Facultad de Ingeniería, Universidad Mariano Gálvez de Guatemala

Código: 013

Hugo Montoya

Imagen que contiene Logotipo

El contenido generado por IA puede ser incorrecto.

**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](#_Toc212207802)

[Introducción 4](#_Toc212207803)

[Planeación y estrategia 5](#_Toc212207804)

[Desarrollo del proyecto 6](#_Toc212207805)

[Como Funciona la calculadora 7](#_Toc212207806)

[Ejemplo practico 10](#_Toc212207807)

[Ejemplo de cómo ingresar una función a la calculadora 13](#_Toc212207808)

[Como se llevó a cabo cada reunión de equipo. 14](#_Toc212207809)

[Líneas del código 15](#_Toc212207810)

[Bibliografía 130](#_Toc212207811)

# Resumen

Este proyecto se basa en la investigación, elaboración y practica 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 practico, 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 fatos 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 grafica 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

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

*Nota.* Interfaz principal de la calculadora.

Figura 2

Vista del cálculo área bajo la curva

Gráfico

El contenido generado por IA puede ser incorrecto.

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

Texto

El contenido generado por IA puede ser incorrecto.

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

# Ejemplo practico

##### 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 esta modelada por la función 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

Sustituimos la función de aceleración

Para encontrar la constante de integración (C1), usamos la condición inicial: el automóvil parte del reposo en t = 0, lo que significa v (0) = 0

Por lo tanto, la función de velocidad es:

1. 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)

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

Calculamos la antiderivada:

0

0

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

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

conclusión: El problema de 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

* 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

En el intervalo [0,3]

Ingresamos la función x\*\*2+2\*x +1 (en Python, los exponentes se escriben con \*\* (por ejemplo, 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 .
* El **área sombreada bajo la curva** entre y .
* 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.

## 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 (suelo)

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"∫ 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}"]

# 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

return 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

return 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

return 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', 'xⁿ'],

['(', ')', '|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 ['⌫', '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', '√(': '√',

'π': 'π', 'e': 'e', '|x|': '|x|',

'x²': 'x²', 'xⁿ': 'xⁿ', '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()

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\_str.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 físicos 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, "📊 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 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 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(self.xmax\_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"🧮 {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')

elif problem\_type == "Velocidad promedio":

distancia = float(self.physics\_param1.get())

tiempo = float(self.physics\_param2.get())

result = self.engine.physics.velocidad\_promedio(distancia, tiempo)

steps = [

f"📏 VELOCIDAD PROMEDIO",

f"Distancia total: {distancia:.4f} m",

f"Tiempo total: {tiempo:.4f} s",

f"",

f"📊 RESULTADOS:",

f"• Velocidad promedio: {result['velocidad\_promedio']:.6f} m/s",

f"• En km/h: {result['kmh']:.6f}",

f"• En mph: {result['mph']:.6f}",

]

elif problem\_type == "Resistencia del aire":

masa = float(self.physics\_param1.get())

area = float(self.physics\_param2.get())

velocidad = float(self.physics\_param3.get() or "10")

result = self.engine.physics.resistencia\_aire\_aproximada(masa, area, velocidad=velocidad)

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}, {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"{title}\n", "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"{step}\n", "area")

elif "RESULTADO" in step.upper() or "VERIFICACIÓN" in step.upper():

self.steps\_text.insert(tk.END, f"{step}\n", "result")

else:

self.steps\_text.insert(tk.END, f"{step}\n", "step")

self.steps\_text.insert(tk.END, f"\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("🧮 CÁLCULO 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"1️⃣ Función ingresada: f(x) = {sp.pretty(expr)}")

# Paso 2: Calcular integral simbólica

try:

integral\_simb = sp.integrate(expr, (x, a, b))

print(f"2️⃣ 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"3️⃣ 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](https://www.geeksforgeeks.org/python/python-tkinter-tutorial/?utm_source=chatgpt.com)
* **Documentación oficial de Tkinter**: Información detallada sobre los módulos y clases disponibles en Tkinter. [Python documentation](https://docs.python.org/3/library/tkinter.html?utm_source=chatgpt.com)
* **Tutorial de Real Python sobre Tkinter**: Un tutorial interactivo que cubre la creación de aplicaciones GUI con Tkinter. [realpython.com](https://realpython.com/python-gui-tkinter/?utm_source=chatgpt.com)

**SymPy (Cálculo Simbólico)**

* **Documentación oficial de SymPy**: La fuente principal para aprender sobre el cálculo simbólico en Python. [docs.sympy.org](https://docs.sympy.org/?utm_source=chatgpt.com)
* **Guía rápida de SymPy en TutorialsPoint**: Una introducción concisa a las funcionalidades de SymPy. [tutorialspoint.com](https://www.tutorialspoint.com/sympy/sympy_quick_guide.htm?utm_source=chatgpt.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](https://matplotlib.org/stable/tutorials/index.html?utm_source=chatgpt.com)
* **Tutorial de W3Schools sobre Matplotlib**: Una guía paso a paso para crear gráficos con Matplotlib. [w3schools.com](https://www.w3schools.com/python/matplotlib_intro.asp?utm_source=chatgpt.com)

**SciPy (Cálculo Numérico)**

* **Documentación oficial de SciPy**: Información detallada sobre las funciones de integración y resolución de ecuaciones diferenciales. [docs.scipy.org](https://docs.scipy.org/doc/scipy/tutorial/integrate.html?utm_source=chatgpt.com)
* **Tutorial de SciPy en GeeksforGeeks**: Ejemplos prácticos de cómo utilizar SciPy para integración numérica. [GeeksforGeeks](https://www.geeksforgeeks.org/python/scipy-integration/?utm_source=chatgpt.com)

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

* **Guía de colaboración en GitHub**: Un recurso completo sobre cómo colaborar eficazmente en proyectos utilizando GitHub. [Medium](https://medium.com/%40jonathanmines/the-ultimate-github-collaboration-guide-df816e98fb67?utm_source=chatgpt.com)
* **Documentación oficial de GitHub sobre colaboración con pull requests**: Información detallada sobre cómo gestionar contribuciones en proyectos colaborativos. [GitHub Docs](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests?utm_source=chatgpt.com)
* **Guía de GitHub para principiantes**: Un tutorial interactivo para aprender los fundamentos de Git y GitHub. [GitHub Docs](https://docs.github.com/get-started/quickstart/hello-world?utm_source=chatgpt.com)
* **Real Python**: Ofrece tutoriales y artículos sobre diversos temas de Python, incluyendo desarrollo de interfaces gráficas y análisis de datos. [realpython.com](https://realpython.com/python-gui-tkinter/?utm_source=chatgpt.com)
* **GeeksforGeeks**: Una plataforma educativa que proporciona ejemplos y explicaciones sobre programación en Python y otras tecnologías. [GeeksforGeeks](https://www.geeksforgeeks.org/python/python-tkinter-tutorial/?utm_source=chatgpt.com)
* **TutorialsPoint**: Ofrece tutoriales sobre diversas bibliotecas de Python, incluyendo SciPy y Matplotlib. [tutorialspoint.com](https://www.tutorialspoint.com/sympy/sympy_quick_guide.htm?utm_source=chatgpt.com)

**Comunicaciones en grupo**

* **Google Meet** plataforma para reuniones virtuales y comunicación verbal
* **WhatsApp** Comunicación escrita e instantánea