UNIVERSIDAD DE GUADALAJARA

Centro Universitario de Ciencias Exactas e Ingenierías

Ingeniería en Computación

7mo semestre

Tema: Funciones de prueba para optimización multimodal, multiobjetivo y multidimensional

Materia
Seminario de Inteligencia Artificial I

Sección
D01

Código:
217565958

Carrera
Ingeniería en Computación.

PRESENTA
Daniel Martínez Martínez

Docente
Alma Yolanda Alanis García

Fecha de entrega:
Jueves, 31 de Agosto de 2023

Funciones de prueba para optimización multimodal, multiobjetivo y multidimensional

Instrucciones

Investiga y grafica en tres dimensiones las funciones de prueba ("benchmark") planteadas en la sección de trabajos de Google Classroom.

A. *Esfera (Unimodal)*

La función para este modelo está dada como:

$$f(x) = \sum_{i=1}^{n} x^2$$

Para 3D:

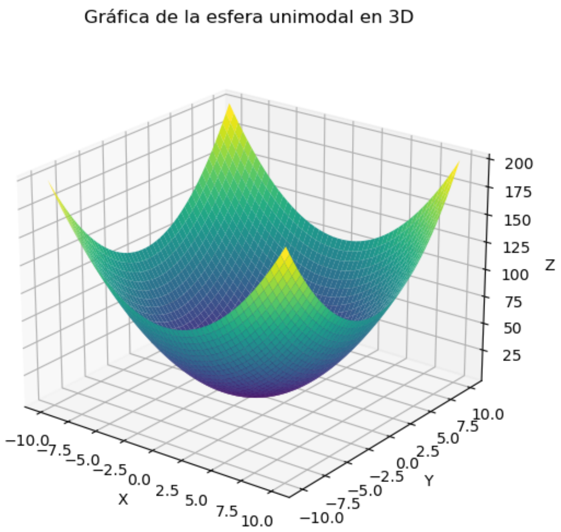
$$f(x, y) = \sum_{i=1}^{n} x_i^2 + y_i^2$$

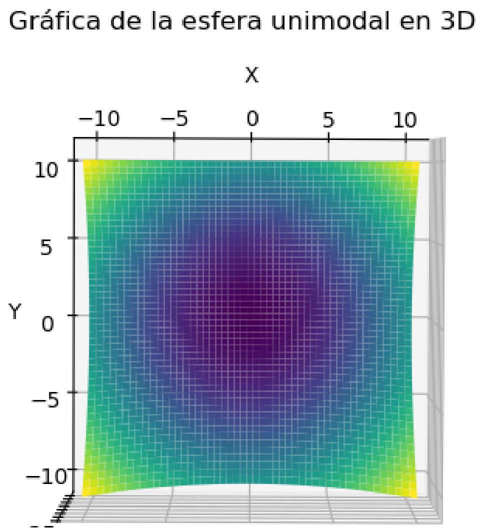
Para esta prueba se emplearon dos arreglos de datos (x,y) con 50 valores equidistantes dado el rango [-10,10]. A continuación, se muestran los resultados...

Código

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Importa la herramienta 3D de Matplotlib

# Crear datos de ejemplo: genera arreglos de valores X e Y, y crea una malla (grid) X-Y
x = np.linspace(-10, 10, 50)  # Crea 50 valores equidistantes en el rango -10 a 10
y = np.linspace(-10, 10, 50)
x, y = np.meshgrid(x, y)  # Crea una malla de valores X e Y para usar en la gráfica

z = x**2 + y**2  # Coordenada z en función de theta

# Crea una figura en 3D
fig = plt.figure()  # Crea una figura para la gráfica
ax = fig.add_subplot(111, projection='3d')  # Agrega un subplot 3D a la figura

# Crear la gráfica de la esfera
surface = ax.plot_surface(x, y, z, cmap='viridis')  # Crea la gráfica de superficie con colores viridis

# Agregar etiquetas y título
ax.set_xlabel('X')  # Agrega etiqueta al eje X
ax.set_ylabel('Y')  # Agrega etiqueta al eje Y
ax.set_zlabel('Z')  # Agrega etiqueta al eje Z
ax.set_title('Gráfica de la esfera unimodal en 3D')  # Agrega un título a la gráfica

# Mostrar la gráfica
plt.show()  # Muestra la gráfica en una ventana emergente
```

Capturas





B. *Step Function*

La función para este modelo está dada como:

$$f(x) = \sum_{i=1}^{n} (floor(x_i + 0.5))^2$$

Para 3D:

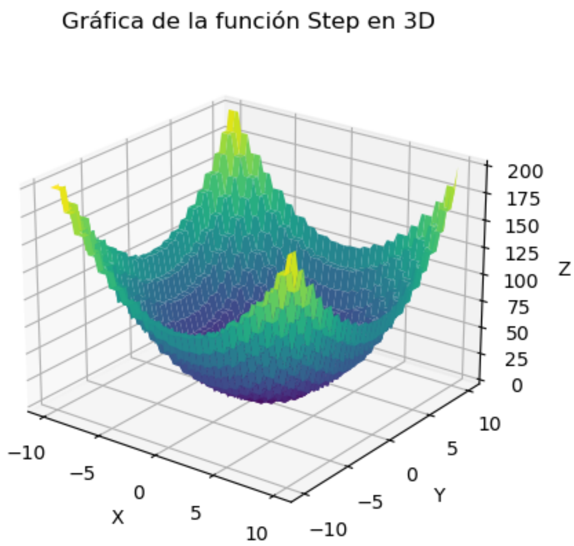
$$f(x, y) = \sum_{i=1}^{n} (floor(x_i))^2 + (floor(y_i))^2$$

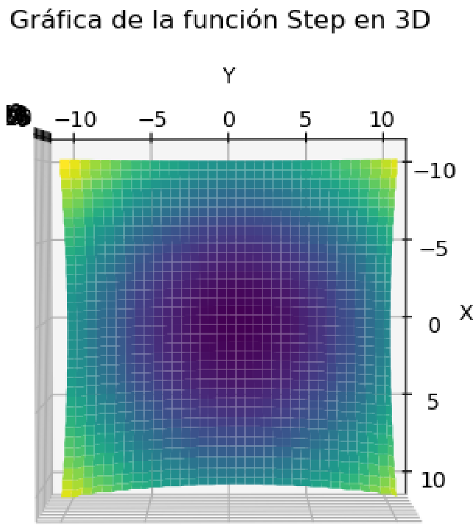
Para esta simulación se emplearon dos arreglos de datos (x,y) con 70 valores equidistantes, abarcando el rango [-10,10]. A continuación, se muestran los resultados...

Código

```
import numpy as np
import matplotlib.pyplot as plt

# Crear datos de ejemplo: genera arreglos de valores X e Y, y crea una malla (grid) X-Y
x = np.linspace(-10, 10, 50)  # Crea 50 valores equidistantes en el rango -10 a 10
y = np.linspace(-10, 10, 50)
x, y = np.meshgrid(x, y)  # Crea una malla de valores X e Y para usar en la gráfica

z = np.floor(x)**2+np.floor(y)**2

fig = plt.figure()  # Crea una figura para la gráfica
ax = fig.add_subplot(111, projection='3d')  # Agrega un subplot 3D a la figura
surface = ax.plot_surface(x, y, z, cmap='viridis')  # Crea la gráfica de superficie con colores viridis

ax.set_xlabel('X')  # Agrega etiqueta al eje X
ax.set_ylabel('Y')  # Agrega etiqueta al eje Y
ax.set_zlabel('Z')  # Agrega etiqueta al eje Z
ax.set_title('Gráfica de la función Step en 3D') # Agrega un título a la gráfica
plt.show()  # Muestra la gráfica en una ventana emergente
```

Capturas





C. *Absolute Function*

La función para este modelo está dada como:

$$f(x) = \sum_{i=1}^{n} |x_i|$$

Para 3D:

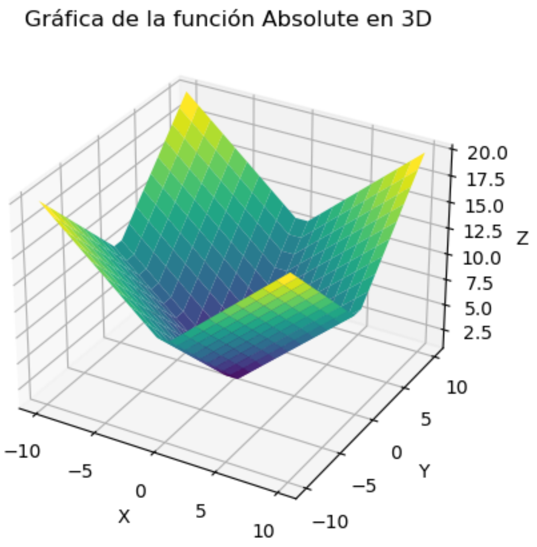
$$f(x, y) = \sum_{i=1}^{n} |x_i| + |y_i|$$

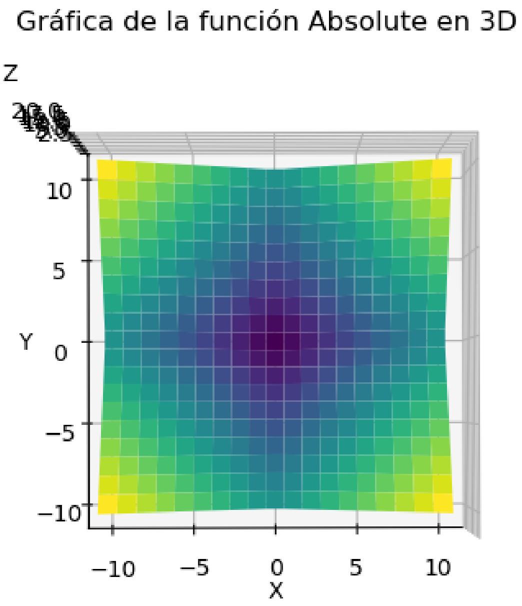
Para esta simulación se emplearon dos arreglos de datos (x,y) con 20 valores equidistantes, abarcando el rango [-10,10]. A continuación se muestran los resultados…

Código

```
import numpy as np  # Importa la biblioteca NumPy para manipulación numérica
import matplotlib.pyplot as plt  # Importa la biblioteca Matplotlib para visualización

# Crear datos de ejemplo: genera arreglos de valores X e Y, y crea una malla (grid) X-Y
x = np.linspace(-10, 10, 20)  # Crea 20 valores equidistantes en el rango -10 a 10
y = np.linspace(-10, 10, 20)
x, y = np.meshgrid(x, y)  # Crea una malla de valores X e Y para usar en la gráfica
z = np.abs(x)+np.abs(y)

fig = plt.figure()  # Crea una figura para la gráfica
ax = fig.add_subplot(111, projection='3d')  # Agrega un subplot 3D a la figura
surface = ax.plot_surface(x, y, z, cmap='viridis')  # Crea la gráfica de superficie con colores viridis

ax.set_xlabel('X')  # Agrega etiqueta al eje X
ax.set_ylabel('Y')  # Agrega etiqueta al eje Y
ax.set_zlabel('Z')  # Agrega etiqueta al eje Z
ax.set_title('Gráfica de la función Absolute en 3D')  # Agrega un título a la gráfica
plt.show()  # Muestra la gráfica en una ventana emergente
```

Capturas





D. *Michalewicz Function*

La función para este modelo está dada como:

$$f(x) = -\sum_{i=1}^{n} sin(x_i)[sin(\frac{ix_i^2}{\pi})]^{2m}$$

Para 3D:

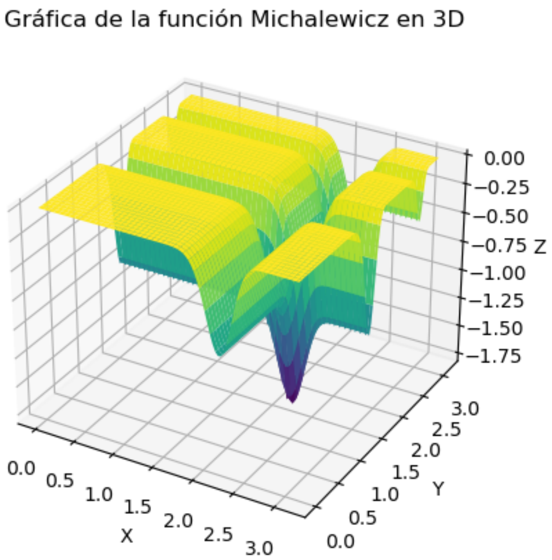
$$f(x, y) = -\{\sum_{i=1}^{n} sin(x_i)[sin(\frac{x_i^2}{\pi})]^{2m} + sin(y_i)[sin(\frac{2y_i^2}{\pi})]^{2m}\}$$

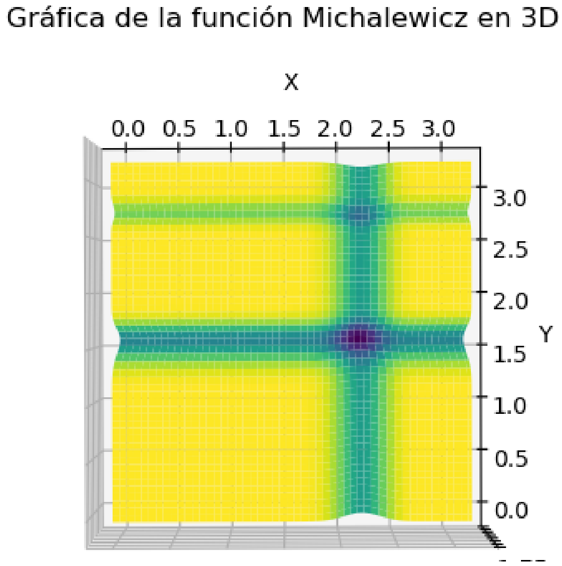
Para esta simulación se emplearon dos arreglos de datos (x,y) con 100 valores equidistantes, abarcando el rango [0,π]. A continuación se muestran los resultados...

Código

```
import numpy as np  # Importa la biblioteca NumPy para manipulación numérica
import matplotlib.pyplot as plt  # Importa la biblioteca Matplotlib para visualización

x = np.linspace(0, np.pi, 100)  # Crea 100 valores equidistantes en el rango 0 a pi
y = np.linspace(0, np.pi, 100)
x, y = np.meshgrid(x, y)  # Crea una malla de valores X e Y para usar en la gráfica
M=10# Parámetro de ajuste para la función de Michalewicz
z = -((np.sin(x) * np.sin((1 * x**2) / np.pi)**(2*M)) + (np.sin(y) * np.sin((2 * y**2) / np.pi)**(2*M)))
fig = plt.figure()  # Crea una figura para la gráfica
ax = fig.add_subplot(111, projection='3d')  # Agrega un subplot 3D a la figura
surface = ax.plot_surface(x, y, z, cmap='viridis')  # Crea la gráfica de superficie con colores viridis
ax.set_xlabel('X')  # Agrega etiqueta al eje X
ax.set_ylabel('Y')  # Agrega etiqueta al eje Y
ax.set_zlabel('Z')  # Agrega etiqueta al eje Z
ax.set_title('Gráfica de la función Michalewicz en 3D')  # Agrega un título a la gráfica
plt.show()  # Muestra la gráfica en una ventana emergente
```

Capturas





E. *EggHolder Function*

La función para este modelo 3D está dada como:

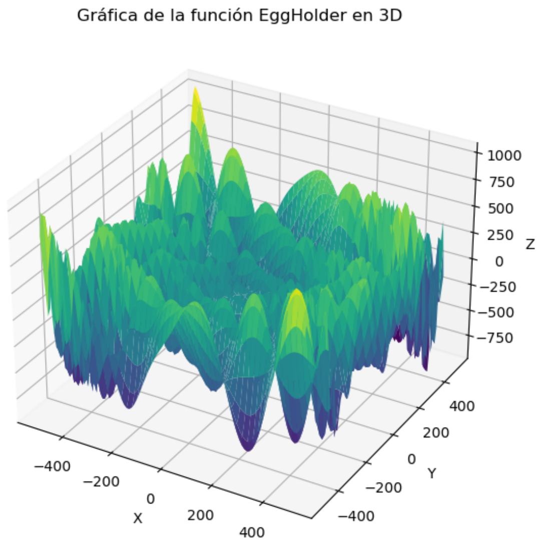
$$f(x, y) = -\left\{\sum_{i=1}^{n} (y + 47) * sin\left(\sqrt{\left|\frac{x}{2} + (y + 47)\right|}\right) + x * sin\left(\sqrt{|x - (y + 47)|}\right)\right\}$$

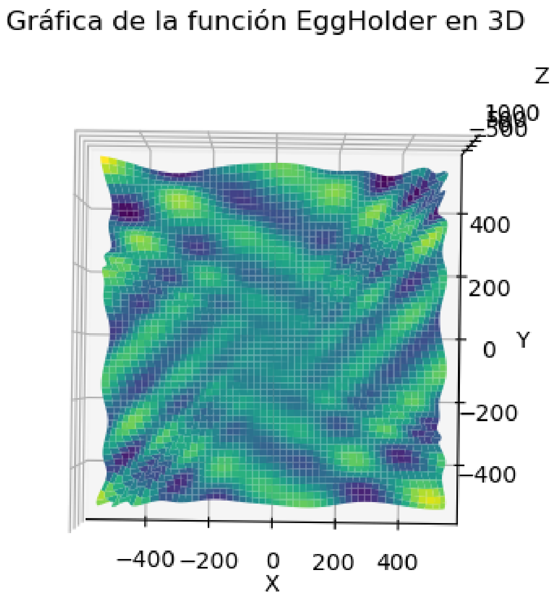
Para esta simulación se emplearon dos arreglos de datos (x,y) con 100 valores equidistantes, abarcando el rango [-512,512]. A continuación se muestran los resultados...

Código

```python
import numpy as np  # Importa la biblioteca NumPy para manipulación numérica
import matplotlib.pyplot as plt  # Importa la biblioteca Matplotlib para visualización

x = np.linspace(-512, 512, 100)  # Crea 100 valores equidistantes en el rango -5 a 5
y = np.linspace(-512, 512, 100)
x, y = np.meshgrid(x, y)  # Crea una malla de valores X e Y para usar en la gráfica

z = -(y + 47) * np.sin(np.sqrt(np.abs(y + x/2 + 47))) - x * np.sin(np.sqrt(np.abs(x - (y + 47))))

fig = plt.figure()  # Crea una figura para la gráfica
ax = fig.add_subplot(111, projection='3d')  # Agrega un subplot 3D a la figura

surface = ax.plot_surface(x, y, z, cmap='viridis')  # Crea la gráfica de superficie con colores viridis

ax.set_xlabel('X')  # Agrega etiqueta al eje X
ax.set_ylabel('Y')  # Agrega etiqueta al eje Y
ax.set_zlabel('Z')  # Agrega etiqueta al eje Z
ax.set_title('Gráfica de la función EggHolder en 3D')  # Agrega un título a la gráfica

plt.show()  # Muestra la gráfica en una ventana emergente
```

Capturas





F. *Weierstrass Function*

La función para este modelo está dada como:

$$f_{\text{Weierstrass}}(\mathbf{x}) = \sum_{i=1}^{n} \left[\sum_{k=0}^{kmax} a^k \cos\left(2\pi b^k (x_i + 0.5)\right) - n \sum_{k=0}^{kmax} a^k \cos(\pi b^k) \right]$$

Para 3D:

$$f(x, y) = \sum_{i=1}^{n} a^k cos(2\pi b^k (x_i + 0.5)) - a^k cos(\pi b^k) + a^k cos(2\pi b^k (y_i + 0.5)) - a^k cos(\pi b^k)$$

Para esta simulación se emplearon dos arreglos de datos (x,y) con 150 valores equidistantes, abarcando el rango [-0.5,0.5], además de los valores $a = 0.5$, $b = 3$, $kmax = 20$. A continuación se muestran los resultados...

Código

```
import numpy as np
import matplotlib.pyplot as plt
def wierstrass(x, y, a, b, k_max):
    result = 0
    for k in range(k_max):
        result += (a**k * np.cos(2 * np.pi * b**k * (x + 0.5)) - a**k*np.cos(np.pi*b**k)) + (a**k * np.cos(2 * np.pi * b**k *
(y + 0.5)) - a**k*np.cos(np.pi*b**k))
    return result
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(-0.5, 0.5, 150)
```

```
y = np.linspace(-0.5, 0.5, 150)
x, y = np.meshgrid(x, y)
z = wierstrass(x, y, a=0.5, b=3, k_max=20)
ax.plot_surface(x, y, z, cmap='viridis')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Función de Weierstrass en 3D')
plt.show()
```

Capturas