INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO
INTELIGENCIA ARTIFICIAL
ING. EN SISTEMAS COMPUTACIONALES

Laboratorio 5: Búsqueda Informada P.II

Alumno: Hurtado Morales Emiliano - 2021630390

Maestro: Andrés García Floriano

Grupo: 6CV3

Fecha de entrega: 10/10/2024

Ciclo Escolar: 2025 – 1

Introducción

La optimización es una rama crucial en matemáticas y ciencias de la computación, utilizada en una amplia gama de campos, desde la inteligencia artificial hasta la ingeniería. El objetivo de la optimización es encontrar los mejores parámetros (mínimos o máximos) de una función bajo ciertas restricciones o sin ellas. Las funciones multimodales, que poseen varios mínimos o máximos locales, presentan un desafío adicional, ya que no todos los métodos de optimización garantizan encontrar el mejor valor global.

Una de estas funciones multimodales es la **función de Himmelblau**, que se define como:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

Esta función tiene **cuatro mínimos locales** distribuidos de manera irregular, lo que la convierte en un caso de estudio ideal para probar y comparar algoritmos de optimización. Los puntos donde se alcanzan estos mínimos son:

$$(3.0, 2.0), (-2.805, 3.131), (-3.779, -3.283), (3.584, -1.848)$$

Cada uno de estos puntos es un mínimo local, y el valor de la función en estos puntos es igual a cero, es decir, $f(x, y) = 0$.

En esta práctica, se propuso encontrar los mínimos de la función de Himmelblau utilizando tres enfoques diferentes:

1. Usar una librería de optimización (**scipy.optimize**).
2. Realizar una búsqueda exhaustiva utilizando fuerza bruta.
3. Implementar el algoritmo de descenso por gradiente manualmente.

Explicación de los Algoritmos

https://github.com/EmilianoHM/Bender-IA/tree/main/Lab_5

Método con librería (Scipy)

En este método, se utilizó la función minimize de la librería **scipy.optimize**, que es una función general para la minimización de funciones multivariables. minimize usa una variedad de algoritmos numéricos, dependiendo del método elegido. En este caso, se usa un algoritmo basado en el **Método de Nelder-Mead** o **BFGS**, dependiendo de la configuración interna.

El **Método de Nelder-Mead** es un algoritmo de optimización que no requiere derivadas. Es adecuado para minimizar funciones no diferenciables o ruidosas y trabaja mediante la manipulación de un **simplex**, una figura geométrica formada por $n + 1$ puntos en un espacio de n dimensiones (por ejemplo, un triángulo en 2D). El algoritmo ajusta iterativamente el simplex a través de las siguientes operaciones:

1. **Reflexión**: El peor vértice se refleja en el lado opuesto al mejor vértice.
2. **Expansión**: Si la reflexión mejora significativamente, se expande el simplex en esa dirección.
3. **Contracción**: Si la reflexión no mejora mucho, se reduce el simplex.
4. **Reducción**: Si toda falla, se encoge el simplex alrededor del mejor vértice.

El proceso continúa hasta que el simplex se ajusta lo suficiente cerca del mínimo local. Aunque es simple y eficaz para problemas en espacios de baja dimensión, **Nelder-Mead** puede quedarse atrapado en mínimos locales y es menos eficiente en dimensiones altas.

Por otro lado, el **BFGS (Broyden-Fletcher-Goldfarb-Shanno)** es un algoritmo de optimización **cuasi-Newtoniano** utilizado para minimizar funciones diferenciables. A diferencia de métodos como el **descenso por gradiente**, BFGS no solo utiliza el gradiente de la función, sino que también actualiza una aproximación a la **matriz Hessiana inversa** (que contiene segundas derivadas), lo que le permite encontrar el mínimo más rápidamente.

1. **Gradiente**: BFGS usa el gradiente de la función para determinar la dirección de búsqueda.
2. **Aproximación de la Hessiana**: En lugar de calcular directamente la matriz Hessiana (computacionalmente costosa), BFGS construye una aproximación eficiente de la inversa de esta matriz, utilizando las diferencias de gradiente entre iteraciones.
3. **Actualización**: Después de cada paso, se actualiza la aproximación de la Hessiana, lo que mejora la dirección y el tamaño del siguiente paso.
4. **Iteración**: El proceso continúa hasta que el cambio en los valores de la función o el gradiente sea lo suficientemente pequeño.

El código que ejecuta este método es el siguiente:

```
# Método 1: Usar una librería (scipy)
def metodo_libreria():
    from scipy.optimize import minimize
    # Definir la función para optimizar
    def himmelblau_opt(xy):
        x, y = xy
        return funcion_himmelblau(x, y)

    # Establecer el valor inicial y los límites
    valor_inicial = np.array([0, 0])
    limites = [(-5, 5), (-5, 5)]

    # Ejecutar el algoritmo de minimización
    resultado = minimize(himmelblau_opt, valor_inicial, bounds=limites)

    # Mostrar resultados
    print(f"\nMétodo con librería:")
    print(f"El mínimo está en x = {resultado.x[0]}, y = {resultado.x[1]}, con un
valor de {resultado.fun}\n")
```

- **Valor inicial**: Se parte de un punto inicial arbitrario en *(0, 0)*, que será el punto de partida para la búsqueda del mínimo.
- **Límites**: Se definen los límites *[−5, 5]* para ambas variables *x* e *y*, para asegurar que la búsqueda se mantenga dentro de la región donde se conocen los mínimos de la función de Himmelblau.
- **Algoritmo**: **minimize** ajusta los parámetros de *x* e *y* iterativamente hasta encontrar un mínimo local. El resultado dependerá de la elección del punto inicial, pero en este caso la función converge rápidamente a uno de los mínimos locales conocidos.

Método de fuerza bruta

El método de fuerza bruta consiste en evaluar exhaustivamente la función en un gran número de puntos en un dominio definido. Este método garantiza encontrar el mínimo global, pero es computacionalmente costoso, especialmente para funciones complejas y de alta dimensionalidad. A pesar de su ineficiencia, es una técnica simple y efectiva cuando se trata de funciones bien definidas en un rango acotado, como en este caso.

El código de la búsqueda por fuerza bruta es el siguiente:

```
# Método 2: Fuerza Bruta
def metodo_fuerza_bruta():
    # Definir los rangos de búsqueda y los pasos
    rango_x = np.linspace(-5, 5, 1000)
    rango_y = np.linspace(-5, 5, 1000)

    # Variables para almacenar el valor mínimo
    valor_minimo = float('inf')
    minimo_x, minimo_y = 0, 0

    # Búsqueda exhaustiva en el espacio definido
    for x in rango_x:
        for y in rango_y:
            valor_actual = funcion_himmelblau(x, y)
            if valor_actual < valor_minimo:
                valor_minimo = valor_actual
                minimo_x = x
                minimo_y = y

    # Mostrar resultados
    print(f"\nMétodo de fuerza bruta:")
    print(f"El mínimo está en x = {minimo_x}, y = {minimo_y}, con un valor de
{valor_minimo}\n")
```

- **Grilla de puntos**: Se crea una grilla de 1000×1000 puntos entre los valores $[-5, 5]$ para x e y.
- **Evaluación de la función**: La función de Himmelblau se evalúa en cada punto de la grilla, y se almacena el valor mínimo encontrado.
- **Garantía de encontrar el mínimo global**: Debido a que la función se evalúa en cada punto, se garantiza que este método encontrará el mínimo global dentro del dominio definido.

Método de descenso por gradiente

El **descenso por gradiente** es un algoritmo iterativo que busca minimizar una función moviéndose en la dirección opuesta al gradiente (derivada) de la función. El gradiente indica la dirección de mayor incremento de la función, por lo que moverse en la dirección contraria lleva hacia un mínimo.

El código del algoritmo de descenso por gradiente es el siguiente:

```
# Método 3: Descenso por gradiente
def metodo_gradiente():
    # Definir el gradiente (derivadas parciales de la función)
    def gradiente_himmelblau(x, y):
        df_dx = 4 * (x**2 + y - 11) * x + 2 * (x + y**2 - 7)
        df_dy = 2 * (x**2 + y - 11) + 4 * (x + y**2 - 7) * y
        return np.array([df_dx, df_dy])

    # Parámetros iniciales
    x_actual, y_actual = 0, 0   # Punto de inicio
    tasa_aprendizaje = 0.001    # Tamaño del paso
    tolerancia = 1e-16          # Criterio de convergencia
    max_iteraciones = 1500      # Máximo número de iteraciones

    # Algoritmo de descenso por gradiente
    for i in range(max_iteraciones):
        gradiente = gradiente_himmelblau(x_actual, y_actual)
        nuevo_x = x_actual - tasa_aprendizaje * gradiente[0]
        nuevo_y = y_actual - tasa_aprendizaje * gradiente[1]

        # Verificar convergencia
        if np.linalg.norm([nuevo_x - x_actual, nuevo_y - y_actual]) < tolerancia:
            print(f"\nConvergencia alcanzada en iteración {i}")
            break

        x_actual, y_actual = nuevo_x, nuevo_y

    # Mostrar resultados
    valor_minimo = funcion_himmelblau(x_actual, y_actual)
    print(f"Método de descenso por gradiente:")
    print(f"El mínimo está en x = {x_actual}, y = {y_actual}, con un valor de
{valor_minimo}\n")
```

- **Gradiente**: Las derivadas parciales de la función respecto a x e y se calculan en cada paso.
- **Actualización de parámetros**: Se ajustan los valores de x e y en la dirección opuesta al gradiente, usando una tasa de aprendizaje de *0.001*.
- **Convergencia**: El proceso continúa hasta que los cambios en x e y son menores a un valor de tolerancia *(1e−16)*, lo que indica que se ha alcanzado un mínimo.

Resultados Esperados

Método con librería (Scipy): Este método es eficiente y converge rápidamente hacia un mínimo local de la función, con el valor más cercano a cero. Dependiendo del punto inicial, puede converger a distintos mínimos locales.

```
Selecciona el método para encontrar el mínimo de la función de Himmelblau:
1. Usar librería (scipy)
2. Fuerza bruta
3. Descenso por gradiente
4. Salir
Introduce el número del método que deseas usar: 1

Método con librería:
El mínimo está en x = 3.0000003942258213, y = 1.9999999922066738, con un valor de 5.6899045650418925e-12
```

Método de fuerza bruta: Debido a su exhaustividad, este método garantiza encontrar el mínimo global de la función, aunque a un alto costo computacional. Los valores esperados de x e y deberían estar muy cercanos a uno de los mínimos globales conocidos.

```
Selecciona el método para encontrar el mínimo de la función de Himmelblau:
1. Usar librería (scipy)
2. Fuerza bruta
3. Descenso por gradiente
4. Salir
Introduce el número del método que deseas usar: 2

Método de fuerza bruta:
El mínimo está en x = 2.997997997997998, y = 1.9969969969969972, con un valor de 0.0004214702438393723
```

Método de descenso por gradiente: Este algoritmo debería converger a un mínimo local. Sin embargo, dependiendo de la elección del punto inicial, podría detenerse en un mínimo local que no sea el global. Con una tasa de aprendizaje adecuada, se espera que el algoritmo encuentre un mínimo satisfactorio, pero no siempre el mínimo global.

```
Selecciona el método para encontrar el mínimo de la función de Himmelblau:
1. Usar librería (scipy)
2. Fuerza bruta
3. Descenso por gradiente
4. Salir

Convergencia alcanzada en iteración 1252
Método de descenso por gradiente:
El mínimo está en x = 2.9999999999999947, y = 2.0000000000000093, con un valor de 1.5556337050958353e-27
```

Conclusión

En esta práctica, se implementaron tres enfoques diferentes para encontrar los mínimos de la función de Himmelblau, cada uno con sus propias ventajas y limitaciones. A través del uso de una librería especializada (scipy), se obtuvo una solución rápida y eficiente, la cual depende de la elección del punto inicial, pero que ofrece resultados sólidos en la mayoría de los casos.

La búsqueda por fuerza bruta, aunque garantiza encontrar el mínimo global, resulta computacionalmente costosa y se vuelve impráctica para problemas más complejos o de mayor dimensionalidad. No obstante, ofrece una solución robusta cuando el rango de búsqueda está bien definido y los recursos computacionales son suficientes.

El descenso por gradiente, por su parte, es una técnica eficiente que puede ajustarse fácilmente para problemas de múltiples dimensiones. Sin embargo, requiere una buena elección de parámetros (tasa de aprendizaje, punto inicial) para evitar caer en mínimos locales no óptimos. Si se implementa correctamente, puede ofrecer una solución rápida y cercana al mínimo global.

En conclusión, la elección del mejor método depende de las características del problema y de los recursos disponibles. Si se prioriza la velocidad y se tolera una solución aproximada, el descenso por gradiente o el uso de scipy son buenas opciones. Si se requiere encontrar el mínimo global con certeza, y los recursos lo permiten, la fuerza bruta es la mejor opción.

Este ejercicio demuestra la importancia de comprender los métodos de optimización disponibles y seleccionar el algoritmo más adecuado según el contexto del problema.