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 del algoritmo de búsqueda local Hill Climbing con paso ascendente, aplicado a una función de prueba en n dimensiones.

2. **Descripción del código**

El programa implementa el algoritmo de Hill Climbing con paso ascendente, el cual evalúa los vecinos inmediatos de la solución actual aumentando o disminuyendo cada dimensión en una cantidad fija denominada *paso*.
Si alguno de los vecinos presenta un valor superior al actual, el algoritmo se mueve hacia esa dirección.

3. **Componentes del Código**

Funciones de Prueba Implementadas

```
def funcion(x):
  return -np.sum(x**2) + 4*np.sum(x)
```

- funcion_obj: función a optimizar.
- x_inicial: vector de inicio generado aleatoriamente.
- max_iter: número máximo de iteraciones.
- paso: tamaño del incremento o decremento en cada variable.
- historia: lista de todos los puntos visitados durante la búsqueda.

Resultados Obtenidos

Primero vamos con ascendente y descendente en dimensión 2

Al igual que el programa anterior. Para hacerlo descendente y ascendente solo hay que cambiar estas líneas

Minimización

```
if valor_positivo> mejor_valor_vecino:
  mejor_valor_vecino = valor_positivo
  mejor_vecino=x_positivo

if valor_negativo> mejor_valor_vecino:
  mejor_valor_vecino=valor_negativo
  mejor_vecino=x_negativo
```

Maximización

```
if valor_positivo< mejor_valor_vecino:
  mejor_valor_vecino = valor_positivo
```

```
    mejor_vecino=x_positivo

if valor_negativo< mejor_valor_vecino:
    mejor_valor_vecino=valor_negativo
    mejor_vecino=x_negativo
```

Minimización 2D

Punto inicial: x = [0.79067174 2.94956213], f(x) = 5.636

Resultado optimizado: x = [1.79067174 1.94956213], f(x) = 7.954

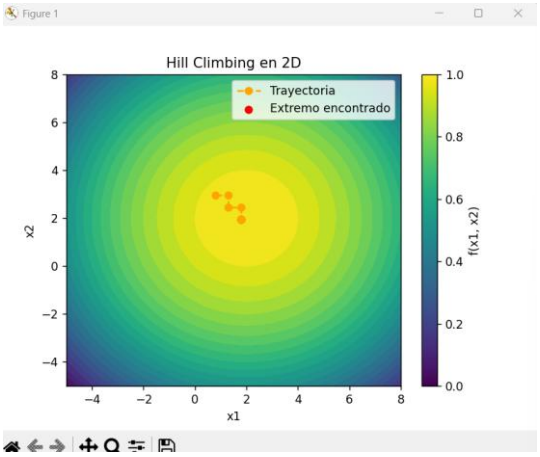

Imagen 1. Descendente en 2D

Maximización 2D

Punto inicial: x = [6.51743605 7.79990616], f(x) = -46.046

Resultado optimizado: x = [6.51743605 7.79990616], f(x) = -46.046

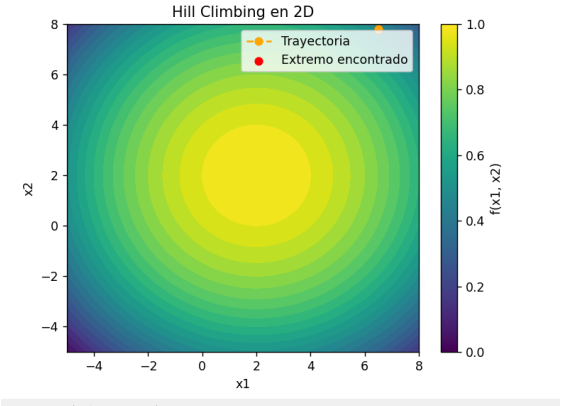

Imagen 2. Ascendente en 2D

Minimización 3D

Punto inicial: x = [6.47102976 -0.69933586 2.10216779], f(x) = -15.287

Resultado optimizado: x = [1.97102976 1.80066414 2.10216779], f(x) = 11.949

Maximización 3D

Punto inicial: x = [-8.28902072 -2.12435009 -5.96884344], f(x) = -174.377

Resultado optimizado: x = [-8.28902072 -2.12435009 -5.96884344], f(x) = -174.377

4. **Ventajas del Método**

Simplicidad: fácil de entender e implementar.

Flexibilidad: aplicable a funciones en cualquier número de dimensiones.

5. **Limitaciones**

Puede quedarse atrapado en óptimos locales si la función tiene múltiples máximos.
El tamaño del paso afecta el resultado: un paso muy grande puede saltarse el óptimo y uno muy pequeño puede ralentizar la convergencia

6. **Conclusiones**

Este código es parte de una de las 4 variaciones del código original de hill climbing, al ser una variación del original las mejoras que se le añadieron no le agregaron mucha dificultad, el solo cambiarle un par de líneas.

Código

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


def funcion(x):
  return -np.sum(x**2) + 4*np.sum(x)

def hill_climbing_paso_ascendente(funcion_obj, x_inicial, max_iter=1000, paso=0.5):
  x_actual = np.array(x_inicial)
  mejor_x = x_actual.copy()
  mejor_valor = funcion_obj(x_actual)
  historia = [x_actual.copy()]
```

```
  for i in range(max_iter):
    valor_actual=funcion_obj(x_actual)
    mejor_vecino=None
    mejor_valor_vecino=valor_actual

    for dim in range(len(x_actual)):
      x_positivo=x_actual.copy()
      x_positivo[dim]+=paso
      valor_positivo=funcion_obj(x_positivo)

      x_negativo=x_actual.copy()
      x_negativo[dim]-=paso
      valor_negativo=funcion_obj(x_negativo)

      if valor_positivo< mejor_valor_vecino:
        mejor_valor_vecino = valor_positivo
        mejor_vecino=x_positivo

      if valor_negativo< mejor_valor_vecino:
        mejor_valor_vecino=valor_negativo
        mejor_vecino=x_negativo

    if mejor_vecino is not None and mejor_valor_vecino > valor_actual:
      x_actual=mejor_vecino
      mejor_valor=mejor_valor_vecino
      mejor_x=x_actual.copy()
      historia.append(x_actual.copy())
    else:
      break


  return mejor_x, mejor_valor, np.array(historia)


n_dim = 3
x_inicial = np.random.uniform(-10, 10, size=n_dim)
max_iter = 1000
paso = 0.5

x_optimo, valor_optimo, historia = hill_climbing_paso_ascendente(funcion, x_inicial, max_iter,
paso)

print(f"Punto inicial: x = {x_inicial}, f(x) = {funcion(x_inicial):.3f}")
print(f"Resultado optimizado: x = {x_optimo}, f(x) = {valor_optimo:.3f}")
```

```
if n_dim == 2:
    x1 = np.linspace(-5, 8, 200)
    x2 = np.linspace(-5, 8, 200)
    X1, X2 = np.meshgrid(x1, x2)
    Z = -X1**2 - X2**2 + 4*X1 + 4*X2

    plt.contourf(X1, X2, Z, levels=30, cmap='viridis')
    plt.plot(historia[:, 0], historia[:, 1], 'o--', color='orange', label='Trayectoria')
    plt.scatter(x_optimo[0], x_optimo[1], color='red', label='Extremo encontrado')
    plt.title("Hill Climbing en 2D")
    plt.xlabel("x1")
    plt.ylabel("x2")
    plt.legend()
    plt.colorbar(label='f(x1, x2)')
    plt.show()
```