Instituto Tecnológico de Aeronáutica - ITA
Inteligência Artificial para Robótica Móvel - CT213
Aluno: Tafnes Silva Barbosa

Relatório do Laboratório 3 - Otimização com Métodos de Busca Local

1 Breve Explicação em Alto Nível da Implementação

Neste laboratório foram implementadas três tipos de otimização diferentes: descida do gradiente, *hill climbing* e *simulated annealing*.

1.1 Descida do Gradiente

A função de descida do gradiente foi implementada usando a seguinte relação de atualização:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{1}$$

onde $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ é dado pela função `gradient_function(theta)`. A cada atualização do valor de $\boldsymbol{\theta}$, o valor antigo é colocado dentro de um vetor contendo o histórico de valores.

Foram usados como critérios de parada o número de iterações (para depois da milésima iteração) e o valor atual do custo da função (para quando menor que 10^{-10}). A taxa de aprendizado usada foi de 0.1 e o chute inicial foi

$$\boldsymbol{\theta}_0 = \begin{bmatrix} v_0 \\ f \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

1.2 *Hill Climbing*

Usou-se uma estratégia 8-conectada para os vizinhos de um certo $\boldsymbol{\theta}$. Uma função `neighbors (theta)` foi implementada para retornar os 8 vizinhos de $\boldsymbol{\theta}$, igualmente distribuídos ao seu redor. Cada vizinho estava a uma distância euclidiana de $\Delta = 2 \cdot 10^{-3}$ e o ângulo entre cada vizinho era de 45°. A equação abaixo mostra como cada vizinho foi encontrado:

$$\boldsymbol{vizinho} = \begin{bmatrix} \theta[0] + \Delta \cdot \cos(\gamma) \\ \theta[1] + \Delta \cdot \sin(\gamma) \end{bmatrix}$$

onde $\gamma \in [0 : 7] \cdot \pi/4$.

A partir dos vizinhos, a função de atualização de $\boldsymbol{\theta}$ foi implementada com os mesmos critérios de parada da descida de gradiente (número de iterações e o valor do custo atual). A cada iteração, checa-se qual vizinho de $\boldsymbol{\theta}$ tem um custo menor entre ele. Depois verifica se o menor custo encontrado é menor que o custo do $\boldsymbol{\theta}$ atual. Caso, ele não encontre vizinhos com custo menor que o atual, ele finaliza a atualização, pois chegou a um mínimo local. Caso ele encontre algum vizinho com custo menor, ele atualiza o $\boldsymbol{\theta}$ e coloca o valor do anterior no vetor contendo o histórico.

Na implementação deste código, foi usada uma variável para guardar o melhor custo atual, dado que o melhor foi iniciado como `None` e a função de custo não foi programada para retornar infinito para `None`.

1.3 *Simulated Annealing*

Para este tipo de atualização, foram implementadas 3 funções:

1. Uma que escolhe um vizinho aleatório a uma distância Δ da mesma forma que os vizinhos do *hill climbing* foram escolhidos, mas com o ângulo γ aleatório uniformemente distribuído entre $(-\pi, \pi)$;

2. Uma função que atualiza o *schedule* (T) para uma dada iteração (i) a partir da equação

$$T = \frac{T_0}{1 + \beta i^2},$$

   onde $T_0 = 1$ e $\beta = 1$;

3. A função que atualiza o valor de $\boldsymbol{\theta}$ usa os mesmos critérios de parada da descida de gradiente (número de iterações e valor do custo atual) com um critério adicional caso o *schedule* seja negativo (no caso do *schedule* e constantes usados nesse laboratório, nunca haverá um valor negativo para ele). Mas, diferentemente do *hill climbing*, ele não para caso não encontre um vizinho com custo menor. Ao escolher um vizinho aleatoriamente, ele atualiza o custo e coloca o anterior no histórico caso o custo do vizinho seja menor que o atual, mas caso não seja, ele escolhe um número aleatória e uniformemente (r) entre 0 e 1 e atualiza o valor atual e coloca o anterior no histórico, caso a comparação conforme à seguinte equação seja verdadeira:

$$r \leq e^{\frac{-\Delta E}{T}}$$

   onde $\Delta E = J(\boldsymbol{vizinho}) - J(\boldsymbol{\theta})$ e J é a função de custo. Conforme T cresce, menor é a chance de a comparação acima ser verdadeira, pois o valor da exponencial cai. No caso do *schedule* usado, vai cair mais rápido, pelo fato do denominador ser proporcional ao quadrado do número da iteração, se comparado com um dos apresentados em aula. Ou seja, quanto mais iterações, menor a chance de pegar um vizinho com custo maior que o atual, dado que, com o tempo, o ΔE “diminua”, considerando o funcionamento da *simulated annealing*.

2 Figuras Comprovando Funcionamento do Código

2.1 Descida do Gradiente

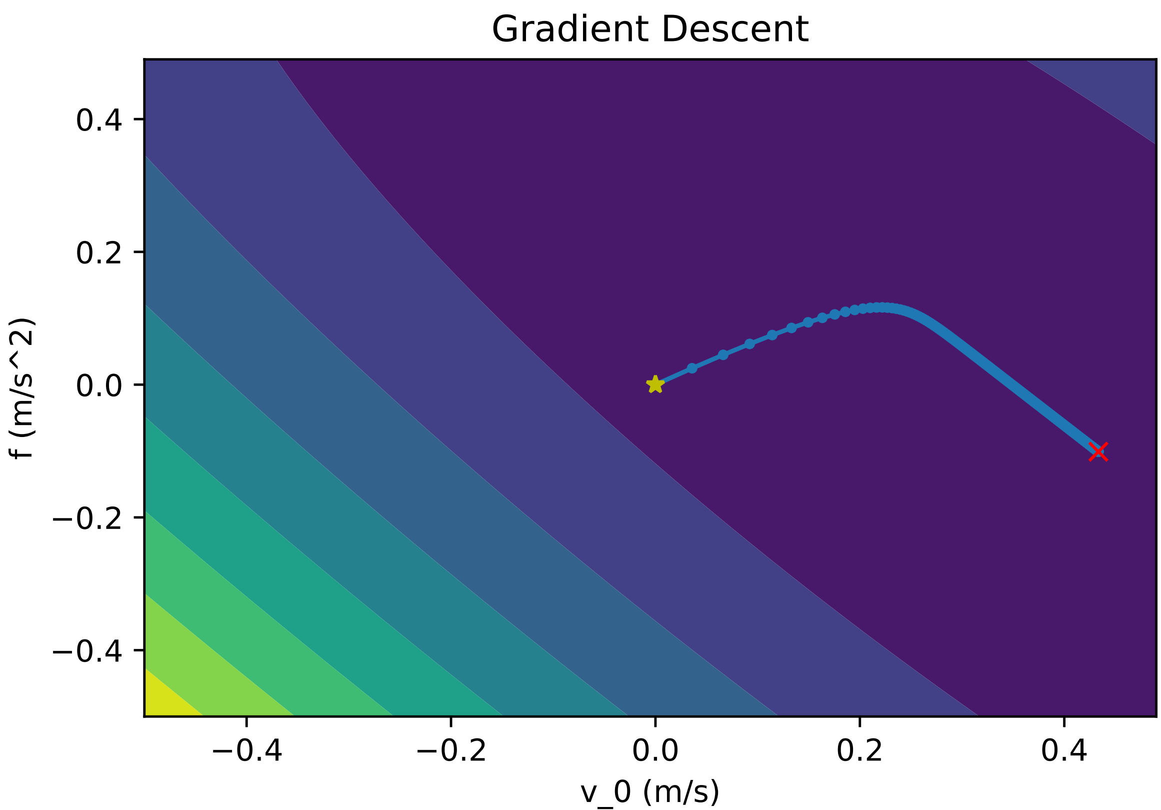

Figura 1: Caminho percorrido pelo método de otimização: Descida do Gradiente.

2.2 *Hill Climbing*

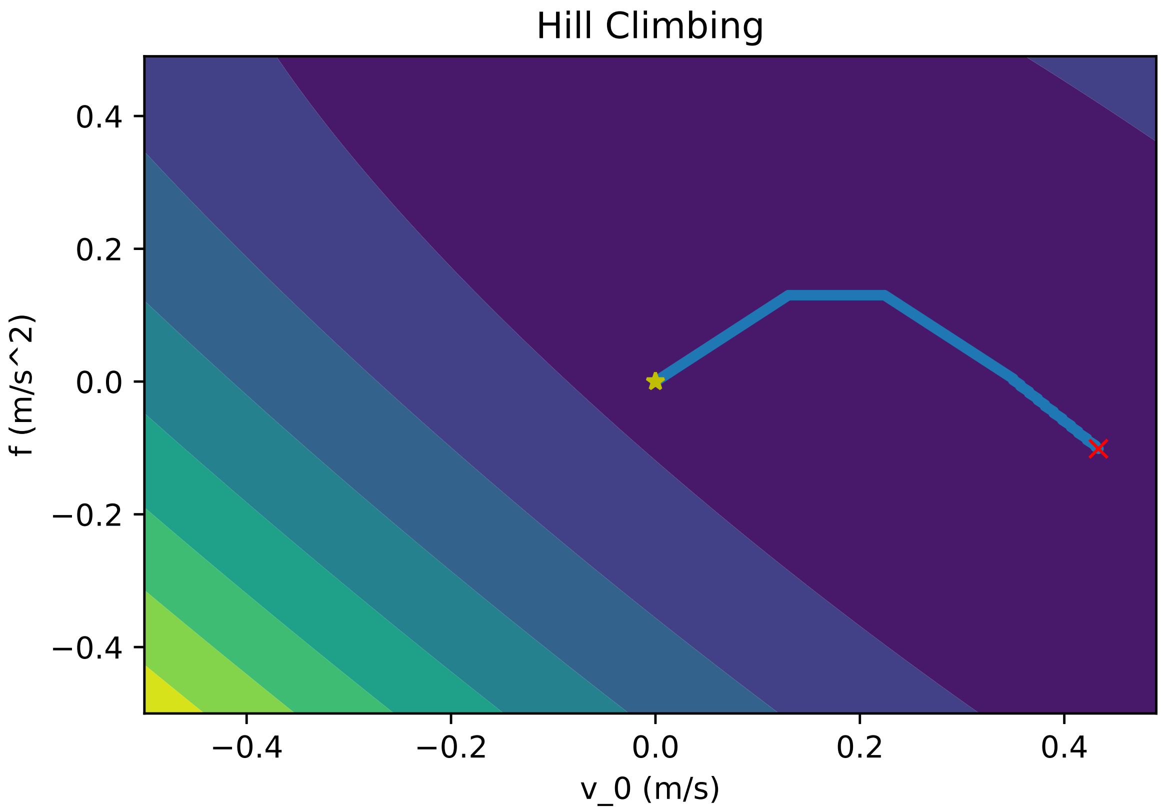

Figura 2: Caminho percorrido pelo método de otimização: *Hill Climbing*.

2.3 *Simulated Annealing*

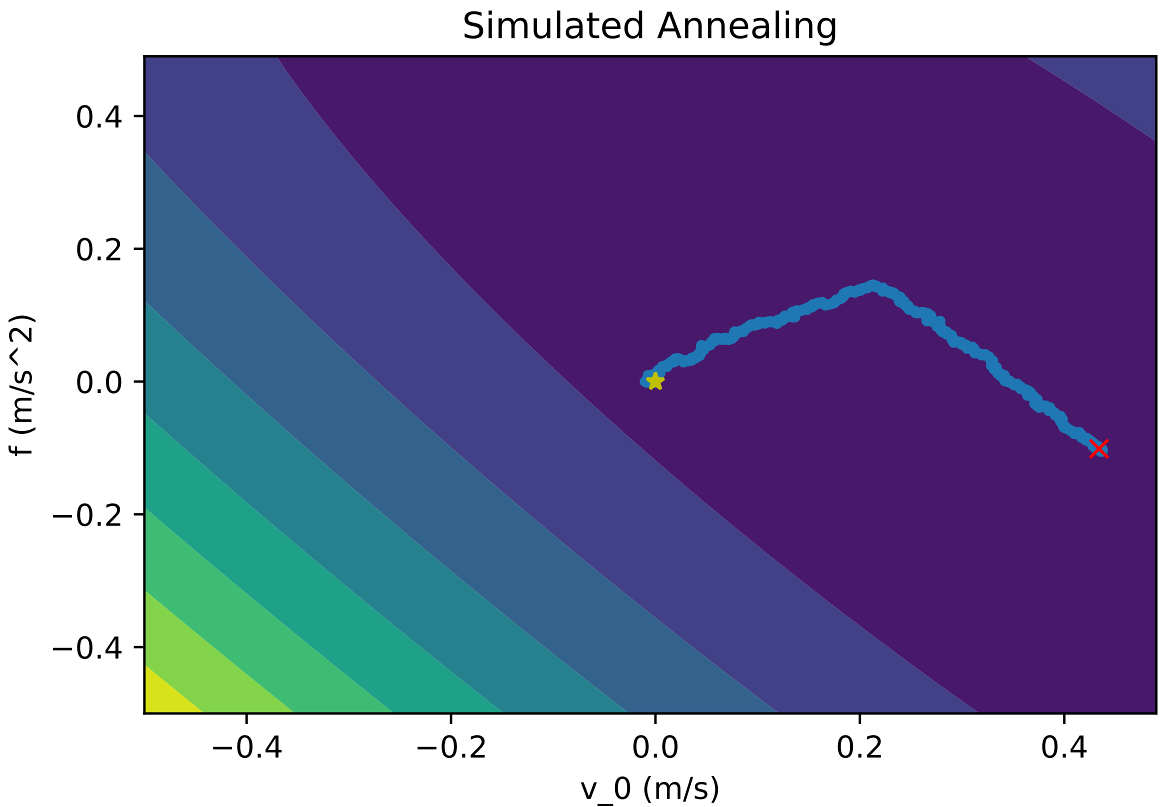

Figura 3: Caminho percorrido pelo método de otimização: *Simulated Annealing*.

3 Comparação entre os métodos

A Figura 4 mostra uma comparação entre os caminhos percorridos por cada método de otimização.

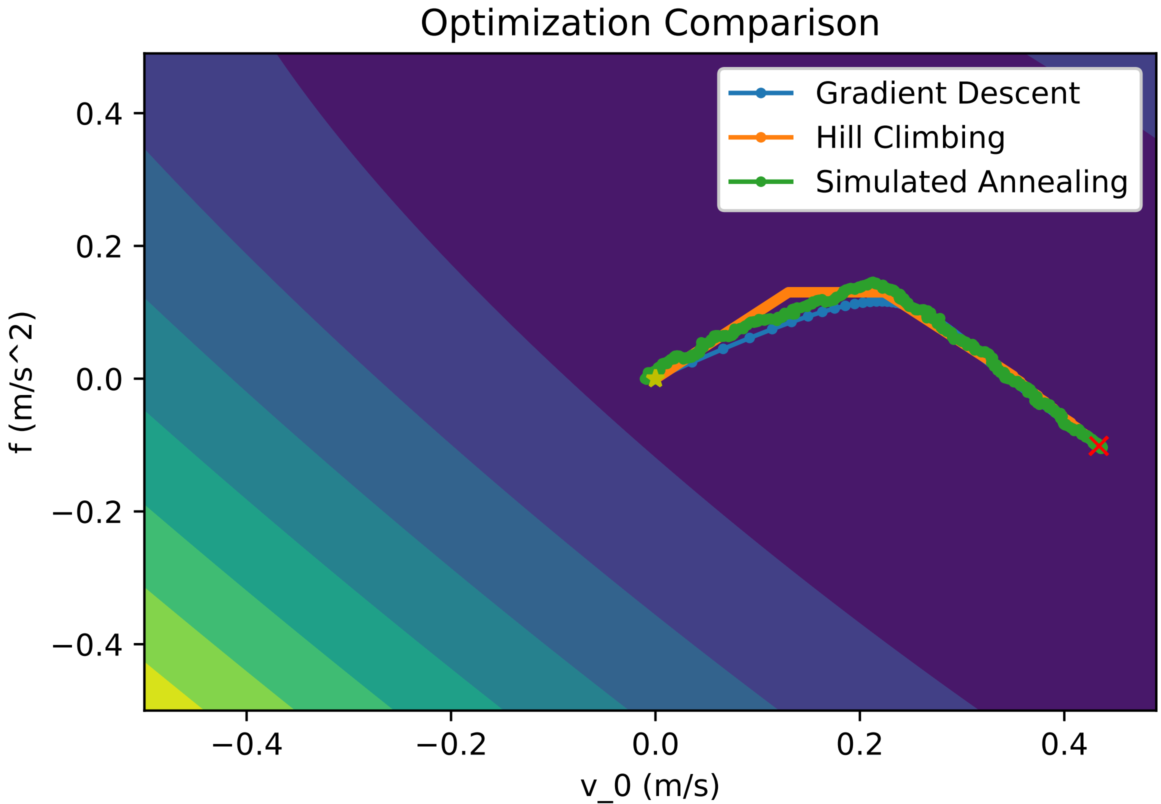

Figura 4: Caminhos percorridos por cada método de otimização.

A Figura 5 mostra os pontos experimentais e as curvas de regressão linear obtidas por cada método.

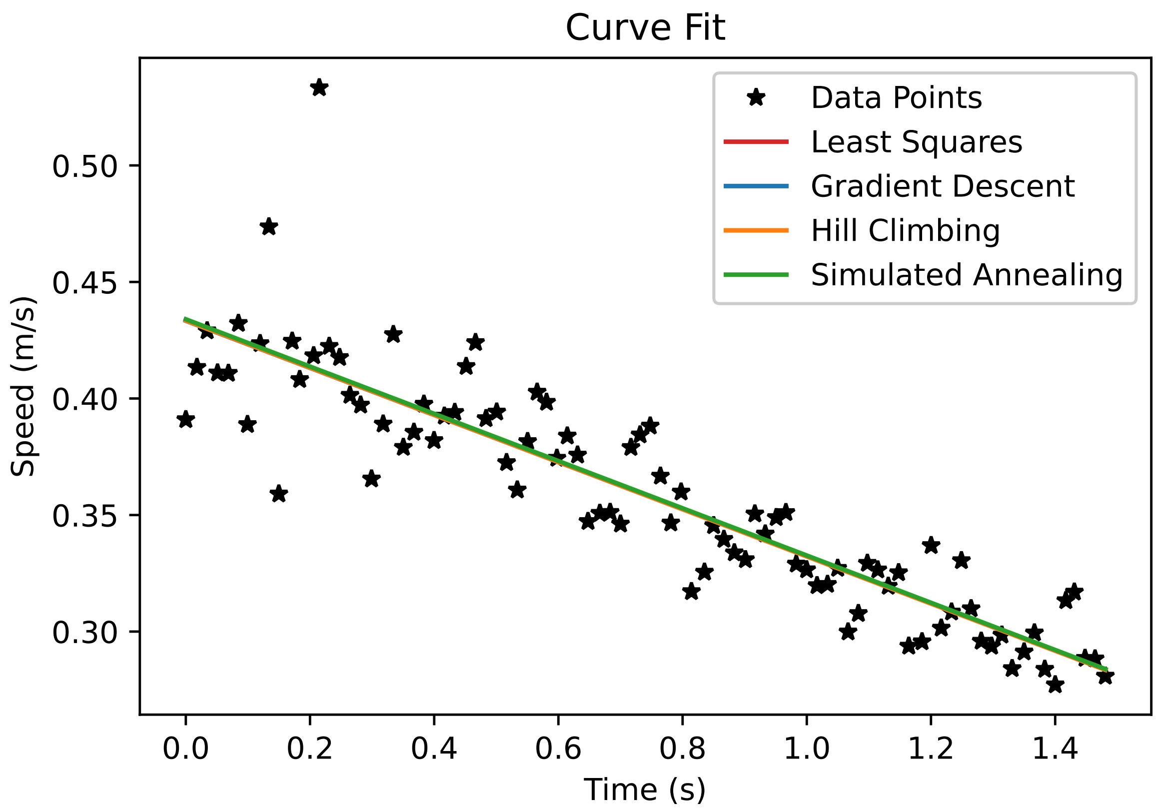

Figura 5: Curvas de regressão linear obtidas por cada método de otimização.

A Tabela 1 mostra a comparação dos parâmetros da regressão linear obtidos pelos métodos de otimização.

Tabela 1: parâmetros da regressão linear obtidos pelos métodos de otimização.

Método	v_0	f
MMQ	0.433373	-0.101021
Descida do gradiente	0.433371	-0.101018
Hill climbing	0.433411	-0.101196
Simulated annealing	0.433977	-0.101345