

Facultad de Ingeniería, Universidad Mariano Gálvez de Guatemala

Código: 013

Hugo Montoya



### **Proyecto Final**

Plan fin de semana sábados

Sección “G”

Josep Abisai Molina Chub	5190-23-15209
Roxana Armenia López Espina	5190-24-19977
Wilber José Pablo Garcia Reyes	5190-23-23594
Julio Cesar Torres del Cid	5190-23-12922
Flor Clariza Miranda Ruiz	5190-24-890

octubre de 2025

**Tabla de Contenido**

Resumen.....	3
Introducción .....	4
Planeación y estrategia.....	5
Desarrollo del proyecto.....	6
Como Funciona la calculadora.....	7
Ejemplo practico .....	10
Ejemplo de cómo ingresar una función a la calculadora .....	13
Como se llevó a cabo cada reunión de equipo.....	14
Líneas del código .....	16
Bibliografía .....	131

## Resumen

Este proyecto se basa en la investigación, elaboración y práctica de un instrumento que utilizamos constantemente en nuestras vidas cotidiana para realizar diversas acciones o procedimientos. Dicho instrumento está estrechamente relacionado con algunos de los temas vistos en la clase de cálculo, los cuales nos permiten comprender fenómenos que ocurren a nuestro alrededor, incluso sin darnos cuenta.

A lo largo del semestre, hemos estudiado diferentes conceptos fundamentales, entre ellos; las integrales definidas e indefinidas, la integración por método de sustitución (U), la integración por partes, las integrales impropias, la integración por fracciones parciales, así como aplicaciones del cálculo en movimiento rectilíneo, volúmenes y superficies de revolución.

El objetivo principal de este proyecto es demostrar como las integrales están presentes en múltiples situaciones cotidianas, muchas veces sin que seamos plenamente conscientes de ello. A través de la investigación y el desarrollo práctico, buscamos identificar y explicar de que manera los principios del cálculo especialmente los métodos de integración se manifiestan en actividades comunes, en el diseño de herramientas, en el movimiento de objetos o en la determinación de áreas y volúmenes en el entorno físico.

Cada integrante da una idea en el cual el líder lo ve y coincide en varios aspectos para tomar una en cuenta y ejecutar al mismo tiempo teniendo uso de tecnología y métodos en el que grupo se reúne y colaborar en desarrollar las ideas.

*Palabras clave:* procedimientos, fenómenos, Integrales.

## Introducción

El presente proyecto tiene como finalidad mostrar el proceso de elaboración de una calculadora, desarrollada con los conocimientos adquiridos en el curso de cálculo. A través de este trabajo se busca aplicar los conocimientos aplicados adquiridos en clase de manera práctica, funcional y útil en distintos contextos.

En el desarrollo del proyecto se explicar paso a paso como se llevó a cabo la creación de la calculadora, desde la planeación y el diseño hasta su programación y funcionamiento. Así mismo se incluirán ejemplos y situaciones de la vida en las que el uso de la calculadora resulta esencial, con el propósito de que el lector o usuario pueda comprender su importancia y aplicabilidad en tareas cotidianas, académicas y profesionales.

De esta manera el proyecto no solo busca reforzar los aprendizajes obtenidos en clase sino también fomentar la creatividad, el razonamiento lógico y la resolución de problemas mediante la implementación de herramientas tecnológicas. En conjunto, este demostrar como la programación puede ofrecer soluciones simples y efectivas a necesidades reales.

## Planeación y estrategia

Para iniciar este trabajo en equipo, fue necesario realizar una planeación organizada que nos permitiera definir claramente como comenzar. En primer lugar, elaboramos una lista con los aspectos más importantes que debíamos considerar, así como los elementos que debía incluir la calculadora. Cada integrante del grupo propuso un tema relacionado con el proyecto, y la asignación de tareas se llevo a cabo tomando en cuenta la disposición, los conocimientos previos y el tiempo que cada uno podía dedicar al trabajo.

Tras una discusión conjunta, el grupo decidió enfocar el proyecto en el desarrollo de una calculadora capaz de resolver distintos tipos de problemas matemáticos, centrándose en el cálculo de integrales. Este enfoque nos permitido aplicar los conceptos aprendidos en clase de manera práctica, integrando tanto la parte teórica del calculo como las herramientas tecnológicas necesarias para su implementación.

Durante la fase de planificación, también se evaluaron diferentes software y entornos e programación que podría facilitar el desarrollo del proyecto. Finalmente, se opto por utilizar Python, debido a su naturaleza dinámica, su sintaxis sencilla y la amplia variedad de librerías especializadas que ofrece, las cuales resultan muy útiles para la resolución de operaciones matemáticas avanzadas y para la creación de interfaces interactivas.

Gracias a esta elección, el grupo pudo estructurar el trabajo de manera eficiente, combinando el aprendizaje colaborativo con la aplicación practica de los conocimientos en programación y matemáticas.

## Desarrollo del proyecto

El proyecto se inició utilizando el lenguaje de programación Python en el cual desarrollaríamos la calculadora utilizando librerías como Os que nos servirá para interactuar con el sistema operativo, en el proyecto se puede utilizar para abrir o guardar archivos con resultados o configuraciones, así mismo se utilizó la librería “**math**” el cual tiene funciones básicas como raíces cuadradas, en el proyecto se utilizara para realizarse cálculos numéricos simples o validar expresiones matemáticas, también se utilizara librerías científicas y matemáticas además de la mencionada anteriormente, una de las principales es **numpy(np)** el cual nos ayudara para manejar los datos de las funciones y los puntos gráficos, el **sympy(SP)** nos ayudara a resolver expresiones matemáticas de forma algebraica, también se utilizaron librerías de visualización las cuales nos permitirá visualizar las funciones graficadas, y luego están las interfaz grafica que nos ayudara a crear la ventana principal de la calculadora con campos de entrada y botones de cálculo, mejorar la apariencia de la interfaz, y los pasos o ventanas de error. Estas librerías trabajan en conjunto para crear una calculadora de integrales interactiva, en resumen, la calculadora será capaz de calcular integrales simbólicas y numéricas, mostrará los resultados de manera visual y ofrecerá una interfaz amigable e interactiva para el usuario.

Las funciones definirán cada procedimiento matemático que hará la calculadora, cada una tiene parámetros y orden para que pueda lograr esos cálculos, en la siguiente parte del código se ve como se desarrolla la interfaz gráfica de la Calculadora, en donde se definirá la velocidad y colores que llevara la calculadora, luego están las utilidades matemáticas en el cual se definió los símbolos, así como la constante física en donde también se utiliza la AnimationSpeed el cual define constante de velocidades para animaciones, esto permite que distintas partes del programa

controlen la velocidad de actualización visual cada valor representa milisegundos entre fotogramas.

MathEngine está librería actúa como el cerebro matemático del programa sirve para derivar, integrar y evaluar funciones simbólicas y numéricas, con pasos explicativos y verificación numérica, interpreta expresiones escritas por el usuario, mostrar paso a paso como se deriva o integra, verificar numéricamente los resultados, reutilizar cálculos gracias al sistema de cache. Esto es un resumen breve de lo que hace el código y como se desarrolló.

### **Como Funciona la calculadora**

La calculadora ULTRA CALC 2D es una aplicación interactiva diseñada para realizar distintos tipos de cálculos matemáticos y representar gráficamente sus resultados. En su pantalla principal, se puede observar una interfaz moderna y funcional que integra varias secciones orientadas a facilitar el uso la compresión de los procedimientos matemáticos.

En la parte central se encuentra una zona gráfica, que representa una porción de un plano cartesiano. Esta sección, denominada Grafica Interactiva, permite visualizar de manera dinámica las funciones introducidas por el usuario, mostrando sus curvas, puntos de intersección y áreas bajo la curva en el caso de las integrales, contribuyendo a una mejor compresión de los conceptos matemáticos.

Ala derecha de la interfaz, se localiza una consola de resultados, en el cual se muestran de forma detallada de los pasos seguidos durante el proceso de resolución, así como los resultados finales de los problemas matemáticos planteados. Esta sección cumple una función educativa importante, ya que permite al usuario observar el desarrollo simbólico y numérico de los cálculos.

Debajo de la consola se encuentra dos botones principales: uno permite exportar los resultados obtenidos (por ejemplo, guardarlos en un archivo externo), y el otro posibilita almacenar los datos dentro de carpetas locales del sistema. Estas funciones ofrecen flexibilidad para conservar, compartir o analizar posteriormente la información generada por la calculadora.

La aplicación incluye una función de ejemplo preestablecida, que el usuario puede utilizar para probar la visualización de la gráfica y familiarizarse con las distintas herramientas disponibles, así mismo, el menú principal ofrece diversas opciones de trabajo entre las cuales destacan:

- Función Principal: permite integrar una función personalizada para su análisis y representación gráfica.
- Funciones 3D: habilita el cálculo y la visualización de funciones tridimensionales ampliando la capacidad de análisis.
- Cálculo de integrales: resuelve integrales definidas e indefinidas, tanto simbólica como numéricamente, además de mostrar el área bajo la curva.
- Problemas de física: ajustar la visualización de la gráfica, los colores, la velocidad de cálculo y otros parámetros de funcionamiento.

En conjunto, ULTRA CALC 2D se representa como una herramienta integral que combina el cálculo simbólico y numérico con una interfaz visual intuitiva, orientada al aprendizaje, la práctica y la experimentación de los conceptos matemáticos vistos en clase.

## Figura 1

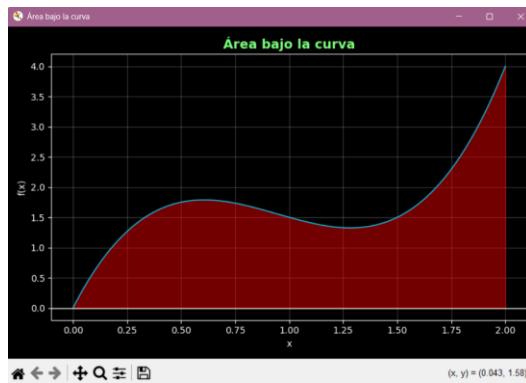
*Vista de la calculadora*



Nota. Interfaz principal de la calculadora.

## Figura 2

*Vista del cálculo área bajo la curva*



Nota: la figura muestra la gráfica de como calcular el área bajo la curva.

### Figura 3

Vista de la consola de los pasos y resultados

The screenshot shows a software interface for solving mathematical problems, specifically for calculating the area under a curve. The title bar says 'Pasos y Resultados - Área & Críticos'. The main content area is titled 'Pasos y Resultados - Área & Críticos' and contains the following steps:

- 1) Interpretación de la función  
 $f(x) = 3x^2 - 8.5x + 7x$
- 2) Objetivo  
Calcular el área bajo la curva en el intervalo [0, 2].  
- Área positiva: porciones donde  $f(x) \geq 0$   
- Área negativa: porciones donde  $f(x) < 0$  (se toma su valor absoluto)  
- Área neta: positiva - negativa
- 3) Método simbólico (si es posible)  
Integral simbólica:  $\int f(x) dx$  en  $[0, 2]$  = 3.333333333333333
- 4) Verificación numérica (más robusta)  
Aproximación numérica (quad): 3.3333333333 ± 3.70e-14  
Área positiva (sumada): 3.3333333350  
Área negativa (sumada en valor absoluto): 0.0000000000

Nota: la figura muestra los pasos del cálculo de la función.

### Ejemplo práctico

#### Viaje en automóvil

Escenario: Un automóvil prueba parte desde el reposo ( $t=0$ ) en la ciudad A y se dirige por una autopista recta hacia la ciudad B. Los ingenieros han determinado que la aceleración del automóvil está modelada por la función  $(t) = 6t + 2 \frac{m^2}{s}$  donde T es tiempo en segundos.

Resolver:

- Velocidad: determine la función de velocidad,  $v(t)$ , del automóvil
- Posición: si la ciudad B está a 1000 metros de distancia de la ciudad A,  
¿Cuánto tiempo, T, tarda el automóvil en llegar a la ciudad B?

Resolución de problema:

1. Determinación de la función de velocidad ( $v(t)$ )

Recordemos que la velocidad es la integral de la aceleración

$$v(t) = \int a(t)dt$$

Sustituimos la función de aceleración

$$v(t) = \int (6t + 2)dt$$

$$v(t) = \frac{6t^2}{2} + 2t + c_1$$

$$v(t) = 3t^2 + 2t + c_1$$

Para encontrar la constante de integración ( $c_1$ ), usamos la condición inicial: el automóvil parte del reposo en  $t = 0$ , lo que significa  $v(0) = 0$

$$v(0) = 3(0)^2 + 2(0) + c_1 = 0$$

$$c_1 = 0$$

Por lo tanto, la función de velocidad es:

$$v(t) = 3t^2 + 2t \text{ m/s}$$

## 2. Determinación del tiempo de Llegada (T)

La posición (o distancia recorrida) es la integral de la velocidad. Para encontrar el tiempo que tarda en recorrer los 100 metros, utilizaremos una integral definida de la velocidad desde el tiempo inicial ( $t = 0$ ) hasta el tiempo final ( $t = T$ )

$$\text{Distancia} = \int_0^t v(t)dt$$

Sustituimos la función de velocidad e igualamos a la distancia total (1000 m)

$$\int_0^T (3t^2 + 2t)dt = 1000$$

Calculamos la antiderivada:

$$\left[ \frac{3t^2}{3} + \frac{2t^2}{3} \right]_0^T = 1000$$

$$[t^3 + t^2]_0^T = 1000$$

$$(T^3 + T^2) - (0^3 + 0^2) = 1000$$

$$T^3 + T^2 = 1000$$

Esta es una ecuación cubica que debe resolverse para T. Busquemos un valor entero cercano.

- si  $T = 9: 9^3 + 9^2 = 729 + 81 = 810$  (*muy bajo*)
- si  $T = 10: 10^3 + 10^2 = 1000 + 100 = 1100$  (*muy alto*)

El valor de T esta entre 9 y 10 segundos. Usando métodos numéricos o factorización (si se factoriza la ecuación como  $T^3 + T^2 - 100 = 0$ ), se encuentra que el valor real es aproximadamente T= 9.61 segundos.

conclusión: El problema muestra como las integrales se usan para reconstruir información (velocidad a partir de aceleración, y posición a partir de velocidad).

La velocidad del automóvil se modela con

$$v(t) = 3t^3 + 2t$$

- El automóvil tarda aproximadamente 9.61 segundos em recorrer la distancia de 1000 metros y llegar a la ciudad B

### Ejemplo de cómo ingresar una función a la calculadora

Problemas para resolver:

Calcular el área bajo la curva de la función

$$F(x) = x^2 + 2x + 1$$

En el intervalo [0,3]

Ingresamos la función  $x^{**}2+2*x +1$  (en Python, los exponentes se escriben con  $**$  (por ejemplo,  $x^{**}2 = x^2$ ))

Selecciona el tipo de cálculo, la opción “Calculo de integrales” en el menú principal, luego elige “Integrales definida” e ingresa los límites de integración

- Límite inferior
- Límite superior

Visualización en la grafica interactiva

Presionar el botón de calcular

La **gráfica interactiva** mostrará:

- La curva de la función  $f(x) = x^2 + 2x + 1$ .
- El **área sombreada bajo la curva** entre  $x = 0$  y  $x = 3$ .
- Los ejes del plano cartesiano y los puntos de intersección.

Con este procedimiento, el usuario puede visualizar y comprender cómo el área bajo la curva representa el valor de la integral definida.

ULTRA CALC 2D no solo ofrece el resultado numérico, sino también el desarrollo simbólico y una representación gráfica clara e interactiva, ideal para el aprendizaje y la práctica del cálculo integral.

### **Como se llevó a cabo cada reunión de equipo.**

El desarrollo del proyecto resultó un proceso exigente pero enriquecedor. Aunque en algunos momentos el trabajo fue pesado debido a la complejidad de las tareas y la coordinación entre los integrantes, también se tornó ligero gracias a la colaboración constante, la buena organización y la disposición del grupo para alcanzar los objetivos propuestos.

Para lograr una comunicación efectiva, fue necesario realizar varias reuniones de coordinación. Se emplearon diferentes medios digitales, entre ellos **Google Meet**, que permitió la realización de **reuniones virtuales** donde se discutieron avances, se resolvieron dudas y se distribuyeron las responsabilidades. Asimismo, se utilizó **WhatsApp** como un canal de comunicación rápida y práctica, lo que facilitó el seguimiento de las tareas, la toma de decisiones y la coordinación en tiempo real.

Cada integrante del grupo aportó ideas valiosas y participó activamente en la construcción del proyecto. Durante las reuniones se expusieron distintos puntos de vista y se debatieron los temas a desarrollar, fomentando así el trabajo colaborativo y el pensamiento crítico. Esta dinámica permitió seleccionar de manera consensuada las funciones y características que formarían parte de la calculadora **ULTRA CALC 2D**.

Además, se implementó el uso de la plataforma **GitHub**, la cual fue fundamental para **compartir y mantener actualizado el código del proyecto**. A través de esta herramienta, los integrantes pudieron trabajar de manera conjunta, subir versiones del programa y revisar los aportes de cada miembro, asegurando una adecuada gestión del desarrollo y evitando conflictos en el código.

Finalmente, también se mantuvo una **comunicación presencial durante las clases**, lo que permitió discutir detalles técnicos, resolver dudas directamente con los compañeros y

fortalecer la coordinación general del grupo. Esta combinación de trabajo virtual y presencial fue clave para el éxito del proyecto, garantizando una colaboración efectiva, organizada y productiva.

### Figura 3

*Vista de las reuniones en grupo*



Nota: la figura muestra un grupo reunido por un canal de mensajería rápida (WhatsApp)

### **Conclusiones:**

El análisis desarrollado a lo largo de este trabajo reafirma que la calculadora de integrales trasciende su papel como simple herramienta de verificación, consolidándose como un **activo estratégico** en la formación y práctica de la Ingeniería. Hemos comprobado que, al automatizar la resolución de operaciones matemáticas que consumen mucho tiempo, como la integración de funciones complejas —esenciales para modelos cinemáticos o de sistemas dinámicos—, estas plataformas permiten una **gestión del tiempo significativamente más eficiente**.

En síntesis, la integración de estas calculadoras en el flujo de trabajo no solo minimiza errores, sino que **potencia la capacidad analítica y la experimentación rápida** con diferentes parámetros del modelo. La habilidad para aplicar el Cálculo a problemas de la vida real, como determinar la posición a partir de la aceleración, es la verdadera meta; la calculadora de integrales es, simplemente, la **forma más efectiva y profesional de alcanzarla**. Es una herramienta indispensable en el *kit* del ingeniero moderno.

### **Líneas del código**

```
import math
import os
import numpy as np
import sympy as sp
from sympy import Symbol
```

```
from sympy.core.sympify import SympifyError
import matplotlib
try:
    matplotlib.use("TkAgg")
except Exception:
    pass
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk
from matplotlib import animation
import tkinter as tk
from tkinter import ttk, scrolledtext, filedialog, messagebox, colorchooser
import time
from PIL import Image, ImageTk
import sys
import scipy.integrate as sci_integrate
from scipy.optimize import fsolve
import matplotlib.patches as patches
from matplotlib.patches import Polygon
from matplotlib.collections import PatchCollection

# ----- ESTILOS VISUALES MEJORADOS -----
plt.style.use('dark_background') # Estilo moderno por defecto
```

```

# Manual implementation of numerical derivative to replace deprecated
scipy.misc.derivative

def num_deriv(f, x, dx=1e-6, n=1):
    """
    Numerical derivative approximation.

    f: function to differentiate
    x: point to evaluate at
    dx: step size
    n: order of derivative (1 for first, 2 for second, etc.)
    """

    if n == 1:
        return (f(x + dx) - f(x - dx)) / (2 * dx)
    elif n == 2:
        return (f(x + dx) - 2 * f(x) + f(x - dx)) / dx**2
    else:
        # For higher orders, recurse
        return num_deriv(lambda t: num_deriv(f, t, dx=dx, n=1), x, dx=dx, n=n-1)

# ----- CONFIGURACIÓN GLOBAL MEJORADA -----
class AppConfig:
    DARK_THEME = {
        "bg": "#0a0f1c",

```

```
"card_bg": "#131a2c",
"accent": "#00d4ff",
"accent_secondary": "#ff6b9d",
"accent_tertiary": "#7eff7e",
"text_primary": "#ffffff",
"text_secondary": "#b0b8d1",
"text_muted": "#8a93b0",
"success": "#00ffaa",
"warning": "#ffd166",
"error": "#ff6b6b",
"border": "#1e2a45",
"button_bg": "#1a243b",
"button_hover": "#243355",
"entry_bg": "#0d1424",
"plot_bg": "#0b1220",
"plot_grid": "#1a2a4a"
}
```

```
LIGHT_THEME = {
"bg": "#f0f4ff",
"card_bg": "#ffffff",
"accent": "#0066ff",
"accent_secondary": "#ff3366",
"accent_tertiary": "#00cc66",
```

```
"text_primary": "#1a1a2e",
"text_secondary": "#4a5568",
"text_muted": "#718096",
"success": "#00b894",
"warning": "#fdcb6e",
"error": "#e84393",
"border": "#e2e8f0",
"button_bg": "#e2e8f0",
"button_hover": "#cbd5e0",
"entry_bg": "#ffffff",
"plot_bg": "#ffffff",
"plot_grid": "#e2e8f0"

}

ANIMATION_SPEEDS = {
    "slow": 100,
    "normal": 50,
    "fast": 20,
    "instant": 0
}

GRADIENT_COLORS = [
    '#00d4ff', '#ff6b9d', '#7eff7e', '#ffd166', '#9d4edd',
    '#ff9e64', '#4cc9f0', '#f72585', '#7209b7', '#3a86ff
]
```

```
# ----- UTILIDADES MATEMÁTICAS (MEJORADAS) -----
```

```
--
```

```
X = sp.Symbol('x')
```

```
Y = sp.Symbol('y')
```

```
T = sp.Symbol('t')
```

```
# Constantes físicas
```

```
GRAVEDAD_TIERRA = 9.80665 # m/s2 más preciso
```

```
# Configuración global de velocidad
```

```
class AnimationSpeed:
```

```
    SLOW = 100 # ms entre frames
```

```
    NORMAL = 50 # ms entre frames
```

```
    FAST = 20 # ms entre frames
```

```
    INSTANT = 0 # Sin animación
```

```
def parse_limit_string(s):
```

```
    """Parsea un string de límite: 'inf', 'oo', '-inf', '-oo', o número."""
```

```
    s = str(s).strip()
```

```
    if s == ":
```

```
        raise ValueError("El límite no puede ser una cadena vacía.")
```

```
    sl = s.lower()
```

```

if sl in ('inf', 'oo', '+inf', '+oo'):

    return sp.oo

if sl in ('-inf', '-oo'):

    return -sp.oo

try:

    return float(s)

except ValueError:

    try:

        return sp.sympify(s)

    except SympifyError:

        raise ValueError(f'Límite inválido: '{s}'. Use números, 'inf', '-inf', 'pi', o 'E'.')

```

```

def safe_sympify(expr_str, symbol_char='x'):

    """Parsea la expresión ingresada por el usuario a SymPy de forma segura."""

    if not str(expr_str).strip():

        raise ValueError("La expresión de la función no puede estar vacía.")

    s = str(expr_str).replace('^', '**')

    local = {

        'x': Symbol('x'),

        'y': Symbol('y'),

        'sin': sp.sin, 'cos': sp.cos, 'tan': sp.tan,

        'asin': sp.asin, 'acos': sp.acos, 'atan': sp.atan,

        'csc': sp.csc, 'sec': sp.sec, 'cot': sp.cot,

```

```

'sinh': sp.sinh, 'cosh': sp.cosh, 'tanh': sp.tanh,
'asinh': sp.asinh, 'acosh': sp.acosh, 'atanh': sp.atanh,
'exp': sp.exp, 'log': sp.log, 'ln': sp.log,
'sqrt': sp.sqrt, 'pi': sp.pi, 'E': sp.E,
'abs': sp.Abs, 'factorial': sp.factorial

}

try:

    expr = sp.sympify(s, locals=local)

    return expr

except (SympifyError, SyntaxError) as e:

    raise ValueError(f"Error de sintaxis en la expresión: '{expr_str}'.\n\nCausa:
{e}\n\nRevise la sintaxis. Use '*' para multiplicar (ej. 2*x) y '**' para potencias (ej. x**2).")

except Exception as e:

    raise ValueError(f"Error desconocido al procesar la expresión: {e}")

```

# ----- FÍSICA ENGINE MEJORADO CON MÉTODOS AVANZADOS --

---

```

class PhysicsEngine:

    def __init__(self):

        self.g = GRAVEDAD_TIERRA

        self._cache = {}



    def _cache_key(self, method, *args):

```

```

    return f'{method}_("'.join(map(str, args))}"
```

```

def caida_libre_tiempo(self, altura, velocidad_inicial=0, densidad_aire=1.225,
                      coeficiente_arrastre=0.5, area=0.1, masa=1.0):
    """Caída libre con resistencia del aire usando EDOs"""

    cache_key = self._cache_key("caida_tiempo", altura, velocidad_inicial,
                               densidad_aire, coeficiente_arrastre, area, masa)

    if cache_key in self._cache:
        return self._cache[cache_key]

    try:
        # Sin resistencia del aire (solución analítica)
        if velocidad_inicial == 0:
            tiempo_simple = math.sqrt(2 * altura / self.g)
        else:
            a = 0.5 * self.g
            b = velocidad_inicial
            c = -altura
            discriminante = b**2 - 4*a*c
            if discriminante < 0:
                raise ValueError("No hay solución real para estos parámetros")
            tiempo_simple = (-b + math.sqrt(discriminante)) / (2*a)
            if tiempo_simple < 0:

```

```

tiempo_simple = (-b - math.sqrt(discriminante)) / (2*a)

# Con resistencia del aire (solución numérica)

def derivada_caida(t, y):
    # y[0] = posición, y[1] = velocidad

    fuerza_gravedad = masa * self.g
    fuerza_arrastre = 0.5 * densidad_aire * coeficiente_arrastre * area * y[1]**2

    if y[1] >= 0:
        fuerza_arrastre = -fuerza_arrastre
    aceleracion = (fuerza_gravedad + fuerza_arrastre) / masa
    return [y[1], aceleracion]

# Resolver EDO

from scipy.integrate import solve_ivp
sol = solve_ivp(derivada_caida, [0, tiempo_simple**2],
                [altura, -velocidad_inicial],
                method='RK45', dense_output=True, rtol=1e-6, atol=1e-6)

# Encontrar tiempo de impacto

def altura_cero(t):
    return sol.sol(t)[0]

tiempo_con_resistencia = fsolve(altura_cero, tiempo_simple)[0]

```

```

velocidad_impacto = sol.sol(tiempo_con_resistencia)[1]

result = {
    'tiempo_sin_resistencia': tiempo_simple,
    'tiempo_con_resistencia': tiempo_con_resistencia,
    'velocidad_impacto': abs(velocidad_impacto),
    'energia_cinetica': 0.5 * masa * velocidad_impacto**2,
    'altura': altura,
    'velocidad_inicial': velocidad_inicial,
    'diferencia_tiempo': tiempo_con_resistencia - tiempo_simple
}

self._cache[cache_key] = result
return result

except Exception as e:
    raise ValueError(f'Error en cálculo de caída libre: {e}')

```

```

def movimiento_proyectil(self, velocidad, angulo, altura_inicial=0,
                        densidad_aire=1.225, coeficiente_arrastre=0.5,
                        area=0.01, masa=0.1):

    """Movimiento de proyectil con resistencia del aire"""

    angulo_rad = math.radians(angulo)

    vx0 = velocidad * math.cos(angulo_rad)

```

```

vy0 = velocidad * math.sin(angulo_rad)

def derivadas_proyectil(t, y):
    # y[0] = x, y[1] = y, y[2] = vx, y[3] = vy
    v = math.sqrt(y[2]**2 + y[3]**2)
    fuerza_arrastre = 0.5 * densidad_aire * coeficiente_arrastre * area * v**2

    ax = - (fuerza_arrastre / masa) * (y[2] / v) if v > 0 else 0
    ay = -self.g - (fuerza_arrastre / masa) * (y[3] / v) if v > 0 else -self.g

    return [y[2], y[3], ax, ay]

from scipy.integrate import solve_ivp

sol = solve_ivp(derivadas_proyectil, [0, 10],
                [0, altura_inicial, vx0, vy0],
                method='RK45', dense_output=True,
                events=lambda t, y: y[1], rtol=1e-6, atol=1e-6) # Evento cuando y=0

(tiempo_vuelo = sol.t_events[0][0] if sol.t_events[0].size > 0 else sol.t[-1]
 alcance = sol.sol(tiempo_vuelo)[0]
 altura_maxima = np.max(sol.sol(sol.t)[1])

```

```

return {

    'alcance': alcance,
    'altura_maxima': altura_maxima,
    'tiempo_vuelo': tiempo_vuelo,
    'velocidad_inicial': velocidad,
    'angulo': angulo,
    'trayectoria': sol
}

```

```

def caida_libre_velocidad(self, velocidad_final, velocidad_inicial=0):
    cache_key = self._cache_key("caida_velocidad", velocidad_final, velocidad_inicial)
    if cache_key in self._cache:
        return self._cache[cache_key]

```

try:

```

    altura = (velocidad_final**2 - velocidad_inicial**2) / (2 * self.g)
    tiempo = (velocidad_final - velocidad_inicial) / self.g

```

```

result = {
    'altura': altura,
    'tiempo': tiempo,
    'velocidad_final': velocidad_final,
    'velocidad_inicial': velocidad_inicial
}
```

```

    }

self._cache[cache_key] = result

return result

except Exception as e:

    raise ValueError(f'Error en cálculo de altura por velocidad: {e}')

def movimiento_rectilineo(self, velocidad_inicial, aceleracion, tiempo):

    cache_key = self._cache_key("movimiento", velocidad_inicial, aceleracion, tiempo)

    if cache_key in self._cache:

        return self._cache[cache_key]

try:

    desplazamiento = velocidad_inicial * tiempo + 0.5 * aceleracion * tiempo**2

    velocidad_final = velocidad_inicial + aceleracion * tiempo

# Añadir análisis de energía (asumiendo masa=1)

    energia_cinetica_inicial = 0.5 * velocidad_inicial**2

    energia_cinetica_final = 0.5 * velocidad_final**2

    trabajo = aceleracion * desplazamiento # Trabajo de la fuerza neta

result = {

    'desplazamiento': desplazamiento,
}

```

```

'velocidad_final': velocidad_final,
'velocidad_inicial': velocidad_inicial,
'aceleracion': aceleracion,
'tiempo': tiempo,
'energia_cinetica_inicial': energia_cinetica_inicial,
'energia_cinetica_final': energia_cinetica_final,
'trabajo': trabajo,
'espacio_fase': {
    'posiciones': np.linspace(0, desplazamiento, 100),
    'velocidades': velocidad_inicial + aceleracion * np.linspace(0, tiempo, 100)
}
}

self._cache[cache_key] = result
return result

except Exception as e:
    raise ValueError(f'Error en cálculo de movimiento rectilíneo: {e}')

def velocidad_promedio(self, distancia_total, tiempo_total):
    cache_key = self._cache_key("velocidad_promedio", distancia_total, tiempo_total)
    if cache_key in self._cache:
        return self._cache[cache_key]

```

try:

```
    velocidad_promedio = distancia_total / tiempo_total
```

```
# Añadir conversión a km/h y mph para nivel ingeniería
```

```
    kmh = velocidad_promedio * 3.6
```

```
    mph = velocidad_promedio * 2.23694
```

```
result = {
```

```
    'velocidad_promedio': velocidad_promedio,
```

```
    'kmh': kmh,
```

```
    'mph': mph,
```

```
    'distancia_total': distancia_total,
```

```
    'tiempo_total': tiempo_total
```

```
}
```

```
self._cache[cache_key] = result
```

```
return result
```

except Exception as e:

```
    raise ValueError(f'Error en cálculo de velocidad promedio: {e}')
```

```
def analisis_energia_cinetica(self, masa, velocidad):
```

```
    """Análisis completo de energía cinética"""

```

```
    energia = 0.5 * masa * velocidad**2
```

```

momento_lineal = masa * velocidad

presion_impacto = momento_lineal / 0.01 # Asumiendo área de impacto de 0.01 m2

return {

    'energia_cinetica': energia,

    'momento_lineal': momento_lineal,

    'presion_impacto': presion_impacto,

    'velocidad_relativista': self._correccion_relativista(velocidad, masa),

    'equivalente_tnt': energia / 4.184e6 # Equivalente en kg de TNT

}

```

```

def _correccion_relativista(self, velocidad, masa):

    """Corrección relativista para velocidades altas"""

    c = 299792458 # Velocidad de la luz

    if velocidad > 0.1 * c:

        gamma = 1 / math.sqrt(1 - (velocidad/c)**2)

        return gamma * masa

    return masa

```

```

def resistencia_aire_aproximada(self, masa, area, coeficiente_arrastre=0.47, #

Coeficiente más realista para esfera

densidad_aire=1.225, velocidad=10): # Densidad a nivel del mar

```

```

cache_key = self._cache_key("resistencia", masa, area, coeficiente_arrastre,
densidad_aire, velocidad)

if cache_key in self._cache:
    return self._cache[cache_key]

try:
    fuerza_resistencia = 0.5 * densidad_aire * velocidad**2 * area *
coeficiente_arrastre

    velocidad_terminal = math.sqrt((2 * masa * self.g) /
(densidad_aire * area * coeficiente_arrastre))

# Simulación numérica simple para trayectoria con resistencia (usando SciPy
ODE)

def ode(t, y):
    pos, vel = y

    accel = -self.g - (0.5 * densidad_aire * vel**2 * area * coeficiente_arrastre /
masa) * np.sign(vel)

    return [vel, accel]

sol = sci_integrate.solve_ivp(ode, [0, 10], [0, 0], t_eval=np.linspace(0, 10, 50),
rtol=1e-6, atol=1e-6)

tiempo_terminal_approx = sol.t[-1] # Aproximación

```

```

result = {
    'fuerza_resistencia': fuerza_resistencia,
    'velocidad_terminal': velocidad_terminal,
    'peso': masa * self.g,
    'relacion_resistencia_peso': fuerza_resistencia / (masa * self.g),
    'tiempo_aprox_terminal': tiempo_terminal_approx,
    'trayectoria': sol.y[0] # Posiciones
}

```

```
self._cache[cache_key] = result
```

```
return result
```

```
except Exception as e:
```

```
    raise ValueError(f"Error en cálculo de resistencia del aire: {e}")
```

```
def clear_cache(self):
```

```
    self._cache.clear()
```

---

```
# ----- MATH ENGINE MEJORADO A NIVEL INGENIERÍA -----
```

---

```
class MathEngine:
```

```
    def __init__(self):
```

```
        self.x = X
```

```
        self.physics = PhysicsEngine()
```

```

self._cache = {}

def _cache_key(self, method, *args):
    return f'{method}_{_join(map(str, args))}'

def derivative(self, func_str, order=1, method="directo"):
    cache_key = self._cache_key("derivative", func_str, order, method)
    if cache_key in self._cache:
        return self._cache[cache_key]

    expr = safe_sympify(func_str, 'x')
    steps = [f"EXPRESIÓN ORIGINAL: f(x) = {sp.latex(expr)}"]

    if method == "directo":
        d = sp.diff(expr, self.x, order)
        steps.append(f"DERIVADA {order}ª (DIRECTA): f{'*' * order}(x) = {sp.latex(d)}")
        steps.append(f"SIMPLIFICADA: {sp.latex(sp.simplify(d))}")

    elif method == "regla_potencia":
        # Implementación manual para regla de potencia
        terms = expr.as_ordered_terms()
        d_terms = []
        for term in terms:

```

```

coeff, pow = term.as_coeff_pow(self.x)

if pow > 0:

    d_terms.append(coeff * pow * self.x**(pow-1))

d = sum(d_terms)

steps.append(f"APLICANDO REGLA DE POTENCIA A CADA TÉRMINO")

steps.append(f"DERIVADA: {sp.latex(d)}")

# Agregar más métodos manuales para cadena, producto, etc.

else:

    # Método completo paso a paso

    d = expr

    for i in range(order):

        d = sp.diff(d, self.x)

        steps.append(f"PASO {i+1}: DERIVADA PARCIAL = {sp.latex(d)}")

    steps.append(f"DERIVADA FINAL: {sp.latex(sp.simplify(d))}")

# Añadir análisis numérico para verificación

f_num = sp.lambdify(self.x, expr, 'numpy')

d_num = lambda x_val: num_deriv(f_num, x_val, dx=1e-6, n=order)

test_point = 1.0

num_val = d_num(test_point)

steps.append(f"VERIFICACIÓN NUMÉRICA EN x={test_point}: ≈ {num_val:.6f}")

```

```

result = (d, steps)

self._cache[cache_key] = result

return result


def eval_derivative(self, func_str, point, order=1, method="directo"):

    cache_key = self._cache_key("eval_derivative", func_str, point, order, method)

    if cache_key in self._cache:

        return self._cache[cache_key]

expr = safe_sympify(func_str, 'x')

d = sp.diff(expr, self.x, order)

try:

    val_sym = sp.N(d.subs(self.x, point))

    steps = [f'f(x) = {sp.latex(expr)}', f'DERIVADA {order}ª: {sp.latex(d)}',

f'EVALUADA EN x={point}: {val_sym}']

# Verificación numérica

f_num = sp.lambdify(self.x, expr, 'numpy')

val_num = num_deriv(f_num, point, dx=1e-8, n=order)

steps.append(f'VERIFICACIÓN NUMÉRICA: ≈ {val_num:.8f} (error relativo:

{abs(val_sym - val_num)/abs(val_sym):.2e})')

result = (val_sym, steps)

self._cache[cache_key] = result

```

```

        return result

    except Exception as e:

        result = (None, [f"No fue posible evaluar la derivada en {point}: {e}"])

        self._cache[cache_key] = result

        return result


def indefinite(self, func_str):

    cache_key = self._cache_key("indefinite", func_str)

    if cache_key in self._cache:

        return self._cache[cache_key]

    expr = safe_sympify(func_str, 'x')

    try:

        res = sp.integrate(expr, self.x)

        steps = [f"f(x) = {sp.latex(expr)}", f"\int f(x) dx = {sp.latex(res)} + C"]

        # Añadir verificación diferenciando

        diff_back = sp.diff(res, self.x)

        steps.append(f"VERIFICACIÓN: d/dx (resultado) = {sp.latex(diff_back)} (debe
coincidir con original)")

        result = (res, steps)

        self._cache[cache_key] = result

        return result

    except Exception as e:

```

```

result = (None, [f"No se pudo integrar simbólicamente: {e}"])
self._cache[cache_key] = result
return result

def definite(self, func_str, a, b):
    cache_key = self._cache_key("definite", func_str, a, b)
    if cache_key in self._cache:
        return self._cache[cache_key]

    expr = safe_sympify(func_str, 'x')
    try:
        res_sym = sp.integrate(expr, (self.x, a, b))
        steps = [f"f(x) = {sp.latex(expr)}", f"Intervalo: [{a}, {b}]",
                 f"RESULTADO SIMBÓLICO: {res_sym}"]
    except:
        pass
    else:
        # Añadir aproximación numérica con SciPy para precisión
        f_num = sp.lambdify(self.x, expr, 'numpy')
        res_num, err = sci_integrate.quad(f_num, a, b)
        steps.append(f"APROXIMACIÓN NUMÉRICA (QUAD): {res_num:.10f} ± {err:.2e}")

    result = (res_sym, steps)
    self._cache[cache_key] = result
    return result

```

```
except Exception as e:
```

```
    try:
```

```
        # Fallback a numérico si simbólico falla
```

```
        f_num = sp.lambdify(self.x, expr, 'numpy')
```

```
        res_num, err = sci_integrate.quad(f_num, a, b)
```

```
        steps = [f"NUMÉRICO (QUAD): {res_num:.10f} ± {err:.2e}"]
```

```
        result = (res_num, steps)
```

```
        self._cache[cache_key] = result
```

```
    return result
```

```
except Exception as e2:
```

```
    result = (None, [f"No se pudo calcular integral definida: {e2}"])
```

```
    self._cache[cache_key] = result
```

```
    return result
```

```
def improper(self, func_str, a, b):
```

```
    cache_key = self._cache_key("improper", func_str, a, b)
```

```
    if cache_key in self._cache:
```

```
        return self._cache[cache_key]
```

```
    expr = safe_sympify(func_str, 'x')
```

```
    steps = [f"f(x) = {sp.latex(expr)}", f"Intentando integral impropia en [{a}, {b}]"]
```

```
    try:
```

```
        res = sp.integrate(expr, (self.x, a, b))
```

```

steps.append(f"RESULTADO SIMBÓLICO: {res}")

# Verificación numérica con límites finitos

if sp.oo in (a, b) or -sp.oo in (a, b):

    finite_a = -1e6 if a == -sp.oo else a

    finite_b = 1e6 if b == sp.oo else b

    f_num = sp.lambdify(self.x, expr, 'numpy')

    res_num, err = sci_integrate.quad(f_num, finite_a, finite_b)

    steps.append(f"APROXIMACIÓN NUMÉRICA (FINITA): {res_num:.10f} ±

{err:.2e}")

result = (res, steps)

self._cache[cache_key] = result

return result

except Exception as e:

    steps.append(f"Error simbólico: {e}. Usando numérico.")

try:

    f_num = sp.lambdify(self.x, expr, 'numpy')

    res_num, err = sci_integrate.quad(f_num, a, b, limit=1000)

    steps.append(f"NUMÉRICO (QUAD IMPROPIO): {res_num:.10f} ±

{err:.2e}")

result = (res_num, steps)

except:

    result = (None, steps)

self._cache[cache_key] = result

```

```

    return result

def partial_fractions(self, func_str):
    cache_key = self._cache_key("partial_fractions", func_str)
    if cache_key in self._cache:
        return self._cache[cache_key]

    expr = safe_sympify(func_str, 'x')
    steps = [f"EXPRESIÓN ORIGINAL: {sp.latex(expr)}"]
    try:
        decomposed = sp.apart(expr, self.x)
        steps.append(f"DESCOMPOSICIÓN: {sp.latex(decomposed)}")
        integ = sp.integrate(decomposed, self.x)
        steps.append(f"INTEGRAL: {sp.latex(integ)} + C")
        # Verificación diferenciando
        diff_back = sp.diff(integ, self.x)
        steps.append(f"VERIFICACIÓN: d/dx = {sp.latex(diff_back)}")
        result = (integ, steps)
        self._cache[cache_key] = result
    except Exception as e:
        result = (None, [f"No es una fracción racional o error: {e}"])
    self._cache[cache_key] = result

```

```

    return result

def integration_by_parts_auto(self, func_str):
    cache_key = self._cache_key("integration_by_parts", func_str)
    if cache_key in self._cache:
        return self._cache[cache_key]

    expr = safe_sympify(func_str, 'x')
    steps = [f"EXPRESIÓN: {sp.latex(expr)}", "MÉTODO: POR PARTES (LIATE)"]
    try:
        from sympy.integrals.manualintegrate import integral_steps
        steps_obj = integral_steps(expr, self.x)
        # Extraer pasos detallados
        for rule in steps_obj:
            steps.append(f"PASO: {rule.rule} - u={rule.u}, dv={rule.dv}")
        result = sp.integrate(expr, self.x)
        steps.append(f"RESULTADO: {sp.latex(result)} + C")
        result_final = (result, steps)
        self._cache[cache_key] = result_final
    except Exception as e:
        result_final = (None, steps + [f"Error: {e}"])
        self._cache[cache_key] = result_final

```

```

    return result_final

def substitution_auto(self, func_str):
    cache_key = self._cache_key("substitution", func_str)
    if cache_key in self._cache:
        return self._cache[cache_key]

    expr = safe_sympify(func_str, 'x')
    steps = [f"FUNCIÓN: {sp.latex(expr)}", "SUSTITUCIÓN AUTOMÁTICA"]
    try:
        from sympy.integrals.manualintegrate import integral_steps
        steps_obj = integral_steps(expr, self.x)
        for rule in steps_obj:
            if hasattr(rule, 'substitution'):
                steps.append(f"SUSTITUCIÓN: u = {rule.substitution}")
        result = sp.integrate(expr, self.x)
        steps.append(f"RESULTADO: {sp.latex(result)} + C")
        result_final = (result, steps)
        self._cache[cache_key] = result_final
    except Exception as e:
        result_final = (None, steps + [f"Error: {e}"])
        self._cache[cache_key] = result_final

```

```

    return result_final

def trig_substitution(self, func_str):

    cache_key = self._cache_key("trig_substitution", func_str)

    if cache_key in self._cache:

        return self._cache[cache_key]

expr = safe_sympify(func_str, 'x')

steps = [f"FUNCIÓN: {sp.latex(expr)}", "SUSTITUCIÓN TRIGONOMÉTRICA"]

try:

    result = sp.integrate(expr, self.x)

    # Detectar tipo de sust. trig.

    if 'sqrt(x**2 + ' in str(expr):

        steps.append("TIPO: x = a tan(θ) o similar")

        steps.append(f"RESULTADO: {sp.latex(result)} + C")

    result_final = (result, steps)

    self._cache[cache_key] = result_final

    return result_final

except Exception as e:

    result_final = (None, steps + [f"Error: {e}"])

    self._cache[cache_key] = result_final

    return result_final

```

```

def numeric_simpson(self, func_str, a, b, n=5000): # Reduced n for performance

    cache_key = self._cache_key("numeric_simpson", func_str, a, b, n)

    if cache_key in self._cache:

        return self._cache[cache_key]

expr = safe_sympify(func_str, 'x')

f = sp.lambdify(self.x, expr, 'numpy')

if n % 2 != 0:

    n += 1

xs = np.linspace(float(a), float(b), n + 1)

try:

    with np.errstate(all='ignore'):

        ys = f(xs)

except Exception as e:

    raise ValueError(f"No se pudo evaluar la función numéricamente: {e}")

ys = np.nan_to_num(ys, nan=0.0, posinf=1e100, neginf=-1e100)

h = (float(b) - float(a)) / n

integral = (h / 3) * (ys[0] + 4 * np.sum(ys[1:-1:2]) + 2 * np.sum(ys[2:-2:2]) + ys[-1])

```

```

# Añadir error estimado (regla de Simpson)

error_est = - ( (b-a)**5 / (180 * n**4) ) * max(np.abs(ys)) # Aproximación burda


steps = [f"REGLA DE SIMPSON CON n={n} SUBINTERVALOS",
         f"RESULTADO APROXIMADO: {integral:.12f}",
         f"ERROR ESTIMADO: ≈ {error_est:.2e}"]

result = (float(integral), steps)

self._cache[cache_key] = result

return result


def critical_points(self, func_str, xlim=(-5, 5)):

    cache_key = self._cache_key("critical_points", func_str, xlim[0], xlim[1])

    if cache_key in self._cache:

        return self._cache[cache_key]

    expr = safe_sympify(func_str, 'x')

    fprime = sp.diff(expr, self.x)

    fsecond = sp.diff(expr, self.x, 2)

    fthird = sp.diff(expr, self.x, 3)

    crits = []

    try:

        # Solución simbólica

```

```

sols_sym = sp.solve(fprime, self.x)

sols_num = [float(s) for s in sols_sym if s.is_real and xlim[0] <= float(s) <=
            xlim[1]]


# Suplemento numérico con fsolve

fprime_num = sp.lambdify(self.x, fprime, 'numpy')

for guess in np.linspace(xlim[0], xlim[1], 50):

    root = fsolve(fprime_num, guess)[0]

    if xlim[0] <= root <= xlim[1] and abs(fprime_num(root)) < 1e-6:

        sols_num.append(root)


sols = sorted(list(set(np.round(sols_num, 8)))))

for s in sols:

    sx = float(s)

    yv = float(expr.subs(self.x, sx))

    sec_val = float(fsecond.subs(self.x, sx))

    if sec_val == 0:

        third_val = float(fthird.subs(self.x, sx))

        kind = 'Inflexión' if third_val != 0 else 'Indeterminado'

    else:

        kind = 'Mínimo local' if sec_val > 0 else 'Máximo local'

    crits.append({'x': sx, 'y': yv, 'tipo': kind, 'curvatura': sec_val})

except Exception:

```

```

    pass

    y_intercept = {'x': 0.0, 'y': float(expr.subs(self.x, 0)) if expr.subs(self.x, 0).is_real
else None, 'tipo': 'Intersección Y'}

    x_intercepts = []

    try:

        # Raíces simbólicas

        roots_sym = sp.solve(expr, self.x)

        roots_num = [float(r) for r in roots_sym if r.is_real and xlim[0] <= float(r) <=
xlim[1]]


        # Suplemento numérico

        f_num = sp.lambdify(self.x, expr, 'numpy')

        for guess in np.linspace(xlim[0], xlim[1], 50):

            root = fsolve(f_num, guess)[0]

            if xlim[0] <= root <= xlim[1] and abs(f_num(root)) < 1e-6:

                roots_num.append(root)

    roots = sorted(list(set(np.round(roots_num, 8)))))

    for r in roots:

        x_intercepts.append({'x': r, 'y': 0.0, 'tipo': 'Raíz'})

except Exception:

    pass

```

```

        result = {"puntos_criticos": crits, "interseccion_y": y_intercept, "raices":
x_intercepts}

        self._cache[cache_key] = result

        return result
    
```

```

def clear_cache(self):

    self._cache.clear()

    self.physics.clear_cache()

```

# ----- COMPONENTES DE INTERFAZ MODERNOS -----

--

```

class ModernButton(ttk.Button):

    def __init__(self, parent, **kwargs):
        style_name = kwargs.pop('style', 'Modern.TButton')
        super().__init__(parent, style=style_name, **kwargs)

```

```

class ModernEntry(ttk.Entry):

    def __init__(self, parent, **kwargs):
        super().__init__(parent, **kwargs)

```

```

class ModernCombobox(ttk.Combobox):

    def __init__(self, parent, **kwargs):
        super().__init__(parent, **kwargs)

```

```
# ----- CALCULADORA COMPACTA MEJORADA -----  
  
class CompactCalculatorPad(ttk.Frame):  
  
    def __init__(self, parent, entry_widget, theme):  
        super().__init__(parent, style="Card.TFrame")  
  
        self.entry = entry_widget  
  
        self.theme = theme  
  
  
    # Configurar grid para calculadora compacta  
  
    for i in range(6):  
        self.grid_rowconfigure(i, weight=1)  
  
    for i in range(5):  
        self.grid_columnconfigure(i, weight=1)  
  
  
    # Botones matemáticos compactos  
  
    math_buttons = [  
        ['sin()', 'cos()', 'tan()', 'π', 'e'],  
        ['log()', 'ln()', '√(', 'x²', 'x³'],  
        ['(', ')', '|x|', '1/x', 'n!'],  
        ['7', '8', '9', '/', '⊗'],  
        ['4', '5', '6', '*', 'C'],  
        ['1', '2', '3', '-', 'AC'],  
        ['0', '.', '=', '+', '**']
```

]

```
# Crear botones compactos

for r, row in enumerate(math_buttons):
    for c, text in enumerate(row):
        if text in [' $\otimes$ ', 'C', 'AC']:
            style = "Control.TButton"
        elif text in ['7', '8', '9', '4', '5', '6', '1', '2', '3', '0', '.', '=', '+', '-', '*', '/', '**']:
            style = "Num.TButton"
        else:
            style = "Math.TButton"

        btn = ModernButton(
            self,
            text=text,
            command=lambda t=text: self.on_press(t),
            style=style
        )
        btn.grid(row=r, column=c, sticky="nsew", padx=1, pady=1)

def get_display_text(self, text):
    symbol_map = {
        'sin()': 'sin', 'cos()': 'cos', 'tan()': 'tan',
```

```

'log(: 'log', 'ln(: 'ln', 'sqrt(: 'sqrt',
'pi': 'pi', 'e': 'e', '|x|': '|x|',
'x^2': 'x^2', 'x^n': 'x^n', '1/x': '1/x', 'n!': 'n!'
}

return symbol_map.get(text, text)

def on_press(self, value):
    if value == '=':
        try:
            # Evaluar la expresión actual
            expr = self.entry.get()
            result = str(sp.sympify(expr.replace('^', '**')))

            self.entry.delete(0, tk.END)
            self.entry.insert(0, result)

        except:
            messagebox.showerror("Error", "No se pudo evaluar la expresión")

    elif value == '⌫':
        cursor_pos = self.entry.index(tk.INSERT)
        if cursor_pos > 0:
            self.entry.delete(cursor_pos - 1, cursor_pos)

    elif value == 'C':
        self.entry.delete(0, tk.END)

    elif value == 'AC':

```

```
    self.entry.delete(0, tk.END)

else:

    self.entry.insert(tk.INSERT, value)

    self.entry.focus_set()
```

## # ----- GESTOR DE GRÁFICAS MEJORADO CON ANÁLISIS DE ÁREA DETALLADO -----

```
class ModernPlotManager:

    def __init__(self, fig, ax, canvas, theme):

        self.fig = fig

        self.ax = ax

        self.canvas = canvas

        self.theme = theme

        self.anim = None

        self.bands = []

        self.annotations = []

        self.animation_speed = AnimationSpeed.NORMAL

        self.animation_enabled = False # Desactivado por defecto

        self.show_velocity = True

        self.current_plot_type = None

        self.area_info = {}

        self.ax2 = None # Eje secundario para velocidad

        self._apply_modern_style()
```

```
def _apply_modern_style(self):  
    """Estilo moderno y atractivo para jóvenes"""  
  
    # Fondo con gradiente moderno  
  
    self.fig.patch.set_facecolor(self.theme['plot_bg'])  
  
    self.ax.set_facecolor(self.theme['plot_bg'])  
  
  
    # Estilo de ejes moderno  
  
    self.ax.tick_params(colors=self.theme['text_primary'], which='both', labelsize=10)  
  
    self.ax.xaxis.label.set_color(self.theme['text_primary'])  
  
    self.ax.yaxis.label.set_color(self.theme['text_primary'])  
  
    self.ax.title.set_color(self.theme['accent'])  
  
    self.ax.title.set_fontweight('bold')  
  
    self.ax.title.set_fontsize(14)  
  
  
    # Ejes con estilo moderno  
  
    self.ax.spines['top'].set_visible(False)  
  
    self.ax.spines['right'].set_visible(False)  
  
    self.ax.spines['left'].set_color(self.theme['accent'])  
  
    self.ax.spines['bottom'].set_color(self.theme['accent'])  
  
    self.ax.spines['left'].set_linewidth(2)  
  
    self.ax.spines['bottom'].set_linewidth(2)
```

```

# Grid moderno

self.ax.grid(True, linestyle='--', linewidth=0.7, color=self.theme['plot_grid'],
alpha=0.8)

self.ax.axhline(0, color=self.theme['accent'], linewidth=2, alpha=0.9)

self.ax.axvline(0, color=self.theme['accent'], linewidth=2, alpha=0.9)

self.ax.set_axisbelow(True)

def _apply_3d_style(self):

    """Estilo moderno para 3D"""

    self.fig.patch.set_facecolor(self.theme['plot_bg'])

    self.ax.set_facecolor(self.theme['plot_bg'])

    self.ax.xaxis.label.set_color(self.theme['text_primary'])

    self.ax.yaxis.label.set_color(self.theme['text_primary'])

    self.ax.zaxis.label.set_color(self.theme['text_primary'])

    self.ax.tick_params(colors=self.theme['text_primary'], which='both', labelsize=10)

    self.ax.title.set_color(self.theme['accent'])

    self.ax.title.set_fontweight('bold')

    self.ax.title.set_fontsize(14)

    self.ax.view_init(elev=20, azim=45)

def set_animation_speed(self, speed):

    self.animation_speed = speed

```

```
def toggle_animation(self):  
    self.animation_enabled = not self.animation_enabled  
    return self.animation_enabled  
  
  
def toggle_velocity(self):  
    self.show_velocity = not self.show_velocity  
    return self.show_velocity  
  
  
def clear(self):  
    """Limpieza completa que incluye el eje secundario"""  
    if self.anim:  
        try:  
            self.anim.event_source.stop()  
        except AttributeError:  
            pass  
        self.anim = None  
  
  
    # Limpiar eje secundario si existe  
    if self.ax2 is not None:  
        try:  
            self.ax2.remove()  
        except:  
            self.ax2 = None
```

```
except:
```

```
    pass
```

```
for artist in self.annotations:
```

```
    try:
```

```
        artist.remove()
```

```
    except (ValueError, AttributeError):
```

```
        pass
```

```
    self.annotations.clear()
```

```
for band in self.bands:
```

```
    try:
```

```
        band.remove()
```

```
    except (ValueError, AttributeError):
```

```
        pass
```

```
    self.bands.clear()
```

```
self.ax.cla()
```

```
if self.current_plot_type == "3d":
```

```
    self._apply_3d_style()
```

```
else:
```

```
    self._apply_modern_style()
```

```
self.current_plot_type = None
```

```
self.area_info = {}

self.canvas.draw_idle()

def _shade_area_simple(self, func_str, a, b, color='red'):
    """Sombrea el área bajo la curva de forma simple en un color."""
    expr = safe_sympify(func_str)
    f = sp.lambdify(X, expr, 'numpy')

    xs = np.linspace(float(a), float(b), 500)
    with np.errstate(all='ignore'):
        ys = f(xs)
        ys = np.nan_to_num(ys, nan=0.0)

    # Calcular el área neta (integral definida)
    area_neta = sci_integrate.trapz(ys, xs)

    # Sombrear el área entre la función y el eje x
    self.ax.fill_between(xs, 0, ys, alpha=0.5, color=color, label='Área bajo la curva')

    return area_neta
```

```

def plot_function(self, func_str, xlim=(-5,5), color_index=0, animate=True,
show_area=True, area_limits=None):

    self.current_plot_type = "function"

    self.fig.clear()

    self.ax = self.fig.add_subplot(111)

    expr = safe_sympify(func_str)

    f = sp.lambdify(X, expr, 'numpy')

    xs = np.linspace(float(xlim[0]), float(xlim[1]), 1000) # Reduced for performance

    with np.errstate(divide='ignore', invalid='ignore'):

        ys = f(xs)

        ys = np.array(ys, dtype=float)

        mask_finite = np.isfinite(ys)

    if not np.any(mask_finite):

        raise ValueError("La función no produce valores finitos en el intervalo
especificado.")

    color = AppConfig.GRADIENT_COLORS[color_index %
len(AppConfig.GRADIENT_COLORS)]

```

```

# Gráfica principal con estilo moderno

line, = self.ax.plot(xs[mask_finite], ys[mask_finite],
                      linewidth=3, color=color, alpha=0.95, solid_capstyle='round',
                      label=f'f(x) = {sp.simplify(expr)}', zorder=5)

self.ax.set_xlim(xlim)

# Ajuste automático de límites Y

valid_ys = ys[mask_finite]

if valid_ys.size > 0:
    q1, q3 = np.percentile(valid_ys, [25, 75])
    iqr = q3 - q1
    lower_bound = q1 - 2 * iqr
    upper_bound = q3 + 2 * iqr

    reasonable_ys = valid_ys[(valid_ys >= lower_bound) & (valid_ys <=
upper_bound)]

if reasonable_ys.size > 0:
    y_margin = (np.max(reasonable_ys) - np.min(reasonable_ys)) * 0.15
    y_min = np.min(reasonable_ys) - y_margin
    y_max = np.max(reasonable_ys) + y_margin
    self.ax.set_ylim(y_min, y_max)

```

```
# Área bajo la curva simple (MODIFICACIÓN PRINCIPAL)

if show_area and area_limits and area_limits[0] is not None and area_limits[1] is not
```

None:

try:

```
a, b = area_limits

if xlim[0] <= a <= xlim[1] and xlim[0] <= b <= xlim[1]:

    area_neta = self._shade_area_simple(func_str, a, b, color='red')

    self.area_info = {'area_neta': area_neta}
```

# Mostrar información numérica en la gráfica

```
self.ax.text(0.02, 0.98, f'Área neta: {area_neta:.6f}',
```

transform=self.ax.transAxes,

```
fontsize=12, verticalalignment='top',
```

```
bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
```

except Exception as e:

```
print(f'Error en área simple: {e}')
```

# Leyenda moderna

if func\_strip():

```
self.ax.legend(facecolor=self.theme['card_bg'], edgecolor=self.theme['border'],
labelcolor=self.theme['text_primary'], fontsize='small',
```

```

loc='upper right', framealpha=0.95, shadow=True)

self._apply_modern_style()

self.canvas.draw_idle()

def plot_3d_function(self, func_str, xlims=(-5,5), ylims=(-5,5)):

    self.current_plot_type = "3d"

    self.fig.clear()

    self.ax = self.fig.add_subplot(111, projection='3d')

try:

    expr = safe_sympify(func_str)

    f = sp.lambdify((X, Y), expr, 'numpy')

nx = 30 # Reduced for performance

ny = 30 # Reduced for performance

xx = np.linspace(xlims[0], xlims[1], nx)

yy = np.linspace(ylims[0], ylims[1], ny)

XX, YY = np.meshgrid(xx, yy)

with np.errstate(divide='ignore', invalid='ignore'):

    ZZ = f(XX, YY)

```

```

ZZ = np.nan_to_num(ZZ, nan=0.0)

surf = self.ax.plot_surface(XX, YY, ZZ, cmap='viridis', alpha=0.8)
self.ax.set_xlabel('X')
self.ax.set_ylabel('Y')
self.ax.set_zlabel('Z = f(x,y)')
self.ax.set_title(f'z = {sp.simplify(expr)}')

self._apply_3d_style()
self.canvas.draw_idle()

except Exception as e:
    messagebox.showerror("Error", f"No se pudo generar la gráfica 3D:\n{str(e)}")

def mark_points(self, points, color='#f59e0b', animate=True):
    # Eliminar animaciones - siempre marcar instantáneamente
    for p in points:
        self._mark_point_instant(p, color)

    self.canvas.draw_idle()

def _mark_point_instant(self, p, color):
    x, y, label = p.get('x'), p.get('y'), p.get('tipo', '')
    if x is None or y is None:

```

```

return

# Punto con efecto moderno

dot = self.ax.scatter([x], [y], color=color, s=120, zorder=10,
                      edgecolors='white', linewidths=2.5, alpha=0.95,
                      marker='o')

# Texto con estilo moderno

ann_text = f" ⚡ {label}\n({x:.4f}, {y:.4f})"

if 'curvatura' in p:
    ann_text += f"\n📈 Curvatura: {p['curvatura']:.4f}"

ann = self.ax.annotate(ann_text, (x, y),
                      textcoords="offset points",
                      xytext=(0, 25),
                      ha='center', va='bottom',
                      fontsize=9,
                      color='white',
                      weight='bold',
                      bbox=dict(boxstyle='round', pad=0.4,
                                fc=self.theme['card_bg'],
                                ec=color,
                                alpha=0.95,

```

```

        linewidth=2))

self.annotations.extend([dot, ann])

def plot_physics_motion(self, physics_data, problem_type):
    """Grafica movimientos fisicos con estilo moderno"""

    self.current_plot_type = "physics"

    self.clear()

    if problem_type == 'caida_libre':
        self._plot_caida_libre_moderna(physics_data)

    elif problem_type == 'movimiento_rectilineo':
        self._plot_movimiento_rectilineo_moderno(physics_data)

    # Título moderno

    title_map = {
        'caida_libre': '🚀 Caída Libre - Análisis Completo',
        'movimiento_rectilineo': '📈 Movimiento Rectilíneo'
    }

    self.ax.set_title(title_map.get(problem_type, '📊 Física'),
                      fontsize=16, fontweight='bold', pad=20, color=self.theme['accent'])

    self.canvas.draw_idle()

```

```
def _plot_caida_libre_moderna(self, data):
    """Grafica moderna de caída libre"""

    tiempo = data.get('tiempo_sin_resistencia', data.get('tiempo', 0))

    altura = data.get('altura', 0)

    velocidad_inicial = data.get('velocidad_inicial', 0)

    t_max = tiempo * 1.5

    t = np.linspace(0, t_max, 100) # Reduced for performance

    # Ecuaciones de movimiento

    h_t = altura + velocidad_inicial * t - 0.5 * GRAVEDAD_TIERRA * t**2

    h_t = np.maximum(h_t, 0)

    v_t = velocidad_inicial - GRAVEDAD_TIERRA * t

    self.ax.clear()

    self._apply_modern_style()

    # Gráfica de altura con estilo moderno

    color_altura = self.theme['accent']

    self.ax.plot(t, h_t, linewidth=3.5, color=color_altura,
                 label=f'Altura (m)', alpha=0.9, zorder=5)
```

```

# Línea del suelo

self.ax.axhline(y=0, color=self.theme['error'], linestyle='--',
                  alpha=0.8, linewidth=2, label='🔴 Suelo', zorder=4)

# Punto de impacto

if tiempo <= t_max:

    impact_point = self.ax.plot([tiempo], [0], 'o', markersize=10,
                                 color=self.theme['error'],
                                 label=f'💥 Impacto: {tiempo:.4f} s', zorder=6)

self.ax.set_xlabel('⌚ Tiempo (s)', fontsize=12, fontweight='bold')

self.ax.set_ylabel('📏 Altura (m)', fontsize=12, fontweight='bold',
                   color=color_altura)

self.ax.tick_params(axis='y', labelcolor=color_altura)

# Gráfica de velocidad en eje secundario

if self.show_velocity:

    if self.ax2 is not None:

        self.ax2.remove()

    self.ax2 = self.ax.twinx()

    color_velocidad = self.theme['accent_secondary']

    self.ax2.plot(t, v_t, linewidth=2.5, color=color_velocidad,

```

```

        linestyle='--', alpha=0.8, label='🚀 Velocidad (m/s)', zorder=3)

self.ax2.set_ylabel('🚀 Velocidad (m/s)', fontsize=12,
                     fontweight='bold', color=color_velocidad)

self.ax2.tick_params(axis='y', labelcolor=color_velocidad)

# Combinar leyendas

lines1, labels1 = self.ax.get_legend_handles_labels()
lines2, labels2 = self.ax2.get_legend_handles_labels()

self.ax.legend(lines1 + lines2, labels1 + labels2,
               loc='upper right', framealpha=0.95)

else:

    self.ax.legend(loc='upper right')

self.ax.set_ylim(bottom=0)
self.ax.set_xlim(left=0)
self.ax.grid(True, alpha=0.3)

def _plot_movimiento_rectilineo_moderno(self, data):

    """Grafica moderna de movimiento rectilíneo"""

    tiempo = data.get('tiempo', 0)
    desplazamiento = data.get('desplazamiento', 0)
    velocidad_inicial = data.get('velocidad_inicial', 0)
    aceleracion = data.get('aceleracion', 0)

```

```

t = np.linspace(0, tiempo, 100) # Reduced for performance

d_t = velocidad_inicial * t + 0.5 * aceleracion * t**2

v_t = velocidad_inicial + aceleracion * t

self.ax.clear()

self._apply_modern_style()

# Gráfica de desplazamiento

color_desplazamiento = self.theme['accent']

self.ax.plot(t, d_t, linewidth=3.5, color=color_desplazamiento,
             label=f'⭐ Desplazamiento (m)', alpha=0.9, zorder=5)

# Punto final

self.ax.plot([tiempo], [desplazamiento], 'o', markersize=10,
             color=self.theme['success'],
             label=f'🎯 Final: {desplazamiento:.4f}m', zorder=6)

self.ax.set_xlabel('⌚ Tiempo (s)', fontsize=12, fontweight='bold')

self.ax.set_ylabel('⭐ Desplazamiento (m)', fontsize=12,
                  fontweight='bold', color=color_desplazamiento)

self.ax.tick_params(axis='y', labelcolor=color_desplazamiento)

```

```

# Gráfica de velocidad

if self.show_velocity:

    if self.ax2 is not None:

        self.ax2.remove()

        self.ax2 = self.ax.twinx()

        color_velocidad = self.theme['accent_secondary']

        self.ax2.plot(t, v_t, linewidth=2.5, color=color_velocidad,
                      linestyle='--', alpha=0.8, label='🚀 Velocidad (m/s)', zorder=4)

        self.ax2.set_ylabel('🚀 Velocidad (m/s)', fontsize=12,
                           fontweight='bold', color=color_velocidad)

        self.ax2.tick_params(axis='y', labelcolor=color_velocidad)

# Leyenda combinada

lines1, labels1 = self.ax.get_legend_handles_labels()

lines2, labels2 = self.ax2.get_legend_handles_labels()

self.ax.legend(lines1 + lines2, labels1 + labels2,
               loc='upper left', framealpha=0.95)

else:

    self.ax.legend(loc='upper left')

self.ax.set_xlim(left=0)

if desplazamiento >= 0:

```

```
    self.ax.set_ylim(bottom=0)

else:

    self.ax.set_ylim(top=0)

self.ax.grid(True, alpha=0.3)

def get_area_info(self):
    """Retorna información detallada del área"""
    return self.area_info

# ----- INTERFAZ PRINCIPAL MEJORADA -----

class UltraCalc2DModernApp:

    def __init__(self, root):
        self.root = root

        self.root.title("Ultra Calc 2D - Matemáticas Visuales Modernas")
        self.root.geometry("1400x900")
        self.root.minsize(1200, 800)

    # Configuración inicial

    self.theme_mode = "dark"

    self.theme = AppConfig.DARK_THEME

    self.animation_speed = "normal"
```

```
# Centrar ventana
self._center_window()

# Configurar estilos
self.setup_styles()

# Mostrar pantalla de inicio
self._create_loading_screen()

# Inicializar motores después de la pantalla de carga
self.root.after(1500, self._initialize_components)

def _center_window(self):
    self.root.update_idletasks()
    width = 1400
    height = 900
    x = (self.root.winfo_screenwidth() // 2) - (width // 2)
    y = (self.root.winfo_screenheight() // 2) - (height // 2)
    self.root.geometry(f'{width}x{height}+{x}+{y}')

def _create_loading_screen(self):
    self.loading_frame = tk.Frame(self.root, bg=self.theme['bg'])
    self.loading_frame.pack(fill=tk.BOTH, expand=True)
```

```
logo_label = tk.Label(  
    self.loading_frame,  
    text="█",  
    font=("Arial", 64),  
    bg=self.theme['bg'],  
    fg=self.theme['accent'])  
  
logo_label.pack(pady=50)  
  
title_label = tk.Label(  
    self.loading_frame,  
    text="ULTRA CALC 2D",  
    font=("Arial", 32, "bold"),  
    bg=self.theme['bg'],  
    fg=self.theme['text_primary'])  
  
title_label.pack(pady=10)  
  
subtitle_label = tk.Label(  
    self.loading_frame,  
    text="Cargando experiencia matemática moderna...",  
    font=("Arial", 14),
```

```
        bg=self.theme['bg'],
        fg=self.theme['text_secondary']
    )
    subtitle_label.pack(pady=20)

self.progress = ttk.Progressbar(
    self.loading_frame,
    mode='indeterminate',
    length=400
)
self.progress.pack(pady=30)
self.progress.start(15)

def _initialize_components(self):
    self.loading_frame.destroy()

# Inicializar motor matemático mejorado
self.engine = MathEngine()

# Crear interfaz principal
self.create_modern_interface()

def setup_styles(self):
```

```
style = ttk.Style()

style.theme_use('clam')

style.configure(".",

    background=self.theme['bg'],

    foreground=self.theme['text_primary'],

    font=("Segoe UI", 10))

style.configure("Card.TFrame",

    background=self.theme['card_bg'],

    relief="flat",

    borderwidth=1)

style.configure("Modern.TButton",

    background=self.theme['button_bg'],

    foreground=self.theme['text_primary'],

    borderwidth=0,

    focuscolor="none",

    padding=(12, 8),

    font=("Segoe UI", 10, "bold"))

style.map("Modern.TButton",

    background=[('active', self.theme['button_hover'])],
```

```
('pressed', self.theme['accent'])],  
foreground=[('active', self.theme['text_primary']),  
           ('pressed', self.theme['card_bg'])])  
  
  
style.configure("Accent.TButton",  
               background=self.theme['accent'],  
               foreground=self.theme['card_bg'],  
               borderwidth=0,  
               padding=(12, 8),  
               font=("Segoe UI", 10, "bold"))  
  
  
style.map("Accent.TButton",  
         background=[('active', self.theme['accent_secondary']),  
                    ('pressed', self.theme['accent_tertiary'])])  
  
  
for btn_type, bg_color in [("Math.TButton", "#2a3b5c"),  
                           ("Num.TButton", self.theme['button_bg']),  
                           ("Control.TButton", "#4a5b7c")]:  
  
    style.configure(btn_type,  
                   background=bg_color,  
                   foreground=self.theme['text_primary'],  
                   borderwidth=0,  
                   padding=(8, 6),
```

```
font=("Segoe UI", 9, "bold"))

style.map(btn_type,
    background=[('active', self.theme['button_hover']),
    ('pressed', self.theme['accent'])])

style.configure("Header.TLabel",
    font=("Segoe UI", 16, "bold"),
    foreground=self.theme['accent'],
    background=self.theme['card_bg'])

style.configure("CardHeader.TLabel",
    font=("Segoe UI", 12, "bold"),
    foreground=self.theme['accent_secondary'],
    background=self.theme['card_bg'])

style.configure("Small.TLabel",
    font=("Segoe UI", 9),
    foreground=self.theme['text_secondary'],
    background=self.theme['card_bg'])

style.configure("TEntry",
    fieldbackground=self.theme['entry_bg'],
```

```
foreground=self.theme['text_primary'],
bordercolor=self.theme['border'],
insertcolor=self.theme['text_primary'],
padding=(8, 6))

style.map("TEntry",
    bordercolor=[('focus', self.theme['accent'])])

style.configure("TCombobox",
    fieldbackground=self.theme['entry_bg'],
    background=self.theme['button_bg'],
    foreground=self.theme['text_primary'],
    arrowcolor=self.theme['accent'])

style.configure("TNotebook",
    background=self.theme['bg'],
    borderwidth=0)

style.configure("TNotebook.Tab",
    background=self.theme['bg'],
    foreground=self.theme['text_muted'],
    padding=(15, 8),
    borderwidth=0,
```

```
font=("Segoe UI", 9, "bold"))

style.map("TNotebook.Tab",
    background=[("selected", self.theme['card_bg'])],
    foreground=[("selected", self.theme['accent'])])

style.configure("Modern.Horizontal.TProgressbar",
    background=self.theme['accent'],
    troughcolor=self.theme['card_bg'],
    borderwidth=0)

def create_modern_interface(self):
    self.create_header()

    # Crear paned window principal con barra de desplazamiento
    main_paned = ttk.PanedWindow(self.root, orient=tk.HORIZONTAL)
    main_paned.pack(fill=tk.BOTH, expand=True, padx=15, pady=10)

    # Left panel con scrollbar
    left_frame = ttk.Frame(main_paned)
    left_canvas = tk.Canvas(left_frame, bg=self.theme['bg'], highlightthickness=0)
    left_scrollbar = ttk.Scrollbar(left_frame, orient="vertical",
        command=left_canvas.yview)
```

```
left_scrollable_frame = ttk.Frame(left_canvas, style="Card.TFrame")

left_scrollable_frame.bind(
    "<Configure>",
    lambda e: left_canvas.configure(scrollregion=left_canvas.bbox("all"))

)

left_canvas.create_window((0, 0), window=left_scrollable_frame, anchor="nw")
left_canvas.configure(yscrollcommand=left_scrollbar.set)

left_canvas.pack(side="left", fill="both", expand=True)
left_scrollbar.pack(side="right", fill="y")

main_paned.add(left_frame, weight=1)

# Center panel
center_panel = ttk.Frame(main_paned, style="Card.TFrame")
main_paned.add(center_panel, weight=3)

# Right panel con scrollbar
right_frame = ttk.Frame(main_paned)
right_canvas = tk.Canvas(right_frame, bg=self.theme['bg'], highlightthickness=0)
```

```
right_scrollbar = ttk.Scrollbar(right_frame, orient="vertical",
                                command=right_canvas.yview)

right_scrollable_frame = ttk.Frame(right_canvas, style="Card.TFrame")

right_scrollable_frame.bind(
    "<Configure>",
    lambda e: right_canvas.configure(scrollregion=right_canvas.bbox("all"))

)

right_canvas.create_window((0, 0), window=right_scrollable_frame, anchor="nw")
right_canvas.configure(yscrollcommand=right_scrollbar.set)

right_canvas.pack(side="left", fill="both", expand=True)
right_scrollbar.pack(side="right", fill="y")

main_paned.add(right_frame, weight=1)

# Configurar los paneles dentro de los frames desplazables

self.create_left_panel(left_scrollable_frame)
self.create_center_panel(center_panel)
self.create_right_panel(right_scrollable_frame)

self.create_status_bar()
```

```
# Configurar el redimensionamiento del canvas

def configure_left_canvas(event):

    left_canvas.itemconfig(left_canvas_window, width=event.width)

    left_canvas_window = left_canvas.create_window((0, 0),
window=left_scrollable_frame, anchor="nw")

    left_canvas.bind('<Configure>', configure_left_canvas)

def configure_right_canvas(event):

    right_canvas.itemconfig(right_canvas_window, width=event.width)

    right_canvas_window = right_canvas.create_window((0, 0),
window=right_scrollable_frame, anchor="nw")

    right_canvas.bind('<Configure>', configure_right_canvas)

def create_header(self):

    header_frame = tk.Frame(self.root, bg=self.theme['card_bg'], height=80)

    header_frame.pack(fill=tk.X, padx=0, pady=0)

    header_frame.pack_propagate(False)

    logo_frame = tk.Frame(header_frame, bg=self.theme['card_bg'])

    logo_frame.pack(side=tk.LEFT, padx=20, pady=20)

    logo_label = tk.Label(logo_frame, text=" ", font=("Arial", 24),
```

```
        bg=self.theme['card_bg'], fg=self.theme['accent'])  
  
    logo_label.pack(side=tk.LEFT)  
  
  
    title_frame = tk.Frame(logo_frame, bg=self.theme['card_bg'])  
    title_frame.pack(side=tk.LEFT, padx=(10, 0))  
  
  
    title_label = tk.Label(title_frame, text="ULTRA CALC 2D",  
                           font=("Segoe UI", 20, "bold"),  
                           bg=self.theme['card_bg'], fg=self.theme['text_primary'])  
    title_label.pack(anchor='w')  
  
  
    subtitle_label = tk.Label(title_frame, text="Matemáticas Visuales Modernas",  
                             font=("Segoe UI", 11),  
                             bg=self.theme['card_bg'], fg=self.theme['text_secondary'])  
    subtitle_label.pack(anchor='w')  
  
  
    controls_frame = tk.Frame(header_frame, bg=self.theme['card_bg'])  
    controls_frame.pack(side=tk.RIGHT, padx=20, pady=20)  
  
  
    self.theme_btn = ModernButton(controls_frame, text="🌙 Modo Noche",  
                                command=self.toggle_theme, style="Modern.TButton")  
    self.theme_btn.pack(side=tk.RIGHT, padx=(10, 0))
```

```

def create_left_panel(self, parent):

    notebook = ttk.Notebook(parent)

    notebook.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    tabs = {

        "🎯 Función": self.create_function_tab,
        "🌐 3D Interactivas": self.create_3d_tab,
        "📊 Derivadas": self.create_derivative_tab,
        "📐 Integrales": self.create_integral_tab,
        "🌍 Física": self.create_physics_tab,
        "⚙️ Opciones": self.create_settings_tab
    }

    for tab_name, tab_creator in tabs.items():

        tab_frame = ttk.Frame(notebook)

        notebook.add(tab_frame, text=tab_name)

        tab_creator(tab_frame)

def create_function_tab(self, parent):

    func_card = self.create_modern_card(parent, "📝 Función Principal")

    ttk.Label(func_card, text="Ingresa tu función f(x):",
             style="Small.TLabel").pack(anchor='w', pady=(0, 5))

```

```

self.func_entry = ModernEntry(func_card, font=("Consolas", 12))

self.func_entry.pack(fill=tk.X, pady=(0, 10))

self.func_entry.insert(0, "7*x - 8.5*x**2 + 3*x**3")

# Calculadora compacta

calc_card = self.create_modern_card(parent, "-Calculadora Compacta")

self.calc_pad = CompactCalculatorPad(calc_card, self.func_entry, self.theme)

self.calc_pad.pack(fill=tk.BOTH, expand=True, pady=5)

range_card = self.create_modern_card(parent, "Rango de Visualización")

range_frame = ttk.Frame(range_card, style="Card.TFrame")

range_frame.pack(fill=tk.X, pady=5)

ttk.Label(range_frame, text="X min", style="Small TLabel").grid(row=0,
column=0, sticky='w')

self.xmin_entry = ModernEntry(range_frame, width=10)

self.xmin_entry.grid(row=1, column=0, padx=(0, 5), sticky='ew')

ttk.Label(range_frame, text="X max", style="Small TLabel").grid(row=0,
column=1, sticky='w')

self xmax_entry = ModernEntry(range_frame, width=10)

self xmax_entry.grid(row=1, column=1, padx=(5, 0), sticky='ew')

```

```
range_frame.columnconfigure((0, 1), weight=1)

self.xmin_entry.insert(0, "0")

self xmax_entry.insert(0, "2")

area_card = self.create_modern_card(parent, "[Integrales Definidas]")

area_frame = ttk.Frame(area_card, style="Card.TFrame")

area_frame.pack(fill=tk.X, pady=5)

ttk.Label(area_frame, text="Límite inferior (a)", style="Small TLabel").grid(row=0,
column=0, sticky='w')

self.lower_entry = ModernEntry(area_frame, width=10)

self.lower_entry.grid(row=1, column=0, padx=(0, 5), sticky='ew')

ttk.Label(area_frame, text="Límite superior (b)",
style="Small TLabel").grid(row=0, column=1, sticky='w')

self.upper_entry = ModernEntry(area_frame, width=10)

self.upper_entry.grid(row=1, column=1, padx=(5, 0), sticky='ew')

area_frame.columnconfigure((0, 1), weight=1)

self.lower_entry.insert(0, "0")

self.upper_entry.insert(0, "1.5")
```

```

action_card = self.create_modern_card(parent, "🚀 Acciones Rápidas")

ModernButton(action_card, text="🎨 Graficar con Área",
             command=self.on_plot, style="Accent.TButton").pack(fill=tk.X, pady=5)

ModernButton(action_card, text="🔍 Analizar Puntos",
             command=self.on_analyze_points, style="Modern.TButton").pack(fill=tk.X,
             pady=5)

ModernButton(action_card, text="🧹 Limpiar Todo",
             command=self.on_clear_all, style="Modern.TButton").pack(fill=tk.X,
             pady=5)

def create_3d_tab(self, parent):
    func3d_card = self.create_modern_card(parent, "🌐 Función 3D z = f(x,y)")

    ttk.Label(func3d_card, text="Ingresa tu función f(x,y):",
              style="Small TLabel").pack(anchor='w', pady=(0, 5))

    self.func3d_entry = ModernEntry(func3d_card, font=("Consolas", 12))
    self.func3d_entry.pack(fill=tk.X, pady=(0, 10))
    self.func3d_entry.insert(0, "sin(x**2 + y**2)")

range3d_card = self.create_modern_card(parent, "📊 Rango de Visualización 3D")

```

```
range3d_frame = ttk.Frame(range3d_card, style="Card.TFrame")
range3d_frame.pack(fill=tk.X, pady=5)

# X range
ttk.Label(range3d_frame, text="X min", style="Small TLabel").grid(row=0,
column=0, sticky='w')
self.x3d_min_entry = ModernEntry(range3d_frame, width=10)
self.x3d_min_entry.grid(row=1, column=0, padx=(0, 5), sticky='ew')

ttk.Label(range3d_frame, text="X max", style="Small TLabel").grid(row=0,
column=1, sticky='w')
self.x3d_max_entry = ModernEntry(range3d_frame, width=10)
self.x3d_max_entry.grid(row=1, column=1, padx=(5, 0), sticky='ew')

# Y range
ttk.Label(range3d_frame, text="Y min", style="Small TLabel").grid(row=2,
column=0, sticky='w', pady=(10,0))
self.y3d_min_entry = ModernEntry(range3d_frame, width=10)
self.y3d_min_entry.grid(row=3, column=0, padx=(0, 5), sticky='ew', pady=(0,10))

ttk.Label(range3d_frame, text="Y max", style="Small TLabel").grid(row=2,
column=1, sticky='w', pady=(10,0))
self.y3d_max_entry = ModernEntry(range3d_frame, width=10)
```

```
self.y3d_max_entry.grid(row=3, column=1, padx=(5, 0), sticky='ew', pady=(0,10))
```

```
range3d_frame.columnconfigure((0, 1), weight=1)
```

```
self.x3d_min_entry.insert(0, "-5")
```

```
self.x3d_max_entry.insert(0, "5")
```

```
self.y3d_min_entry.insert(0, "-5")
```

```
self.y3d_max_entry.insert(0, "5")
```

```
action3d_card = self.create_modern_card(parent, "🚀 Acciones 3D")
```

```
ModernButton(action3d_card, text="🌐 Graficar 3D",
```

```
command=self.on_plot_3d, style="Accent.TButton").pack(fill=tk.X, pady=5)
```

```
ModernButton(action3d_card, text="🧹 Limpiar Gráfica",
```

```
command=self.on_clear_all, style="Modern.TButton").pack(fill=tk.X,
```

```
pady=5)
```

```
def create_derivative_tab(self, parent):
```

```
deriv_card = self.create_modern_card(parent, "📈 Cálculo de Derivadas")
```

```
ttk.Label(deriv_card, text="Método de derivación:",
```

```
style="Small TLabel").pack(anchor='w', pady=(5, 0))
```

```
self.deriv_method = ModernCombobox(deriv_card, values=[
```

```

    "Auto (completo)",
    "Paso a paso completo",
    "Solo regla de la potencia",
    "Solo regla de la cadena",
    "Directo (SymPy)"

], state='readonly')

self.deriv_method.pack(fill=tk.X, pady=5)

self.deriv_method.set("Auto (completo)")


ttk.Label(deriv_card, text="Orden:", style="Small.TLabel").pack(anchor='w',
pady=(10, 0))

self.deriv_order = ModernCombobox(deriv_card, values=["1","2","3","4"],
state='readonly', width=8)

self.deriv_order.pack(fill=tk.X, pady=5)

self.deriv_order.set("1")


ttk.Label(deriv_card, text="Evaluar en x =", style="Small.TLabel").pack(anchor='w', pady=(10, 0))

self.eval_point = ModernEntry(deriv_card)

self.eval_point.pack(fill=tk.X, pady=5)

self.eval_point.insert(0, "1")



ModernButton(deriv_card, text="  Calcular Derivada",

```

```

        command=lambda: self.on_calculate("derivada"),
style="Accent.TButton").pack(fill=tk.X, pady=15)

def create_integral_tab(self, parent):
    integral_card = self.create_modern_card(parent, "CALCULO Cálculo de Integrales")

    ttk.Label(integral_card, text="Método de cálculo:",
style="Small.TLabel").pack(anchor='w', pady=(5, 0))

    self.calc_type = ModernCombobox(integral_card, values=[
        "Auto (mejor intento)", "Integral indefinida", "Integral definida",
        "Integral impropia", "Sustitución", "Por partes", "Fracciones parciales",
        "Sust. trigonométrica", "Numérico (Simpson)"
    ], state='readonly')

    self.calc_type.pack(fill=tk.X, pady=5)
    self.calc_type.set("Auto (mejor intento)")

    ModernButton(integral_card, text="CALCULAR Calcular Integral",
command=lambda: self.on_calculate("integral"),
style="Accent.TButton").pack(fill=tk.X, pady=15)

def create_physics_tab(self, parent):
    physics_card = self.create_modern_card(parent, "PROBLEMAS Problemas de Física")

```

```

ttk.Label(physics_card, text="Tipo de problema:",
style="Small.TLabel").pack(anchor='w', pady=(5, 0))

self.physics_problem = ModernCombobox(physics_card, values=[

    "Caída libre avanzada",
    "Movimiento de proyectil",
    "Movimiento rectilíneo",
    "Velocidad promedio",
    "Resistencia del aire",
    "Análisis de energía"

], state='readonly')

self.physics_problem.pack(fill=tk.X, pady=5)

self.physics_problem.set("Caída libre avanzada")

self.physics_params_frame = ttk.Frame(physics_card, style="Card.TFrame")
self.physics_params_frame.pack(fill=tk.X, pady=10)

self.physics_problem.bind('<<ComboboxSelected>>', self._update_physics_params)

physics_action_frame = ttk.Frame(physics_card, style="Card.TFrame")
physics_action_frame.pack(fill=tk.X, pady=10)

ModernButton(physics_action_frame, text="⚙ Resolver Problema Físico",

```

```

        command=self.on_physics_calculate,
style="Accent.TButton").pack(fill=tk.X, pady=5)

self._update_physics_params()

def create_settings_tab(self, parent):

    speed_card = self.create_modern_card(parent, "🎬 Velocidad de Animación")

    self.speed_var = tk.StringVar(value="normal")

    speeds = [("🐢 Lenta", "slow"), ("🚶 Normal", "normal"),
              ("🏎️ Rápida", "fast"), ("⚡ Instantánea", "instant")]

    for text, value in speeds:
        ttk.Radiobutton(speed_card, text=text, variable=self.speed_var,
                        value=value, command=self.on_speed_change,
style="TRadiobutton").pack(anchor='w', pady=2)

    toggle_card = self.create_modern_card(parent, "🎛️ Configuración Visual")

    self.animation_var = tk.BooleanVar(value=False) # Desactivado por defecto

    ttk.Checkbutton(toggle_card, text="🎬 Activar animaciones",
variable=self.animation_var,

```

```

        command=self.on_animation_toggle,
style="Small.TLabel").pack(anchor='w', pady=5)

self.velocity_var = tk.BooleanVar(value=True)

ttk.Checkbutton(toggle_card, text="  Mostrar línea de velocidad",
variable=self.velocity_var,
        command=self.on_velocity_toggle,
style="Small.TLabel").pack(anchor='w', pady=5)

examples_card = self.create_modern_card(parent, "  Ejemplos Rápidos")
examples = ["x**3 - 3*x + 2", "sin(x)*cos(x)", "exp(-x**2)", "1/(1+x**2)"]

for ex in examples:
    btn = ModernButton(examples_card, text=ex,
                       command=lambda e=ex: self._set_example(e),
                       style="Modern.TButton")
    btn.pack(fill=tk.X, pady=2)

def create_center_panel(self, parent):
    graph_frame = ttk.Frame(parent, style="Card.TFrame", padding=10)
    graph_frame.pack(fill=tk.BOTH, expand=True)

```

```
    self.graph_title = ttk.Label(graph_frame, text="⚙️ Gráfica Interactiva",
style="CardHeader.TLabel")

    self.graph_title.pack(anchor='w', pady=(0, 10))

self.fig, self.ax = plt.subplots(figsize=(8, 6), dpi=100)

self.canvas = FigureCanvasTkAgg(self.fig, master=graph_frame)
canvas_widget = self.canvas.get_tk_widget()
canvas_widget.pack(fill=tk.BOTH, expand=True)

toolbar = NavigationToolbar2Tk(self.canvas, graph_frame)
toolbar.update()

self.plot_manager = ModernPlotManager(self.fig, self.ax, self.canvas, self.theme)

def create_right_panel(self, parent):
    results_frame = ttk.Frame(parent, style="Card.TFrame", padding=10)
    results_frame.pack(fill=tk.BOTH, expand=True)

    title_frame = ttk.Frame(results_frame, style="Card.TFrame")
    title_frame.pack(fill=tk.X, pady=(0, 10))

    ttk.Label(title_frame, text="📋 Pasos y Resultados",
style="CardHeader.TLabel").pack(side=tk.LEFT)
```

```
self.steps_counter = ttk.Label(title_frame, text="0 pasos", style="Small.TLabel")
self.steps_counter.pack(side=tk.RIGHT)

text_frame = ttk.Frame(results_frame, style="Card.TFrame")
text_frame.pack(fill=tk.BOTH, expand=True)

self.steps_text = scrolledtext.ScrolledText(
    text_frame,
    wrap=tk.WORD,
    font=("Consolas", 10),
    bg=self.theme['card_bg'],
    fg=self.theme['text_primary'],
    insertbackground=self.theme['text_primary'],
    relief='flat',
    borderwidth=0,
    padx=10,
    pady=10,
    height=15 # Altura fija para mejor organización
)
self.steps_text.pack(fill=tk.BOTH, expand=True)

# Configurar tags para estilizado
```

```

    self.steps_text.tag_configure("title", foreground=self.theme['accent'], font=("Segoe
UI", 12, "bold"))

    self.steps_text.tag_configure("step", foreground=self.theme['text_secondary'],
font=("Consolas", 10))

    self.steps_text.tag_configure("result", foreground=self.theme['success'],
font=("Consolas", 10, "bold"))

    self.steps_text.tag_configure("area", foreground=self.theme['accent_tertiary'],
font=("Consolas", 10, "bold"))

export_frame = ttk.Frame(results_frame, style="Card.TFrame")
export_frame.pack(fill=tk.X, pady=(10, 0))

ModernButton(export_frame, text="📄 Exportar Resultados",
             command=self.export_steps, style="Modern.TButton").pack(side=tk.LEFT,
fill=tk.X, expand=True, padx=(0, 5))

ModernButton(export_frame, text="〽️ Exportar Gráfica",
             command=self.export_plot, style="Modern.TButton").pack(side=tk.LEFT,
fill=tk.X, expand=True, padx=(5, 0))

def create_status_bar(self):

    status_frame = tk.Frame(self.root, bg=self.theme['card_bg'], height=30)
    status_frame.pack(fill=tk.X, side=tk.BOTTOM)

```

```

status_frame.pack_propagate(False)

self.status_var = tk.StringVar(value="🚀 Listo para calcular...")

status_label = tk.Label(status_frame, textvariable=self.status_var,
                       bg=self.theme['card_bg'], fg=self.theme['text_secondary'],
                       font=("Segoe UI", 9))

status_label.pack(side=tk.LEFT, padx=15, pady=5)

# Información del área actual

self.area_info_var = tk.StringVar(value="Área: No calculada")

area_label = tk.Label(status_frame, textvariable=self.area_info_var,
                      bg=self.theme['card_bg'], fg=self.theme['accent_tertiary'],
                      font=("Segoe UI", 9))

area_label.pack(side=tk.RIGHT, padx=15, pady=5)

def create_modern_card(self, parent, title):

    card = ttk.Frame(parent, style="Card.TFrame", padding=15)

    card.pack(fill=tk.X, pady=8)

    if title:

        title_label = ttk.Label(card, text=title, style="CardHeader TLabel")

        title_label.pack(anchor='w', pady=(0, 10))

```

```
return card
```

```
def toggle_theme(self):

    if self.theme_mode == "dark":

        self.theme_mode = "light"

        self.theme = AppConfig.LIGHT_THEME

        self.theme_btn.config(text="☀️ Modo Día")

    else:

        self.theme_mode = "dark"

        self.theme = AppConfig.DARK_THEME

        self.theme_btn.config(text="🌙 Modo Noche")

    self.setup_styles()

    self.plot_manager.theme = self.theme

    if self.plot_manager.current_plot_type == "3d":

        self.plot_manager._apply_3d_style()

    else:

        self.plot_manager._apply_modern_style()

    self.plot_manager.canvas.draw_idle()

    self.steps_text.config(bg=self.theme['card_bg'], fg=self.theme['text_primary'])

    self._set_status("Tema cambiado correctamente ✨")
```

```
def on_speed_change(self):
```

```

speed_map = {
    "slow": AnimationSpeed.SLOW,
    "normal": AnimationSpeed.NORMAL,
    "fast": AnimationSpeed.FAST,
    "instant": AnimationSpeed.INSTANT
}

new_speed = speed_map.get(self.speed_var.get(), AnimationSpeed.NORMAL)
self.plot_manager.set_animation_speed(new_speed)

speed_names = {
    "slow": "Lenta 🌱",
    "normal": "Normal 🚶",
    "fast": "Rápida 🐾",
    "instant": "Instantánea ⚡"
}

self._set_status(f"Velocidad: {speed_names[self.speed_var.get()]}",
clear_after_ms=2000)

def on_animation_toggle(self):
    new_state = self.plot_manager.toggle_animation()
    status = "activadas 🎵" if new_state else "desactivadas"

```

```

self._set_status(f"Animaciones {status}", clear_after_ms=2000)

def on_velocity_toggle(self):
    new_state = self.plot_manager.toggle_velocity()
    status = "activada 🎨" if new_state else "desactivada"
    self._set_status(f"Línea de velocidad {status}", clear_after_ms=2000)

def _update_physics_params(self, event=None):
    for widget in self.physics_params_frame.winfo_children():
        widget.destroy()

    problem_type = self.physics_problem.get()

    if problem_type == "Caída libre avanzada":
        ttk.Label(self.physics_params_frame, text="Altura (metros):",
                 style="Small.TLabel").pack(anchor='w')
        self.physics_param1 = ModernEntry(self.physics_params_frame)
        self.physics_param1.pack(fill=tk.X, pady=5)
        self.physics_param1.insert(0, "100")

    ttk.Label(self.physics_params_frame, text="Velocidad inicial (m/s, opcional):",
                 style="Small.TLabel").pack(anchor='w')
    self.physics_param2 = ModernEntry(self.physics_params_frame)

```

```
self.physics_param2.pack(fill=tk.X, pady=5)
self.physics_param2.insert(0, "0")

ttk.Label(self.physics_params_frame, text="Masa (kg, opcional):",
style="Small.TLabel").pack(anchor='w')
self.physics_param3 = ModernEntry(self.physics_params_frame)
self.physics_param3.pack(fill=tk.X, pady=5)
self.physics_param3.insert(0, "1.0")

elif problem_type == "Movimiento de proyectil":
    ttk.Label(self.physics_params_frame, text="Velocidad inicial (m/s):",
style="Small.TLabel").pack(anchor='w')
    self.physics_param1 = ModernEntry(self.physics_params_frame)
    self.physics_param1.pack(fill=tk.X, pady=5)
    self.physics_param1.insert(0, "50")

    ttk.Label(self.physics_params_frame, text="Ángulo (grados):",
style="Small.TLabel").pack(anchor='w')
    self.physics_param2 = ModernEntry(self.physics_params_frame)
    self.physics_param2.pack(fill=tk.X, pady=5)
    self.physics_param2.insert(0, "45")
```

```
        ttk.Label(self.physics_params_frame, text="Altura inicial (m, opcional):",
style="Small.TLabel").pack(anchor='w')

        self.physics_param3 = ModernEntry(self.physics_params_frame)
        self.physics_param3.pack(fill=tk.X, pady=5)
        self.physics_param3.insert(0, "0")

    elif problem_type == "Movimiento rectilíneo":

        ttk.Label(self.physics_params_frame, text="Velocidad inicial (m/s):",
style="Small.TLabel").pack(anchor='w')

        self.physics_param1 = ModernEntry(self.physics_params_frame)
        self.physics_param1.pack(fill=tk.X, pady=5)
        self.physics_param1.insert(0, "10")

        ttk.Label(self.physics_params_frame, text="Aceleración (m/s2):",
style="Small.TLabel").pack(anchor='w')

        self.physics_param2 = ModernEntry(self.physics_params_frame)
        self.physics_param2.pack(fill=tk.X, pady=5)
        self.physics_param2.insert(0, "2")

        ttk.Label(self.physics_params_frame, text="Tiempo (segundos):",
style="Small.TLabel").pack(anchor='w')

        self.physics_param3 = ModernEntry(self.physics_params_frame)
        self.physics_param3.pack(fill=tk.X, pady=5)
```

```
    self.physics_param3.insert(0, "5")

elif problem_type == "Velocidad promedio":
    ttk.Label(self.physics_params_frame, text="Distancia total (metros):",
              style="Small.TLabel").pack(anchor='w')
    self.physics_param1 = ModernEntry(self.physics_params_frame)
    self.physics_param1.pack(fill=tk.X, pady=5)
    self.physics_param1.insert(0, "1000")

    ttk.Label(self.physics_params_frame, text="Tiempo total (segundos):",
              style="Small.TLabel").pack(anchor='w')
    self.physics_param2 = ModernEntry(self.physics_params_frame)
    self.physics_param2.pack(fill=tk.X, pady=5)
    self.physics_param2.insert(0, "60")

elif problem_type == "Resistencia del aire":
    ttk.Label(self.physics_params_frame, text="Masa (kg):",
              style="Small.TLabel").pack(anchor='w')
    self.physics_param1 = ModernEntry(self.physics_params_frame)
    self.physics_param1.pack(fill=tk.X, pady=5)
    self.physics_param1.insert(0, "1")
```

```
        ttk.Label(self.physics_params_frame, text="Área frontal (m2):",
style="Small.TLabel").pack(anchor='w')

        self.physics_param2 = ModernEntry(self.physics_params_frame)
        self.physics_param2.pack(fill=tk.X, pady=5)
        self.physics_param2.insert(0, "0.1")



        ttk.Label(self.physics_params_frame, text="Velocidad (m/s, opcional):",
style="Small.TLabel").pack(anchor='w')

        self.physics_param3 = ModernEntry(self.physics_params_frame)
        self.physics_param3.pack(fill=tk.X, pady=5)
        self.physics_param3.insert(0, "10")



    elif problem_type == "Análisis de energía":

        ttk.Label(self.physics_params_frame, text="Masa (kg):",
style="Small.TLabel").pack(anchor='w')

        self.physics_param1 = ModernEntry(self.physics_params_frame)
        self.physics_param1.pack(fill=tk.X, pady=5)
        self.physics_param1.insert(0, "1.0")



        ttk.Label(self.physics_params_frame, text="Velocidad (m/s):",
style="Small.TLabel").pack(anchor='w')

        self.physics_param2 = ModernEntry(self.physics_params_frame)
        self.physics_param2.pack(fill=tk.X, pady=5)
```

```
self.physics_param2.insert(0, "10")

def _set_status(self, message, clear_after_ms=None):
    self.status_var.set(f'🔴 {message}')
    if clear_after_ms:
        self.root.after(clear_after_ms, lambda: self.status_var.set("🚀 Listo"))

def _set_example(self, example):
    self.func_entry.delete(0, tk.END)
    self.func_entry.insert(0, example)

def on_plot(self):
    try:
        func_str = self.func_entry.get().strip()
        if not func_str:
            raise ValueError("¡Ingresa una función para graficar!")

        xmin = float(self.xmin_entry.get())
        xmax = float(selfxmax_entry.get())

        lower_raw = self.lower_entry.get().strip()
        upper_raw = self.upper_entry.get().strip()
```

```

if lower_raw:
    a = parse_limit_string(lower_raw)

else:
    a = xmin


if upper_raw:
    b = parse_limit_string(upper_raw)

else:
    b = xmax


# Asegurar límites finitos para la gráfica

if not (isinstance(a, (int, float)) and np.isfinite(a)):
    a = xmin

if not (isinstance(b, (int, float)) and np.isfinite(b)):
    b = xmax


self._set_status("🌐 Generando visualización...")

self.plot_manager.clear()

self.graph_title.config(text=f"📊 Gráfica: {func_str}")


self.plot_manager.plot_function(
    func_str,
)

```

```

        xlim=(xmin, xmax),
        color_index=0,
        animate=False, # Desactivado por defecto
        show_area=True,
        area_limits=(a, b)
    )

# Obtener información del área y mostrarla en el panel de resultados
area_info = self.plot_manager.get_area_info()
if area_info:
    area_neta = area_info.get('area_neta', 0)
    self.area_info_var.set(f"Área neta: {area_neta:.6f}")

```

# Mostrar información detallada del área en el panel de resultados

info\_text = f""""

## INFORMACIÓN DEL ÁREA BAJO LA CURVA

Función: {func\_str}

Límites: [{a:.2f}, {b:.2f}]

Área neta: {area\_neta:.8f}

-  El área representa la integral definida de la función en el intervalo especificado.

""""

```

    self.steps_text.delete('1.0', tk.END)
    self._append_steps([info_text])

    self._set_status("✅ Gráfica con área generada exitosamente!")

```

except Exception as e:

```

    messagebox.showerror("Error", f"No se pudo generar la gráfica:\n{str(e)}")
    self._set_status("❌ Error en la gráfica")

```

```
def on_plot_3d(self):
```

```
try:
```

```

    func_str = self.func3d_entry.get().strip()
    if not func_str:

```

```
        raise ValueError("¡Ingresa una función 3D para graficar!")
```

```

    x_min = float(self.x3d_min_entry.get())
    x_max = float(self.x3d_max_entry.get())
    y_min = float(self.y3d_min_entry.get())
    y_max = float(self.y3d_max_entry.get())

```

```
    self._set_status("🌐 Generando visualización 3D...")
```

```
    self.plot_manager.clear()
```

```

self.graph_title.config(text=f'🌐 Gráfica 3D: {func_str}')

self.plot_manager.plot_3d_function(
    func_str,
    xlims=(x_min, x_max),
    ylims=(y_min, y_max)
)

self._append_steps([f'🌐 Gráfica 3D generada para z = {func_str}'], "🌐  
GRÁFICA 3D")

self._set_status("✅ Gráfica 3D generada exitosamente!")

except Exception as e:
    messagebox.showerror("Error", f"No se pudo generar la gráfica 3D:\n{str(e)}")
    self._set_status("❌ Error en la gráfica 3D")

def on_calculate(self, calc_type):
    try:
        func_str = self.func_entry.get().strip()
        if not func_str:
            raise ValueError("Ingresa una función para calcular.")
    
```

```

self.steps_text.delete('1.0', tk.END)

if calc_type == "derivada":

    order = int(self.deriv_order.get())

    point_str = self.eval_point.get().strip()

    method = self.deriv_method.get()

method_map = {

    "Auto (completo)": "completo",

    "Paso a paso completo": "completo",

    "Solo regla de la potencia": "regla_potencia",

    "Solo regla de la cadena": "regla_cadena",

    "Directo (SymPy)": "directo"

}

solver_method = method_map.get(method, "directo")

if point_str:

    point = float(point_str)

    res, steps = self.engine.eval_derivative(func_str, point, order, solver_method)

else:

    res, steps = self.engine.derivative(func_str, order, solver_method)

```

```
else: # integral

    method = self.calc_type.get()

if method == "Auto (mejor intento)":  
    res, steps = self.engine.indefinite(func_str)

elif method == "Integral indefinida":  
    res, steps = self.engine.indefinite(func_str)

elif method == "Integral definida":  
    a = parse_limit_string(self.lower_entry.get())  
    b = parse_limit_string(self.upper_entry.get())  
    res, steps = self.engine.definite(func_str, a, b)  
    self.on_plot()

elif method == "Integral impropia":  
    a = parse_limit_string(self.lower_entry.get())  
    b = parse_limit_string(self.upper_entry.get())  
    res, steps = self.engine.improper(func_str, a, b)  
    self.on_plot()

elif method == "Sustitución":  
    res, steps = self.engine.substitution_auto(func_str)

elif method == "Por partes":  
    res, steps = self.engine.integration_by_parts_auto(func_str)

elif method == "Fracciones parciales":  
    res, steps = self.engine.partial_fractions(func_str)
```

```

        elif method == "Sust. trigonométrica":
            res, steps = self.engine.trig_substitution(func_str)

        else: # Numérico (Simpson)

            a = parse_limit_string(self.lower_entry.get())
            b = parse_limit_string(self.upper_entry.get())
            res, steps = self.engine.numeric_simpson(func_str, a, b)
            self.on_plot()

            self.append_steps(steps, f'{grid} {calc_type.upper()}'")
            self.set_status("✅ Cálculo completado!")

    except Exception as e:
        messagebox.showerror("Error", f"No se pudo realizar el cálculo:\n{str(e)}")
        self.set_status("❌ Error en el cálculo")

def on_physics_calculate(self):
    try:
        problem_type = self.physics_problem.get()

        if problem_type == "Caída libre avanzada":
            altura = float(self.physics_param1.get())
            velocidad_inicial = float(self.physics_param2.get() or "0")
            masa = float(self.physics_param3.get() or "1.0")
    
```

```

result = self.engine.physics.caida_libre_tiempo(altura, velocidad_inicial,
masa=masa)

steps = [
    f'  CAÍDA LIBRE AVANZADA - Con resistencia del aire",
    f'Altura inicial: {altura:.4f} m",
    f'Velocidad inicial: {velocidad_inicial:.4f} m/s",
    f'Masa: {masa:.4f} kg",
    f"",
    f'  RESULTADOS:",
    f'• Tiempo sin resistencia: {result['tiempo_sin_resistencia']:.6f} s",
    f'• Tiempo con resistencia: {result['tiempo_con_resistencia']:.6f} s",
    f'• Diferencia: {result['diferencia_tiempo']:.6f} s
    ({result['diferencia_tiempo']}/{result['tiempo_sin_resistencia'])*100:.1f}%)",
    f'• Velocidad de impacto: {result['velocidad_impacto']:.6f} m/s",
    f'• Energía cinética: {result['energia_cinetica']:.6f} J",
    f"",
    f'  La resistencia del aire aumenta el tiempo de caída en
    {result['diferencia_tiempo']:.3f} segundos."
]
self.plot_manager.plot_physics_motion(result, 'caida_libre')

```

```

elif problem_type == "Movimiento de proyectil":
    velocidad = float(self.physics_param1.get())
    angulo = float(self.physics_param2.get())
    altura_inicial = float(self.physics_param3.get() or "0")

    result = self.engine.physics.movimiento_proyectil(velocidad, angulo,
                                                      altura_inicial)

    steps = [
        f"🚀 MOVIMIENTO DE PROYECTIL",
        f"Velocidad inicial: {velocidad:.4f} m/s",
        f"Ángulo: {angulo:.4f}°",
        f"Altura inicial: {altura_inicial:.4f} m",
        f"",
        f"📊 RESULTADOS:",
        f"• Alcance: {result['alcance']:.6f} m",
        f"• Altura máxima: {result['altura_maxima']:.6f} m",
        f"• Tiempo de vuelo: {result['tiempo_vuelo']:.6f} s",
        f"",
        f"💡 El proyectil alcanza {result['altura_maxima']:.1f} metros de altura."
    ]

```

```

elif problem_type == "Movimiento rectilíneo":
    velocidad_inicial = float(self.physics_param1.get())

```

```

aceleracion = float(self.physics_param2.get())
tiempo = float(self.physics_param3.get())

result = self.engine.physics.movimiento_rectilineo(velocidad_inicial,
aceleracion, tiempo)

steps = [
    f'💡 MOVIMIENTO RECTILÍNEO UNIFORMEMENTE ACELERADO',
    f'Velocidad inicial: {velocidad_inicial:.4f} m/s',
    f'Aceleración: {aceleracion:.4f} m/s2',
    f'Tiempo: {tiempo:.4f} s',
    f'',

    f'📊 RESULTADOS:',
    f'• Desplazamiento: {result["desplazamiento"]:.6f} m',
    f'• Velocidad final: {result["velocidad_final"]:.6f} m/s',
    f'• Energía cinética inicial: {result["energia_cinetica_inicial"]:.6f}',
    f'• Energía cinética final: {result["energia_cinetica_final"]:.6f}',
    f'• Trabajo neto: {result["trabajo"]:.6f}',
    f'',

    f'💡 Se requiere {result["trabajo"]:.1f} Julios de trabajo.'
]

self.plot_manager.plot_physics_motion(result, 'movimiento_rectilineo')

```



```

steps = [
    f' ⚙ RESISTENCIA DEL AIRE (Aproximación Ingeniería)" ,
    f'Masa del objeto: {masa:.4f} kg",
    f'Área frontal: {area:.4f} m2" ,
    f'Velocidad: {velocidad:.4f} m/s",
    f"",
    f' 📈 RESULTADOS:" ,
    f'• Fuerza de resistencia: {result['fuerza_resistencia']:.6f} N",
    f'• Peso del objeto: {result['peso']:.6f} N",
    f'• Velocidad terminal: {result['velocidad_terminal']:.6f} m/s",
    f'• Tiempo aprox. a terminal: {result['tiempo_aprox_terminal']:.6f} s",
    f'• Trayectoria simulada (posiciones finales): {result['trayectoria'][-1]:.6f}
    m"
]

```

```

elif problem_type == "Análisis de energía":
    masa = float(self.physics_param1.get())
    velocidad = float(self.physics_param2.get())

```

```

result = self.engine.physics.analisis_energia_cinetica(masa, velocidad)
steps = [
    f' 🔥 ANÁLISIS DE ENERGÍA CINÉTICA",
    f'Masa: {masa:.4f} kg",

```

```
f"Velocidad: {velocidad:.4f} m/s",
f"",
f" RESULTADOS:",
f"• Energía cinética: {result['energia_cinetica']:.6f} J",
f"• Momento lineal: {result['momento_lineal']:.6f} kg·m/s",
f"• Presión de impacto: {result['presion_impacto']:.6f} Pa",
f"• Equivalente en TNT: {result['equivalente_tnt']:.10f} kg",
f"",
f" Esta energía equivale a {result['equivalente_tnt']*1000:.3f} gramos de
TNT."
]
```

```
self.steps_text.delete('1.0', tk.END)
self._append_steps(steps, f" FÍSICA: {problem_type}")
```

```
self._set_status("✅ Cálculo de física completado", clear_after_ms=3000)
```

```
except Exception as e:
    messagebox.showerror("Error", f"No se pudo resolver el problema:\n{e}")
```

```
def on_analyze_points(self):
    try:
        func_str = self.func_entry.get().strip()
```

```

if not func_str:
    raise ValueError("Ingresa una función para analizar.")

xmin = float(self.xmin_entry.get())
xmax = float(self xmax_entry.get())

self._set_status(f'🔍 Analizando {func_str}...')

analysis = self.engine.critical_points(func_str, xlim=(xmin, xmax))

self.steps_text.delete('1.0', tk.END)
self.append_steps([f'Análisis de f(x) = {func_str}'], "📊 ANÁLISIS DE
PUNTOS")

all_points = []

if analysis['interseccion_y'] and analysis['interseccion_y'].get('y') is not None:
    iy = analysis['interseccion_y']
    self.steps_text.insert(tk.END, f"\n• Intersección Y: ({iy['x']:.6f},\n{iy['y']:.6f})\n")
    all_points.append(iy)

if analysis['raices']:

```

```

    self.steps_text.insert(tk.END, f"\n• Raíces encontradas:
{len(analysis['raices'])}\n")

    for r in analysis['raices']:
        self.steps_text.insert(tk.END, f" - ({r['x']:.6f}, {r['y']:.6f})\n")
        all_points.append(r)

if analysis['puntos_criticos']:
    self.steps_text.insert(tk.END, f"\n• Puntos críticos:
{len(analysis['puntos_criticos'])}\n")

    for pc in analysis['puntos_criticos']:
        self.steps_text.insert(tk.END, f" - {pc['tipo']}: ({pc['x']:.6f}, {pc['y']:.6f}) -
Curvatura: {pc['curvatura']:.6f}\n")
        all_points.append(pc)

self.plot_manager.mark_points(all_points, animate=False) # Desactivado
self._set_status("✅ Análisis completado!", clear_after_ms=3000)

except Exception as e:
    messagebox.showerror("Error", f"No se pudo analizar la función:\n{e}")

def _append_steps(self, steps, title=None):
    if title:
        self.steps_text.insert(tk.END, f"\n{title}", "title")

```

```

self.steps_text.insert(tk.END, "*len(title) + "\n\n")

for step in steps:
    if "ÁREA" in step.upper() or "AREA" in step.upper():
        self.steps_text.insert(tk.END, f"\n{step}\n", "area")
    elif "RESULTADO" in step.upper() or "VERIFICACIÓN" in step.upper():
        self.steps_text.insert(tk.END, f"\n{step}\n", "result")
    else:
        self.steps_text.insert(tk.END, f"\n{step}\n", "step")

    self.steps_text.insert(tk.END, "\n{'=50}\n\n")
    self.steps_text.see(tk.END)

def export_steps(self):
    try:
        content = self.steps_text.get('1.0', tk.END).strip()
        if not content:
            messagebox.showinfo("Info", "No hay contenido para exportar.")
        return

    filename = filedialog.asksaveasfilename(
        defaultextension=".txt",
        filetypes=[("Archivos de texto", "*.txt")]
)

```

```
)
```

```
if filename:
```

```
    with open(filename, 'w', encoding='utf-8') as f:
```

```
        f.write("ULTRA CALC 2D - RESULTADOS\n")
```

```
        f.write("*50 + \n\n")
```

```
        f.write(content)
```

```
    self._set_status("  Resultados exportados!")
```

```
    messagebox.showinfo("Éxito", f"Resultados guardados en:\n{filename}")
```

```
except Exception as e:
```

```
    messagebox.showerror("Error", f"No se pudo exportar:\n{str(e)}")
```

```
def export_plot(self):
```

```
    try:
```

```
        filename = filedialog.asksaveasfilename(
```

```
            defaultextension=".png",
```

```
            filetypes=[("PNG", "*.png"), ("PDF", "*.pdf")])
```

```
)
```

```
if filename:
```

```
    self.fig.savefig(filename, dpi=300, bbox_inches='tight',
```

```

        facecolor=self.theme['plot_bg'])

self._set_status("✅ Gráfica exportada!")

messagebox.showinfo("Éxito", f"Gráfica guardada en:\n{filename}")

```

except Exception as e:

```
    messagebox.showerror("Error", f"No se pudo exportar la gráfica:\n{str(e)}")
```

```
def on_clear_all(self):
```

```
    """Limpieza completa que incluye gráficas de física"""

    self.plot_manager.clear()
```

```
    self.steps_text.delete('1.0', tk.END)
```

```
    self.graph_title.config(text="🎲 Gráfica Interactiva")
```

```
    self.area_info_var.set("Área: No calculada")
```

```
    self._set_status("🧹 Todo limpiado!", clear_after_ms=2000)
```

```
#
```

---

```
# 🔥 INTEGRACIÓN AUTOMÁTICA DE CÁLCULO DE ÁREA BAJO LA CURVA
```

```
#
```

---

```
import matplotlib.pyplot as plt
```

```
import sympy as sp
```

```
import numpy as np
```

```
from scipy.integrate import quad
```

```

def calcular_area_y_graficar(func_str, a=-5, b=5, puntos=1000):
    """
    Calcula el área bajo la curva de f(x) en [a,b], muestra pasos
    y genera una nueva gráfica sombreada en rojo.

    """
    print("\n" + "="*70)
    print("CALCULO DE ÁREA BAJO LA CURVA".center(70))
    print("=*70)

x = sp.Symbol('x')
try:
    expr = sp.sympify(func_str.replace('^', '**'))
except Exception as e:
    print(f"🔴 Error al interpretar la función: {e}")
    return

# Paso 1: Mostrar la función
print(f" 1 Función ingresada: f(x) = {sp.pretty(expr)}")

# Paso 2: Calcular integral simbólica
try:
    integral_simb = sp.integrate(expr, (x, a, b))

```

```
print(f" 2 Integral simbólica: ∫ f(x) dx = {sp.pretty(integral_simb)}")
```

```
except Exception:
```

```
    integral_simb = None
```

```
    print("⚠️ No se pudo obtener integral simbólica exacta.")
```

```
# Paso 3: Cálculo numérico (verificación)
```

```
f_num = sp.lambdify(x, expr, 'numpy')
```

```
try:
```

```
    area_num, err = quad(f_num, a, b)
```

```
    print(f" 3 Aproximación numérica: {area_num:.6f} ± {err:.2e}")
```

```
except Exception as e:
```

```
    print(f" ✗ Error en cálculo numérico: {e}")
```

```
return
```

```
# Paso 4: Mostrar resultado final
```

```
print("-"*70)
```

```
print(f"\033[92m ✓ Área bajo la curva en [{a}, {b}] = {area_num:.6f}\033[0m")
```

```
print("-"*70)
```

```
# Paso 5: Graficar función y área
```

```
xs = np.linspace(a, b, puntos)
```

```
ys = f_num(xs)
```

```

plt.figure(figsize=(8, 5))

plt.plot(xs, ys, color='#00d4ff', linewidth=3, label=f'f(x) = {func_str}')

plt.fill_between(xs, ys, 0, where=(ys >= 0), color='red', alpha=0.35, label='Área
positiva')

plt.fill_between(xs, ys, 0, where=(ys < 0), color='red', alpha=0.15, label='Área
negativa')

```

```

plt.axhline(0, color='white', linewidth=1.5)

plt.title("🌈 Área bajo la curva", fontsize=14, weight='bold', color="#7eff7e")

plt.xlabel("x", fontsize=11)

plt.ylabel("f(x)", fontsize=11)

plt.legend(facecolor="#131a2c", edgecolor="#7eff7e", fontsize=9)

plt.grid(alpha=0.3)

plt.show()

```

#

---

```
# 🔗 VINCULAR FUNCIÓN AL FLUJO DE GRAFICACIÓN PRINCIPAL
```

#

---

```
# Guardar método original
```

---

```
_original_plot_function = ModernPlotManager.plot_function
```

```
def _plot_function_con_area(self, func_str, xlim=(-5,5), color_index=0, animate=True,
show_area=True, area_limits=None):
```

""""

Extiende la función original de graficación para incluir  
el cálculo del área bajo la curva en un gráfico separado.

""""

```
# Llamar al método original
_original_plot_function(self, func_str, xlim, color_index, animate, show_area,
area_limits)
```

try:

```
# Después de graficar, lanzar el análisis de área
print("\n⭐️ Generando gráfico adicional de área bajo la curva...")
```

```
calcular_area_y_graficar(func_str, xlim[0], xlim[1])
```

except Exception as e:

```
print(f"⚠️ Error en cálculo de área adicional: {e}")
```

```
# Reemplazar el método original por el extendido
```

```
ModernPlotManager.plot_function = _plot_function_con_area
```

```
print("✅ [INTEGRACIÓN ACTIVA] Cada vez que se grafique una función, se
calculará y mostrará el área bajo la curva.")
```

```
# ----- EJECUCIÓN PRINCIPAL -----  
  
if __name__ == "__main__":  
  
    try:  
  
        print("🚀 INICIANDO ULTRA CALC 2D - VERSIÓN MEJORADA...")  
  
        print("🌐 Cargando interfaz moderna...")  
  
        print("💻 Inicializando motor matemático avanzado...")  
  
        print("🌐 Configurando física profesional...")  
  
  
        root = tk.Tk()  
  
        app = UltraCalc2DModernApp(root)  
  
  
        print("✅ SISTEMA MEJORADO LISTO!")  
  
        print("💡 NUEVAS CARACTERÍSTICAS:")  
  
        print("• Calculadora compacta y eficiente")  
  
        print("• Física avanzada con resistencia del aire")  
  
        print("• Gráficas modernas y atractivas")  
  
        print("• Área bajo la curva automática en rojo")  
  
        print("• Valor numérico del área en gráfica y panel")  
  
        print("• Información detallada de áreas")  
  
        print("• Presentación más ordenada y profesional")  
  
        print("• Gráficas 3D interactivas")
```

```
root.mainloop()

except Exception as e:
    print(f"❌ ERROR: {e}")

    print("💡 Verifica las dependencias: pip install numpy sympy matplotlib tkinter
pillow scipy")
```

## Bibliografía

### 1. Python para todos – Charles Severance

Este libro es una introducción accesible a la programación en Python, ideal para principiantes. Cubre desde los fundamentos del lenguaje hasta conceptos más avanzados.

### 2. Python Crash Course – Eric Matthes

Una guía práctica que enseña Python a través de proyectos reales, incluyendo aplicaciones gráficas y análisis de datos.

### 3. Python Data Science Handbook – Jake VanderPlas

Este manual es esencial para quienes deseen aplicar Python en ciencia de datos, cubriendo bibliotecas como NumPy, Matplotlib, Pandas y Scikit-learn.

### 4. Fluent Python – Luciano Ramalho

Un libro avanzado que profundiza en las características más poderosas y complejas de Python, adecuado para desarrolladores experimentados.

## Recursos en línea

### **Tkinter (Interfaz Gráfica de Usuario)**

- **Tutorial de Tkinter en GeeksforGeeks:** Una guía completa para comenzar con Tkinter, desde la instalación hasta la creación de aplicaciones GUI básicas. [GeeksforGeeks](#)
- **Documentación oficial de Tkinter:** Información detallada sobre los módulos y clases disponibles en Tkinter. [Python documentation](#)
- **Tutorial de Real Python sobre Tkinter:** Un tutorial interactivo que cubre la creación de aplicaciones GUI con Tkinter. [realpython.com](#)

### **Sympy (Cálculo Simbólico)**

- **Documentación oficial de Sympy:** La fuente principal para aprender sobre el cálculo simbólico en Python. [docs.sympy.org](#)
- **Guía rápida de Sympy en TutorialsPoint:** Una introducción concisa a las funcionalidades de Sympy. [tutorialspoint.com](#)

### **Matplotlib (Visualización de Datos)**

- **Tutoriales oficiales de Matplotlib:** Una colección de tutoriales que cubren desde gráficos básicos hasta técnicas avanzadas de visualización. [matplotlib.org](#)
- **Tutorial de W3Schools sobre Matplotlib:** Una guía paso a paso para crear gráficos con Matplotlib. [w3schools.com](#)

### **SciPy (Cálculo Numérico)**

- **Documentación oficial de SciPy:** Información detallada sobre las funciones de integración y resolución de ecuaciones diferenciales. [docs.scipy.org](#)
- **Tutorial de SciPy en GeeksforGeeks:** Ejemplos prácticos de cómo utilizar SciPy para integración numérica. [GeeksforGeeks](#)

### **GitHub (Control de Versiones y Colaboración)**

- **Guía de colaboración en GitHub:** Un recurso completo sobre cómo colaborar eficazmente en proyectos utilizando GitHub. [Medium](#)
- **Documentación oficial de GitHub sobre colaboración con pull requests:** Información detallada sobre cómo gestionar contribuciones en proyectos colaborativos. [GitHub Docs](#)
- **Guía de GitHub para principiantes:** Un tutorial interactivo para aprender los fundamentos de Git y GitHub. [GitHub Docs](#)
- **Real Python:** Ofrece tutoriales y artículos sobre diversos temas de Python, incluyendo desarrollo de interfaces gráficas y análisis de datos. [realpython.com](#)
- **GeeksforGeeks:** Una plataforma educativa que proporciona ejemplos y explicaciones sobre programación en Python y otras tecnologías. [GeeksforGeeks](#)
- **TutorialsPoint:** Ofrece tutoriales sobre diversas bibliotecas de Python, incluyendo SciPy y Matplotlib. [tutorialspoint.com](#)

### **Comunicaciones en grupo**

- **Google Meet** Plataforma para reuniones virtuales y comunicación verbal
- **WhatsApp** Comunicación escrita e instantánea