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 reinicios aleatorios, aplicado a funciones de prueba en n dimensiones.
El objetivo del algoritmo es encontrar el extremo global de la función mediante iteraciones locales y múltiples puntos de inicio aleatorios.

2. Descripción del código

El código desarrollado implementa el algoritmo de Hill Climbing de manera generalizada para cualquier número de dimensiones (n).
Se incorporan **reinicios aleatorios**, lo que permite ejecutar varias búsquedas desde puntos iniciales distintos y comparar los resultados para seleccionar el mejor valor encontrado.

Los parámetros principales del algoritmo son:

- max_iter: número máximo de iteraciones por reinicio.
- paso: tamaño del movimiento aleatorio en cada dimensión.
- num_reinicios: cantidad de reinicios aleatorios para aumentar la probabilidad de encontrar el mínimo global.

3. Componentes del Código

Funciones de Prueba Implementadas

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

def hill_climbing_ND(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):
    x_nuevo = x_actual + np.random.uniform(-paso, paso, size=x_actual.shape)
    valor_actual = funcion_obj(x_actual)
    valor_nuevo = funcion_obj(x_nuevo)

    if valor_nuevo < valor_actual:
      x_actual = x_nuevo
      historia.append(x_actual.copy())

    if valor_nuevo < mejor_valor:
      mejor_x = x_nuevo
```

funcion_obj: función objetivo a minimizar.

x_inicial: vector inicial de búsqueda.

max_iter: número máximo de iteraciones.

paso: tamaño del movimiento en cada dimensión.

for i in range(num_reinicios):

Esta línea es la que permite los múltiples reinicios

Resultados Obtenidos

Se generan múltiples puntos iniciales aleatorios, ejecutando el Hill Climbing desde cada uno y guardando la trayectoria.

Primero vamos con ascendente y descendente en dimensión 2

Para hacerlo descendente y ascendente solo hay que cambiar estas líneas

Minimización

```
if valor_nuevo <  valor_actual:
   x_actual = x_nuevo
   historia.append(x_actual.copy())

if valor_nuevo <  mejor_valor:
   mejor_x = x_nuevo
   mejor_valor = valor_nuevo

if valor_optimo < mejor_global_valor:
   mejor_global_valor = valor_optimo
   mejor_global_x = x_optimo

mejor_global_valor = np.inf
```

Maximización

```
    if valor_candidato < valor_actual:

       actual = candidato

       valor_actual = valor_candidato
```

```
        if valor_actual < mejor_valor:
            mejor = actual.copy()
            mejor_valor = valor_actual
mejor_global_valor = -np.inf
```

Minimización 2D

Mejor punto global encontrado: [-4.21119454 -138.14734652]

Mejor valor global: -19671.858

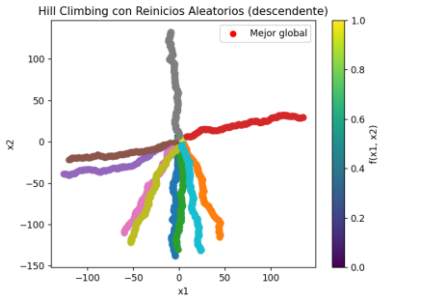

Imagen 1. Descendente en 2D

Maximización 2D

Mejor punto global encontrado: [1.99525741 2.00095119]

Mejor valor global: 8.000

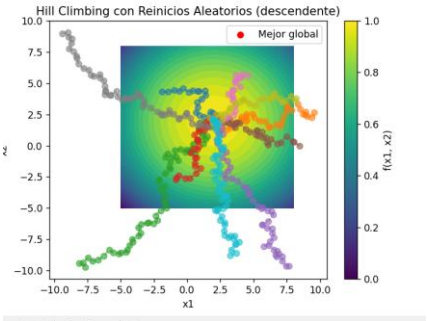

Imagen 2. Ascendente en 2D

Minimización 3D

Mejor punto global encontrado: [-13.44280274 138.41645271 31.41783637]

Mejor valor global: -19701.338

Maximización 3D

Mejor punto global encontrado: [2.01658038 1.99118284 1.98604223]

Mejor valor global: 11.999

4. **Ventajas del Método**

Simplicidad: fácil de entender e implementar.

Flexibilidad: aplicable a funciones en cualquier número de dimensiones.

Mayor probabilidad de encontrar mínimo global gracias a los reinicios aleatorios.

5. **Limitaciones**

Óptimos locales: si la función tiene múltiples mínimos muy cercanos, algunos reinicios podrían quedarse atrapados.

Dependencia de parámetros: el tamaño del paso y el número de reinicios afectan el rendimiento.

6. **Conclusiones**

La implementación de Hill Climbing con reinicios aleatorios permite mejorar los resultados de una búsqueda local simple, aumentando la probabilidad de encontrar el mínimo global. Este solo implementa un reinicio en la función principal, por lo que no es algo difícil de implementar.

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_ND(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):
    x_nuevo = x_actual + np.random.uniform(-paso, paso, size=x_actual.shape)
```

```
        valor_actual = funcion_obj(x_actual)
        valor_nuevo = funcion_obj(x_nuevo)

        if valor_nuevo < valor_actual:
            x_actual = x_nuevo
            historia.append(x_actual.copy())

        if valor_nuevo < mejor_valor:
            mejor_x = x_nuevo
            mejor_valor = valor_nuevo

    return mejor_x, mejor_valor, np.array(historia)


n_dim = 3
max_iter = 1000
paso = 0.5
num_reinicios=10
mejor_global_x = None
mejor_global_valor = np.inf
historia_global = []

for i in range(num_reinicios):
    x_inicial = np.random.uniform(-10, 10, size=n_dim)
    x_optimo, valor_optimo, historia = hill_climbing_ND(funcion, x_inicial, max_iter, paso)
    historia_global.append(historia)

    if valor_optimo < mejor_global_valor:
        mejor_global_valor = valor_optimo
        mejor_global_x = x_optimo

print("\n=== RESULTADO FINAL ===")
print(f"Mejor punto global encontrado: {mejor_global_x}")
print(f"Mejor valor global: {mejor_global_valor:.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')

    for historia in historia_global:
```

```
    plt.plot(historia[:, 0], historia[:, 1], 'o--', alpha=0.6)

# Marcar el mejor punto global
plt.scatter(mejor_global_x[0], mejor_global_x[1], color='red', label='Mejor global')
plt.title("Hill Climbing con Reinicios Aleatorios (descendente)")
plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.colorbar(label='f(x1, x2)')
plt.show()
```