

**Aplicativo Interactivo Para Cálculo Simbólico, Numérico Y Visualización de Conceptos de
Cálculo Multivariado**

David Fernando Ramírez de la Parra

Daniers Alexander Solarte Limas

Juan Felipe Mora Revelo

Universidad Cooperativa de Colombia

Ingeniería de software – Cuarto semestre

Cálculo Multivariado

Profesor: Juan Pablo Granja Hinestrosa

Pasto, Colombia

27 de mayo de 2025

Aplicativo Interactivo Para Cálculo Simbólico, Numérico Y Visualización de Conceptos de Cálculo Multivariado

Informe técnico.

Introducción y contexto del problema.

En el estudio del Cálculo Multivariado, muchos de los conceptos clave como derivadas parciales, optimización con restricciones e integración múltiple presentan una dificultad significativa para los estudiantes, especialmente por la abstracción y complejidad matemática que implican. Estas dificultades se acentúan al intentar visualizar las superficies o interpretar gráficamente regiones de integración en dos o tres dimensiones.

Este proyecto nace con el objetivo de crear una herramienta computacional interactiva que facilite el aprendizaje y la aplicación de los conceptos más importantes del cálculo multivariado, mediante un enfoque visual, simbólico y numérico. La aplicación desarrollada permite no solo calcular derivadas parciales o integrales múltiples, sino también generar representaciones gráficas intuitivas y obtener resultados exactos con notación matemática formal, integrando tecnologías como SymPy para el cálculo simbólico y Matplotlib para la visualización profesional.

Fundamentos teóricos aplicados.

El programa se basa en los siguientes conceptos de cálculo multivariado:

- **Derivadas parciales:** Se calculan con respecto a una o más variables, permitiendo analizar cómo varía una función multivariable respecto a una de sus entradas.
- **Puntos críticos y clasificación:** Se identifican los máximos, mínimos y puntos silla de una función de dos variables utilizando la segunda derivada y el determinante de la matriz Hessiana.
- **Multiplicadores de Lagrange:** Método para hallar máximos o mínimos de funciones con restricciones, útil en optimización bajo condiciones impuestas.
- **Integración doble y triple:** Utilizada para calcular áreas, volúmenes y otras magnitudes físicas sobre regiones definidas en el plano o el espacio.

- **Superficies cuadráticas:** Clasificación y reconocimiento automático de superficies como elipsoides, paraboloides e hiperboloides, mediante su forma analítica.

El uso de herramientas computacionales refuerza la comprensión de estos temas al permitir ver gráficamente las funciones, sus derivadas, regiones de integración y restricciones.

Diseño de la solución.

Lenguaje y herramientas utilizadas:

- **Python 3.**
- **Librería Sympy:** Para el manejo simbólico de expresiones matemáticas.
- **Librería Matplotlib:** Para visualizaciones 2D y 3D.
- **Librería NumPy:** Para cálculos numéricos y creación de mallas de puntos.

Flujo de trabajo:

1. El usuario ingresa una función de varias variables.
2. El usuario selecciona el tipo de operación a realizar (derivada parcial, integral doble, optimización, etc.).
3. El sistema procesa el cálculo y muestra el resultado simbólico o numérico correspondiente.
4. En caso de requerir visualización, el usuario debe activar manualmente la función de graficación para observar la superficie o región relacionada, ya que esta no se genera automáticamente al obtener el resultado.

Arquitectura del programa:

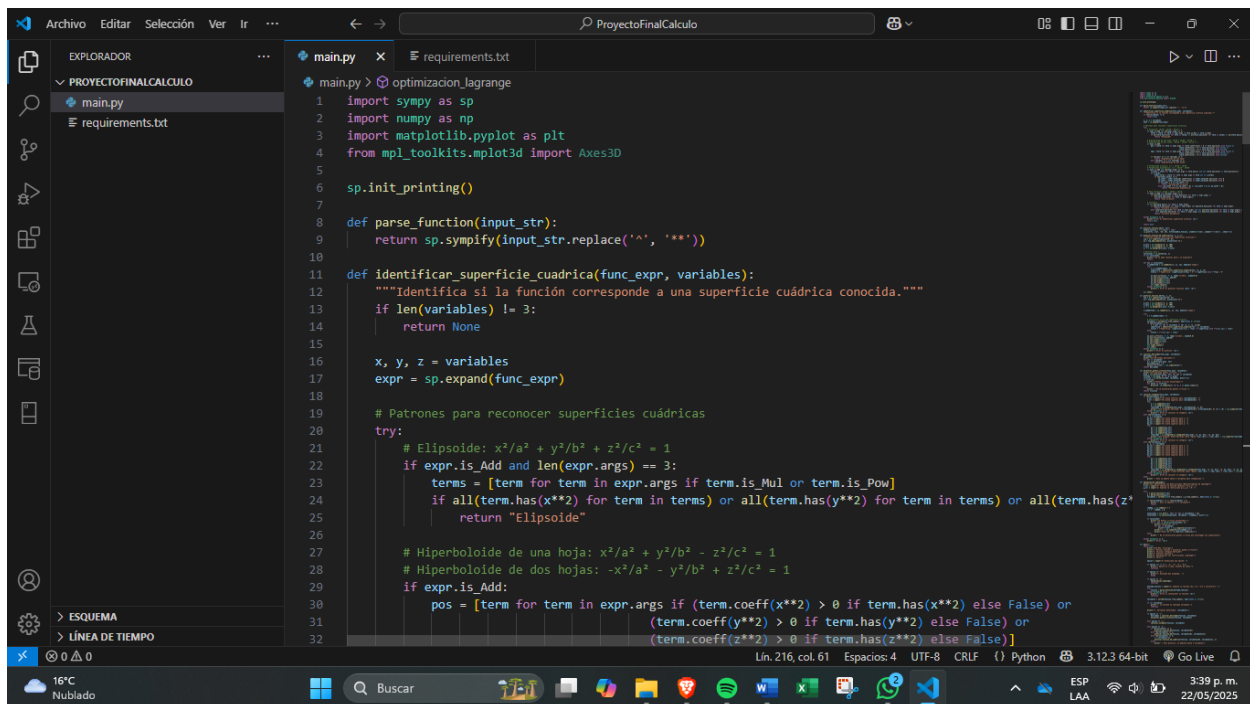
El programa fue desarrollado en un entorno modular, dividiendo las tareas de la siguiente manera:

- **Módulo de entrada de funciones:** Permite al usuario ingresar funciones multivariadas de forma simbólica.
- **Módulo de cálculo simbólico:** Procesa derivadas parciales, gradientes, puntos críticos y optimización con restricciones usando SymPy.

- **Módulo de integración:** Calcula integrales definidas simples, dobles y triples de funciones ingresadas.
- **Módulo de visualización:** Genera gráficas interactivas en 2D y 3D de funciones, regiones y superficies cuadráticas.
- **Interfaz de usuario:** Menú de navegación que permite al usuario seleccionar fácilmente entre las opciones disponibles.

Capturas de pantalla y resultados.

Código:



```

1  import sympy as sp
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from mpl_toolkits.mplot3d import Axes3D
5
6  sp.init_printing()
7
8  def parse_function(input_str):
9      return sp.sympify(input_str.replace('^', '**'))
10
11 def identificar_superficie_cuadratica(func_expr, variables):
12     """Identifica si la función corresponde a una superficie cuadrática conocida."""
13     if len(variables) != 3:
14         return None
15
16     x, y, z = variables
17     expr = sp.expand(func_expr)
18
19     # Patrones para reconocer superficies cuadráticas
20     try:
21         # Elipsoide: x²/a² + y²/b² + z²/c² = 1
22         if expr.is_Add and len(expr.args) == 3:
23             terms = [term for term in expr.args if term.is_Mul or term.is_Pow]
24             if all(term.has(x**2) for term in terms) or all(term.has(y**2) for term in terms) or all(term.has(z**2) for term in terms):
25                 return "Elipsoide"
26
27         # Hiperboloide de una hoja: x²/a² + y²/b² - z²/c² = 1
28         # Hiperboloide de dos hojas: -x²/a² - y²/b² + z²/c² = 1
29         if expr.is_Add:
30             pos = [term for term in expr.args if (term.coeff(x**2) > 0 if term.has(x**2) else False) or
31                   (term.coeff(y**2) > 0 if term.has(y**2) else False) or
32                   (term.coeff(z**2) > 0 if term.has(z**2) else False)]

```

This screenshot shows a VS Code editor window titled 'ProyectoFinalCalculo'. The Explorer sidebar on the left shows a project folder 'PROYECTOFINALCALCULO' containing 'main.py' and 'requirements.txt'. The main editor displays the 'main.py' file with the following code:

```
152
153 def encontrar_puntos_criticos(func_expr, variables):
154     print("\n Buscando puntos criticos...")
155     grad = [sp.diff(func_expr, var) for var in variables]
156     sistema = [sp.Eq(g, 0) for g in grad]
157     criticos = sp.solve(sistema, variables, dict=True)
158     if criticos:
159         print("✅ Puntos criticos encontrados:")
160         for punto in criticos:
161             print((k: sp.simplify(v) for k, v in punto.items()))
162     else:
163         print("⚠ No se encontraron puntos criticos.")
164     return criticos
165
166 def calcular_integral(func_expr, variables):
167     if len(variables) == 1:
168         a_str = input(f"📄 Limite inferior para {variables[0]}: ")
169         b_str = input(f"📄 Limite superior para {variables[0]}: ")
170         try:
171             a = sp.simplify(a_str)
172             b = sp.simplify(b_str)
173             resultado = sp.integrate(func_expr, (variables[0], a, b))
174             print(f"📄 Integral definida: ∫ f({variables[0]}) d(variables[0]) de {a} a {b} = {sp.simplify(resultado)}")
175         except Exception as e:
176             print(f"❌ Error al calcular la integral: {e}")
177     elif len(variables) == 2:
178         x, y = variables
179         ax_str = input("📄 Limite inferior para x: ")
180         bx_str = input("📄 Limite superior para x: ")
181         ay_str = input("📄 Limite inferior para y: ")
182         by_str = input("📄 Limite superior para y: ")
183         try:
184             ax = sp.simplify(ax_str)
185             bx = sp.simplify(bx_str)
186             ay = sp.simplify(ay_str)
187             by = sp.simplify(by_str)
188             resultado = sp.integrate(sp.integrate(func_expr, (x, ax, bx)), (y, ay, by))
189             print(f"📄 Integral doble definida sobre región [{ax},{bx}] x [{ay},{by}] = {sp.simplify(resultado)}")
190         except Exception as e:
191             print(f"❌ Error al calcular la integral: {e}")
192     elif len(variables) == 3:
193         x, y, z = variables
194         ax_str = input("📄 Limite inferior para x: ")
195         bx_str = input("📄 Limite superior para x: ")
196         ay_str = input("📄 Limite inferior para y: ")
197         by_str = input("📄 Limite superior para y: ")
198         az_str = input("📄 Limite inferior para z: ")
199         bz_str = input("📄 Limite superior para z: ")
200         try:
201             ax = sp.simplify(ax_str)
202             bx = sp.simplify(bx_str)
203             ay = sp.simplify(ay_str)
204             by = sp.simplify(by_str)
205             az = sp.simplify(az_str)
206             bz = sp.simplify(bz_str)
207             resultado = sp.integrate(sp.integrate(sp.integrate(func_expr, (x, ax, bx)), (y, ay, by)), (z, az, bz))
208             print(f"📄 Integral triple definida sobre región [{ax},{bx}] x [{ay},{by}] x [{az},{bz}] = {sp.simplify(resultado)}")
209         except Exception as e:
210             print(f"❌ Error al calcular la integral: {e}")
211     else:
212         print("❌ No se puede calcular la integral con más de 3 variables.")
213         return None
214
215 if __name__ == '__main__':
216     func_expr = input("📄 Función a integrar: ")
217     variables = input("📄 Variables: ").split(',')
218     variables = [var.strip() for var in variables]
219     encontrar_puntos_criticos(func_expr, variables)
220     calcular_integral(func_expr, variables)
```

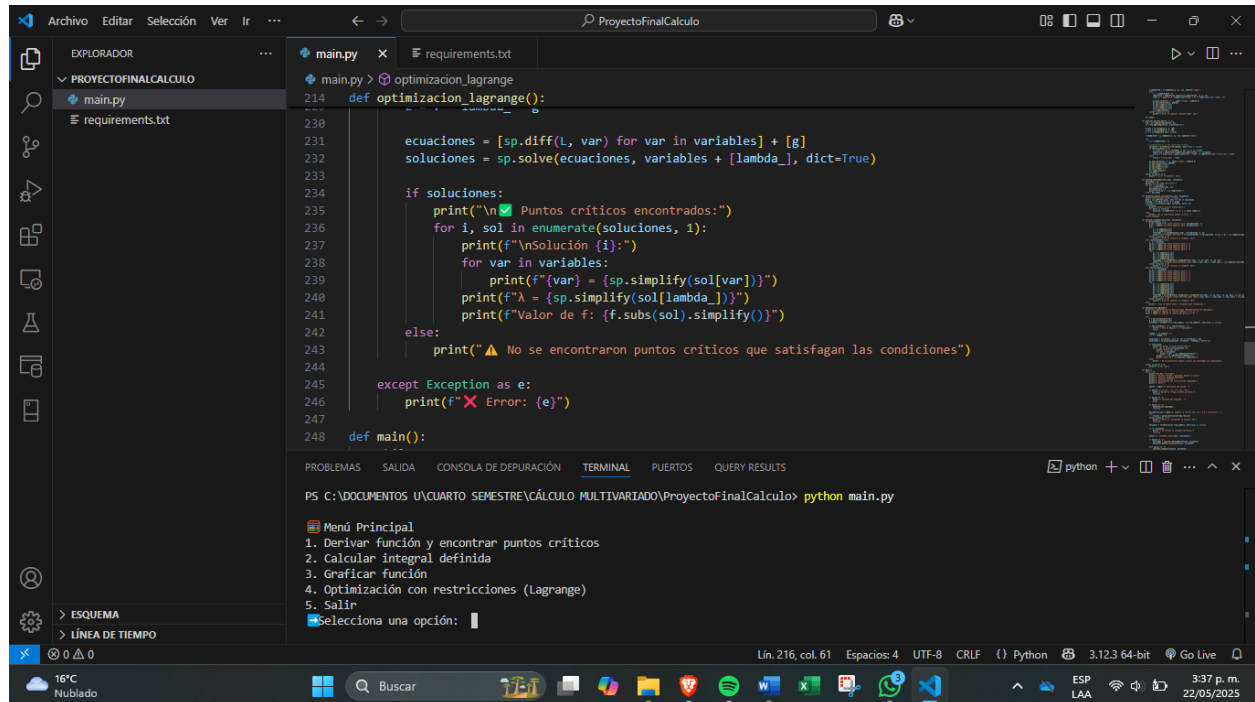
The status bar at the bottom indicates 'Lín. 216, col. 61', 'Espacios: 4', 'UTF-8', 'CRLF', 'Python', '3.12.3 64-bit', and 'Go Live'.

This screenshot shows a VS Code editor window titled 'ProyectoFinalCalculo'. The Explorer sidebar on the left shows a project folder 'PROYECTOFINALCALCULO' containing 'main.py' and 'requirements.txt'. The main editor displays the 'main.py' file with the following code:

```
166 def calcular_integral(func_expr, variables):
167     print(f"📄 Integral definida: ∫ f({variables[0]}) d(variables[0]) de {a} a {b} = {sp.simplify(resultado)}")
168 except Exception as e:
169     print(f"❌ Error al calcular la integral: {e}")
170
171 elif len(variables) == 2:
172     x, y = variables
173     ax_str = input("📄 Limite inferior para x: ")
174     bx_str = input("📄 Limite superior para x: ")
175     ay_str = input("📄 Limite inferior para y: ")
176     by_str = input("📄 Limite superior para y: ")
177     try:
178         ax = sp.simplify(ax_str)
179         bx = sp.simplify(bx_str)
180         ay = sp.simplify(ay_str)
181         by = sp.simplify(by_str)
182         resultado = sp.integrate(sp.integrate(func_expr, (x, ax, bx)), (y, ay, by))
183         print(f"📄 Integral doble definida sobre región [{ax},{bx}] x [{ay},{by}] = {sp.simplify(resultado)}")
184     except Exception as e:
185         print(f"❌ Error al calcular la integral: {e}")
186
187 elif len(variables) == 3:
188     x, y, z = variables
189     ax_str = input("📄 Limite inferior para x: ")
190     bx_str = input("📄 Limite superior para x: ")
191     ay_str = input("📄 Limite inferior para y: ")
192     by_str = input("📄 Limite superior para y: ")
193     az_str = input("📄 Limite inferior para z: ")
194     bz_str = input("📄 Limite superior para z: ")
195     try:
196         ax = sp.simplify(ax_str)
197         bx = sp.simplify(bx_str)
198         ay = sp.simplify(ay_str)
199         by = sp.simplify(by_str)
200         az = sp.simplify(az_str)
201         bz = sp.simplify(bz_str)
202         resultado = sp.integrate(sp.integrate(sp.integrate(func_expr, (x, ax, bx)), (y, ay, by)), (z, az, bz))
203         print(f"📄 Integral triple definida sobre región [{ax},{bx}] x [{ay},{by}] x [{az},{bz}] = {sp.simplify(resultado)}")
204     except Exception as e:
205         print(f"❌ Error al calcular la integral: {e}")
206
207 else:
208     print("❌ No se puede calcular la integral con más de 3 variables.")
209     return None
210
211 if __name__ == '__main__':
212     func_expr = input("📄 Función a integrar: ")
213     variables = input("📄 Variables: ").split(',')
214     variables = [var.strip() for var in variables]
215     encontrar_puntos_criticos(func_expr, variables)
216     calcular_integral(func_expr, variables)
```

The status bar at the bottom indicates 'Lín. 216, col. 61', 'Espacios: 4', 'UTF-8', 'CRLF', 'Python', '3.12.3 64-bit', and 'Go Live'.

Menú consola:



The screenshot shows a VS Code editor with a file named `main.py` open. The script defines a function `optimizacion_lagrange()` that uses SymPy to find critical points of a function. The console output shows the program running and displaying a menu.

```
def optimizacion_lagrange():
    ecuaciones = [sp.diff(L, var) for var in variables] + [g]
    soluciones = sp.solve(ecuaciones, variables + [lambda_], dict=True)

    if soluciones:
        print("\n✅ Puntos críticos encontrados:")
        for i, sol in enumerate(soluciones, 1):
            print(f"\nSolución {i}:")
            for var in variables:
                print(f"    {var} = {sp.simplify(sol[var])}")
            print(f"    λ = {sp.simplify(sol[lambda_])}")
            print(f"    Valor de f: {f.subs(sol).simplify()}")
    else:
        print("\n⚠️ No se encontraron puntos críticos que satisfagan las condiciones")

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

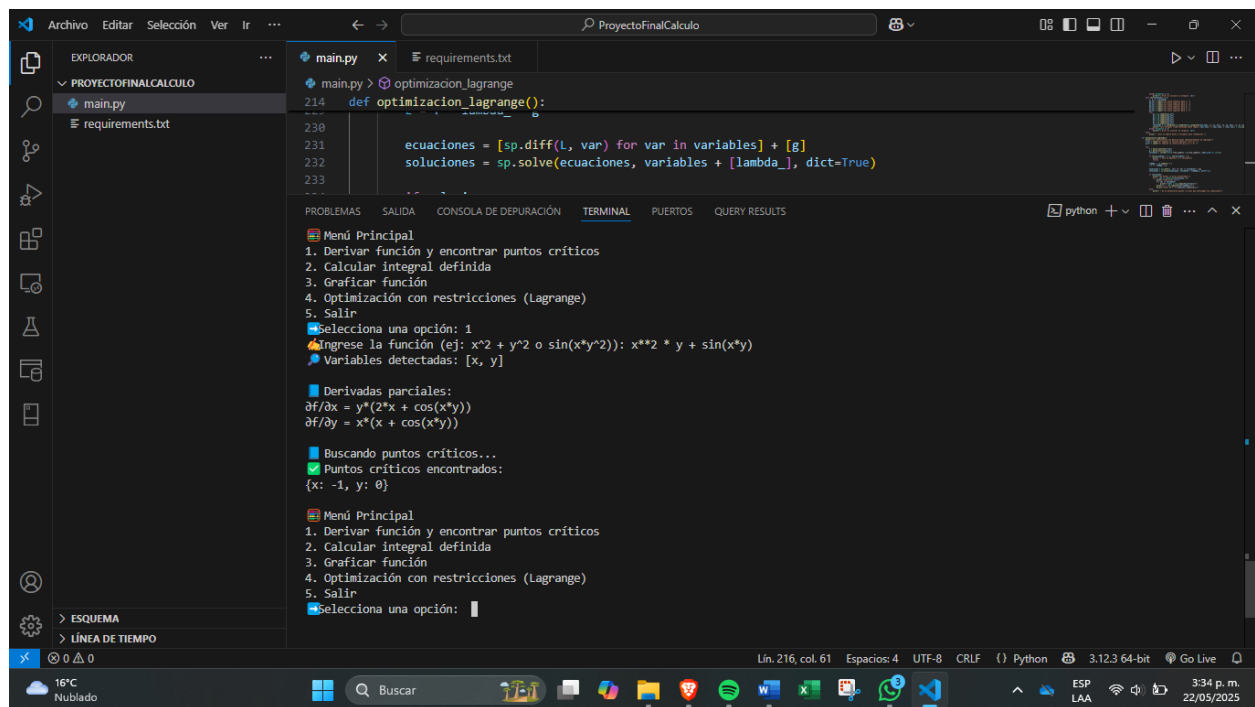
def main():
    # ...
```

Terminal output:

```
PS C:\DOCUMENTOS\UNCUARTO SEMESTRE\CÁLCULO MULTIVARIADO\ProyectoFinalCalculo> python main.py

Menú Principal
1. Derivar función y encontrar puntos críticos
2. Calcular integral definida
3. Graficar función
4. Optimización con restricciones (Lagrange)
5. Salir
Selecciona una opción: 
```

Resultado opción 1 del programa – Derivar función y encontrar puntos críticos:



The screenshot shows the console output for option 1. The user has entered the function $x^2 + y^2 + \sin(xy^2)$. The program has detected the variables x and y , calculated the partial derivatives, and found the critical points at $(-1, 0)$.

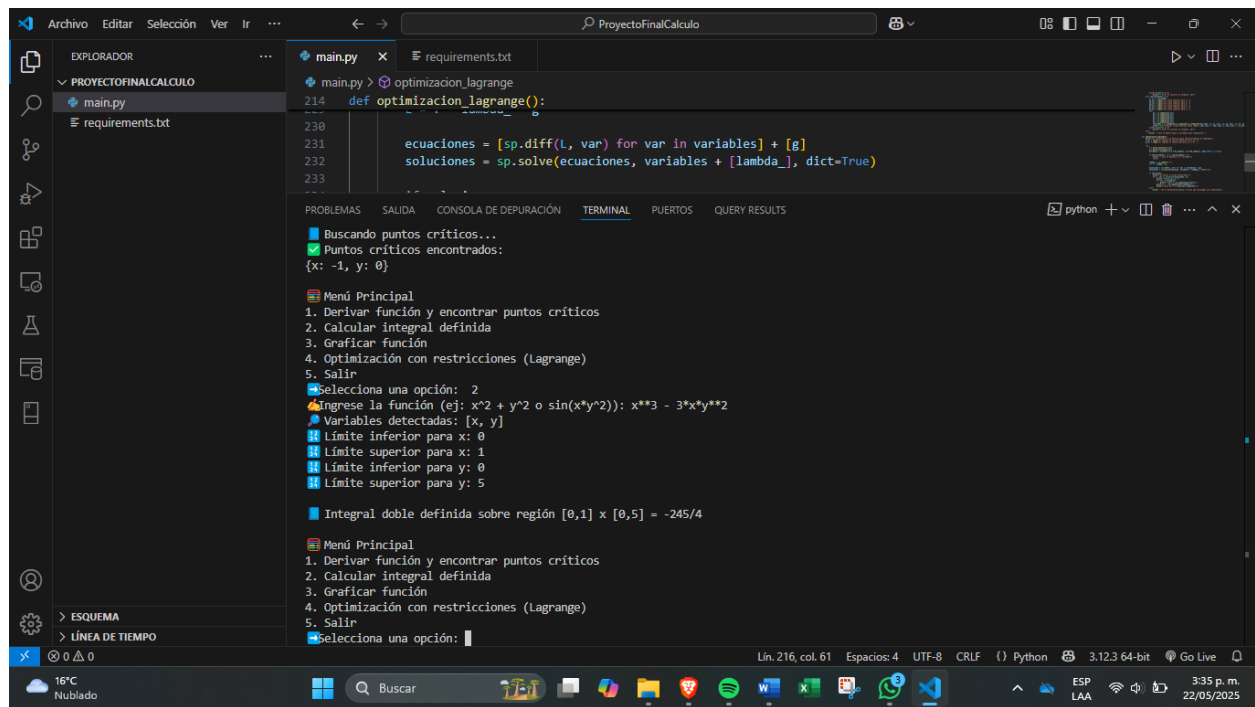
```
Menú Principal
1. Derivar función y encontrar puntos críticos
2. Calcular integral definida
3. Graficar función
4. Optimización con restricciones (Lagrange)
5. Salir
Selecciona una opción: 1
Ingrese la función (ej: x^2 + y^2 o sin(x*y^2)): x**2 * y + sin(x*y)
Variables detectadas: [x, y]

Derivadas parciales:
∂f/∂x = y*(2*x + cos(x*y))
∂f/∂y = x*(x + cos(x*y))

Buscando puntos críticos...
✅ Puntos críticos encontrados:
{x: -1, y: 0}

Menú Principal
1. Derivar función y encontrar puntos críticos
2. Calcular integral definida
3. Graficar función
4. Optimización con restricciones (Lagrange)
5. Salir
Selecciona una opción: 
```

Resultado opción 2 del programa – Calcular integral definida:



The screenshot shows a Python IDE with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The code editor displays a function `optimizacion_lagrange()` that uses SymPy to find critical points and solve a system of equations. The terminal shows the output of the program, including a menu, user input for a function and variables, and the result of a double integral calculation over the region $[0,1] \times [0,5]$.

```
def optimizacion_lagrange():
    ecuaciones = [sp.diff(L, var) for var in variables] + [g]
    soluciones = sp.solve(ecuaciones, variables + [lambda_], dict=True)
```

Buscando puntos críticos...
✓ Puntos críticos encontrados:
{x: -1, y: 0}

Menú Principal
1. Derivar función y encontrar puntos críticos
2. Calcular integral definida
3. Graficar función
4. Optimización con restricciones (Lagrange)
5. Salir

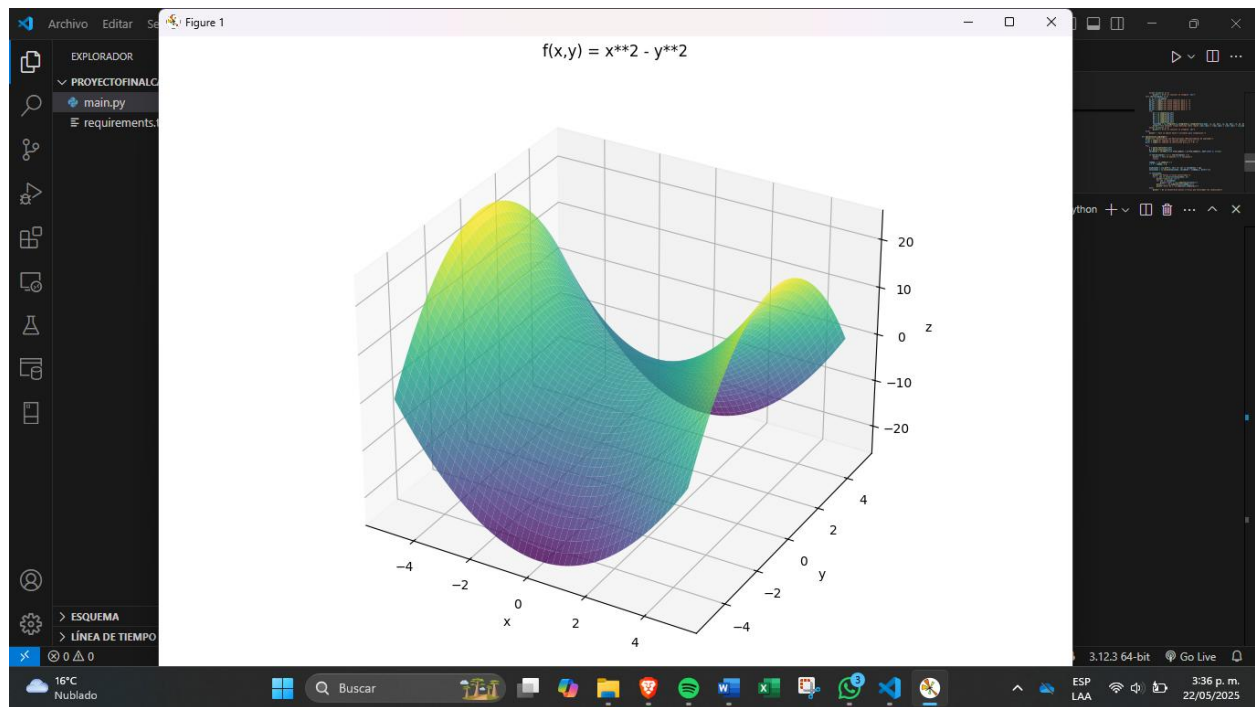
Selecciona una opción: 2
Ingrese la función (ej: x^2 + y^2 o sin(x*y^2)): x**3 - 3*x*y**2
Variables detectadas: [x, y]
Límite inferior para x: 0
Límite superior para x: 1
Límite inferior para y: 0
Límite superior para y: 5

Integral doble definida sobre región $[0,1] \times [0,5] = -245/4$

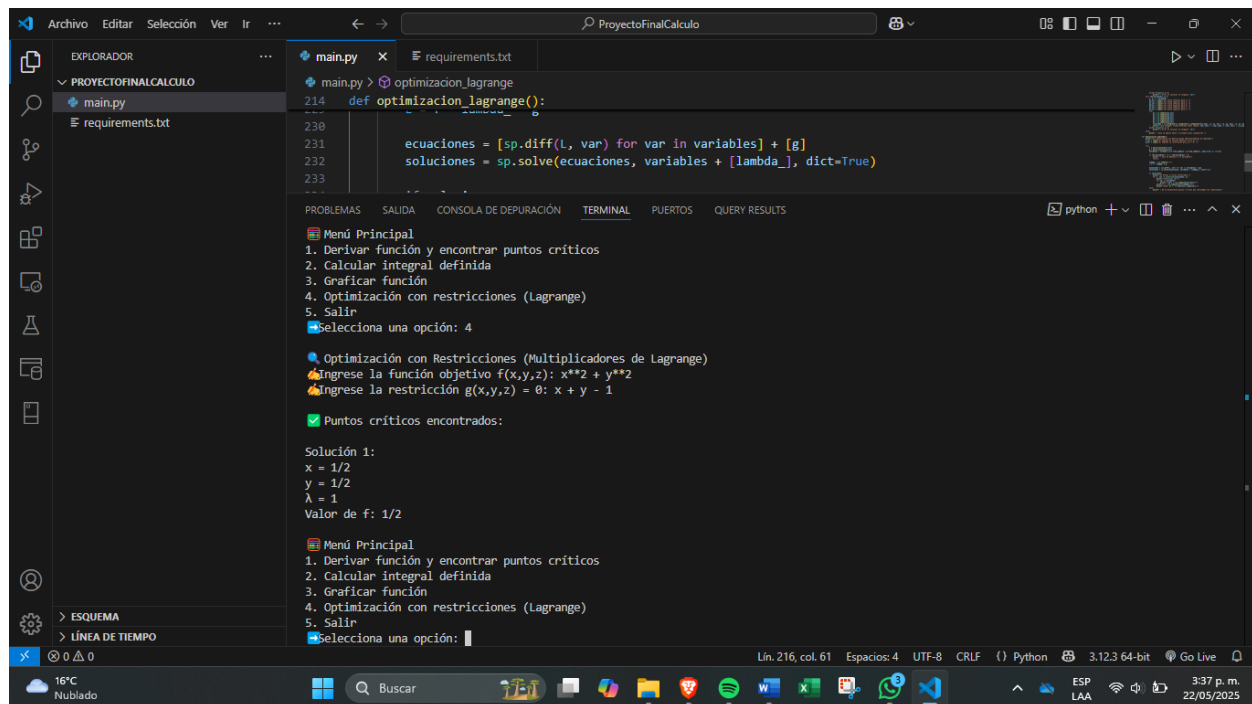
Menú Principal
1. Derivar función y encontrar puntos críticos
2. Calcular integral definida
3. Graficar función
4. Optimización con restricciones (Lagrange)
5. Salir

Selecciona una opción:

Resultado opción 3 del programa – Graficar función:



Resultado opción 4 del programa – Optimización con restricciones (Lagrange):



```
def optimization_lagrange():
    ecuaciones = [sp.diff(L, var) for var in variables] + [g]
    soluciones = sp.solve(ecuaciones, variables + [lambda_], dict=True)
```

Menú Principal
1. Derivar función y encontrar puntos críticos
2. Calcular integral definida
3. Graficar función
4. Optimización con restricciones (Lagrange)
5. Salir
Selecciona una opción: 4

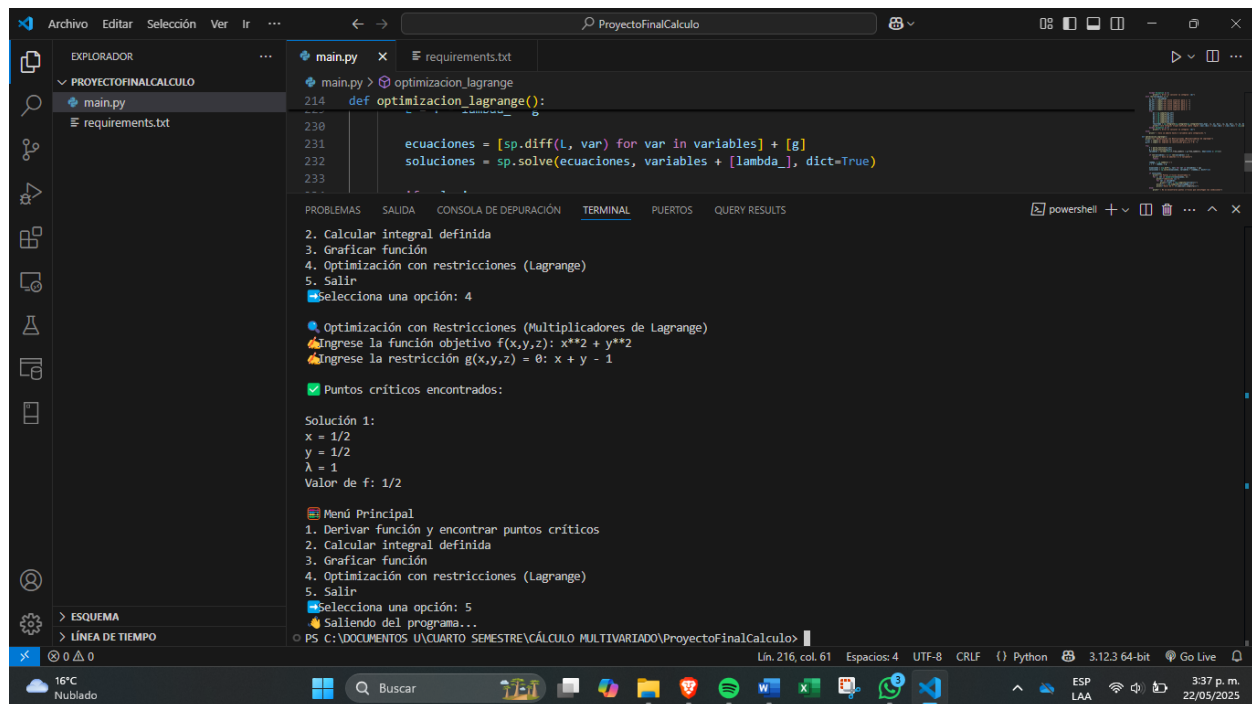
Optimización con Restricciones (Multiplicadores de Lagrange)
Ingrese la función objetivo f(x,y,z): x**2 + y**2
Ingrese la restricción g(x,y,z): x + y - 1

✓ Puntos críticos encontrados:

Solución 1:
x = 1/2
y = 1/2
λ = 1
Valor de f: 1/2

Menú Principal
1. Derivar función y encontrar puntos críticos
2. Calcular integral definida
3. Graficar función
4. Optimización con restricciones (Lagrange)
5. Salir
Selecciona una opción:

Resultado opción 5 del programa – Salir del programa o reiniciar:



```
def optimization_lagrange():
    ecuaciones = [sp.diff(L, var) for var in variables] + [g]
    soluciones = sp.solve(ecuaciones, variables + [lambda_], dict=True)
```

2. Calcular integral definida
3. Graficar función
4. Optimización con restricciones (Lagrange)
5. Salir
Selecciona una opción: 4

Optimización con Restricciones (Multiplicadores de Lagrange)
Ingrese la función objetivo f(x,y,z): x**2 + y**2
Ingrese la restricción g(x,y,z): x + y - 1

✓ Puntos críticos encontrados:

Solución 1:
x = 1/2
y = 1/2
λ = 1
Valor de f: 1/2

Menú Principal
1. Derivar función y encontrar puntos críticos
2. Calcular integral definida
3. Graficar función
4. Optimización con restricciones (Lagrange)
5. Salir
Selecciona una opción: 5
Saliendo del programa...

PS C:\DOCUMENTOS\UVCUARTO SEMESTRE\CÁLCULO MULTIVARIADO\ProyectoFinalCalculo>

Conclusiones y posibles mejoras.

El desarrollo de este aplicativo permitió integrar de manera efectiva el análisis simbólico, numérico y gráfico de funciones multivariables. Se logró cumplir con todos los requisitos propuestos para el proyecto, incluyendo una interfaz funcional, ingreso de funciones simbólicas, visualizaciones en 2D y 3D, y herramientas de cálculo avanzadas como los multiplicadores de Lagrange.

El programa no solo representa una herramienta útil para estudiantes de cálculo multivariado, sino también una plataforma potencial para profesionales que requieran hacer análisis matemático en ingeniería o ciencias aplicadas.

Posibles mejoras futuras:

- Incluir más métodos numéricos de integración para funciones sin solución simbólica.
- Ampliar la interfaz con opciones más detalladas para ingreso de restricciones complejas.
- Agregar exportación de resultados en formato PDF o LaTeX.
- Desarrollar una versión web accesible desde navegadores sin necesidad de instalación.

Bibliografía.

Foundation, P. S. (2024). *python.org*. Obtenido de Python Programming Language – Official Website.: <https://www.python.org/>

Hunter, J. (2024). *matplotlib.org*. Obtenido de Matplotlib: Visualization with Python.: <https://matplotlib.org/>

Oliphant, T. E. (18 de Junio de 2007). *ieeexplore.ieee.org*. Obtenido de Python for Scientific Computing. Computing in Science & Engineering: <https://ieeexplore.ieee.org/document/4160250>

Stewart, J. (2012). *Cálculo Multivariable (7.^a ed.)*. . Cengage Learning.

Team, S. D. (2024). *sympy.org*. Obtenido de SymPy: Python library for symbolic mathematics.: <https://www.sympy.org/en/index.html>