Documentação do Projeto 3: Busca Complexa (Hill Climbing)

1. Visão Geral do Projeto

Este projeto demonstra um algoritmo de **Busca Local** aplicado a um problema de **otimização matemática**. O objetivo não é encontrar um caminho (como nos labirintos), mas sim encontrar o ponto de máximo ou mínimo valor em uma função matemática complexa.

A aplicação é executada via console e permite ao usuário escolher entre:

1. Minimizar a função: $f(x) = -10x^4 + 5x^6 + 1$
2. Maximizar a função: $f(x) = -x^2 + 4x + 4$

O programa então utiliza o algoritmo *Hill Climbing* para "escalar" a função até encontrar um pico (máximo local) ou um vale (mínimo local).

2. Como Foi Desenvolvido

O projeto foi desenvolvido inteiramente em **Python**, sem a necessidade de bibliotecas gráficas, focando puramente na lógica do algoritmo de busca.

- **Linguagem:** Python.
- **Biblioteca `random`:** Utilizada para definir os pontos de partida de cada tentativa de busca.
- **Interface:** Baseada em terminal (`input()` e `print()`). O usuário escolhe a operação (minimizar ou maximizar) por texto.
- **Lógica Principal:** O código implementa uma variação importante do algoritmo chamada **Hill Climbing com Reinício Aleatório (*Random Restarts*)**. Como o *Hill Climbing* simples pode ficar preso em "ótimos locais" (picos ou vales que não são os melhores globais), esta técnica executa o algoritmo várias vezes (`MAX_RANDOM_RESTARTS`) a partir de diferentes pontos iniciais aleatórios, aumentando a chance de encontrar o ótimo global.

3. Algoritmo Utilizado

Busca Local: Hill Climbing (Subida de Encosta)

O *Hill Climbing* é um algoritmo de busca local que tenta iterativamente encontrar um estado melhor movendo-se na direção de maior "inclinação" (ou "subida").

O processo funciona da seguinte forma:

1. **Início:** Começa em um estado aleatório (um ponto `x` aleatório na função).

2. **Avaliação:** Calcula o valor da função nesse ponto (y).
3. **Vizinhança:** Olha para os "vizinhos" imediatos (os pontos x - STEP_SIZE e x + STEP_SIZE).
4. **Movimento:**
    - Avalia o valor da função (y) nos vizinhos.
    - Se um dos vizinhos tiver um valor melhor (maior, se estiver maximizando; menor, se estiver minimizando) do que o ponto atual, o algoritmo "se move" para esse vizinho.
    - Esse vizinho se torna o novo ponto atual.
5. **Término (Ótimo Local):** O processo é repetido até que o algoritmo chegue a um ponto onde nenhum dos vizinhos é melhor que ele. Neste momento, o algoritmo atingiu um "pico" (máximo local) ou "vale" (mínimo local) e para.

```python
def func_to_max(x) -> int:
    # Função para maximizar
    return -1*x**2 + 4*x + 4

    Gabriel Castelo, ontem • hill_climbing funcionando
def func_to_min(x) -> int:
    # Função para minimizar
    return -10*x**4 + 5*x**6 + 1


def hill_climb(start, min: bool = False) -> int | bool:
    """ Executa o algoritmo de Hill Climbing a partir de um ponto inicial.
    Retorna o melhor resultado encontrado."""
    current_x = start
    # Avalia o valor inicial e escolhe a função correta com base no objetivo
    current_y = func_to_min(current_x) if min else func_to_max(current_x)

    count = 0  # Contador de passos

    # Loop até que não haja mais melhorias
    while True:
        if count >= MAX_STEPS:
            print("Número máximo de passos atingido.")
            return None, None
        next_x = None
        next_y = current_y
        # Gera vizinhos e avalia
        for candidate in linspace(current_x):
            if min:
                # Avalia a função de minimização
                value = func_to_min(candidate)
                if value < current_y:
                    # Atualiza se encontrar um valor melhor
                    next_x = candidate
                    next_y = value
            else:
                # Avalia a função de maximização
                value = func_to_max(candidate)
                if value > current_y:
                    # Atualiza se encontrar um valor melhor
                    next_x = candidate
                    next_y = value
        # Se nenhum vizinho for melhor, termina
        if next_x is None:
            break
        # Move para o próximo ponto
        current_x = next_x
        current_y = next_y

        count += 1  # Incrementa o contador de passos
    return current_x, current_y
```

Figura 01: Visão geral das funções e do algoritmo hill_climbing

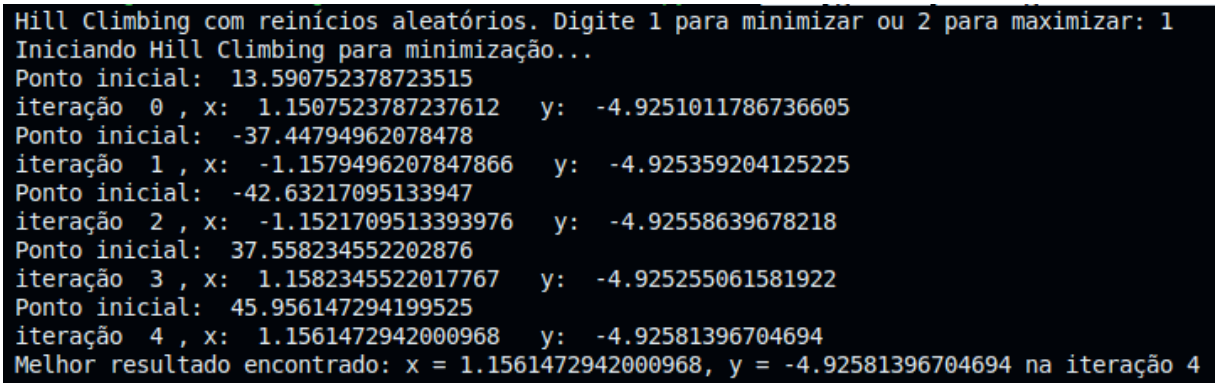

Figura 02: Visão geral da saída do programa