Universidad Politécnica de Aguascalientes

Ingeniería en Sistemas Computacionales

Manual Técnico

Materia: Matemáticas para ingeniería II

Docente: Isaac Vázquez Mendoza

Equipo 4

Integrantes:

Vázquez Ríos Juan Ramon UP230190
Olmedo Cruz Danna Giselle UP230183
Surdez Espeleta Andrea Mariana UP230188
Martin Cornejo Diego Ernesto UP230346
López Nava Alberto Misael UP239756

Grado y Grupo: ISC06A

Newton-Raphson

1. Introducción

Esta aplicación implementa el Método de Newton-Raphson, un algoritmo numérico utilizado para encontrar aproximaciones a las raíces de ecuaciones no lineales de la forma:

$$f(x) = 0$$

El método usa una sucesión de aproximaciones que mejoran en cada iteración mediante la fórmula:

$$x_{k+1} = x_{k+1} \quad - \frac{f(x_k)}{f'(x_k)}$$

La herramienta desarrollada permite:

- Ingresar una función f(x) y su derivada f'(x).
- Definir un valor inicial x_0 y número de iteraciones.
- Visualizar una tabla detallada del proceso iterativo.
- Graficar la función y cada recta tangente generada por el método.

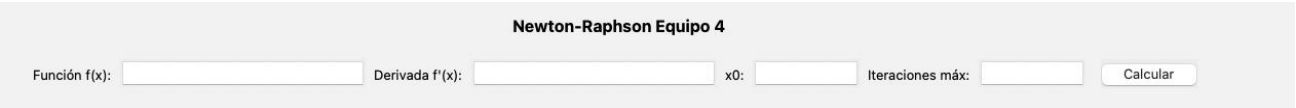
Es una aplicación ideal para cursos de métodos numéricos y análisis matemático.

2. Alcance y Propósito

Entrada

Generada por el usuario

- Función: $f(x)$ escrita como expresión simbólica válida (ej. x**2 - 4*x).
- Derivada: $f'(x)$, la derivada exacta de la función.
- Valor inicial: x0
- Iteraciones máximas.



Salida

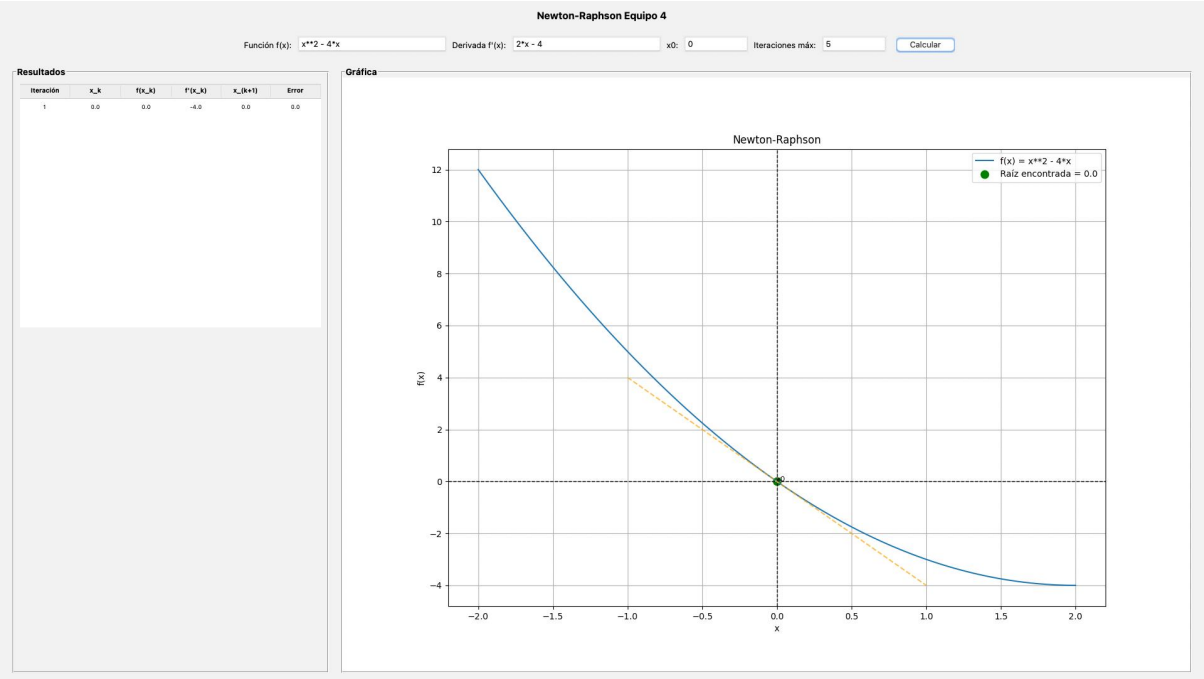
Tabla con:

- Iteración
- x_k
- $f(x_k)$

- $f'(x_k)$
- x_{k+1}
- *Error*

Gráfica completa de:

- f(x)
- Puntos de aproximación
- Rectas tangentes
- Raíz aproximada



3. Arquitectura del sistema

3.1 Tecnologías utilizadas

Componente	Tecnología
Lenguaje principal	Python 3.x
Procesamiento simbólico	“sympy” (“sympify”, “lambdify”, “Symbol”)
Cálculo numérico	“numpy” (usado internamente por “lambdify”)
Interfaz gráfica	“tkinter” + “ttk”
Visualización	“matplotlib” con “FigureCanvasTkAgg”

3.2. Estructura del código

- Archivo principal: NewtonRaphson.py
- Función principal: newton_raphson()
- Componentes de la GUI:
- Sección de entrada
- Tabla de resultados
- Gráfica
- Estilos y estructura visual

4. Algoritmo Implementado

4.1 Fórmula matemática

$$x_{k+1} = x_{k+1} \quad - \frac{f(x_k)}{f'(x_k)}$$

$$erro = |x_{k+1} - x_k|$$

El proceso continúa hasta:

- Alcanzar el número máximo de iteraciones
- Que el error sea menor a 10^{-7}

4.2 Validación de la derivada

Antes de ejecutar el método, el sistema verifica que la derivada ingresada coincide con:

$$f'(x) = \frac{d}{dx}(f(x))$$

Si no coincide, se detiene la ejecución y muestra un mensaje de error.

Esto evita resultados incorrectos.

5. Interfaz de Usuario (GUI)

5.1 Diseño visual

- Pantalla completa (-fullscreen)
- Fondo: #F2F2F2
- Tipografía: Segoe UI
- Títulos en negrita

5.2 Componentes principales

Componente	Descripción
Entradas	f(x), f'(x), x0, iteraciones
Botón	"Calcular" — ejecuta newton_raphson()
Tabla	Iter, xk, f(xk), f'(xk), x(k+1), error
Gráfica	Función, tangentes, puntos y raíz

6. Flujo de Datos

El usuario ingresa:

- Función
- Derivada
- Valor inicial
- Número de iteraciones

Al presionar Calcular:

- Se limpia la tabla.
- Se limpia la gráfica.
- Se revisa que la derivada sea correcta.
- Se convierten f y f' a funciones ejecutables mediante lambdify.
- Se calcula iterativamente Newton-Raphson.
- Se guardan todos los datos en una lista datos.{

Se grafica:

- Función
- Puntos de iteración
- Rectas tangentes
- Raíz estimada

La tabla se llena fila por fila con los valores generados.

7. Manejo de Errores

Todo el cuerpo de la función está dentro de un try-except.
Errores controlados:

- Derivada incorrecta → mensaje de error.
- Entrada inválida (float() falla).
- Expresión mal escrita (sympify falla).
- f'(x) = 0 → no se puede continuar.
- Operaciones no permitidas (ej. división por cero).

En todos los casos se notifica mediante messagebox.showerror().

8. Limitaciones Conocidas

- Requiere derivada: No aplica si $f'(x)$ es difícil de obtener o no existe.
- Derivada cero: Si $f'(x_n) = 0$ = 0 o es muy pequeña, el método falla por división inestable.
- Sensibilidad al valor inicial: Un mal x_0 puede causar divergencia o convergencia a otra raíz.
- Problemas con raíces múltiples: Convergencia lenta o no garantizada.
- No garantiza convergencia global: Solo converge si el punto inicial está cerca de la raíz.
- Funciones irregulares: En funciones muy no lineales puede oscilar o alejarse.

Regla del Trapecio

1. Introducción

El módulo implementa el método numérico de la Regla del Trapecio para aproximar el valor de una integral definida. Utiliza una interfaz gráfica desarrollada en Tkinter, con cálculo simbólico mediante SymPy y graficación con Matplotlib.

2. Alcance y Propósito

Este componente permite calcular integrales definidas de funciones ingresadas por el usuario, representando visualmente el área bajo la curva y los trapecios utilizados en la aproximación. Su propósito es apoyar el análisis numérico y la interpretación gráfica del método.

3. Arquitectura del sistema

El sistema se compone de tres módulos principales:

Módulo de Entrada (GUI):

- Recibe función, límites de integración y número de particiones.

Módulo Numérico:

- Conversión de la función a objeto simbólico.
- Generación de particiones.
- Aplicación de la Regla del Trapecio.

Módulo de Visualización:

- Gráfica de la función.
- Representación de los trapecios.
- Visualización del resultado.

4. Algoritmo Implementado

El método usa la fórmula clásica:

$$\text{Área} \approx \frac{h}{2}[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)]$$

- Definir el intervalo [a,b].
- Dividir en nnn subintervalos de ancho h.
- Evaluar la función en cada punto x_i.
- Aplicar la fórmula del trapecio.
- Generar gráfica con los trapecios utilizados.

5. Interfaz de Usuario (GUI)

El usuario cuenta con:

- Campo para ingresar la función f(x).
- Campo para el límite inferior aaa.
- Campo para el límite superior bbb.
- Campo para el número de particiones nnn.
- Botón para ejecutar el cálculo.
- Área de texto para mostrar el resultado.

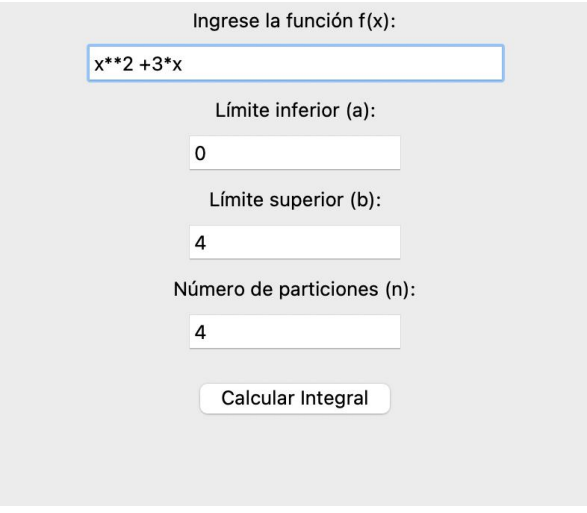
Se usa Tkinter para la creación de la ventana, controles y mensajes emergentes.
3

Componente	Descripción
Entrada Función	Campo de texto donde se ingresa la función $f(x)$.
Entrada Derivada	Campo para capturar manualmente $f'(x)$, que será verificada contra la derivada real.
Entrada x_0	Campo para el valor inicial del método.
Entrada Iteraciones	Campo para el número máximo de iteraciones permitidas.
Tabla de Resultados	Treeview que muestra por iteración: , x_k, $f(x_k)$, $f'(x_k)$, x_{k+1} y el error.
Gráfica	Muestra la función, las tangentes en cada iteración y la raíz encontrada.

6. Flujo de Datos

Entrada:

- Función como cadena de texto. (ej. x**2 + 3*x)
- Valores numéricos para a = 0, b = 4 y n = 4.

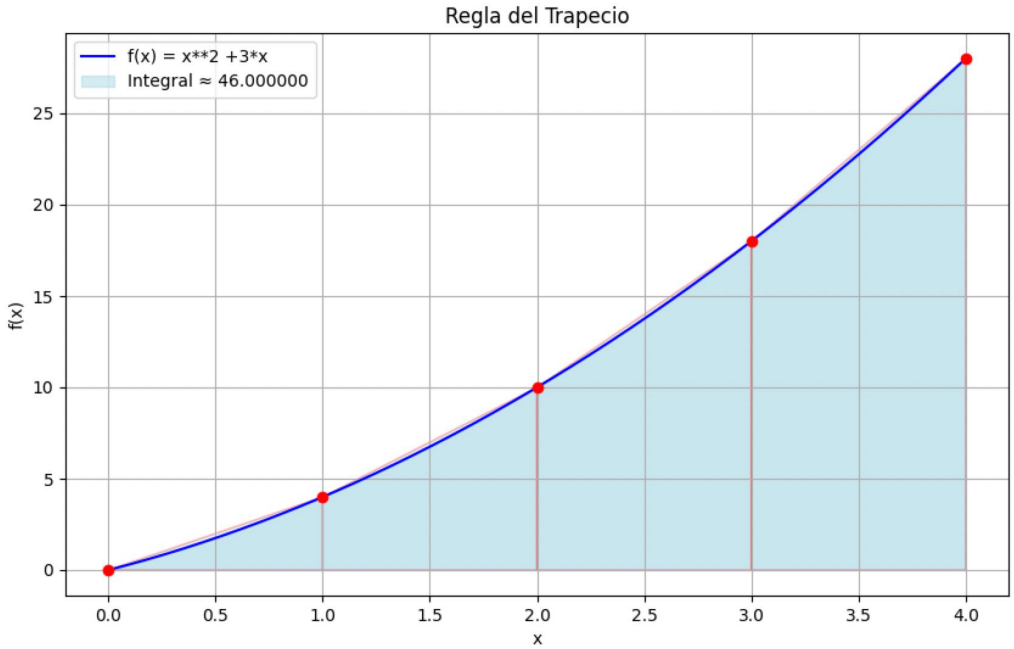


Proceso:

- Conversión de la función a expresión simbólica.
- Evaluación numérica.
- Aplicación del método del Trapecio.
- Generación de gráfica.

Salida:

- Valor aproximado de la integral.
- Gráfica de la función y los trapecios.
- Mensajes de error en caso necesario.



7. **Manejo de Errores**

- Validación de que los campos no estén vacíos.
- Conversión segura de los valores numéricos.
- Gestión de funciones mal escritas usando excepciones de SymPy.
- Captura de errores matemáticos (división por cero, evaluaciones inválidas).
- Notificación mediante messagebox.showerror().

8. **Limitaciones Conocidas**

- Precisión limitada en funciones muy curvas.
- Requiere un número grande de particiones para buena exactitud.
- Mal desempeño en funciones oscilatorias o con pendientes abruptas.
- No detecta discontinuidades dentro del intervalo.
- Sensible al intervalo si es demasiado grande sin aumentar n.

Coeficiente Constante

1. Introducción

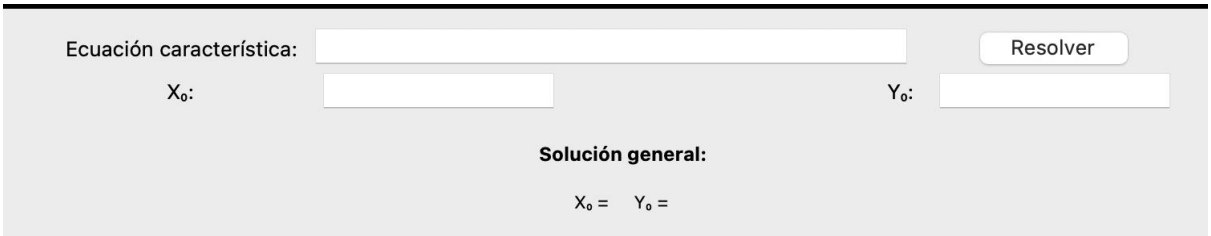
Esta aplicación implementa un solucionador para Ecuaciones Diferenciales Ordinarias Lineales Homogéneas con Coeficientes Constantes. El software se centra en resolver la Ecuación Característica asociada a la EDO mediante el método numérico de Newton-Raphson.

La herramienta tiene fines educativos, permitiendo al usuario visualizar cómo las raíces del polinomio característico determinan la forma de la solución general de la ecuación diferencial.

2. Alcance y Propósito

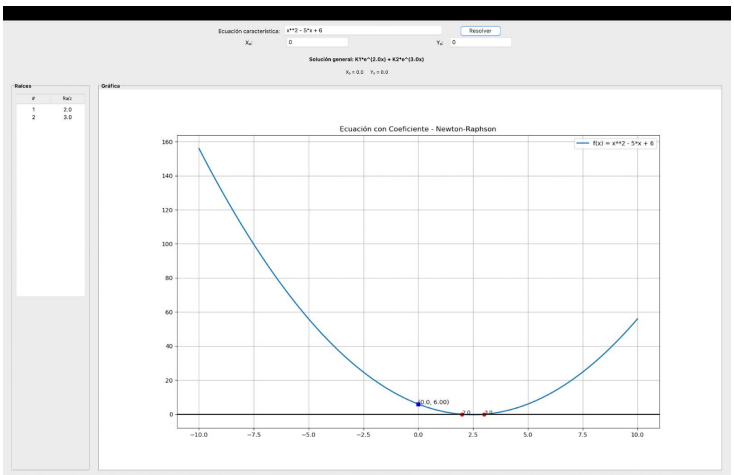
Entrada:

- Ecuación Característica: El usuario ingresa el polinomio auxiliar (ej. x**2 - 5*x + 6) que representa la EDO.
- Condiciones Iniciales: Valores para x_0 y y_0 .



Salida:

- Tabla de Raíces: Lista de las raíces reales encontradas por el método numérico.
- Solución General: Construcción textual de la solución $y(x)$ basada en las raíces.
- Gráfica: Visualización del polinomio característico $f(x)$ y ubicación de sus raíces.



3. Arquitectura del Sistema

Tecnologías utilizadas

- Lenguaje principal: Python
- Procesamiento simbólico: sympy (para interpretar el polinomio ingresado y calcular su derivada automáticamente).
- Cálculo numérico: numpy (evaluación eficiente de funciones).
- Interfaz gráfica: tkinter + ttk
- Visualización: matplotlib con FigureCanvasTkAgg

Estructura del código

- Funciones Auxiliares:normalizar(): Prepara la cadena de texto para ser leída por SymPy (convierte ^ a **).

- newton_find_root(): Implementación pura del algoritmo numérico.
- Función Principal: resolver() — Orquesta la lectura de datos, el cálculo simbólico, la búsqueda iterativa de raíces y el graficado.
- Interfaz: Construcción visual con frames para entrada, resultados y gráficos.

4. Algoritmo Implementado

Fundamento Teórico

Para una EDO de la forma ay″+by′+cy=0, se resuelve la ecuación característica ar2+br+c=0. La solución general se construye como:

y(x)=C1 e^r1 x+C2 e^r2 x+…

Método de Newton-Raphson

Para encontrar las raíces r del polinomio, se utiliza la fórmula iterativa:

$$x_{k+1} = x_{k+1} \quad - \frac{f(x_k)}{f^{'}(x_k)}$$

El algoritmo itera hasta que la diferencia $|x_{k+1} - x_k|$ es menor a una tolerancia definida o se alcanza el límite de iteraciones.

Estrategia de Búsqueda de Raíces

Dado que Newton-Raphson encuentra solo una raíz cercana al punto de inicio, el algoritmo realiza un **barrido** probando 40 puntos iniciales diferentes en el intervalo [−10,10] para intentar capturar todas las raíces reales del polinomio.

5. Interfaz de Usuario (GUI)

Diseño visual

- Modo: Pantalla completa (-fullscreen).
- Estilo: Uso de LabelFrames para agrupar "Raíces" y "Gráfica".
- Tipografía: Segoe UI.

Componentes principales

Componente	Descripción
Entrada Polinomio	Campo de texto para la ecuación característica.
Entradas X0 ,Y0	Campos para condiciones iniciales.
Etiqueta Solución	Muestra dinámicamente el string de la solución general (ej. K1 e^2x+K2 e^3x).
Tabla de Resultados	Treeview que enumera las raíces encontradas.
Gráfica	Muestra la curva del polinomio característico y marca las raíces con puntos rojos.

6. Flujo de Datos

- El usuario ingresa el polinomio y presiona "Resolver".
- El sistema normaliza el texto y usa SymPy para derivar la función simbólicamente.
- Se convierten las funciones simbólicas a funciones numéricas (Lambdify).
- Se ejecuta un bucle que prueba múltiples puntos de inicio para Newton-Raphson.
- Las raíces únicas encontradas se almacenan y se muestran en la tabla.
- Se construye la cadena de texto de la Solución General.
- Se grafica el polinomio marcando visualmente las raíces detectadas.

7. Manejo de Errores

- Bloque Try-Except: Protege la ejecución principal.
- Validación Matemática: Dentro de Newton-Raphson, se verifica si la derivada f′(x) es cero (if dfx == 0) para evitar la división por cero, retornando None en ese caso.
- Filtrado de Duplicados: Se redondean los resultados a 6 decimales para evitar que pequeñas variaciones numéricas se interpreten como raíces distintas.

8. Limitaciones Conocidas

Tipo	Descripción
Raíces Complejas	El algoritmo de Newton-Raphson implementado solo busca en el dominio real. Si la EDO tiene solución oscilatoria (raíces complejas conjugadas), el programa no las encontrará.
Raíces Repetidas	El constructor de la solución general asume raíces distintas. No aplica la regla de multiplicar por x (ej. C1 e^rx+C2 xe^rx) en caso de multiplicidad.
Cálculo de Constantes	Aunque el programa pide X0 y Y0 , no resuelve el sistema de ecuaciones para encontrar los valores de las constantes K1 ,K2 . Solo muestra la solución general con constantes arbitrarias.
Dominio de Búsqueda	La búsqueda de raíces se limita al intervalo [−10,10]. Raíces fuera de este rango podrían no ser detectadas.
Gráfica	Lo que se grafica es el polinomio característico, no la curva solución de la ecuación diferencial y(x).

Euler

1. Introducción

Esta aplicación implementa el Método de Euler, un algoritmo numérico de primer orden para resolver problemas de valor inicial (PVI) en ecuaciones diferenciales ordinarias (EDOs) de la forma:

$$\frac{dx}{dt} = f(t, x), x(t_0) = x_0$$

La herramienta está diseñada para uso educativo, permitiendo visualizar paso a paso la solución numérica mediante una interfaz gráfica intuitiva. Aunque es el método más sencillo, sirve como base para comprender técnicas más avanzadas como el método de Euler mejorado o Runge-Kutta.

2. Alcance y Propósito

Entrada:

El usuario define:

- La función f(t, x)
- Condiciones iniciales t_0 y x_0
- Tamaño del paso h (puede ser positivo o negativo).

Salida:

- Tabla con los valores de cada iteración: t_k, x_k, $x_k + 1$.
- Gráfica de la solución aproximada x(t)

3. Arquitectura del Sistema

3.1 Tecnologías utilizadas

Componente	Tecnología
Lenguaje principal	Python 3.x
Procesamiento simbólico	“sympy” (“sympify”, “lambdify”, “Symbol”)
Cálculo numérico	“numpy” (usado internamente por “lambdify”)
Interfaz gráfica	“tkinter” + “ttk”
Visualización	“matplotlib” con “FigureCanvasTkAgg”

3.2. Estructura del código

- Archivo único: Euler.py
- Función principal: euler_method() — encapsula toda la lógica de cálculo.
- Interfaz: Construida con Tkinter y dividida en secciones lógicas (entrada, resultados, gráfica).

4. Algoritmo Implementado

4.1. Fórmula del Método de Euler

$$xk + 1 = xk + h \cdot f(tk, xk)$$

$$tk + 1 = tk + h$$

4.2. Condición de terminación del bucle

```
tf = x0  # EL LÍMITE LIMITADOR
...
while (h > 0 and tk_ < tf) or (h < 0 and tk_ > tf):
```

Esto significa que el bucle se detiene cuando la variable independiente *t* alcanza el valor inicial de la variable dependiente x0.

5. Interfaz de Usuario (GUI)

5.1. Diseño visual

- Color de fondo: #F2F2F2
- Tipografía: Segoe UI (títulos en negrita)
- Modo: Pantalla completa (fullscreen)

5.2. Componentes principales

Componente	Descripción
Campos de entrada	f(t, x), t0, x0, h
Botón "Calcular"	Ejecuta euler_method()
Tabla de resultados	ttk.Treeview con columnas: Iter, t_k, x_k, x_(k+1)
Gráfica	Línea con marcadores, ejes etiquetados, cuadrícula activada

6. Flujo de Datos

1. El usuario ingresa los datos.
2. Al presionar "Calcular":
    - La tabla y la gráfica se limpian.
    - Se analiza y convierte f(t, x) a una función ejecutable (lambdify).
    - Se itera usando el método de Euler hasta alcanzar tf = x0.
    - Cada paso se registra en la tabla y se almacena para graficar.
3. Se muestra la gráfica final.

7. Manejo de Errores

- Todo el cuerpo de euler_method() está envuelto en un bloque try-except.
- Errores comunes capturados:
    - Expresión inválida en f(t, x) → sympify falla.
    - Paso h = 0 → se lanza ValueError.
    - Entradas no numéricas → float() falla.
    - Evaluación de función no definida (ej. división por cero).
- En caso de error, se inserta una fila en la tabla con el mensaje: ["Error", "", "", mensaje].

8. Limitaciones Conocidas

Tipo	Descripción
Lógica de terminación	El valor final tf se iguala a x0, lo que produce resultados incorrectos o bucles infinitos si t0 < x0 y h > 0, o viceversa.
Precisión numérica	El método de Euler acumula errores rápidamente; no es adecuado para problemas rígidos o con alta sensibilidad.
Falta de validación de rango	No se verifica si tf es razonable respecto a t0 y h.
Sin solución analítica	A diferencia de otras versiones del proyecto, esta implementación no incluye cálculo simbólico de la solución exacta.

Euler Mejorado

1. Introducción

El Método de Euler Mejorado, también conocido como Método de Heun o Método Predictor-Corrector, es una técnica numérica de segundo orden (O(h2)) utilizada para resolver Ecuaciones Diferenciales Ordinarias con condiciones iniciales de la forma dx/dy =f(x,y).

El objetivo principal es aumentar la precisión del Método de Euler básico (O(h)) mediante el promedio de las pendientes al inicio y al final del intervalo de integración.

El método de Heun utiliza las siguientes fórmulas:

Concepto Teórico	**Fórmula Teórica (Notación Matemática)**	**Implementación en Python (Variables)**	**Función y Pasos**
Paso 1: Predictor (Pendiente Inicial)	k1 = f(tk, xk)	k1 = f(tk_, xk)	Calcula la pendiente en el punto inicial del intervalo.
Paso 2: Estimación y Pendiente Predictora	k2 = f(tk+1, x^*k+1) donde x^*k+1 = xk + hk1	k2 = f(tk_ + h, xk + h * k1)	Estima un valor temporal (x^*k+1) y usa ese valor para calcular la pendiente (k2) al final del intervalo.
Paso 3: Corrector (Valor Final)	xk+1 = xk + h/2(k1 + k2)	x_next = xk + (h/2) * (k1 + k2)	Calcula el nuevo valor final promediando las dos pendientes (k1 y k2).

Requisitos de Entrada (Input)

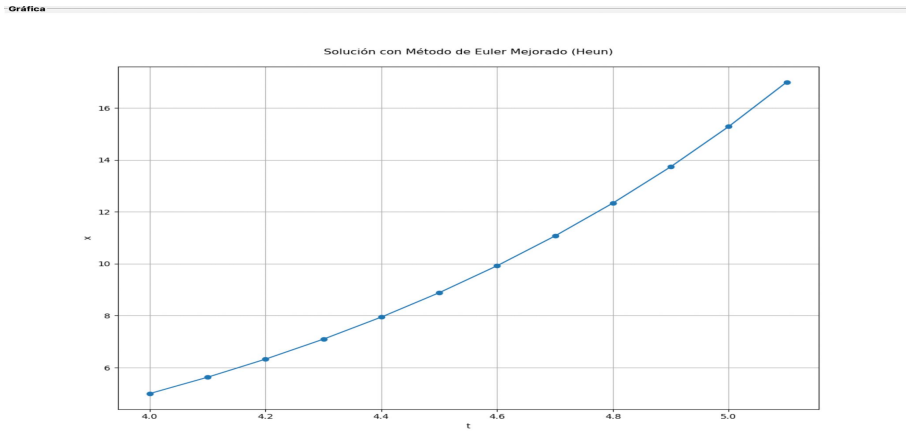
El usuario debe proveer los siguientes parámetros a través de la interfaz gráfica:

- **Función f(t,x):** La expresión matemática de la EDO, dx/dt.
- **Condición Inicial t0 :** Valor inicial de la variable independiente.
- **Condición Inicial x0 :** Valor inicial de la variable dependiente.
- **Tamaño del Paso h:** Determina el tamaño del salto en cada iteración. Debe ser distinto de cero.
- **Condición de Parada tf (Límite Limitador):** El valor tf para la terminación del bucle se fija, por requisito del curso, al valor de la condición inicial x0 (tf = x0).

Salidas Generadas (Output)

- **Tabla de Resultados:** Muestra los valores de cada iteración k, tk , xk , y el valor calculado xk+1 con una precisión de 5 decimales.

- **Gráfica de la Solución:** Representación visual de los puntos discretos (tk ,xk) que forman la solución aproximada x(t).



Robustez y Manejo de Errores

El código está diseñado con un control de errores que abarca las principales fallas del usuario y del sistema.

- **Validación de Paso:** Se incluye una verificación explícita para evitar que el paso h sea cero, lo que provocaría un bucle infinito o una falla lógica.
- **Control General:** Un bloque try-except general captura fallas en la entrada de datos (ej. texto en lugar de números) o errores de sintaxis en la función f(t,x), reportando el error directamente en la tabla de resultados.

Runge Kutta 4

1. Introducción

Esta aplicación implementa el Método de Runge-Kutta de cuarto orden (RK4), uno de los algoritmos numéricos más utilizados para resolver problemas de valor inicial (PVI) en ecuaciones diferenciales ordinarias (EDOs) de primer orden de la forma:

$$\frac{dx}{dt} = f(t, x)$$

$$x(t_0) = x_0$$

A diferencia del método de Euler (primer orden), RK4 ofrece una alta precisión mediante la combinación ponderada de cuatro estimaciones de la pendiente en cada paso. La herramienta incluye una interfaz gráfica que permite al usuario ingresar la función, condiciones iniciales y tamaño del paso, y visualizar tanto los resultados numéricos como la gráfica de la solución aproximada.

2. Alcance y Propósito

- Entrada: El usuario define:
  - La función
  - $f(t,x)$ como expresión simbólica válida (ej. "t - x").
  - Condiciones iniciales t_0 y x_0.
  - Tamaño del paso h (positivo o negativo).
- Salida:
  - Tabla detallada con: iteración, t_n, x_n, y los cuatro incrementos k_1, k_2, k_3, k_4, y x_{n+1}.
  - Gráfica de la solución aproximada $x(t)$.

- Uso: Educativo y de demostración, ideal para cursos de métodos numéricos o ecuaciones diferenciales.

3. Arquitectura del Sistema

3.1. Tecnologías utilizadas

Componente	Tecnología
Lenguaje principal	Python 3.x
Procesamiento simbólico	sympy (sympify, lambdify, Symbol)
Cálculo numérico	numpy (usado por lambdify)
Interfaz gráfica	tkinter + ttk
Visualización	matplotlib con FigureCanvasTkAgg

3.2. Estructura del código

- Archivo: RungeKutta4.py
- Función principal: runge_kutta_4() — encapsula toda la lógica numérica y de visualización.
- Interfaz: Diseño modular con secciones claras: entrada, tabla de resultados, y gráfica.

4. Algoritmo Implementado

4.1. Fórmula del Método RK4

El método RK4 calcula cuatro pendientes intermedias y las combina en una fórmula ponderada:

$$ɣ1=h\cdot\ ɔ(x̃,Ⱥ⁄sɣ)$$

$$ɣ2=h\cdot\ ɔ(x̃+h2,\ Ⱥ⁄sɣ+ɣ12)$$

$$ɣ3=h\cdot\ ɔ(x̃+h2,\ Ⱥ⁄sɣ+ɣ22)$$

$$ɣ4=h\cdot\ ɔ(x̃+h,\ Ⱥ⁄sɣ+ɣ3)$$

$$Ⱥ⁄sɣ+1=Ⱥ⁄sɣ+16(ɣ1+2ɣ2+2ɣ3+ɣ4)$$

$$x̃ɣ+1=x̃ɣ+h$$

- Orden del método: 4
- Error local: ů(ℎ5)
- Error global: ů(ℎ4)

5. Interfaz de Usuario (GUI)

5.1. Diseño visual

- Modo: Pantalla completa (-fullscreen)
- Color de fondo: #F2F2F2
- Tipografía: Segoe UI (títulos en negrita, tamaño diferenciado)

5.2. Componentes principales

Componente	Descripción
Campos de entrada	f(t, x), t0, x0, h
Botón "Calcular"	Ejecuta runge_kutta_4()
Tabla de resultados	ttk.Treeview con columnas: Iter, t_k, x_k, k1, k2, k3, k4, x_(k+1)

Gráfica	Línea con marcadores, ejes etiquetados (t vs x), cuadrícula activada

6. Flujo de Datos

1. El usuario ingresa los datos.
2. Al presionar "Calcular":
    - Se limpian tabla y gráfica.
    - Se convierte f(t, x) a función ejecutable (lambdify).
    - Se itera con RK4 hasta alcanzar tf = x0.
    - Cada paso registra los cuatro k_i y el nuevo valor x_{k+1}.
3. Se grafica la solución aproximada.

7. Manejo de Errores

- Todo el cuerpo de runge_kutta_4() está envuelto en un bloque try-except.
- Errores comunes capturados:
    - Expresión inválida en f(t, x) → sympify falla.
    - Paso h = 0 → se lanza ValueError.
    - Entradas no numéricas → float() falla.
    - Evaluación de función no definida (ej. log(-1), división por cero).

8. Conclusión

Esta aplicación es una implementación clara y educativa del método RK4, destacando su estructura de cuatro pendientes y ofreciendo transparencia total en los cálculos intermedios.

La integración de SymPy, NumPy, Tkinter y Matplotlib demuestra un buen uso de la pila científica de Python, ideal para prototipado rápido en contextos académicos.