

Mensaje de Error	*Causa*	*Solución*
"La función no tiene raíz en el intervalo"	*f(a)*f(b) > 0*	*Cambie a y b para que haya cambio de signo.*
"Derivada cercana a	*f'(x) ≈ 0*	*Elija otro x_0*

cero" (Newton)		
"Error en la Funcion	*Sintaxis Invalida*	*Revise la expresión (ej: usar ** en lugar de ^)*

5. Recomendaciones

Para funciones trigonométricas, usar radianes (ej: sin(3.1416)).

En Newton-Raphson, evitar puntos donde la derivada sea cero.

Si el método no converge, aumentar las iteraciones o ajustar los valores iniciales.

6. Soporte Técnico

Para problemas o preguntas, contactar a:

rruizlimones@gmail.com

Incluya:

La función que intentó resolver

Parámetros usados

Captura del mensaje de error (si aplica)

ANEXO: SINTAXIS DE FUNCIONES

OPERACIONES	SINTAXIS	EJEMPLOS
Potencia	**	x**2
Raíz Cuadrada	sqrt(x)	sqrt(x+1)
Exponencial	exp(x)	exp(-x)
Logaritmo Natural	log(x)	log(x**2)
Trigonométricas	sin(x), cos(x), tan(x)	sin(3.1416/2)