Josue Santana Robledo Corona 325073061

Maestría en Ciencias de la Robótica e Inteligencia Artificial

Algoritmos Bio-Inspirados

Centro Universitario de Ciencias Exactas e Ingeniería.

Mtro. Carlos Alberto López Franco

1. **Introducción**

Este reporte presenta la implementación de un conjunto completo de funciones de prueba para algoritmos de optimización en múltiples dimensiones. Las funciones de prueba son esenciales para evaluar y comparar el desempeño de diferentes algoritmos de optimización, ya que cada función presenta características únicas.

2. **Descripción del código**

El código implementa un sistema interactivo que permite visualizar y evaluar 12 funciones de prueba clásicas de optimización. El sistema soporta tanto visualización 2D como evaluación numérica en n dimensiones, adaptándose automáticamente según la función seleccionada y la dimensionalidad especificada.

3. **Componentes del Código**

Funciones de Prueba Implementadas

```
def esfera_2d(x1, x2):
  return x1 ** 2 + x2 ** 2


def esfera_n(x):
  return np.sum(x ** 2)


def quadrico_nd(x):
  x = np.array(x)
  n = len(x)
  total = 0
  for i in range(1, n + 1):
    suma_parcial = np.sum(x[:i])
    total += suma_parcial ** 2
  return total


def ackley_2d(x1, x2):
  return -20 * np.exp(-0.2 * np.sqrt(0.5 * (x1 ** 2 + x2 ** 2))) - \
    np.exp(0.5 * (np.cos(2 * np.pi * x1) + np.cos(2 * np.pi * x2))) + 20 + np.e


def bohachevsky_2d(x1, x2):
  return x1 ** 2 + 2 * x2 ** 2 - 0.3 * np.cos(3 * np.pi * x1) - 0.4 * np.cos(4 * np.pi * x2) +
0.7
```

```
def colville_2d(x1, x2):
  # Para Colville usamos solo 2 variables para graficar, fijando las otras 2
  x3, x4 = 1, 1  # Valores fijos para graficar
  return 100 * (x2 - x1 ** 2) ** 2 + (1 - x1) ** 2 + 90 * (x4 - x3 ** 2) ** 2 + (1 - x3) ** 2 + \
    10.1 * ((x2 - 1) ** 2 + (x4 - 1) ** 2) + 19.8 * (x2 - 1) * (x4 - 1)


def easom_2d(x1, x2):
  return -np.cos(x1) * np.cos(x2) * np.exp(-(x1 - np.pi) ** 2 - (x2 - np.pi) ** 2)


def griewank_2d(x1, x2):
  return 1 + (x1 ** 2 + x2 ** 2) / 4000 - np.cos(x1 / np.sqrt(1)) * np.cos(x2 / np.sqrt(2))


def hyperellipsoid_2d(x1, x2):
  return 1 ** 2 * x1 ** 2 + 2 ** 2 * x2 ** 2


def rastrigin_2d(x1, x2):
  return (x1 ** 2 - 10 * np.cos(2 * np.pi * x1) + 10) + (x2 ** 2 - 10 * np.cos(2 * np.pi * x2) +
10)


def rosenbrock_2d(x1, x2):
  return 100 * (x2 - x1 ** 2) ** 2 + (1 - x1) ** 2


def schwefel_2d(x1, x2):
  return x1 * np.sin(np.sqrt(np.abs(x1))) + x2 * np.sin(np.sqrt(np.abs(x2))) + 418.9829 *
2
```

Estas son las funciones que vamos a usar

4. Resultados Obtenidos

Vamos a exponer todas las gráficas

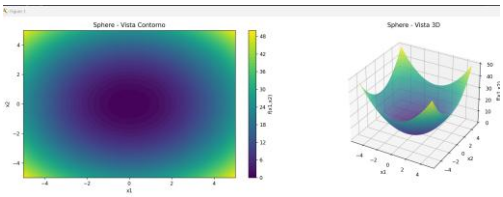

Imagen 1

Sphere

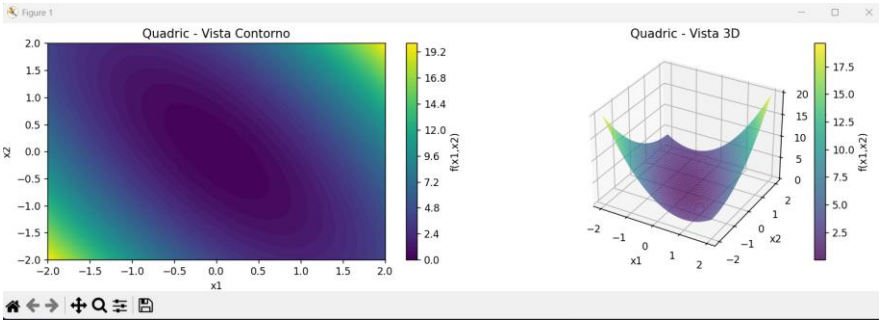

Imagen 2

Quadric

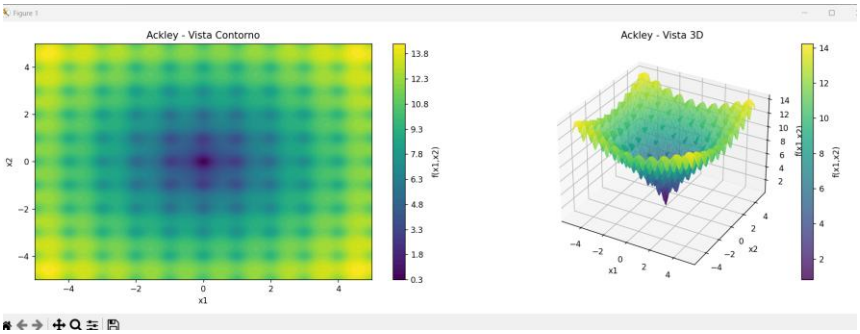

Imagen 3

Ackley

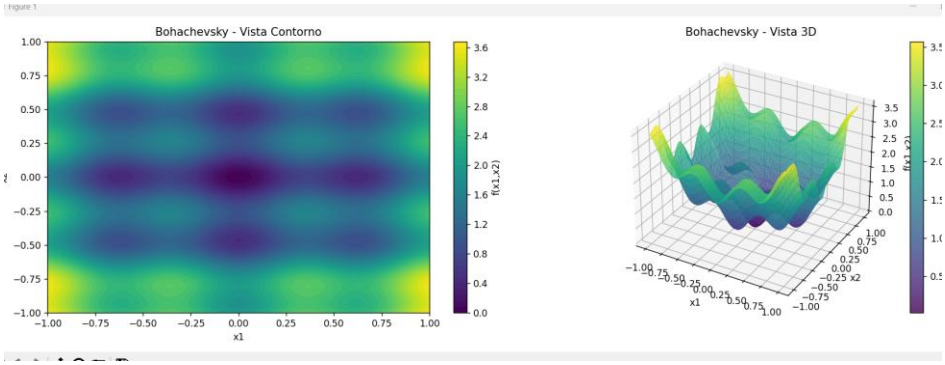

Imagen 4

Bohachevsky

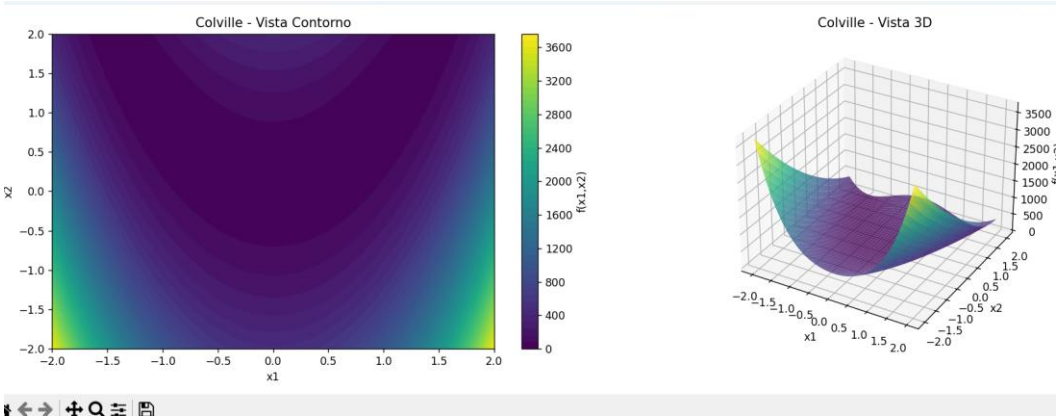

Imagen 5

Colville

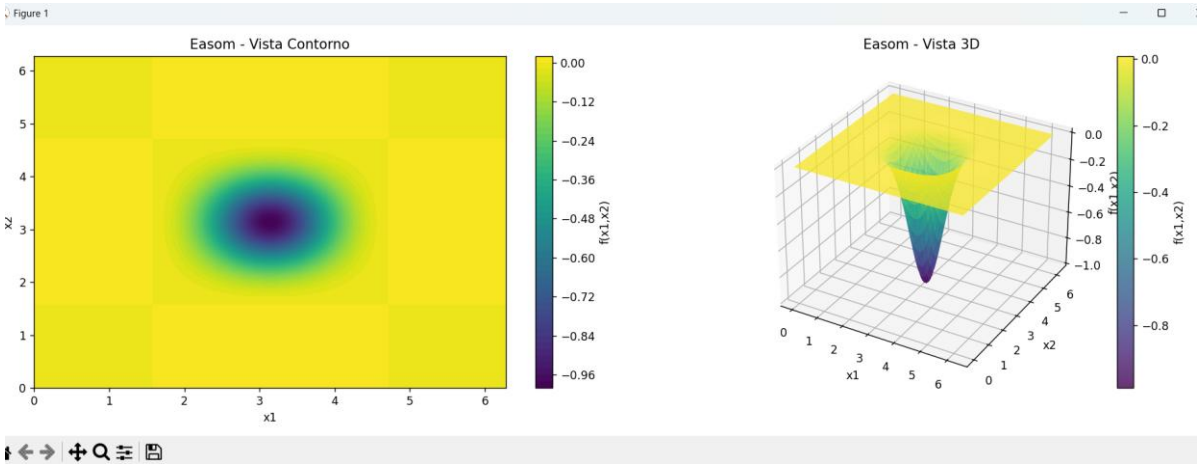

Imagen 6

Easom

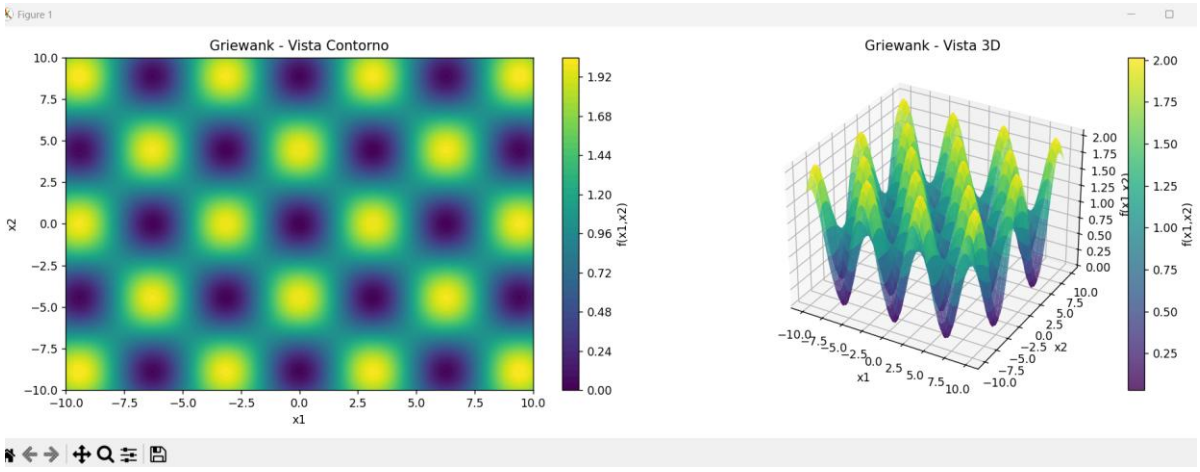

Imagen 7

Griewank

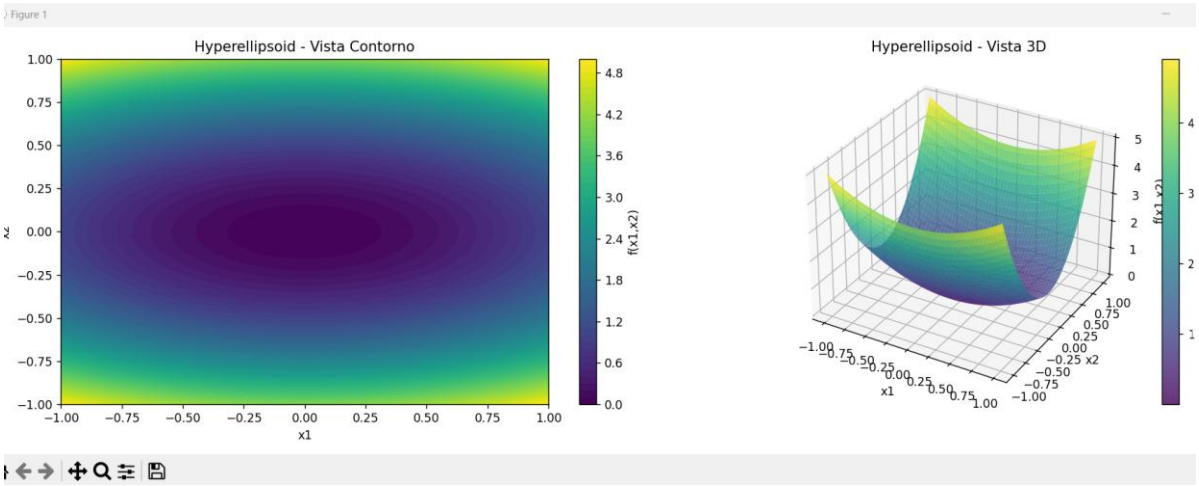

Imagen 8

Hyperellipsoid

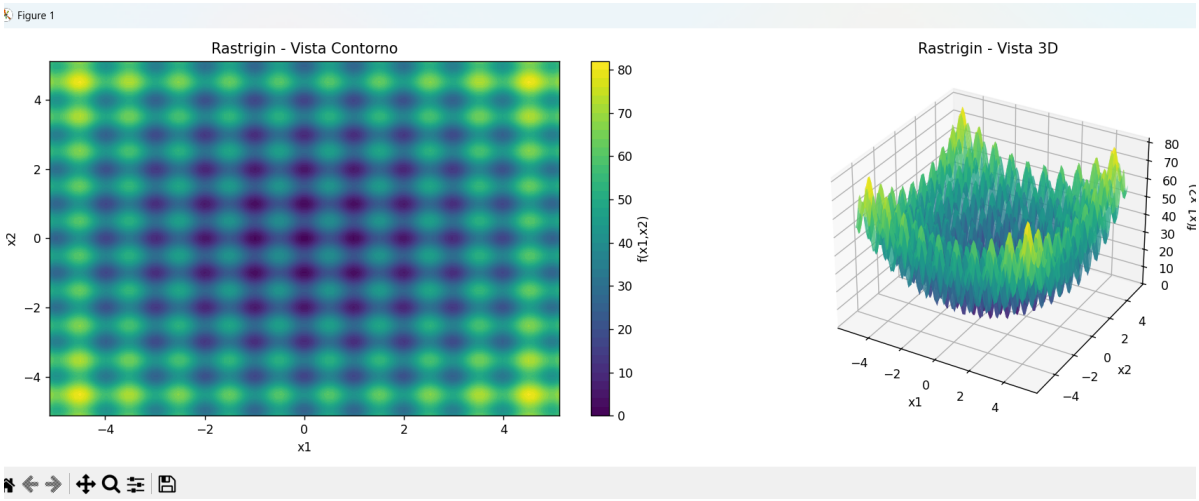

Imagen 9

Rastrigin

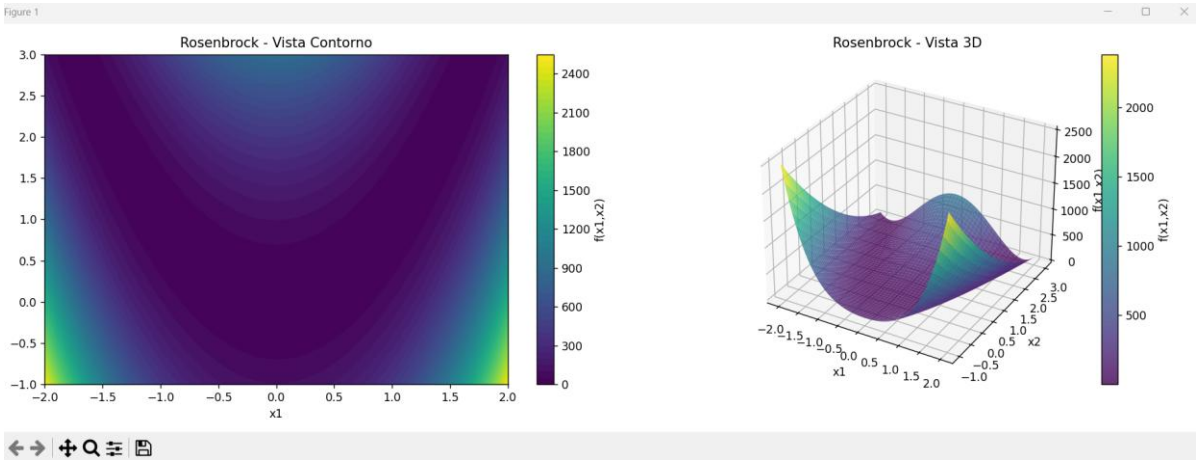

Imagen 10

Rosenbrock

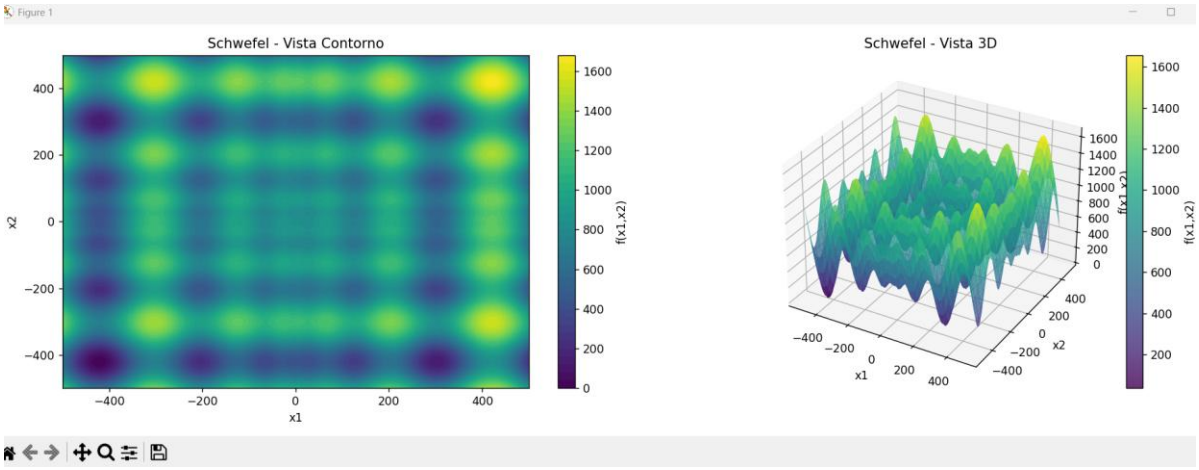

Imagen 11

Schwefel

5. Ventajas del Método

Completitud: Cubre 12 funciones de prueba estándar en la literatura

Flexibilidad: Soporta tanto visualización como evaluación numérica

Interfaz amigable: Menú interactivo fácil de usar

Adaptabilidad: Maneja diferentes dominios y características por función

Escalabilidad: Fácil agregar nuevas funciones al sistema

6. Limitaciones

Visualización limitada: Solo gráficas 2D, no se pueden visualizar funciones en n>2 dimensiones directamente

Adaptaciones necesarias: Algunas funciones como Colville requieren fijar variables para visualización 2D

Evaluación puntual: Para nD solo se muestran evaluaciones en puntos específicos, no superficies completas

Interfaz de consola: Limitada compared to interfaces gráficas modernas

7. Conclusiones

La implementación de este sistema de funciones de prueba proporciona una herramienta completa para el desarrollo y evaluación de algoritmos de optimización.Si costó trabajo, desde entender la función cuando estaba en el pdf, hasta la implementación, la forma que lo hice fue la mejor manera que encontré, así puedo usar una sola función para graficar todas las funciones, y el menú que hice me ayudó más a mi mismo, ya que iba descartando las que ya tenía.

8. Código

```
import numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D



def esfera_2d(x1, x2):

  return x1 ** 2 + x2 ** 2



def esfera_n(x):

  return np.sum(x ** 2)



def quadrico_nd(x):
```

```
    x = np.array(x)
    n = len(x)
    total = 0
    for i in range(1, n + 1):
        suma_parcial = np.sum(x[:i])
        total += suma_parcial ** 2
    return total


def ackley_2d(x1, x2):
    return -20 * np.exp(-0.2 * np.sqrt(0.5 * (x1 ** 2 + x2 ** 2))) - \
        np.exp(0.5 * (np.cos(2 * np.pi * x1) + np.cos(2 * np.pi * x2))) + 20 + np.e


def bohachevsky_2d(x1, x2):
    return x1 ** 2 + 2 * x2 ** 2 - 0.3 * np.cos(3 * np.pi * x1) - 0.4 * np.cos(4 * np.pi * x2) + 0.7


def colville_2d(x1, x2):
    # Para Colville usamos solo 2 variables para graficar, fijando las otras 2
    x3, x4 = 1, 1  # Valores fijos para graficar
    return 100 * (x2 - x1 ** 2) ** 2 + (1 - x1) ** 2 + 90 * (x4 - x3 ** 2) ** 2 + (1 - x3) ** 2 + \
        10.1 * ((x2 - 1) ** 2 + (x4 - 1) ** 2) + 19.8 * (x2 - 1) * (x4 - 1)


def easom_2d(x1, x2):
    return -np.cos(x1) * np.cos(x2) * np.exp(-(x1 - np.pi) ** 2 - (x2 - np.pi) ** 2)
```

```
def griewank_2d(x1, x2):
    return 1 + (x1 ** 2 + x2 ** 2) / 4000 - np.cos(x1 / np.sqrt(1)) * np.cos(x2 / np.sqrt(2))


def hyperellipsoid_2d(x1, x2):
    return 1 ** 2 * x1 ** 2 + 2 ** 2 * x2 ** 2


def rastrigin_2d(x1, x2):
    return (x1 ** 2 - 10 * np.cos(2 * np.pi * x1) + 10) + (x2 ** 2 - 10 * np.cos(2 * np.pi * x2) + 10)


def rosenbrock_2d(x1, x2):
    return 100 * (x2 - x1 ** 2) ** 2 + (1 - x1) ** 2


def schwefel_2d(x1, x2):
    return x1 * np.sin(np.sqrt(np.abs(x1))) + x2 * np.sin(np.sqrt(np.abs(x2))) + 418.9829 * 2


def graficar_funcion_2d(func, title, x_range=(-5, 5), y_range=(-5, 5)):
    """Función auxiliar para graficar cualquier función en 2D"""
    x = np.linspace(x_range[0], x_range[1], 100)
    y = np.linspace(y_range[0], y_range[1], 100)
    X, Y = np.meshgrid(x, y)
    Z = func(X, Y)

    fig = plt.figure(figsize=(15, 5))
```

```
    # Subplot 1: Contorno 2D
    plt.subplot(1, 2, 1)
    contour = plt.contourf(X, Y, Z, levels=50, cmap='viridis')
    plt.colorbar(contour, label='f(x1,x2)')
    plt.title(f'{title} - Vista Contorno')
    plt.xlabel('x1')
    plt.ylabel('x2')

    # Subplot 2: 3D
    plt.subplot(1, 2, 2, projection='3d')
    surf = plt.gca().plot_surface(X, Y, Z, cmap='viridis', alpha=0.8)
    plt.colorbar(surf, label='f(x1,x2)')
    plt.title(f'{title} - Vista 3D')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.gca().set_zlabel('f(x1,x2)')

    plt.tight_layout()
    plt.show()



while True:
    print("""¿Qué función desea graficar?
    1. Esfera 2D
    2. Esfera N-dim (evaluación)
    3. Quadric 2D
    4. Ackley 2D
    5. Bohachevsky 2D
    6. Colville 2D
```

```
7. Easom 2D
8. Griewank 2D
9. Hyperellipsoid 2D
10. Rastrigin 2D
11. Rosenbrock 2D
12. Schwefel 2D
0. Salir
""")
opcion = input("Selecciona una opción: ").strip()

if opcion == "1":
    graficar_funcion_2d(esfera_2d, "Sphere", (-5, 5), (-5, 5))

elif opcion == "2":
    n_dim = int(input("¿Cuántas dimensiones? "))
    punto = np.random.uniform(-10, 10, n_dim)
    resultado = esfera_n(punto)
    print(f"\nSphere {n_dim}D:")
    print(f"Punto evaluado: {punto}")
    print(f"Resultado: {resultado}")
    minimo = esfera_n(np.zeros(n_dim))
    print(f"Mínimo global en {n_dim}D: {minimo}")

elif opcion == "3":
    # Para Quadric necesitamos una función que tome x1, x2
    def quadric_2d(x1, x2):
        puntos = np.column_stack([x1.ravel(), x2.ravel()])
        resultados = np.array([quadrico_nd(p) for p in puntos])
        return resultados.reshape(x1.shape)
```

```
    graficar_funcion_2d(quadric_2d, "Quadric", (-2, 2), (-2, 2))

elif opcion == "4":
    graficar_funcion_2d(ackley_2d, "Ackley", (-5, 5), (-5, 5))

elif opcion == "5":
    graficar_funcion_2d(bohachevsky_2d, "Bohachevsky", (-1, 1), (-1, 1))

elif opcion == "6":
    graficar_funcion_2d(colville_2d, "Colville", (-2, 2), (-2, 2))

elif opcion == "7":
    graficar_funcion_2d(easom_2d, "Easom", (0, 2 * np.pi), (0, 2 * np.pi))

elif opcion == "8":
    graficar_funcion_2d(griewank_2d, "Griewank", (-10, 10), (-10, 10))

elif opcion == "9":
    graficar_funcion_2d(hyperellipsoid_2d, "Hyperellipsoid", (-1, 1), (-1, 1))

elif opcion == "10":
    graficar_funcion_2d(rastrigin_2d, "Rastrigin", (-5.12, 5.12), (-5.12, 5.12))

elif opcion == "11":
    graficar_funcion_2d(rosenbrock_2d, "Rosenbrock", (-2, 2), (-1, 3))

elif opcion == "12":
```

```
    graficar_funcion_2d(schwefel_2d, "Schwefel", (-500, 500), (-500, 500))

elif opcion == "0":
    print("Saliendo del programa. ")
    break

else:
    print("Opción no válida. Intenta otra vez.")

respuesta = input("\n¿Deseas probar otra función? (sí/no): ").strip().lower()
if respuesta in ['no', 'n', '2']:
    print("Saliendo del programa. ¡Hasta luego!")
    break
```