



**Universidad Cooperativa
de Colombia**

Fecha: 5/17/2025

Proyecto final: GeoCalculus Multivriable

Docente: Juan Granja

Carrera: Ingeniería de software

Integrantes:

Juan Rafael Calzada

Jonar Andrés Martínez

Juan Camilo Martínez

En el estudio del cálculo multivariado, una de las necesidades principales es visualizar funciones con más de una variable, analizar sus derivadas parciales y comprender su comportamiento gráfico tanto en dos como en tres dimensiones. Muchos estudiantes y profesionales enfrentan dificultades al interpretar funciones multivariadas debido a la falta de herramientas interactivas y visuales accesibles.

El presente proyecto, titulado **GeoCalculus Multivariable**, surge como una solución educativa y práctica que permite al usuario introducir funciones matemáticas y obtener sus gráficas en 2D y 3D, así como calcular sus derivadas parciales. Este sistema se ha desarrollado utilizando Python, combinando herramientas como **Tkinter** para la interfaz gráfica, **Matplotlib** para las gráficas y **SymPy** para el cálculo simbólico.

Fundamentos Teóricos

El proyecto "GeoCalculus Multivariable" se apoya en múltiples conceptos del cálculo diferencial multivariado y en herramientas del álgebra simbólica, la visualización matemática y la computación científica. A continuación, se describen los principales fundamentos teóricos que sustentan su funcionamiento:

1. Funciones multivariables

Una función multivariable es una función que depende de dos o más variables independientes, como $f(x,y)$, $f(x,y,z)$, etc. Estas funciones se utilizan para modelar fenómenos físicos, económicos y de ingeniería donde varios factores afectan simultáneamente un resultado. El programa admite funciones con una o más variables, lo cual permite tanto el estudio univariable (gráficas 2D) como multivariable (gráficas 3D).

2. Derivadas parciales

Las derivadas parciales representan la tasa de cambio de una función multivariable respecto a una de sus variables, manteniendo las demás constantes. Son fundamentales para entender el comportamiento local de una superficie o campo escalar.

Matemáticamente, si se tiene una función $f(x,y)$, la derivada parcial respecto a x se define como:

$$\frac{\partial f}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h, y) - f(x, y)}{h}$$

El programa utiliza estas derivadas para mostrar cómo cambia la función con respecto a cada variable. Se calculan utilizando el módulo `sympy.diff`, que permite obtener las derivadas simbólicas.

3. Gradiente

La aplicación identifica las variables de la función (como x , y , etc.) y calcula las **derivadas parciales** respecto a cada una. Estas derivadas indican cómo cambia la función al modificar cada variable individualmente.

Luego, **agrupa todas las derivadas parciales en un vector**, que representa el **gradiente**. Este vector apunta en la dirección de máximo crecimiento de la función.

4. Visualización gráfica de funciones

- **Gráficas 2D:** Para funciones univariantes $f(x)$, se representa la relación entre una variable independiente y una dependiente. Estas gráficas permiten entender el crecimiento, decrecimiento, extremos y concavidad de una función.
- **Gráficas 3D:** Para funciones bivariantes $f(x,y)$, la representación en el espacio tridimensional permite visualizar superficies. Estas gráficas muestran detalles como crestas, valles, puntos silla y comportamiento global de la función.

Se emplea la biblioteca matplotlib para construir estas representaciones, utilizando mallas generadas por `numpy.meshgrid` en el caso 3D, y funciones de interpolación mediante `lambdify`.

5. Álgebra simbólica con SymPy

El núcleo del procesamiento simbólico del programa se basa en la librería `sympy`, que permite:

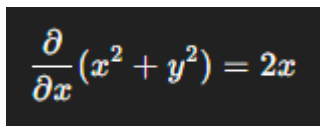
- Convertir cadenas de texto en expresiones simbólicas (función `sympify`).
- Identificar variables independientes automáticamente (`expr.free_symbols`).
- Derivar expresiones simbólicas con respecto a variables específicas (`diff`).
- Traducir expresiones a notación LaTeX (`latex`) para visualizarlas de forma legible.

Este enfoque hace posible un procesamiento algebraico automático de expresiones matemáticas, lo que facilita la enseñanza, el aprendizaje y la verificación manual por parte del estudiante.

6. Representación simbólica en LaTeX

El uso de LaTeX permite mostrar las derivadas parciales de manera legible, como si fueran escritas en papel. Esto es especialmente útil en el entorno académico, ya que mejora la comprensión visual de los resultados matemáticos.

Se utiliza matplotlib para renderizar el texto en LaTeX dentro de las gráficas, mostrando fórmulas como:



$$\frac{\partial}{\partial x}(x^2 + y^2) = 2x$$

Esto refuerza el aprendizaje teórico, al relacionar directamente la expresión simbólica con su resultado derivado.

7. Evaluación numérica de funciones

Para graficar las funciones, es necesario transformar las expresiones simbólicas en funciones numéricas que puedan aplicarse sobre arrays de numpy. Esto se logra mediante `sympy.lambdify`, que traduce una expresión simbólica a una función que acepta entradas de tipo float o `numpy.ndarray`. Así, se puede evaluar eficientemente la función sobre un dominio continuo de valores y construir las gráficas correspondientes.

8. Interacción hombre-máquina en el aprendizaje del cálculo

Aunque no es un fundamento matemático en sí mismo, la interacción que este programa ofrece permite un enfoque más experimental y exploratorio del cálculo. Los usuarios pueden ingresar sus propias funciones, observar las gráficas, calcular derivadas y experimentar con ejemplos aleatorios, promoviendo así un aprendizaje activo, visual e intuitivo.

Diseño de la solución

1. Entrada y análisis de la función

El primer módulo recibe la entrada del usuario en forma de cadena de texto, por ejemplo: `"x**2 + y**2"`. Esta entrada es procesada por `sympy.sympify()` para convertirla en una expresión simbólica válida. El sistema valida automáticamente:

- La sintaxis de la función (verifica paréntesis, operadores, variables).
- Las variables involucradas (`expr.free_symbols`) para determinar si se debe graficar en 2D o 3D.
- El número de variables, permitiendo una visualización dinámica que se adapta al contexto.

Ventaja: El usuario puede ingresar funciones sin preocuparse por detalles de implementación, promoviendo la exploración libre de ideas matemáticas.

2. Cálculo de derivadas parciales

Una vez definida la función simbólica, se generan las derivadas parciales respecto a cada variable utilizando `sympy.diff()`. Esto se realiza en un ciclo que recorre todas las variables identificadas en la expresión:

```
for var in expr.free_symbols:
```

```
    derivadas[var] = diff(expr, var)
```

Estas derivadas son guardadas en un diccionario para facilitar su posterior visualización y análisis.

Valor: Al mostrar todas las derivadas parciales, se refuerza la comprensión del cálculo local de una superficie o función escalar.

3. Visualización gráfica dinámica

Según el número de variables detectadas, el sistema elige entre dos tipos de gráficas:

- **Gráfica 2D:** Si solo hay una variable (e.g., xxx), se genera una gráfica de líneas usando `matplotlib.pyplot.plot`.
- **Gráfica 3D:** Si hay dos variables (e.g., xxx y yyy), se genera una superficie en 3D con `matplotlib.pyplot.plot_surface`, utilizando una malla de valores generada con `numpy.meshgrid`.

Aspecto técnico relevante: La función original es convertida en una función numérica con `sympy.lambdify()` para poder evaluar arrays de `numpy`, lo cual permite una visualización eficiente sobre un dominio continuo.

4. Visualización simbólica de derivadas (LaTeX render)

Cada derivada parcial es representada simbólicamente en una celda gráfica utilizando `matplotlib.pyplot.text()` y `sympy.latex()`.

Esto permite integrar los resultados simbólicos directamente en la interfaz visual, haciendo del sistema una herramienta que fusiona teoría matemática con intuición visual.

5. Generación de funciones aleatorias (modo exploratorio)

El sistema incluye un generador de funciones aleatorias multivariantes, diseñado para fomentar el descubrimiento y el análisis exploratorio. Estas funciones se generan combinando operadores, funciones trigonométricas, polinomios y raíces. Por ejemplo:

$$f(x, y) = \sin(x*y) + x^2 - \ln(y)$$

Uso pedagógico: Este módulo permite crear ejercicios aleatorios para práctica autónoma o evaluación formativa.

6. Interfaz e interacción con el usuario

El diseño de la interfaz en consola está pensado para ser claro y funcional. El usuario tiene la posibilidad de:

- Ingresar su propia función.
- Solicitar una función aleatoria.
- Ver las derivadas parciales simbólicamente.
- Ver la gráfica correspondiente.

Además, se puede extender fácilmente a una interfaz gráfica con bibliotecas como tkinter o streamlit.

7. Estructura del código y modularidad

El sistema está dividido en funciones independientes, lo que permite su mantenimiento, extensión y reutilización:

- `procesar_funcion()`: convierte texto a expresión simbólica.
- `calcular_derivadas()`: obtiene derivadas parciales.
- `graficar_2d()` y `graficar_3d()`: generan visualizaciones.
- `mostrar_derivadas_latex()`: presenta resultados simbólicamente.

Esta estructura cumple con principios de ingeniería de software como separación de responsabilidades, reutilización y escalabilidad.

8. Posibilidades de ampliación

El diseño modular permite futuras mejoras, tales como:

- Incorporación de campos vectoriales y gradientes.
- Cálculo del jacobiano y hessiano.
- Análisis de máximos/mínimos locales con pruebas de la segunda derivada.
- Implementación web para educación remota.

Capturas de Pantalla y Resultados

Figura 1: Interfaz Principal

Panel de entrada (izquierda) y pestañas de visualización (derecha).

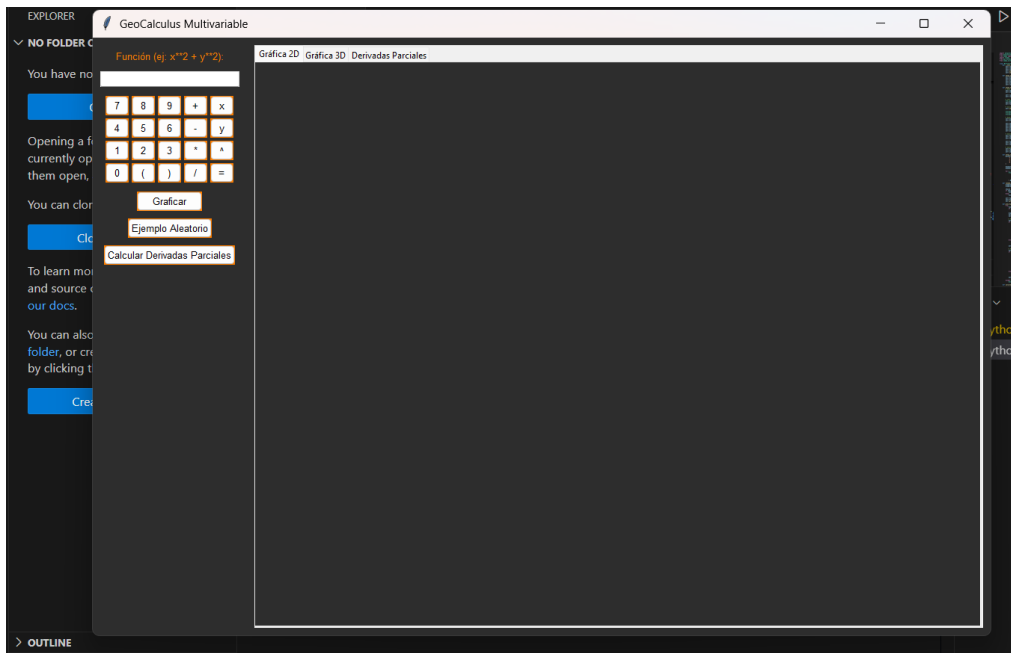


Figura 2: Gráfica 3D de $\sin(x) + \cos(y)$

Descripción: Superficie parabólica generada por el aplicativo.

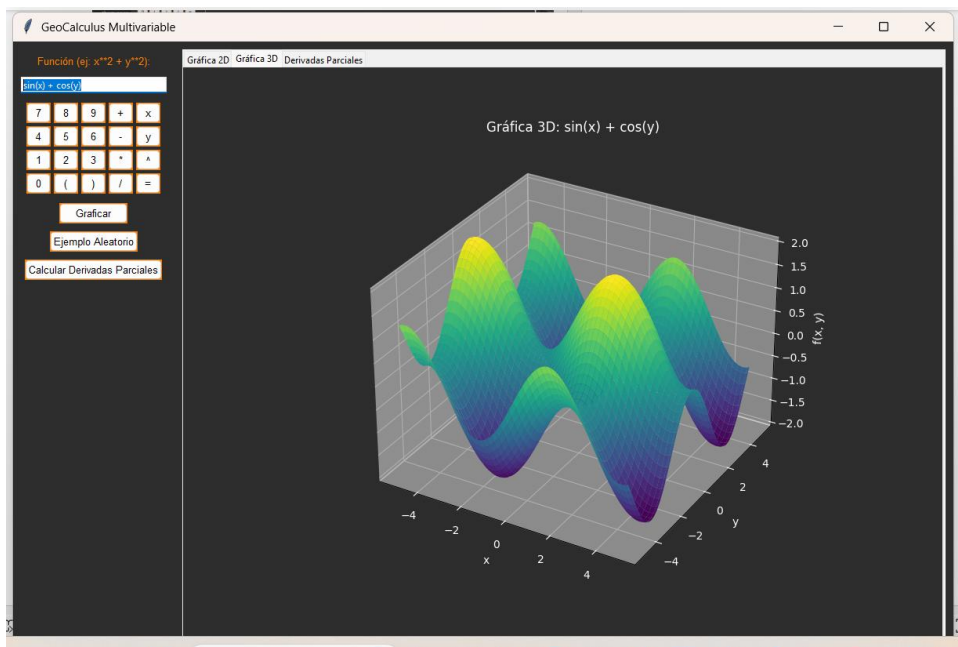


Figura 3: Gráfica 2D de sin(x)

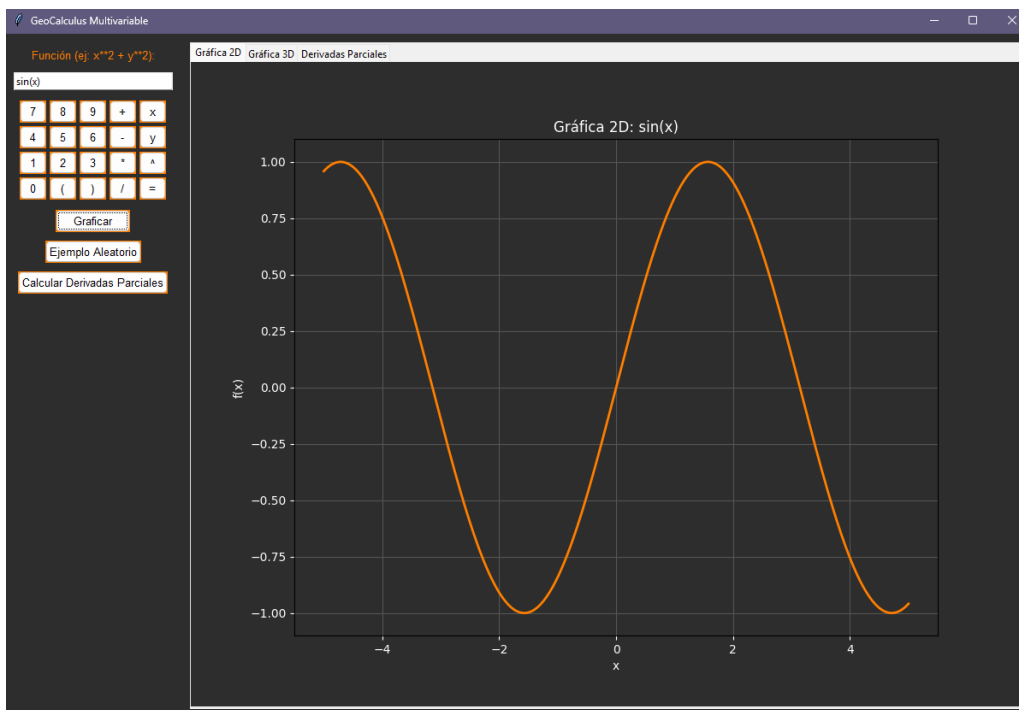


Figura 4: Derivadas parciales de $x^2 + 2y - \sin(x)$

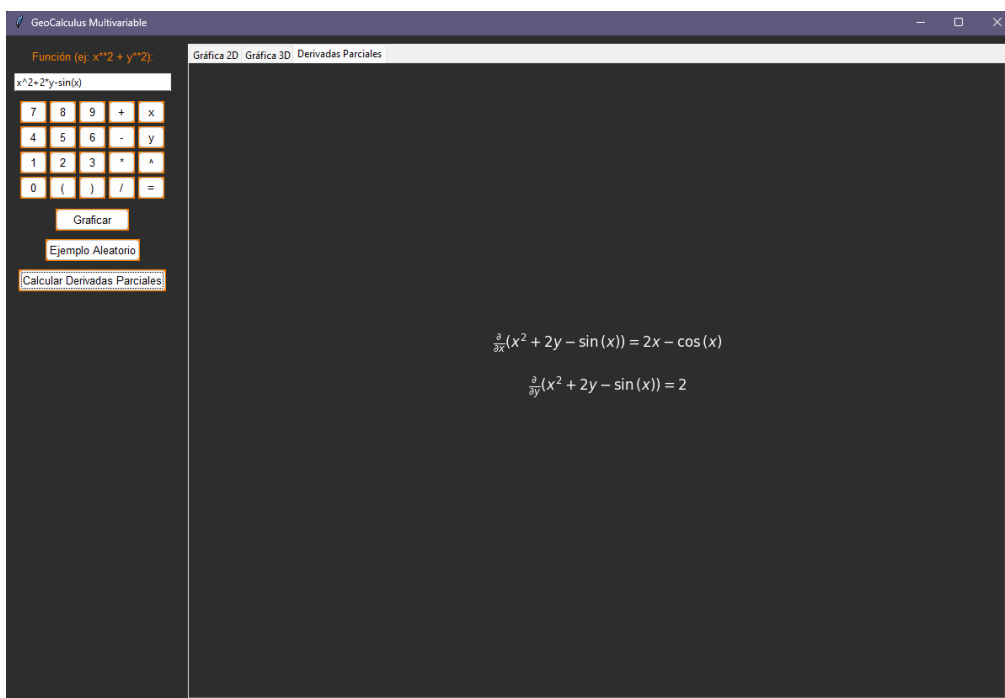


Figura 5: Gradiente de $x^2+2y-\sin(x)$

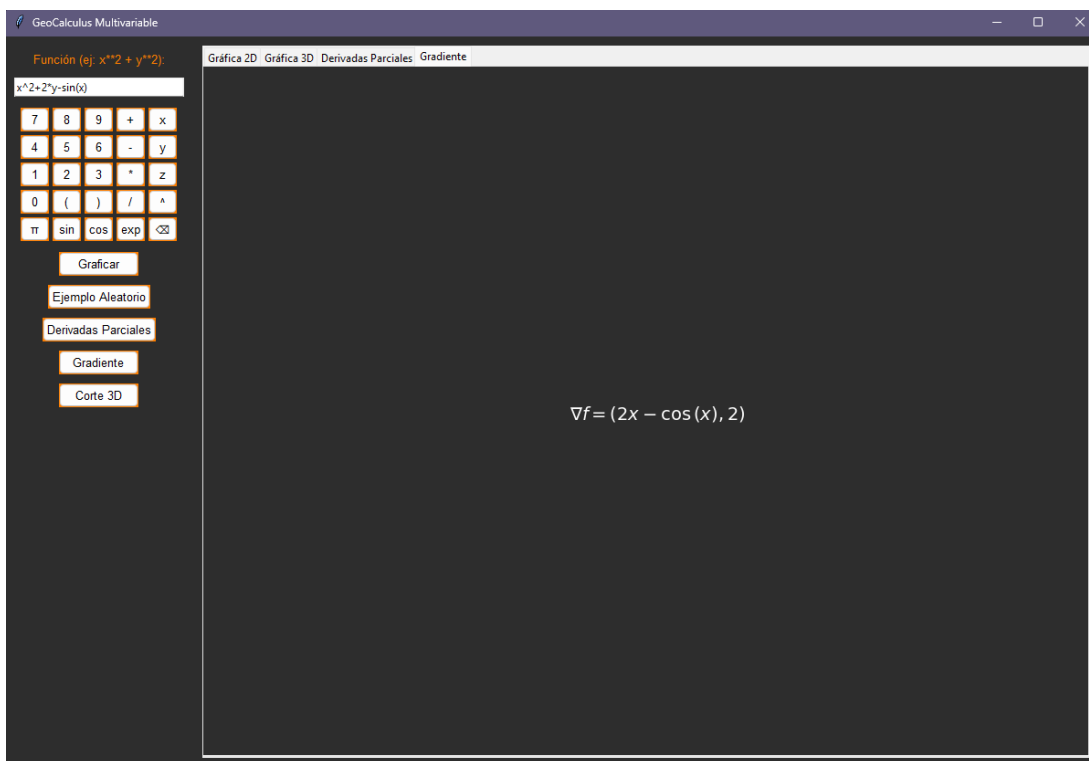
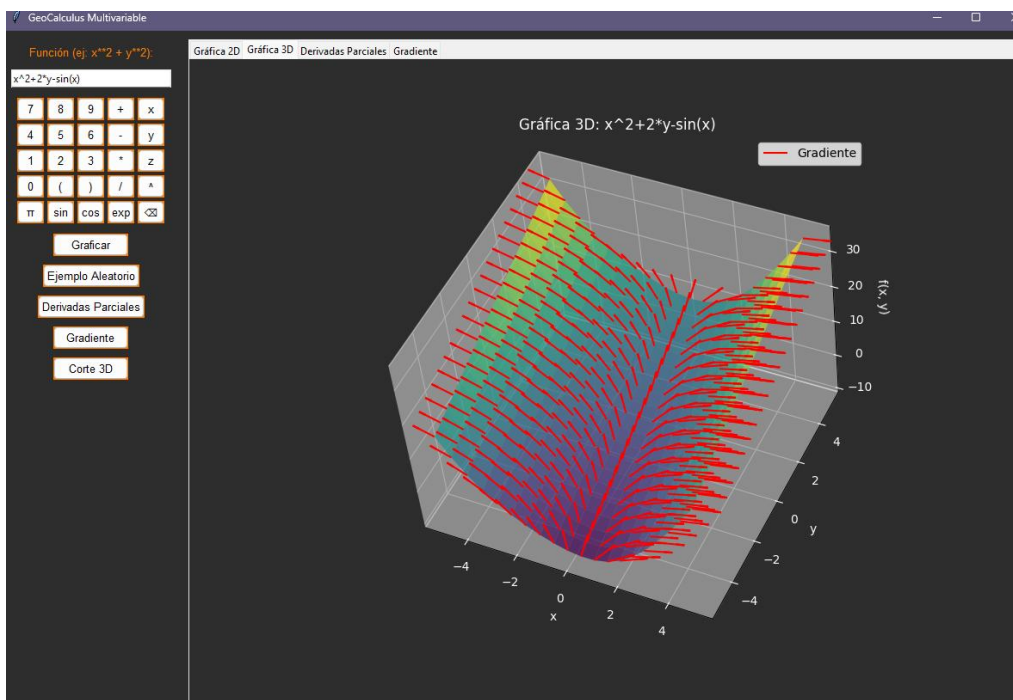


Figura 6: Gráfica y gradiente de $x^2+2y-\sin(x)$



Resultado

La aplicación permite calcular automáticamente el gradiente de funciones multivariadas escritas por el usuario. Identifica las variables, deriva respecto a cada una y muestra el vector gradiente de forma clara. Esto facilita el análisis matemático y reduce errores al derivar manualmente, siendo útil para estudiantes y profesionales que trabajan con cálculo vectorial o funciones de varias variables.

Conclusiones y Mejoras Futuras

Conclusiones

El proyecto **GeoCalculus Multivariable** logra cumplir con su objetivo educativo de permitir la interpretación gráfica y analítica de funciones multivariadas. Facilita el aprendizaje y análisis del cálculo multivariado mediante una herramienta práctica, visual y de fácil interacción.

Además, al incorporar tanto el análisis simbólico como el numérico, el sistema ofrece una visión completa del comportamiento de funciones, algo especialmente útil en el contexto académico.

Posibles mejoras

1. Validación avanzada de entrada:
 - Detectar errores de sintaxis en la función escrita por el usuario.
 - Mostrar mensajes claros cuando la entrada no sea válida.
2. Soporte para funciones con más operaciones
 - Incluir funciones trigonométricas, logarítmicas, exponenciales y combinaciones más complejas.
3. Interfaz gráfica más intuitiva
 - Mejorar el diseño visual con estilos modernos (por ejemplo, usando ttk o bibliotecas como customtkinter).
 - Añadir etiquetas más claras y separadores entre la entrada y el resultado.
4. Historial de cálculos
 - Guardar funciones anteriores y sus gradientes para consultarlos luego.
5. Exportar resultados
 - Opción para guardar el resultado del gradiente en un archivo (como .txt o .pdf).

Link repositorio: <https://github.com/Juan-Camilo-Martinez-B/CalculoMultivariado.git>

Referencias bibliográficas

- Anton, H., Bivens, I., & Davis, S. (2013). Cálculo multivariable (10ª ed.). Cengage Learning.
- Grinberg, M. (2018). Fluent Python: Clear, Concise, and Effective Programming. O'Reilly Media.
- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. <https://doi.org/10.1038/s41586-020-2649-2>
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Langtangen, H. P. (2016). A Primer on Scientific Programming with Python (5th ed.). Springer. <https://doi.org/10.1007/978-3-662-49887-3>
- McKinney, W. (2012). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media.
- Oliphant, T. E. (2007). Python for Scientific Computing. *Computing in Science & Engineering*, 9(3), 10–20. <https://doi.org/10.1109/MCSE.2007.58>
- Stewart, J. (2018). Cálculo multivariable (8ª ed.). Cengage Learning.
- SymPy Development Team. (2024). SymPy: Python library for symbolic mathematics. <https://www.sympy.org>